**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** | **Rohit Gautam** |
| **Roll No** | **22/FCA/BCA(DS&BDA)022** |
| **Programme** | **Bachelor of Computer Applications** |
| **Semester** | **5th Semester** |
| **Section** | **E** |
| **Department** | **Computer Applications** |
| **Batch** | **2022-25** |
|  | |
| **Submitted To** | |
| **Faculty Name** | **Ms. Iram Fatima** |

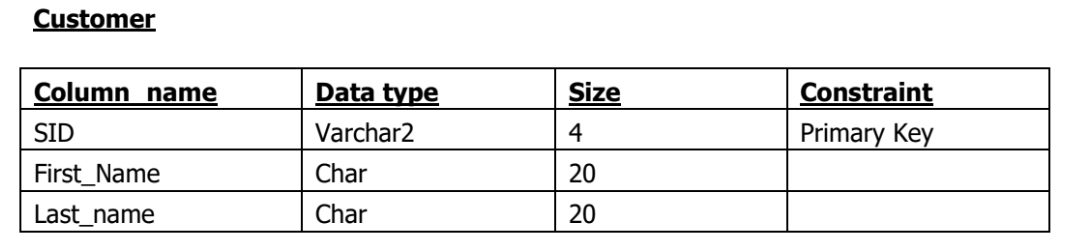
|  |  |
| --- | --- |
|  | **SCHOOL**  **OF**  **COMPUTER APPLICATIONS** |
| **Manav Rachna Campus Rd, Gadakhor Basti Village, Sector 43, Faridabad, Haryana 121004** | |

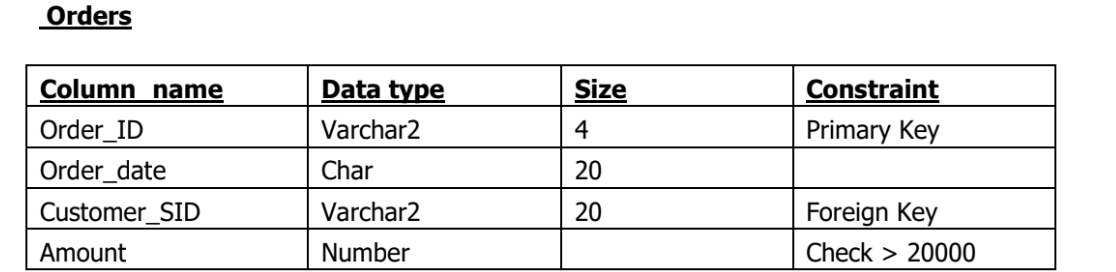
** SCHOOL OF COMPUTER APPLICATIONS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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:**

****

****

**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),

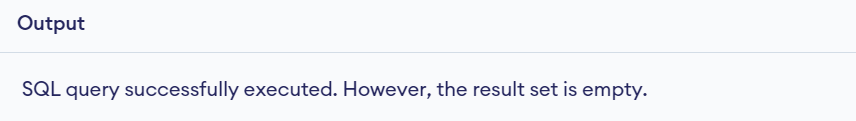
Customer\_SID Varchar2(20),

Amount Number,

CONSTRAINT fk\_customer FOREIGN KEY (Customer\_SID) REFERENCES Customer(SID),

CONSTRAINT chk\_amount CHECK (Amount > 2000)

**Output**

****

**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),

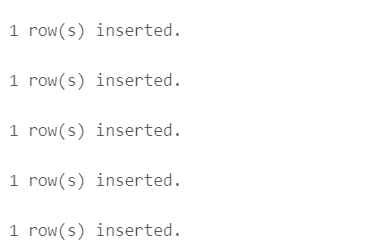
INSERT INTO Orders (Order\_Id, Order\_date, Customer\_SID, Amount)

VALUES ('O4', '4 Sep', 'C4', 6000),

INSERT INTO Orders (Order\_Id, Order\_date, Customer\_SID, Amount)

VALUES ('O5', '5 Sep', 'C5', 3500);

