RDBMS LAB
BCA-DS-552

Manav Rachna International Institute of Research and Studies

School of Computer Applications

Department of Computer Applications

Submitted By	
Student Name	
Roll No	
Programme	**Bachelor of Computer Applications**
Semester	**5th Semester**
Section	**B**
Department	**Computer Applications**
Batch	**2022-25**
Submitted To	
Faculty Name	

MANAV RACHNA
||vidyanatariksha||
Manav Rachna International Institute of Research and Studies
Deemed to be University under section 3 of the UGC Act, 1956

SCHOOL

OF

COMPUTER APPLICATIONS

Manav Rachna Campus Rd, Gadakhor Basti Village, Sector 43, Faridabad, Haryana 121004

MANAV RACHNA
vidyanatariksha
Manav Rachna International Institute of Research and Studies
Deemed to be University under section 3 of the UGC Act, 1956

SCHOOL OF COMPUTER APPLICATIONS

Index

S. No	**Program**	**Date of Submission**	**Remarks**	**Signature**
1.	Create the following tables Customer and Orders	01/08/24		
2.	Insert five records for each table	02/08/24		
3.	Customer_SID column in the ORDERS table is a foreign key pointing to the SID column in the CUSTOMER table.	08/08/24		
4.	Insert five records for both tables	09/08/24		
5.	List the details of the customers along with the amount.	16/08/24		
6.	List the customers whose names end with “s”.	21/08/24		
7.	List the orders where amount is between 21000 and 30000	23/08/24		
8.	List the orders where amount is increased by 500 and replace with name “new amount”.	28/08/24		
9.	Display the order_id and total amount of orders	30/08/24		
10.	Calculate the total amount of orders that has more than 15000.	04/09/24		
11.	Display all the string functions used in SQL.	06/09/24		
12.	Create the following tables Student and Student1	13/09/24		
13.	Display all the contents of student and student1 using union clause.	16/09/24		
14.	Find out the intersection of student and student1 tables.	20/09/24		
15.	Display the names of student and student1 tables using left, right ,inner and full join.	23/09/24		
16.	Write a PL/SQL block to calculate total salary of employee having employee number 100.	27/09/24		
17.	Write a PL/SQL code to find the greatest of three numbers.	30/09/24		
18.	Write a PL/SQL code to print the numbers from 1 to n.	04/10/24		
19.	Write a PL/SQL code to reverse a string using for loop.	07/10/24		
20	Write a PL/SQL code to find the sum of n numbers.	14/10/24		

21.	Consider a PL/SQL code to display the empno, ename, job of employees of department number 10.	21/10/24		
22.	Consider a PL/SQL code to display the employee number & name of top five highest paid employees.	25/10/24		
23.	Consider a PL/SQL procedure that accepts 2 numbers & return addition, subtraction, multiplication & division of two numbers using stored procedure AND local procedure.	04/11/24		
24.	Consider a PL/SQL code that accepts 2 numbers & return addition, subtraction, multiplication & division of two numbers using stored functions and local function.	08/11/24		
25.	Write a PL/SQL block to show the use of NO_DATA FOUND exception.	11/11/24		
26.	Write a PL/SQL block to show the use of TOO_MANY ROWS exception.	18/11/24		
27.	Write a PL/SQL block to show the use of ZERO_DIVIDE exception.	18/11/24		
28.	To create a trigger on the emp table, which store the empno& operation in the table auditor for each operation i.e. Insert, Update & Delete.	25/11/24		
29.	To create a trigger so that no operation can be performed on emp table.	25/11/24		

Experiment No: 1

Experiment 1:- Create the following tables:

Customer

Column name	**Data type**	**Size**	**Constraint**
SID	Varchar2	4	Primary Key
First_Name	Char	20	
Last_name	Char	20	

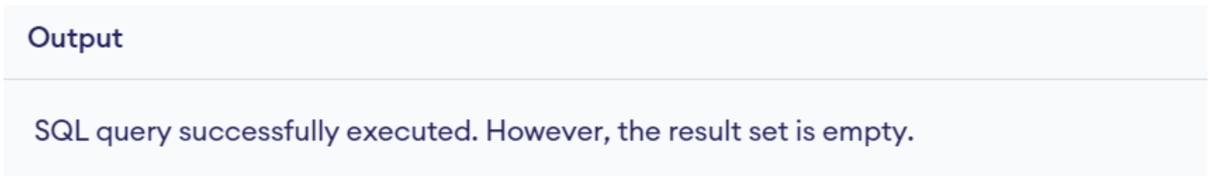
Orders

Column name	**Data type**	**Size**	**Constraint**
Order_ID	Varchar2	4	Primary Key
Order_date	Char	20	
Customer_SID	Varchar2	20	Foreign Key
Amount	Number		Check > 20000

Input

```
Create table Customers(
  SID Varchar2(4) Primary key,
  First_Name Char(20),
  Last_Name Char(20)
  );
Create table Orders(
  Order_Id Varchar2(4) Primary key,
  Order_date Char(20),
  Customer_SID Varchar2(20),
  Amount Number,
  CONSTRAINT fk_customer FOREIGN KEY (Customer_SID) REFERENCES 
Customer(SID),
  CONSTRAINT chk_amount CHECK (Amount > 2000)
```

Output



```
Output

SQL query successfully executed. However, the result set is empty.
```

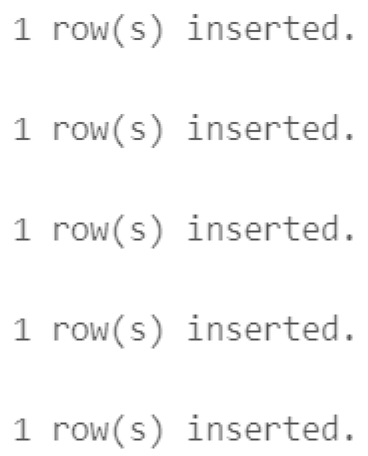
Experiment No: 2

Experiment 2:- Insert five records for each table

Input

```
INSERT INTO Customers (SID, First_Name, Last_Name) VALUES ('C1', 'John', 'Doe');
INSERT INTO Customers (SID, First_Name, Last_Name) VALUES ('C2', 'Robert', 'Luna');
INSERT INTO Customers (SID, First_Name, Last_Name) VALUES ('C3', 'David', 'Robinson');
INSERT INTO Customers (SID, First_Name, Last_Name) VALUES ('C4', 'Alice', 'Smith');
INSERT INTO Customers (SID, First_Name, Last_Name) VALUES ('C5', 'Betty', 'Doe');

INSERT INTO Orders (Order_Id, Order_date, Customer_SID, Amount)
VALUES ('O1', '1 Sep', 'C1', 3000);
INSERT INTO Orders (Order_Id, Order_date, Customer_SID, Amount)
VALUES  ('O2', '2 Sep', 'C2', 4500),
INSERT INTO Orders (Order_Id, Order_date, Customer_SID, Amount)
VALUES  ('O3', '3 Sep', 'C3', 5000),
INSERT INTO Orders (Order_Id, Order_date, Customer_SID, Amount)
VALUES  ('O4', '4 Sep', 'C4', 6000),
INSERT INTO Orders (Order_Id, Order_date, Customer_SID, Amount)
VALUES ('O5', '5 Sep', 'C5', 3500);
```

Output



```
1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.
```

Experiment No: 3

Experiment 3:- Customer_SID column in the ORDERS table is a foreign key pointing to the SID column in the CUSTOMER table.

