**SQL HANDSON ASSESSMENT**

**Question-1**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE Employees(

Employee\_id INT PRIMARY KEY,

Name VARCHAR(100),

Position VARCHAR(100),

Salary INT

);

CREATE TABLE Employees\_Audit(

Audit\_id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

Employee\_id INT,

Operation VARCHAR(10),

Change\_date DATE,

Old\_name VARCHAR(100),

Old\_position VARCHAR(100),

Old\_salary INT,

New\_name VARCHAR(100),

New\_position VARCHAR(100),

New\_salary INT

)

CREATE OR REPLACE FUNCTION Employees\_Audit\_Trigger()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' THEN

INSERT INTO Employees\_Audit (

Employee\_id, Operation, Change\_date,

Old\_name, Old\_position, Old\_salary,

New\_name, New\_position, New\_salary

) VALUES (

NEW.Employee\_id,

TG\_OP,

CURRENT\_DATE,

NULL, NULL, NULL, -- No old values on INSERT

NEW.Name, NEW.Position, NEW.Salary

);

ELSIF TG\_OP = 'UPDATE' THEN

INSERT INTO Employees\_Audit (

Employee\_id, Operation, Change\_date,

Old\_name, Old\_position, Old\_salary,

New\_name, New\_position, New\_salary

) VALUES (

NEW.Employee\_id,

TG\_OP,

CURRENT\_DATE,

OLD.Name, OLD.Position, OLD.Salary,

NEW.Name, NEW.Position, NEW.Salary

);

ELSIF TG\_OP = 'DELETE' THEN

INSERT INTO Employees\_Audit (

Employee\_id, Operation, Change\_date,

Old\_name, Old\_position, Old\_salary,

New\_name, New\_position, New\_salary

) VALUES (

OLD.Employee\_id,

TG\_OP,

CURRENT\_DATE,

OLD.Name, OLD.Position, OLD.Salary,

NULL, NULL, NULL -- No new values on DELETE

);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER employee\_trigger

AFTER INSERT OR UPDATE OR DELETE on Employees

FOR EACH ROW

EXECUTE PROCEDURE Employees\_Audit\_Trigger();

-- Insert Queries

INSERT INTO Employees VALUES(1, 'Vivek', 'CEO', 690000);

INSERT INTO Employees VALUES(2, 'Vinay', 'CTO', 29000);

INSERT INTO Employees VALUES(3, 'Manju', 'HR', 10000);

INSERT INTO Employees VALUES(4, 'Manoj', 'IT', 20000);

-- Update Queries

UPDATE Employees SET Position='Finance' WHERE Employee\_id=4;

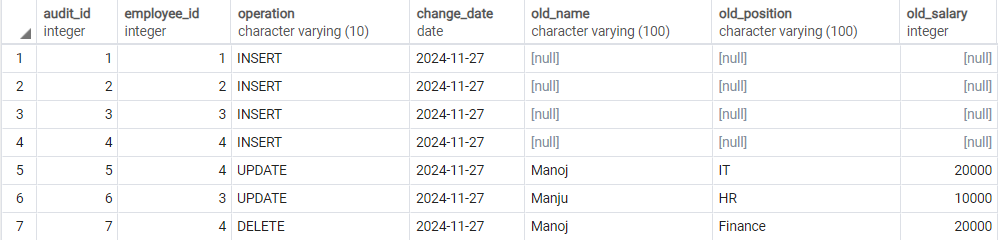
UPDATE Employees SET Salary=43000 WHERE Employee\_id=3;

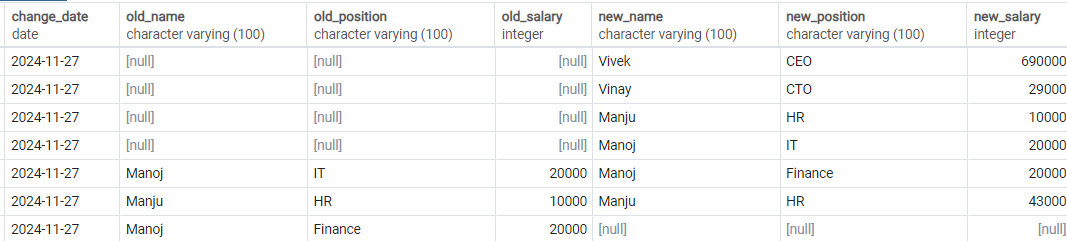
-- DELETE Queries

DELETE FROM Employees WHERE Employee\_id=4;

-- Viewing the Audit Logs

SELECT \* FROM Employees\_Audit;





**Question-2**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE Orders(

Order\_id INT PRIMARY KEY,

Customer\_id INT,

Order\_Date DATE,

Status VARCHAR(50),

Total\_Amount INT

);

-- FUNCTION TO CHANGE AND UPDATE PRODUCT STATUS BASED ON THE VALUES PROVIDED

CREATE OR REPLACE FUNCTION Order\_Status\_Update(

P\_Order\_id INT,

Payment\_Received BOOLEAN,

Shipment\_Made BOOLEAN,

Delivery\_Confirmed BOOLEAN

) RETURNS VOID

LANGUAGE plpgsql

AS $$

DECLARE

Current\_Status VARCHAR(50);

P\_Order\_Date DATE;

BEGIN

IF NOT EXISTS (SELECT 1 FROM Orders WHERE Order\_id = P\_Order\_id) THEN

RAISE NOTICE 'Order with ID % does not exist.', Order\_id;

RETURN;

END IF;

SELECT Status, Order\_date INTO Current\_Status, P\_Order\_Date FROM Orders WHERE Order\_id = P\_Order\_id;

IF Current\_Status = 'Pending' AND Payment\_Received THEN

UPDATE Orders SET Status = 'Processed' WHERE Order\_id = P\_Order\_id;

RAISE NOTICE 'Order status updated to Processed.';

ELSIF Current\_Status = 'Processed' AND Shipment\_Made THEN

UPDATE Orders SET Status = 'Shipped' WHERE Order\_id = P\_Order\_id;

RAISE NOTICE 'Order status updated to Shipped.';

ELSIF Current\_Status = 'Shipped' AND Delivery\_Confirmed THEN

UPDATE Orders SET Status = 'Delivered' WHERE Order\_id = P\_Order\_id;

RAISE NOTICE 'Order status updated to Delivered.';

ELSIF Current\_Status = 'Pending' AND CURRENT\_DATE - P\_Order\_Date > 7 THEN

UPDATE Orders SET Status = 'Cancelled' WHERE Order\_id = P\_Order\_id;

RAISE NOTICE 'Order status updated to Cancelled due to overdue.';

ELSE

RAISE NOTICE 'No Status Change Required for Order %.', P\_Order\_id;

END IF;

END;

$$;

-- INSERT QUERIES

INSERT INTO Orders (Order\_id, Customer\_id, Order\_Date, Status, Total\_Amount)

VALUES (1, 101, '2024-11-25', 'Pending', 1500);

INSERT INTO Orders (Order\_id, Customer\_id, Order\_Date, Status, Total\_Amount)

