**Week 2- PL/SQL Programming**

**Exercise 1: Control Structures**

**Scenario 1:**  
1. Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**CODE:**  
DO

$$

DECLARE

rec RECORD;

v\_age INT;

BEGIN

FOR rec IN SELECT CustomerID, DOB FROM Customers LOOP

v\_age := EXTRACT(YEAR FROM AGE(CURRENT\_DATE, rec.DOB));

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = rec.CustomerID;

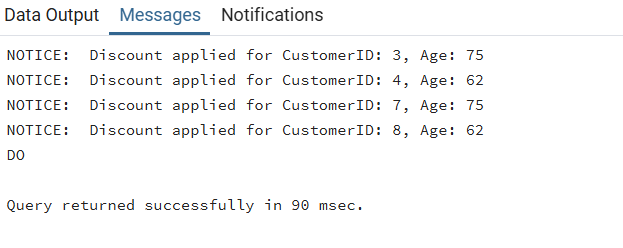
RAISE NOTICE 'Discount applied for CustomerID: %, Age: %', rec.CustomerID, v\_age;

END IF;

END LOOP;

END

$$;

**OUTPUT:  
**

**Scenario 2:**

2. Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**CODE:**

DO

$$

DECLARE

rec RECORD;

BEGIN

FOR rec IN SELECT CustomerID, Balance FROM Customers LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = TRUE

WHERE CustomerID = rec.CustomerID;

RAISE NOTICE 'VIP status given to CustomerID: %, Balance: %', rec.CustomerID, rec.Balance;

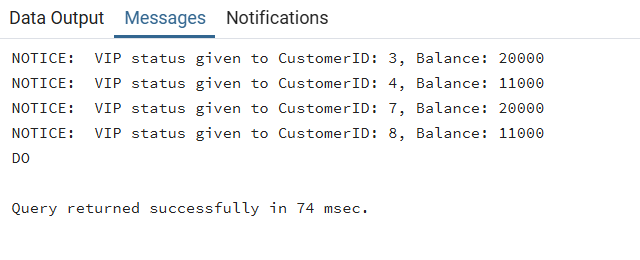
END IF;

END LOOP;

END

$$;

**OUTPUT:**

****

**Scenario 3:**

3. Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**CODE:**

DO

$$

DECLARE

rec RECORD;

BEGIN

FOR rec IN

SELECT L.LoanID, L.CustomerID, L.EndDate, C.Name

FROM Loans L

JOIN Customers C ON L.CustomerID = C.CustomerID

WHERE L.EndDate BETWEEN CURRENT\_DATE AND CURRENT\_DATE + INTERVAL '30 days'

LOOP

RAISE NOTICE 'Reminder: Loan ID % for customer % (ID: %) is due on %',

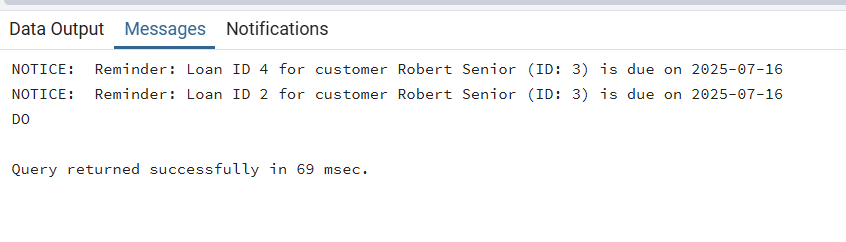
rec.LoanID, rec.Name, rec.CustomerID, rec.EndDate;

END LOOP;

END

$$;

**OUTPUT:**

****

**Exercise 3: Stored Procedures**

**Scenario 1:**

1. Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**CODE:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest()

LANGUAGE plpgsql

AS

$$

DECLARE

rec RECORD;

v\_interest NUMERIC;

BEGIN

FOR rec IN SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings' LOOP

v\_interest := rec.Balance \* 0.01;

UPDATE Accounts

SET Balance = Balance + v\_interest,

LastModified = CURRENT\_DATE

WHERE AccountID = rec.AccountID;

RAISE NOTICE 'Interest added to AccountID %: +%', rec.AccountID, v\_interest;

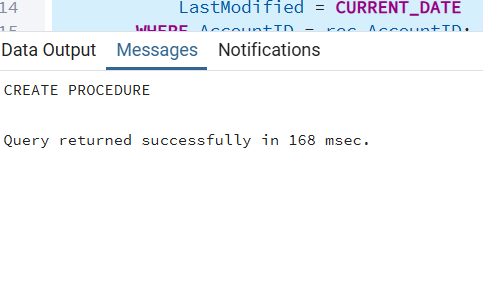
END LOOP;