**Output**



**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.**

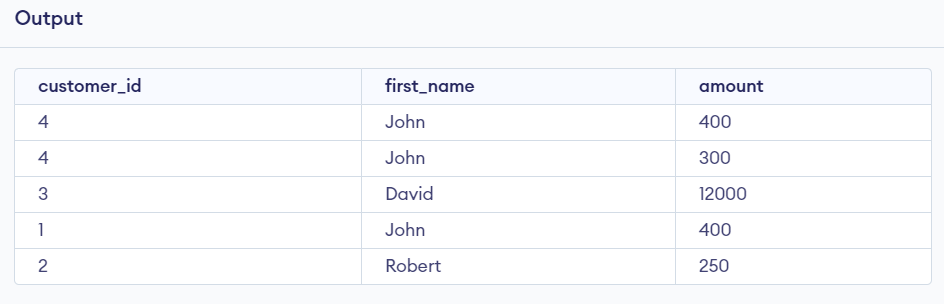
**Input**

SELECT Customers.customer\_id, Customers.first\_name, Orders.amount

FROM Customers

JOIN Orders ON Customers.customer\_id = Orders.customer\_id;

**Output**



**Experiment No: 6**

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

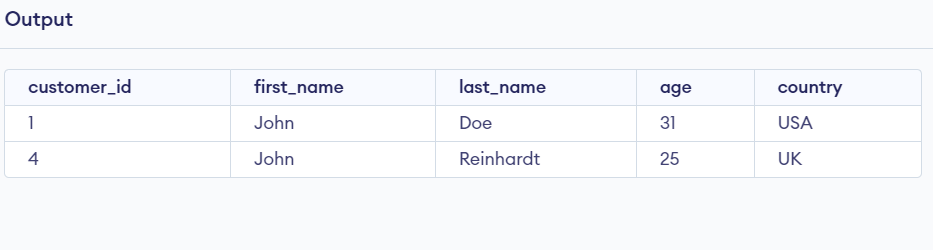
**Input**

SELECT \*

from Customers

where first\_name like '%n';

**Output**



**Experiment No: 7**

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

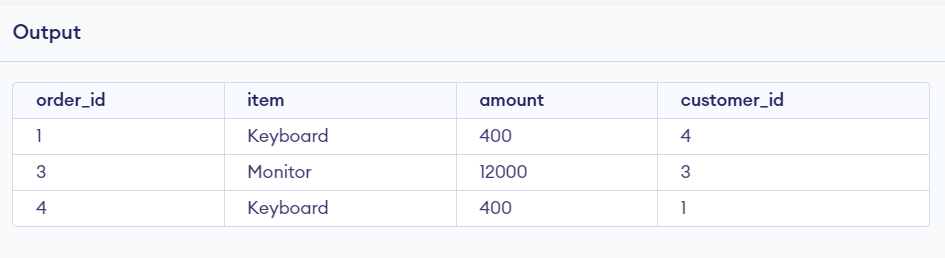
**Input**

SELECT \*

from Orders

where amount between 400 and 12000;

**Output**



**Experiment No: 8**

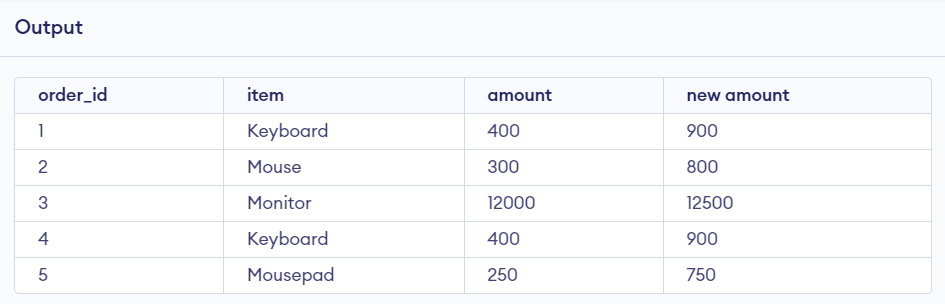
**Experiment 8:- List the orders where amount is increased by 500 and replace with name “new amount”.**

**Input**

SELECT order\_id, item,amount, amount + 500 AS "new amount"

FROM Orders;

