Week 2: PL/SQL

**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.

# Implementation:

# -- Database 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

# );

# -- Sample Data

# INSERT INTO CUSTOMERS (CUSTOMERID, NAME, DOB, BALANCE, LASTMODIFIED)

# VALUES (1, 'Vivek', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

# INSERT INTO CUSTOMERS (CUSTOMERID, NAME, DOB, BALANCE, LASTMODIFIED)

# VALUES (2, 'Raju', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

# INSERT INTO ACCOUNTS (ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED)

# VALUES (1, 1, 'Savings', 1000, SYSDATE);

# INSERT INTO ACCOUNTS (ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED)

# VALUES (2, 2, 'Checking', 1500, SYSDATE);

# INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

# VALUES (1, 1, SYSDATE, 200, 'Deposit');

# INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

# VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

# INSERT INTO LOANS (LOANID, CUSTOMERID, LOANAMOUNT, INTERESTRATE, STARTDATE, ENDDATE)

# VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

# INSERT INTO EMPLOYEES (EMPLOYEEID, NAME, POSITION, SALARY, DEPARTMENT, HIREDATE)

# VALUES (1, 'Sethu Raman', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

# INSERT INTO EMPLOYEES (EMPLOYEEID, NAME, POSITION, SALARY, DEPARTMENT, HIREDATE)

# VALUES (2, 'Abdul Hashim', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

# \*/

# -- ========================================================================

# -- EXERCISE 1: CONTROL STRUCTURES

# -- ========================================================================

# /\*

# Scenario 1: Apply loan discount for senior customers (age > 60)

# Scenario 2: Set VIP status for high-balance customers (> $10,000)

# Scenario 3: Generate loan due reminders for next 30 days

# \*/

# -- SCENARIO 1: Senior Customer Loan Discount (Different approach using WHILE loop)

# SET SERVEROUTPUT ON;

# DECLARE

# TYPE customer\_record\_type IS RECORD (

# cust\_id CUSTOMERS.CUSTOMERID%TYPE,

# customer\_age NUMBER

# );

# 

# customer\_info customer\_record\_type;

# loan\_count NUMBER := 0;

# discount\_applied NUMBER := 0;

# 

# CURSOR senior\_customers IS

# SELECT CUSTOMERID,

# FLOOR(MONTHS\_BETWEEN(SYSDATE, DOB) / 12) AS CUSTOMER\_AGE

# FROM CUSTOMERS;

# BEGIN

# DBMS\_OUTPUT.PUT\_LINE('=== SENIOR CUSTOMER DISCOUNT PROCESSING ===');

# 

# FOR customer\_info IN senior\_customers LOOP

# BEGIN

# -- Check if customer qualifies for senior discount

# IF customer\_info.CUSTOMER\_AGE > 60 THEN

# -- Apply 1% discount to all loans for this customer

# UPDATE LOANS

# SET INTERESTRATE = INTERESTRATE - 1.0

# WHERE CUSTOMERID = customer\_info.CUSTOMERID

# AND INTERESTRATE > 1.0; -- Ensure rate doesn't go below 1%

# 

# -- Count affected loans

# loan\_count := SQL%ROWCOUNT;

# 

# IF loan\_count > 0 THEN

# discount\_applied := discount\_applied + loan\_count;

# DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || customer\_info.CUSTOMERID ||

# ' (Age: ' || customer\_info.CUSTOMER\_AGE ||

# ') - Discount applied to ' || loan\_count || ' loan(s)');

# END IF;

# ELSE

# DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || customer\_info.CUSTOMERID ||

# ' (Age: ' || customer\_info.CUSTOMER\_AGE ||

# ') - No discount (under 60)');

# END IF;

# EXCEPTION

# WHEN OTHERS THEN

# DBMS\_OUTPUT.PUT\_LINE('Error processing customer ' || customer\_info.CUSTOMERID || ': ' || SQLERRM);

# END;

# END LOOP;

# 

# DBMS\_OUTPUT.PUT\_LINE('Total loans with discount applied: ' || discount\_applied);

# COMMIT;

