PLSQL

**Exercise1: Control Structures**

**Sc1:**

Declare

custid Customers.customerid%type;

custage number;

intrate number;

cursor cu is

select c.customerid, trunc((c.lastmodified-c.dob)/365) as age, l.InterestRate

from customers c join loans l on c.customerid=l.customerid;

begin

open cu;

loop

fetch cu into custid, custage, intrate;

exit when cu%notfound;

if custage>60 then

update loans

set interestrate=intrate\*0.99

where customerid=custid;

end if;

end loop;

close cu;

End;

**Sc2:**

ALTER TABLE Customers

ADD IsVIP NUMBER(1) DEFAULT 0;

BEGIN

FOR customer\_record IN (

SELECT CustomerID, Balance

FROM Customers

) LOOP

IF customer\_record.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 1

WHERE CustomerID = customer\_record.CustomerID;

END IF;

END LOOP;

END;

**Sc3:**

declare

custid Customers.customerid%type;

custname Customers.name%type;

daysleft number;

cursor cu is

select c.customerid, c.name, trunc((l.enddate-sysdate)/365) as ed

from customers c join loans l on c.customerid=l.customerid;

Begin

open cu;

loop

fetch cu into custid, custname, daysleft;

exit when cu%notfound;

if daysleft<=30 then

dbms\_output.put\_line('Mr/Mrs '||custname||' with a customerid '||custid||' Loan has to be paid in '||daysleft||' days');

end if;

end loop;

close cu;

end;

**Exercise 2: Error Handling**

**Sc1:**

create or replace procedure SafeTransferFunds

(fromac In Number, toac In Number, amount IN number) is

inf\_excp EXCEPTION;

Begin

select Balance from Accounts

where AccountID=fromac and amount<Balance;

if SQL%NOTFOUND THEN

RAISE inf\_excp;

End if;

update Balance from Accounts

set Balance=Balance-amount

Where AccountID=fromac;

update Balance from Accounts

set Balance=Balance+amount

Where AccountID=toac;

EXCEPTION

WHEN inf\_excp THEN

rollback;

DBMS\_OUTPUT.PUT\_LINE('Insufficient Fund in acount no.'||fromac);

end;

BEGIN

SafeTransferFunds(1, 2, 10000);

END;

**Sc2:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_EmployeeID IN Employees.EmployeeID%TYPE,

p\_Percentage IN NUMBER

) IS

v\_OldSalary Employees.Salary%TYPE;

BEGIN

SELECT Salary

INTO v\_OldSalary

FROM Employees

WHERE EmployeeID = p\_EmployeeID;

UPDATE Employees

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

WHERE EmployeeID = p\_EmployeeID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

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

END UpdateSalary;

/

BEGIN

UpdateSalary(p\_EmployeeID => 1, p\_Percentage => 10);

END;

/

BEGIN

UpdateSalary(123,10);

END;

/

**Sc3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_CustomerID IN Customers.CustomerID%TYPE,

p\_Name IN Customers.Name%TYPE,

p\_DOB IN Customers.DOB%TYPE,

p\_Balance IN Customers.Balance%TYPE,

p\_LastModified IN Customers.LastModified%TYPE

) IS

BEGIN

BEGIN

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END;

END AddNewCustomer;

/

BEGIN

AddNewCustomer(

p\_CustomerID => 1001,

p\_Name => 'John Doe',

p\_DOB => TO\_DATE('1980-01-15', 'YYYY-MM-DD'),

p\_Balance => 5000,

p\_LastModified => SYSDATE

);

END;

/

BEGIN

AddNewCustomer(

p\_CustomerID => 1002,

p\_Name => 'Johnnnnn Doe',

p\_DOB => TO\_DATE('1980-01-15', 'YYYY-MM-DD'),

p\_Balance => 500053,

p\_LastModified => SYSDATE

);

END;

/

BEGIN

AddNewCustomer(

p\_CustomerID => 1001,

p\_Name => 'John Doe',

p\_DOB => TO\_DATE('1980-01-15', 'YYYY-MM-DD'),

p\_Balance => 503300,

p\_LastModified => SYSDATE

);

