**1. Control Structures**

**Scenario 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.

**Query:**

BEGIN

FOR cust IN (SELECT customer\_id, Age FROM Customers) LOOP

IF cust.age > 60 THEN

UPDATE Loans

SET interest\_rate = interest\_rate - 1

WHERE customer\_id = cust.customer\_id;

END IF;

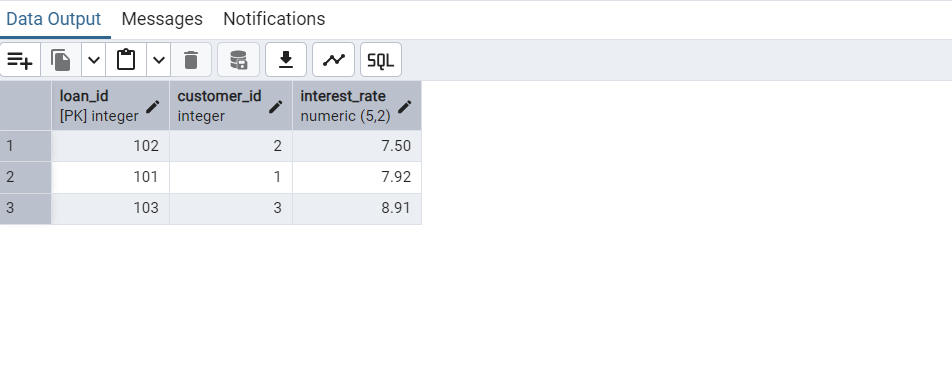
END LOOP;

COMMIT;

END;

SELECT \* FROM loans;

**Output:**

****

**Scenario 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.

**Query:**

BEGIN

FOR cust IN SELECT customer\_id, balance FROM customers LOOP

IF cust.balance > 10000 THEN

UPDATE customers

SET is\_vip = TRUE

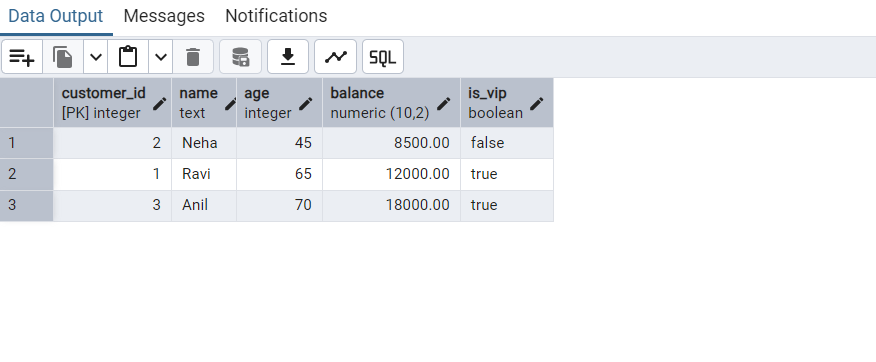
WHERE customer\_id = cust.customer\_id;

END IF;

END LOOP;

END ;

SELECT \* FROM customers;

**Output:**

**Scenario 3:**

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

**Query:**

DECLARE

v\_due\_date DATE := SYSDATE + 30;

BEGIN

FOR loan\_rec IN (

SELECT LoanID, CustomerID, DueDate

FROM Loans

WHERE DueDate <= v\_due\_date

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Customer ' || loan\_rec.CustomerID ||

', your loan (ID: ' || loan\_rec.LoanID ||

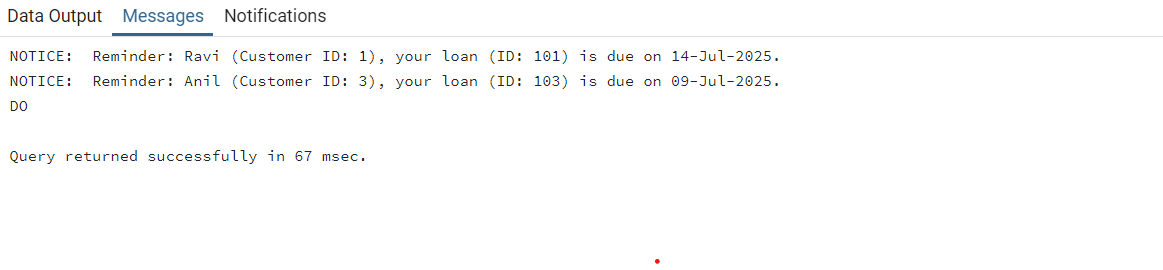
') is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY') || '.'

