1 Demonstrate the use of group by and order by clause.

→

```
-- Step 1: Create the sales table
CREATE TABLE sales (
   sale_id      NUMBER PRIMARY KEY,
   product_name  VARCHAR2(50),
   category     VARCHAR2(50),
   amount       NUMBER,
   sale_date    DATE
);
-- Step 2: Insert records into the sales table
INSERT INTO sales VALUES (1, 'Sofa', 'Furniture', 500, TO_DATE('2024-03-01', 'YYYY-MM-DD'));
INSERT INTO sales VALUES (2, 'Chair', 'Furniture', 200, TO_DATE('2024-03-02', 'YYYY-MM-DD'));
INSERT INTO sales VALUES (3, 'Table', 'Furniture', 300, TO_DATE('2024-03-03', 'YYYY-MM-DD'));
INSERT INTO sales VALUES (4, 'TV', 'Electronics', 700, TO_DATE('2024-03-04', 'YYYY-MM-DD'));
INSERT INTO sales VALUES (5, 'Laptop', 'Electronics', 1200, TO_DATE('2024-03-05', 'YYYY-MM-DD'));
INSERT INTO sales VALUES (6, 'Phone', 'Electronics', 800, TO_DATE('2024-03-06', 'YYYY-MM-DD'));
-- Step 4: Verify inserted data
SELECT * FROM sales;
SELECT category, SUM(amount) AS total_sales
FROM sales
GROUP BY category
```

ORDER BY total_sales DESC;

OUTPUT:

Results Explain Describe Saved SQL History

CATEGORY	TOTAL_SALES
Electronics	2700
Furniture	1000

2 rows returned in 0.00 seconds CSV Export

2.Consider the following schema for a Hospital Database: DOCTOR (Did, Dname, DAddress, Qualification) PATIENTMASTER (Pcode, Pname, Padd, age, gender, bloodgroup, Pid) ADMMITTEDPATIENT(Pcode, EntryDate, DischargeDate, WardNo, Disease)

a) Find the detail of the doctor who is treating the patient of ward no3

b) Find the detail of patient who are admitted within period 03/03/2020 to 25/05/2020

c) Find the detail of patient who are suffered from blood cancer

d) Create view on DOCTOR & PATIENTASTER tables.

→

-- a) Find the detail of the doctor treating patients in Ward No 3

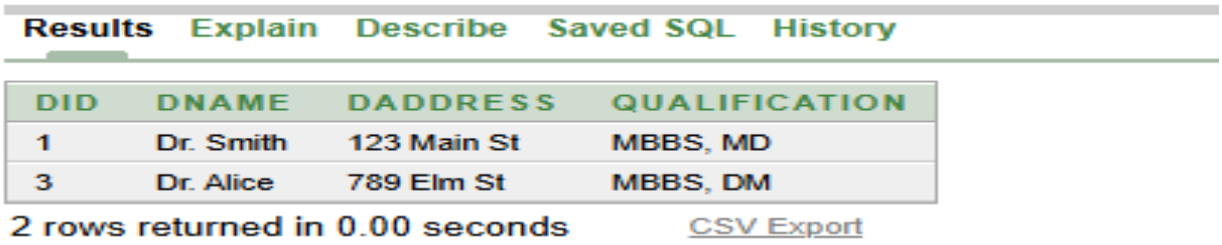
SELECT DOCTOR.Did, DOCTOR.Dname, DOCTOR.DAddress, DOCTOR.Qualification

FROM DOCTOR

JOIN ADMMITTEDPATIENT ON DOCTOR.Did = ADMMITTEDPATIENT.Did

WHERE ADMMITTEDPATIENT.WardNo = 3;

OUTPUT:

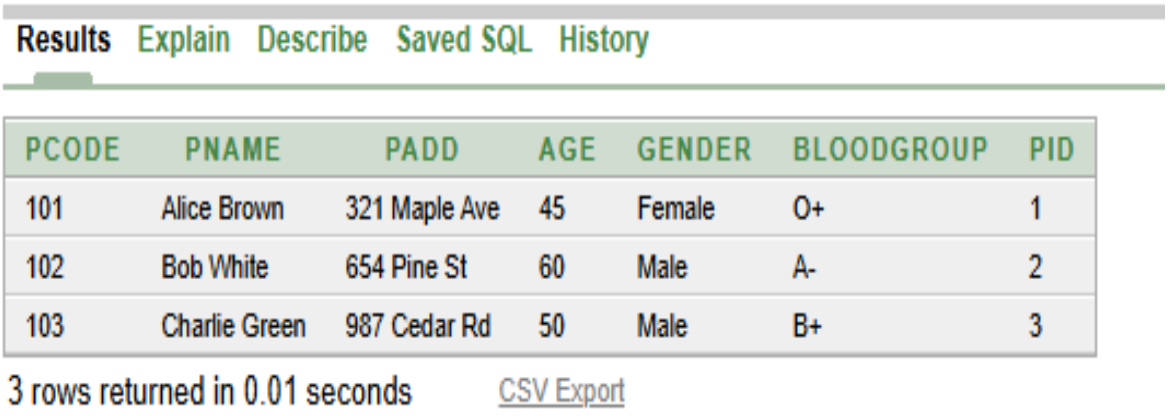


-- b) Find the details of patients admitted between '2020-03-03' and '2020-05-25'

SELECT P.Pcode, P.Pname, P.Padd, P.age, P.gender, P.bloodgroup, P.Pid

```
FROM PATIENTMASTER P
JOIN ADMMITTEDPATIENT AP ON P.Pcode = AP.Pcode
WHERE AP.EntryDate BETWEEN TO_DATE('2020-03-03', 'YYYY-MM-DD') AND
TO_DATE('2020-05-25', 'YYYY-MM-DD');
```

OUTPUT:



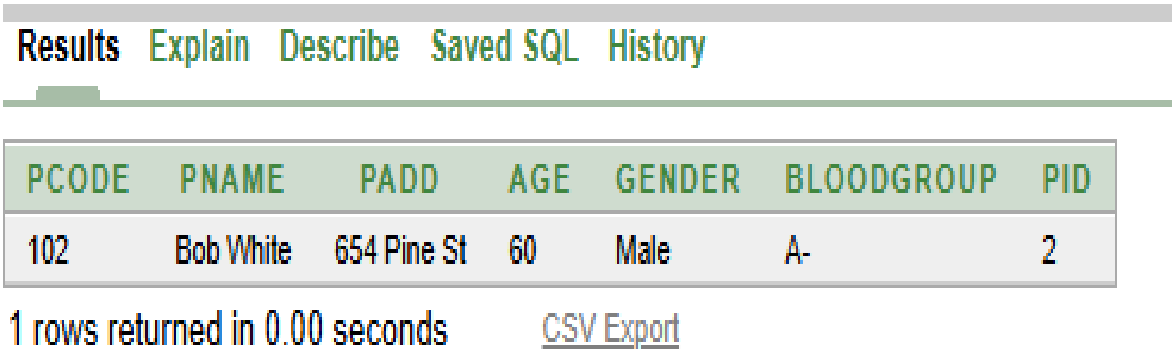
Results Explain Describe Saved SQL History

PCODE	PNAME	PADD	AGE	GENDER	BLOODGROUP	PID
101	Alice Brown	321 Maple Ave	45	Female	O+	1
102	Bob White	654 Pine St	60	Male	A-	2
103	Charlie Green	987 Cedar Rd	50	Male	B+	3

3 rows returned in 0.01 seconds CSV Export

```
-- c) Find the details of patients diagnosed with 'Blood Cancer'
SELECT P.Pcode, P.Pname, P.Padd, P.age, P.gender, P.bloodgroup, P.Pid
FROM PATIENTMASTER P
JOIN ADMMITTEDPATIENT AP ON P.Pcode = AP.Pcode
WHERE AP.Disease = 'Blood Cancer';
```

OUTPUT:



Results Explain Describe Saved SQL History

PCODE	PNAME	PADD	AGE	GENDER	BLOODGROUP	PID
102	Bob White	654 Pine St	60	Male	A-	2

1 rows returned in 0.00 seconds CSV Export

-- d) Retrieve data from DoctorPatientView

SELECT * FROM DoctorPatientView;

OUTPUT:

Results Explain Describe Saved SQL History

DID	DNAME	DADDRESS	QUALIFICATION	PCODE
1	Dr. Smith	123 Main St	MBBS, MD	101
2	Dr. John	456 Oak St	MBBS, MS	102
3	Dr. Alice	789 Elm St	MBBS, DM	103

3 rows returned in 0.00 seconds CSV Export

PNAME	PADD	AGE	GENDER	BLOODGROUP	PID
Alice Brown	321 Maple Ave	45	Female	O+	1
Bob White	654 Pine St	60	Male	A-	2
Charlie Green	987 Cedar Rd	50	Male	B+	3

3.

Create department table with the following structure

Field Name	Data Type
Deptno	Int
DeptName	Varchar(30)
Location	Varchar(30)

a) Add column designation to the department table.
b) Insert values into the table.
c) List the records of dept table grouped by deptno.
d) Update the record where deptno is 9.
e) Delete any column data from the table.

