WEEK-2

PLSQL\_EXERCISES

**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.

CODE

TABLES CREATION AND VALUES INSERTION

*CREATE TABLE Customers (*

*CustomerID NUMBER PRIMARY KEY,*

*Name VARCHAR2(100),*

*DOB DATE,*

*Balance NUMBER,*

*LastModified DATE*

*);*

*CREATE TABLE Loans (*

*LoanID NUMBER PRIMARY KEY,*

*CustomerID NUMBER,*

*LoanAmount NUMBER,*

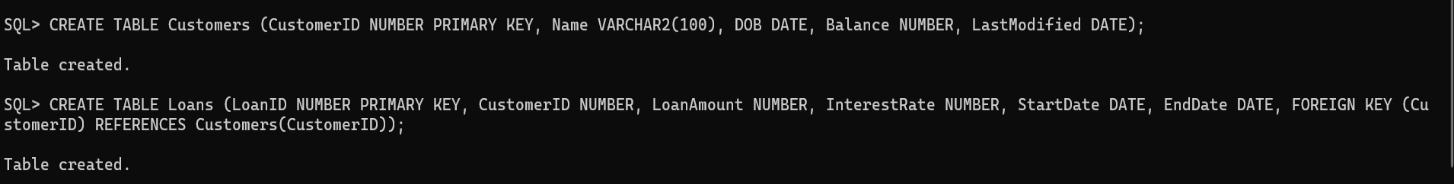
*InterestRate NUMBER,*

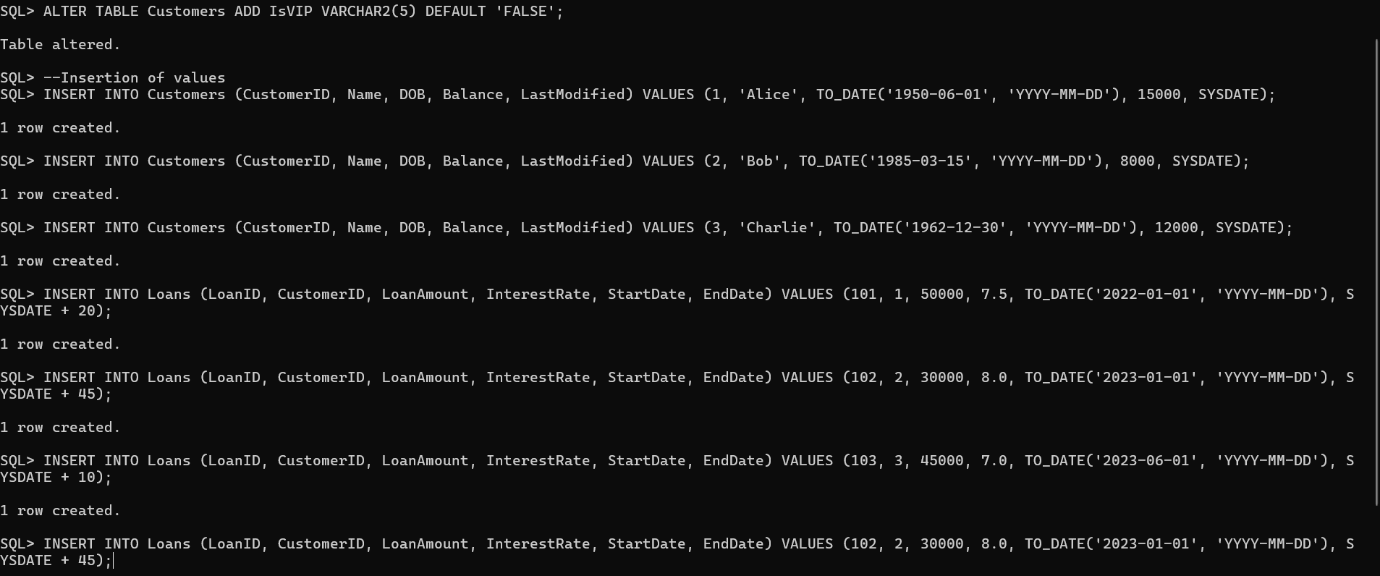
*StartDate DATE,*

*EndDate DATE,*

*FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)*

*);*

**

**

**SCENARIO 1**

*SET SERVEROUTPUT ON;*

*BEGIN*

*FOR cust\_rec IN (*

*SELECT c.CustomerID, l.LoanID, l.InterestRate,*

*FLOOR(MONTHS\_BETWEEN(SYSDATE, c.DOB)/12) AS Age*

*FROM Customers c*

*JOIN Loans l ON c.CustomerID = l.CustomerID*

*)*

*LOOP*

*IF cust\_rec.Age > 60 THEN*

*UPDATE Loans*

*SET InterestRate = InterestRate - 1*

*WHERE LoanID = cust\_rec.LoanID;*

*DBMS\_OUTPUT.PUT\_LINE('Discount applied to LoanID: ' || cust\_rec.LoanID || ' (Customer Age: ' || cust\_rec.Age || ')');*