VALUES (2, 102, '2024-11-24', 'Processed', 2500);

INSERT INTO Orders (Order\_id, Customer\_id, Order\_Date, Status, Total\_Amount)

VALUES (3, 103, '2024-11-23', 'Shipped', 2000);

INSERT INTO Orders (Order\_id, Customer\_id, Order\_Date, Status, Total\_Amount)

VALUES (4, 104, '2024-11-21', 'Delivered', 3000);

INSERT INTO Orders (Order\_id, Customer\_id, Order\_Date, Status, Total\_Amount)

VALUES (5, 105, '2024-11-25', 'Pending', 1200);

INSERT INTO Orders (Order\_id, Customer\_id, Order\_Date, Status, Total\_Amount)

VALUES (6, 106, '2024-11-10', 'Pending', 1800);

-- Operations

SELECT Order\_Status\_Update(1, TRUE, FALSE, FALSE) -- Status changed to Processed

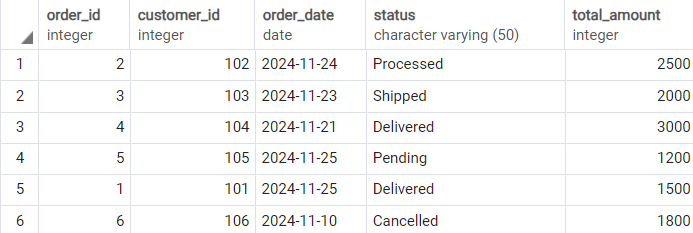
SELECT Order\_Status\_Update(1, TRUE, TRUE, FALSE) -- Status changed to Shipped

SELECT Order\_Status\_Update(1, TRUE, TRUE, TRUE) -- Status changed to Processed

SELECT Order\_Status\_Update(6, FALSE, FALSE, FALSE)

-- Viewing the orders

SELECT \* FROM Orders;



**Question-3**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE Students(

Student\_id INT PRIMARY KEY,

Name VARCHAR(100),

Score INT

)

-- INSERT QUERIES

INSERT INTO Students (Student\_id, Name, Score)

VALUES (1, 'Alice', 85);

INSERT INTO Students (Student\_id, Name, Score)

VALUES (2, 'Bob', 92);

INSERT INTO Students (Student\_id, Name, Score)

VALUES (3, 'Charlie', 78);

INSERT INTO Students (Student\_id, Name, Score)

VALUES (4, 'David', 88);

INSERT INTO Students (Student\_id, Name, Score)

VALUES (5, 'Eva', 95);

INSERT INTO Students (Student\_id, Name, Score)

VALUES (6, 'Frank', 80);

INSERT INTO Students (Student\_id, Name, Score)

VALUES (7, 'Grace', 91);

INSERT INTO Students (Student\_id, Name, Score)

VALUES (8, 'Hannah', 74);

INSERT INTO Students (Student\_id, Name, Score)

VALUES (9, 'Ivy', 89);

INSERT INTO Students (Student\_id, Name, Score)

VALUES (10, 'Jack', 82);

SELECT \* FROM Students

-- PROCEDURE PART

CREATE OR REPLACE FUNCTION Student\_Grade()

RETURNS VOID as $$

DECLARE

record Students%ROWTYPE;

BEGIN

ALTER TABLE Students ADD COLUMN Grade VARCHAR(50);

ALTER TABLE Students ADD COLUMN Remarks VARCHAR(50);

FOR record in (SELECT \* FROM Students)

LOOP

UPDATE Students

SET Grade =

CASE

WHEN Score >= 90 THEN 'A'

WHEN Score BETWEEN 80 AND 89 THEN 'B'

WHEN Score BETWEEN 70 AND 79 THEN 'C'

WHEN Score BETWEEN 60 AND 69 THEN 'D'

ELSE 'F'

END,

Remarks =

CASE

WHEN Score >= 90 THEN 'Excellent'

WHEN Score BETWEEN 80 AND 89 THEN 'Good Job'

WHEN Score BETWEEN 70 and 79 THEN 'Fair Effort'

WHEN Score < 60 THEN 'Needs Improvement'

ELSE ''

END

WHERE Student\_id = record.Student\_id;

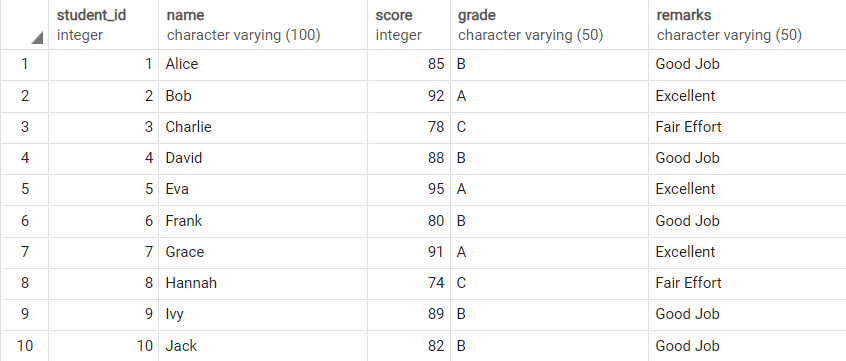
END LOOP;

END;

$$ LANGUAGE plpgsql;

SELECT Student\_Grade();

SELECT \* FROM Students



**Question-4**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE Students\_2(

Student\_id INT PRIMARY KEY,

Name VARCHAR(100),

Score INT

);

CREATE TABLE audit\_log (

log\_id INT PRIMARY KEY,

student\_id INT,

log\_message VARCHAR(255),

log\_date DATE

);

CREATE SEQUENCE audit\_log\_seq

START WITH 1

INCREMENT BY 1

NO CYCLE;

-- INSERT QUERIES

-- INSERT QUERIES

INSERT INTO Students\_2 (Student\_id, Name, Score)

VALUES (1, 'Alice', 85);

INSERT INTO Students\_2 (Student\_id, Name, Score)

VALUES (2, 'Bob', 92);

INSERT INTO Students\_2 (Student\_id, Name, Score)

VALUES (3, 'Charlie', 78);

INSERT INTO Students\_2 (Student\_id, Name, Score)

VALUES (4, 'David', 70);

INSERT INTO Students\_2 (Student\_id, Name, Score)

VALUES (5, 'Eva', 68);

INSERT INTO Students\_2 (Student\_id, Name, Score)

VALUES (6, 'Frank', 80);

INSERT INTO Students\_2 (Student\_id, Name, Score)

VALUES (7, 'Grace', 21);

INSERT INTO Students\_2 (Student\_id, Name, Score)

VALUES (8, 'Hannah', 74);

INSERT INTO Students\_2 (Student\_id, Name, Score)

VALUES (9, 'Ivy', 89);

INSERT INTO Students\_2 (Student\_id, Name, Score)

VALUES (10, 'Jack', 43);

-- CREATING THE PROCEDURE

ALTER TABLE Students\_2 ADD COLUMN Grade VARCHAR(50);