This was already handled when the Orders table was created with a FOREIGN KEY constraint.

Experiment No: 4

Experiment 4:- Insert five records for both tables

The records for both the Customer and Orders tables were inserted.

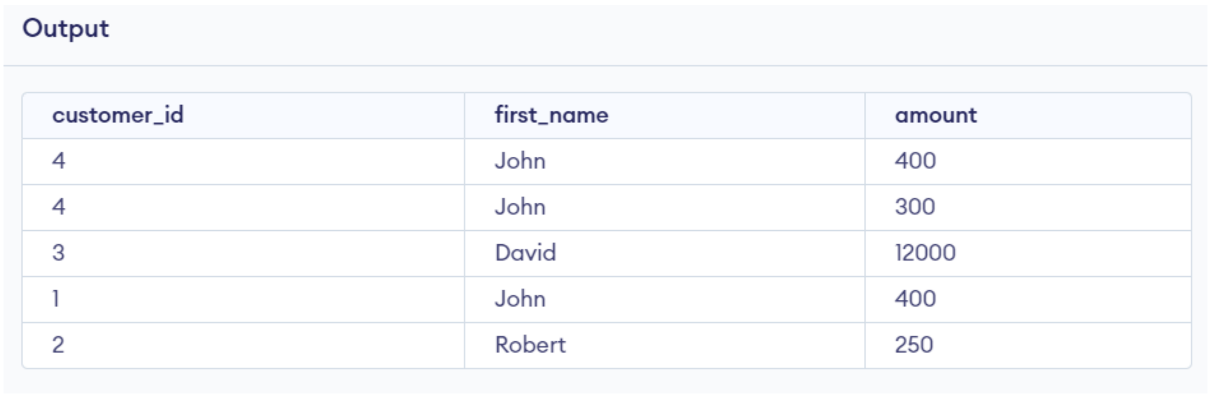
Experiment No: 5

Experiment 5:- List the details of the customers along with the amount.

Input

```
SELECT Customers.customer_id, Customers.first_name, Orders.amount

FROM Customers

JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

Output



Output

customer_id	first_name	amount
4	John	400
4	John	300
3	David	12000
1	John	400
2	Robert	250

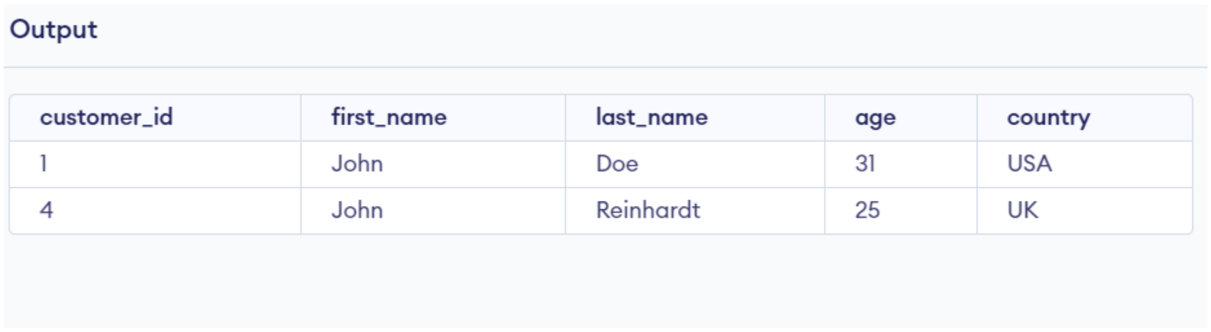
Experiment No: 6

Experiment 6:- List the customers whose names end with “s”.

Input

```
SELECT *
from Customers
where first_name like '%n';
```

Output



Output

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
4	John	Reinhardt	25	UK

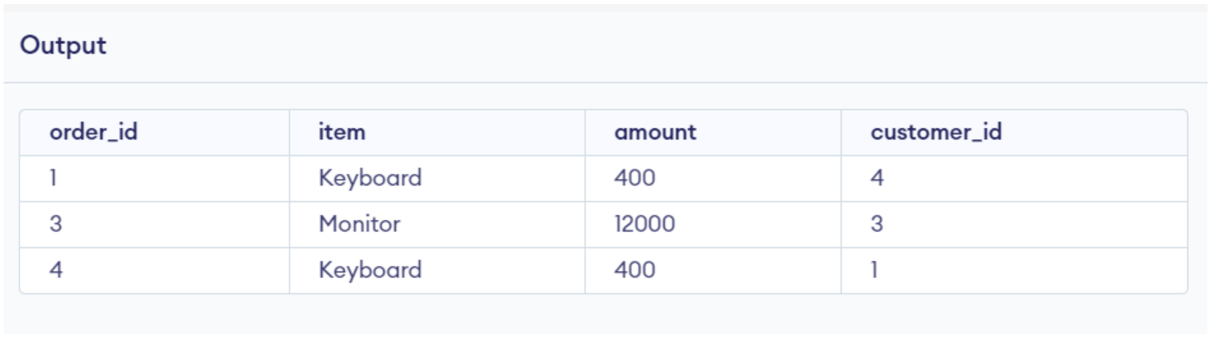
Experiment No: 7

Experiment 7:- List the orders where amount is between 21000 and 30000

Input