**Output**



**Experiment No: 9**

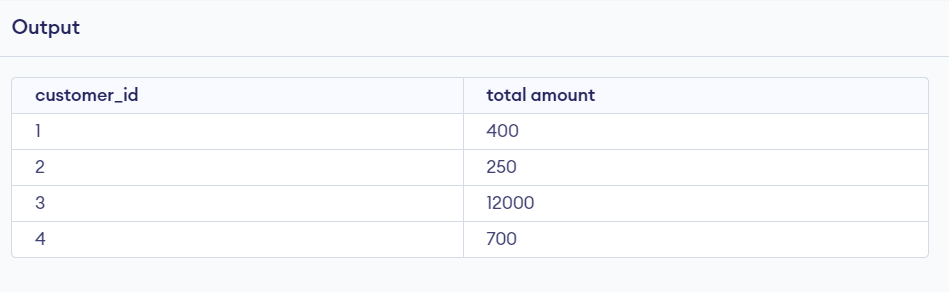
**Experiment 8:- Display the order\_id and total amount of orders.**

**Input**

SELECT customer\_id, sum(amount) AS "total amount"

FROM Orders

group by customer\_id

****

**Experiment No: 10**

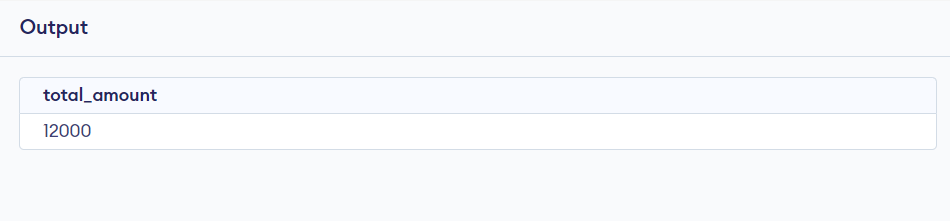
**Experiment 10:- Calculate the total amount of orders that has more than 15000.**

**Input**

SELECT sum(amount) as total\_amount

FROM Orders

where amount>400;



**Experiment No: 11**

**11: Display all the string functions used in SQL**.

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',

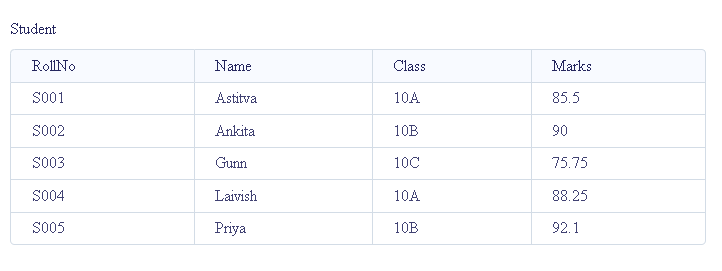
'10C', 79.40);

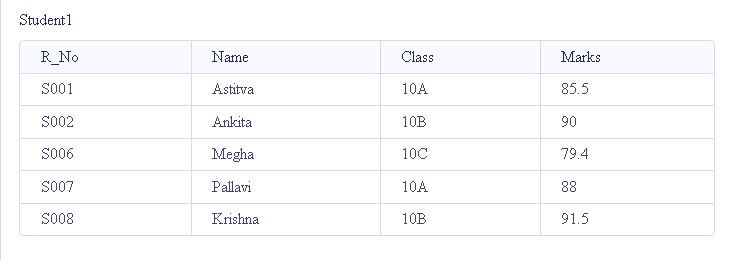
INSERT INTO Student1 (R\_No, Name, Class, Marks) VALUES ('S007', 'Pallavi',

'10A', 88.00);

INSERT INTO Student1 (R\_No, Name, Class, Marks) VALUES ('S008', 'Krishna',

'10B', 91.50)





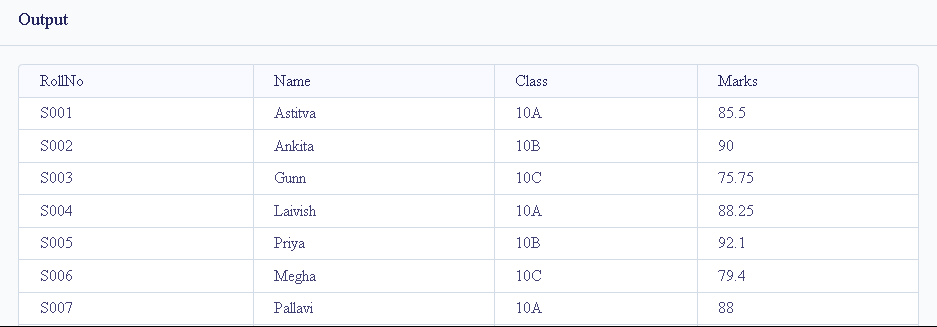
**Experiment No: 13**

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

SELECT \* FROM Student

UNION

SELECT \* FROM Student1;