-- STEP 1: CREATE THE DEPARTMENT TABLE

```
CREATE TABLE DEPARTMENT (
   DeptNo INT PRIMARY KEY,  -- Department Number (Primary Key)
   DeptName VARCHAR(30),    -- Department Name
   Location VARCHAR(30)     -- Location of the Department
);
```

OUTPUT:


```
Table created.
```

0.04 seconds

-- STEP 2: ADD A NEW COLUMN "DESIGNATION" TO THE TABLE

```
ALTER TABLE DEPARTMENT ADD Designation VARCHAR(50);
```

OUTPUT:

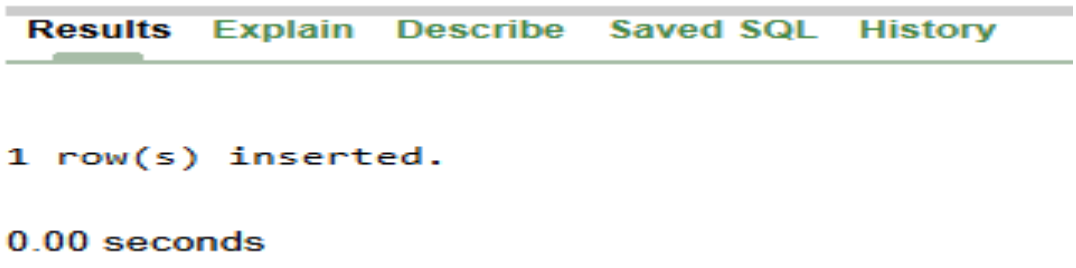

```
Table altered.
```

0.08 seconds

-- STEP 3: INSERT VALUES INTO THE DEPARTMENT TABLE

```
INSERT INTO DEPARTMENT (DeptNo, DeptName, Location, Designation)
VALUES (1, 'HR', 'New York', 'Manager');
INSERT INTO DEPARTMENT (DeptNo, DeptName, Location, Designation)
VALUES (2, 'Finance', 'London', 'Accountant');
INSERT INTO DEPARTMENT (DeptNo, DeptName, Location, Designation)
VALUES (3, 'IT', 'San Francisco', 'Developer');
INSERT INTO DEPARTMENT (DeptNo, DeptName, Location, Designation)
VALUES (4, 'Sales', 'Los Angeles', 'Sales Executive');
VALUES (5, 'Marketing', 'Chicago', 'Marketing Manager');
INSERT INTO DEPARTMENT (DeptNo, DeptName, Location, Designation)
VALUES (6, 'Operations', 'Houston', 'Operations Head');
INSERT INTO DEPARTMENT (DeptNo, DeptName, Location, Designation)
VALUES (7, 'Customer Service', 'Miami', 'Customer Support Lead');
INSERT INTO DEPARTMENT (DeptNo, DeptName, Location, Designation)
VALUES (8, 'R&D', 'Seattle', 'Research Scientist');
INSERT INTO DEPARTMENT (DeptNo, DeptName, Location, Designation)
VALUES (9, 'Administration', 'Washington DC', 'Admin Head');
```

OUTPUT:



-- STEP 4: LIST RECORDS GROUPED BY DEPTNO

```
SELECT DeptNo, DeptName, Location, Designation
```

FROM DEPARTMENT

GROUP BY DeptNo, DeptName, Location, Designation;

OUTPUT:

Results Explain Describe Saved SQL History

DEPTNO	DEPTNAME	LOCATION	DESIGNATION
2	Finance	London	Accountant
3	IT	San Francisco	Developer
8	R&D	Seattle	Research Scientist
5	Marketing	Chicago	Marketing Manager
6	Operations	Houston	Operations Head
1	HR	New York	Manager
4	Sales	Los Angeles	Sales Executive
7	Customer Service	Miami	Customer Support Lead
9	Administration	Washington DC	Admin Head

9 rows returned in 0.00 seconds CSV Export

-- STEP 5: UPDATE RECORD WHERE DEPTNO IS 9

UPDATE DEPARTMENT

SET Location = 'Chicago', Designation = 'Senior Manager'

WHERE DeptNo = 9;

OUTPUT:

Results Explain Describe Saved SQL History

```
1 row(s) updated.

0.00 seconds
```

-- STEP 6: DELETE DATA FROM A SPECIFIC COLUMN (SET TO NULL)

UPDATE DEPARTMENT

SET Designation = NULL

WHERE DeptNo = 2;

OUTPUT:


```
1 row(s) updated.

0.02 seconds
```

-- STEP 7: DROP THE COLUMN "DESIGNATION" FROM THE TABLE

ALTER TABLE DEPARTMENT DROP COLUMN Designation;

OUTPUT:

4.Create database using following schema. Apply Integrity Constraints and answer the following queries using SQL. DOCTOR (Did, Dname, Daddress, qualification) PATIENT (Pid, Pname, age, gender) Integrity Constraints:

1. The values of any attributes should not be null.

2. Did should be unique constraints.

3. Pid should be unique constraints.

4. Gender value should be M (male) or F (female).

Queries: a) Insert at least 10 records in table.

b) Find details of all doctors.

c) Delete the records from DOCTOR where qualification is M.S

d) Find details of patient where age is less than 40.

e) Update the patient name where patient id is 5.

→

```
--Use it when table Already exist
DROP TABLE DOCTOR CASCADE CONSTRAINTS;
DROP TABLE PATIENT;


--Step 1: Create Tables with Integrity Constraints
-- Creating DOCTOR Table with Constraints
CREATE TABLE DOCTOR (
    Did INT PRIMARY KEY,
    Dname VARCHAR(50) NOT NULL,
    Daddress VARCHAR(100) NOT NULL,
    qualification VARCHAR(50) NOT NULL
);
CREATE TABLE PATIENT (
    Pid INT PRIMARY KEY,
```

```
    Pname VARCHAR(50) NOT NULL,
    age INT NOT NULL,
    gender CHAR(1) CHECK (gender IN ('M', 'F')) NOT NULL
);
--Step 2: Insert at Least 10 Records
--Inserting into DOCTOR Table
INSERT INTO DOCTOR (Did, Dname, Daddress, qualification) VALUES (1, 'Dr. Smith', 'New York', 'MBBS');
INSERT INTO DOCTOR (Did, Dname, Daddress, qualification) VALUES (2, 'Dr. Johnson', 'Los Angeles', 'MD');
INSERT INTO DOCTOR (Did, Dname, Daddress, qualification) VALUES (3, 'Dr. Brown', 'Chicago', 'M.S');
INSERT INTO DOCTOR (Did, Dname, Daddress, qualification) VALUES (4, 'Dr. Williams', 'Houston', 'MBBS');
INSERT INTO DOCTOR (Did, Dname, Daddress, qualification) VALUES (5, 'Dr. Taylor', 'San Francisco', 'MD');
INSERT INTO DOCTOR (Did, Dname, Daddress, qualification) VALUES (6, 'Dr. Anderson', 'Seattle', 'M.S');
INSERT INTO DOCTOR (Did, Dname, Daddress, qualification) VALUES (7, 'Dr. Thomas', 'Boston', 'MBBS');
INSERT INTO DOCTOR (Did, Dname, Daddress, qualification) VALUES (8, 'Dr. Jackson', 'Miami', 'MD');
INSERT INTO DOCTOR (Did, Dname, Daddress, qualification) VALUES (9, 'Dr. White', 'Denver', 'M.S');
INSERT INTO DOCTOR (Did, Dname, Daddress, qualification) VALUES (10, 'Dr. Harris', 'Atlanta', 'MBBS');
--Inserting into PATIENT Table
INSERT INTO PATIENT (Pid, Pname, age, gender) VALUES (1, 'John Doe', 35, 'M');
```

INSERT INTO PATIENT (Pid, Pname, age, gender) VALUES (2, 'Jane Smith', 28, 'F');

INSERT INTO PATIENT (Pid, Pname, age, gender) VALUES (3, 'Michael Brown', 42, 'M');

INSERT INTO PATIENT (Pid, Pname, age, gender) VALUES (4, 'Emily Davis', 22, 'F');

INSERT INTO PATIENT (Pid, Pname, age, gender) VALUES (5, 'David Wilson', 30, 'M');

INSERT INTO PATIENT (Pid, Pname, age, gender) VALUES (6, 'Sarah Johnson', 50, 'F');

INSERT INTO PATIENT (Pid, Pname, age, gender) VALUES (7, 'James Taylor', 18, 'M');

INSERT INTO PATIENT (Pid, Pname, age, gender) VALUES (8, 'Anna White', 38, 'F');

INSERT INTO PATIENT (Pid, Pname, age, gender) VALUES (9, 'Robert Anderson', 60, 'M');

INSERT INTO PATIENT (Pid, Pname, age, gender) VALUES (10, 'Olivia Martinez', 25, 'F');

--Step 3: Queries

--b) Find details of all doctors

SELECT * FROM DOCTOR;

OUTPUT:

Results Explain Describe Saved SQL History

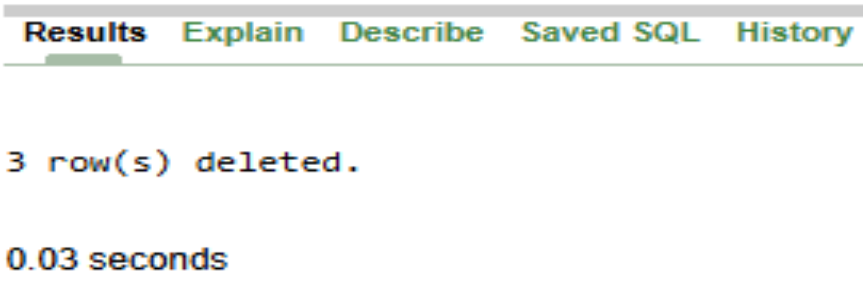
DID	DNAME	DADDRESS	QUALIFICATION
1	Dr. Smith	New York	MBBS
2	Dr. Johnson	Los Angeles	MD
3	Dr. Brown	Chicago	M.S
4	Dr. Williams	Houston	MBBS
5	Dr. Taylor	San Francisco	MD
6	Dr. Anderson	Seattle	M.S
7	Dr. Thomas	Boston	MBBS
8	Dr. Jackson	Miami	MD
9	Dr. White	Denver	M.S
10	Dr. Harris	Atlanta	MBBS