```
SELECT *
from Orders
where amount between 400 and 12000;
```

Output



Output

order_id	item	amount	customer_id
1	Keyboard	400	4
3	Monitor	12000	3
4	Keyboard	400	1

Experiment No: 8

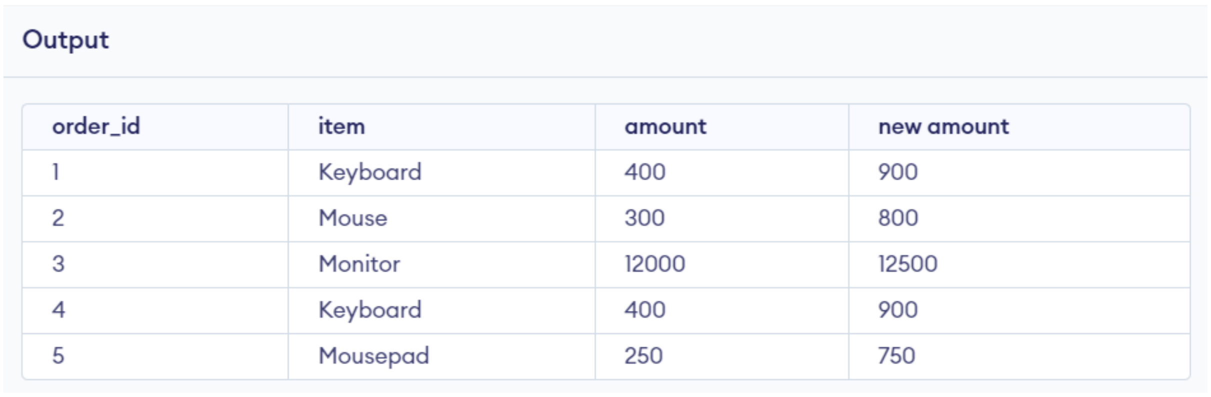
Experiment 8:- List the orders where amount is increased by 500 and replace with name “new amount”.

Input

SELECT order_id, item,amount, amount + 500 AS "new amount"

FROM Orders;

Output



Output

order_id	item	amount	new amount
1	Keyboard	400	900
2	Mouse	300	800
3	Monitor	12000	12500
4	Keyboard	400	900
5	Mousepad	250	750

Experiment No: 9

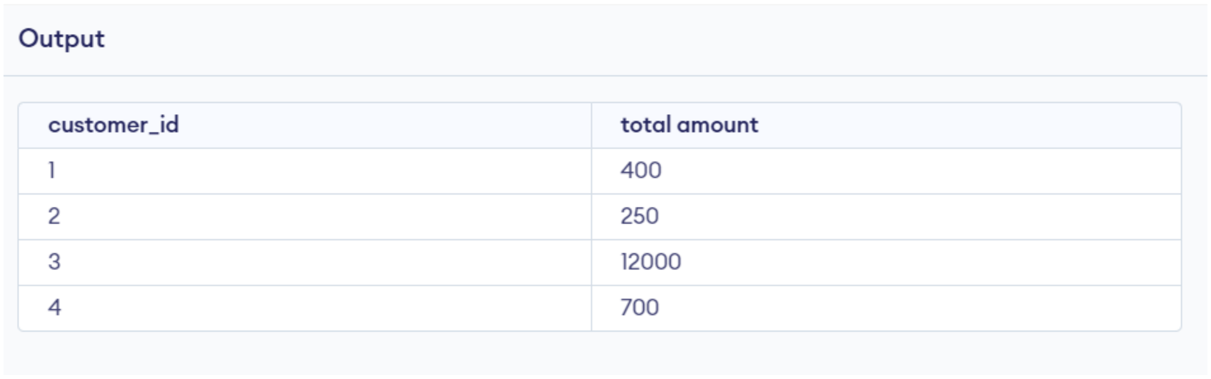
Experiment 8:- Display the order_id and total amount of orders.

Input

SELECT customer_id, sum(amount) AS "total amount"

FROM Orders

group by customer_id



Output

customer_id	total amount
1	400
2	250
3	12000
4	700

Experiment No: 10

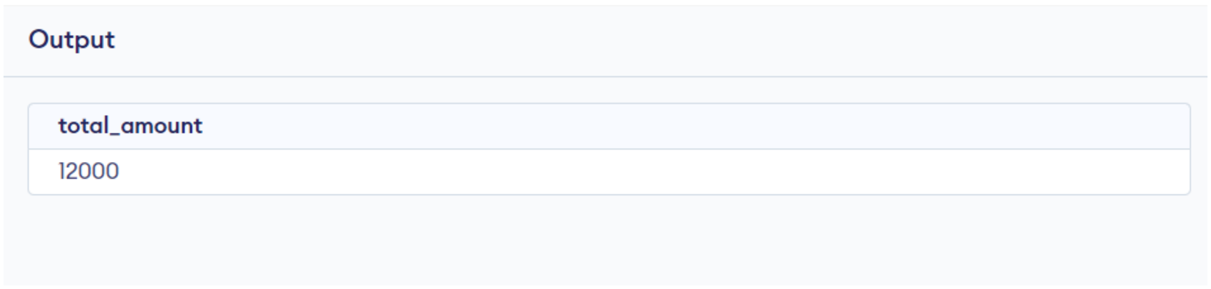
Experiment 10:- Calculate the total amount of orders that has more than 15000.

Input

SELECT sum(amount) as total_amount

FROM Orders

where amount>400;



Output

```
total_amount
12000
```

Experiment No: 11

<u>11: Display all the string functions used in SQL</u>.

UPPER(string) - Converts the string to uppercase.

LOWER(string) - Converts the string to lowercase.

SUBSTR(string, start_position, length) - Extracts a substring. LENGTH(string) - Returns the length of a string.

TRIM(string) - Removes spaces from both sides of the string. CONCAT(string1, string2) - Concatenates two strings.

REPLACE(string, search_string, replace_string) - Replaces occurrences of search string with replace string.

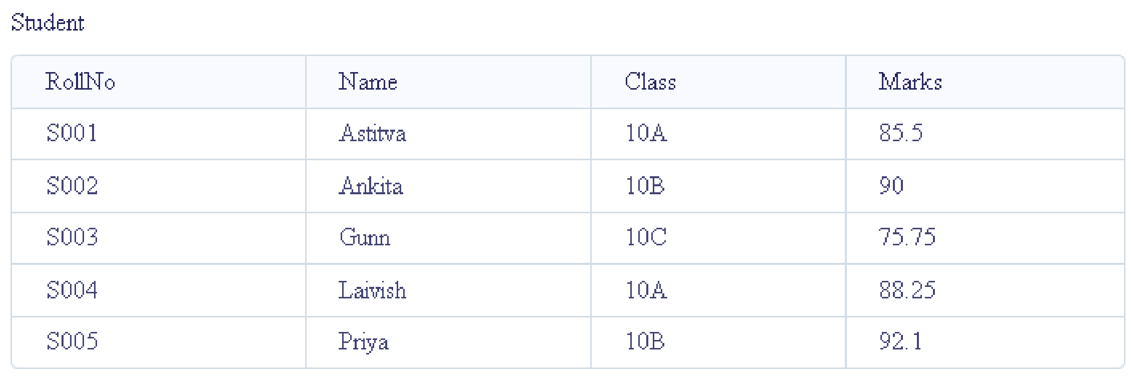
INSTR(string, substring) - Finds the position of a substring in a string.

Experiment No: 12

12: Create the following tables.

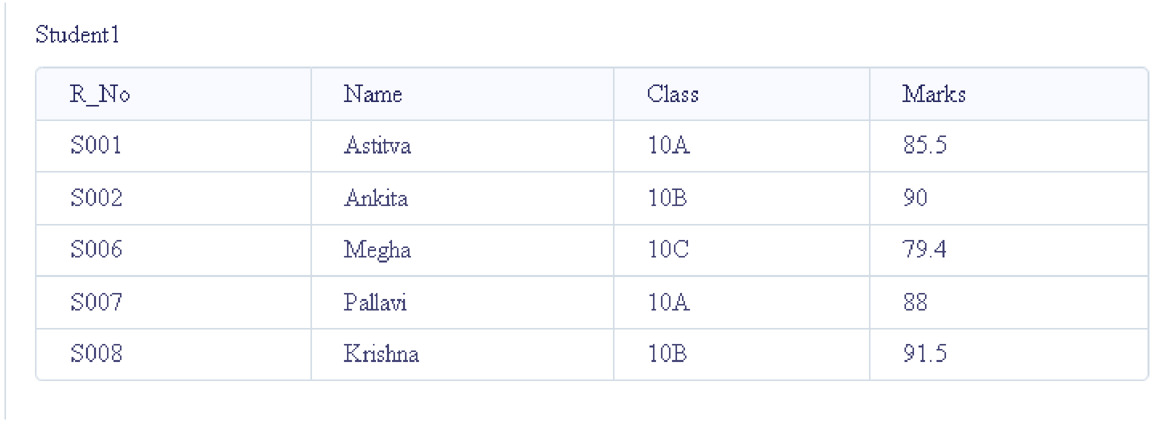
```
CREATE TABLE Student (
 RollNo VARCHAR2(20) PRIMARY KEY,
Name CHAR(20), Class VARCHAR2(20),
 Marks NUMBER(6,2)
  );