# END;

# /

# -- SCENARIO 2: VIP Status Assignment (Using different column approach)

# ALTER TABLE CUSTOMERS ADD VIP\_STATUS VARCHAR2(5) DEFAULT 'FALSE';

# SET SERVEROUTPUT ON;

# DECLARE

# high\_balance\_threshold NUMBER := 10000;

# vip\_count NUMBER := 0;

# regular\_count NUMBER := 0;

# 

# CURSOR balance\_check IS

# SELECT CUSTOMERID, NAME, BALANCE

# FROM CUSTOMERS

# ORDER BY BALANCE DESC;

# 

# client\_rec balance\_check%ROWTYPE;

# BEGIN

# DBMS\_OUTPUT.PUT\_LINE('=== VIP STATUS ASSIGNMENT ===');

# DBMS\_OUTPUT.PUT\_LINE('VIP Threshold: $' || high\_balance\_threshold);

# 

# OPEN balance\_check;

# LOOP

# FETCH balance\_check INTO client\_rec;

# EXIT WHEN balance\_check%NOTFOUND;

# 

# CASE

# WHEN client\_rec.BALANCE > high\_balance\_threshold THEN

# UPDATE CUSTOMERS

# SET VIP\_STATUS = 'TRUE'

# WHERE CUSTOMERID = client\_rec.CUSTOMERID;

# 

# vip\_count := vip\_count + 1;

# DBMS\_OUTPUT.PUT\_LINE('VIP: ' || client\_rec.NAME ||

# ' (ID: ' || client\_rec.CUSTOMERID ||

# ') - Balance: $' || client\_rec.BALANCE);

# ELSE

# UPDATE CUSTOMERS

# SET VIP\_STATUS = 'FALSE'

# WHERE CUSTOMERID = client\_rec.CUSTOMERID;

# 

# regular\_count := regular\_count + 1;

# DBMS\_OUTPUT.PUT\_LINE('Regular: ' || client\_rec.NAME ||

# ' (ID: ' || client\_rec.CUSTOMERID ||

# ') - Balance: $' || client\_rec.BALANCE);

# END CASE;

# END LOOP;

# CLOSE balance\_check;

# 

# DBMS\_OUTPUT.PUT\_LINE('Summary - VIP Customers: ' || vip\_count || ', Regular Customers: ' || regular\_count);

# COMMIT;

# END;

# /

# -- SCENARIO 3: Loan Due Reminders (Enhanced with categorization)

# SET SERVEROUTPUT ON;

# DECLARE

# reminder\_period NUMBER := 30;

# urgent\_period NUMBER := 7;

# critical\_count NUMBER := 0;

# warning\_count NUMBER := 0;

# days\_remaining NUMBER;

# 

# CURSOR upcoming\_loans IS

# SELECT l.LOANID, l.CUSTOMERID, c.NAME, l.ENDDATE, l.LOANAMOUNT

# FROM LOANS l

# INNER JOIN CUSTOMERS c ON l.CUSTOMERID = c.CUSTOMERID

# WHERE l.ENDDATE BETWEEN SYSDATE AND (SYSDATE + reminder\_period)

# ORDER BY l.ENDDATE ASC;

# 

# loan\_info upcoming\_loans%ROWTYPE;

# BEGIN

# DBMS\_OUTPUT.PUT\_LINE('=== LOAN DUE REMINDERS ===');

# DBMS\_OUTPUT.PUT\_LINE('Checking loans due within ' || reminder\_period || ' days');

# 

# OPEN upcoming\_loans;

# LOOP

# FETCH upcoming\_loans INTO loan\_info;

# EXIT WHEN upcoming\_loans%NOTFOUND;

# 

# days\_remaining := TRUNC(loan\_info.ENDDATE - SYSDATE);

# 

# IF days\_remaining <= urgent\_period THEN

# critical\_count := critical\_count + 1;

# DBMS\_OUTPUT.PUT\_LINE('\*\* URGENT \*\* Loan #' || loan\_info.LOANID ||

# ' for ' || loan\_info.NAME ||

# ' (Amount: $' || loan\_info.LOANAMOUNT ||

# ') due in ' || days\_remaining || ' days!');

# ELSE

# warning\_count := warning\_count + 1;

# DBMS\_OUTPUT.PUT\_LINE('Warning: Loan #' || loan\_info.LOANID ||

# ' for ' || loan\_info.NAME ||

# ' (Amount: $' || loan\_info.LOANAMOUNT ||

# ') due in ' || days\_remaining || ' days');

# END IF;

# END LOOP;

# CLOSE upcoming\_loans;

# 

# IF (critical\_count + warning\_count) = 0 THEN

# DBMS\_OUTPUT.PUT\_LINE('No loans due within the next ' || reminder\_period || ' days.');

# ELSE

# DBMS\_OUTPUT.PUT\_LINE('Summary - Critical: ' || critical\_count || ', Warnings: ' || warning\_count);

# END IF;

# END;

# /

**Output (Scenario 1):**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.**Output (Scenario 2):**

A screenshot of a computer program

AI-generated content may be incorrect.Output (Scenario 3):

Week 2: PL/SQL

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.

