**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**Question:** 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.

🡪CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Age NUMBER,

InterestRate NUMBER

);

🡪INSERT INTO Customers VALUES (1, 'Ram', 45, 8.5);

INSERT INTO Customers VALUES (2, 'Sita', 65, 9.0);

INSERT INTO Customers VALUES (3, 'Laxman', 70, 10.0);

COMMIT;

🡪select \* from customers;



🡪BEGIN

FOR customer\_rec IN (

SELECT CustomerID, Age, InterestRate FROM Customers

) LOOP

IF customer\_rec.Age > 60 THEN

UPDATE Customers

SET InterestRate = InterestRate - 1

WHERE CustomerID = customer\_rec.CustomerID;

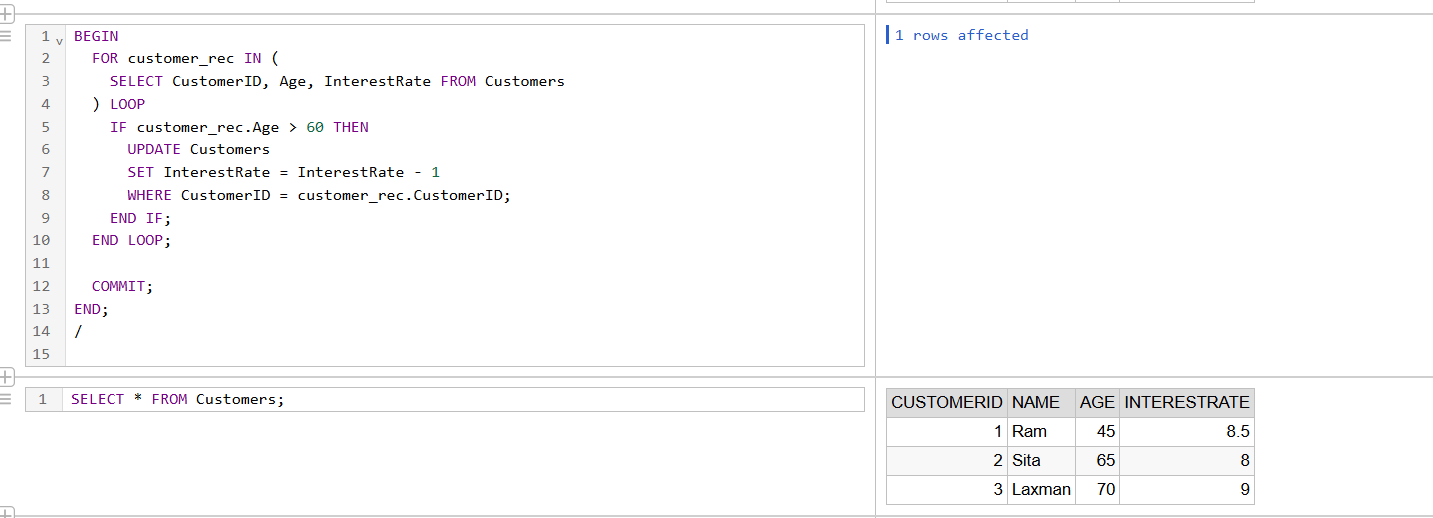
END IF;

END LOOP;

COMMIT;END;/

🡪SELECT \* FROM Customers;

**Output:**

****

**Scenario 2:** A customer can be promoted to VIP status based on their balance.

**Question:** 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.

🡪ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

ALTER TABLE Customers ADD Balance NUMBER;

🡪UPDATE Customers SET Balance = 15000 WHERE CustomerID = 2;

UPDATE Customers SET Balance = 8000 WHERE CustomerID = 1;

UPDATE Customers SET Balance = 12000 WHERE CustomerID = 3;

COMMIT;

🡪BEGIN

FOR customer\_rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF customer\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = customer\_rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;/

🡪SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

**Output:**

****

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

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

🡪CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

🡪INSERT INTO Loans VALUES (101, 1, SYSDATE + 10); -- Due in 10 days

INSERT INTO Loans VALUES (102, 2, SYSDATE + 35); -- Not due in next 30 days

INSERT INTO Loans VALUES (103, 3, SYSDATE + 5); -- Due in 5 days

COMMIT;

