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

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

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

VALUES (2, 'Jane Smith', 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, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

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

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

SELECT \* FROM Customers;

SELECT \* FROM Accounts ;

SELECT \* FROM Transactions ;

SELECT \* FROM Loans ;

SELECT \* FROM Employees ;

-- exercise 1

--scenario 1

DECLARE

CURSOR customer\_cursor IS

SELECT Customers.CustomerID, Customers.DOB, Loans.LoanID, Loans.InterestRate

FROM Customers

JOIN Loans ON Customers.CustomerID = Loans.CustomerID;

v\_age NUMBER;

BEGIN

FOR record IN customer\_cursor LOOP

-- Calculate age

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

-- Check if customer is above 60 years old

IF v\_age > 60 THEN

-- Apply 1% discount to the loan interest rate

UPDATE Loans

SET InterestRate = record.InterestRate - 1

WHERE LoanID = record.LoanID;

END IF;

END LOOP;

COMMIT;

END;

SELECT \* FROM Customers;

SELECT \* FROM Loans;

--scenario 2

ALTER TABLE Customers ADD IsVIP CHAR(1) DEFAULT 'N' CHECK (IsVIP IN ('Y', 'N'));

UPDATE Customers SET IsVIP = 'N';

COMMIT;

BEGIN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID IN (

SELECT CustomerID

FROM Accounts

WHERE Balance > 10000

);

COMMIT;

END;

SELECT \* FROM Customers;

--scenario 3

DECLARE

CURSOR loan\_cursor IS

SELECT Customers.Name, Loans.LoanID, Loans.EndDate

FROM Loans

JOIN Customers ON Loans.CustomerID = Customers.CustomerID

WHERE Loans.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR record IN loan\_cursor

LOOP

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

' has a loan (ID: ' || record.LoanID ||

') due on ' || TO\_CHAR(record.EndDate, 'YYYY-MM-DD') ||

'. Please contact them.');

END LOOP;

END;

--exercise 2

--scenario 1

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id NUMBER,

p\_to\_account\_id NUMBER,

p\_amount NUMBER

) IS

insufficient\_funds EXCEPTION;

v\_from\_balance NUMBER;

v\_to\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_from\_balance FROM Accounts WHERE AccountID = p\_from\_account\_id FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in the from account.');

WHEN OTHERS THEN

ROLLBACK;

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

END SafeTransferFunds;

--scenario 2

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id NUMBER,

p\_percentage NUMBER

) IS

employee\_not\_found EXCEPTION;

v\_current\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_current\_salary FROM Employees WHERE EmployeeID = p\_employee\_id;

UPDATE Employees SET Salary = Salary + (Salary \* p\_percentage / 100) WHERE EmployeeID = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END UpdateSalary;

--scenario 3

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

) IS

customer\_exists EXCEPTION;

BEGIN

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

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END AddNewCustomer;

--exercise 3

--scenario 1

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END ProcessMonthlyInterest;

--scenario 2

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department VARCHAR2,

p\_bonus\_percentage NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END UpdateEmployeeBonus;

--scenario 3

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id NUMBER,

p\_to\_account\_id NUMBER,

p\_amount NUMBER

) IS

insufficient\_funds EXCEPTION;

v\_from\_balance NUMBER;

SELECT Balance INTO v\_from\_balance FROM Accounts WHERE AccountID = p\_from\_account\_id FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in the from account.');

WHEN OTHERS THEN

ROLLBACK;

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

END TransferFunds;

--exercise 4

--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;

EXCEPTION

WHEN OTHERS THEN

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

RETURN NULL;

END CalculateAge;

--scenario 2

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_loan\_duration\_years NUMBER

)

RETURN NUMBER IS

v\_monthly\_installment NUMBER;

v\_monthly\_rate NUMBER;

v\_total\_months NUMBER;

BEGIN

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

v\_total\_months := p\_loan\_duration\_years \* 12;

v\_monthly\_installment := p\_loan\_amount \* v\_monthly\_rate / (1 - POWER(1 + v\_monthly\_rate, -v\_total\_months));

RETURN v\_monthly\_installment;