CREATE OR REPLACE FUNCTION Student\_Grade\_Log()

RETURNS VOID AS $$

DECLARE

rec RECORD;

s\_grade VARCHAR(2);

s\_log VARCHAR(100);

BEGIN

FOR rec in (SELECT \* FROM Students\_2)

LOOP

IF rec.Score >= 90 THEN

s\_grade := 'A';

ELSIF rec.Score BETWEEN 80 AND 89 THEN

s\_grade := 'B';

ELSIF rec.Score BETWEEN 70 AND 79 THEN

s\_grade := 'C';

ELSIF rec.Score BETWEEN 60 AND 69 THEN

s\_grade := 'D';

ELSE

s\_grade := 'F';

END IF;

UPDATE Students\_2 SET Grade = s\_grade WHERE Student\_id = rec.Student\_id;

s\_log := 'Student with ID ' || rec.Student\_id || ' assigned Grade: ' || s\_grade;

INSERT INTO audit\_log(

log\_id, student\_id, log\_message, log\_date

) VALUES(

NEXTVAL('audit\_log\_seq'), rec.Student\_id, s\_log, CURRENT\_DATE

);

END LOOP;

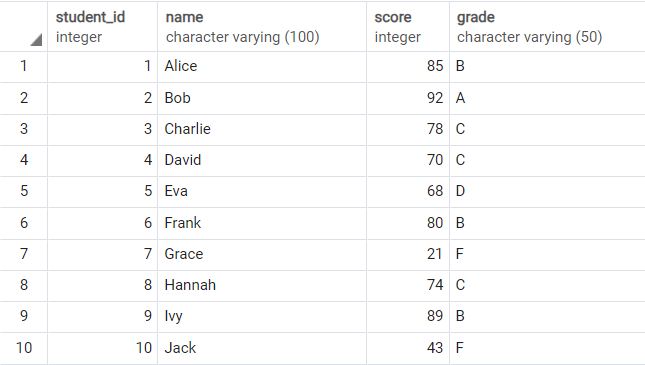
END;

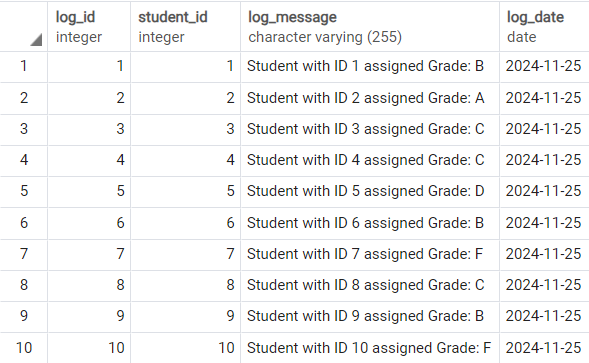
$$ LANGUAGE plpgsql;

SELECT Student\_Grade\_Log();

SELECT \* FROM Students\_2;

SELECT \* FROM audit\_log;





**Question-5**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE Orders\_2(

Order\_id INT PRIMARY KEY,

Customer\_id INT,

Order\_Date DATE,

Total\_Amount NUMERIC(10, 2),

Discounted\_Amount NUMERIC(10, 2)

);

INSERT INTO Orders\_2 (Order\_id, Customer\_id, Order\_Date, Total\_Amount)

VALUES (1, 101, '2024-11-25', 700);

INSERT INTO Orders\_2 (Order\_id, Customer\_id, Order\_Date, Total\_Amount)

VALUES (2, 102, '2024-11-20', 300);

INSERT INTO Orders\_2 (Order\_id, Customer\_id, Order\_Date, Total\_Amount)

VALUES (3, 103, '2024-11-25', 200);

INSERT INTO Orders\_2 (Order\_id, Customer\_id, Order\_Date, Total\_Amount)

VALUES (4, 104, '2024-11-20', 499.99);

INSERT INTO Orders\_2 (Order\_id, Customer\_id, Order\_Date, Total\_Amount)

VALUES (5, 105, '2024-11-10', 199.99);

select \* from Orders\_2

-- PROCEDURE PART

CREATE OR REPLACE FUNCTION Order\_Discount()

RETURNS VOID AS $$

DECLARE

order\_count INT;

order\_no INT;

discount NUMERIC(10, 2);

total\_amt NUMERIC(10, 2);

BEGIN

SELECT COUNT(\*) INTO order\_count FROM Orders\_2;

order\_no := 1;

WHILE order\_no <= order\_count

LOOP

SELECT Total\_Amount into total\_amt FROM Orders\_2 WHERE Order\_id=order\_no;

UPDATE Orders\_2

SET Discounted\_Amount =

CASE

WHEN total\_amt >= 500 THEN total\_amt - 0.2 \* total\_amt

WHEN total\_amt BETWEEN 200 AND 499.99 THEN total\_amt - 0.1 \* total\_amt

WHEN total\_amt < 200 THEN total\_amt

END

WHERE Order\_id = order\_no;

order\_no := order\_no + 1;

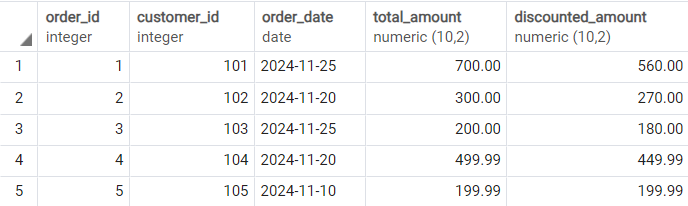
END LOOP;

END;

$$ LANGUAGE plpgsql

SELECT Order\_Discount();

select \* from Orders\_2



**Question-6**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE Employees\_2(

EmployeeID INT PRIMARY KEY,

Name VARCHAR(100),

Status VARCHAR(50)

);

INSERT INTO Employees\_2 VALUES(1, 'Vivek', 'Active');

INSERT INTO Employees\_2 VALUES(2, 'Vinay', 'Active');

INSERT INTO Employees\_2 VALUES(3, 'Manju', 'On Leave');

INSERT INTO Employees\_2 VALUES(4, 'Manoj', 'Resigned');

INSERT INTO Employees\_2 VALUES(5, 'Jake', 'Active');

INSERT INTO Employees\_2 VALUES(6, 'Logan', 'On Leave');

INSERT INTO Employees\_2 VALUES(7, 'Josh', 'Active');

SELECT \* FROM Employees\_2;

DO $$

DECLARE

rec Record;

BEGIN

FOR rec in (SELECT \* FROM Employees\_2)

LOOP

IF rec.Status = 'On Leave' THEN

CONTINUE;

ELSE

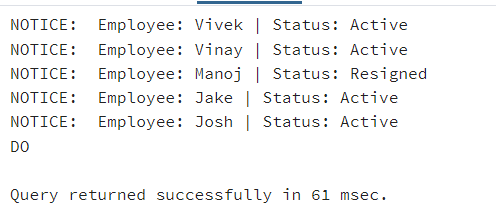
RAISE NOTICE 'Employee: % | Status: %', rec.Name, rec.Status;

