Experiment– 6

Aim: To perform various operations on views using SQL on the EMPLOYEE table. This includes:

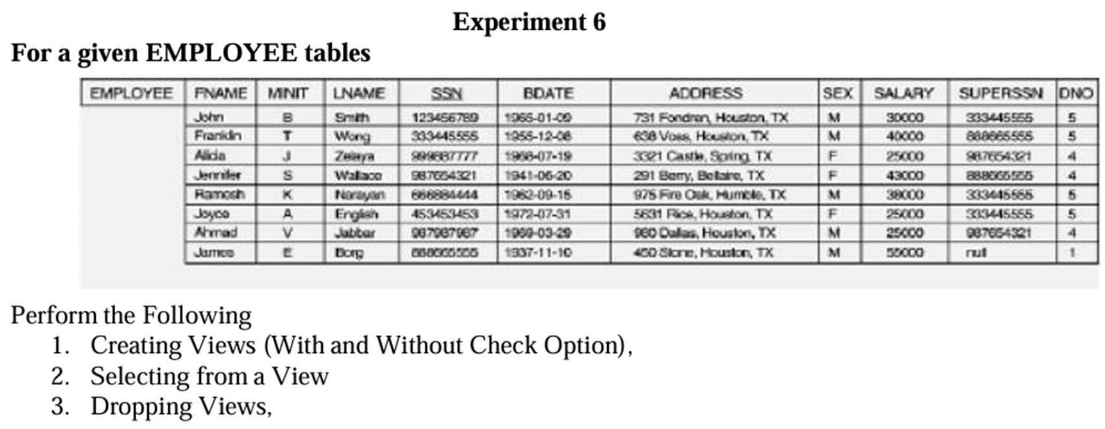
1. Creating views with and without the CHECK OPTION.
2. Selecting data from a view.
3. Dropping views from the database.

Objective:

The objective of this experiment is to:

1. Understand the concept of **views** in relational databases and their importance in simplifying complex queries.
2. Learn how to create views with constraints using the CHECK OPTION to ensure data integrity.
3. Practice selecting data from views to retrieve specific records based on predefined criteria.
4. Demonstrate how to drop views when they are no longer needed, freeing up database resources.

Given Problem: -



Experiment 6

For a given EMPLOYEE tables

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

Perform the Following

1. Creating Views (With and Without Check Option),
2. Selecting from a View
3. Dropping Views,

1. Creating Views (With and Without Check Option)

- **Create View without Check Option**:

CREATE VIEW Employee_View AS

SELECT FNAME, LNAME, SEX, SALARY

FROM EMPLOYEE;

This query creates a simple view called Employee_View that shows the first name, last name, sex, and salary of employees.

- **Create View with Check Option**:

CREATE VIEW High_Salary_View AS

SELECT FNAME, LNAME, SALARY

FROM EMPLOYEE

WHERE SALARY > 40000

WITH CHECK OPTION;

This creates a view called High_Salary_View, showing only employees with a salary greater than 40,000. The WITH CHECK OPTION ensures that any future changes to the base table through this view must satisfy the condition SALARY > 40000.

2. Selecting from a View

To select data from the view:

- **Selecting from Employee_View**:

SELECT * FROM Employee_View;

This will return the details of employees (first name, last name, sex, and salary) as per the defined view.

- **Selecting from High_Salary_View**:

SELECT * FROM High_Salary_View;

SELECT * FROM High_Salary_View;

SELECT * FROM High_Salary_View;

This query will return details of employees whose salary is greater than 40,000.

3. Dropping Views

To drop a view when it is no longer needed, use the following syntax:

- **Dropping Employee_View**:

DROP VIEW Employee_View;

- **Dropping High_Salary_View**:

DROP VIEW High_Salary_View;

Experiment- 7

Aim:

To write a PL/SQL program using a FOR loop to:

1. Insert ten rows into a database table.
2. Print integers from 1 to 10 using the FOR loop.

Objective:

The objective of this experiment is to:

1. Understand the usage of loops in PL/SQL to automate repetitive tasks.
2. Learn how to insert multiple rows into a table using a loop.
3. Practice how to print integers using the FOR loop in PL/SQL.