EXCEPTION

WHEN OTHERS THEN

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

RETURN NULL;

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

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

RETURN FALSE;

WHEN OTHERS THEN

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

RETURN FALSE;

END HasSufficientBalance;

--exercise 5

--scenario 1

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

--scenario 2

CREATE TABLE AuditLog (

AuditID NUMBER PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

LogDate DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (AuditID, 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 TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

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;

IF :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds for withdrawal.');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

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

END IF;

ELSE

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

END IF;

END;

--exercise 6

--scenario 1

DECLARE

CURSOR cur\_monthly\_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);

v\_customer\_id Customers.CustomerID%TYPE;

v\_name Customers.Name%TYPE;

v\_transaction\_id Transactions.TransactionID%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\_id, v\_transaction\_date, v\_amount, v\_transaction\_type;

EXIT WHEN cur\_monthly\_transactions%NOTFOUND;

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

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

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

DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || v\_transaction\_date);

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

DBMS\_OUTPUT.PUT\_LINE('Transaction 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 := 50;

BEGIN

OPEN cur\_accounts;

LOOP

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

EXIT WHEN cur\_accounts%NOTFOUND;

IF v\_balance >= v\_annual\_fee THEN

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee,

LastModified = SYSDATE

WHERE AccountID = v\_account\_id;

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

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance for Account ID: ' || v\_account\_id);

END IF;

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 Loans.InterestRate%TYPE;

BEGIN

OPEN cur\_loans;

LOOP

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

EXIT WHEN cur\_loans%NOTFOUND;

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

UPDATE Loans

SET InterestRate = v\_new\_interest\_rate,

LastModified = SYSDATE

WHERE LoanID = v\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Interest rate updated for Loan ID: ' || v\_loan\_id || ' to ' || v\_new\_interest\_rate || '%');

END LOOP;

CLOSE cur\_loans;

COMMIT;

END;

--exercise 7

--scenario 1

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

PROCEDURE UpdateCustomerDetails(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END CustomerManagement;

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) IS

BEGIN

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

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Customer with ID ' || p\_CustomerID || ' already exists.');

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) IS

BEGIN

UPDATE Customers

SET Name = p\_Name,

DOB = p\_DOB,

Balance = p\_Balance,

LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Customer with ID ' || p\_CustomerID || ' not found.');

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(p\_CustomerID 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('Customer with ID ' || p\_CustomerID || ' not found.');

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

--scenario 2

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2, p\_HireDate DATE);

PROCEDURE UpdateEmployeeDetails(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2);

FUNCTION CalculateAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER;

END EmployeeManagement;

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2, p\_HireDate 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);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Employee with ID ' || p\_EmployeeID || ' already exists.');

END HireEmployee;

PROCEDURE UpdateEmployeeDetails(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2) IS

BEGIN

UPDATE Employees

SET Name = p\_Name,

Position = p\_Position,

Salary = p\_Salary,

Department = p\_Department,

HireDate = HireDate

WHERE EmployeeID = p\_EmployeeID;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Employee with ID ' || p\_EmployeeID || ' not found.');

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

SELECT Salary \* 12 INTO v\_Salary

FROM Employees

WHERE EmployeeID = p\_EmployeeID;

RETURN v\_Salary;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Employee with ID ' || p\_EmployeeID || ' not found.');

RETURN NULL;

END CalculateAnnualSalary;

END EmployeeManagement;

--scenario 3

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_AccountType VARCHAR2, p\_Balance NUMBER);

PROCEDURE CloseAccount(p\_AccountID NUMBER);

FUNCTION GetTotalCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END AccountOperations;

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_AccountType VARCHAR2, p\_Balance NUMBER) IS

BEGIN

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

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Account with ID ' || p\_AccountID || ' already exists.');

END OpenAccount;

PROCEDURE CloseAccount(p\_AccountID NUMBER) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_AccountID;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Account with ID ' || p\_AccountID || ' not found.');

END CloseAccount;

FUNCTION GetTotalCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_TotalBalance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN v\_TotalBalance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Customer with ID ' || p\_CustomerID || ' not found.');

RETURN NULL;

END GetTotalCustomerBalance;

END AccountOperations;