END IF;

END LOOP;

END;

$$ LANGUAGE plpgsql;



**Question-7**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE Employees\_3(

Employee\_id INT PRIMARY KEY,

First\_Name VARCHAR(50),

Last\_Name VARCHAR(50),

Salary NUMERIC(10, 2),

Performance\_Rating INT CHECK(Performance\_Rating BETWEEN 1 AND 5)

);

CREATE TABLE Salary\_Adjustments(

Rating INT,

Adjustment\_Percentage INT

);

INSERT INTO Salary\_Adjustments VALUES (5, 20), (4, 10), (3, 5), (2, 0), (1, -5);

SELECT \* FROM Salary\_Adjustments;

INSERT INTO Employees\_3 VALUES(1, 'Vivek', 'Nair', 45000, 5);

INSERT INTO Employees\_3 VALUES(2, 'Vinay', 'Nair', 15000, 1);

INSERT INTO Employees\_3 VALUES(3, 'Jake', 'Paul', 6200, 4);

INSERT INTO Employees\_3 VALUES(4, 'Logan', 'Paul', 68000, 3);

INSERT INTO Employees\_3 VALUES(5, 'Linkin', 'Park', 32000, 2);

INSERT INTO Employees\_3 VALUES(6, 'Bruno', 'Mars', 73000, 1);

INSERT INTO Employees\_3 VALUES(7, 'John', 'Doe', 85000, 4);

INSERT INTO Employees\_3 VALUES(8, 'Josh', 'Rich', 21000, 3);

SELECT \* FROM Employees\_3;

-- PROCEDURE CREATION

CREATE OR REPLACE FUNCTION Employee\_Salary\_Adjustments()

RETURNS VOID AS $$

DECLARE

emp\_rating INT;

salary\_diff NUMERIC(10, 2);

rec RECORD;

BEGIN

FOR rec in (SELECT \* FROM Employees\_3)

LOOP

SELECT Adjustment\_Percentage/100.0 INTO salary\_diff FROM Salary\_Adjustments

WHERE Rating = rec.Performance\_Rating;

UPDATE Employees\_3

SET Salary = Salary + salary\_diff \* Salary

WHERE Employee\_id = rec.Employee\_id;

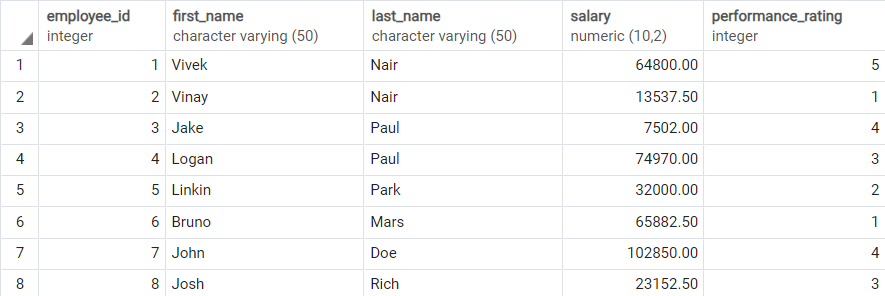
END LOOP;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM Employee\_Salary\_Adjustments();

SELECT \* FROM Employees\_3;



**Question-8**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE Performance\_Rating(

Rating INT,

Bonus\_Percentage INT

);

INSERT INTO Performance\_Rating VALUES (5, 25), (4, 15), (3, 10), (2, 5), (1, 0);

-- Here we are referring to the Employee\_3 Table for Bonus Calculation

SELECT \* FROM Employees\_3;

CREATE OR REPLACE FUNCTION Employee\_Bonus(

Emp\_ID INT

) RETURNS DECIMAL as $$

DECLARE

bonus DECIMAL(10, 2);

emp\_rating INT;

BEGIN

SELECT Performance\_Rating INTO emp\_rating FROM Employees\_3 WHERE Employee\_id = Emp\_ID;

bonus := (SELECT Bonus\_Percentage FROM Performance\_Rating WHERE Rating = emp\_rating) / 100.0 \*

(SELECT salary FROM Employees\_3 WHERE Employee\_id = Emp\_ID);

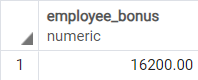
RETURN bonus;

END;

$$ LANGUAGE plpgsql

-- Calling the function with corresponding Employee ID whose bonus is to be found

SELECT \* FROM Employee\_Bonus(1);



**Question-9**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE Orders\_3 (

order\_id INT PRIMARY KEY,

customer\_id INT,

order\_date DATE

);

CREATE TABLE Order\_Items (

order\_id INT,

item\_id INT,

quantity INT,

PRIMARY KEY (order\_id, item\_id)

);

CREATE TABLE Inventory (

item\_id INT PRIMARY KEY,

item\_name VARCHAR(100),

stock\_level INT,

price NUMERIC(10, 2)

);

CREATE TABLE Order\_Logs (

log\_id INT PRIMARY KEY,

order\_id INT,

log\_message VARCHAR(255),

log\_date DATE

);

CREATE SEQUENCE order\_logs\_seq

START WITH 1

INCREMENT BY 1

NO CYCLE;

-- Creation of Procedure Part

CREATE OR REPLACE FUNCTION Process\_Customer\_Order(p\_order\_id INT)

RETURNS VOID as $$

DECLARE

rec RECORD;

p\_item\_id INT;

p\_quantity INT;

p\_stock\_level INT;

p\_price NUMERIC(10, 2);

p\_total\_price NUMERIC(10, 2) := 0;

p\_order\_exists BOOLEAN;

BEGIN

SELECT EXISTS (SELECT 1 FROM Orders\_3 WHERE Order\_Id = p\_order\_id)

INTO p\_order\_exists;

IF NOT p\_order\_exists THEN

RAISE EXCEPTION 'Invalid Order ID: %', p\_order\_id;

END IF;

FOR rec in (SELECT \* FROM Order\_Items WHERE order\_id = p\_order\_id)

LOOP

p\_item\_id:= rec.item\_id;

p\_quantity := rec.quantity;

SELECT stock\_level, price into p\_stock\_level, p\_price

FROM Inventory

WHERE item\_id = p\_item\_id;

IF p\_stock\_level < p\_quantity THEN

RAISE EXCEPTION 'Insufficient stock for item ID: %', p\_item\_id;

END IF;

UPDATE Inventory

SET stock\_level = stock\_level - p\_quantity

WHERE item\_id = p\_item\_id;

p\_total\_price := p\_total\_price + (p\_quantity \* p\_price);

END LOOP;

INSERT INTO Order\_Logs VALUES(

NEXTVAL('order\_logs\_seq'),

p\_order\_id,

'Order processed successfully. Total Value: ' || p\_total\_price,

CURRENT\_DATE

);

RAISE NOTICE 'Order processed successfully. Total Value: %', p\_total\_price;

EXCEPTION

WHEN OTHERS THEN

INSERT INTO order\_logs

VALUES (NEXTVAL('order\_logs\_seq'), p\_order\_id, 'Error processing order: ' || SQLERRM, CURRENT\_DATE);

