**Exercise 1:**

(SCENARIO 1)

DECLARE

CURSOR customer\_cursor IS

SELECT CustomerID, DOB

FROM Customers;

v\_CurrentDate DATE := SYSDATE;

v\_Age NUMBER;

BEGIN

FOR i IN customer\_cursor LOOP

-- Calculate age

v\_Age := FLOOR(MONTHS\_BETWEEN(v\_CurrentDate, i.DOB) / 12);

-- Check if customer is over 60 years old

IF v\_Age > 60 THEN

-- Update loan interest rates for this customer

UPDATE Loans

SET InterestRate = InterestRate – 0.01\*InterestRate

WHERE CustomerID = i.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

(SCENARIO 2)

--Adding column IsVIP to customers table

ALTER TABLE Customers ADD (IsVIP CHAR(1) DEFAULT 'N');

DECLARE

CURSOR customer\_cursor IS

SELECT CustomerID, Balance

FROM Customers;

BEGIN

FOR i IN customer\_cursor LOOP

-- Check if customer balance is over $10,000

IF i.Balance > 10000 THEN

-- Update VIP status for this customer

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = i.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

(SCENARIO 3)

DECLARE

CURSOR loan\_cursor IS

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR i IN loan\_cursor LOOP

-- Print a reminder message

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ' || i.Name ||

' (ID: ' || i.CustomerID ||

') has a loan (ID: ' || i.LoanID ||

') that is due on ' || i.EndDate);

END LOOP;

END;

/

**Exercise 2:**

(SCENARIO 1)

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) AS

v\_FromBalance NUMBER;

v\_ToBalance NUMBER;

BEGIN

-- Check if the amount is positive

IF p\_Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Transfer amount must be greater than zero.');

END IF;

-- Get current balances

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID;

SELECT Balance INTO v\_ToBalance

FROM Accounts

WHERE AccountID = p\_ToAccountID;

-- Check for sufficient funds

IF v\_FromBalance < p\_Amount THEN

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

END IF;

-- Perform the transfer

BEGIN

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

RAISE;

END;

END;

/

(SCENARIO 2)

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_EmployeeID IN NUMBER,

p\_Percentage IN NUMBER

) AS

BEGIN

-- Check if percentage is valid

IF p\_Percentage <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Percentage must be greater than zero.');

END IF;

-- Update salary

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_EmployeeID;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Employee ID does not exist.');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

RAISE;

END;

END;

/

(SCENARIO 3)

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

) AS

BEGIN

-- Check if customer already exists

BEGIN

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

VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: A customer with this ID already exists.');

WHEN OTHERS THEN

ROLLBACK;

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

RAISE;

END;

END;

/

**Exercise 3:**

(SCENARIO 1)

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

-- Update all savings accounts by applying 1% interest

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error processing monthly interest: ' || SQLERRM);

RAISE;

END;

END;

/

(SCENARIO 2)

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPercentage IN NUMBER

) AS

BEGIN

-- Check if the bonus percentage is valid

IF p\_BonusPercentage <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Bonus percentage must be greater than zero.');

END IF;

-- Update the salary of employees in the specified department

BEGIN

UPDATE Employees

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

WHERE Department = p\_Department;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

RAISE;

END;

END;

/

(SCENARIO 3)

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) AS

v\_FromBalance NUMBER;

v\_ToBalance NUMBER;

BEGIN

-- Check if the amount is positive

IF p\_Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Transfer amount must be greater than zero.');

END IF;

-- Retrieve current balances

BEGIN

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID;

SELECT Balance INTO v\_ToBalance

FROM Accounts

WHERE AccountID = p\_ToAccountID;

-- Check if the source account exists

IF v\_FromBalance IS NULL THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Source account does not exist.');

END IF;

-- Check if the destination account exists

IF v\_ToBalance IS NULL THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Destination account does not exist.');

END IF;

-- Check for sufficient funds

IF v\_FromBalance < p\_Amount THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Insufficient funds in the source account.');

END IF;

-- Perform the transfer

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: One or both account IDs do not exist.');

WHEN OTHERS THEN

ROLLBACK;

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

RAISE;

END;

END;

/

**Exercise 4:**

(SCENARIO 1)

CREATE OR REPLACE FUNCTION CalculateAge (

p\_DOB IN DATE

) RETURN NUMBER AS

v\_Age NUMBER;

BEGIN

-- Calculate age based on date of birth

v\_Age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

RETURN v\_Age;

END;

/

(SCENARIO 2)

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_LoanAmount IN NUMBER,

p\_AnnualInterestRate IN NUMBER,

p\_DurationYears IN NUMBER

) RETURN NUMBER AS

v\_MonthlyInterestRate NUMBER;

v\_NumberOfInstallments NUMBER;

v\_MonthlyInstallment NUMBER;