10 rows returned in 0.01 seconds CSV Export

--c) Delete records from DOCTOR where qualification is M.S

DELETE FROM DOCTOR WHERE qualification = 'M.S';

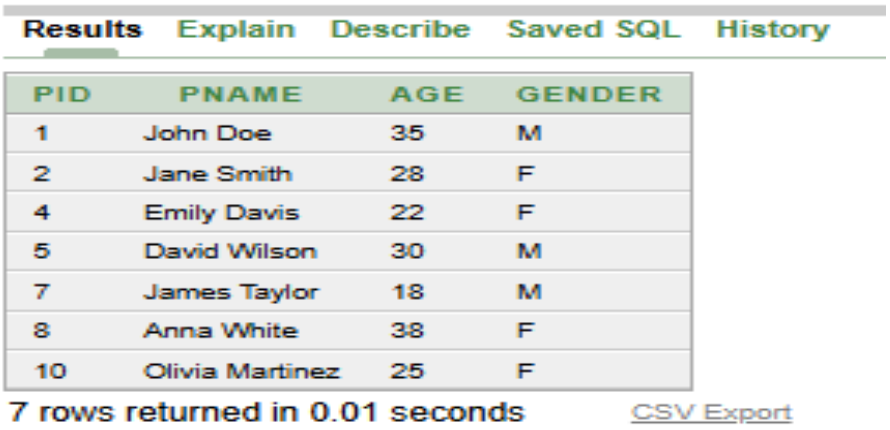
OUTPUT:



d) Find details of patients where age is less than 40

SELECT * FROM PATIENT WHERE age < 40;

OUTPUT:



--e) Update the patient name where patient ID is 5

UPDATE PATIENT

SET Pname = 'David Johnson'

WHERE Pid = 5;

OUTPUT:

Results Explain Describe Saved SQL History

```
1 row(s) updated.
```

0.01 seconds

5.Write a PL/SQL code to create an employee database with the tables and fields specified as below. Employee[Emp no Employee name Street City] Works[Emp no Company_name Joining_date Designation Salary] Company[Emp no City] Manages[Emp no Manager_name,Mang_no]

```
-- 1 CREATE EMPLOYEE TABLE
CREATE TABLE Employee (
   emp_no NUMBER PRIMARY KEY,              -- Unique employee number
   emp_name VARCHAR2(100) NOT NULL,        -- Employee name (Not NULL)
   street VARCHAR2(100) NOT NULL,          -- Street address (Not NULL)
   city VARCHAR2(50) NOT NULL              -- City (Not NULL)
);
```

OUTPUT:


```
-- 2 CREATE WORKS TABLE (Stores Employee's Job Information)
CREATE TABLE Works (
   emp_no NUMBER,                          -- Employee number (Foreign Key)
   company_name VARCHAR2(100) NOT NULL,    -- Company name (Not NULL)
   joining_date DATE NOT NULL,             -- Joining date (Not NULL)
   designation VARCHAR2(50) NOT NULL,      -- Designation (Not NULL)
   salary NUMBER(10, 2) NOT NULL,          -- Salary (Not NULL)
   PRIMARY KEY (emp_no, company_name),     -- Composite Primary Key
   FOREIGN KEY (emp_no) REFERENCES Employee(emp_no)
```

);

OUTPUT:


```
Table created.

0.01 seconds
```

```
-- 3 CREATE COMPANY TABLE (Stores Company's City Information)
CREATE TABLE Company (
   emp_no NUMBER,                    -- Employee number (Foreign Key)
   city VARCHAR2(50) NOT NULL,       -- City (Not NULL)
   PRIMARY KEY (emp_no),
   FOREIGN KEY (emp_no) REFERENCES Employee(emp_no)
);
```

OUTPUT:


```
Table created.

0.02 seconds
```

```
-- 4 CREATE MANAGES TABLE (Stores Manager Details)
CREATE TABLE Manages (
   emp_no NUMBER,                    -- Employee number (Foreign Key)
   manager_name VARCHAR2(100) NOT NULL,   -- Manager name (Not NULL)
   mang_no NUMBER NOT NULL,          -- Manager number (Not NULL)
   PRIMARY KEY (emp_no),
```

FOREIGN KEY (emp_no) REFERENCES Employee(emp_no)

);

OUTPUT:


```
Table created.

0.02 seconds
```

-- 5 INSERT RECORDS INTO EMPLOYEE TABLE

INSERT INTO Employee (emp_no, emp_name, street, city)

VALUES (1, 'John Doe', '123 Elm St', 'New York');

INSERT INTO Employee (emp_no, emp_name, street, city)

VALUES (2, 'Jane Smith', '456 Oak St', 'Chicago');

OUTPUT:


```
1 row(s) inserted.

0.02 seconds
```

-- 6 INSERT RECORDS INTO WORKS TABLE

INSERT INTO Works (emp_no, company_name, joining_date, designation, salary) VALUES (1, 'ABC Corp', TO_DATE('2023-01-01', 'YYYY-MM-DD'), 'Software Engineer', 85000);

INSERT INTO Works (emp_no, company_name, joining_date, designation, salary) VALUES (2, 'XYZ Ltd', TO_DATE('2022-06-15', 'YYYY-MM-DD'), 'Data Analyst', 75000);

OUTPUT:

1 row(s) inserted.

0.00 seconds

-- 7 INSERT RECORDS INTO COMPANY TABLE

INSERT INTO Company (emp_no, city) VALUES (1, 'New York');

INSERT INTO Company (emp_no, city) VALUES (2, 'Chicago');

OUTPUT:

1 row(s) inserted.

0.01 seconds

-- 8 INSERT RECORDS INTO MANAGES TABLE

INSERT INTO Manages (emp_no, manager_name, mang_no) VALUES (1, 'Sarah Johnson', 101);

INSERT INTO Manages (emp_no, manager_name, mang_no) VALUES (2, 'Michael Brown', 102);

1 row(s) inserted.

0.00 seconds

OUTPUT:

6. PL/SQL code to retrieve the employee name, join date, and designation from employee database of an employee whose number is input by the user.

→

```
-- 1: Create the Employee Table
CREATE TABLE Employee (
   emp_no NUMBER PRIMARY KEY,
   emp_name VARCHAR2(100) NOT NULL,
   join_date DATE NOT NULL,
   designation VARCHAR2(50) NOT NULL
);
-- 2: Insert Sample Data
INSERT INTO Employee (emp_no, emp_name, join_date, designation) VALUES
(1, 'John Doe', TO_DATE('2022-01-10', 'YYYY-MM-DD'), 'Software Engineer');
INSERT INTO Employee (emp_no, emp_name, join_date, designation) VALUES
(2, 'Jane Smith', TO_DATE('2021-06-15', 'YYYY-MM-DD'), 'Data Analyst');
INSERT INTO Employee (emp_no, emp_name, join_date, designation) VALUES
(3, 'Mark Johnson', TO_DATE('2020-09-25', 'YYYY-MM-DD'), 'HR Manager');
COMMIT;
--3: Create the Stored Procedure
CREATE OR REPLACE PROCEDURE get_employee_details (
   emp_number IN NUMBER,
   emp_name OUT VARCHAR2,
   join_date OUT DATE,
   designation OUT VARCHAR2
)
IS
BEGIN
```

```
    -- Fetch employee details based on the given employee number
    SELECT emp_name, join_date, designation
    INTO emp_name, join_date, designation
    FROM Employee
    WHERE emp_no = emp_number;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        emp_name := 'Not Found';
        join_date := NULL;
        designation := 'Not Found';
    WHEN OTHERS THEN
        emp_name := 'Error';
        join_date := NULL;
        designation := 'Error';
END get_employee_details;
/
```

OUTPUT:


```
Procedure created.

0.03 seconds
```

```
--4: Run PL/SQL Block to Get Employee Details
DECLARE
    v_emp_no NUMBER;
```

```
    v_emp_name VARCHAR2(100);
    v_join_date DATE;
    v_designation VARCHAR2(50);
BEGIN
    -- Assign Employee Number (Change Manually or Use SQL*Plus for Input)
    v_emp_no := 1;  -- Change this number to test different employees
    -- Call the Procedure
    get_employee_details(v_emp_no, v_emp_name, v_join_date,
v_designation);
    -- Display Output
    DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_emp_name);
    DBMS_OUTPUT.PUT_LINE('Join Date: ' || TO_CHAR(v_join_date, 'YYYY-MM-
DD'));
    DBMS_OUTPUT.PUT_LINE('Designation: ' || v_designation);
END;
/
```

OUTPUT:


```
Employee Name: John Doe
Join Date: 2022-01-10
Designation: Software Engineer

Statement processed.


0.01 seconds
```

7 write a PL\SQL code to update the salary of employees who earn less than the average salary using cursor.

