**SOLUTIONS**

**Exercise 1: Control Structures**

**Scenario 1:** Apply a discount to loan interest rates for customers above 60 years old

DECLARE

CURSOR c\_customers IS

SELECT CustomerID, TRUNC(MONTHS\_BETWEEN(SYSDATE, DOB) / 12) AS Age

FROM Customers;

v\_customerID Customers.CustomerID%TYPE;

v\_age NUMBER;

BEGIN

FOR c IN c\_customers LOOP

v\_customerID := c.CustomerID;

v\_age := c.Age;

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = v\_customerID;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 2:** Promote customers to VIP status based on their balance

DECLARE

CURSOR c\_customers IS

SELECT CustomerID, Balance

FROM Customers;

v\_customerID Customers.CustomerID%TYPE;

v\_balance Customers.Balance%TYPE;

BEGIN

FOR c IN c\_customers LOOP

v\_customerID := c.CustomerID;

v\_balance := c.Balance;

IF v\_balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = v\_customerID;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 3:** Send reminders to customers whose loans are due within the next 30 days

DECLARE

CURSOR c\_loans 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;

v\_loanID Loans.LoanID%TYPE;

v\_customerID Loans.CustomerID%TYPE;

v\_customerName Customers.Name%TYPE;

v\_endDate Loans.EndDate%TYPE;

BEGIN

FOR c IN c\_loans LOOP

v\_loanID := c.LoanID;

v\_customerID := c.CustomerID;

v\_customerName := c.Name;

v\_endDate := c.EndDate;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ' || v\_customerName || ' (ID: ' || v\_customerID ||

') has a loan (ID: ' || v\_loanID || ') due on ' || TO\_CHAR(v\_endDate, 'YYYY-MM-DD'));

END LOOP;

END;

/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_fromAccountID IN NUMBER,

p\_toAccountID IN NUMBER,

p\_amount IN NUMBER

) AS

insufficient\_funds EXCEPTION;

v\_fromBalance NUMBER;

BEGIN

SELECT Balance INTO v\_fromBalance

FROM Accounts

WHERE AccountID = p\_fromAccountID

FOR UPDATE;

IF v\_fromBalance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

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 insufficient\_funds THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END SafeTransferFunds;

/

**Scenario 2:** Manage errors when updating employee salaries.

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employeeID IN NUMBER,

p\_percentage IN NUMBER

) AS

employee\_not\_found EXCEPTION;

v\_salary Employees.Salary%TYPE;

BEGIN

SELECT Salary INTO v\_salary

FROM Employees

WHERE EmployeeID = p\_employeeID

FOR UPDATE;

UPDATE Employees

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

WHERE EmployeeID = p\_employeeID;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE employee\_not\_found;

WHEN employee\_not\_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;

/

**Scenario 3:** Ensure data integrity when adding a new customer.

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customerID IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) AS

customer\_exists EXCEPTION;

BEGIN

-- Insert a new customer

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

RAISE customer\_exists;

WHEN customer\_exists THEN

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

WHEN OTHERS THEN

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

END AddNewCustomer;

/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

v\_interestRate CONSTANT NUMBER := 0.01;

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* v\_interestRate)

WHERE AccountType = 'Savings';

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

END ProcessMonthlyInterest;

/

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonusPercentage IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

END UpdateEmployeeBonus;

/

**Scenario 3:** Customers should be able to transfer funds between their accounts.

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_fromAccountID IN NUMBER,

p\_toAccountID IN NUMBER,

p\_amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

v\_fromBalance NUMBER;

BEGIN

SELECT Balance INTO v\_fromBalance

FROM Accounts

WHERE AccountID = p\_fromAccountID

FOR UPDATE;

IF v\_fromBalance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

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 insufficient\_funds THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END TransferFunds;

/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Exercise 4: Functions**

**Scenario 1:** Calculate the age of customers for eligibility checks.

CREATE OR REPLACE FUNCTION CalculateAge(

p\_dob IN 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:** The bank needs to compute the monthly installment for a loan.