*ELSE*

*DBMS\_OUTPUT.PUT\_LINE('No discount for LoanID: ' || cust\_rec.LoanID || ' (Customer Age: ' || cust\_rec.Age || ')');*

*END IF;*

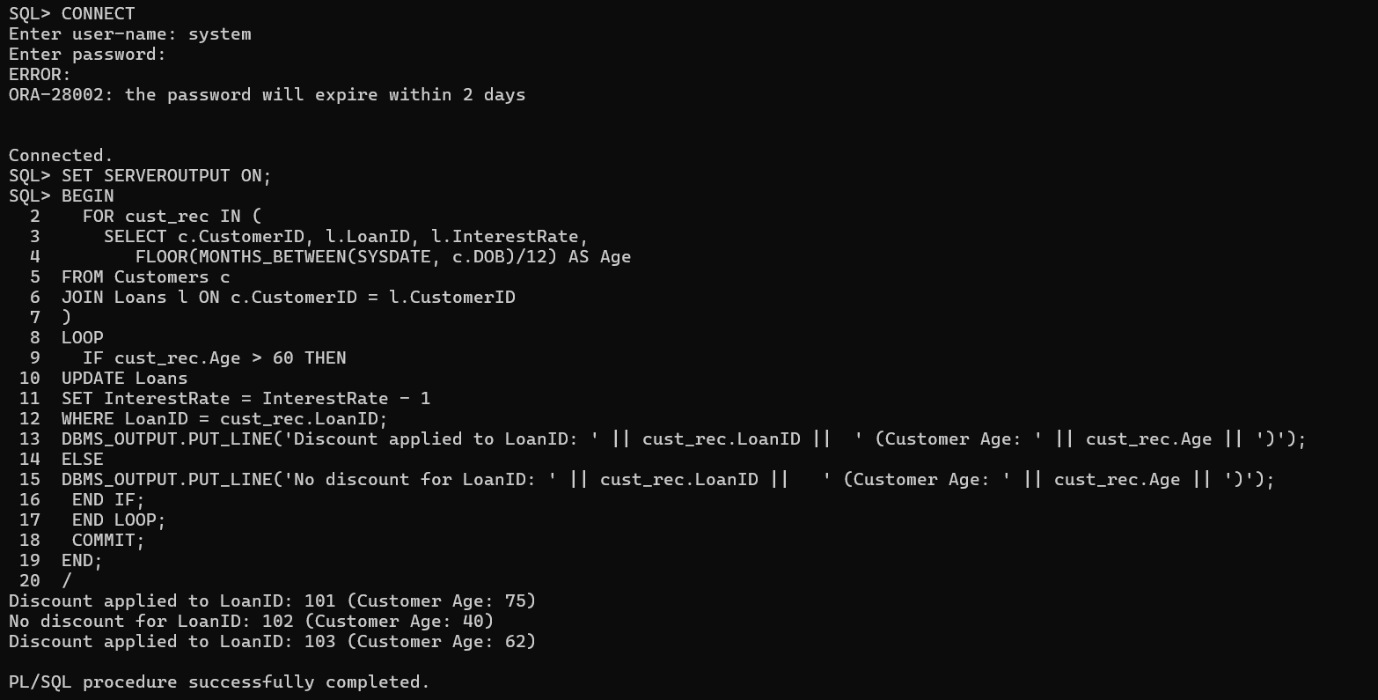
*END LOOP;*

*COMMIT;*

*END;*

*/*

OUTPUT

**

**SCENARIO 2**

*BEGIN*

*FOR cust\_rec IN (SELECT CustomerID, Balance FROM Customers) LOOP*

*IF cust\_rec.Balance > 10000 THEN*

*UPDATE Customers*

*SET IsVIP = 'TRUE'*

*WHERE CustomerID = cust\_rec.CustomerID;*

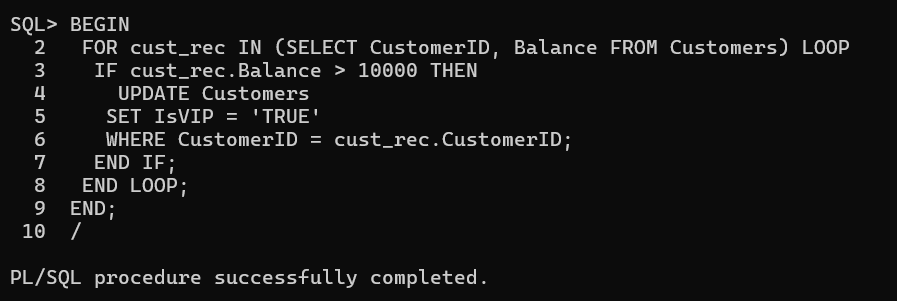
*END IF;*

*END LOOP;*

*END;*

*/*