CREATE TABLE Student1 (
 R_No VARCHAR2(20) PRIMARY KEY,
 Name CHAR(20),
 Class VARCHAR2(20),
 Marks NUMBER(6,2)
);

INSERT INTO Student (RollNo, Name, Class, Marks) VALUES ('S001',
'Astitva', '10A', 85.50);
 INSERT INTO Student (RollNo, Name, Class, Marks) VALUES ('S002',
'Ankita', '10B', 90.00);
 INSERT INTO Student (RollNo, Name, Class, Marks) VALUES ('S003',
'Gunn', '10C', 75.75);
 INSERT INTO Student (RollNo, Name, Class, Marks) VALUES ('S004',
'Laivish', '10A', 88.25);
 INSERT INTO Student (RollNo, Name, Class, Marks) VALUES ('S005',
'Priya', '10B', 92.10);

INSERT INTO Student1 (R_No, Name, Class, Marks) VALUES ('S001',
'Astitva',
'10A', 85.50);
```

```
INSERT INTO Student1 (R_No, Name, Class, Marks) VALUES ('S002',
'Ankita',

'10B', 90.00);

INSERT INTO Student1 (R_No, Name, Class, Marks) VALUES ('S006',
'Megha',

'10C', 79.40);

INSERT INTO Student1 (R_No, Name, Class, Marks) VALUES ('S007',
'Pallavi',

'10A', 88.00);

INSERT INTO Student1 (R_No, Name, Class, Marks) VALUES ('S008',
'Krishna',

'10B', 91.50)
```



Student

RollNo	Name	Class	Marks
S001	Astitva	10A	85.5
S002	Ankita	10B	90
S003	Gunn	10C	75.75
S004	Laivish	10A	88.25
S005	Priya	10B	92.1



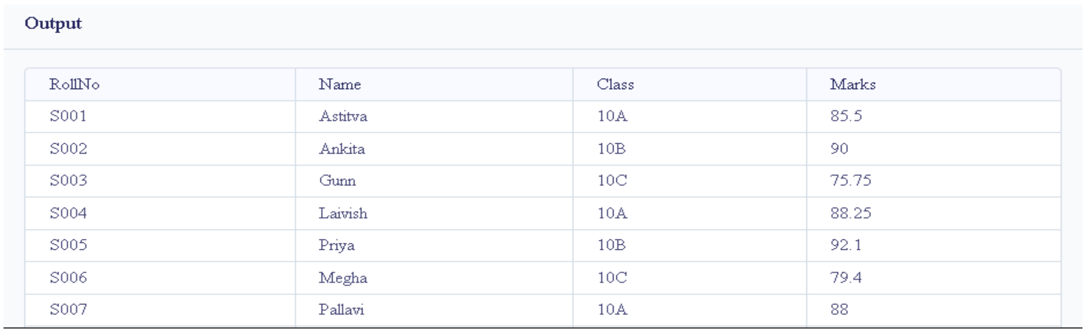
Student1

R_No	Name	Class	Marks
S001	Astitva	10A	85.5
S002	Ankita	10B	90
S006	Megha	10C	79.4
S007	Pallavi	10A	88
S008	Krishna	10B	91.5

Experiment No: 13

13: Display all the contents of student and student1 using union clause.

```
SELECT * FROM Student
UNION
 SELECT * FROM Student1;
```



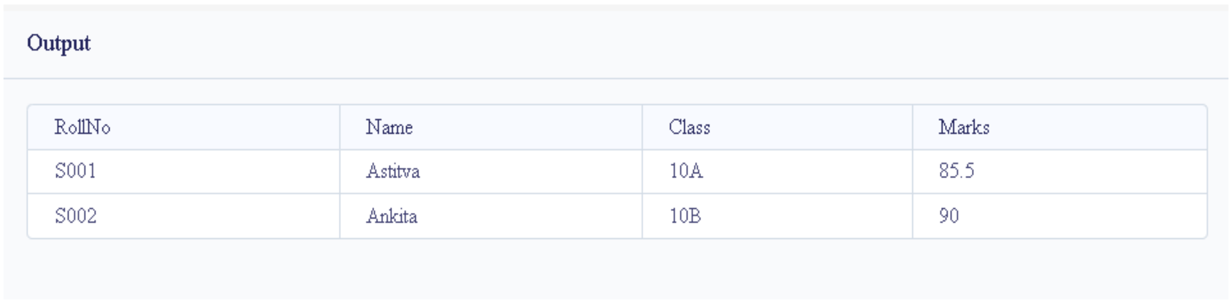
Output

RollNo	Name	Class	Marks
S001	Astitva	10A	85.5
S002	Ankita	10B	90
S003	Gunn	10C	75.75
S004	Laivish	10A	88.25
S005	Priya	10B	92.1
S006	Megha	10C	79.4
S007	Pallavi	10A	88

Experiment No: 14

14: Find out the intersection of student and student1 tables.

