**Exercise 1: Control Structures**

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

BEGIN

FOR rec IN (SELECT CustomerID, LoanID, InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM DOB) > 60) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

END LOOP;

END;

/

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

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

END LOOP;

END;

/

ALTER TABLE Customers ADD (IsVIP VARCHAR2(5));

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

BEGIN

FOR rec IN (SELECT CustomerID, LoanID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ' || rec.CustomerID || ', your loan ' || rec.LoanID || ' is due on ' || rec.EndDate || '.');

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1: SafeTransferFunds procedure**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_FromAccountID NUMBER, p\_ToAccountID NUMBER, p\_Amount NUMBER) AS

insufficient\_funds EXCEPTION;

PRAGMA EXCEPTION\_INIT(insufficient\_funds, -20001);

BEGIN

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

IF SQL%NOTFOUND OR SQL%ROWCOUNT = 0 THEN

RAISE insufficient\_funds;

END IF;

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: UpdateSalary procedure**

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_EmployeeID NUMBER, p\_Percentage NUMBER) AS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_EmployeeID;

IF SQL%NOTFOUND THEN

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

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END UpdateSalary;

/

**Scenario 3: AddNewCustomer procedure**

CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) AS

customer\_exists EXCEPTION;

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

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END AddNewCustomer;

/

**Exercise 3: Stored Procedures**

**Scenario 1: ProcessMonthlyInterest procedure**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

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

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END ProcessMonthlyInterest;

/

**Scenario 2: UpdateEmployeeBonus procedure**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_Department VARCHAR2, p\_BonusPercentage NUMBER) AS

BEGIN

UPDATE Employees

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

WHERE Department = p\_Department;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END UpdateEmployeeBonus;

/

**Scenario 3: TransferFunds procedure**

CREATE OR REPLACE PROCEDURE TransferFunds(p\_FromAccountID NUMBER, p\_ToAccountID NUMBER, p\_Amount NUMBER) AS

BEGIN

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

IF SQL%NOTFOUND OR SQL%ROWCOUNT = 0 THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END TransferFunds;

/

**Exercise 4: Functions**

**Scenario 1: CalculateAge function**

CREATE OR REPLACE FUNCTION CalculateAge(p\_DOB DATE) RETURN NUMBER AS

v\_Age NUMBER;

BEGIN

v\_Age := TRUNC((SYSDATE - p\_DOB) / 365.25);

RETURN v\_Age;

END CalculateAge;

/

**Scenario 2: CalculateMonthlyInstallment function**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_LoanAmount NUMBER, p\_InterestRate NUMBER, p\_DurationYears NUMBER) RETURN NUMBER AS

v\_MonthlyInstallment NUMBER;

BEGIN

v\_MonthlyInstallment := p\_LoanAmount \* (p\_InterestRate / 1200) / (1 - POWER(1 + p\_InterestRate / 1200, -p\_DurationYears \* 12));

RETURN v\_MonthlyInstallment;

END CalculateMonthlyInstallment;

/

**Scenario 3: HasSufficientBalance function**

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_AccountID NUMBER, p\_Amount NUMBER) RETURN BOOLEAN AS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_AccountID;

RETURN v\_Balance >= p\_Amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END HasSufficientBalance;

/

**Exercise 5: Triggers**

**Scenario 1: UpdateCustomerLastModified trigger**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Scenario 2: LogTransaction trigger**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

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

VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType);

END;

/

**Scenario 3: CheckTransactionRules trigger**

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' AND :NEW.Amount > v\_Balance THEN

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

ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

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

END IF;

END;

/

**Exercise 6: Cursors**

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

DECLARE

CURSOR cur\_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);

BEGIN

FOR rec IN cur\_Transactions LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec.Name || ' (' || rec.CustomerID || ')');

DBMS\_OUTPUT.PUT\_LINE(' Date: ' || rec.TransactionDate || ', Amount: ' || rec.Amount || ', Type: ' || rec.TransactionType);

END LOOP;

END;

/

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

DECLARE

CURSOR cur\_Accounts IS

SELECT AccountID, Balance

FROM Accounts;

v\_AnnualFee NUMBER := 50; -- Define the annual fee amount

BEGIN

FOR rec IN cur\_Accounts LOOP

UPDATE Accounts

SET Balance = Balance - v\_AnnualFee

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

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

DECLARE

CURSOR cur\_Loans IS

SELECT LoanID, InterestRate

FROM Loans;

v\_NewInterestRate NUMBER := 6; -- Define the new interest rate

BEGIN

FOR rec IN cur\_Loans LOOP

UPDATE Loans

SET InterestRate = v\_NewInterestRate

WHERE LoanID = rec.LoanID;

END LOOP;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

**Exercise 7: Packages**

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

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);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

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;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

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

RETURN 0;

END GetCustomerBalance;

END CustomerManagement;

/

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

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);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

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

WHERE EmployeeID = p\_EmployeeID;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(p\_EmployeeID 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

RETURN 0;

END CalculateAnnualSalary;

END EmployeeManagement;

/

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

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 GetTotalBalance(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);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END OpenAccount;

PROCEDURE CloseAccount(p\_AccountID NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_AccountID;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END CloseAccount;

FUNCTION GetTotalBalance(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

RETURN 0;

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*

*);*

**VALUES TO BE INSERTED**

*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'));*