OUTPUT

**

**SCENARIO 3**

*BEGIN*

*FOR loan\_rec IN (*

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

*FROM Loans l*

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

*WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30 ) LOOP*

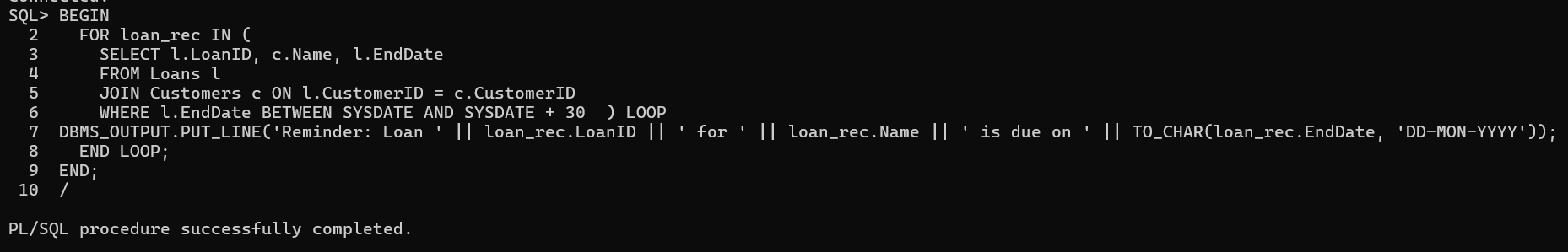
*DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || loan\_rec.LoanID || ' for ' || loan\_rec.Name || ' is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-MON-YYYY'));*

*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.