Program:

```
-- Insert 10 Rows into a Database Table Using PL/SQL FOR Loop
-- Assume the table is named 'number_table' with one column 'num'
CREATE TABLE number_table (
  num NUMBER
);

-- PL/SQL block to insert 10 rows into the 'number_table'
BEGIN
  FOR i IN 1..10 LOOP
    INSERT INTO number_table (num) VALUES (i);
  END LOOP;

  -- Commit the transaction to save the changes
  COMMIT;
END;
/
```

Explanation:

- The table number_table has a single column num where we will insert integers from 1 to 10.

- The FOR loop iterates through values from 1 to 10 and inserts them into the number_table using the INSERT INTO statement.
- After inserting, the COMMIT statement ensures the changes are saved permanently to the database.

```
-- PL/SQL block to print integers from 1 to 10
BEGIN
  FOR i IN 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE('The value of i is: ' || i);
  END LOOP;
END;
/
```

Explanation:

- This program simply prints integers from 1 to 10 using the FOR loop.
- The DBMS_OUTPUT.PUT_LINE procedure is used to print the current value of i during each iteration of the loop.

Output: -

SELECT * FROM number_table;

num
1
2
3
4
5
6
7
8
9
10

Program 2

```
The value of i is: 1
The value of i is: 2
The value of i is: 3
The value of i is: 4
The value of i is: 5
The value of i is: 6
The value of i is: 7
The value of i is: 8
The value of i is: 9
The value of i is: 10
```

Experiment- 8

Aim:

To write a PL/SQL program using a cursor to retrieve and display the top five highest-paid employees from the EMPLOYEE table.

Objective:

The objective of this experiment is to:

1. Understand how to declare and use cursors in PL/SQL to handle result sets.
2. Retrieve and display records for specific conditions using cursors.
3. Practice selecting the top 5 employees based on salary.

Program: Cursor to Select Top 5 Highest Paid Employees

```
-- Assume the EMPLOYEE table is already created with relevant fields


CREATE TABLE EMPLOYEE (

   EmpNo NUMBER,

   Name VARCHAR2(50),

   Salary NUMBER,

   Designation VARCHAR2(50),

   DeptID NUMBER

);


-- Insert sample data into the EMPLOYEE table

INSERT INTO EMPLOYEE (EmpNo, Name, Salary, Designation, DeptID)

VALUES (1, 'John Doe', 60000, 'Manager', 101);
```

```
INSERT INTO EMPLOYEE (EmpNo, Name, Salary, Designation, DeptID)
VALUES (2, 'Jane Smith', 50000, 'Developer', 102);
INSERT INTO EMPLOYEE (EmpNo, Name, Salary, Designation, DeptID)
VALUES (3, 'Alice Brown', 75000, 'Senior Developer', 101);
INSERT INTO EMPLOYEE (EmpNo, Name, Salary, Designation, DeptID)
VALUES (4, 'Bob Martin', 45000, 'Tester', 103);
INSERT INTO EMPLOYEE (EmpNo, Name, Salary, Designation, DeptID)
VALUES (5, 'Charlie White', 85000, 'CTO', 101);
INSERT INTO EMPLOYEE (EmpNo, Name, Salary, Designation, DeptID)
VALUES (6, 'Emily Green', 72000, 'Project Manager', 102);

-- PL/SQL Block to use a cursor to retrieve the top 5 highest-paid employees
DECLARE
  CURSOR emp_cursor IS
    SELECT EmpNo, Name, Salary, Designation, DeptID
    FROM EMPLOYEE
    ORDER BY Salary DESC
    FETCH FIRST 5 ROWS ONLY;  -- Top 5 highest salaries

  emp_record emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;

  DBMS_OUTPUT.PUT_LINE('EmpNo | Name        | Salary | Designation        |
DeptID');
```

```
  DBMS_OUTPUT.PUT_LINE('-----------------------------------------------------------');

  LOOP
    FETCH emp_cursor INTO emp_record;
    EXIT WHEN emp_cursor%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE(emp_record.EmpNo || '    | ' || emp_record.Name || ' | ' ||
                         emp_record.Salary || '  | ' || emp_record.Designation || ' | ' ||
                         emp_record.DeptID);
  END LOOP;

  CLOSE emp_cursor;
END;
/
```