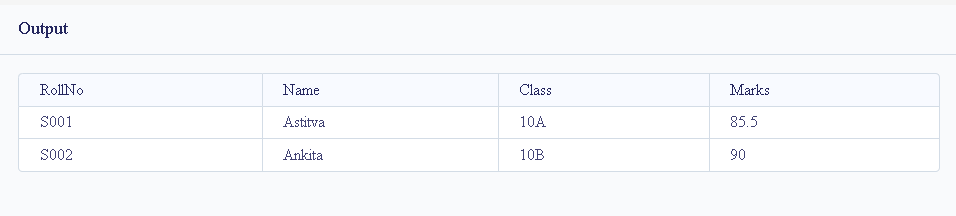
**Experiment No: 14**

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

SELECT \* FROM Student

INTERSECT

SELECT \* FROM Student1;



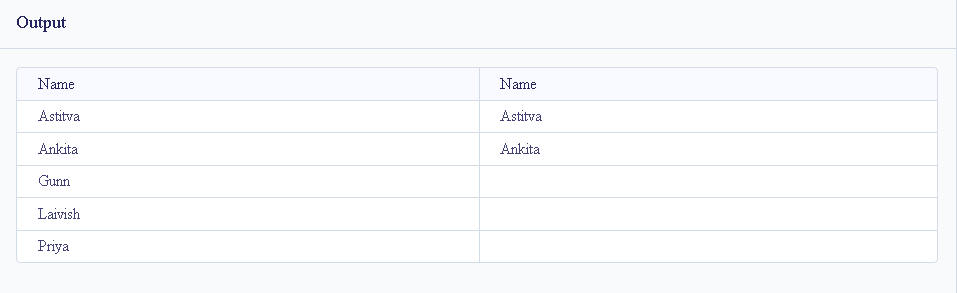
**Experiment No: 15**

**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;

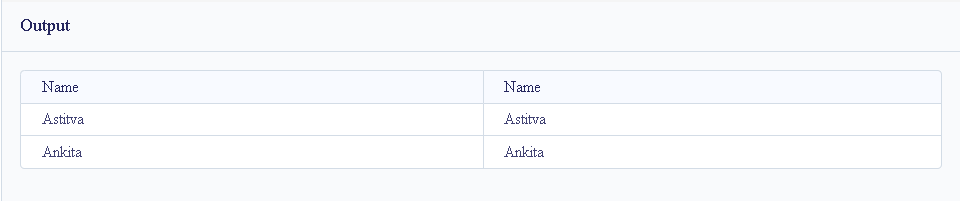


RIGHT and FULL OUTER JOINs are not currently supported.

SELECT Student.Name, Student1.Name

FROM Student

INNER JOIN Student1 ON Student.RollNo = Student1.R\_No;



**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);

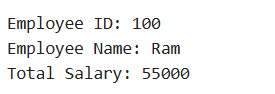
DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || emp\_name);

DBMS\_OUTPUT.PUT\_LINE('Total Salary: ' || total\_salary);

END;

/

**Output**

****

**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**



**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);

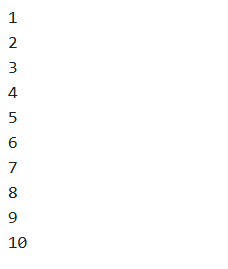
i := i + 1; -- Increment the counter

END LOOP;

END;

/

**Output**



**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

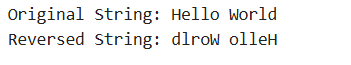
DBMS\_OUTPUT.PUT\_LINE('Original String: ' || original\_string);

DBMS\_OUTPUT.PUT\_LINE('Reversed String: ' || reversed\_string);

END;