END;

/

**Exercise 3: Stored Procedures**

**Sc1:**

create or replace procedure ProcessMonthlyInterest Is

Begin

update Accounts

set Balance=Balance\*1.01

where AccountType='Savings';

end;

Begin

ProcessMonthlyInterest;

End;

**Sc2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN Employees.Department%TYPE,

p\_BonusPercentage IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_Department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salaries updated with a bonus of ' || p\_BonusPercentage || '% for department ' || p\_Department || '.');

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END UpdateEmployeeBonus;

/

SELECT \* FROM EMPLOYEES;

BEGIN

UpdateEmployeeBonus(p\_Department => 'Sales', p\_BonusPercentage => 10);

END;

/

SELECT \* FROM EMPLOYEES;

**Sc3**:

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_SourceAccountID IN Accounts.AccountID%TYPE,

p\_DestinationAccountID IN Accounts.AccountID%TYPE,

p\_Amount IN NUMBER

) IS

v\_SourceBalance Accounts.Balance%TYPE;

BEGIN

SELECT Balance INTO v\_SourceBalance

FROM Accounts

WHERE AccountID = p\_SourceAccountID;

IF v\_SourceBalance < p\_Amount THEN

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

END IF;

BEGIN

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_SourceAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_DestinationAccountID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully.');

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK; -- Rollback the transaction in case of an error

END;

END TransferFunds;

/

BEGIN

TransferFunds(p\_SourceAccountID => 1, p\_DestinationAccountID => 2, p\_Amount => 500);

END;

/

BEGIN

TransferFunds(p\_SourceAccountID => 1, p\_DestinationAccountID => 2, p\_Amount => 50000000000);

END;

/

**Exercise 4: Functions**

**Sc1:**

create or replace function CalculateAge(dob date) return number Is

age number;

Begin

age:=Trunc((sysdate-dob)/365);

return age;

end;

declare

dob date;

age number;

Begin

select dob into dob from Customers where customerid=1;

age:=CalculateAge(dob);

DBMS\_OUTPUT.PUT\_LINE('The Age of The Customer is '||age);

End;

**Sc2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_LoanAmount IN NUMBER,

p\_InterestRate IN NUMBER,

p\_LoanDurationYears IN NUMBER

) RETURN NUMBER IS

v\_MonthlyInstallment NUMBER;

v\_MonthlyInterestRate NUMBER;

v\_NumPayments NUMBER;

BEGIN

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

v\_NumPayments := p\_LoanDurationYears \* 12;

IF v\_MonthlyInterestRate > 0 THEN

v\_MonthlyInstallment := (p\_LoanAmount \* v\_MonthlyInterestRate) /

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

ELSE

-- If interest rate is 0, simply divide loan amount by number of payments

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

END IF;

RETURN v\_MonthlyInstallment;

END CalculateMonthlyInstallment;

/

DECLARE

v\_MonthlyInstallment NUMBER;

BEGIN

v\_MonthlyInstallment := CalculateMonthlyInstallment(

p\_LoanAmount => 10000,

p\_InterestRate => 5,

p\_LoanDurationYears => 10

);

DBMS\_OUTPUT.PUT\_LINE('The monthly installment is: ' || v\_MonthlyInstallment);

END;

/

**Sc3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_AccountID IN Accounts.AccountID%TYPE,

p\_Amount IN NUMBER

) RETURN BOOLEAN IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = p\_AccountID;

IF v\_Balance >= p\_Amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

WHEN OTHERS THEN

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

RETURN FALSE;

END HasSufficientBalance;

/

DECLARE

v\_HasSufficient BOOLEAN;

BEGIN

v\_HasSufficient := HasSufficientBalance(p\_AccountID => 222, p\_Amount => 500);

IF v\_HasSufficient THEN

DBMS\_OUTPUT.PUT\_LINE('The account has sufficient balance.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('The account does not have sufficient balance.');

END IF;

END;

/

**Exercise 5: Triggers**

**Sc1:**

create or replace trigger UpdateCustomerLastModified

before update on customers

for each row

Begin

:NEW.LastModified := SYSDATE;

