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));

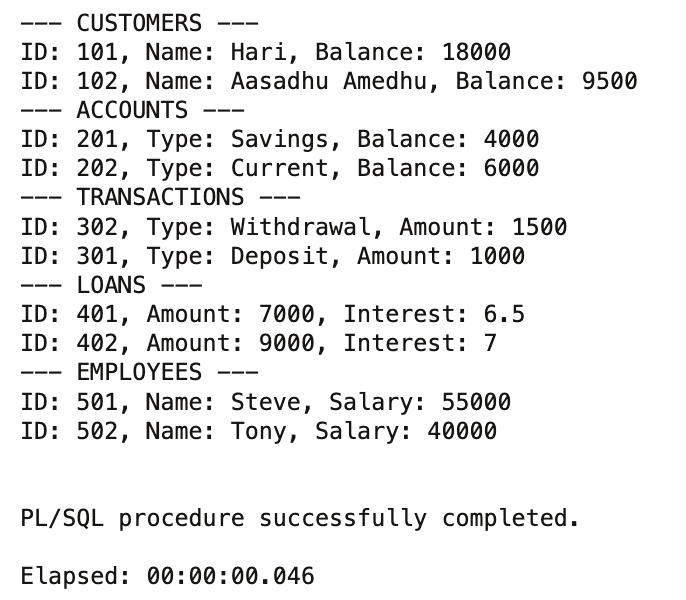
INSERT INTO Loans VALUES (402, 102, 9000, 7, SYSDATE, SYSDATE + 10); -- due in 10 days

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:

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

BEGIN

FOR rec IN (

SELECT l.LoanID, l.InterestRate, c.DOB

FROM Loans l

INNER JOIN Customers c ON c.CustomerID = l.CustomerID

) LOOP

IF TRUNC(MONTHS\_BETWEEN(SYSDATE, rec.DOB)/12) > 60 THEN

UPDATE Loans

SET InterestRate = rec.InterestRate - 1

WHERE LoanID = rec.LoanID;

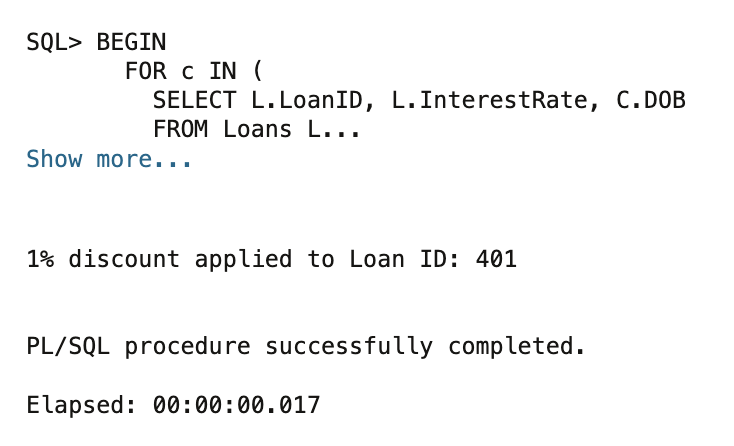
DBMS\_OUTPUT.PUT\_LINE('Interest adjusted for Loan #' || rec.LoanID);

END IF;

END LOOP;

END;

/

**Output:**

## Scenario 2: VIP Flag Based on Balance:

BEGIN

FOR data IN (

SELECT CustomerID, Balance FROM Customers

) LOOP

UPDATE Customers

SET IsVIP = CASE

WHEN data.Balance > 10000 THEN 'TRUE'

ELSE 'FALSE'

END

WHERE CustomerID = data.CustomerID;

IF data.Balance > 10000 THEN

DBMS\_OUTPUT.PUT\_LINE('VIP tagged: Customer ' || data.CustomerID);

END IF;

END LOOP;

END;