🡪BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, l.CustomerID, c.Name, l.DueDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Dear ' || loan\_rec.Name ||

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

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

);

END LOOP;

END;

/

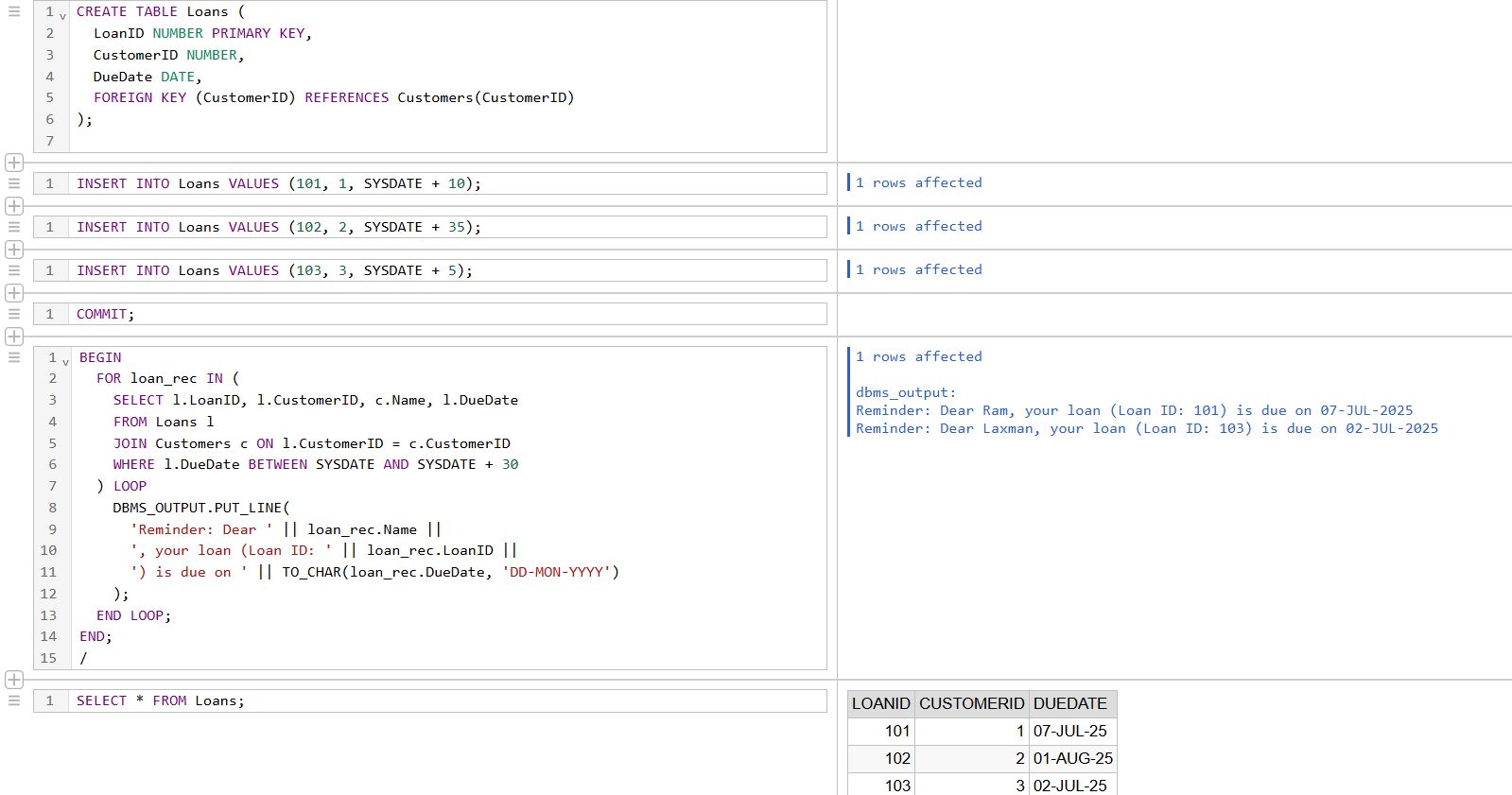
**Output:**

**Reminder: Dear Ram, your loan (Loan ID: 101) is due on 07-JUL-2025**

**Reminder: Dear Laxman, your loan (Loan ID: 103) is due on 02-JUL-2025**

🡪SELECT \* FROM Loans;

