**Prerequisites: Setting Up Your Environment**

Before running the PL/SQL blocks, you need to have some tables and data to work with. You can use the following SQL statements to create the necessary Customers, Loans, and Accounts tables.

**Run these CREATE TABLE and INSERT statements first:**

SQL

-- Create the Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

DateOfBirth DATE

);

-- Create the Loans table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER REFERENCES Customers(CustomerID),

InterestRate NUMBER(5, 2),

DueDate DATE

);

-- Create the Accounts table and add a VIP flag to Customers

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER REFERENCES Customers(CustomerID),

Balance NUMBER(12, 2)

);

-- Add the IsVIP column to the Customers table for Scenario 2

ALTER TABLE Customers ADD (IsVIP VARCHAR2(5) DEFAULT 'FALSE');

-- Insert sample data

-- Customer over 60

INSERT INTO Customers (CustomerID, CustomerName, DateOfBirth) VALUES (101, 'John Smith', DATE '1955-05-15');

-- Customer under 60

INSERT INTO Customers (CustomerID, CustomerName, DateOfBirth) VALUES (102, 'Jane Doe', DATE '1992-08-20');

-- Customer over 60

INSERT INTO Customers (CustomerID, CustomerName, DateOfBirth) VALUES (103, 'Peter Jones', DATE '1948-11-30');

-- Customer with a high balance

INSERT INTO Customers (CustomerID, CustomerName, DateOfBirth) VALUES (104, 'Mary Williams', DATE '1985-01-25');

-- Insert sample loans

INSERT INTO Loans (LoanID, CustomerID, InterestRate, DueDate) VALUES (1, 101, 5.50, DATE '2025-10-15');

INSERT INTO Loans (LoanID, CustomerID, InterestRate, DueDate) VALUES (2, 102, 4.75, DATE '2026-01-20');

INSERT INTO Loans (LoanID, CustomerID, InterestRate, DueDate) VALUES (3, 103, 6.00, DATE '2025-11-01');

-- Loan due within 30 days from today (assuming today is around late June 2025)

INSERT INTO Loans (LoanID, CustomerID, InterestRate, DueDate) VALUES (4, 102, 4.50, SYSDATE + 15);

-- Another loan due soon

INSERT INTO Loans (LoanID, CustomerID, InterestRate, DueDate) VALUES (5, 104, 3.90, SYSDATE + 25);

-- Insert sample account balances

INSERT INTO Accounts (AccountID, CustomerID, Balance) VALUES (901, 101, 5000.00);

INSERT INTO Accounts (AccountID, CustomerID, Balance) VALUES (902, 102, 8500.00);

INSERT INTO Accounts (AccountID, CustomerID, Balance) VALUES (903, 103, 12500.00); -- High balance customer

INSERT INTO Accounts (AccountID, CustomerID, Balance) VALUES (904, 104, 25000.00); -- High balance customer

COMMIT;

**Exercise 1: Solutions**

**Scenario 1: Discount for Customers Over 60**

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

**Solution:**

This block uses a CURSOR to loop through each customer. It calculates their age and, if it's over 60, it runs an UPDATE statement to reduce the interest rate on their loans by 1% (multiplying by 0.99).

SQL

-- Run this command once per session to see the output messages

SET SERVEROUTPUT ON;

DECLARE

-- Declare a cursor to select all customers

CURSOR customer\_cursor IS

SELECT CustomerID, DateOfBirth

FROM Customers;

v\_customer\_id Customers.CustomerID%TYPE;

v\_dob Customers.DateOfBirth%TYPE;

v\_age NUMBER;

v\_rows\_updated NUMBER;

BEGIN

OPEN customer\_cursor;

LOOP

-- Fetch customer data into variables

FETCH customer\_cursor INTO v\_customer\_id, v\_dob;

EXIT WHEN customer\_cursor%NOTFOUND; -- Exit loop when no more customers

-- Calculate age in years

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, v\_dob) / 12);

-- Check if the customer's age is greater than 60

IF v\_age > 60 THEN

-- Apply a 1% discount (new rate = old rate \* 0.99)

UPDATE Loans

SET InterestRate = InterestRate \* 0.99

WHERE CustomerID = v\_customer\_id;

-- Get the number of loans updated for this customer

v\_rows\_updated := SQL%ROWCOUNT;

-- Print a confirmation message

IF v\_rows\_updated > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount for CustomerID: ' || v\_customer\_id || ' (Age: ' || v\_age || '). Updated ' || v\_rows\_updated || ' loan(s).');

END IF;

END IF;

END LOOP;

CLOSE customer\_cursor;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Discount process completed.');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

ROLLBACK; -- Undo changes if an error happens

END;

/

**Scenario 2: Promote Customer to VIP Status**

**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.

**Solution:**

This block iterates through each account. If an account's balance is over $10,000, it updates the IsVIP flag on the corresponding customer's record.

SQL

-- Run this command once per session to see the output messages

SET SERVEROUTPUT ON;

DECLARE

-- Cursor to get customers and their total balance

CURSOR account\_cursor IS

SELECT CustomerID, SUM(Balance) AS TotalBalance

FROM Accounts

GROUP BY CustomerID;

BEGIN

-- A FOR loop is a more concise way to handle cursors

FOR rec IN account\_cursor LOOP

-- Check if the total balance exceeds the VIP threshold

IF rec.TotalBalance > 10000 THEN

-- Update the flag in the Customers table

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CustomerID || ' promoted to VIP status.');

END IF;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('VIP status check completed.');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

ROLLBACK;

END;

/

**Alternative (and more efficient) Solution:**

For this specific task, a single SQL statement is much more efficient than a PL/SQL loop because it processes the data in a set rather than row-by-row.

SQL

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID IN (

SELECT CustomerID

FROM Accounts

GROUP BY CustomerID

HAVING SUM(Balance) > 10000

);

COMMIT;

**Scenario 3: Loan Due Date Reminders**

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

**Solution:**

This solution uses a CURSOR with a JOIN to get both the customer's name and the loan's due date. The WHERE clause filters for loans due between today (SYSDATE) and 30 days from now (SYSDATE + 30). The FOR loop makes iterating and printing the results clean and simple.

SQL

-- Run this command once per session to see the output messages

SET SERVEROUTPUT ON;

DECLARE

-- Cursor to select loans due in the next 30 days, joining with Customers to get the name

CURSOR due\_loans\_cursor IS

SELECT

c.CustomerName,

l.DueDate,

l.LoanID

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Sending Loan Reminders ---');

-- Loop through all records found by the cursor

FOR loan\_rec IN due\_loans\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder for: ' || loan\_rec.CustomerName ||

'. Your Loan #' || loan\_rec.LoanID ||

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

);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('--- End of Reminders ---');

END;

/