Digital Nurture 4.0 – Week 2

**(i)PL/SQL programming**

**Creating Tables:**

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    AccountType VARCHAR2(20),

    Balance NUMBER,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID NUMBER PRIMARY KEY,

    AccountID NUMBER,

    TransactionDate DATE,

    Amount NUMBER,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID));

CREATE TABLE Loans (

    LoanID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    LoanAmount NUMBER,

    InterestRate NUMBER,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

**Insert Values Into The Tables:**

INSERT INTO Customers VALUES (101, 'Hari', TO\_DATE('1955-08-10', 'YYYY-MM-DD'), 18000, SYSDATE);

INSERT INTO Customers VALUES (102, 'Aasadhu Amedhu', TO\_DATE('1988-02-05', 'YYYY-MM-DD'), 9500, SYSDATE);

INSERT INTO Accounts VALUES (201, 101, 'Savings', 4000, SYSDATE);

INSERT INTO Accounts VALUES (202, 102, 'Current', 6000, SYSDATE);

INSERT INTO Transactions VALUES (301, 201, SYSDATE, 1000, 'Deposit');

INSERT INTO Transactions VALUES (302, 202, SYSDATE, 1500, 'Withdrawal');

INSERT INTO Loans VALUES (401, 101, 7000, 6.5, SYSDATE, ADD\_MONTHS(SYSDATE, 5));

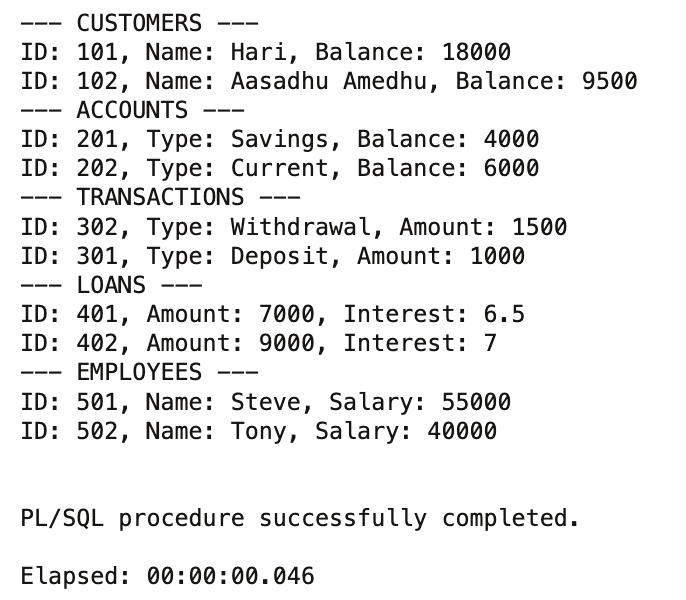
INSERT INTO Loans VALUES (402, 102, 9000, 7, SYSDATE, SYSDATE + 10); -- due in 10 days

INSERT INTO Employees VALUES (501, 'Steve', 'Analyst', 55000, 'Finance', TO\_DATE('2020-01-15', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (502, 'Tony', 'Clerk', 40000, 'Operations', TO\_DATE('2018-03-20', 'YYYY-MM-DD'));

COMMIT;

**Output:**



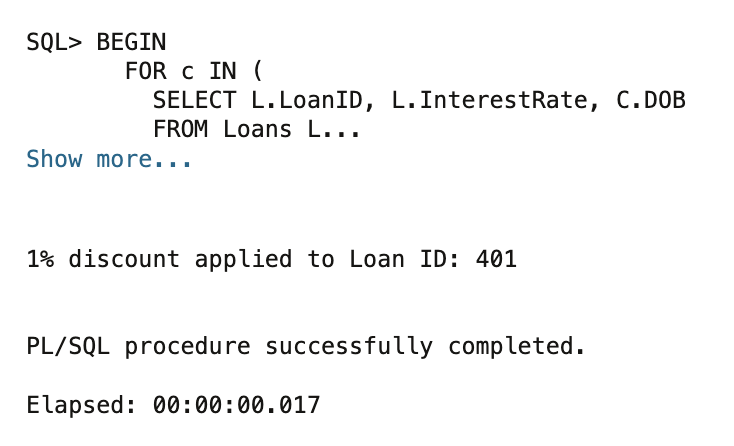
Exercise 1: Control Structures

## Scenario 1: Age-Based Loan Discount:

If a customer is older than 60 years, their loan interest rate is reduced by 1%. We calculate age using MONTHS\_BETWEEN and apply the discount accordingly.