**Output:**

****

**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

**Question:** 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.

**🡪**ALTER TABLE Customers ADD AccountType VARCHAR2(20);

🡪UPDATE Customers SET AccountType = 'Savings', Balance = 10000 WHERE CustomerID = 1;

UPDATE Customers SET AccountType = 'Current', Balance = 20000 WHERE CustomerID = 2;

UPDATE Customers SET AccountType = 'Savings', Balance = 15000 WHERE CustomerID = 3;

COMMIT;

🡪CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR cust\_rec IN (

SELECT CustomerID, Balance

FROM Customers

WHERE AccountType = 'Savings'

) LOOP

UPDATE Customers

SET Balance = Balance + (Balance \* 0.01)

WHERE CustomerID = cust\_rec.CustomerID;

END LOOP;

COMMIT;

END;

/

🡪BEGIN

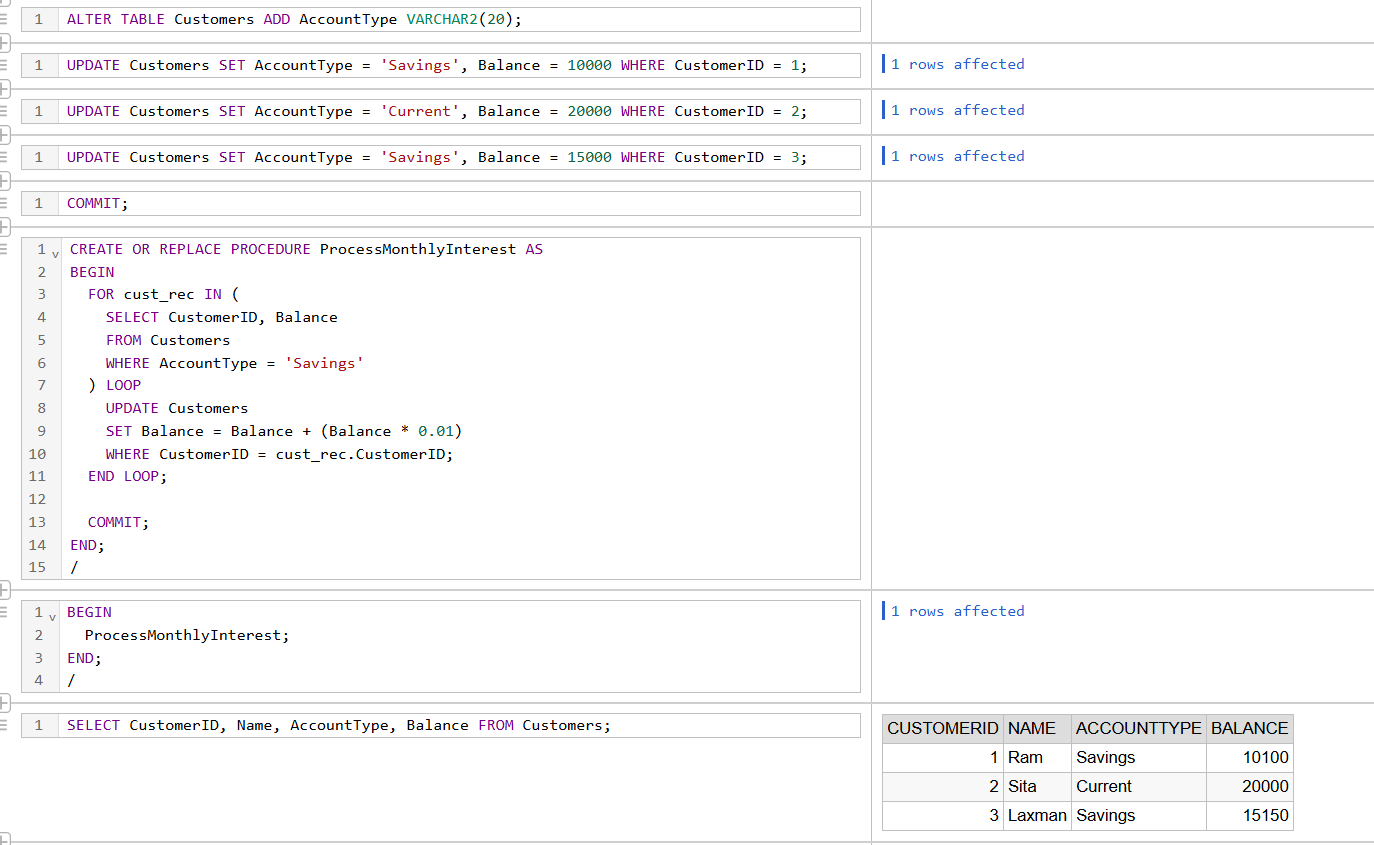
ProcessMonthlyInterest;