/

**Output**



**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**

****

**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

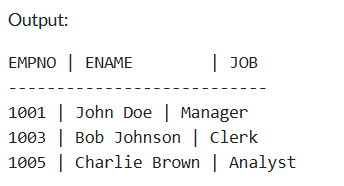
DBMS\_OUTPUT.PUT\_LINE(employees(i).empno || ' | ' || employees(i).ename || ' | ' || employees(i).job);

END IF;

END LOOP;

END; /

**Output**



**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 || ' | ' ||

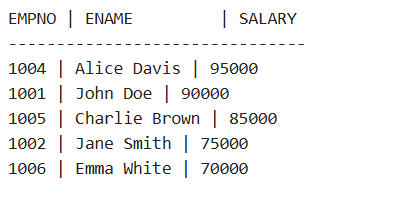
sorted\_employees(i).ename || ' | ' ||

sorted\_employees(i).salary);

END LOOP;

END;

**Output**

****

**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

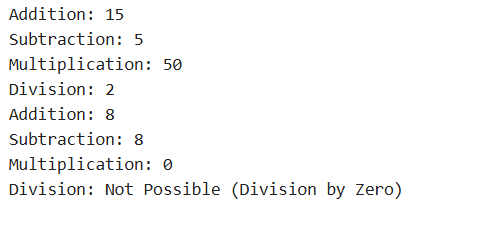
calculate\_operations(10, 5); -- Example with valid inputs

calculate\_operations(8, 0); -- Example to test division by zero

END;

/

**Output**



**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

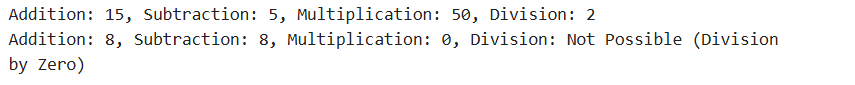
DBMS\_OUTPUT.PUT\_LINE(result);

result := perform\_operations(8, 0); -- Call with division by zero

DBMS\_OUTPUT.PUT\_LINE(result);

END; /

**Output**



**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**



**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**



**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**



**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;

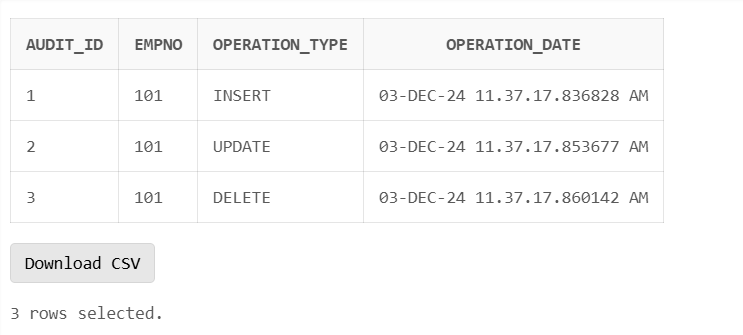
-- Delete an employee

DELETE FROM emp WHERE empno = 101;

-- Check the auditor table for logged operations

SELECT \* FROM auditor;

**Output**



**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');

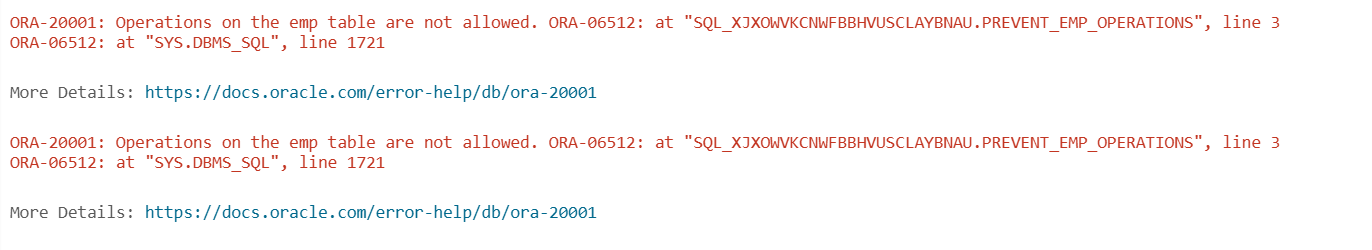
-- Trying to update an employee (this will fail)

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

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

DELETE FROM emp WHERE empno = 101;

**Output**



**Conclusion:**

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