**Superset ID: 6394725**

**PL/SQL programming**

**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.

**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.

**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.

**Solution:**

For implementing the above scenario, first we to create the tables and insert values into the table

**Step 1: Create Tables**

* Customer Table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Age NUMBER,

Balance NUMBER,

IsVIP CHAR(1) DEFAULT 'N'

);

* Loans Table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

InterestRate NUMBER(5,2),

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

**Step 2: Insert values into table**

* Customer table

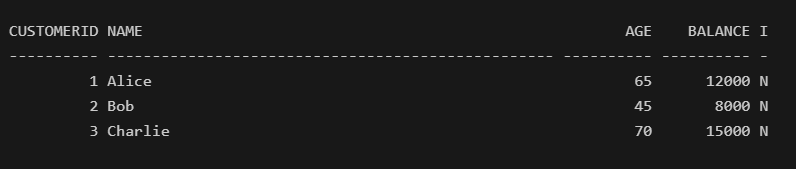
-- Insert sample customers

INSERT INTO Customers VALUES (1, 'Alice', 65, 12000, 'N');

INSERT INTO Customers VALUES (2, 'Bob', 45, 8000, 'N');

INSERT INTO Customers VALUES (3, 'Charlie', 70, 15000, 'N');

Output:



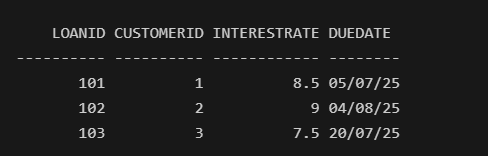
* Loans table

INSERT INTO Loans VALUES (101, 1, 8.5, SYSDATE + 10);

INSERT INTO Loans VALUES (102, 2, 9.0, SYSDATE + 40);

INSERT INTO Loans VALUES (103, 3, 7.5, SYSDATE + 25);

Output:



**Step 3: Implement the scenarios**

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

-- Run the update block

BEGIN

FOR cust IN (SELECT CustomerID FROM Customers WHERE Age > 60) LOOP

UPDATE Loans

SET InterestRate = InterestRate - (InterestRate \* 0.01)

WHERE CustomerID = cust.CustomerID;

END LOOP;

COMMIT;

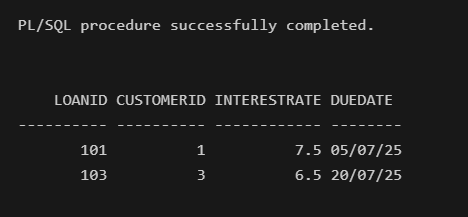
END;

/

-- Then see updated values

SELECT \* FROM Loans WHERE CustomerID IN (SELECT CustomerID FROM Customers WHERE Age > 60);

Output:



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

BEGIN

FOR cust IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

END LOOP;

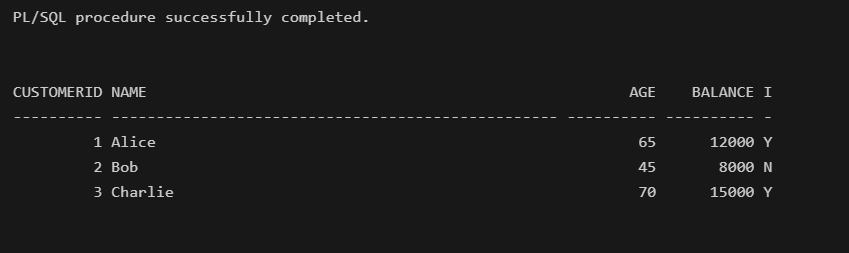
COMMIT;

END;

/

SELECT \* FROM Customers;

**Output:**



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

BEGIN

FOR loan\_rec IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || loan\_rec.Name ||

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

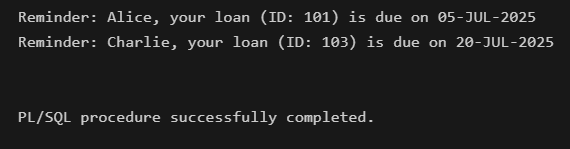
') is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

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.

