**Exercise 1: Control Structures**

**Scenario 1:**

DECLARE

v\_customer\_age NUMBER;

v\_discounted\_rate NUMBER;

BEGIN

FOR rec IN (SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate

FROM Customers JOIN Loans l ON c.CustomerID = l.CustomerID) LOOP

v\_customer\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12);

IF v\_customer\_age > 60 THEN

v\_discounted\_rate := rec.InterestRate - 1;

UPDATE Loans SET InterestRate = v\_discounted\_rate WHERE LoanID = rec.LoanID;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 2:**

**--**Adding isVIP column to table to use the IsVIP flag

ALTER TABLE Customers ADD (IsVIP VARCHAR2(5) DEFAULT 'FALSE');

BEGIN

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

IF rec.Balance > 10000 THEN

UPDATE Customers SET IsVIP = 'TRUE' WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 3:**

DECLARE

v\_due\_date DATE;

v\_customer\_name VARCHAR2(100);

v\_customer\_id NUMBER;

BEGIN

FOR rec IN (SELECT l.LoanID, l.EndDate, c.Name, c.CustomerID

FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID

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

v\_due\_date := rec.EndDate;

v\_customer\_name := rec.Name;

v\_customer\_id := rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan with ID ' || rec.LoanID || ' for customer ' || v\_customer\_name || ' (CustomerID: ' || v\_customer\_id || ') is due on ' || O\_CHAR(v\_due\_date, 'YYYY-MM-DD') || '.');

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id NUMBER,

p\_to\_account\_id NUMBER,

p\_amount NUMBER

) AS

v\_from\_balance NUMBER;

v\_to\_balance NUMBER;

BEGIN

-- Start the transaction

SAVEPOINT sp\_before\_transfer;

-- Check the balance of the source account

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in the source account.');

END IF;

-- Deduct the amount from the source account

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account\_id;

-- Add the amount to the destination account

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK TO sp\_before\_transfer;

DBMS\_OUTPUT.PUT\_LINE('Error during fund transfer: ' || SQLERRM);

END;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id NUMBER,

p\_percentage NUMBER

) AS

v\_old\_salary NUMBER;

BEGIN

-- Check if the employee exists and get the current salary

SELECT Salary INTO v\_old\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id

FOR UPDATE;­

-- Update the salary

UPDATE Employees

SET Salary = Salary + (Salary \* p\_percentage / 100),

HireDate = HireDate -- Retain original HireDate

WHERE EmployeeID = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee with ID ' || p\_employee\_id || ' does not exist.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error updating salary: ' || SQLERRM);

ROLLBACK;

END;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

) AS

BEGIN

-- Attempt to insert the new customer

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customer\_id || ' already exists.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error adding new customer: ' || SQLERRM);

ROLLBACK;

END;

/

**Exercise 3: Stored Procedures**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR rec IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

END;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department VARCHAR2,

p\_bonus\_percentage NUMBER

) AS

BEGIN

FOR rec IN (SELECT EmployeeID, Salary FROM Employees WHERE Department = p\_department) LOOP

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percentage / 100),

HireDate = HireDate -- Retain original HireDate

WHERE EmployeeID = rec.EmployeeID;

END LOOP;

COMMIT;

END;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id NUMBER,

p\_to\_account\_id NUMBER,

p\_amount NUMBER

) AS

v\_from\_balance NUMBER;

v\_to\_balance NUMBER;

BEGIN

-- Check the balance of the source account

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in the source account.');

END IF;

-- Deduct the amount from the source account

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account\_id;

-- Add the amount to the destination account

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error during fund transfer: ' || SQLERRM);

END;

/

**Exercise 4: Functions**

**Scenario 1:**

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob DATE

) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END CalculateAge;

/

**Scenario 2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_duration\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_interest\_rate NUMBER;

v\_num\_payments NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

v\_monthly\_interest\_rate := p\_interest\_rate / 1200;

v\_num\_payments := p\_duration\_years \* 12;

IF v\_monthly\_interest\_rate = 0 THEN

v\_monthly\_installment := p\_loan\_amount / v\_num\_payments;

ELSE

v\_monthly\_installment := p\_loan\_amount \* v\_monthly\_interest\_rate /

(1 - POWER(1 + v\_monthly\_interest\_rate, -v\_num\_payments));

END IF;

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

**Scenario 3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END HasSufficientBalance;

/

**Exercise 5: Triggers**

**Scenario 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Scenario 2:**

--Table fot logs

CREATE TABLE AuditLog (

LogID NUMBER PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

LogDate DATE

);

--Sequence to generate primary keys in Audit log

CREATE SEQUENCE AuditLog\_seq START WITH 1 INCREMENT BY 1;

--Trigger for logs

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (LogID, TransactionID, AccountID, TransactionDate, Amount, TransactionType, LogDate)

VALUES (AuditLog\_seq.NEXTVAL, :NEW.TransactionID, :NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType, SYSDATE);

END;

/

**Scenario 3:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID FOR UPDATE;

IF :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance for withdrawal.');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Deposit amount must be positive.');