→

```
-- Drop the table if it already exists
DROP TABLE employees PURGE;


-- Create the employees table
CREATE TABLE employees (
   EMPLOYEE_ID NUMBER PRIMARY KEY,
   EMPLOYEE_NAME VARCHAR2(100),
   SALARY NUMBER(10,2),
   DEPARTMENT VARCHAR2(50)
);
-- Insert sample employee records
INSERT INTO employees (EMPLOYEE_ID, EMPLOYEE_NAME, SALARY, DEPARTMENT) VALUES (101, 'Alice', 4000, 'IT');
INSERT INTO employees (EMPLOYEE_ID, EMPLOYEE_NAME, SALARY, DEPARTMENT) VALUES (102, 'Bob', 3500, 'HR');
INSERT INTO employees (EMPLOYEE_ID, EMPLOYEE_NAME, SALARY, DEPARTMENT) VALUES (103, 'Charlie', 3000, 'Finance');
INSERT INTO employees (EMPLOYEE_ID, EMPLOYEE_NAME, SALARY, DEPARTMENT) VALUES (104, 'David', 4500, 'Marketing');
INSERT INTO employees (EMPLOYEE_ID, EMPLOYEE_NAME, SALARY, DEPARTMENT) VALUES (105, 'Emma', 5000, 'IT');
INSERT INTO employees (EMPLOYEE_ID, EMPLOYEE_NAME, SALARY, DEPARTMENT) VALUES (106, 'Frank', 2800, 'HR');
-- Commit the changes
COMMIT;
```

```
DECLARE
   -- Cursor to select employees earning below the average salary
   CURSOR emp_cursor IS
      SELECT EMPLOYEE_ID, SALARY
      FROM employees
      WHERE SALARY < (SELECT AVG(SALARY) FROM employees);
   -- Variables to store fetched employee details
   v_emp_id employees.EMPLOYEE_ID%TYPE;
   v_salary employees.SALARY%TYPE;
   v_increment NUMBER := 500;  -- Fixed salary increment
BEGIN
   -- Open the cursor
   OPEN emp_cursor;
   -- Process each employee
   LOOP
      FETCH emp_cursor INTO v_emp_id, v_salary;
      EXIT WHEN emp_cursor%NOTFOUND;  -- Exit loop when all records are
processed
      -- Update salary
      UPDATE employees
      SET SALARY = SALARY + v_increment
      WHERE EMPLOYEE_ID = v_emp_id;
      -- Display updated salary details
      DBMS_OUTPUT.PUT_LINE('Updated Employee ID: ' || v_emp_id || ' |
New Salary: ' || (v_salary + v_increment));
   END LOOP;
   -- Commit the changes
```

```
    COMMIT;
    -- Close the cursor
    CLOSE emp_cursor;
    -- Success message
    DBMS_OUTPUT.PUT_LINE('Salaries updated successfully.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
        ROLLBACK;  -- Rollback changes if any error occurs
END;
/
```

OUTPUT:

Results Explain Describe Saved SQL History

```
Updated Employee ID: 102 | New Salary: 4000
Updated Employee ID: 103 | New Salary: 3500
Updated Employee ID: 106 | New Salary: 3300
Salaries updated successfully.

1 row(s) updated.
```

0.08 seconds

SELECT * FROM employees ORDER BY SALARY;

OUTPUT:

Results Explain Describe Saved SQL History

EMPLOYEE_ID	EMPLOYEE_NAME	SALARY	DEPARTMENT
106	Frank	3300	HR
103	Charlie	3500	Finance
101	Alice	4000	IT
102	Bob	4000	HR
104	David	4500	Marketing
105	Emma	5000	IT

6 rows returned in 0.00 seconds CSV Export

10 Write a PL/SQL procedure to find the number of students ranging from 100-70%, 69-60%, 59-50% & below 49% in each course from the student_course table given by the procedure as parameter.

→

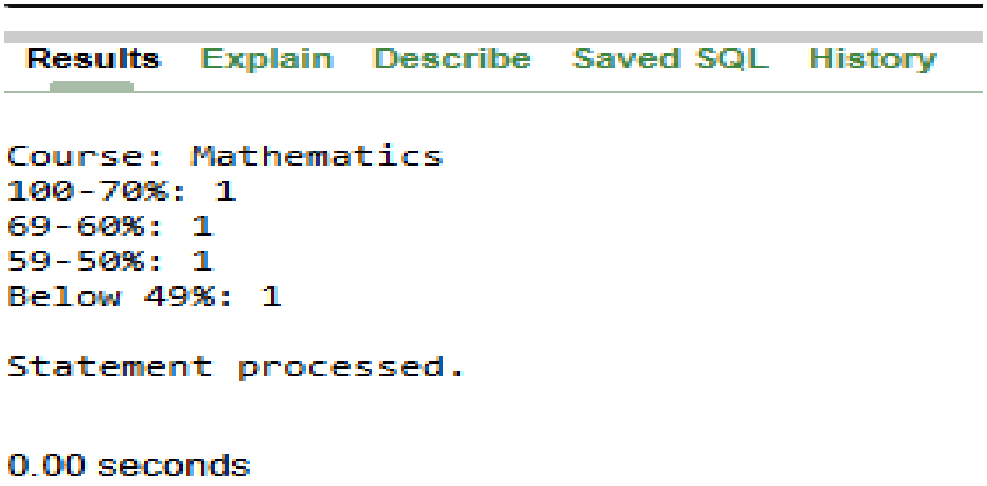
```
--Step 1: Create the Table
CREATE TABLE student_course (
   student_id NUMBER PRIMARY KEY,
   course_name VARCHAR2(100),
   percentage NUMBER(5,2)
);
--Step 2: Insert Sample Data
INSERT INTO student_course VALUES (1, 'Mathematics', 75);
INSERT INTO student_course VALUES (2, 'Mathematics', 65);
INSERT INTO student_course VALUES (3, 'Mathematics', 55);
INSERT INTO student_course VALUES (4, 'Mathematics', 45);
INSERT INTO student_course VALUES (5, 'Physics', 80);
INSERT INTO student_course VALUES (6, 'Physics', 70);
INSERT INTO student_course VALUES (7, 'Physics', 50);
INSERT INTO student_course VALUES (8, 'Physics', 40);
COMMIT;
--Step 3: Create the Procedure
CREATE OR REPLACE PROCEDURE student_grade_distribution (
   p_course_name IN VARCHAR2
)
IS
   v_100_70   NUMBER := 0;
   v_69_60    NUMBER := 0;
```

```
    v_59_50    NUMBER := 0;
    v_below_49 NUMBER := 0;
BEGIN
    -- Count students in each percentage range
    SELECT COUNT(*) INTO v_100_70 FROM student_course
    WHERE course_name = p_course_name AND percentage BETWEEN 70 AND
100;
    SELECT COUNT(*) INTO v_69_60 FROM student_course
    WHERE course_name = p_course_name AND percentage BETWEEN 60 AND
69;
    SELECT COUNT(*) INTO v_59_50 FROM student_course
    WHERE course_name = p_course_name AND percentage BETWEEN 50 AND
59;
    SELECT COUNT(*) INTO v_below_49 FROM student_course
    WHERE course_name = p_course_name AND percentage < 49;
    -- Display the results
    DBMS_OUTPUT.PUT_LINE('Course: ' || p_course_name);
    DBMS_OUTPUT.PUT_LINE('100-70%: ' || v_100_70);
    DBMS_OUTPUT.PUT_LINE('69-60%: ' || v_69_60);
    DBMS_OUTPUT.PUT_LINE('59-50%: ' || v_59_50);
    DBMS_OUTPUT.PUT_LINE('Below 49%: ' || v_below_49);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END student_grade_distribution;
/
```

Step 4: Execute the Procedure

```
BEGIN
   DBMS_OUTPUT.ENABLE;
END;
/
BEGIN
   student_grade_distribution('Mathematics');
END;
/
```

OUTPUT:

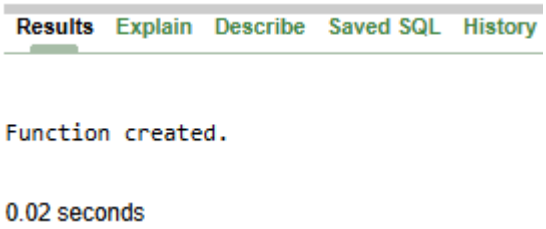


1 Create a store function that accepts 2 numbers and returns the addition of passed values. Also, write the code to call your function.

→

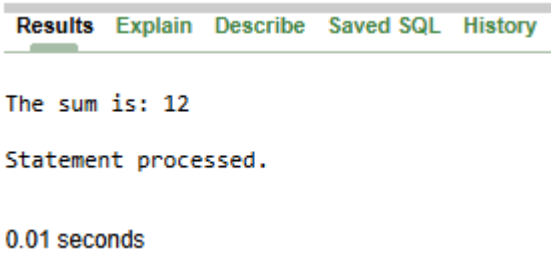
```
-- Create the function
CREATE OR REPLACE FUNCTION store(num1 IN NUMBER, num2 IN NUMBER)
RETURN NUMBER
IS
   sum_result NUMBER;
BEGIN
   sum_result := num1 + num2;
   RETURN sum_result;
END;
/
```

OUTPUT:




```
-- Calling the function
DECLARE
   result NUMBER;
BEGIN
   result := store(5, 7); -- Example values
   DBMS_OUTPUT.PUT_LINE('The sum is: ' || result);
END;
/
```

OUTPUT:


Results
Explain
Describe
Saved SQL
History
The sum is: 12
Statement processed.
0.01 seconds

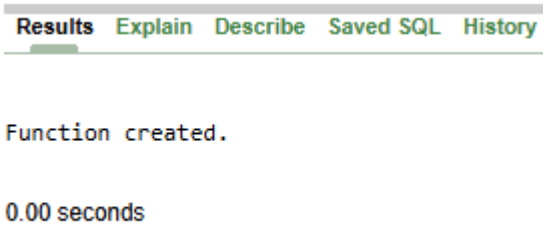
12 Write a PL/SQL function that accepts the department number and returns the total salary of the department. Also, write a function to call the function.