Output: -

```
EmpNo | Name         | Salary | Designation        | DeptID
-----------------------------------------------------------
5     | Charlie White | 85000  | CTO               | 101
3     | Alice Brown   | 75000  | Senior Developer  | 101
6     | Emily Green   | 72000  | Project Manager   | 102
1     | John Doe      | 60000  | Manager           | 101
2     | Jane Smith    | 50000  | Developer         | 102
```

Experiment- 9

Aim:

To demonstrate how to embed PL/SQL in a high-level host language (such as C or Java) to perform a banking debit transaction.

Objective:

The objective of this experiment is to:

1. Illustrate how to embed and execute PL/SQL blocks inside a high-level programming language like Java.
2. Implement a banking debit transaction using PL/SQL in Java, simulating account balance deduction.
3. Use the necessary Java Database Connectivity (JDBC) for interacting with a database.

Program: Embedding PL/SQL in Java for a Debit Transaction

In this example, we demonstrate embedding PL/SQL into a **Java** program using **JDBC** for executing a **debit transaction** from a bank account.

Banking Debit Transaction Logic:

- A BANK_ACCOUNT table will be used to simulate account information.
- A PL/SQL block will deduct a specified amount from the account balance if sufficient funds exist.

Steps:

1. **Create the Bank Account Table**.
2. **Write the PL/SQL block for Debit Transaction**.
3. **Embed the PL/SQL block in a Java Program**.

Step 1: Create the Bank Account Table

```
CREATE TABLE BANK_ACCOUNT (
    AccountNo NUMBER PRIMARY KEY,

    Name VARCHAR2(50),

    Balance NUMBER

);


-- Insert sample data

INSERT INTO BANK_ACCOUNT (AccountNo, Name, Balance) VALUES (1001, 'John
Doe', 5000);

INSERT INTO BANK_ACCOUNT (AccountNo, Name, Balance) VALUES (1002, 'Jane
Smith', 3000);

COMMIT;
```

Step 2: PL/SQL Block for Debit Transaction

This PL/SQL block will debit an account and check for sufficient balance before deducting.

```
DECLARE

  v_balance BANK_ACCOUNT.Balance%TYPE;

BEGIN

  -- Fetch current balance

  SELECT Balance INTO v_balance FROM BANK_ACCOUNT WHERE AccountNo =
1001;


  IF v_balance >= 1000 THEN

    -- If sufficient balance, deduct the amount

    UPDATE BANK_ACCOUNT

    SET Balance = Balance - 1000

    WHERE AccountNo = 1001;
    DBMS_OUTPUT.PUT_LINE('Debit transaction successful. Amount deducted: 1000');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Insufficient balance.');
  END IF;

  COMMIT;
END;
/
```

Step 3: Embedding PL/SQL in Java using JDBC

Here, the **Java** program connects to the database and executes the PL/SQL block to perform the debit transaction.

```
import java.sql.*;

public class BankTransaction {
   public static void main(String[] args) {
      Connection conn = null;
      CallableStatement stmt = null;

      try {
         // Load and register Oracle JDBC Driver (or any other DB driver)
         Class.forName("oracle.jdbc.driver.OracleDriver");

         // Establish connection to the database
```

```
    conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
"username", "password");


    // PL/SQL block for debit transaction

    String plsqlBlock = "{ DECLARE "

            + "  v_balance BANK_ACCOUNT.Balance%TYPE; "

            + "BEGIN "

            + "  SELECT Balance INTO v_balance FROM BANK_ACCOUNT WHERE
AccountNo = 1001; "

            + "  IF v_balance >= 1000 THEN "

            + "    UPDATE BANK_ACCOUNT SET Balance = Balance - 1000
WHERE AccountNo = 1001; "

            + "    DBMS_OUTPUT.PUT_LINE('Debit transaction successful.
Amount deducted: 1000'); "

            + "  ELSE "

            + "    DBMS_OUTPUT.PUT_LINE('Insufficient balance.'); "

            + "  END IF; "

            + "  COMMIT; "

            + "END; }";


    // Prepare and execute the PL/SQL block

    stmt = conn.prepareCall(plsqlBlock);

    stmt.execute();


    System.out.println("Transaction executed successfully!");
```

```
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // Clean up the environment
            try {
                if (stmt != null) stmt.close();
                if (conn != null) conn.close();
            } catch (SQLException se) {
                se.printStackTrace();
            }
        }
    }
}
```