END IF;

END IF;

END;

/

**Exercise 6: Cursors**

**Scenario 1:**

DECLARE

CURSOR cur\_monthly\_transactions IS

SELECT c.CustomerID, c.Name, t.TransactionDate, t.Amount, t.TransactionType

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE t.TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE);

v\_customer\_id Customers.CustomerID%TYPE;

v\_name Customers.Name%TYPE;

v\_transaction\_date Transactions.TransactionDate%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_transaction\_type Transactions.TransactionType%TYPE;

BEGIN

OPEN cur\_monthly\_transactions;

LOOP

FETCH cur\_monthly\_transactions INTO v\_customer\_id, v\_name, v\_transaction\_date, v\_amount, v\_transaction\_type;

EXIT WHEN cur\_monthly\_transactions%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id || ' - Name: ' || v\_name);

DBMS\_OUTPUT.PUT\_LINE('Date: ' || v\_transaction\_date || ' - Amount: ' || v\_amount || ' - Type: ' || v\_transaction\_type);

DBMS\_OUTPUT.PUT\_LINE('---------------------------');

END LOOP;

CLOSE cur\_monthly\_transactions;

END;

/

**Scenario 2:**

DECLARE

CURSOR cur\_accounts IS

SELECT AccountID, Balance

FROM Accounts;

v\_account\_id Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_annual\_fee CONSTANT NUMBER := 100; -- Assuming annual fee is 100

BEGIN

OPEN cur\_accounts;

LOOP

FETCH cur\_accounts INTO v\_account\_id, v\_balance;

EXIT WHEN cur\_accounts%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee,

LastModified = SYSDATE

WHERE AccountID = v\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Annual fee of ' || v\_annual\_fee || ' applied to Account ID: ' || v\_account\_id);

END LOOP;

CLOSE cur\_accounts;

COMMIT;

END;

/

**Scenario 3:**

DECLARE

CURSOR cur\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

v\_loan\_id Loans.LoanID%TYPE;

v\_interest\_rate Loans.InterestRate%TYPE;

v\_new\_interest\_rate NUMBER;

BEGIN

OPEN cur\_loans;

LOOP

FETCH cur\_loans INTO v\_loan\_id, v\_interest\_rate;

EXIT WHEN cur\_loans%NOTFOUND;

-- Assuming a new policy to increase interest rate by 0.5%

v\_new\_interest\_rate := v\_interest\_rate + 0.5;

UPDATE Loans SET InterestRate = v\_new\_interest\_rate WHERE LoanID = v\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Updated Loan ID: ' || v\_loan\_id || ' - New Interest Rate: ' || v\_new\_interest\_rate);

END LOOP;

CLOSE cur\_loans;

COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

);

PROCEDURE UpdateCustomerDetails (

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_balance NUMBER

);

FUNCTION GetCustomerBalance (

p\_customer\_id NUMBER

) RETURN NUMBER;

PROCEDURE GenerateMonthlyStatements;

PROCEDURE ApplyAnnualFee;

PROCEDURE UpdateLoanInterestRates;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

) IS

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails (

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_balance NUMBER

) IS

BEGIN

UPDATE Customers

SET Name = p\_name,

Balance = p\_balance,

LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance (

p\_customer\_id NUMBER

) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Customers

WHERE CustomerID = p\_customer\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END GetCustomerBalance;

PROCEDURE GenerateMonthlyStatements IS

CURSOR cur\_monthly\_transactions IS

SELECT c.CustomerID, c.Name, t.TransactionDate, t.Amount, t.TransactionType

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE t.TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE);

v\_customer\_id Customers.CustomerID%TYPE;

v\_name Customers.Name%TYPE;

v\_transaction\_date Transactions.TransactionDate%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_transaction\_type Transactions.TransactionType%TYPE;

BEGIN

OPEN cur\_monthly\_transactions;

LOOP

FETCH cur\_monthly\_transactions INTO v\_customer\_id, v\_name, v\_transaction\_date, v\_amount, v\_transaction\_type;

EXIT WHEN cur\_monthly\_transactions%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id || ' - Name: ' || v\_name);

DBMS\_OUTPUT.PUT\_LINE('Date: ' || v\_transaction\_date || ' - Amount: ' || v\_amount || ' - Type: ' || v\_transaction\_type);

DBMS\_OUTPUT.PUT\_LINE('---------------------------');

END LOOP;

CLOSE cur\_monthly\_transactions;

END GenerateMonthlyStatements;

PROCEDURE ApplyAnnualFee IS

CURSOR cur\_accounts IS

SELECT AccountID, Balance

FROM Accounts;

v\_account\_id Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_annual\_fee CONSTANT NUMBER := 100; -- Assuming annual fee is 100

BEGIN

OPEN cur\_accounts;

LOOP

FETCH cur\_accounts INTO v\_account\_id, v\_balance;

EXIT WHEN cur\_accounts%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee,

LastModified = SYSDATE

WHERE AccountID = v\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Annual fee of ' || v\_annual\_fee || ' applied to Account ID: ' || v\_account\_id);