CODE

*CREATE TABLE Employees (*

*EmployeeID NUMBER PRIMARY KEY,*

*Name VARCHAR2(100),*

*Position VARCHAR2(50),*

*Salary NUMBER,*

*Department VARCHAR2(50),*

*HireDate DATE*

*);*

*CREATE TABLE Accounts (*

*AccountID NUMBER PRIMARY KEY,*

*CustomerID NUMBER,*

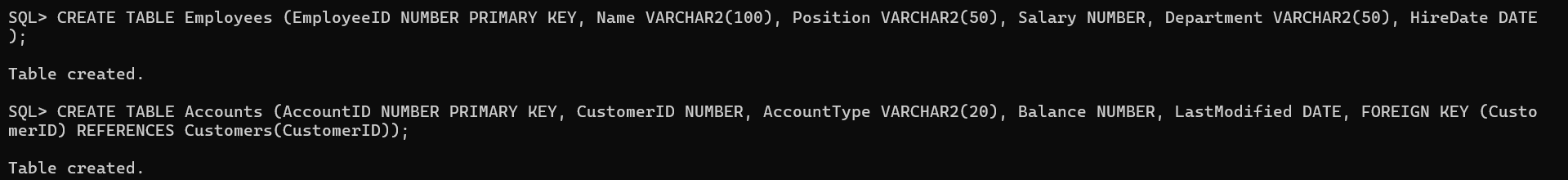
*AccountType VARCHAR2(20),*

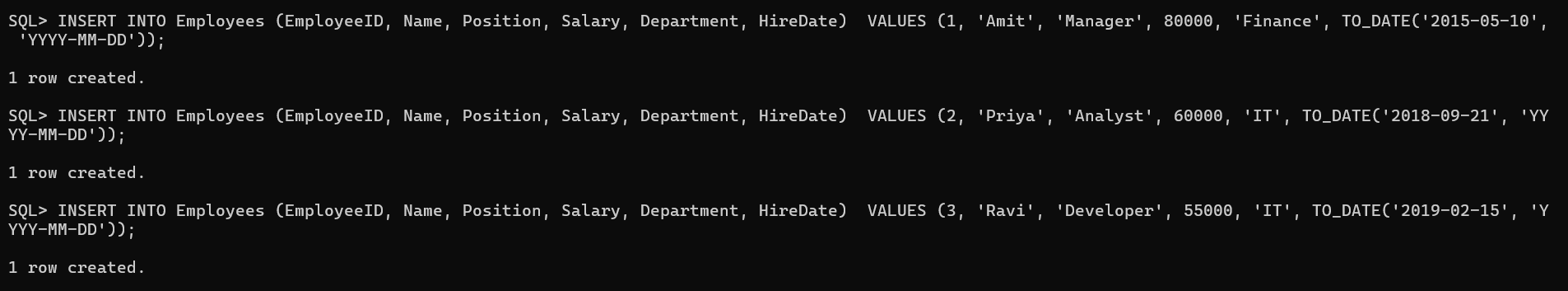
*Balance NUMBER,*

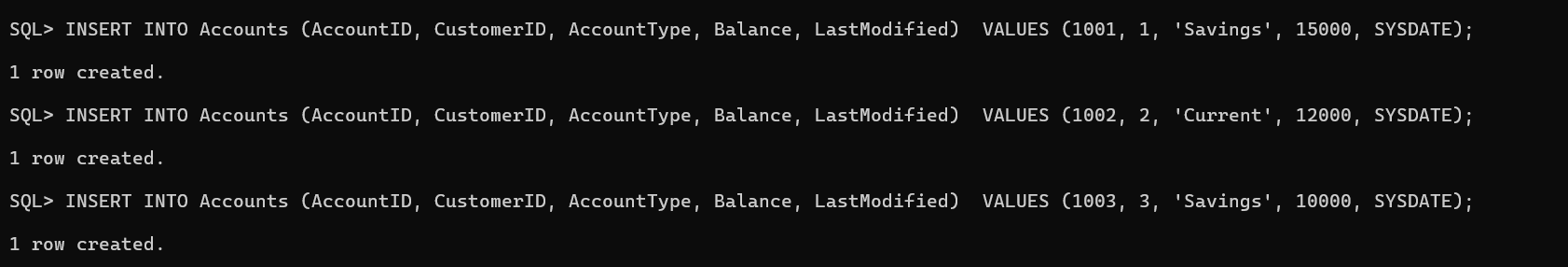
*LastModified DATE,*

*FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)*

*);*







**SCENARIO 1**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings' ) LOOP

UPDATE Accounts

SET Balance = acc.Balance \* 1.01,

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE(' Interest added to AccountID: ' || acc.AccountID);

END LOOP;

COMMIT;

END;

