**Digital Nurture 4.0 – Week 2**

**1.PL SQL Programming**

**Exercise 1: Control Structures**

**Program:**

*Scenario 1: Discount for Senior Citizens*

This PL/SQL block retrieves all customers from the table. It checks each customer's age in a loop. If a customer is older than 60, it applies a 1% discount to their loan interest rate by updating the value. The update is done using control structures like IF within a loop. Finally, the changes are saved using COMMIT.

**PL/SQL Code**:

CREATE TABLE Customers (

CustomerID NUMBER,

Age NUMBER,

InterestRate NUMBER(5,2),

Balance NUMBER,

IsVIP VARCHAR2(5)

);

INSERT INTO Customers VALUES (101, 65, 10.5, 15000, 'FALSE');

INSERT INTO Customers VALUES (102, 45, 12.0, 8000, 'FALSE');

INSERT INTO Customers VALUES (103, 70, 11.0, 12000, 'FALSE');

INSERT INTO Customers VALUES (104, 30, 9.5, 5000, 'FALSE');

COMMIT;

BEGIN

FOR rec IN (SELECT CustomerID, Age, InterestRate FROM Customers) LOOP

IF rec.Age > 60 THEN

UPDATE Customers

SET InterestRate = InterestRate - 1

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to Customer ID: ' || rec.CustomerID);

END IF;

END LOOP;

COMMIT;

END;

/

BEGIN

FOR rec IN (SELECT \* FROM Customers) LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || rec.CustomerID ||

', Age: ' || rec.Age ||

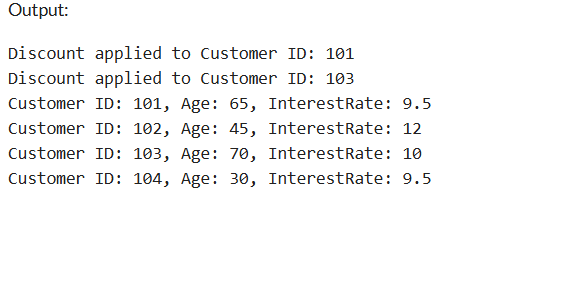
', InterestRate: ' || rec.InterestRate);

END LOOP;

END;

/

**Output:**

****

*Scenario 2: Set VIP Status*

This PL/SQL block checks all customers' account balances from the Customers table. It uses a loop to go through each customer record. If a customer's balance is greater than $10,000, their IsVIP flag is updated to 'TRUE'. A message is printed for each VIP promotion using DBMS\_OUTPUT.PUT\_LINE. Finally, all updates are saved to the database using COMMIT.

**PL/SQL CODE:**

CREATE TABLE Customers (

CustomerID NUMBER,

Age NUMBER,

InterestRate NUMBER(5,2),

Balance NUMBER,

IsVIP VARCHAR2(5)

);

INSERT INTO Customers VALUES (201, 40, 10.0, 9500, 'FALSE');

INSERT INTO Customers VALUES (202, 55, 9.0, 12000, 'FALSE');

INSERT INTO Customers VALUES (203, 67, 11.0, 20000, 'FALSE');

INSERT INTO Customers VALUES (204, 33, 8.5, 5000, 'FALSE');

COMMIT;

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

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

END IF;

END LOOP;

COMMIT;

END;

/

BEGIN

FOR rec IN (SELECT \* FROM Customers) LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || rec.CustomerID ||

', Balance: ' || rec.Balance ||

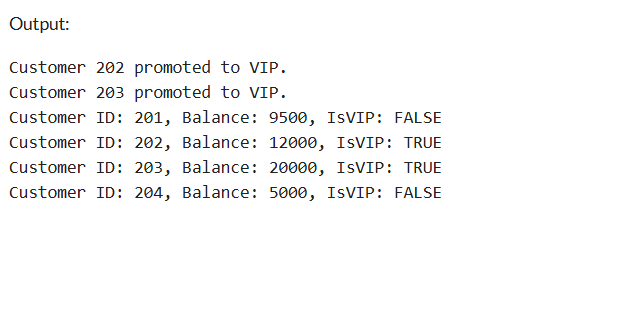
', IsVIP: ' || rec.IsVIP);

END LOOP;

END;

/

**Output:**

****

*Scenario 3: Loan Due Reminder*

This PL/SQL block retrieves all loans that are due within the next 30 days using a date comparison with SYSDATE. It uses a loop to go through each loan that meets the condition. For each record, it prints a reminder message including the Loan ID, Customer ID, and due date. This helps simulate how a bank would notify customers. Since no data is modified, there’s no need to commit changes.

**PL/SQL CODE:**

CREATE TABLE Loans (

LoanID NUMBER,

CustomerID NUMBER,

DueDate DATE

);

INSERT INTO Loans VALUES (301, 201, SYSDATE + 10);

INSERT INTO Loans VALUES (302, 202, SYSDATE + 35);

INSERT INTO Loans VALUES (303, 203, SYSDATE + 5);

INSERT INTO Loans VALUES (304, 204, SYSDATE + 60);

COMMIT;

BEGIN

FOR rec IN (

SELECT LoanID, CustomerID, DueDate

FROM Loans

WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID ||

' for Customer ' || rec.CustomerID ||

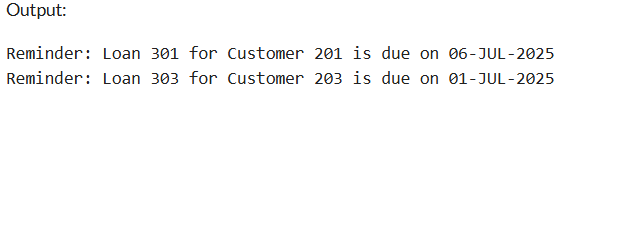
' is due on ' || TO\_CHAR(rec.DueDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Output:**



**Exercise 3: Stored Procedures**

Program:

*Scenario 1: Process Monthly Interest*

This stored procedure calculates interest for all savings accounts. It loops through each record in the Accounts table where the AccountType is 'SAVINGS'. It increases the balance by 1% of the current amount. It uses a cursor to access each record and applies the interest. Finally, it updates the table and commits the transaction.