```
SELECT * FROM Student
INTERSECT
SELECT * FROM Student1;
```



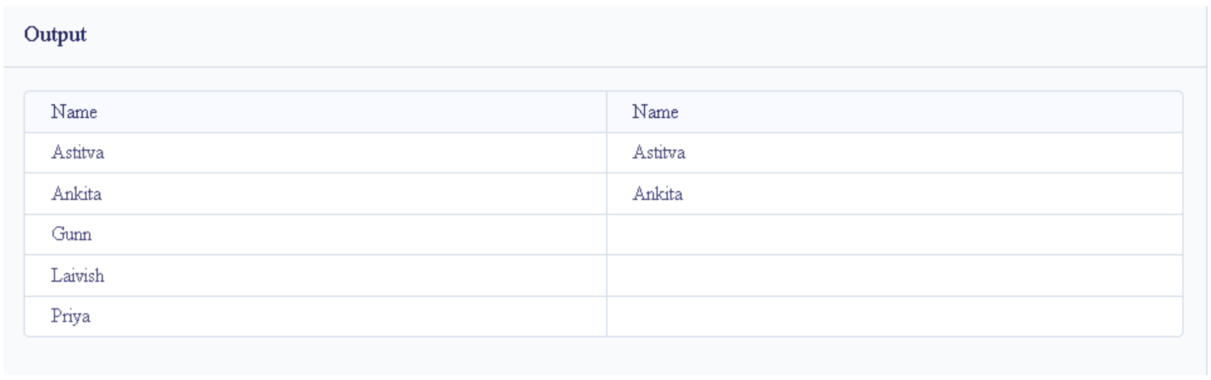
Output

RollNo	Name	Class	Marks
S001	Astitva	10A	85.5
S002	Ankita	10B	90

Experiment No: 15

15: Display the names of student and student1 tables using left, right ,inner and full join.

```
SELECT Student.Name, Student1.Name
 FROM Student
 LEFT JOIN Student1 ON Student.RollNo = Student1.R_No;
```

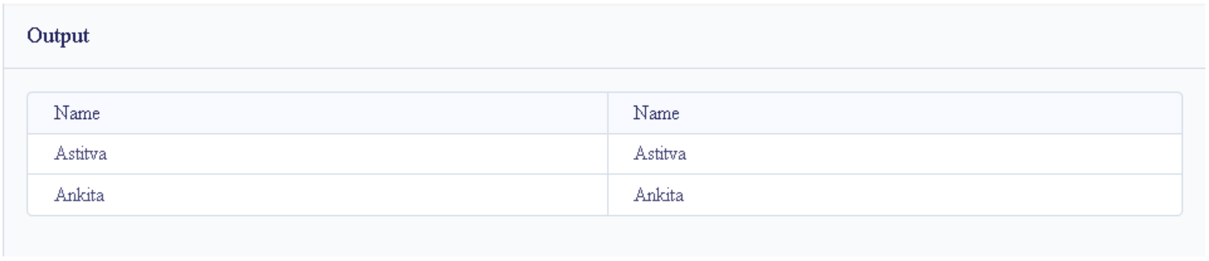


Output

Name	Name
Astitva	Astitva
Ankita	Ankita
Gunn	
Laivish	
Priya	

RIGHT and FULL OUTER JOINs are not currently supported.

```
SELECT Student.Name, Student1.Name
 FROM Student
INNER JOIN Student1 ON Student.RollNo = Student1.R_No;
```



Output

Name	Name
Astitva	Astitva
Ankita	Ankita

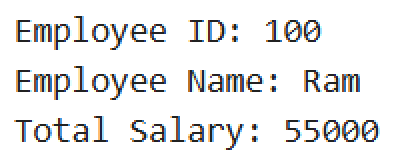
Experiment No: 16

Experiment 16:- Write a PL/SQL block to calculate total salary of employee having employee number 100.

Input

```
Create table Customers( DECLARE
   emp_id NUMBER := 100;
   emp_name VARCHAR2(50) := 'Ram';
   base_salary NUMBER := 50000;  -- Example base salary
   total_salary NUMBER;
BEGIN
   -- Simulate calculation of total salary (for example, adding a bonus)
   total_salary := base_salary + (base_salary * 0.10);  -- Adding 10% bonus

   -- Display the total salary
   DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_id);
   DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_name);
   DBMS_OUTPUT.PUT_LINE('Total Salary: ' || total_salary);
END;
/
```

Output



```
Employee ID: 100
Employee Name: Ram
Total Salary: 55000
```

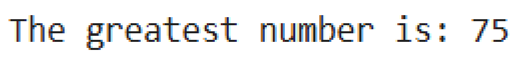
Experiment No: 17

Experiment 17:- Write a PL/SQL code to find the greatest of three numbers.

Input

```
DECLARE
   num1 NUMBER := 25;
   num2 NUMBER := 75;
   num3 NUMBER := 50;
   greatest NUMBER;
BEGIN
   -- Compare the three numbers to find the greatest
   IF (num1 >= num2) AND (num1 >= num3) THEN
      greatest := num1;
   ELSIF (num2 >= num1) AND (num2 >= num3) THEN
      greatest := num2;
   ELSE
      greatest := num3;
   END IF;

   -- Display the greatest number
   DBMS_OUTPUT.PUT_LINE('The greatest number is: ' || greatest);
END;
/
```

Output



```
The greatest number is: 75
```

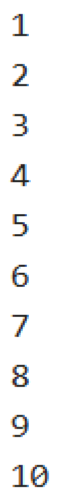
Experiment No: 18

Experiment 18:- Write a PL/SQL code to print the numbers from 1 to n.

Input

```
DECLARE
   n NUMBER := 10;  -- Set the value of n here
   i NUMBER := 1;   -- Initialize the counter
BEGIN
   WHILE i <= n LOOP
      DBMS_OUTPUT.PUT_LINE(i);
      i := i + 1;  -- Increment the counter
   END LOOP;
END;
/
```

Output



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

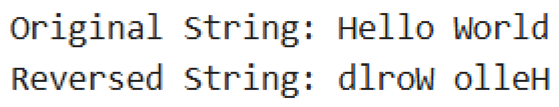
Experiment No: 19

Experiment 19:- Write a PL/SQL code to reverse a string using for loop.

Input

```
DECLARE
   original_string VARCHAR2(100) := 'Hello World';  -- Input string to reverse
   reversed_string VARCHAR2(100) := '';
BEGIN
   -- Loop through the original string in reverse order
   FOR i IN REVERSE 1..LENGTH(original_string) LOOP
      reversed_string := reversed_string || SUBSTR(original_string, i, 1);
   END LOOP;

   -- Display the reversed string
   DBMS_OUTPUT.PUT_LINE('Original String: ' || original_string);
   DBMS_OUTPUT.PUT_LINE('Reversed String: ' || reversed_string);
END;
/
```

Output



```
Original String: Hello World
Reversed String: dlroW olleH
```

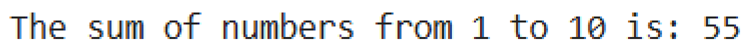
Experiment No: 20

Experiment 20:- Write a PL/SQL code to find the sum of n numbers.

Input

```
DECLARE
   n NUMBER := 10;     -- Set the value of n here
   sum NUMBER := 0;    -- Initialize the sum to 0
BEGIN
   FOR i IN 1..n LOOP
      sum := sum + i;  -- Add each number from 1 to n
   END LOOP;