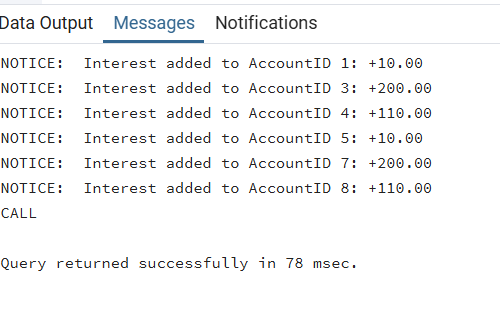
END;

$$;

CALL ProcessMonthlyInterest();

**OUTPUT:**





**Scenario 2:**

1. Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**CODE:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept\_name VARCHAR,

bonus\_percent NUMERIC

)

LANGUAGE plpgsql

AS

$$

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonus\_percent / 100),

HireDate = HireDate -- Just to trigger LastModified if needed

WHERE Department = dept\_name;

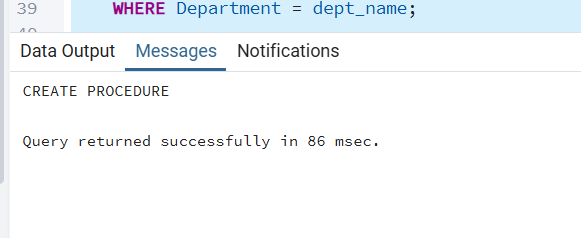
RAISE NOTICE 'Bonus of %%% applied to department: %', bonus\_percent, dept\_name;

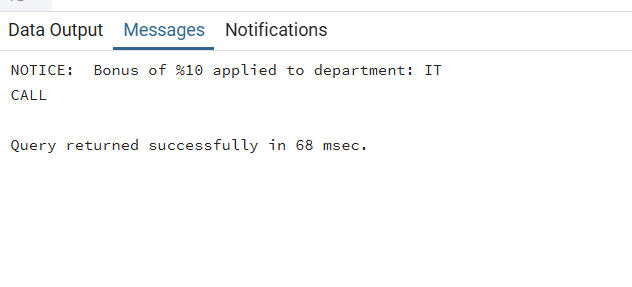
END;

$$;

CALL UpdateEmployeeBonus('IT', 10);

**OUTPUT:**

****

****

**Scenario 3:**

**3.** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

**CODE:**

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_acc INT,

to\_acc INT,

amt NUMERIC

)

LANGUAGE plpgsql

AS

$$

DECLARE

from\_balance NUMERIC;

BEGIN

SELECT Balance INTO from\_balance FROM Accounts WHERE AccountID = from\_acc;

IF from\_balance < amt THEN

RAISE NOTICE 'Insufficient balance in account %', from\_acc;

RETURN;

END IF;

UPDATE Accounts

SET Balance = Balance - amt,

LastModified = CURRENT\_DATE

WHERE AccountID = from\_acc;

UPDATE Accounts

SET Balance = Balance + amt,

LastModified = CURRENT\_DATE

WHERE AccountID = to\_acc;

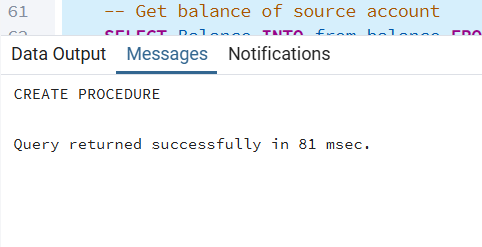
RAISE NOTICE 'Transferred % from Account % to Account %', amt, from\_acc, to\_acc;

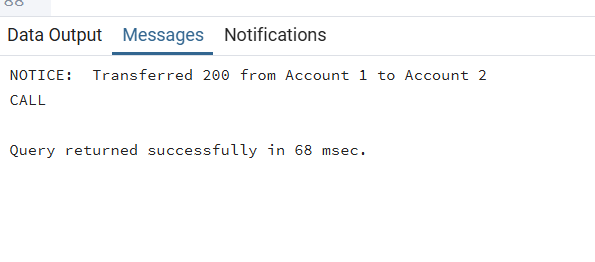
END;

$$;

CALL TransferFunds(1, 2, 200);

**OUTPUT:**

****

****