→

```
DROP TABLE employees PURGE;
-- 1. Create the employees table
CREATE TABLE employees (
   employee_id NUMBER PRIMARY KEY,
   employee_name VARCHAR2(100),
   salary NUMBER(10,2),
   department_id NUMBER
);
-- 2. Insert sample data into employees table
INSERT INTO employees VALUES (1, 'Alice', 5000, 10);
INSERT INTO employees VALUES (2, 'Bob', 7000, 10);
INSERT INTO employees VALUES (3, 'Charlie', 6000, 20);
INSERT INTO employees VALUES (4, 'David', 8000, 20);
INSERT INTO employees VALUES (5, 'Eve', 5500, 10);
-- Commit changes
COMMIT;
-- 3. Create the function to calculate total salary of a department
CREATE OR REPLACE FUNCTION get_total_salary(dept_id IN NUMBER)
RETURN NUMBER
IS
   total_salary NUMBER;
BEGIN
   -- Calculate total salary for the given department
   SELECT SUM(salary) INTO total_salary
```

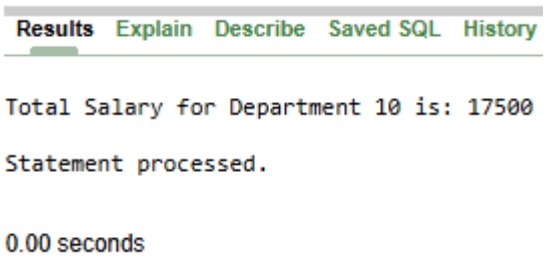
```
    FROM employees
    WHERE department_id = dept_id;
    RETURN NVL(total_salary, 0); -- Return 0 if no employees found
END;
/
```

OUTPUT:




```
-- 5. Call the function and display the result
DECLARE
    dept_salary NUMBER;
    dept_id NUMBER := 10; -- Example department ID
BEGIN
    dept_salary := get_total_salary(dept_id);
    DBMS_OUTPUT.PUT_LINE('Total Salary for Department ' || dept_id || ' is: '
|| dept_salary);
END;
/
```

OUTPUT:



13 Write a PL/SQL code to create,

1. Package specification
2. Package body

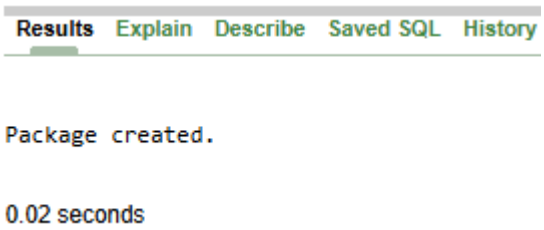
For the insert, retrieve, update, and delete operations on a student table.

→

1 Create the Package Specification

```
CREATE OR REPLACE PACKAGE student_pkg AS
   -- Procedure to insert a student
   PROCEDURE insert_student(p_id NUMBER, p_name VARCHAR2, p_age NUMBER, p_course VARCHAR2);
   -- Procedure to retrieve student details
   PROCEDURE get_student(p_id NUMBER);
   -- Procedure to update student details
   PROCEDURE update_student(p_id NUMBER, p_name VARCHAR2, p_age NUMBER, p_course VARCHAR2);
   -- Procedure to delete a student
   PROCEDURE delete_student(p_id NUMBER);
END student_pkg;
/
```

OUTPUT:



2 Create the Package Body

```
CREATE OR REPLACE PACKAGE BODY student_pkg AS
   -- Insert student details
```

```
  PROCEDURE insert_student(p_id NUMBER, p_name VARCHAR2, p_age
NUMBER, p_course VARCHAR2) IS
  BEGIN
    INSERT INTO student (student_id, student_name, age, course)
    VALUES (p_id, p_name, p_age, p_course);
    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Student inserted successfully.');
  END insert_student;
  -- Retrieve student details
  PROCEDURE get_student(p_id NUMBER) IS
    v_name student.student_name%TYPE;
    v_age student.age%TYPE;
    v_course student.course%TYPE;
  BEGIN
    SELECT student_name, age, course INTO v_name, v_age, v_course
    FROM student
    WHERE student_id = p_id;
    DBMS_OUTPUT.PUT_LINE('Student Name: ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Age: ' || v_age);
    DBMS_OUTPUT.PUT_LINE('Course: ' || v_course);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Student not found.');
  END get_student;
  -- Update student details
  PROCEDURE update_student(p_id NUMBER, p_name VARCHAR2, p_age
NUMBER, p_course VARCHAR2) IS
```

```
    BEGIN
        UPDATE student
        SET student_name = p_name, age = p_age, course = p_course
        WHERE student_id = p_id;
        IF SQL%ROWCOUNT = 0 THEN
            DBMS_OUTPUT.PUT_LINE('No student found with the given ID.');
        ELSE
            COMMIT;
            DBMS_OUTPUT.PUT_LINE('Student updated successfully.');
        END IF;
    END update_student;
    -- Delete a student record
    PROCEDURE delete_student(p_id NUMBER) IS
    BEGIN
        DELETE FROM student WHERE student_id = p_id;
        IF SQL%ROWCOUNT = 0 THEN
            DBMS_OUTPUT.PUT_LINE('No student found with the given ID.');
        ELSE
            COMMIT;
            DBMS_OUTPUT.PUT_LINE('Student deleted successfully.');
        END IF;
    END delete_student;
END student_pkg;
/
```

OUTPUT:


```
Package Body created.

0.01 seconds
```

--Test package

-- Insert a student

```
BEGIN
   student_pkg.insert_student(1, 'John Doe', 22, 'Computer Science');
END;
/
```

OUTPUT:


```
Student inserted successfully.

Statement processed.

0.02 seconds
```

-- Retrieve student details

```
BEGIN
   student_pkg.get_student(1);
END;
/
```

OUTPUT:


```
Student Name: John Doe
Age: 22
Course: Computer Science

Statement processed.

0.00 seconds
```

-- Update student details

```
BEGIN
   student_pkg.update_student(1, 'John Smith', 23, 'Information Technology');
END;
/
```

OUTPUT:


```
Student updated successfully.

Statement processed.

0.00 seconds
```

-- Delete a student

```
BEGIN
   student_pkg.delete_student(1);
END;
/
```

OUTPUT:


```
Student deleted successfully.

Statement processed.

0.00 seconds
```

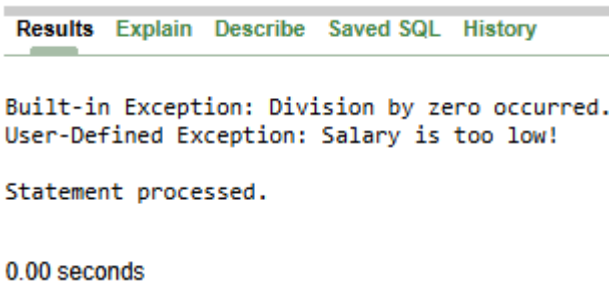
14 Write a program to illustrate user-defined exceptions, built-in exceptions, and raise application error exceptions.

→

```
--------------------------------------------------------------------------------
-- PL/SQL PROGRAM TO DEMONSTRATE ALL EXCEPTIONS TOGETHER
-- This program covers:
-- 1. User-defined exceptions
-- 2. Built-in exceptions
-- 3. Raise application error exceptions
--------------------------------------------------------------------------------
DECLARE
   -- User-Defined Exception
   ex_salary_too_low EXCEPTION;
   PRAGMA EXCEPTION_INIT(ex_salary_too_low, -20001);
   -- Variables
   v_salary NUMBER := 8000;  -- Change values to test different cases
   v_divide NUMBER;
BEGIN
   ----------------------------------------------------------------------------
   -- Built-in Exception: ZERO_DIVIDE (Triggers First)
   ----------------------------------------------------------------------------
   BEGIN
      v_divide := 10 / 0; -- Causes ORA-01476 (division by zero)
   EXCEPTION
      WHEN ZERO_DIVIDE THEN
         DBMS_OUTPUT.PUT_LINE('Built-in Exception: Division by zero
occurred.');
```

```
    END;
    ----------------------------------------------------------------------------
    -- User-Defined Exception: Salary Too Low (Triggers Second)
    ----------------------------------------------------------------------------
    IF v_salary < 10000 THEN
        RAISE ex_salary_too_low; -- Manually raising user-defined exception
    END IF;
    ----------------------------------------------------------------------------
    -- Raise_Application_Error: Custom Business Rule (Triggers Third)
    ----------------------------------------------------------------------------
    IF v_salary > 50000 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Salary cannot exceed 50,000.');
    END IF;
    DBMS_OUTPUT.PUT_LINE('Salary is within the valid range.');
EXCEPTION
    WHEN ex_salary_too_low THEN
        DBMS_OUTPUT.PUT_LINE('User-Defined Exception: Salary is too low!');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Some other error occurred: ' || SQLERRM);
END;
/
```

OUTPUT:

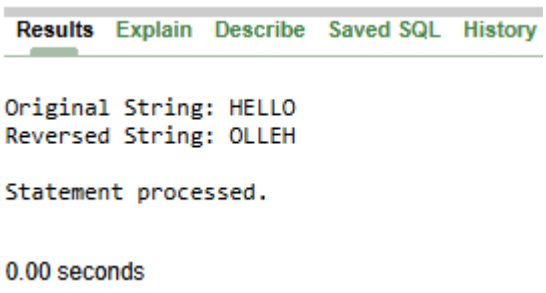


15 Write a program Reversing a String Using PL/SQL Block.

→

```
--------------------------------------------------------------------------------
-- PL/SQL PROGRAM TO REVERSE A STRING
--------------------------------------------------------------------------------
DECLARE
   v_input_string  VARCHAR2(100) := 'HELLO';  -- Input string to reverse
   v_reversed_string VARCHAR2(100) := '';
   v_length NUMBER;
BEGIN
   v_length := LENGTH(v_input_string);
   -- Loop through the string from the end to the beginning
   FOR i IN REVERSE 1..v_length LOOP
      v_reversed_string := v_reversed_string || SUBSTR(v_input_string, i, 1);
   END LOOP;
   -- Display the reversed string
   DBMS_OUTPUT.PUT_LINE('Original String: ' || v_input_string);
   DBMS_OUTPUT.PUT_LINE('Reversed String: ' || v_reversed_string);
END;
/
```

OUTPUT:



17 **Employee Bonus Calculation Using Cursor**

- Write a PL/SQL program using an **explicit cursor** to calculate and display a **10% bonus** for all employees whose salary is greater than **50,000**.
- Assume a table **EMPLOYEES** with columns **EMPLOYEE_ID, NAME, and SALARY**.

