**PL/SQL programming (Additional Exercise)**

**Exercise 2: Error Handling**

**Scenario 1: Safe fund transfer**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amt NUMBER) IS

e\_insufficient\_funds EXCEPTION;

from\_balance NUMBER;

BEGIN

SELECT Balance INTO from\_balance FROM Accounts WHERE AccountID = p\_from FOR UPDATE;

IF from\_balance < p\_amt THEN

RAISE e\_insufficient\_funds;

END IF;

UPDATE Accounts SET Balance = Balance - p\_amt WHERE AccountID = p\_from;

UPDATE Accounts SET Balance = Balance + p\_amt WHERE AccountID = p\_to;

COMMIT;

EXCEPTION

WHEN e\_insufficient\_funds THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds.');

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

**Scenario 2: Update salary**

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_empid NUMBER, p\_pct NUMBER) IS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_empid;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Employee not found.');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

**Scenario 3: Add new customer**

CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer already exists.');

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

**Exercise 4: Functions**

**Scenario 1: Calculate age**

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE) RETURN NUMBER IS

BEGIN

RETURN TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

END;

/

**Scenario 2: Calculate monthly installment**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_amt NUMBER, p\_rate NUMBER, p\_years NUMBER) RETURN NUMBER IS

r NUMBER := p\_rate / (12 \* 100);

n NUMBER := p\_years \* 12;

BEGIN

RETURN (p\_amt \* r \* POWER(1 + r, n)) / (POWER(1 + r, n) - 1);

END;

/

**Scenario 3: Check sufficient balance**

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_acctid NUMBER, p\_amt NUMBER) RETURN BOOLEAN IS

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Accounts WHERE AccountID = p\_acctid;

RETURN bal >= p\_amt;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

**Exercise 5: Triggers**

**Scenario 1: Update last modified**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Scenario 2: Log transactions**

CREATE TABLE AuditLog (

LogID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,

AccountID NUMBER,

Action VARCHAR2(50),

Timestamp DATE DEFAULT SYSDATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog(AccountID, Action)

VALUES (:NEW.AccountID, 'Transaction Added');

END;

/

**Scenario 3: Enforce transaction rules**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Accounts WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > bal THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Withdrawal exceeds balance.');

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

RAISE\_APPLICATION\_ERROR(-20004, 'Deposit must be positive.');

END IF;

END;

/

**Exercise 6: Cursors**

**Scenario 1: Generate statements**

DECLARE

CURSOR trans\_cursor IS

SELECT A.CustomerID, T.\*

FROM Transactions T

JOIN Accounts A ON T.AccountID = A.AccountID

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

v\_name VARCHAR2(100);

BEGIN

FOR txn IN trans\_cursor LOOP

SELECT Name INTO v\_name FROM Customers WHERE CustomerID = txn.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || v\_name || ', Amount: ' || txn.Amount || ', Type: ' || txn.TransactionType);

END LOOP;

END;

/

**Scenario 2: Apply annual fee**

DECLARE

CURSOR acct\_cursor IS SELECT AccountID FROM Accounts;

BEGIN

FOR acct IN acct\_cursor LOOP

UPDATE Accounts

SET Balance = Balance - 100

WHERE AccountID = acct.AccountID;

END LOOP;

COMMIT;

END;

/

**Scenario 3: Update loan interest**

DECLARE

CURSOR loan\_cursor IS SELECT LoanID, InterestRate FROM Loans;

BEGIN

FOR loan IN loan\_cursor LOOP

UPDATE Loans

SET InterestRate = loan.InterestRate + 0.5

WHERE LoanID = loan.LoanID;

END LOOP;

COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1: Customer Management**

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2);

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

BEGIN

INSERT INTO Customers VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE, NULL);

END;

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2) IS

BEGIN

UPDATE Customers SET Name = p\_name WHERE CustomerID = p\_id;

END;

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER IS

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Customers WHERE CustomerID = p\_id;

RETURN bal;

END;

END;

/

**Scenario 2: Employee Management**

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2, p\_hire DATE);

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_pos VARCHAR2);

FUNCTION AnnualSalary(p\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2, p\_hire DATE) IS

BEGIN

INSERT INTO Employees VALUES(p\_id, p\_name, p\_pos, p\_sal, p\_dept, p\_hire);

END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_pos VARCHAR2) IS

BEGIN

UPDATE Employees SET Position = p\_pos WHERE EmployeeID = p\_id;

END;

FUNCTION AnnualSalary(p\_id NUMBER) RETURN NUMBER IS

sal NUMBER;

BEGIN

SELECT Salary INTO sal FROM Employees WHERE EmployeeID = p\_id;

RETURN sal \* 12;

END;

END;

/

**Scenario 3: Account Operations**

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(p\_id NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_bal NUMBER);

PROCEDURE CloseAccount(p\_id NUMBER);

FUNCTION TotalBalance(p\_custid NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(p\_id NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_bal NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES(p\_id, p\_custid, p\_type, p\_bal, SYSDATE);

END;

PROCEDURE CloseAccount(p\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_id;

END;

FUNCTION TotalBalance(p\_custid NUMBER) RETURN NUMBER IS

total NUMBER;

BEGIN

SELECT SUM(Balance) INTO total FROM Accounts WHERE CustomerID = p\_custid;

RETURN NVL(total, 0);

END;

END;

/