END;

/

🡪SELECT CustomerID, Name, AccountType, Balance FROM Customers

**Output:**

****

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

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

🡪CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Department VARCHAR2(50),

Salary NUMBER

);

🡪INSERT INTO Employees VALUES (1, 'Anjali', 'HR', 50000);

INSERT INTO Employees VALUES (2, 'Raj', 'IT', 60000);

INSERT INTO Employees VALUES (3, 'Meena', 'HR', 55000);

INSERT INTO Employees VALUES (4, 'Karan', 'Finance', 70000);

COMMIT;

🡪select \* from Employees;



🡪CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE Department = p\_Department;

COMMIT;

END;

/

🡪BEGIN

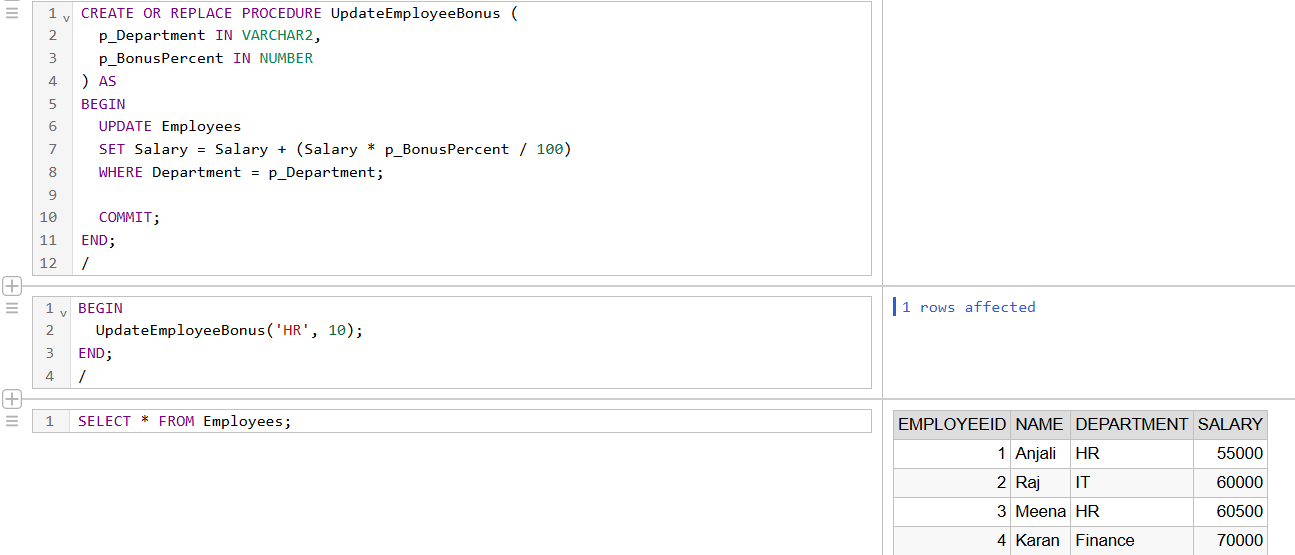
UpdateEmployeeBonus('HR', 10);

END;

/

🡪SELECT \* FROM Employees;

**Output:**

****

**Scenario 3:** Customers should be able to transfer funds between their accounts.

**Question:** 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.

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromCustomerID IN NUMBER,

p\_ToCustomerID IN NUMBER,

p\_Amount IN NUMBER

) AS

v\_FromBalance NUMBER;

BEGIN

SELECT Balance INTO v\_FromBalance

FROM Customers

WHERE CustomerID = p\_FromCustomerID;