BEGIN  
 FOR c IN (  
 SELECT L.LoanID, L.InterestRate, C.DOB  
 FROM Loans L  
 JOIN Customers C ON L.CustomerID = C.CustomerID  
 ) LOOP  
 IF FLOOR(MONTHS\_BETWEEN(SYSDATE, c.DOB) / 12) > 60 THEN  
 UPDATE Loans  
 SET InterestRate = InterestRate - 1  
 WHERE LoanID = c.LoanID;  
 DBMS\_OUTPUT.PUT\_LINE('1% discount applied to Loan ID: ' || c.LoanID);  
 END IF;  
 END LOOP;  
END;  
/

**Output:**

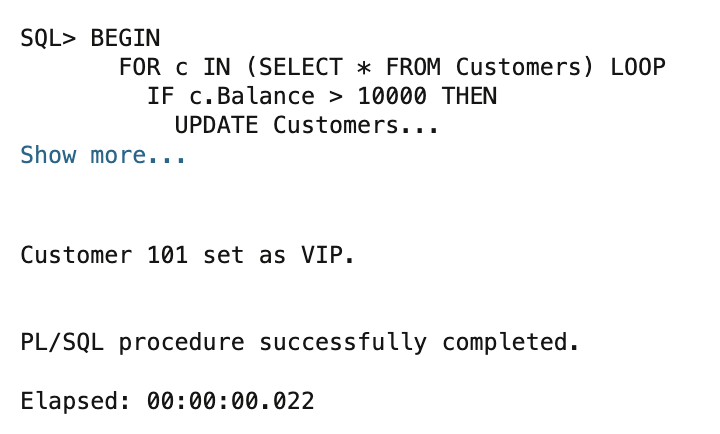


## Scenario 2: VIP Flag Based on Balance:

If a customer has a balance over $10,000, they are flagged as VIP.

BEGIN  
 FOR c IN (SELECT \* FROM Customers) LOOP  
 IF c.Balance > 10000 THEN  
 UPDATE Customers  
 SET IsVIP = 'TRUE'  
 WHERE CustomerID = c.CustomerID;  
 DBMS\_OUTPUT.PUT\_LINE('Customer ' || c.CustomerID || ' set as VIP.');  
 ELSE  
 UPDATE Customers  
 SET IsVIP = 'FALSE'  
 WHERE CustomerID = c.CustomerID;  
 END IF;  
 END LOOP;  
END;  
/

**Output:**

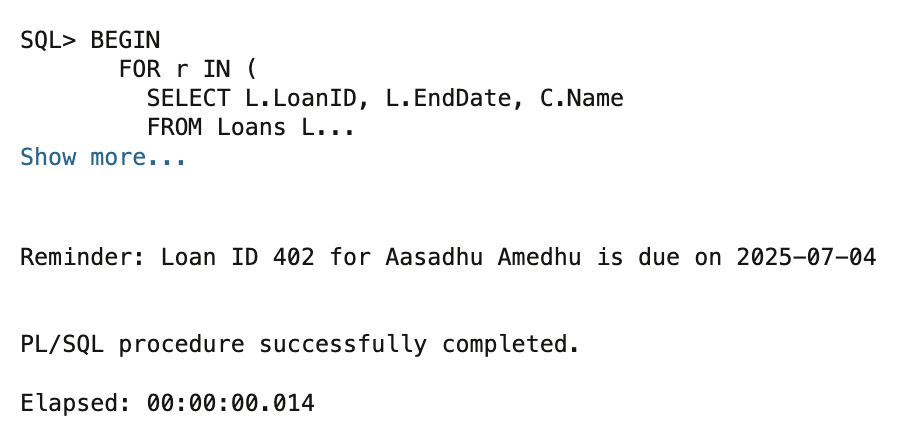


## Scenario 3: Loan Due Reminders:

Loans due within 30 days trigger a reminder.