RAISE NOTICE 'Error processing order ID %: %', p\_order\_id, SQLERRM;

END;

$$ LANGUAGE plpgsql;

-- Inserting sample orders

INSERT INTO orders\_3 VALUES (1, 1, CURRENT\_DATE);

INSERT INTO orders\_3 VALUES (2, 2, CURRENT\_DATE);

-- Inserting sample order items

INSERT INTO order\_items (order\_id, item\_id, quantity) VALUES (1, 101, 2);

INSERT INTO order\_items (order\_id, item\_id, quantity) VALUES (1, 102, 1);

INSERT INTO order\_items (order\_id, item\_id, quantity) VALUES (2, 101, 5);

INSERT INTO order\_items (order\_id, item\_id, quantity) VALUES (2, 102, 3);

-- Inserting sample inventory

INSERT INTO inventory (item\_id, item\_name, stock\_level, price) VALUES (101, 'Item A', 10, 20.00);

INSERT INTO inventory (item\_id, item\_name, stock\_level, price) VALUES (102, 'Item B', 5, 15.00);

SELECT \* FROM Orders\_3;

SELECT \* FROM Order\_Items;

SELECT \* FROM Inventory;

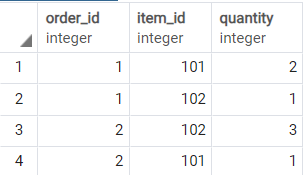
SELECT Process\_Customer\_Order(2);

SELECT \* FROM Order\_Logs;

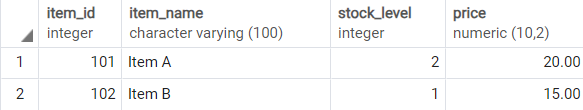
Orders



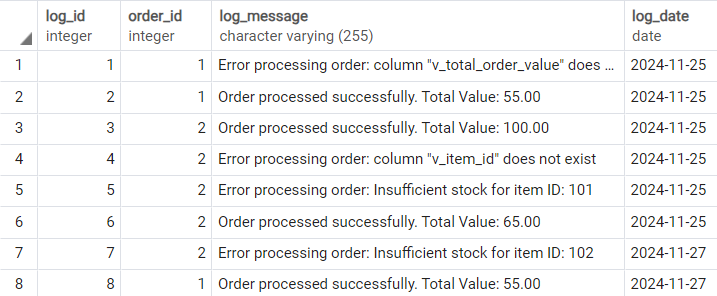
Order\_Items

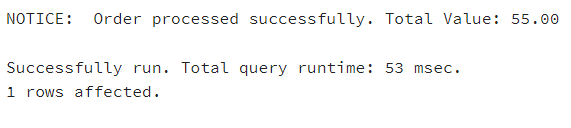


Inventory



Order\_Logs





**Question-10**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE OR REPLACE FUNCTION update\_employee\_salary(

emp\_id INT,

new\_salary NUMERIC

) RETURNS VOID AS

$$

DECLARE

v\_current\_salary NUMERIC;

min\_salary\_threshold NUMERIC;

BEGIN

SELECT salary

INTO v\_current\_salary

FROM Employee\_4

WHERE employee\_id = emp\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Error: Employee with ID % not found.', emp\_id;

END IF;

IF new\_salary < min\_salary\_threshold THEN

RAISE EXCEPTION 'Error: The new salary % is below the minimum allowed threshold of %.', new\_salary, min\_salary\_threshold;

END IF;

UPDATE Employee\_4

SET salary = new\_salary

WHERE employee\_id = emp\_id;

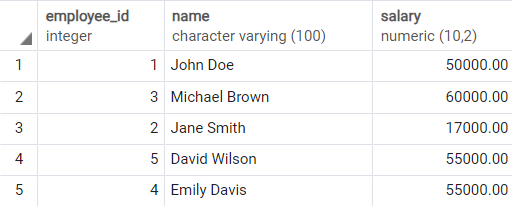
RAISE NOTICE 'Employee salary updated successfully to %.', new\_salary;

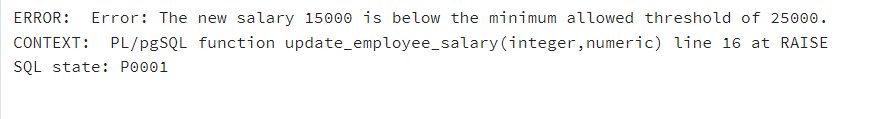
END;

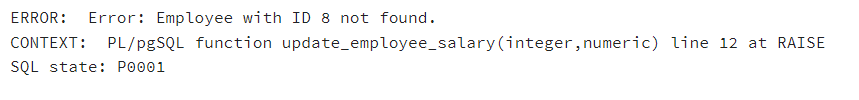
$$ LANGUAGE plpgsql;

SELECT update\_employee\_salary(4, 55000);

SELECT update\_employee\_salary(8, 55000);







**Question-11**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE departments (

department\_id INT PRIMARY KEY,

department\_name VARCHAR(100)

);

CREATE TABLE employees\_5 (

employee\_id INT PRIMARY KEY,

name VARCHAR(100),

department\_id INT,

salary NUMERIC

);

INSERT INTO departments (department\_id, department\_name) VALUES (1, 'Human Resources');

INSERT INTO departments (department\_id, department\_name) VALUES (2, 'Finance');

INSERT INTO departments (department\_id, department\_name) VALUES (3, 'IT');

INSERT INTO departments (department\_id, department\_name) VALUES (4, 'Marketing');

INSERT INTO departments (department\_id, department\_name) VALUES (5, 'Operations');

INSERT INTO employees\_5 (employee\_id, name, department\_id, salary) VALUES (1, 'Alice', 1, 60000);

INSERT INTO employees\_5 (employee\_id, name, department\_id, salary) VALUES (2, 'Bob', 2, 75000);

INSERT INTO employees\_5 (employee\_id, name, department\_id, salary) VALUES (3, 'Charlie', 3, 80000);

INSERT INTO employees\_5 (employee\_id, name, department\_id, salary) VALUES (4, 'Diana', 4, 50000);

INSERT INTO employees\_5 (employee\_id, name, department\_id, salary) VALUES (5, 'Ethan', 5, 70000);

INSERT INTO employees\_5 (employee\_id, name, department\_id, salary) VALUES (6, 'Fiona', 3, 85000);

INSERT INTO employees\_5 (employee\_id, name, department\_id, salary) VALUES (7, 'George', 2, 62000);

INSERT INTO employees\_5 (employee\_id, name, department\_id, salary) VALUES (8, 'Hannah', 4, 58000);

INSERT INTO employees\_5 (employee\_id, name, department\_id, salary) VALUES (9, 'Ian', 1, 63000);

INSERT INTO employees\_5 (employee\_id, name, department\_id, salary) VALUES (10, 'Julia', 5, 69000);

DO $$

DECLARE

rec RECORD;

total\_salary NUMERIC;

BEGIN

FOR rec in (SELECT \* FROM departments)