# Implementation:

# -- Database 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

# );

# -- Sample Data

# INSERT INTO CUSTOMERS (CUSTOMERID, NAME, DOB, BALANCE, LASTMODIFIED)

# VALUES (1, 'Vivek', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

# INSERT INTO CUSTOMERS (CUSTOMERID, NAME, DOB, BALANCE, LASTMODIFIED)

# VALUES (2, 'Raju', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

# INSERT INTO ACCOUNTS (ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED)

# VALUES (1, 1, 'Savings', 1000, SYSDATE);

# INSERT INTO ACCOUNTS (ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED)

# VALUES (2, 2, 'Checking', 1500, SYSDATE);

# INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

# VALUES (1, 1, SYSDATE, 200, 'Deposit');

# INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

# VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

# INSERT INTO LOANS (LOANID, CUSTOMERID, LOANAMOUNT, INTERESTRATE, STARTDATE, ENDDATE)

# VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

# INSERT INTO EMPLOYEES (EMPLOYEEID, NAME, POSITION, SALARY, DEPARTMENT, HIREDATE)

# VALUES (1, 'Sethu Raman', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

# INSERT INTO EMPLOYEES (EMPLOYEEID, NAME, POSITION, SALARY, DEPARTMENT, HIREDATE)

# VALUES (2, 'Abdul Hashim', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

# \*/

# -- ========================================================================

# -- EXERCISE 3: STORED PROCEDURES (Modified Implementation)

# -- ========================================================================

# /\*

# Exercise 3: Stored Procedures

# Scenario 1: Monthly interest processing for savings accounts

# Scenario 2: Department-based employee bonus system

# Scenario 3: Enhanced fund transfer with validation

# \*/

# -- SCENARIO 1: Monthly Interest Processing

# CREATE OR REPLACE PROCEDURE APPLY\_SAVINGS\_INTEREST(

# p\_interest\_rate IN NUMBER DEFAULT 1.0

# ) AS

# accounts\_processed NUMBER := 0;

# total\_interest\_added NUMBER := 0;

# interest\_amount NUMBER;

# 

# CURSOR savings\_accounts IS

# SELECT ACCOUNTID, BALANCE, CUSTOMERID

# FROM ACCOUNTS

# WHERE ACCOUNTTYPE = 'Savings'

# AND BALANCE > 0

# FOR UPDATE;

# 

# account\_info savings\_accounts%ROWTYPE;

# BEGIN

# DBMS\_OUTPUT.PUT\_LINE('=== MONTHLY INTEREST PROCESSING ===');

# DBMS\_OUTPUT.PUT\_LINE('Interest Rate: ' || p\_interest\_rate || '%');

# 

# OPEN savings\_accounts;

# LOOP

# FETCH savings\_accounts INTO account\_info;

# EXIT WHEN savings\_accounts%NOTFOUND;

# 

# interest\_amount := account\_info.BALANCE \* (p\_interest\_rate / 100);

# 

# UPDATE ACCOUNTS

# SET BALANCE = BALANCE + interest\_amount,

# LASTMODIFIED = SYSDATE

# WHERE ACCOUNTID = account\_info.ACCOUNTID;

# 

# accounts\_processed := accounts\_processed + 1;

# total\_interest\_added := total\_interest\_added + interest\_amount;

# 

# DBMS\_OUTPUT.PUT\_LINE('Account ' || account\_info.ACCOUNTID ||

# ' (Customer: ' || account\_info.CUSTOMERID ||

# ') - Interest: $' || ROUND(interest\_amount, 2));

# END LOOP;

# CLOSE savings\_accounts;

# 

# COMMIT;

# DBMS\_OUTPUT.PUT\_LINE('Processing Complete - Accounts: ' || accounts\_processed ||

# ', Total Interest: $' || ROUND(total\_interest\_added, 2));

# EXCEPTION

# WHEN OTHERS THEN

# ROLLBACK;

# DBMS\_OUTPUT.PUT\_LINE('Interest processing failed: ' || SQLERRM);

# RAISE;

# END APPLY\_SAVINGS\_INTEREST;

# /

# -- EXECUTION

# BEGIN

# APPLY\_SAVINGS\_INTEREST(1.5);

# END;

# /

# -- SCENARIO 2: Department Bonus System

# CREATE OR REPLACE PROCEDURE DISTRIBUTE\_DEPARTMENT\_BONUS(

# p\_dept\_name IN EMPLOYEES.DEPARTMENT%TYPE,

# p\_bonus\_percent IN NUMBER

# ) AS

# employees\_updated NUMBER := 0;

# total\_bonus\_amount NUMBER := 0;

# old\_salary NUMBER;

# bonus\_amount NUMBER;

# dept\_exists NUMBER;

# CURSOR dept\_employees IS

# SELECT EMPLOYEEID, NAME, SALARY

# FROM EMPLOYEES

# WHERE UPPER(DEPARTMENT) = UPPER(p\_dept\_name)

# FOR UPDATE;

# emp\_record dept\_employees%ROWTYPE;

# BEGIN

# SELECT COUNT(\*) INTO dept\_exists

# FROM EMPLOYEES

# WHERE UPPER(DEPARTMENT) = UPPER(p\_dept\_name);

# 

# IF dept\_exists = 0 THEN

# RAISE\_APPLICATION\_ERROR(-20001, 'Department "' || p\_dept\_name || '" not found');

# END IF;

# IF p\_bonus\_percent <= 0 OR p\_bonus\_percent > 100 THEN

# RAISE\_APPLICATION\_ERROR(-20002, 'Bonus percentage must be between 0 and 100');

# END IF;

# DBMS\_OUTPUT.PUT\_LINE('=== DEPARTMENT BONUS DISTRIBUTION ===');

# DBMS\_OUTPUT.PUT\_LINE('Department: ' || p\_dept\_name);

# DBMS\_OUTPUT.PUT\_LINE('Bonus Percentage: ' || p\_bonus\_percent || '%');

# OPEN dept\_employees;

# LOOP

# FETCH dept\_employees INTO emp\_record;

# EXIT WHEN dept\_employees%NOTFOUND;

# old\_salary := emp\_record.SALARY;

# bonus\_amount := old\_salary \* (p\_bonus\_percent / 100);

# UPDATE EMPLOYEES

# SET SALARY = SALARY + bonus\_amount,

# HIREDATE = SYSDATE

# WHERE EMPLOYEEID = emp\_record.EMPLOYEEID;

# employees\_updated := employees\_updated + 1;

# total\_bonus\_amount := total\_bonus\_amount + bonus\_amount;

# DBMS\_OUTPUT.PUT\_LINE('Employee: ' || emp\_record.NAME ||

# ' (ID: ' || emp\_record.EMPLOYEEID ||

# ') - Bonus: $' || ROUND(bonus\_amount, 2) ||

# ' - New Salary: $' || ROUND(old\_salary + bonus\_amount, 2));

# END LOOP;

# CLOSE dept\_employees;

# COMMIT;

# DBMS\_OUTPUT.PUT\_LINE('Bonus Distribution Complete - Employees: ' || employees\_updated ||

# ', Total Bonus: $' || ROUND(total\_bonus\_amount, 2));

# EXCEPTION

# WHEN OTHERS THEN

# ROLLBACK;

# DBMS\_OUTPUT.PUT\_LINE('Bonus distribution failed: ' || SQLERRM);

# RAISE;

# END DISTRIBUTE\_DEPARTMENT\_BONUS;

# /

# -- EXECUTION

# BEGIN

# DISTRIBUTE\_DEPARTMENT\_BONUS('IT', 5.0);

# END;

# /

# -- SCENARIO 3: Enhanced Fund Transfer with Comprehensive Validation

# -- Create or Replace Procedure

# CREATE OR REPLACE PROCEDURE EXECUTE\_FUND\_TRANSFER(

# p\_source\_account IN ACCOUNTS.ACCOUNTID%TYPE,

# p\_destination\_account IN ACCOUNTS.ACCOUNTID%TYPE,

# p\_transfer\_amount IN NUMBER

# ) AS

# source\_balance ACCOUNTS.BALANCE%TYPE;

# dest\_balance ACCOUNTS.BALANCE%TYPE;

# source\_customer ACCOUNTS.CUSTOMERID%TYPE;

# dest\_customer ACCOUNTS.CUSTOMERID%TYPE;

# 

# -- Custom exceptions

# insufficient\_funds EXCEPTION;

# invalid\_amount EXCEPTION;

# account\_not\_found EXCEPTION;