);

END LOOP;

END;

**Output:**

****

**2. Stored Procedures:**

**Scenario 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.

**Query:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

SET Balance = Balance + (acc.Balance \* 0.01)

WHERE AccountID = acc.AccountID;

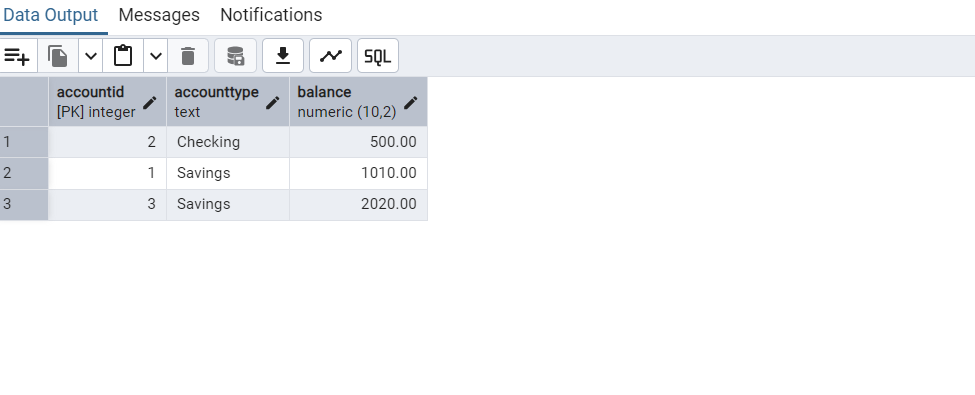
END LOOP;

COMMIT;

END;

SELECT \* FROM Accounts;

**Output:**



**Scenario 2:**

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

**Query:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_DepartmentID IN NUMBER,

p\_BonusPercent IN NUMBER

) AS

BEGIN

FOR emp IN (

SELECT EmployeeID, Salary

FROM Employees

WHERE DepartmentID = p\_DepartmentID

) LOOP

UPDATE Employees

SET Salary = Salary + (emp.Salary \* p\_BonusPercent / 100)

WHERE EmployeeID = emp.EmployeeID;

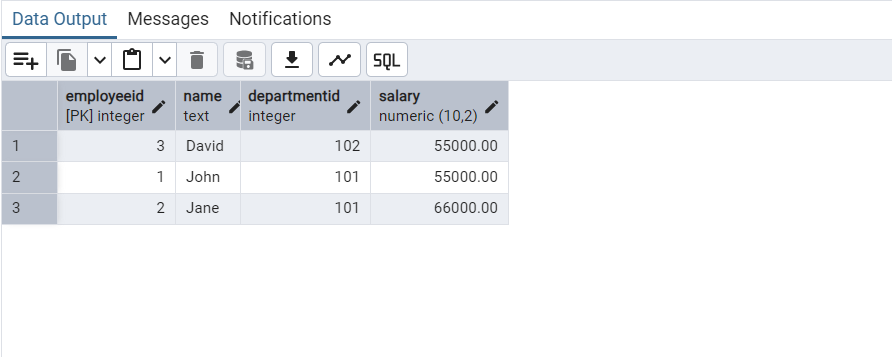
END LOOP;

COMMIT;

END;

SELECT \* FROM Employees;

**Output:**



**Scenario 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.

**Query:**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) AS

v\_Balance NUMBER;

BEGIN

-- Fetch balance of source account

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = p\_FromAccountID;

-- Check if balance is sufficient

IF v\_Balance >= p\_Amount THEN

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

END IF;

END;

SELECT \* FROM Accounts;

**Output:**