LOOP

SELECT COALESCE(SUM(salary), 0)

INTO total\_salary

FROM employees\_5

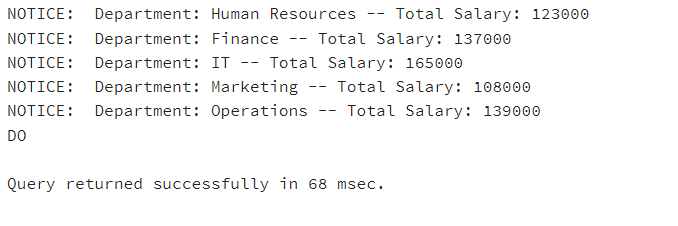
WHERE department\_id = rec.department\_id;

RAISE NOTICE 'Department: % -- Total Salary: %',rec.department\_name, total\_salary;

END LOOP;

END;

$$ LANGUAGE plpgsql;



**Question-12**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-- Here we are referring to the Employee\_3 and Performance\_Rating Table for Bonus Calculation

SELECT \* FROM Employees\_3;

SELECT \* FROM Performance\_Rating;

CREATE OR REPLACE FUNCTION Employee\_Bonus\_Procedure()

RETURNS VOID as $$

DECLARE

rec RECORD;

bonus DECIMAL(10, 2);

emp\_rating INT;

BEGIN

FOR rec IN (SELECT \* FROM Employees\_3)

LOOP

bonus := rec.salary \* (select bonus\_percentage FROM Performance\_Rating

WHERE rating = rec.performance\_rating)/100;

RAISE NOTICE 'Employee: % Salary: Rs. % Bonus: Rs. %',rec.first\_name,rec.salary,bonus;

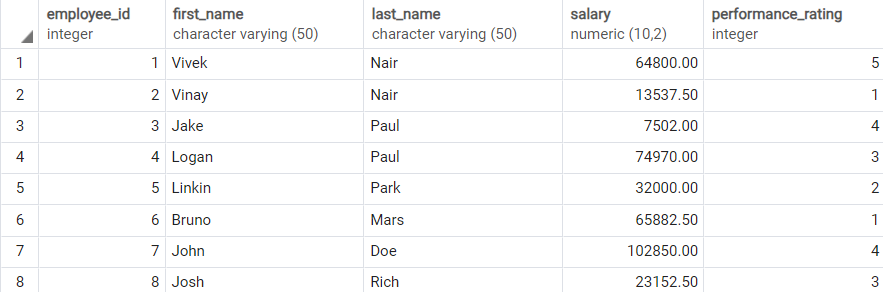
END LOOP;

END;

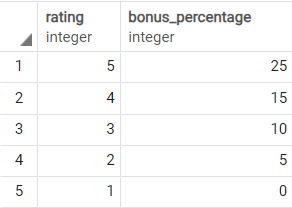
$$ LANGUAGE plpgsql

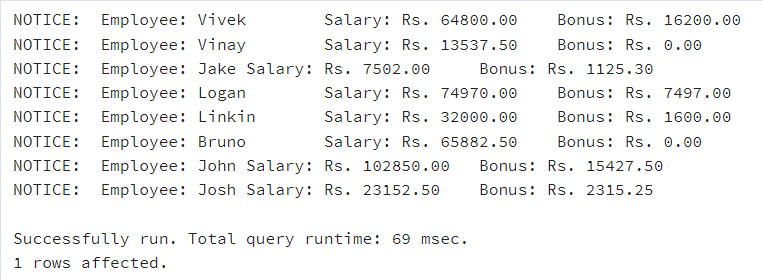
-- Calling the function with corresponding Employee ID whose bonus is to be found

SELECT \* FROM Employee\_Bonus\_Procedure();

Employees\_3

Performance\_Rating





**Question-13**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE EMPLOYEES (

Employee\_id NUMBER PRIMARY KEY,

First\_Name VARCHAR2(100),

Last\_Name VARCHAR2(100),

Hire\_date DATE,

Salary NUMBER(10, 2),

Department\_id NUMBER

);

-- Create the package specification

CREATE OR REPLACE PACKAGE employee\_mgmt\_pkg AS

-- Procedure to add a new employee

PROCEDURE add\_employee(

p\_employee\_id IN EMPLOYEES.Employee\_id%TYPE,

p\_first\_name IN EMPLOYEES.First\_Name%TYPE,

p\_last\_name IN EMPLOYEES.Last\_Name%TYPE,

p\_hire\_date IN EMPLOYEES.Hire\_date%TYPE,

p\_salary IN EMPLOYEES.Salary%TYPE,

p\_department\_id IN EMPLOYEES.Department\_id%TYPE

);

-- Procedure to update employee salary

PROCEDURE update\_employee(

p\_employee\_id IN EMPLOYEES.Employee\_id%TYPE,

p\_salary IN EMPLOYEES.Salary%TYPE

);

-- Function to retrieve employee details

FUNCTION get\_employee\_details(

p\_employee\_id IN EMPLOYEES.Employee\_id%TYPE

) RETURN EMPLOYEES%ROWTYPE;

END employee\_mgmt\_pkg;

/

-- Create the package body

CREATE OR REPLACE PACKAGE BODY employee\_mgmt\_pkg AS

-- Procedure to add a new employee

PROCEDURE add\_employee(

p\_employee\_id IN EMPLOYEES.Employee\_id%TYPE,

p\_first\_name IN EMPLOYEES.First\_Name%TYPE,

p\_last\_name IN EMPLOYEES.Last\_Name%TYPE,

p\_hire\_date IN EMPLOYEES.Hire\_date%TYPE,

p\_salary IN EMPLOYEES.Salary%TYPE,

p\_department\_id IN EMPLOYEES.Department\_id%TYPE

) IS

BEGIN

INSERT INTO EMPLOYEES

VALUES (p\_employee\_id, p\_first\_name, p\_last\_name, p\_hire\_date, p\_salary, p\_department\_id);

DBMS\_OUTPUT.PUT\_LINE('Employee added successfully.');

END add\_employee;

-- Procedure to update employee salary

PROCEDURE update\_employee(

p\_employee\_id IN EMPLOYEES.Employee\_id%TYPE,

p\_salary IN EMPLOYEES.Salary%TYPE

) IS

BEGIN

UPDATE EMPLOYEES

SET Salary = p\_salary

WHERE Employee\_id = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('No employee found with the given ID.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Employee salary updated successfully.');

END IF;

END update\_employee;

-- Function to retrieve employee details

FUNCTION get\_employee\_details(

p\_employee\_id IN EMPLOYEES.Employee\_id%TYPE

) RETURN EMPLOYEES%ROWTYPE IS

v\_employee EMPLOYEES%ROWTYPE;

BEGIN

SELECT \*

INTO v\_employee

FROM EMPLOYEES

WHERE Employee\_id = p\_employee\_id;

RETURN v\_employee;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No employee found with the given ID.');

RETURN NULL;

END get\_employee\_details;

END employee\_mgmt\_pkg;