   -- Display the result
   DBMS_OUTPUT.PUT_LINE('The sum of numbers from 1 to ' || n || ' is: ' || sum);
END;
/
```

Output



```
The sum of numbers from 1 to 10 is: 55
```

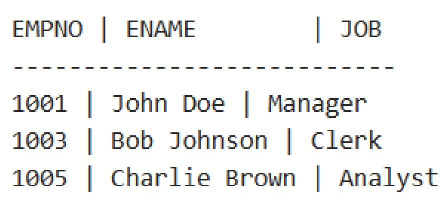
Experiment No: 21

Experiment 21:- Consider a PL/SQL code to display the empno, ename, job of employees of department number 10.

Input

```
DECLARE
  TYPE emp_record IS RECORD (
    empno   NUMBER,
    ename   VARCHAR2(50),
    job     VARCHAR2(50),
    deptno  NUMBER  );
  TYPE emp_table IS TABLE OF emp_record INDEX BY PLS_INTEGER;
  employees emp_table;
BEGIN
  employees(1) := emp_record(1001, 'John Doe', 'Manager', 10);
  employees(2) := emp_record(1002, 'Jane Smith', 'Analyst', 20);
  employees(3) := emp_record(1003, 'Bob Johnson', 'Clerk', 10);
  employees(4) := emp_record(1004, 'Alice Davis', 'Developer', 30);
  employees(5) := emp_record(1005, 'Charlie Brown', 'Analyst', 10);
  DBMS_OUTPUT.PUT_LINE('EMPNO | ENAME       | JOB');
  DBMS_OUTPUT.PUT_LINE('--------------------------');
  FOR i IN employees.FIRST .. employees.LAST LOOP
    IF employees(i).deptno = 10 THEN
      DBMS_OUTPUT.PUT_LINE(employees(i).empno || ' | ' || employees(i).ename || ' | ' || employees(i).job);
    END IF;
  END LOOP;
END;   /
```

Output



```
EMPNO | ENAME          | JOB
---------------------------
1001 | John Doe | Manager
1003 | Bob Johnson | Clerk
1005 | Charlie Brown | Analyst
```

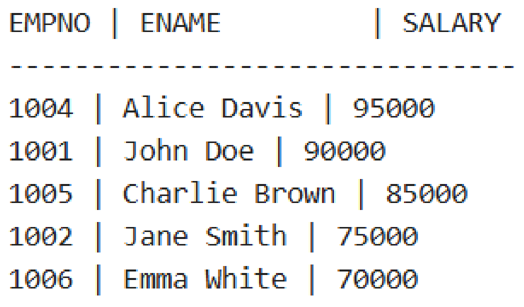
Experiment No: 22

Experiment 22:- Consider a PL/SQL code to display the employee number & name of top five highest paid employees.

Input

```
DECLARE
  -- Define a PL/SQL table type to hold employee records
  TYPE emp_record IS RECORD (
    empno   NUMBER,
    ename   VARCHAR2(50),
    salary  NUMBER
  );

  TYPE emp_table IS TABLE OF emp_record INDEX BY PLS_INTEGER;
  employees emp_table;

  -- Variable to store sorted employees
  sorted_employees emp_table;

BEGIN
  -- Populate the table with sample data
  employees(1) := emp_record(1001, 'John Doe', 90000);
  employees(2) := emp_record(1002, 'Jane Smith', 75000);
  employees(3) := emp_record(1003, 'Bob Johnson', 60000);
  employees(4) := emp_record(1004, 'Alice Davis', 95000);
  employees(5) := emp_record(1005, 'Charlie Brown', 85000);
  employees(6) := emp_record(1006, 'Emma White', 70000);
  employees(7) := emp_record(1007, 'Liam Green', 65000);

  -- Sort the employees based on salary (simple bubble sort)
  DECLARE
    i PLS_INTEGER;
    j PLS_INTEGER;
    temp emp_record;
  BEGIN
    FOR i IN employees.FIRST .. employees.LAST LOOP
      FOR j IN i + 1 .. employees.LAST LOOP
        IF employees(i).salary < employees(j).salary THEN
          temp := employees(i);
          employees(i) := employees(j);
          employees(j) := temp;
        END IF;
      END LOOP;
    END LOOP;
    -- Store the top 5 in sorted_employees
    FOR i IN 1 .. 5 LOOP
      sorted_employees(i) := employees(i);
    END LOOP;
  END;
  -- Display the top 5 highest paid employees
  DBMS_OUTPUT.PUT_LINE('EMPNO | ENAME          | SALARY');
  DBMS_OUTPUT.PUT_LINE('------------------------------');
  FOR i IN sorted_employees.FIRST .. sorted_employees.LAST LOOP
    DBMS_OUTPUT.PUT_LINE(sorted_employees(i).empno || ' | ' ||
                  sorted_employees(i).ename || ' | ' ||
                  sorted_employees(i).salary);
  END LOOP;
END;
```

Output



```
EMPNO | ENAME          | SALARY
-------------------------------
1004 | Alice Davis | 95000
1001 | John Doe | 90000
1005 | Charlie Brown | 85000
1002 | Jane Smith | 75000
1006 | Emma White | 70000
```

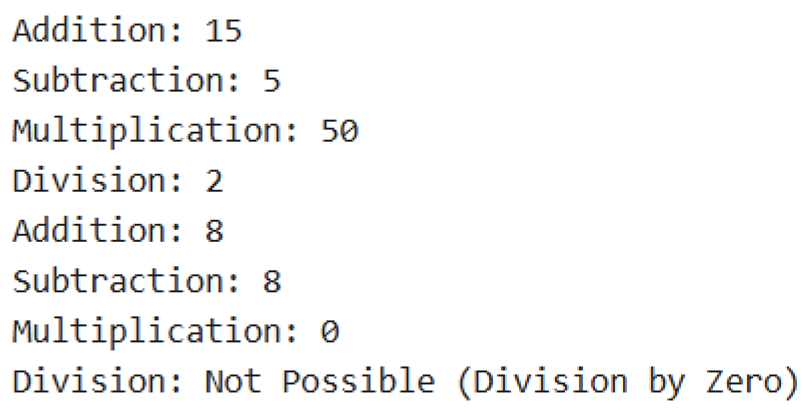
Experiment No: 23

Experiment 23:- Consider a PL/SQL procedure that accepts 2 numbers & return addition, subtraction, multiplication & division of two numbers using stored procedure AND local procedure.

Input

```
CREATE OR REPLACE PROCEDURE calculate_operations (
  num1 IN NUMBER,
  num2 IN NUMBER
)
IS
  PROCEDURE local_operations (
    a IN NUMBER,
    b IN NUMBER
  )
  IS
    add_result NUMBER;
    sub_result NUMBER;
    mul_result NUMBER;
    div_result NUMBER;
  BEGIN
    -- Perform calculations
    add_result := a + b;
    sub_result := a - b;
    mul_result := a * b;