Output:

When the program is run, if AccountNo = 1001 has sufficient balance, the output would be:

```
Transaction executed successfully!

Debit transaction successful. Amount deducted: 1000
```

If there is **insufficient balance**, the output would be:

```
Transaction executed successfully!

Insufficient balance.
```

Experiment- 10

Aim:

To write a PL/SQL procedure that inserts a tuple consisting of an integer i and a string 'xxx' into a given relation.

Objective:

The objective of this experiment is to demonstrate how to create and execute a PL/SQL procedure that inserts values into a table dynamically, using an input integer i and a fixed string 'xxx'. The procedure will use SQL INSERT command within the PL/SQL block to insert the tuple into the relation.

Example Table: TEST_TABLE

ID	**NAME**
1	abc
2	def

PL/SQL Procedure:

```
CREATE OR REPLACE PROCEDURE insert_tuple(i IN NUMBER) IS
BEGIN
  -- Insert a tuple into the given table
  INSERT INTO TEST_TABLE (ID, NAME) VALUES (i, 'xxx');

  -- Commit the transaction to save the changes
  COMMIT;

  -- Output message
  DBMS_OUTPUT.PUT_LINE('Tuple (' || i || ', ''xxx'') inserted successfully.');
END;
```

Execution:

```
BEGIN
   insert_tuple(5);  -- Calls the procedure with i = 5
END;
/
```

Output:

When the procedure is executed with the input i = 5, it inserts the tuple (5, 'xxx') into the TEST_TABLE. The output will be:

```
Tuple (5, 'xxx') inserted successfully.
```

Query to Check the Table:

```
SELECT * FROM TEST_TABLE;
```

Output:

ID	**NAME**
1	abc
2	def
5	xxx

Experiment 11: Hello World Program

Aim:
To write a PL/SQL block to print "Hello World".

Objective:
To demonstrate the basic structure of a PL/SQL program and how to output a simple message.

Code:

```
BEGIN

  DBMS_OUTPUT.PUT_LINE('Hello World');

END;

/
```

Output:

```
Hello World
```

Experiment 12: Program to Add Two Numbers

Aim:
To write a PL/SQL program to add two numbers and display the result.

Objective:
To demonstrate how to perform arithmetic operations in PL/SQL.

Code:

```
DECLARE

  num1 NUMBER := 10;

  num2 NUMBER := 20;

  sum NUMBER;

BEGIN

  sum := num1 + num2;

  DBMS_OUTPUT.PUT_LINE('Sum = ' || sum);

END;

/
```

Output:

```
Sum = 30
```

Experiment 13: Program to Check Even or Odd

Aim:
To write a PL/SQL program to check whether a number is even or odd.

Objective:
To demonstrate conditional statements in PL/SQL.

Code:

```
DECLARE

  num NUMBER := 5;

BEGIN

  IF MOD(num, 2) = 0 THEN

    DBMS_OUTPUT.PUT_LINE(num || ' is Even');

  ELSE

    DBMS_OUTPUT.PUT_LINE(num || ' is Odd');

  END IF;

END;

/
```

Output:

```
5 is Odd
```

Experiment 14: Program to Find Factorial of a Number

Aim:
To write a PL/SQL program to find the factorial of a given number.

Objective:
To demonstrate the use of loops in PL/SQL.

Code:

```
DECLARE

  num NUMBER := 5;

  fact NUMBER := 1;

BEGIN

  FOR i IN 1..num LOOP

    fact := fact * i;

  END LOOP;

  DBMS_OUTPUT.PUT_LINE('Factorial = ' || fact);

END;

/
```

Output:

```
Factorial = 120
```

Experiment 15: Program to Reverse a String

Aim:
To write a PL/SQL program to reverse a given string.