DBMS\_OUTPUT.PUT\_LINE('Customer record is updated');

end;

update customers

set balance=1000

where Customerid=1;

**Sc2:**

CREATE TABLE AuditLog (

TransactionID NUMBER PRIMARY KEY,

AuditDate DATE,

Action VARCHAR2(50),

FOREIGN KEY (TransactionID) REFERENCES Transactions(TransactionID)

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, AuditDate, Action)

VALUES (

:NEW.TransactionID,

SYSDATE,

'INSERT'

);

END;

/

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (11, 8, TO\_DATE('2024-07-28', 'YYYY-MM-DD'), 500.00, 'Withdrawal');

SELECT \* FROM AuditLog;

**Sc3:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' THEN

IF :NEW.Amount > v\_balance THEN

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

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. Must be DEPOSIT or WITHDRAWAL.');

END IF;

END;

/

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (11, 5, TO\_DATE('2024-07-20', 'YYYY-MM-DD'), 500999999.00, 'WITHDRAWAL')

**Exercise 6: Cursors**

**Sc1:**

Declare

custname Customers.Name%type;

transdate date;

amount number;

transtype Transactions.TransactionType%type;

cursor GenerateMonthlyStatements is

select 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 to\_char(sysdate,'yyyy-mm')=to\_char(t.TransactionDate,'yyyy-mm');

Begin

open GenerateMonthlyStatements;

loop

fetch GenerateMonthlyStatements

into custname, transdate, amount, transtype;

exit when GenerateMonthlyStatements%notfound;

DBMS\_OUTPUT.PUT\_LINE(custname||' made a '||transtype||' of amount '||amount||'on '||transdate);

end loop;

close GenerateMonthlyStatements;

End;

**Sc2:**

DECLARE

CURSOR ApplyAnnualFee IS

SELECT AccountID, Balance

FROM Accounts;

v\_Account ApplyAnnualFee%ROWTYPE;

v\_AnnualFee NUMBER := 100;

BEGIN

OPEN ApplyAnnualFee;

LOOP

FETCH ApplyAnnualFee INTO v\_Account;

EXIT WHEN ApplyAnnualFee%NOTFOUND;

IF v\_Account.Balance >= v\_AnnualFee THEN

UPDATE Accounts

SET Balance = Balance - v\_AnnualFee

WHERE AccountID = v\_Account.AccountID;

ELSE

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

END IF;

END LOOP;

CLOSE ApplyAnnualFee;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

END;

/

**Sc3:**

DECLARE

CURSOR UpdateLoanInterestRates IS

SELECT LoanID, InterestRate

FROM Loans;

v\_Loan UpdateLoanInterestRates%ROWTYPE;

v\_NewInterestRate NUMBER := 0.5;

BEGIN

OPEN UpdateLoanInterestRates;

LOOP

FETCH UpdateLoanInterestRates INTO v\_Loan;

EXIT WHEN UpdateLoanInterestRates%NOTFOUND;

UPDATE Loans

SET InterestRate = InterestRate + v\_NewInterestRate

WHERE LoanID = v\_Loan.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Updated Loan ID: ' || v\_Loan.LoanID ||

', New Interest Rate: ' || (v\_Loan.InterestRate + v\_NewInterestRate));

END LOOP;

CLOSE UpdateLoanInterestRates;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

END;

/

**Exercise 7: Packages**

**Sc1:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

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

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

FUNCTION GetCustomerBalance(p\_CustomerID IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

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

BEGIN

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('Customer with ID ' || p\_CustomerID || ' already exists.');

WHEN OTHERS THEN

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

END;

END AddNewCustomer;

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

BEGIN

BEGIN

UPDATE Customers

SET Name = p\_Name, Balance = p\_Balance, LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

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

END;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(p\_CustomerID IN NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

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 || ' does not exist.');

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END;

END GetCustomerBalance;

END CustomerManagement;

/

BEGIN

CustomerManagement.AddNewCustomer(

p\_CustomerID => 123,

p\_Name => 'Johnyyyy Doeeeeee',

p\_DOB => TO\_DATE('1990-01-01', 'YYYY-MM-DD'),

p\_Balance => 10002

);