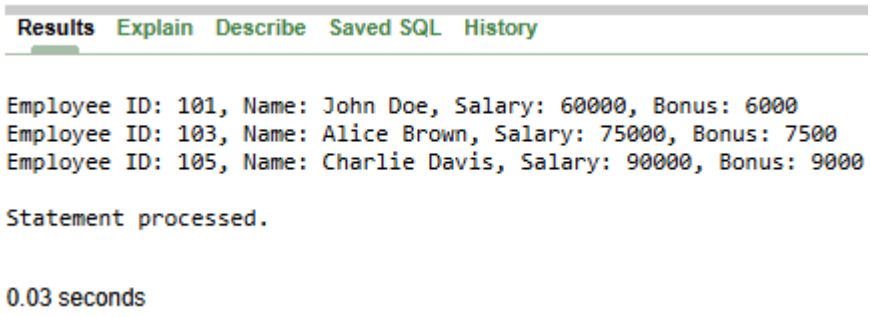
→

```
--Use it if table is already exist

DROP TABLE employees CASCADE CONSTRAINTS;


-- Step 1: Create the Employees Table

CREATE TABLE employees (

   employee_id NUMBER PRIMARY KEY,

   employee_name VARCHAR2(100),

   salary NUMBER(10,2)

);

--Step 2: Insert Sample Data

INSERT INTO employees (employee_id, employee_name, salary) VALUES (101, 'John Doe', 60000);

INSERT INTO employees (employee_id, employee_name, salary) VALUES (102, 'Jane Smith', 45000);

INSERT INTO employees (employee_id, employee_name, salary) VALUES (103, 'Alice Brown', 75000);

INSERT INTO employees (employee_id, employee_name, salary) VALUES (104, 'Bob Johnson', 30000);

INSERT INTO employees (employee_id, employee_name, salary) VALUES (105, 'Charlie Davis', 90000);

COMMIT;

--Step 3: PL/SQL Block to Calculate Bonus Using Cursor
```

```
DECLARE
   -- Declare a cursor to select employees with salary > 50,000
   CURSOR emp_cursor IS
      SELECT employee_id, employee_name, salary
      FROM employees
      WHERE salary > 50000;
   -- Variables to hold cursor values
   v_employee_id employees.employee_id%TYPE;
   v_employee_name employees.employee_name%TYPE;
   v_salary employees.salary%TYPE;
   v_bonus NUMBER(10,2);
BEGIN
   -- Open the cursor
   OPEN emp_cursor;
   LOOP
      -- Fetch data from cursor into variables
      FETCH emp_cursor INTO v_employee_id, v_employee_name, v_salary;
      -- Exit the loop when no more rows are found
      EXIT WHEN emp_cursor%NOTFOUND;
      -- Calculate 10% bonus
      v_bonus := v_salary * 0.10;
      -- Display employee details and calculated bonus
      DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id ||
                           ', Name: ' || v_employee_name ||
                           ', Salary: ' || v_salary ||
                           ', Bonus: ' || v_bonus);
```

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

OUTPUT:

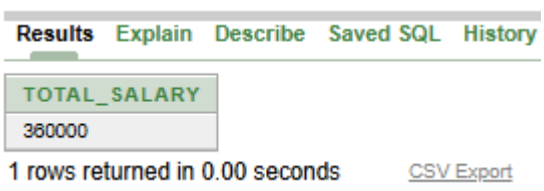


18 Write a SQL Program to implement Aggregate Functions

→

```
--Use it when table is already exist
DROP TABLE employees CASCADE CONSTRAINTS;


--1: Create the Employees Table
CREATE TABLE employees (
    employee_id NUMBER PRIMARY KEY,
    employee_name VARCHAR2(100),
    salary NUMBER(10,2),
    department VARCHAR2(50)
);
--2: Insert Sample Data
INSERT INTO employees VALUES (1, 'Alice', 60000, 'HR');
INSERT INTO employees VALUES (2, 'Bob', 75000, 'Finance');
INSERT INTO employees VALUES (3, 'Charlie', 50000, 'IT');
INSERT INTO employees VALUES (4, 'David', 85000, 'HR');
INSERT INTO employees VALUES (5, 'Emma', 90000, 'Finance');
COMMIT;
--3: Use Aggregate Functions
-- Total Salary
SELECT SUM(salary) AS total_salary FROM employees;
```

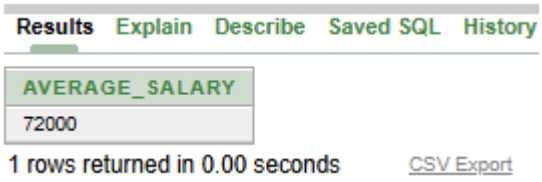
OUTPUT:



-- Average Salary

SELECT AVG(salary) AS average_salary FROM employees;

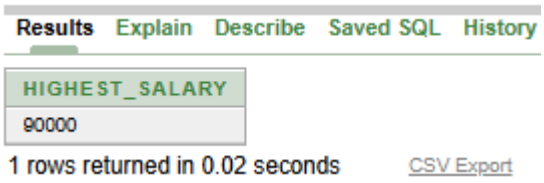
OUTPUT:



-- Highest Salary

SELECT MAX(salary) AS highest_salary FROM employees;

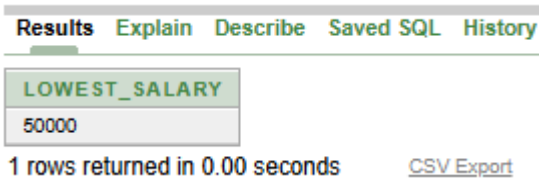
OUTPUT:



-- Lowest Salary

SELECT MIN(salary) AS lowest_salary FROM employees;

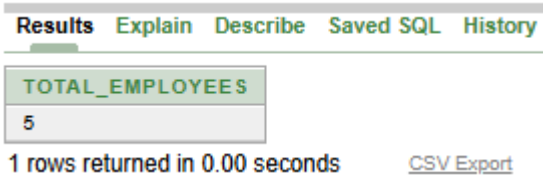
OUTPUT:



-- Count of Employees

SELECT COUNT(*) AS total_employees FROM employees;

OUTPUT:



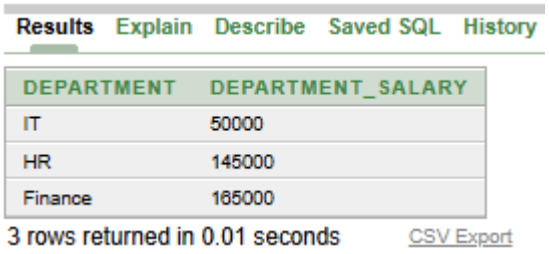
-- Group by Department

SELECT department, SUM(salary) AS department_salary

FROM employees

GROUP BY department;

OUTPUT:



Results Explain Describe Saved SQL History

DEPARTMENT	DEPARTMENT_SALARY
IT	50000
HR	145000
Finance	165000

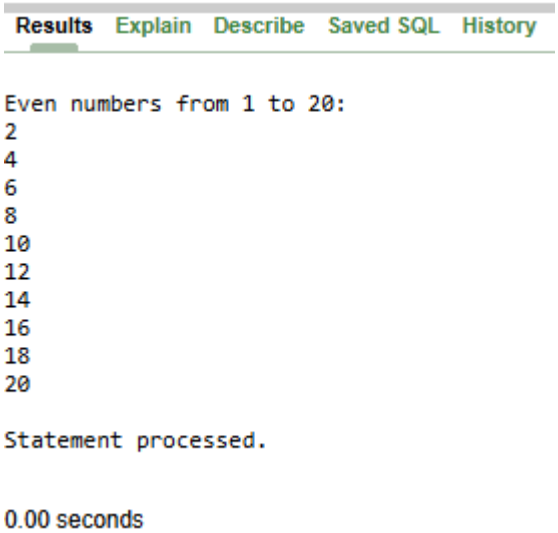
3 rows returned in 0.01 seconds CSV Export

19 Write PL/SQL code for finding Even Numbers.

→

```
DECLARE
   v_num NUMBER := 2; -- Starting number
BEGIN
   DBMS_OUTPUT.PUT_LINE('Even numbers from 1 to 20:');


   WHILE v_num <= 20 LOOP
      DBMS_OUTPUT.PUT_LINE(v_num);
      v_num := v_num + 2; -- Increment by 2 to get the next even number
   END LOOP;
END;
/
```

OUTPUT:

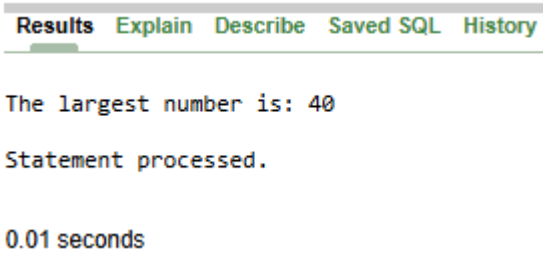


20 Write PL/SQL code to find Largest of three numbers.

→

```
DECLARE
   num1 NUMBER := 25;  -- Change values as needed
   num2 NUMBER := 40;
   num3 NUMBER := 15;
   largest NUMBER;
BEGIN
   -- Finding the largest number using IF-ELSE statements
   IF num1 >= num2 AND num1 >= num3 THEN
      largest := num1;
   ELSIF num2 >= num1 AND num2 >= num3 THEN
      largest := num2;
   ELSE
      largest := num3;
   END IF;

   DBMS_OUTPUT.PUT_LINE('The largest number is: ' || largest);
END;
/
```

OUTPUT:



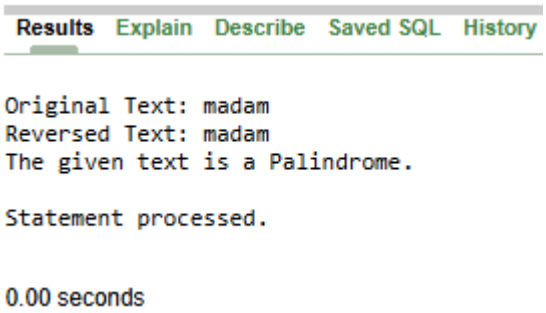
21 Write PL/SQL code to accept the text and reverse the text and test whether the given character is Palindrome or not.