IF v\_FromBalance < p\_Amount THEN

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

END IF;

UPDATE Customers

SET Balance = Balance - p\_Amount

WHERE CustomerID = p\_FromCustomerID;

UPDATE Customers

SET Balance = Balance + p\_Amount

WHERE CustomerID = p\_ToCustomerID;

COMMIT;

END;

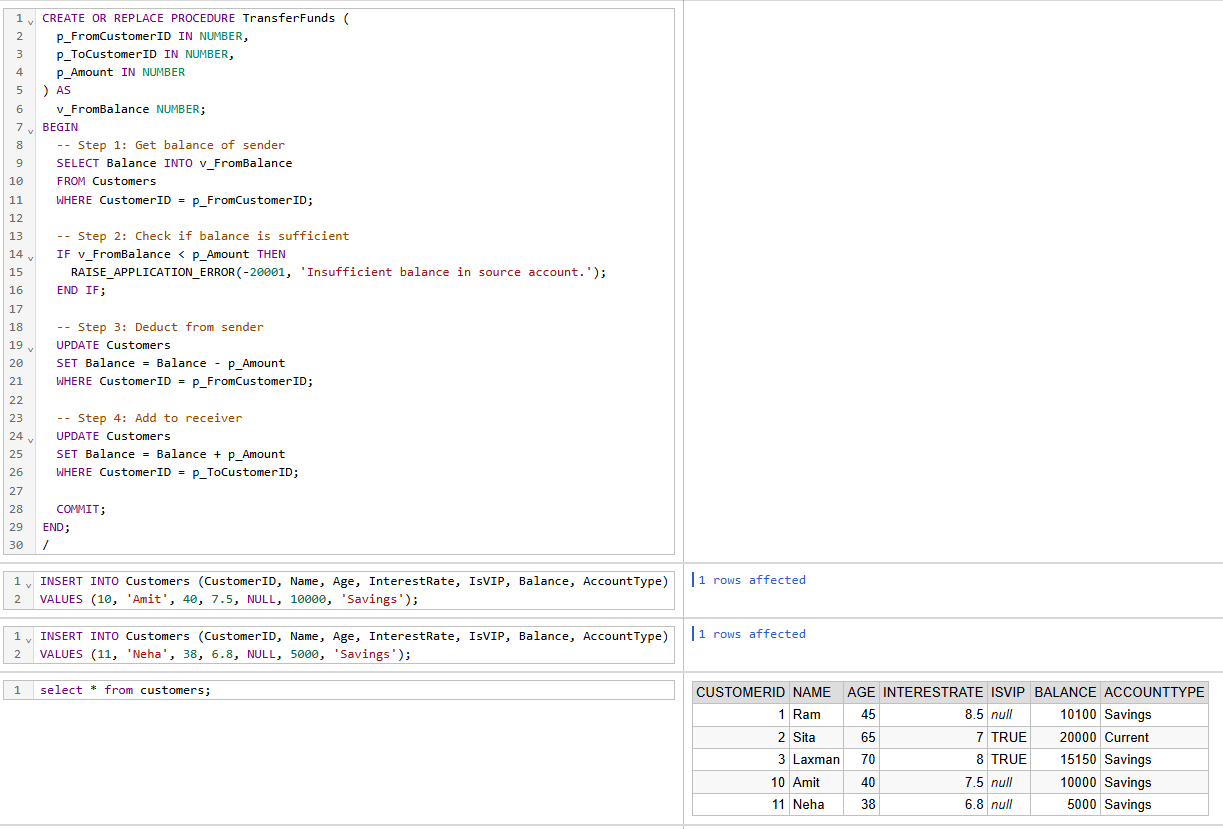
/

INSERT INTO Customers (CustomerID, Name, Age, InterestRate, IsVIP, Balance, AccountType)

VALUES (10, 'Amit', 40, 7.5, NULL, 10000, 'Savings');

INSERT INTO Customers (CustomerID, Name, Age, InterestRate, IsVIP, Balance, AccountType)

VALUES (11, 'Neha', 38, 6.8, NULL, 5000, 'Savings');



BEGIN

TransferFunds(10, 11, 2000);

END;

/

SELECT CustomerID, Name, Balance FROM Customers;

**Output:**