/

BEGIN

employee\_mgmt\_pkg.add\_employee(101, 'John', 'Doe', TO\_DATE('2024-01-15', 'YYYY-MM-DD'), 50000, 10);

employee\_mgmt\_pkg.add\_employee(102, 'Jane', 'Smith', TO\_DATE('2023-03-10', 'YYYY-MM-DD'), 60000, 20);

employee\_mgmt\_pkg.add\_employee(103, 'Josh', 'Peralta', TO\_DATE('2023-02-22', 'YYYY-MM-DD'), 90000, 10);

employee\_mgmt\_pkg.add\_employee(104, 'Jimmy', 'Neutron', TO\_DATE('2023-06-03', 'YYYY-MM-DD'), 55500, 10);

END;

/

BEGIN

employee\_mgmt\_pkg.update\_employee(101, 55000);

END;

/

DECLARE

emp\_details EMPLOYEES%ROWTYPE;

BEGIN

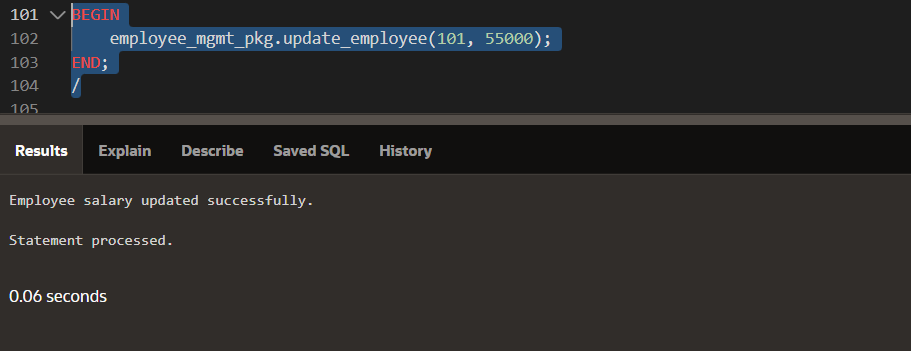
emp\_details := employee\_mgmt\_pkg.get\_employee\_details(101);

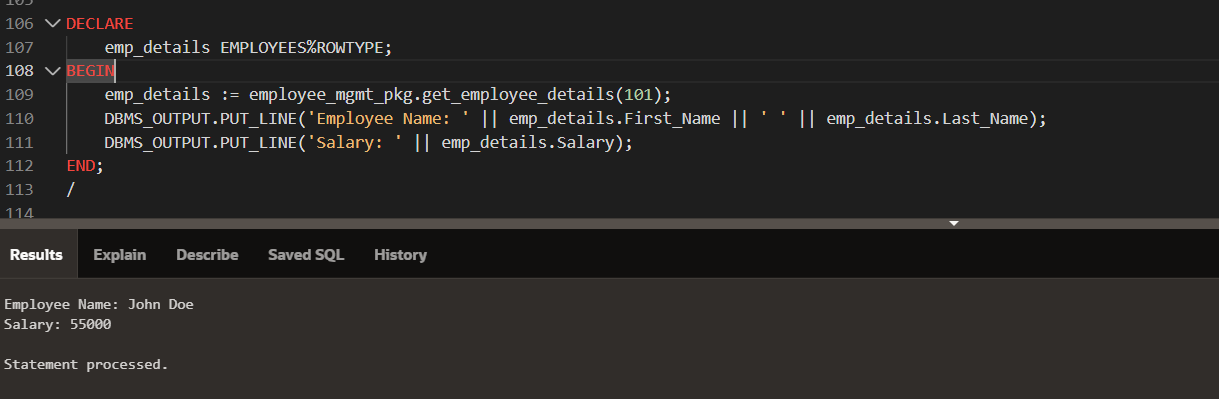
DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || emp\_details.First\_Name || ' ' || emp\_details.Last\_Name);

DBMS\_OUTPUT.PUT\_LINE('Salary: ' || emp\_details.Salary);

END;

/





**Question-14**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE ORDERS (

ORDER\_ID NUMBER PRIMARY KEY,

CUSTOMER\_ID NUMBER NOT NULL,

ORDER\_DATE DATE NOT NULL

);

CREATE TABLE ORDER\_LINES (

ORDER\_LINE\_ID NUMBER PRIMARY KEY,

ORDER\_ID NUMBER NOT NULL,

PRODUCT\_ID NUMBER NOT NULL,

QUANTITY NUMBER NOT NULL,

UNIT\_PRICE NUMBER(10, 2) NOT NULL

);

-- Create the package specification

CREATE OR REPLACE PACKAGE order\_mgmt AS

PROCEDURE create\_order(

p\_customer\_id IN NUMBER,

p\_order\_date IN DATE,

p\_order\_items IN SYS\_REFCURSOR

);

FUNCTION calculate\_order\_total(

p\_order\_id IN NUMBER

) RETURN NUMBER;

END order\_mgmt;

/

-- Create the package body

CREATE OR REPLACE PACKAGE BODY order\_mgmt AS

-- Procedure to create a new order

PROCEDURE create\_order(

p\_customer\_id IN NUMBER,

p\_order\_date IN DATE,

p\_order\_items IN SYS\_REFCURSOR

) IS

v\_order\_id NUMBER;

v\_product\_id NUMBER;

v\_quantity NUMBER;

v\_unit\_price NUMBER;

v\_order\_line\_id NUMBER := 1; -- Order Line ID counter

BEGIN

-- Generate a new order ID (using a sequence or max + 1)

SELECT NVL(MAX(ORDER\_ID), 0) + 1 INTO v\_order\_id FROM ORDERS;

INSERT INTO ORDERS (ORDER\_ID, CUSTOMER\_ID, ORDER\_DATE)

VALUES (v\_order\_id, p\_customer\_id, p\_order\_date);

LOOP

FETCH p\_order\_items INTO v\_product\_id, v\_quantity, v\_unit\_price;

EXIT WHEN p\_order\_items%NOTFOUND;

INSERT INTO ORDER\_LINES (ORDER\_LINE\_ID, ORDER\_ID, PRODUCT\_ID, QUANTITY, UNIT\_PRICE)

VALUES (v\_order\_line\_id, v\_order\_id, v\_product\_id, v\_quantity, v\_unit\_price);

v\_order\_line\_id := v\_order\_line\_id + 1;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Order created successfully with Order ID: ' || v\_order\_id);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20001, 'Error occurred while creating the order: ' || SQLERRM);

END create\_order;

FUNCTION calculate\_order\_total(

p\_order\_id IN NUMBER

) RETURN NUMBER IS

v\_total\_amount NUMBER := 0;

BEGIN

SELECT SUM(QUANTITY \* UNIT\_PRICE)

INTO v\_total\_amount

FROM ORDER\_LINES

WHERE ORDER\_ID = p\_order\_id;

RETURN NVL(v\_total\_amount, 0);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Error calculating order total: ' || SQLERRM);

END calculate\_order\_total;

END order\_mgmt;

/

-- USING THE FUNCTIONS

DECLARE

v\_order\_items SYS\_REFCURSOR;