# same\_account\_error EXCEPTION;

# BEGIN

# -- Input validation

# IF p\_transfer\_amount <= 0 THEN

# RAISE invalid\_amount;

# END IF;

# 

# IF p\_source\_account = p\_destination\_account THEN

# RAISE same\_account\_error;

# END IF;

# 

# DBMS\_OUTPUT.PUT\_LINE('=== FUND TRANSFER PROCESSING ===');

# DBMS\_OUTPUT.PUT\_LINE('From Account: ' || p\_source\_account || ' To Account: ' || p\_destination\_account);

# DBMS\_OUTPUT.PUT\_LINE('Amount: $' || p\_transfer\_amount);

# 

# -- Lock and validate source account

# BEGIN

# SELECT BALANCE, CUSTOMERID INTO source\_balance, source\_customer

# FROM ACCOUNTS

# WHERE ACCOUNTID = p\_source\_account

# FOR UPDATE;

# EXCEPTION

# WHEN NO\_DATA\_FOUND THEN

# RAISE account\_not\_found;

# END;

# 

# -- Lock and validate destination account

# BEGIN

# SELECT BALANCE, CUSTOMERID INTO dest\_balance, dest\_customer

# FROM ACCOUNTS

# WHERE ACCOUNTID = p\_destination\_account

# FOR UPDATE;

# EXCEPTION

# WHEN NO\_DATA\_FOUND THEN

# RAISE account\_not\_found;

# END;

# 

# -- Check sufficient funds

# IF source\_balance < p\_transfer\_amount THEN

# RAISE insufficient\_funds;

# END IF;

# 

# -- Execute transfer

# UPDATE ACCOUNTS

# SET BALANCE = BALANCE - p\_transfer\_amount,

# LASTMODIFIED = SYSDATE

# WHERE ACCOUNTID = p\_source\_account;

# 

# UPDATE ACCOUNTS

# SET BALANCE = BALANCE + p\_transfer\_amount,

# LASTMODIFIED = SYSDATE

# WHERE ACCOUNTID = p\_destination\_account;

# 

# -- Record transaction history (optional)

# INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

# VALUES (transactions\_seq.NEXTVAL, p\_source\_account, SYSDATE, p\_transfer\_amount, 'Transfer');

# 

# INSERT INTO TRANSACTIONS (TRANSACTIONID, ACCOUNTID, TRANSACTIONDATE, AMOUNT, TRANSACTIONTYPE)

# VALUES (transactions\_seq.NEXTVAL, p\_destination\_account, SYSDATE, p\_transfer\_amount, 'Transfer');

# 

# COMMIT;

# 

# DBMS\_OUTPUT.PUT\_LINE('Transfer Successful!');

# DBMS\_OUTPUT.PUT\_LINE('Source Account Balance: $' || (source\_balance - p\_transfer\_amount));

# DBMS\_OUTPUT.PUT\_LINE('Destination Account Balance: $' || (dest\_balance + p\_transfer\_amount));

# EXCEPTION

# WHEN insufficient\_funds THEN

# ROLLBACK;

# DBMS\_OUTPUT.PUT\_LINE('Transfer Failed: Insufficient funds (Available: $' || source\_balance || ')');

# WHEN invalid\_amount THEN

# ROLLBACK;

# DBMS\_OUTPUT.PUT\_LINE('Transfer Failed: Invalid transfer amount');

# WHEN account\_not\_found THEN

# ROLLBACK;

# DBMS\_OUTPUT.PUT\_LINE('Transfer Failed: One or both accounts not found');

# WHEN same\_account\_error THEN

# ROLLBACK;

# DBMS\_OUTPUT.PUT\_LINE('Transfer Failed: Source and destination accounts cannot be the same');

# WHEN OTHERS THEN

# ROLLBACK;

# DBMS\_OUTPUT.PUT\_LINE('Transfer Failed: ' || SQLERRM);

# RAISE;

# END EXECUTE\_FUND\_TRANSFER;

# /

# -- Call the procedure

# BEGIN

# EXECUTE\_FUND\_TRANSFER(1, 2, 100); -- Modify account IDs and amount as needed

# END;

**Output (Scenario 1):**

A screenshot of a computer

AI-generated content may be incorrect.

**Output (Scenario 2):**

A screenshot of a computer screen

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.**Output (Scenario 3):**