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loanAmount IN NUMBER,

p\_interestRate IN NUMBER,

p\_durationYears IN NUMBER

) RETURN NUMBER IS

v\_monthlyInstallment NUMBER;

v\_ratePerMonth NUMBER;

v\_numberOfMonths NUMBER;

BEGIN

v\_ratePerMonth := p\_interestRate / 100 / 12;

v\_numberOfMonths := p\_durationYears \* 12;

v\_monthlyInstallment := (p\_loanAmount \* v\_ratePerMonth) /

(1 - POWER(1 + v\_ratePerMonth, -v\_numberOfMonths));

RETURN v\_monthlyInstallment;

EXCEPTION

WHEN OTHERS THEN

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

RETURN NULL;

END CalculateMonthlyInstallment;

/

**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_accountID IN NUMBER,

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

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

RETURN FALSE;

WHEN OTHERS THEN

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

RETURN FALSE;

END HasSufficientBalance;

/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Exercise 5: Triggers**

**Scenario 1:** Automatically update the last modified date when a customer's record is updated.

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

/

**Scenario 2:** Maintain an audit log for all transactions.

CREATE TABLE AuditLog (

AuditID NUMBER PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

LogTimestamp DATE

);

#Creating trigger

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (AuditID, TransactionID, AccountID, TransactionDate, Amount, TransactionType, LogTimestamp)

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

END LogTransaction;

/

**Scenario 3:** Enforce business rules on deposits and withdrawals.

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

FOR UPDATE;

IF :NEW.TransactionType = 'Withdrawal' THEN

IF v\_balance < :NEW.Amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error: Insufficient balance for withdrawal.');

END IF;

END IF;

IF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

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

END IF;

END IF;

END CheckTransactionRules;

/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Exercise 6: Cursors**

**Scenario 1:** Generate monthly statements for all customers.

DECLARE

CURSOR c\_transactions IS

SELECT t.TransactionID, t.AccountID, t.TransactionDate, t.Amount, t.TransactionType, a.CustomerID, c.Name

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

v\_transactionID Transactions.TransactionID%TYPE;

v\_accountID Transactions.AccountID%TYPE;

v\_transactionDate Transactions.TransactionDate%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_transactionType Transactions.TransactionType%TYPE;

v\_customerID Accounts.CustomerID%TYPE;

v\_customerName Customers.Name%TYPE;

BEGIN

OPEN c\_transactions;

LOOP

FETCH c\_transactions INTO v\_transactionID, v\_accountID, v\_transactionDate, v\_amount, v\_transactionType, v\_customerID, v\_customerName;

EXIT WHEN c\_transactions%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customerID || ', Name: ' || v\_customerName);

DBMS\_OUTPUT.PUT\_LINE('Transaction ID: ' || v\_transactionID || ', Date: ' || v\_transactionDate ||

', Amount: ' || v\_amount || ', Type: ' || v\_transactionType);

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

END LOOP;

CLOSE c\_transactions;

END;

/

**Scenario 2:** Apply annual fee to all accounts.

DECLARE

CURSOR c\_accounts IS

SELECT AccountID, Balance

FROM Accounts;

v\_accountID Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_annualFee CONSTANT NUMBER := 50;

BEGIN

OPEN c\_accounts;

LOOP

FETCH c\_accounts INTO v\_accountID, v\_balance;

EXIT WHEN c\_accounts%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - v\_annualFee

WHERE AccountID = v\_accountID;

END LOOP;

CLOSE c\_accounts;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

**Scenario 3:** Update the interest rate for all loans based on a new policy.

DECLARE

CURSOR c\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

v\_loanID Loans.LoanID%TYPE;

v\_interestRate Loans.InterestRate%TYPE;

v\_newInterestRate CONSTANT NUMBER := 4.5;

BEGIN

OPEN c\_loans;

LOOP

FETCH c\_loans INTO v\_loanID, v\_interestRate;

EXIT WHEN c\_loans%NOTFOUND;

UPDATE Loans

SET InterestRate = v\_newInterestRate

WHERE LoanID = v\_loanID;

END LOOP;

CLOSE c\_loans;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

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;

/

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 with ID ' || p\_customerID || ' already exists.');

WHEN OTHERS THEN

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

END AddCustomer;

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 with ID ' || p\_customerID || ' does not exist.');

WHEN OTHERS THEN

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

END UpdateCustomer;

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 with ID ' || p\_customerID || ' does not exist.');

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2:** Create a package to manage employee data.

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;

/

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 with ID ' || p\_employeeID || ' already exists.');

WHEN OTHERS THEN

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

END HireEmployee;

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 with ID ' || p\_employeeID || ' does not exist.');

WHEN OTHERS THEN

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

END UpdateEmployee;

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 with ID ' || p\_employeeID || ' does not exist.');

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3:** Group all account-related operations into a package.

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;

/

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 with ID ' || p\_accountID || ' already exists.');

WHEN OTHERS THEN

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

END OpenAccount;

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 with ID ' || p\_accountID || ' does not exist.');

WHEN OTHERS THEN

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

END CloseAccount;

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 v\_totalBalance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: No accounts found for Customer ID ' || p\_customerID || '.');

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END GetTotalBalance;

END AccountOperations;

/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Schema to be Created**

*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*

*);*

**Example Scripts for Sample Data Insertion**

*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'));*