    -- Check for division by zero
    IF b != 0 THEN
      div_result := a / b;
    ELSE
      div_result := NULL; -- Division not possible
    END IF;
    -- Display results
    DBMS_OUTPUT.PUT_LINE('Addition: ' || add_result);
    DBMS_OUTPUT.PUT_LINE('Subtraction: ' || sub_result);
    DBMS_OUTPUT.PUT_LINE('Multiplication: ' || mul_result);
    IF div_result IS NOT NULL THEN
      DBMS_OUTPUT.PUT_LINE('Division: ' || div_result);
    ELSE
      DBMS_OUTPUT.PUT_LINE('Division: Not Possible (Division by Zero)');
    END IF;
  END local_operations;
BEGIN
  -- Call local procedure
  local_operations(num1, num2);
END calculate_operations;
/

BEGIN
  calculate_operations(10, 5); -- Example with valid inputs
  calculate_operations(8, 0); -- Example to test division by zero
END;
/
```

Output



```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2
Addition: 8
Subtraction: 8
Multiplication: 0
Division: Not Possible (Division by Zero)
```

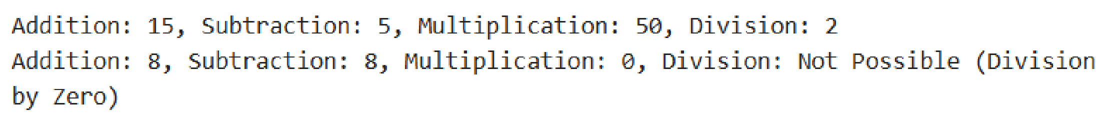
Experiment No: 24

Experiment 24:- Consider a PL/SQL code that accepts 2 numbers & return addition, subtraction, multiplication & division of two numbers using stored functions and local function.

Input

```
SET SERVEROUTPUT ON; -- Stored Function

CREATE OR REPLACE FUNCTION perform_operations(num1 IN NUMBER, num2 IN NUMBER)

RETURN VARCHAR2

IS    -- Local Function for Division

  FUNCTION divide_numbers(a IN NUMBER, b IN NUMBER) RETURN VARCHAR2 IS

  BEGIN

    IF b != 0 THEN

      RETURN TO_CHAR(a / b); -- Convert division result to string

    ELSE

      RETURN 'Not Possible (Division by Zero)'; -- Handle division by zero

    END IF;  END divide_numbers;

BEGIN

  RETURN 'Addition: ' || TO_CHAR(num1 + num2) || ', ' ||   'Subtraction: ' || TO_CHAR(num1 -
num2) || ', ' || 'Multiplication: ' || TO_CHAR(num1 * num2) || ', ' ||    'Division: ' ||
divide_numbers(num1, num2);

END perform_operations; /

DECLARE  -- Anonymous Block to Call the Function

  result VARCHAR2(500);

BEGIN

  result := perform_operations(10, 5); -- Call with valid inputs

  DBMS_OUTPUT.PUT_LINE(result);

  result := perform_operations(8, 0); -- Call with division by zero

  DBMS_OUTPUT.PUT_LINE(result);

END; /
```

Output



```
Addition: 15, Subtraction: 5, Multiplication: 50, Division: 2
Addition: 8, Subtraction: 8, Multiplication: 0, Division: Not Possible (Division
by Zero)
```

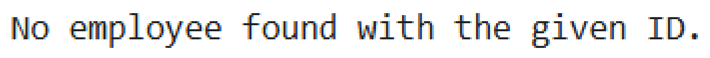
Experiment No: 25

Experiment 25:- Write a PL/SQL block to show the use of NO_DATA FOUND exception.

Input

```
CREATE TABLE emp (
    empno NUMBER PRIMARY KEY,
    ename VARCHAR2(100) );
-- Insert sample data
INSERT INTO emp (empno, ename) VALUES (1001, 'John Doe');
INSERT INTO emp (empno, ename) VALUES (1002, 'Jane Smith');
COMMIT;
SET SERVEROUTPUT ON;
BEGIN
   -- Declare variables
   DECLARE
      v_employee_name VARCHAR2(100);
   BEGIN
      -- Attempt to fetch an employee name for a non-existent employee ID
      SELECT ename
      INTO v_employee_name
      FROM emp
      WHERE empno = 9999; -- This employee ID does not exist
      -- If no exception occurs, display the employee name
      DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_employee_name);
   EXCEPTION
      WHEN NO_DATA_FOUND THEN -- Handle the exception
         DBMS_OUTPUT.PUT_LINE('No employee found with the given ID.');
   END;
END;
```

Output



```
No employee found with the given ID.
```

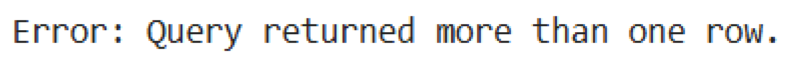
Experiment No: 26

Experiment 26:- Write a PL/SQL block to show the use of TOO_MANY ROWS exception.

Input

```
-- Create a sample table
CREATE TABLE emp (
    empno NUMBER PRIMARY KEY,
    ename VARCHAR2(100),
    deptno NUMBER
);
-- Insert sample data
INSERT INTO emp (empno, ename, deptno) VALUES (1001, 'John Doe', 10);
INSERT INTO emp (empno, ename, deptno) VALUES (1002, 'Jane Smith', 10);
INSERT INTO emp (empno, ename, deptno) VALUES (1003, 'Alice Brown', 20);
COMMIT;
-- PL/SQL block to demonstrate TOO_MANY_ROWS exception
SET SERVEROUTPUT ON;
BEGIN
   -- Declare variables
   DECLARE
      v_employee_name VARCHAR2(100);
   BEGIN
      -- Attempt to fetch an employee name where multiple rows exist
      SELECT ename
      INTO v_employee_name
      FROM emp
      WHERE deptno = 10; -- More than one employee in department 10
      -- If no exception occurs, display the employee name
      DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_employee_name);
   EXCEPTION
      WHEN TOO_MANY_ROWS THEN
         -- Handle the exception
      DBMS_OUTPUT.PUT_LINE('Error: Query returned more than one row.');
  END;
END;
/
```

Output



```
Error: Query returned more than one row.
```

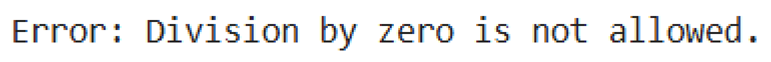
Experiment No: 27

Experiment 27:- Write a PL/SQL block to show the use of ZERO_DIVIDE exception.

Input

```
SET SERVEROUTPUT ON;
BEGIN
  DECLARE
    num1 NUMBER := 10;
    num2 NUMBER := 0;
    result NUMBER;
  BEGIN
    result := num1 / num2; -- Attempt division by zero
    DBMS_OUTPUT.PUT_LINE('Result: ' || result);
  EXCEPTION
    WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE('Error: Division by zero is not allowed.');
  END;
END;
/
```

Output