END LOOP;

CLOSE cur\_accounts;

COMMIT;

END ApplyAnnualFee;

PROCEDURE UpdateLoanInterestRates IS

CURSOR cur\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

v\_loan\_id Loans.LoanID%TYPE;

v\_interest\_rate Loans.InterestRate%TYPE;

v\_new\_interest\_rate NUMBER;

BEGIN

OPEN cur\_loans;

LOOP

FETCH cur\_loans INTO v\_loan\_id, v\_interest\_rate;

EXIT WHEN cur\_loans%NOTFOUND;

-- Assuming a new policy to increase interest rate by 0.5%

v\_new\_interest\_rate := v\_interest\_rate + 0.5;

UPDATE Loans

SET InterestRate = v\_new\_interest\_rate

WHERE LoanID = v\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Updated Loan ID: ' || v\_loan\_id || ' - New Interest Rate: ' || v\_new\_interest\_rate);

END LOOP;

CLOSE cur\_loans;

COMMIT;

END UpdateLoanInterestRates;

END CustomerManagement;

/

**Scenario 2:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee (

p\_employee\_id NUMBER,

p\_name VARCHAR2,

p\_position VARCHAR2,

p\_salary NUMBER,

p\_department VARCHAR2

);

PROCEDURE UpdateEmployeeDetails (

p\_employee\_id NUMBER,

p\_name VARCHAR2,

p\_salary NUMBER

);

FUNCTION CalculateAnnualSalary (

p\_employee\_id NUMBER

) RETURN NUMBER;

PROCEDURE UpdateSalary (

p\_employee\_id NUMBER,

p\_percentage NUMBER

);

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee (

p\_employee\_id NUMBER,

p\_name VARCHAR2,

p\_position VARCHAR2,

p\_salary NUMBER,

p\_department VARCHAR2

) IS

BEGIN

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (p\_employee\_id, p\_name, p\_position, p\_salary, p\_department, SYSDATE);

END HireEmployee;

PROCEDURE UpdateEmployeeDetails (

p\_employee\_id NUMBER,

p\_name VARCHAR2,

p\_salary NUMBER

) IS

BEGIN

UPDATE Employees

SET Name = p\_name,

Salary = p\_salary

WHERE EmployeeID = p\_employee\_id;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary (

p\_employee\_id NUMBER

) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

RETURN v\_salary \* 12; -- Annual salary

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END CalculateAnnualSalary;

PROCEDURE UpdateSalary (

p\_employee\_id NUMBER,

p\_percentage NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary \* (1 + p\_percentage / 100)

WHERE EmployeeID = p\_employee\_id;

IF SQL%NOTFOUND THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Employee ID does not exist.');

END IF;

END UpdateSalary;

END EmployeeManagement;

/

**Scenario 3:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount (

p\_account\_id NUMBER,

p\_customer\_id NUMBER,

p\_account\_type VARCHAR2,

p\_balance NUMBER

);

PROCEDURE CloseAccount (

p\_account\_id NUMBER

);

FUNCTION GetTotalBalance (

p\_customer\_id NUMBER

) RETURN NUMBER;

PROCEDURE TransferFunds (

p\_source\_account\_id NUMBER,

p\_target\_account\_id NUMBER,

p\_amount NUMBER

);

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenNewAccount (

p\_account\_id NUMBER,

p\_customer\_id NUMBER,

p\_account\_type VARCHAR2,

p\_balance NUMBER

) IS

BEGIN

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (p\_account\_id, p\_customer\_id, p\_account\_type, p\_balance, SYSDATE);

END OpenNewAccount;

PROCEDURE CloseAccount (

p\_account\_id NUMBER

) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_account\_id;

END CloseAccount;

FUNCTION GetTotalBalance (

p\_customer\_id NUMBER

) RETURN NUMBER IS

v\_total\_balance NUMBER;

BEGIN

SELECT NVL(SUM(Balance), 0) INTO v\_total\_balance

FROM Accounts

WHERE CustomerID = p\_customer\_id;

RETURN v\_total\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END GetTotalBalance;

PROCEDURE TransferFunds (

p\_source\_account\_id NUMBER,

p\_target\_account\_id NUMBER,

p\_amount NUMBER

) IS

v\_source\_balance NUMBER;

v\_target\_balance NUMBER;

BEGIN

-- Check balances

SELECT Balance INTO v\_source\_balance

FROM Accounts

WHERE AccountID = p\_source\_account\_id;

SELECT Balance INTO v\_target\_balance

FROM Accounts

WHERE AccountID = p\_target\_account\_id;

IF v\_source\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient funds in source account.');

END IF;

-- Perform transfer

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_source\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_target\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_amount || ' from Account ID ' || p\_source\_account\_id || ' to Account ID ' || p\_target\_account\_id);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20003, 'One or both account IDs do not exist.');

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Error occurred during fund transfer.');

END TransferFunds;

END AccountOperations;

/