**Exercise 1: Control Structures**

**Scenario 1**:

DECLARE

CURSOR c\_customers IS

SELECT CustomerID, SYSDATE - DOB AS age FROM Customers WHERE SYSDATE - DOB > 60\*365;

v\_customer\_id Customers.CustomerID%TYPE;

BEGIN

FOR r\_customer IN c\_customers LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = r\_customer.CustomerID;

END LOOP;

END;

/

**Scenario 2**:

BEGIN

FOR r\_customer IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = TRUE

WHERE CustomerID = r\_customer.CustomerID;

END LOOP;

END;

/

**Scenario 3**:

DECLARE

CURSOR c\_loans IS

SELECT CustomerID FROM Loans WHERE EndDate <= SYSDATE + 30;

v\_customer\_id Loans.CustomerID%TYPE;

BEGIN

FOR r\_loan IN c\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Your loan is due in the next 30 days, CustomerID: ' || r\_loan.CustomerID);

END LOOP;

END;

/

**Exercise 2: Error Handling**

**Scenario 1**:

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN Accounts.AccountID%TYPE,

p\_to\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

BEGIN

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

IF SQL%NOTFOUND OR SQL%ROWCOUNT = 0 THEN

RAISE insufficient\_funds;

END IF;

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.');

WHEN OTHERS THEN

ROLLBACK;

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

END SafeTransferFunds;

/

**Scenario 2**:

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN Employees.EmployeeID%TYPE,

p\_percentage IN NUMBER

) IS

employee\_not\_found EXCEPTION;

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

IF SQL%NOTFOUND OR SQL%ROWCOUNT = 0 THEN

RAISE employee\_not\_found;

END IF;

COMMIT;

EXCEPTION

WHEN employee\_not\_found THEN

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

WHEN OTHERS THEN

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

END UpdateSalary;

/

**Scenario 3**:

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN Customers.CustomerID%TYPE,

p\_name IN Customers.Name%TYPE,

p\_dob IN Customers.DOB%TYPE,

p\_balance IN Customers.Balance%TYPE

) IS

duplicate\_customer 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;

RAISE duplicate\_customer;

WHEN duplicate\_customer THEN

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

WHEN OTHERS THEN

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

END AddNewCustomer;

/

**Exercise 3: Stored Procedures**

**Scenario 1**:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

END ProcessMonthlyInterest;

/

**Scenario 2**:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN Employees.Department%TYPE,

p\_bonus\_percentage IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

END UpdateEmployeeBonus;

/

**Scenario 3**:

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN Accounts.AccountID%TYPE,

p\_to\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

BEGIN

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

IF SQL%NOTFOUND OR SQL%ROWCOUNT = 0 THEN

RAISE insufficient\_funds;

END IF;

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.');

WHEN OTHERS THEN

ROLLBACK;

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

END TransferFunds;

/

**Exercise 4: Functions**

**Scenario 1**:

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob IN DATE

) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END CalculateAge;

/

**Scenario 2**:

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount IN NUMBER,

p\_interest\_rate IN NUMBER,

p\_loan\_duration\_years IN NUMBER

) RETURN NUMBER IS

v\_monthly\_installment NUMBER;

BEGIN

v\_monthly\_installment := (p\_loan\_amount \* p\_interest\_rate / 1200) / (1 - POWER(1 + p\_interest\_rate / 1200, -p\_loan\_duration\_years \* 12));

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

**Scenario 3**:

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id 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\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END HasSufficientBalance;

/

**Exercise 5: Triggers**

**Scenario 1**:

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

/

**Scenario 2**:

-- log\_transaction.sql

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 LogTransaction;

/

**Scenario 3**:

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > (SELECT Balance FROM Accounts WHERE AccountID = :NEW.AccountID) THEN

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

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

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

END IF;

END CheckTransactionRules;

/

**Exercise 6: Cursors**

**Scenario 1**:

DECLARE

CURSOR c\_transactions IS

SELECT \* FROM Transactions WHERE TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE);

v\_transaction Transactions%ROWTYPE;

BEGIN

OPEN c\_transactions;

LOOP

FETCH c\_transactions INTO v\_transaction;

EXIT WHEN c\_transactions%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('CustomerID: ' || v\_transaction.AccountID || ' Amount: ' || v\_transaction.Amount);

END LOOP;

CLOSE c\_transactions;

END;

/

**Scenario 2**:

DECLARE

CURSOR c\_accounts IS

SELECT AccountID, Balance FROM Accounts;

v\_account Accounts%ROWTYPE;

BEGIN

OPEN c\_accounts;

LOOP

FETCH c\_accounts INTO v\_account;

EXIT WHEN c\_accounts%NOTFOUND;

UPDATE Accounts

SET Balance = v\_account.Balance - 50

WHERE AccountID = v\_account.AccountID;