/

**Output:**

## 

## Scenario 3: Loan Due Reminders:

BEGIN

FOR due IN (

SELECT l.LoanID, l.EndDate, c.Name

FROM Loans l

JOIN Customers c ON c.CustomerID = l.CustomerID

WHERE l.EndDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('⚠️ Reminder: ' || due.Name ||

', your Loan #' || due.LoanID ||

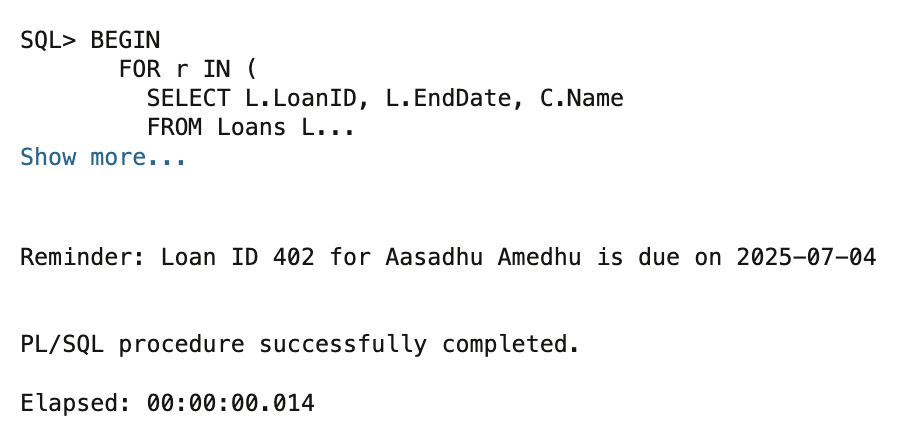
' is due on ' || TO\_CHAR(due.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Output:**

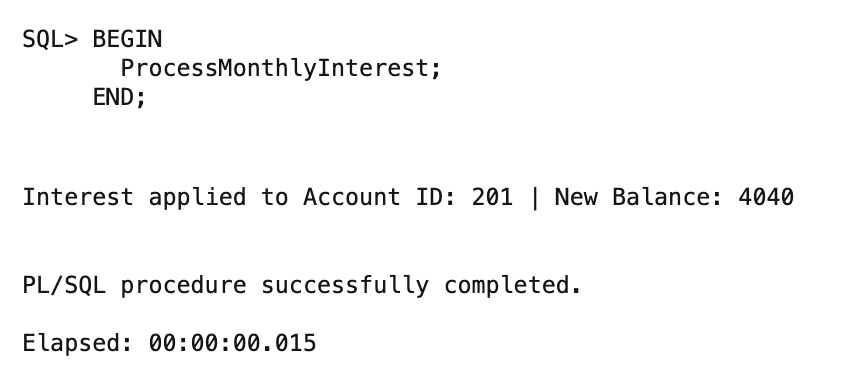


Exercise 3: Stored Procedures

## Scenario 1: Process Monthly Interest:

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:

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:**

A screenshot of a computer screen

AI-generated content may be incorrect.

## Scenario 3: Transfer Funds Between Accounts:

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_acc\_id IN NUMBER,

to\_acc\_id IN NUMBER,

amount\_val IN NUMBER

) IS

current\_balance NUMBER;

BEGIN

-- Lock the source account row to prevent concurrent changes

SELECT Balance INTO current\_balance

FROM Accounts

WHERE AccountID = from\_acc\_id

FOR UPDATE;

-- Check for sufficient funds

IF current\_balance >= amount\_val THEN

-- Deduct from sender

UPDATE Accounts

SET Balance = Balance - amount\_val

WHERE AccountID = from\_acc\_id;

-- Add to receiver

UPDATE Accounts

SET Balance = Balance + amount\_val

WHERE AccountID = to\_acc\_id;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount\_val ||

' from Account ' || from\_acc\_id ||

' to Account ' || to\_acc\_id);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient funds in Account ID: ' || from\_acc\_id);

END IF;

END;

/**Output:**

A screenshot of a computer screen

AI-generated content may be incorrect.