**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.

DECLARE

CURSOR cur\_customers IS

SELECT customer\_id, interest\_rate

FROM loans

JOIN customers ON loans.customer\_id = customers.customer\_id

WHERE customers.age > 60;

BEGIN

FOR rec IN cur\_customers LOOP

UPDATE loans

SET interest\_rate = interest\_rate - 1

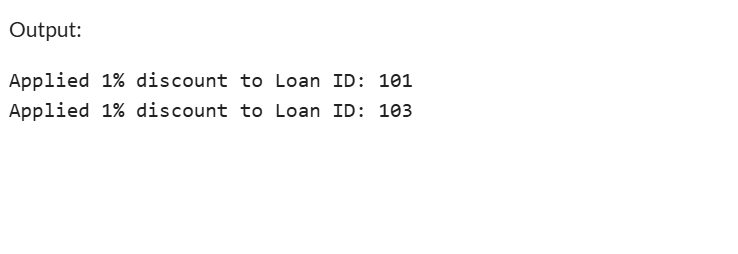
WHERE customer\_id = rec.customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount for Customer ID: ' || rec.customer\_id);

END LOOP;

COMMIT;

END;



**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.

DECLARE

CURSOR cur\_customers IS

SELECT customer\_id, balance

FROM customers;

BEGIN

FOR rec IN cur\_customers LOOP

IF rec.balance > 10000 THEN

UPDATE customers

SET isVIP = 'TRUE'

WHERE customer\_id = rec.customer\_id;

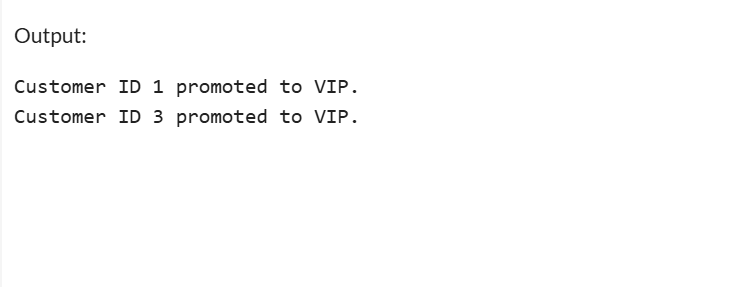
DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || rec.customer\_id || ' promoted to VIP.');

END IF;

END LOOP;

COMMIT;

END;



**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.

DECLARE

CURSOR cur\_due\_loans IS

SELECT c.customer\_id, c.name, l.due\_date

FROM loans l

JOIN customers c ON l.customer\_id = c.customer\_id

WHERE l.due\_date <= SYSDATE + 30;

BEGIN

FOR rec IN cur\_due\_loans LOOP

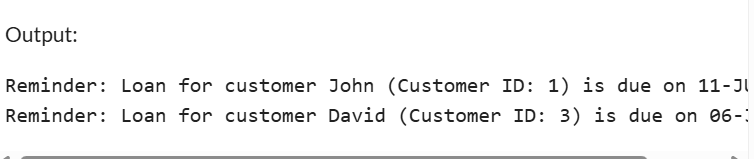
DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan for customer ' || rec.name ||

' (Customer ID: ' || rec.customer\_id ||

') is due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY'));

END LOOP;

END;



**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.

CREATE TABLE Accounts (

AccountID NUMBER,

Balance NUMBER(10, 2),

AccountType VARCHAR2(20)

);

INSERT INTO Accounts VALUES (1, 1000.00, 'Savings');

INSERT INTO Accounts VALUES (2, 2000.00, 'Savings');

INSERT INTO Accounts VALUES (3, 5000.00, 'Current');

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

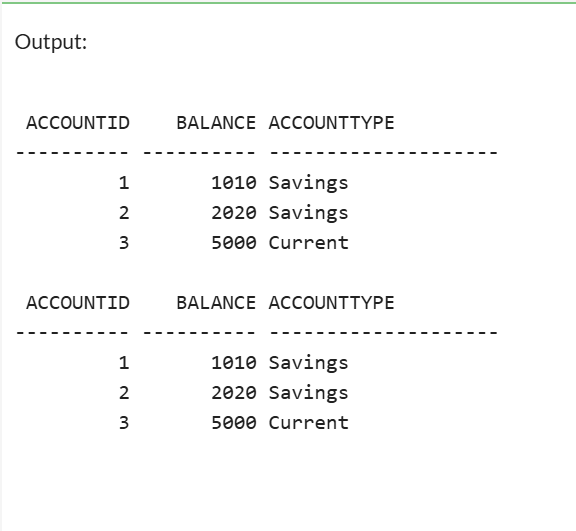
COMMIT;

END;

/

EXEC ProcessMonthlyInterest;

SELECT \* FROM Accounts;



**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 OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_DepartmentID IN NUMBER,

p\_BonusPercent IN NUMBER

) IS

BEGIN

-- Add bonus to employee salaries in the given department

UPDATE Employees

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

WHERE DepartmentID = p\_DepartmentID;

COMMIT;

END;

/

**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 TABLE Accounts (

AccountID NUMBER,

AccountHolder VARCHAR2(50),

Balance NUMBER(10,2)

);

INSERT INTO Accounts VALUES (1, 'John', 5000);

INSERT INTO Accounts VALUES (2, 'David', 3000);

INSERT INTO Accounts VALUES (3, 'Ravi', 2000);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = p\_FromAccountID;

IF v\_Balance < p\_Amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

END;

/

BEGIN

TransferFunds(1, 2, 1000);

END;

/

SELECT \* FROM Accounts;