END LOOP;

CLOSE c\_accounts;

COMMIT;

END;

/

**Scenario 3**:

DECLARE

CURSOR c\_loans IS

SELECT LoanID, InterestRate FROM Loans;

v\_loan Loans%ROWTYPE;

BEGIN

OPEN c\_loans;

LOOP

FETCH c\_loans INTO v\_loan;

EXIT WHEN c\_loans%NOTFOUND;

UPDATE Loans

SET InterestRate = v\_loan.InterestRate + 1

WHERE LoanID = v\_loan.LoanID;

END LOOP;

CLOSE c\_loans;

COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1**:

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id IN Customers.CustomerID%TYPE, p\_name IN Customers.Name%TYPE, p\_dob IN Customers.DOB%TYPE, p\_balance IN Customers.Balance%TYPE);

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN Customers.CustomerID%TYPE, p\_name IN Customers.Name%TYPE, p\_dob IN Customers.DOB%TYPE, p\_balance IN Customers.Balance%TYPE);

FUNCTION GetCustomerBalance(p\_customer\_id IN Customers.CustomerID%TYPE) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id IN Customers.CustomerID%TYPE, p\_name IN Customers.Name%TYPE, p\_dob IN Customers.DOB%TYPE, p\_balance IN Customers.Balance%TYPE) IS

BEGIN

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

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN Customers.CustomerID%TYPE, p\_name IN Customers.Name%TYPE, p\_dob IN Customers.DOB%TYPE, p\_balance IN Customers.Balance%TYPE) IS

BEGIN

UPDATE Customers

SET Name = p\_name, DOB = p\_dob, Balance = p\_balance, LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(p\_customer\_id IN Customers.CustomerID%TYPE) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Customers

WHERE CustomerID = p\_customer\_id;

RETURN v\_balance;

END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2**:

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN Employees.EmployeeID%TYPE, p\_name IN Employees.Name%TYPE, p\_position IN Employees.Position%TYPE, p\_salary IN Employees.Salary%TYPE, p\_department IN Employees.Department%TYPE, p\_hiredate IN DATE);

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN Employees.EmployeeID%TYPE, p\_name IN Employees.Name%TYPE, p\_position IN Employees.Position%TYPE, p\_salary IN Employees.Salary%TYPE, p\_department IN Employees.Department%TYPE);

FUNCTION CalculateAnnualSalary(p\_employee\_id IN Employees.EmployeeID%TYPE) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN Employees.EmployeeID%TYPE, p\_name IN Employees.Name%TYPE, p\_position IN Employees.Position%TYPE, p\_salary IN Employees.Salary%TYPE, p\_department IN Employees.Department%TYPE, p\_hiredate IN DATE) IS

BEGIN

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

VALUES (p\_employee\_id, p\_name, p\_position, p\_salary, p\_department, p\_hiredate);

END HireEmployee;

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN Employees.EmployeeID%TYPE, p\_name IN Employees.Name%TYPE, p\_position IN Employees.Position%TYPE, p\_salary IN Employees.Salary%TYPE, p\_department IN Employees.Department%TYPE) IS

BEGIN

UPDATE Employees

SET Name = p\_name, Position = p\_position, Salary = p\_salary, Department = p\_department

WHERE EmployeeID = p\_employee\_id;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(p\_employee\_id IN Employees.EmployeeID%TYPE) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary \* 12 INTO v\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

RETURN v\_salary;

END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3**:

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id IN Accounts.AccountID%TYPE, p\_customer\_id IN Accounts.CustomerID%TYPE, p\_account\_type IN Accounts.AccountType%TYPE, p\_balance IN Accounts.Balance%TYPE);

PROCEDURE CloseAccount(p\_account\_id IN Accounts.AccountID%TYPE);

FUNCTION GetTotalBalance(p\_customer\_id IN Accounts.CustomerID%TYPE) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id IN Accounts.AccountID%TYPE, p\_customer\_id IN Accounts.CustomerID%TYPE, p\_account\_type IN Accounts.AccountType%TYPE, p\_balance IN Accounts.Balance%TYPE) IS

BEGIN

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

VALUES (p\_account\_id, p\_customer\_id, p\_account\_type, p\_balance, SYSDATE);

END OpenAccount;

PROCEDURE CloseAccount(p\_account\_id IN Accounts.AccountID%TYPE) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_account\_id;

END CloseAccount;

FUNCTION GetTotalBalance(p\_customer\_id IN Accounts.CustomerID%TYPE) RETURN NUMBER IS

v\_total\_balance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_total\_balance

FROM Accounts

WHERE CustomerID = p\_customer\_id;

RETURN v\_total\_balance;

END GetTotalBalance;

END AccountOperations;

/