END;

/

SELECT \* FROM CUSTOMERS;

BEGIN

CustomerManagement.UpdateCustomerDetails(

p\_CustomerID => 123,

p\_Name => 'John Doe Updated',

p\_Balance => 15002

);

END;

/

SELECT \* FROM CUSTOMERS;

DECLARE

v\_Balance NUMBER;

BEGIN

v\_Balance := CustomerManagement.GetCustomerBalance(p\_CustomerID => 123);

DBMS\_OUTPUT.PUT\_LINE('Customer Balance: ' || v\_Balance);

END;

/

**Sc2:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

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

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

FUNCTION CalculateAnnualSalary(p\_EmployeeID IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

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

BEGIN

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END;

END HireNewEmployee;

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

BEGIN

BEGIN

UPDATE Employees

SET Name = p\_Name, Position = p\_Position, Salary = p\_Salary, Department = p\_Department

WHERE EmployeeID = p\_EmployeeID;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

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

END;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(p\_EmployeeID IN NUMBER) RETURN NUMBER IS

v\_Salary NUMBER;

v\_AnnualSalary NUMBER;

BEGIN

BEGIN

SELECT Salary INTO v\_Salary

FROM Employees

WHERE EmployeeID = p\_EmployeeID;

v\_AnnualSalary := v\_Salary \* 12;

RETURN v\_AnnualSalary;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END;

END CalculateAnnualSalary;

END EmployeeManagement;

/

BEGIN

EmployeeManagement.HireNewEmployee(

p\_EmployeeID => 101,

p\_Name => 'Alice Smith',

p\_Position => 'Software Engineer',

p\_Salary => 5000,

p\_Department => 'IT'

);

END;

/

SELECT \* FROM EMPLOYEES;

BEGIN

EmployeeManagement.UpdateEmployeeDetails(

p\_EmployeeID => 101,

p\_Name => 'Alice Johnson',

p\_Position => 'Senior Software Engineer',

p\_Salary => 6000,

p\_Department => 'IT'

);

END;

/

SELECT \* FROM EMPLOYEES;

DECLARE

v\_AnnualSalary NUMBER;

BEGIN

v\_AnnualSalary := EmployeeManagement.CalculateAnnualSalary(p\_EmployeeID => 101);

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || v\_AnnualSalary);

END;

/

**Sc3:**

CREATE OR REPLACE PACKAGE AccountOperations AS

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

PROCEDURE CloseAccount(p\_AccountID IN NUMBER);

FUNCTION GetTotalCustomerBalance(p\_CustomerID IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

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

BEGIN

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('Account with ID ' || p\_AccountID || ' already exists.');

WHEN OTHERS THEN

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

END;

END OpenNewAccount;

PROCEDURE CloseAccount(p\_AccountID IN NUMBER) IS

BEGIN

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_AccountID;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

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

END;

END CloseAccount;

FUNCTION GetTotalCustomerBalance(p\_CustomerID IN NUMBER) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

BEGIN

SELECT SUM(Balance) INTO v\_TotalBalance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

IF v\_TotalBalance IS NULL THEN

v\_TotalBalance := 0;

END IF;

RETURN v\_TotalBalance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

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

RETURN NULL;

END;

END GetTotalCustomerBalance;

END AccountOperations;

/

BEGIN

AccountOperations.OpenNewAccount(

p\_AccountID => 2001,

p\_CustomerID => 1,

p\_AccountType => 'Savings',

p\_Balance => 5000

);

END;

/

SELECT \* FROM ACCOUNTS;

BEGIN

AccountOperations.CloseAccount(p\_AccountID => 21);

END;

/

SELECT \* FROM ACCOUNTS;

DECLARE

v\_TotalBalance NUMBER;

BEGIN

v\_TotalBalance := AccountOperations.GetTotalCustomerBalance(p\_CustomerID => 1);

DBMS\_OUTPUT.PUT\_LINE('Total Balance for Customer 1: ' || v\_TotalBalance);

END;

/