BEGIN

-- Open a cursor with order items data

OPEN v\_order\_items FOR

SELECT 101 AS PRODUCT\_ID, 2 AS QUANTITY, 50 AS UNIT\_PRICE FROM DUAL

UNION ALL

SELECT 102, 1, 100 FROM DUAL;

-- Call the create\_order procedure

order\_mgmt.create\_order(

p\_customer\_id => 123,

p\_order\_date => SYSDATE,

p\_order\_items => v\_order\_items

);

END;

/

DECLARE

v\_total NUMBER;

BEGIN

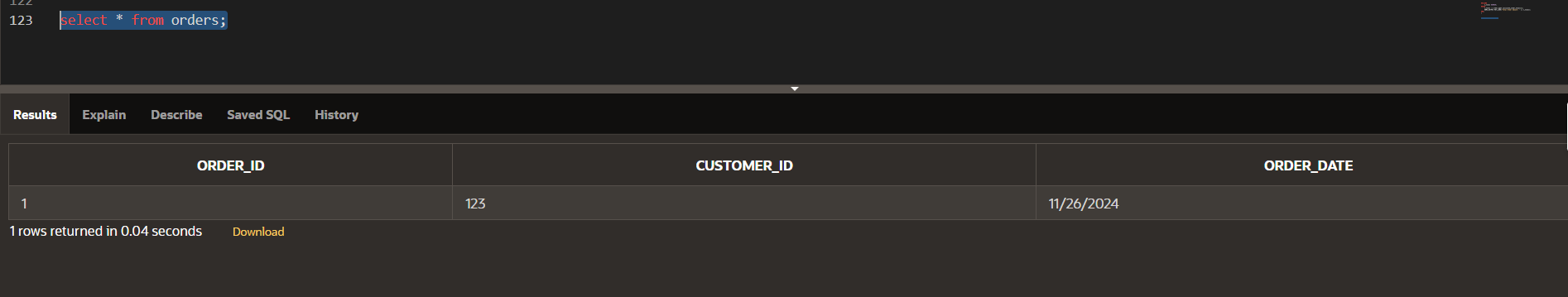
v\_total := order\_mgmt.calculate\_order\_total(1);

DBMS\_OUTPUT.PUT\_LINE('Total Order Amount: ' || v\_total);

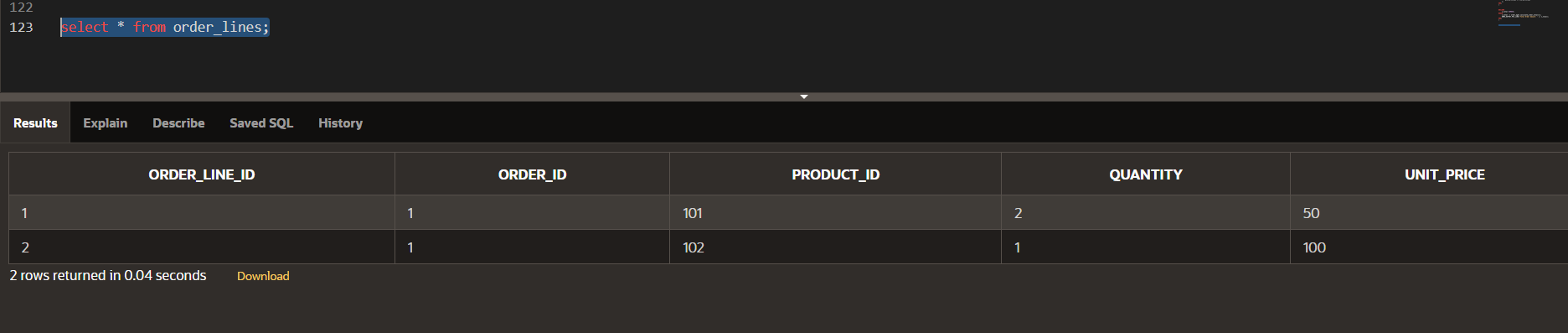
END;

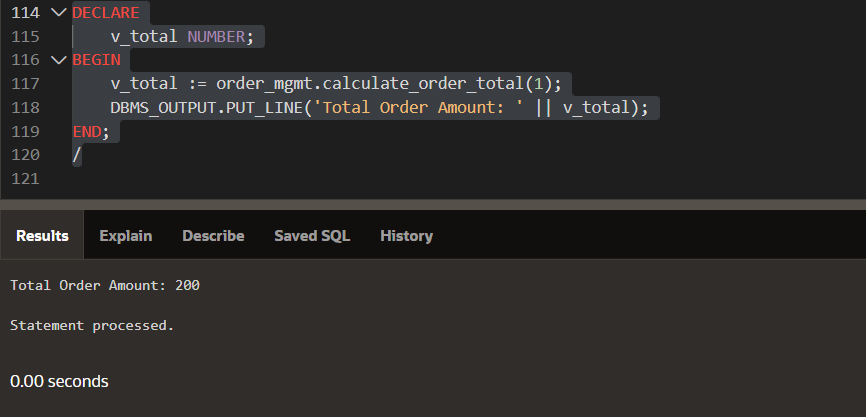
/

Orders



Order\_Lines





**Question-15**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE OR REPLACE FUNCTION Employee\_Salary\_Trigger()

RETURNS TRIGGER AS $$

DECLARE

min\_salary NUMERIC := 25000;

BEGIN

IF NEW.Salary < min\_salary THEN

RAISE EXCEPTION 'New salary cannot be less than the minimum salary threshold!';

ELSE

IF TG\_OP = 'INSERT' THEN

INSERT INTO Employees\_Audit (

Employee\_id, Operation, Change\_date,

Old\_name, Old\_position, Old\_salary,

New\_name, New\_position, New\_salary

) VALUES (

NEW.Employee\_id,

TG\_OP,

CURRENT\_DATE,

NULL, NULL, NULL,

NEW.Name, NEW.Position, NEW.Salary

);

ELSIF TG\_OP = 'UPDATE' THEN

INSERT INTO Employees\_Audit (

Employee\_id, Operation, Change\_date,

Old\_name, Old\_position, Old\_salary,

New\_name, New\_position, New\_salary

) VALUES (

NEW.Employee\_id,

TG\_OP,

CURRENT\_DATE,

OLD.Name, OLd.Position, OLD.Salary,

NEW.Name, NEW.Position, NEW.Salary

);

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

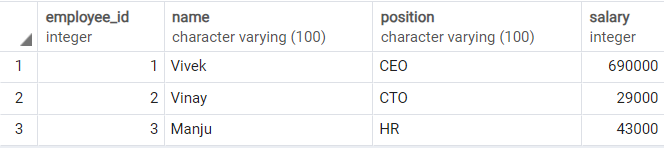
UPDATE Employees SET Salary = 40000 WHERE employee\_id = 2;

INSERT INTO Employees VALUES(4, 'Jake', 'IT', 200000);

select \* from employees;

select \* from employees\_audit;

Employees



Employees\_Audit