**PL/SQL CODE:**

CREATE TABLE Accounts (

AccountID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER

);

INSERT INTO Accounts VALUES (1001, 'SAVINGS', 10000);

INSERT INTO Accounts VALUES (1002, 'CURRENT', 15000);

INSERT INTO Accounts VALUES (1003, 'SAVINGS', 20000);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

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;

/

BEGIN

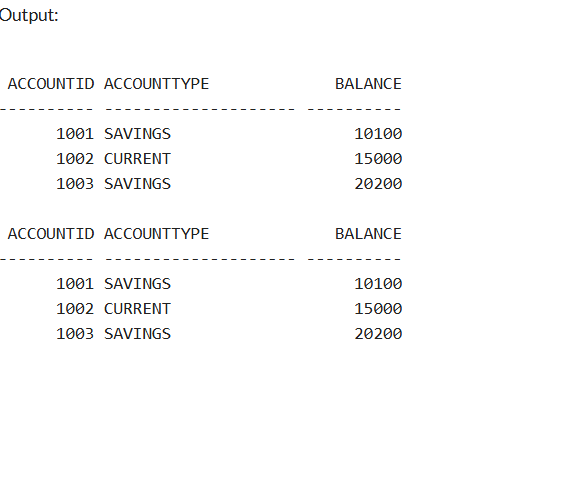
ProcessMonthlyInterest;

END;

/

SELECT \* FROM Accounts;

**Output:**



*Scenario 2: Update Employee Bonus*

This procedure gives a salary bonus to employees in a specific department. It takes two input parameters: the department name and bonus percentage. It updates all employee salaries in that department by increasing them based on the bonus percentage. The procedure is reusable for any department and percentage. A COMMIT finalizes the changes.

**PL/SQL CODE:**

CREATE TABLE Employees (

EmpID NUMBER,

Name VARCHAR2(50),

Department VARCHAR2(30),

Salary NUMBER

);

INSERT INTO Employees VALUES (1, 'Anu', 'HR', 30000);

INSERT INTO Employees VALUES (2, 'Ravi', 'IT', 40000);

INSERT INTO Employees VALUES (3, 'Meena', 'HR', 35000);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept IN VARCHAR2,

bonusPercent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonusPercent / 100)

WHERE Department = dept;

COMMIT;

END;

/

BEGIN

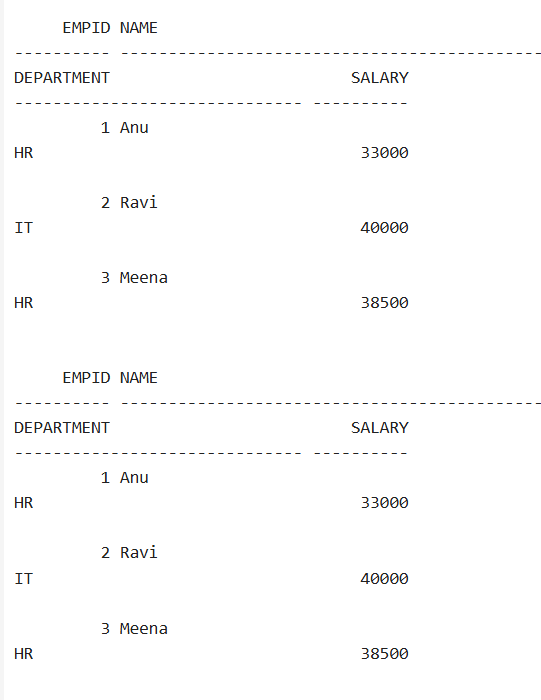
UpdateEmployeeBonus('HR', 10);

END;

/

SELECT \* FROM Employees;

**Output:**



*Scenario 3: Transfer Funds*

This enhanced version of TransferFunds first checks whether both source and destination accounts exist. It then verifies that the source account has enough funds. If all conditions are met, it debits the source and credits the destination. If not, it prints a message instead of raising an error. This is safer and more informative for user-driven apps*.*

**PL/SQL CODE:**

DROP TABLE Accounts;

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

AccountType VARCHAR2(20),

Balance NUMBER

);

INSERT INTO Accounts VALUES (3001, 'SAVINGS', 7000);

INSERT INTO Accounts VALUES (3002, 'SAVINGS', 2500);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

fromAcc IN NUMBER,

toAcc IN NUMBER,

amt IN NUMBER

) IS

fromBalance NUMBER;

toExists NUMBER;

BEGIN

SELECT Balance INTO fromBalance FROM Accounts WHERE AccountID = fromAcc;

SELECT COUNT(\*) INTO toExists FROM Accounts WHERE AccountID = toAcc;

IF toExists = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Destination account ' || toAcc || ' does not exist.');

RETURN;

END IF;

IF fromBalance < amt THEN

DBMS\_OUTPUT.PUT\_LINE('Insufficient funds in account ' || fromAcc);

RETURN;

END IF;

UPDATE Accounts SET Balance = Balance - amt WHERE AccountID = fromAcc;

UPDATE Accounts SET Balance = Balance + amt WHERE AccountID = toAcc;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('₹' || amt || ' transferred from ' || fromAcc || ' to ' || toAcc);

END;

/

BEGIN

TransferFunds(3001, 3002, 2000);

END;

/

SELECT \* FROM Accounts;

**Output:**