```
Error: Division by zero is not allowed.
```

Experiment No: 28

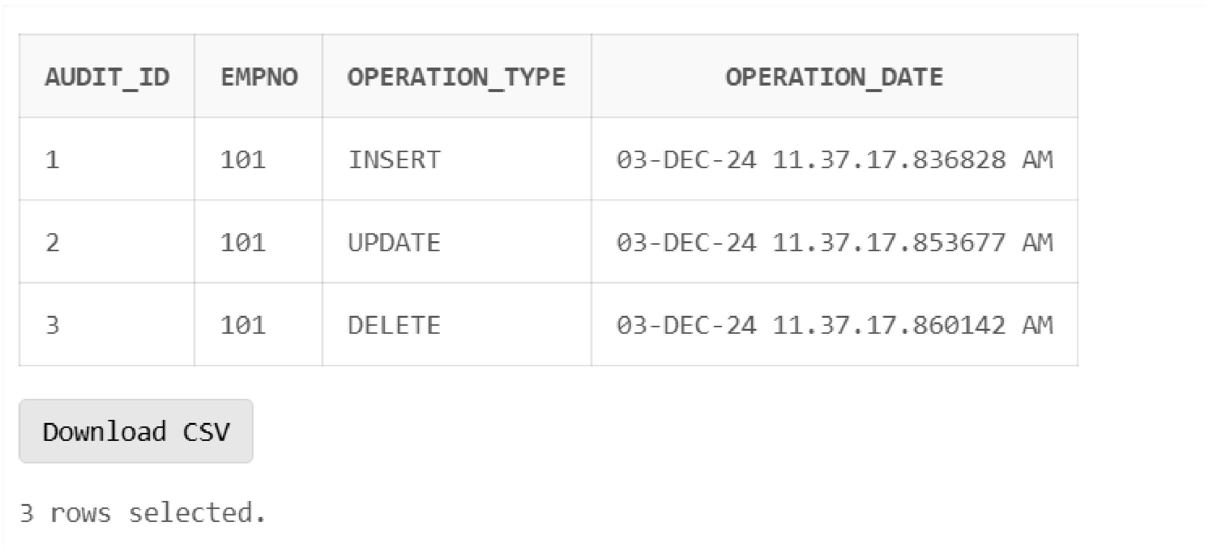
Experiment 28:- To create a trigger on the emp table, which store the empno& operation in the table auditor for each operation i.e. Insert, Update & Delete.exception.

Input

```
-- Create the employee table (emp)
CREATE TABLE emp (
    empno NUMBER PRIMARY KEY,
    ename VARCHAR2(50),
    job VARCHAR2(50)
);

-- Create the auditor table to store operations
CREATE TABLE auditor (
    audit_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    empno NUMBER,
    operation_type VARCHAR2(10),
    operation_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- Create a trigger to log INSERT operations
CREATE OR REPLACE TRIGGER emp_audit_trigger
AFTER INSERT OR UPDATE OR DELETE
ON emp
FOR EACH ROW
BEGIN
    -- Insert log into the auditor table
    IF INSERTING THEN
        INSERT INTO auditor (empno, operation_type)
        VALUES (:NEW.empno, 'INSERT');
    ELSIF UPDATING THEN
        INSERT INTO auditor (empno, operation_type)
        VALUES (:NEW.empno, 'UPDATE');
```

```
  ELSIF DELETING THEN
    INSERT INTO auditor (empno, operation_type)
    VALUES (:OLD.empno, 'DELETE');
  END IF;
END;
/
-- Insert a new employee
INSERT INTO emp (empno, ename, job) VALUES (101, 'John Doe', 'Manager');

-- Update an employee
UPDATE emp SET ename = 'Johnathan Doe' WHERE empno = 101;

-- Delete an employee
DELETE FROM emp WHERE empno = 101;
-- Check the auditor table for logged operations
SELECT * FROM auditor;
```

Output



AUDIT_ID	EMPNO	OPERATION_TYPE	OPERATION_DATE
1	101	INSERT	03-DEC-24 11.37.17.836828 AM
2	101	UPDATE	03-DEC-24 11.37.17.853677 AM
3	101	DELETE	03-DEC-24 11.37.17.860142 AM

```
3 rows selected.
```

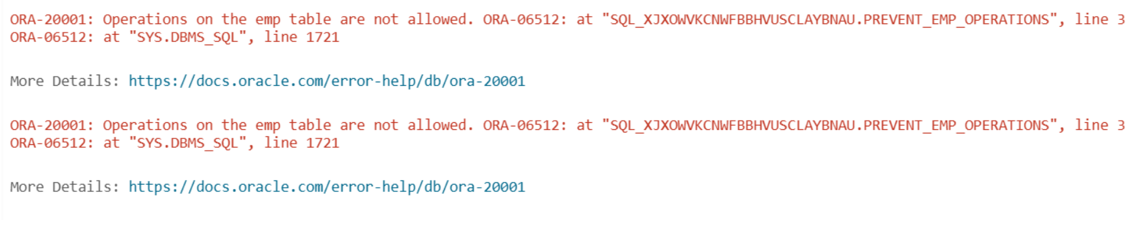
Experiment No: 29

Experiment 29:- To create a trigger so that no operation can be performed on emp table.

Input

```
-- Create the trigger to prevent all operations on the emp table

CREATE OR REPLACE TRIGGER prevent_emp_operations

BEFORE INSERT OR UPDATE OR DELETE

ON emp

BEGIN

    -- Raise an exception to prevent the operation

    RAISE_APPLICATION_ERROR(-20001, 'Operations on the emp table are not allowed.');

END;

/

-- Trying to insert a new employee (this will fail)

INSERT INTO emp (empno, ename, job) VALUES (101, 'John Doe', 'Manager');

-- Trying to update an employee (this will fail)

UPDATE emp SET ename = 'Jane Doe' WHERE empno = 101;


-- Trying to delete an employee (this will fail)

DELETE FROM emp WHERE empno = 101;
```

Output



```
ORA-20001: Operations on the emp table are not allowed. ORA-06512: at "SQL_XJXOWVKCNWFBBHVUSCLAYBNAU.PREVENT_EMP_OPERATIONS", line 3
ORA-06512: at "SYS.DBMS_SQL", line 1721

More Details: https://docs.oracle.com/error-help/db/ora-20001

ORA-20001: Operations on the emp table are not allowed. ORA-06512: at "SQL_XJXOWVKCNWFBBHVUSCLAYBNAU.PREVENT_EMP_OPERATIONS", line 3
ORA-06512: at "SYS.DBMS_SQL", line 1721

More Details: https://docs.oracle.com/error-help/db/ora-20001
```

Conclusion:

This trigger will effectively prevent any operation (INSERT, UPDATE, DELETE) from being performed on the emp table by raising an exception.