Objective:
To manipulate strings using PL/SQL.

Code:

```
DECLARE

  str VARCHAR2(100) := 'PLSQL';

  rev_str VARCHAR2(100) := '';

BEGIN

  FOR i IN REVERSE 1..LENGTH(str) LOOP

    rev_str := rev_str || SUBSTR(str, i, 1);

  END LOOP;

  DBMS_OUTPUT.PUT_LINE('Reversed String = ' || rev_str);

END;

/
```

Output:

```
Reversed String = LSQLP
```

Experiment 16: Display First 10 Natural Numbers Using a Loop

Aim:
To write a PL/SQL program to display the first 10 natural numbers using a loop.

Objective:
To demonstrate loops in PL/SQL.

Code:

```
BEGIN
  FOR i IN 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE(i);
  END LOOP;
END;
/
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Experiment 17: Program to Find Maximum of Two Numbers

Aim:
To write a PL/SQL program to find the maximum of two numbers.

Objective:
To demonstrate conditional comparisons in PL/SQL.

Code:

```
DECLARE

  num1 NUMBER := 15;

  num2 NUMBER := 20;

BEGIN

  IF num1 > num2 THEN

    DBMS_OUTPUT.PUT_LINE('Max = ' || num1);

  ELSE

    DBMS_OUTPUT.PUT_LINE('Max = ' || num2);

  END IF;

END;

/
```

Output:

```
Max = 20
```

Experiment 18: Check if a String is a Palindrome

Aim:
To write a PL/SQL program to check if a string is a palindrome.

Objective:
To manipulate and compare strings in PL/SQL.

Code:

```
DECLARE

  str VARCHAR2(100) := 'MADAM';

  rev_str VARCHAR2(100) := '';

BEGIN

  FOR i IN REVERSE 1..LENGTH(str) LOOP

    rev_str := rev_str || SUBSTR(str, i, 1);

  END LOOP;


  IF str = rev_str THEN

    DBMS_OUTPUT.PUT_LINE(str || ' is a palindrome');

  ELSE

    DBMS_OUTPUT.PUT_LINE(str || ' is not a palindrome');

  END IF;

END;

/
```

Output:

```
MADAM is a palindrome
```

Experiment 19: Program to Calculate Fibonacci Series

Aim:
To write a PL/SQL program to generate the Fibonacci series up to a given number of terms.

Objective:
To demonstrate recursion and sequence generation in PL/SQL.

Code:

```
DECLARE
  num1 NUMBER := 0;
  num2 NUMBER := 1;
  num3 NUMBER;
  n NUMBER := 10;  -- Number of terms
BEGIN
  DBMS_OUTPUT.PUT_LINE(num1);
  DBMS_OUTPUT.PUT_LINE(num2);

  FOR i IN 3..n LOOP
    num3 := num1 + num2;
    DBMS_OUTPUT.PUT_LINE(num3);
    num1 := num2;
    num2 := num3;
  END LOOP;
END;
/
```

Output:

```
0

1

1

2

3

5

8

13

21

34
```

Experiment 20: Check if a Number is an Armstrong Number

Aim:
To write a PL/SQL program to check if a given number is an Armstrong number.

Objective:
To implement mathematical logic in PL/SQL to check if the sum of the cubes of the digits of a number is equal to the number itself.

Code:

```
DECLARE
  num NUMBER := 153;
  temp NUMBER;
  digit NUMBER;
  sum NUMBER := 0;
BEGIN
  temp := num;

  WHILE temp > 0 LOOP
    digit := MOD(temp, 10);
    sum := sum + POWER(digit, 3);
    temp := FLOOR(temp / 10);
  END LOOP;

  IF sum = num THEN
    DBMS_OUTPUT.PUT_LINE(num || ' is an Armstrong number');
  ELSE
```

```
    DBMS_OUTPUT.PUT_LINE(num || ' is not an Armstrong number');
  END IF;
END;
/
```

Output:

```
153 is an Armstrong number
```