BEGIN

-- Calculate monthly interest rate

v\_MonthlyInterestRate := p\_AnnualInterestRate / 100 / 12;

-- Calculate total number of installments

v\_NumberOfInstallments := p\_DurationYears \* 12;

-- Calculate monthly installment using the formula

IF v\_MonthlyInterestRate > 0 THEN

v\_MonthlyInstallment := p\_LoanAmount \* v\_MonthlyInterestRate \*

POWER(1 + v\_MonthlyInterestRate, v\_NumberOfInstallments) /

(POWER(1 + v\_MonthlyInterestRate, v\_NumberOfInstallments) - 1);

ELSE

v\_MonthlyInstallment := p\_LoanAmount / v\_NumberOfInstallments;

END IF;

RETURN v\_MonthlyInstallment;

END;

/

(SCENARIO 3)

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_AccountID IN NUMBER,

p\_Amount IN NUMBER

) RETURN BOOLEAN AS

v\_Balance NUMBER;

BEGIN

-- Retrieve the balance of the account

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = p\_AccountID;

-- Check if the account balance is sufficient

RETURN v\_Balance >= p\_Amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Account does not exist

RETURN FALSE;

WHEN OTHERS THEN

-- Handle any other errors

DBMS\_OUTPUT.PUT\_LINE('Error checking balance: ' || SQLERRM);

RETURN FALSE;

END;

/

**Exercise 5:**

(SCENARIO 1)

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

-- Set LastModified to the current date

:NEW.LastModified := SYSDATE;

END;

/

(SCENARIO 2)

--Creating an Auditlog table

CREATE TABLE AuditLog (

AuditID NUMBER PRIMARY KEY,

TransactionID NUMBER,

AuditDate DATE,

Action VARCHAR2(50),

Details VARCHAR2(255)

);

--LogTransaction trigger

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

-- Insert a record into the AuditLog table

INSERT INTO AuditLog (AuditID, TransactionID, AuditDate, Action, Details)

VALUES (

AuditLog\_SEQ.NEXTVAL, -- Assuming there's a sequence named AuditLog\_SEQ for generating IDs

:NEW.TransactionID,

SYSDATE,

'INSERT',

'Transaction amount: ' || :NEW.Amount || ', Type: ' || :NEW.TransactionType

);

END;

/

(SCENATIO 3)

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_Balance NUMBER;

BEGIN

-- Retrieve the balance of the account

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

-- Check if the transaction type is 'Withdrawal'

IF :NEW.TransactionType = 'Withdrawal' THEN

-- Ensure the withdrawal amount does not exceed the account balance

IF :NEW.Amount > v\_Balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal amount exceeds account balance.');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

-- Ensure the deposit amount is positive

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be greater than zero.');

END IF;

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'Invalid transaction type.');

END IF;

END;

/

**Exercise 6:**

(SCENARIO 1)

DECLARE

CURSOR c\_Transactions IS

SELECT c.CustomerID, c.Name, t.TransactionID, 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 EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE)

ORDER BY c.CustomerID, t.TransactionDate;

v\_CustomerID Customers.CustomerID%TYPE;

v\_CustomerName Customers.Name%TYPE;

v\_TransactionID Transactions.TransactionID%TYPE;

v\_TransactionDate Transactions.TransactionDate%TYPE;

v\_Amount Transactions.Amount%TYPE;

v\_TransactionType Transactions.TransactionType%TYPE;

BEGIN

FOR r IN c\_Transactions LOOP

v\_CustomerID := r.CustomerID;

v\_CustomerName := r.Name;

v\_TransactionID := r.TransactionID;

v\_TransactionDate := r.TransactionDate;

v\_Amount := r.Amount;

v\_TransactionType := r.TransactionType;

-- Print statement (or could be stored in a table, file, etc.)

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_CustomerID);

DBMS\_OUTPUT.PUT\_LINE('Customer Name: ' || v\_CustomerName);

DBMS\_OUTPUT.PUT\_LINE('Transaction ID: ' || v\_TransactionID);

DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || TO\_CHAR(v\_TransactionDate, 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Amount: ' || v\_Amount);

DBMS\_OUTPUT.PUT\_LINE('Transaction Type: ' || v\_TransactionType);

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

END LOOP;

END;

/

(SCENARIO 2)

DECLARE

CURSOR c\_Accounts IS

SELECT AccountID, Balance

FROM Accounts;

v\_AccountID Accounts.AccountID%TYPE;

v\_Balance Accounts.Balance%TYPE;

v\_AnnualFee NUMBER := 50; -- Example annual maintenance fee

BEGIN

FOR r IN c\_Accounts LOOP

v\_AccountID := r.AccountID;

v\_Balance := r.Balance;

-- Deduct the annual maintenance fee

UPDATE Accounts

SET Balance = Balance - v\_AnnualFee

WHERE AccountID = v\_AccountID;