BEGIN  
 FOR r IN (  
 SELECT L.LoanID, L.EndDate, C.Name  
 FROM Loans L  
 JOIN Customers C ON L.CustomerID = C.CustomerID  
 WHERE L.EndDate BETWEEN SYSDATE AND SYSDATE + 30  
 ) LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || r.LoanID ||   
 ' for ' || r.Name ||   
 ' is due on ' || TO\_CHAR(r.EndDate, 'YYYY-MM-DD'));  
 END LOOP;  
END;  
/

**Output:**



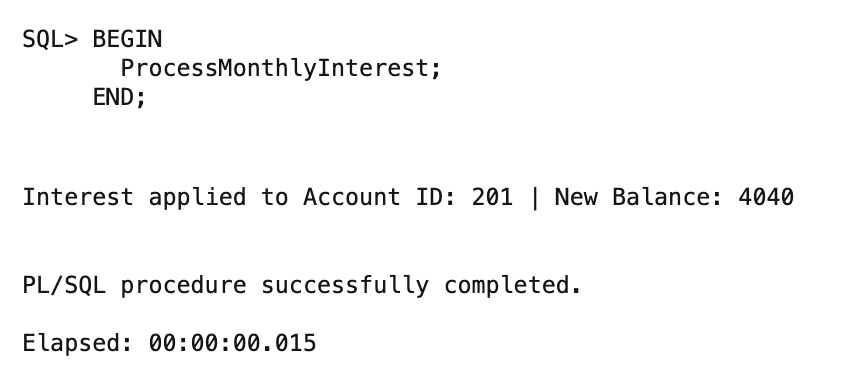
Exercise 3: Stored Procedures

## Scenario 1: Process Monthly Interest:

Applies 1% monthly interest to all savings accounts.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  
BEGIN  
 FOR acc IN (  
 SELECT AccountID, Balance  
 FROM Accounts  
 WHERE AccountType = 'Savings'  
 ) LOOP  
 UPDATE Accounts  
 SET Balance = Balance + (Balance \* 0.01)  
 WHERE AccountID = acc.AccountID;  
  
 DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || acc.AccountID ||  
 ' | New Balance: ' || TO\_CHAR(acc.Balance + (acc.Balance \* 0.01)));  
 END LOOP;  
END;  
/

**Output:**

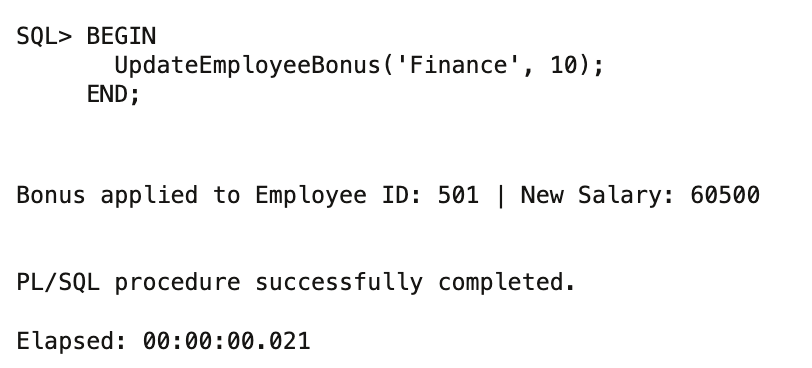


## Scenario 2: Update Employee Bonus:

Increases salary of employees in a department by a bonus percentage.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (  
 p\_department IN VARCHAR2,  
 p\_bonus\_percent IN NUMBER  
) IS  
BEGIN  
 FOR emp IN (  
 SELECT EmployeeID, Salary  
 FROM Employees  
 WHERE Department = p\_department  
 ) LOOP  
 UPDATE Employees  
 SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)  
 WHERE EmployeeID = emp.EmployeeID;  
  
 DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID: ' || emp.EmployeeID ||  
 ' | New Salary: ' || TO\_CHAR(emp.Salary + (emp.Salary \* p\_bonus\_percent / 100)));  
 END LOOP;  
END;  
/

**Output:**



## Scenario 3: Transfer Funds Between Accounts:

Transfers funds between accounts if sufficient balance exists.

CREATE OR REPLACE PROCEDURE TransferFunds (  
 p\_from\_account IN NUMBER,  
 p\_to\_account IN NUMBER,  
 p\_amount IN NUMBER  
) IS  
 v\_balance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_balance  
 FROM Accounts  
 WHERE AccountID = p\_from\_account  
 FOR UPDATE;  
  
 IF v\_balance >= p\_amount THEN  
 UPDATE Accounts  
 SET Balance = Balance - p\_amount  
 WHERE AccountID = p\_from\_account;  
  
 UPDATE Accounts  
 SET Balance = Balance + p\_amount  
 WHERE AccountID = p\_to\_account;  
  
 DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_amount ||   
 ' from Account ' || p\_from\_account ||   
 ' to Account ' || p\_to\_account);  
 ELSE  
 DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in Account ID: ' || p\_from\_account);  
 END IF;  
END;  
/

**Output:**