**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.

**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.

**Solution:**

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

**Step-1:** Create and insert values into the table “SavingsAccounts”.

CREATE TABLE SavingsAccounts (

account\_id NUMBER PRIMARY KEY,

balance NUMBER

);

INSERT INTO SavingsAccounts VALUES (101, 10000);

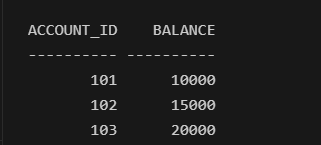
INSERT INTO SavingsAccounts VALUES (102, 15000);

INSERT INTO SavingsAccounts VALUES (103, 20000);

COMMIT;

select \* from SavingsAccounts;

Output:



**Step-2 :** Create & Execute Procedure for above Scenario

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE SavingsAccounts

SET balance = balance + (balance \* 0.01); -- 1% interest

COMMIT;

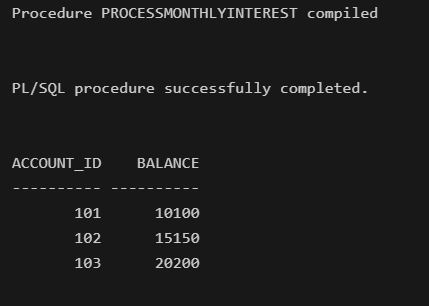
END;

/

EXEC ProcessMonthlyInterest;

SELECT \* FROM SavingsAccounts;

Output :



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

**Step-1:** Create and insert values into the table “Employees”.

CREATE TABLE Employees (

emp\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

salary NUMBER,

department\_id NUMBER

);

INSERT INTO Employees VALUES (1, 'Alice', 50000, 101);

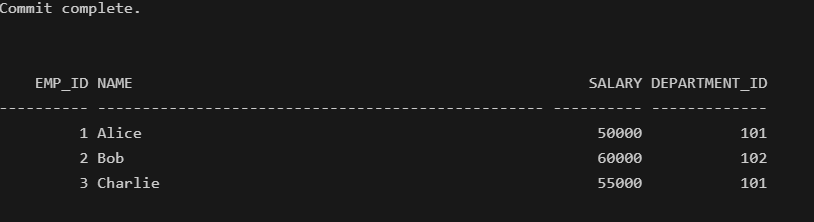
INSERT INTO Employees VALUES (2, 'Bob', 60000, 102);

INSERT INTO Employees VALUES (3, 'Charlie', 55000, 101);

COMMIT;

SELECT \* FROM Employees;

Output:



**Step-2 :** Create & Execute Procedure for above Scenario

BEGIN

EXECUTE IMMEDIATE '

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_dept\_id IN NUMBER,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET salary = salary + (salary \* p\_bonus\_percent / 100)

WHERE department\_id = p\_dept\_id;

COMMIT;

END;';

END;

/

BEGIN

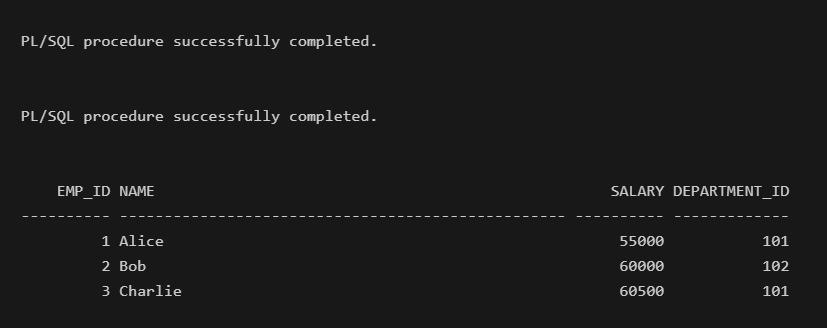
UpdateEmployeeBonus(101, 10);

END;

/

SELECT \* FROM Employees;

Output:



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

**Step-1:** Create and insert values into the table “Employees”.

CREATE TABLE Accounts (

account\_id NUMBER PRIMARY KEY,

balance NUMBER

);

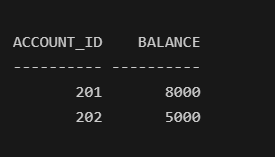
INSERT INTO Accounts VALUES (201, 8000);

INSERT INTO Accounts VALUES (202, 5000);

COMMIT;

SELECT \* FROM Accounts;

Output :



**Step-2 :** Create & Execute Procedure for above Scenario

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_source\_account IN NUMBER,

p\_target\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Get balance of source account

SELECT balance INTO v\_balance

FROM Accounts

WHERE account\_id = p\_source\_account;

IF v\_balance < p\_amount THEN

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

END IF;

-- Deduct from source account

UPDATE Accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_source\_account;

-- Add to target account

UPDATE Accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_target\_account;

COMMIT;

END;

/

EXEC TransferFunds(201, 202, 2000);

SELECT \* FROM Accounts;

Output :