/

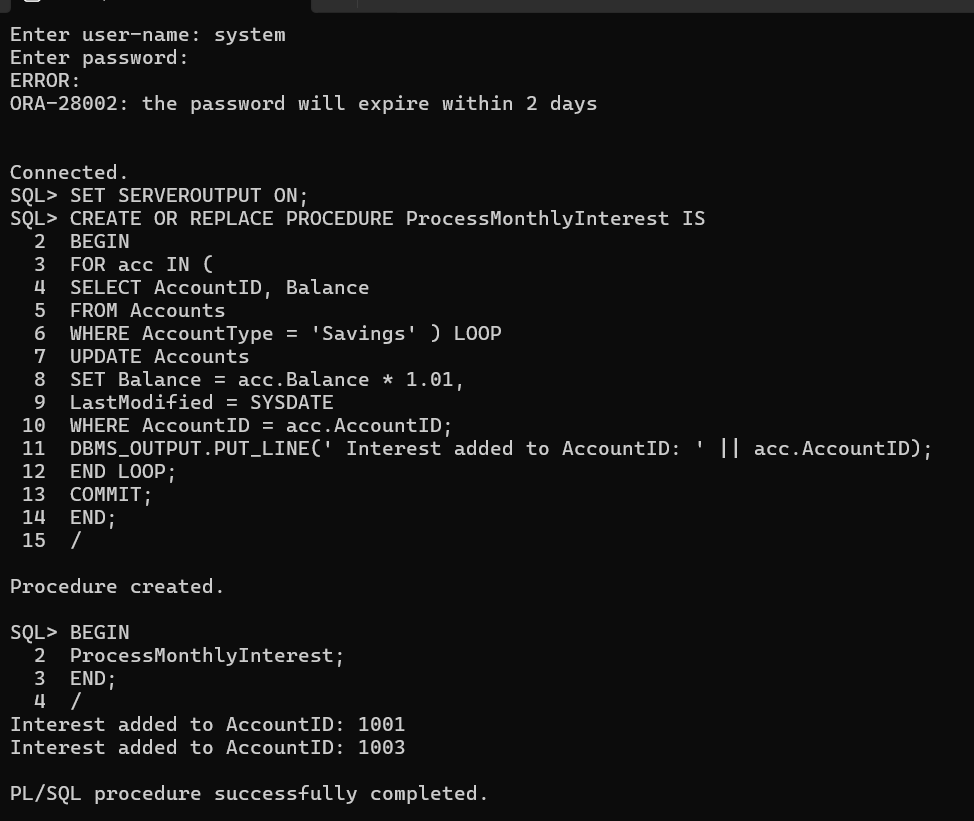
BEGIN

ProcessMonthlyInterest;

END;

/

**OUTPUT:**



**SCENARIO 2**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_BonusPercent / 100)

WHERE Department = p\_Department;

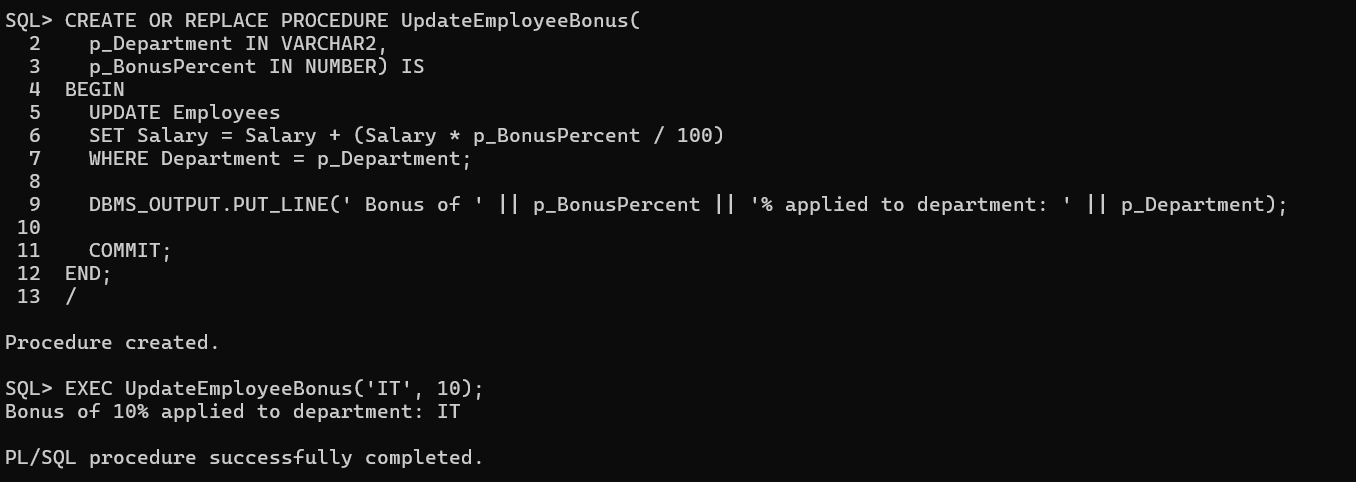
DBMS\_OUTPUT.PUT\_LINE(' Bonus of ' || p\_BonusPercent || '% applied to department: ' || p\_Department);

COMMIT;

END;

/

EXEC UpdateEmployeeBonus('IT', 10);



**SCENARIO 3**

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_account IN NUMBER,

to\_account IN NUMBER,

amount IN NUMBER

) IS

source\_balance NUMBER;

BEGIN

-- Lock and check source balance

SELECT Balance INTO source\_balance

FROM Accounts

WHERE AccountID = from\_account

FOR UPDATE;

IF source\_balance < amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance.');

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - amount,

LastModified = SYSDATE

WHERE AccountID = from\_account;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + amount,

LastModified = SYSDATE

WHERE AccountID = to\_account;

-- Confirmation message

DBMS\_OUTPUT.PUT\_LINE(amount || ' transferred from AccountID ' || from\_account || ' to AccountID ' || to\_account);

COMMIT;

END;

/

EXEC TransferFunds(1001, 1002, 1000);

OUTPUT