-- Print a confirmation message

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || v\_AccountID || ' - Fee applied. New Balance: ' || (v\_Balance - v\_AnnualFee));

END LOOP;

COMMIT;

END;

/

(SCENARIO 3)

DECLARE

CURSOR c\_Loans IS

SELECT LoanID, InterestRate

FROM Loans;

v\_LoanID Loans.LoanID%TYPE;

v\_OldInterestRate Loans.InterestRate%TYPE;

v\_NewInterestRate Loans.InterestRate%TYPE;

BEGIN

FOR r IN c\_Loans LOOP

v\_LoanID := r.LoanID;

v\_OldInterestRate := r.InterestRate;

-- Apply new interest rate policy (example: increase all rates by 1%)

v\_NewInterestRate := v\_OldInterestRate + 1;

-- Update the interest rate

UPDATE Loans

SET InterestRate = v\_NewInterestRate

WHERE LoanID = v\_LoanID;

-- Print a confirmation message

DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || v\_LoanID || ' - Old Interest Rate: ' || v\_OldInterestRate || ', New Interest Rate: ' || v\_NewInterestRate);

END LOOP;

COMMIT;

END;

/

**Exercise 7:**

(SCENARIO 1)

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

);

PROCEDURE UpdateCustomer (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

);

FUNCTION GetCustomerBalance (

p\_CustomerID IN NUMBER

) RETURN NUMBER;

END CustomerManagement;

/

--Package Body (CustomerManagement.pkb)

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

) IS

BEGIN

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

VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID already exists.');

WHEN OTHERS THEN

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

ROLLBACK;

END;

PROCEDURE UpdateCustomer (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

) IS

BEGIN

UPDATE Customers

SET Name = p\_Name,

DOB = p\_DOB,

Balance = p\_Balance,

LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer not found.');

WHEN OTHERS THEN

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

ROLLBACK;

END;

FUNCTION GetCustomerBalance (

p\_CustomerID IN NUMBER

) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance

FROM Customers

WHERE CustomerID = p\_CustomerID;

RETURN v\_Balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer not found.');

RETURN NULL;

END;

END CustomerManagement;

/

(SCENARIO 2)

--Package Specification (EmployeeManagement.pks)

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee (

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2,

p\_HireDate IN DATE

);

PROCEDURE UpdateEmployee (

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2,

p\_HireDate IN DATE

);

FUNCTION CalculateAnnualSalary (

p\_EmployeeID IN NUMBER

) RETURN NUMBER;

END EmployeeManagement;

/

--Package Body (EmployeeManagement.pkb)

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee (

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2,

p\_HireDate IN DATE

) IS

BEGIN

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

VALUES (p\_EmployeeID, p\_Name, p\_Position, p\_Salary, p\_Department, p\_HireDate);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID already exists.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error hiring employee: ' || SQLERRM);

ROLLBACK;

END;

PROCEDURE UpdateEmployee (

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2,

p\_HireDate IN DATE

) IS

BEGIN

UPDATE Employees

SET Name = p\_Name,

Position = p\_Position,

Salary = p\_Salary,

Department = p\_Department,

HireDate = p\_HireDate

WHERE EmployeeID = p\_EmployeeID;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee not found.');

WHEN OTHERS THEN

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

ROLLBACK;

END;

FUNCTION CalculateAnnualSalary (

p\_EmployeeID IN NUMBER

) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

SELECT Salary INTO v\_Salary

FROM Employees

WHERE EmployeeID = p\_EmployeeID;

RETURN v\_Salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee not found.');

RETURN NULL;

END;

END EmployeeManagement;

/

(SCENARIO 3)

--Package Specification (AccountOperations.pks)

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount (

p\_AccountID IN NUMBER,

p\_CustomerID IN NUMBER,

p\_AccountType IN VARCHAR2,

p\_Balance IN NUMBER

);

PROCEDURE CloseAccount (

p\_AccountID IN NUMBER

);

FUNCTION GetTotalBalance (

p\_CustomerID IN NUMBER

) RETURN NUMBER;

END AccountOperations;

/

--Package Body (AccountOperations.pkb)

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount (

p\_AccountID IN NUMBER,

p\_CustomerID IN NUMBER,

p\_AccountType IN VARCHAR2,

p\_Balance IN NUMBER

) IS

BEGIN

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

VALUES (p\_AccountID, p\_CustomerID, p\_AccountType, p\_Balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account ID already exists.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error opening account: ' || SQLERRM);

ROLLBACK;

END;

PROCEDURE CloseAccount (

p\_AccountID IN NUMBER

) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_AccountID;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error closing account: ' || SQLERRM);

ROLLBACK;

END;

FUNCTION GetTotalBalance (

p\_CustomerID IN NUMBER

) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_TotalBalance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN NVL(v\_TotalBalance, 0);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer not found or no accounts.');

RETURN 0;

END;

END AccountOperations;

/