→

```
DECLARE
   v_text VARCHAR2(100);
   v_reversed_text VARCHAR2(100);
BEGIN
   -- Accept input text
   v_text := 'madam';  -- Change this value to test different cases

   -- Reverse the text using a loop
   v_reversed_text := '';
   FOR i IN REVERSE 1 .. LENGTH(v_text) LOOP
      v_reversed_text := v_reversed_text || SUBSTR(v_text, i, 1);
   END LOOP;

   -- Display the original and reversed text
   DBMS_OUTPUT.PUT_LINE('Original Text: ' || v_text);
   DBMS_OUTPUT.PUT_LINE('Reversed Text: ' || v_reversed_text);

   -- Check if it's a palindrome
   IF v_text = v_reversed_text THEN
      DBMS_OUTPUT.PUT_LINE('The given text is a Palindrome.');
   ELSE
      DBMS_OUTPUT.PUT_LINE('The given text is NOT a Palindrome.');
   END IF;
END;
```

/

OUTPUT:

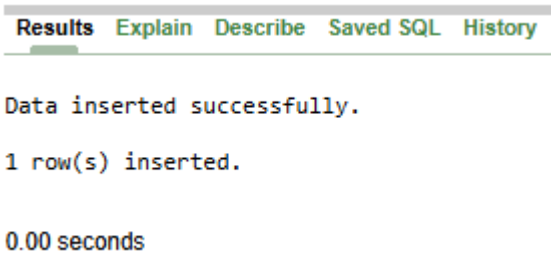

Results Explain Describe Saved SQL History
Original Text: madam
Reversed Text: madam
The given text is a Palindrome.
Statement processed.
0.00 seconds

22 Write PL/SQL code to Insert values in created tables.

→

```
-- Step 1: Create the employees table
CREATE TABLE employees (
   employee_id NUMBER PRIMARY KEY,
   employee_name VARCHAR2(100),
   salary NUMBER(10,2)
);

-- Step 2: PL/SQL block to insert values
DECLARE
   v_employee_id NUMBER := 101;
   v_employee_name VARCHAR2(100) := 'John Doe';
   v_salary NUMBER(10,2) := 55000;
BEGIN
   INSERT INTO employees (employee_id, employee_name, salary)
   VALUES (v_employee_id, v_employee_name, v_salary);

   COMMIT; -- Save changes
   DBMS_OUTPUT.PUT_LINE('Data inserted successfully.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

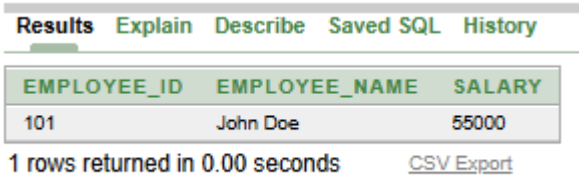
OUTPUT:



--Step 3: Check inserted data

SELECT * FROM employees;

OUTPUT:



23 Write PL/SQL code to UPDATE values in created tables by using Implicit Cursors.

→

```
--Use it if table exist
DROP TABLE employees;


-- 1 Create the employees table
BEGIN
   EXECUTE IMMEDIATE 'CREATE TABLE employees (
      employee_id NUMBER PRIMARY KEY,
      employee_name VARCHAR2(100),
      salary NUMBER(10,2)
   )';
EXCEPTION
   WHEN OTHERS THEN
      IF SQLCODE = -955 THEN
         DBMS_OUTPUT.PUT_LINE('Table already exists.');
      ELSE
         DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
      END IF;
END;
/
-- 2 Insert sample data (if table is empty)
BEGIN
   INSERT INTO employees (employee_id, employee_name, salary)
   SELECT 101, 'John Doe', 50000 FROM DUAL
   WHERE NOT EXISTS (SELECT 1 FROM employees WHERE employee_id = 101);
```

```
    INSERT INTO employees (employee_id, employee_name, salary)
    SELECT 102, 'Jane Smith', 45000 FROM DUAL
    WHERE NOT EXISTS (SELECT 1 FROM employees WHERE employee_id = 102);
    COMMIT;
END;
/
-- 3 PL/SQL block to UPDATE salary using Implicit Cursor
DECLARE
    v_employee_id NUMBER := 101;  -- Change as needed
    v_new_salary NUMBER := 60000; -- New salary value

BEGIN
    -- Update the salary
    UPDATE employees
    SET salary = v_new_salary
    WHERE employee_id = v_employee_id;

    -- Check if the update was successful
    IF SQL%ROWCOUNT > 0 THEN
        DBMS_OUTPUT.PUT_LINE('Employee salary updated successfully.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('No matching employee found.');
    END IF;
END;
/
```

OUTPUT:


```
Employee salary updated successfully.

1 row(s) updated.
```

0.00 seconds

--4 Check Update table

SELECT * FROM employees;

OUTPUT:

EMPLOYEE_ID	EMPLOYEE_NAME	SALARY
101	John Doe	60000
102	Jane Smith	45000

2 rows returned in 0.02 seconds CSV Export

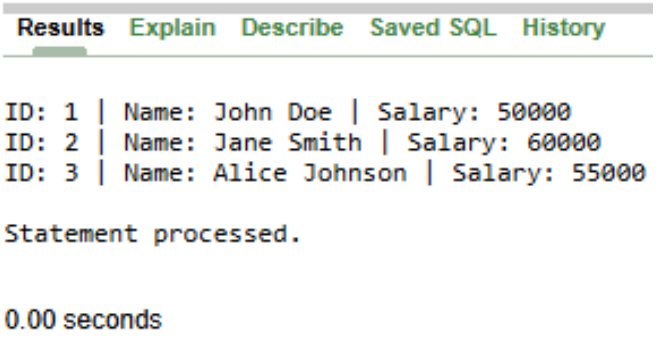
24 Write PL/SQL code to display Employee details using Explicit Cursors.

→

```
--Use it if table already exist
DROP TABLE employees;


--Step 1: Create the Table (if not already created)
CREATE TABLE employees (
   employee_id NUMBER PRIMARY KEY,
   employee_name VARCHAR2(100),
   salary NUMBER(10,2)
);
--Step 2: Insert Sample Data
INSERT INTO employees VALUES (1, 'John Doe', 50000);
INSERT INTO employees VALUES (2, 'Jane Smith', 60000);
INSERT INTO employees VALUES (3, 'Alice Johnson', 55000);
COMMIT;
--Step 3: PL/SQL Block to Display Employee Details Using Explicit Cursor
DECLARE
   CURSOR emp_cursor IS
      SELECT employee_id, employee_name, salary FROM employees;
   v_emp_id employees.employee_id%TYPE;
   v_emp_name employees.employee_name%TYPE;
   v_salary employees.salary%TYPE;
BEGIN
   OPEN emp_cursor;
   LOOP
```

```
    FETCH emp_cursor INTO v_emp_id, v_emp_name, v_salary;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('ID: ' || v_emp_id || ' | Name: ' ||
v_emp_name || ' | Salary: ' || v_salary);
  END LOOP;
  CLOSE emp_cursor;
END;
/
```

OUTPUT:



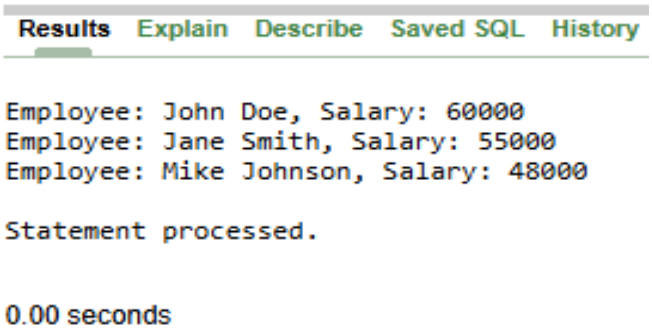
25 Write PL/SQL code in Cursor to display employee names and salary.

→

```
--Use it if table already exist
DROP TABLE employees;


-- Creating the employees table
CREATE TABLE employees (
   employee_id NUMBER PRIMARY KEY,
   employee_name VARCHAR2(100),
   salary NUMBER(10,2)
);
-- Inserting sample data
INSERT INTO employees (employee_id, employee_name, salary) VALUES (1, 'John Doe', 60000);
INSERT INTO employees (employee_id, employee_name, salary) VALUES (2, 'Jane Smith', 55000);
INSERT INTO employees (employee_id, employee_name, salary) VALUES (3, 'Mike Johnson', 48000);
COMMIT;
-- PL/SQL block using an explicit cursor to display employee names and salary
DECLARE
   CURSOR emp_cursor IS
      SELECT employee_name, salary FROM employees;
   v_name employees.employee_name%TYPE;
   v_salary employees.salary%TYPE;
BEGIN
   OPEN emp_cursor;
```

```
    LOOP
        FETCH emp_cursor INTO v_name, v_salary;
        EXIT WHEN emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Employee: ' || v_name || ', Salary: ' ||
v_salary);
    END LOOP;
    CLOSE emp_cursor;
END;
/
```

OUTPUT:



26 Write PL/SQL Programs in Cursors using two cursors at a time.

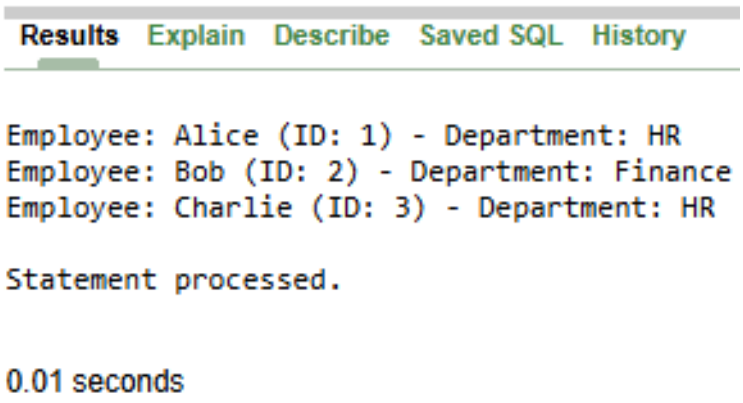
→

```
--Use it if table already exist
DROP TABLE Employees;
DROP TABLE departments;

--Step 1: Create Tables
-- Create Employees Table
CREATE TABLE employees (
   employee_id NUMBER PRIMARY KEY,
   employee_name VARCHAR2(100),
   salary NUMBER(10,2),
   department_id NUMBER
);
-- Create Departments Table
CREATE TABLE departments (
   department_id NUMBER PRIMARY KEY,
   department_name VARCHAR2(100)
);
--Step 2: Insert Sample Data
-- Insert Sample Employees
INSERT INTO employees VALUES (1, 'Alice', 60000, 101);
INSERT INTO employees VALUES (2, 'Bob', 55000, 102);
INSERT INTO employees VALUES (3, 'Charlie', 70000, 101);
-- Insert Sample Departments
INSERT INTO departments VALUES (101, 'HR');
```

```
INSERT INTO departments VALUES (102, 'Finance');
INSERT INTO departments VALUES (103, 'IT');
COMMIT;
--Step 3: PL/SQL Program Using Two Cursors
DECLARE
   -- Cursor to fetch employee details
   CURSOR emp_cursor IS
      SELECT employee_id, employee_name, department_id FROM employees;
   -- Cursor to fetch department details
   CURSOR dept_cursor IS
      SELECT department_id, department_name FROM departments;
   -- Variables to store fetched values
   v_emp_id employees.employee_id%TYPE;
   v_emp_name employees.employee_name%TYPE;
   v_emp_dept_id employees.department_id%TYPE;
   v_dept_id departments.department_id%TYPE;
   v_dept_name departments.department_name%TYPE;
BEGIN
   -- Open and fetch from first cursor
   OPEN emp_cursor;
   LOOP
      FETCH emp_cursor INTO v_emp_id, v_emp_name, v_emp_dept_id;
      EXIT WHEN emp_cursor%NOTFOUND;
      -- Open and fetch from second cursor
      OPEN dept_cursor;
      LOOP
```

```
        FETCH dept_cursor INTO v_dept_id, v_dept_name;
        EXIT WHEN dept_cursor%NOTFOUND;
        -- Match employee department with department table
        IF v_emp_dept_id = v_dept_id THEN
           DBMS_OUTPUT.PUT_LINE('Employee: ' || v_emp_name || ' (ID: ' ||
v_emp_id || ') - Department: ' || v_dept_name);
        END IF;
      END LOOP;
      CLOSE dept_cursor;
   END LOOP;
   CLOSE emp_cursor;
END;
/
```

OUTPUT:

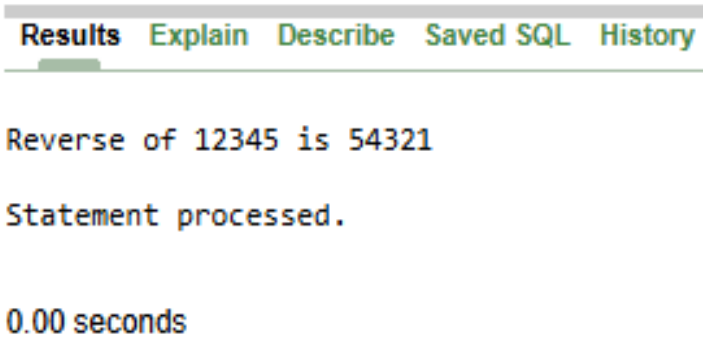


27 Write PL/SQL code in Procedure to find Reverse number

→

```
-- Create or replace the procedure
CREATE OR REPLACE PROCEDURE Reverse_Number(n IN NUMBER, rev OUT NUMBER) AS
   temp NUMBER := n;
   remainder NUMBER;
   result NUMBER := 0;
BEGIN
   WHILE temp > 0 LOOP
      remainder := MOD(temp, 10);
      result := result * 10 + remainder;
      temp := TRUNC(temp / 10);
   END LOOP;
   rev := result;
END Reverse_Number;  -- Ensure you add the procedure name at the END statement
/
-- Declare a block to test the procedure
DECLARE
   num NUMBER := 12345;
   reversed NUMBER;
BEGIN
   Reverse_Number(num, reversed);
   DBMS_OUTPUT.PUT_LINE('Reverse of ' || num || ' is ' || reversed);
END;
/
```

OUTPUT:


Results Explain Describe Saved SQL History
Reverse of 12345 is 54321
Statement processed.
0.00 seconds

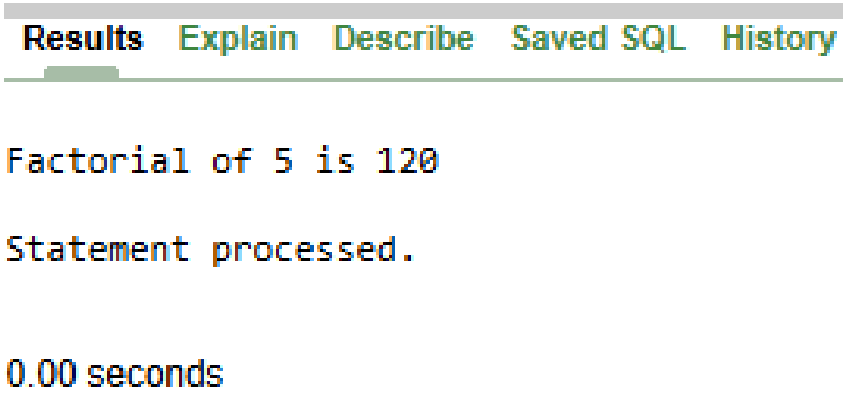
28 Write PL/SQL code in Procedure to find Factorial of a given number by using call Procedure.

→

```
--Step 1: Create the Procedure
CREATE OR REPLACE PROCEDURE Find_Factorial(
   num IN NUMBER,
   fact OUT NUMBER
) AS
   result NUMBER := 1;
   i NUMBER;
BEGIN
   IF num < 0 THEN
      fact := NULL;
   ELSE
      FOR i IN 1..num LOOP
         result := result * i;
      END LOOP;
      fact := result;
   END IF;
END;
/
--Step 2: Call the Procedure
DECLARE
   number_input NUMBER := 5;
   factorial_result NUMBER;
BEGIN
   Find_Factorial(number_input, factorial_result);
```

```
   DBMS_OUTPUT.PUT_LINE('Factorial of ' || number_input || ' is ' ||
factorial_result);

END;

/
```

OUTPUT:



29 **Write a procedure to retrieve the salary of a particular employee.**

→

```
--Use it if Table already exist.
DROP TABLE employees;


--Step 1: Create the Employee Table
CREATE TABLE employees (
   emp_id NUMBER PRIMARY KEY,
   emp_name VARCHAR2(100),
   salary NUMBER
);
--Step 2: Insert Sample Data
INSERT INTO employees VALUES (101, 'Alice', 50000);
INSERT INTO employees VALUES (102, 'Bob', 60000);
INSERT INTO employees VALUES (103, 'Charlie', 55000);
COMMIT;
--Step 3: Create the Procedure
CREATE OR REPLACE PROCEDURE Get_Salary(empId IN NUMBER, empSalary
OUT NUMBER) AS
BEGIN
   SELECT salary INTO empSalary FROM employees WHERE emp_id = empId;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      empSalary := NULL;
      DBMS_OUTPUT.PUT_LINE('Employee not found.');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An error occurred.');
```

```
END;
/
--Step 4: Call the Procedure
DECLARE
   salary NUMBER;
BEGIN
   Get_Salary(102, salary);
   IF salary IS NOT NULL THEN
      DBMS_OUTPUT.PUT_LINE('Salary: ' || salary);
   ELSE
      DBMS_OUTPUT.PUT_LINE('No salary found for the given Employee ID.');
   END IF;
END;
/
```

OUTPUT: