**PL/SQL**

**Exercise 1: Control Structures**

**Scenario 1:**

DECLARE

v\_customer\_id Customers.CustomerID%TYPE;

v\_dob Customers.DOB%TYPE;

v\_age NUMBER;

v\_current\_rate Loans.InterestRate%TYPE;

BEGIN

FOR cur IN (SELECT CustomerID, DOB FROM Customers) LOOP

v\_customer\_id := cur.CustomerID;

v\_dob := cur.DOB;

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

IF v\_age > 60 THEN

FOR loan\_cur IN (SELECT LoanID, InterestRate FROM Loans WHERE CustomerID = v\_customer\_id) LOOP

v\_current\_rate := loan\_cur.InterestRate;

UPDATE Loans

SET InterestRate = v\_current\_rate - 1

WHERE LoanID = loan\_cur.LoanID;

END LOOP;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 2:**

ALTER TABLE Customers ADD (IsVIP CHAR(1) DEFAULT 'N');

DECLARE

v\_customer\_id Customers.CustomerID%TYPE;

v\_balance Customers.Balance%TYPE;

BEGIN

FOR cur IN (SELECT CustomerID, Balance FROM Customers) LOOP

v\_customer\_id := cur.CustomerID;

v\_balance := cur.Balance;

IF v\_balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = v\_customer\_id;

ELSE

UPDATE Customers

SET IsVIP = 'N'

WHERE CustomerID = v\_customer\_id;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 3:**

DECLARE

v\_customer\_id Loans.CustomerID%TYPE;

v\_customer\_name Customers.Name%TYPE;

v\_loan\_id Loans.LoanID%TYPE;

v\_end\_date Loans.EndDate%TYPE;

v\_due\_date DATE := SYSDATE + 30;

BEGIN

FOR cur IN (

SELECT l.LoanID, l.CustomerID, l.EndDate, c.Name

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND v\_due\_date

) LOOP

v\_loan\_id := cur.LoanID;

v\_customer\_id := cur.CustomerID;

v\_customer\_name := cur.Name;

v\_end\_date := cur.EndDate;

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Customer ' || v\_customer\_name ||

' (ID: ' || v\_customer\_id ||

') has a loan (ID: ' || v\_loan\_id ||

') due on ' || TO\_CHAR(v\_end\_date, 'YYYY-MM-DD') ||

'. Please remind them.'

);

END LOOP;

COMMIT;

END;

/

**Exercise 2: Error Handling**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_source\_account IN NUMBER,

p\_target\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_source\_balance Accounts.Balance%TYPE;

v\_target\_balance Accounts.Balance%TYPE;

insufficient\_funds EXCEPTION;

invalid\_amount EXCEPTION;

account\_not\_found EXCEPTION;

BEGIN

IF p\_amount <= 0 THEN

RAISE invalid\_amount;

END IF;

SELECT Balance INTO v\_source\_balance

FROM Accounts

WHERE AccountID = p\_source\_account

FOR UPDATE;

IF v\_source\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

SELECT Balance INTO v\_target\_balance

FROM Accounts

WHERE AccountID = p\_target\_account

FOR UPDATE;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_source\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_target\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer completed successfully.');

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK;

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

WHEN invalid\_amount THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Transfer amount must be greater than zero.');

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: One or both account IDs are invalid.');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred. ' || SQLERRM);

END SafeTransferFunds;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) IS

v\_current\_salary Employees.Salary%TYPE;

BEGIN

IF p\_percentage <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Percentage must be greater than zero.');

END IF;

SELECT Salary INTO v\_current\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id

FOR UPDATE;

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

COMMIT;

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

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred. ' || SQLERRM);

END UpdateSalary;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

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

COMMIT;

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: A customer with this ID already exists.');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred. ' || SQLERRM);

END;

END AddNewCustomer;

/

**Exercise 3: Stored Procedures**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all savings accounts.');

END ProcessMonthlyInterest;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonuses updated for employees in department ' || p\_department || '.');

END UpdateEmployeeBonus;

/

**Scenario 3:**

*CREATE OR REPLACE PROCEDURE TransferFunds(*

*p\_source\_account IN NUMBER,*

*p\_target\_account IN NUMBER,*

*p\_amount IN NUMBER*

*) IS*

*v\_source\_balance Accounts.Balance%TYPE;*

*BEGIN*

*IF p\_amount <= 0 THEN*

*RAISE\_APPLICATION\_ERROR(-20001, 'Amount must be greater than zero.');*

*END IF;*

*SELECT Balance INTO v\_source\_balance*

*FROM Accounts*

*WHERE AccountID = p\_source\_account*

*FOR UPDATE;*

*IF v\_source\_balance < p\_amount THEN*

*RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient funds in source account.');*

*END IF;*

*UPDATE Accounts*

*SET Balance = Balance - p\_amount*

*WHERE AccountID = p\_source\_account;*

*UPDATE Accounts*

*SET Balance = Balance + p\_amount*

*WHERE AccountID = p\_target\_account;*

*COMMIT;*

*DBMS\_OUTPUT.PUT\_LINE('Transfer completed successfully.');*

*EXCEPTION*

*WHEN NO\_DATA\_FOUND THEN*

*ROLLBACK;*

*DBMS\_OUTPUT.PUT\_LINE('Error: One or both account IDs are invalid.');*

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred. ' || 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\_annual\_interest\_rate IN NUMBER,

p\_loan\_duration\_years IN NUMBER

) RETURN NUMBER IS

v\_monthly\_interest\_rate NUMBER;

v\_total\_payments NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

v\_monthly\_interest\_rate := p\_annual\_interest\_rate / 12 / 100;

v\_total\_payments := p\_loan\_duration\_years \* 12;

IF v\_monthly\_interest\_rate = 0 THEN

v\_monthly\_installment := p\_loan\_amount / v\_total\_payments;

ELSE

v\_monthly\_installment := p\_loan\_amount \*

(v\_monthly\_interest\_rate \* POWER(1 + v\_monthly\_interest\_rate, v\_total\_payments)) /

(POWER(1 + v\_monthly\_interest\_rate, v\_total\_payments) - 1);

END IF;

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

**Scenario 3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_account\_id IN NUMBER,

p\_amount IN NUMBER

) RETURN BOOLEAN IS

v\_balance Accounts.Balance%TYPE;

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;

WHEN OTHERS 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:**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (

LogID,

TransactionID,

AccountID,

TransactionDate,

Amount,

TransactionType,

LogDate

) VALUES (

AuditLog\_seq.NEXTVAL

:NEW.TransactionID,

:NEW.AccountID,

:NEW.TransactionDate,

:NEW.Amount,

:NEW.TransactionType,

SYSDATE

);

END LogTransaction;

**/**

**Scenario 3:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance Accounts.Balance%TYPE;

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID

FOR UPDATE;

IF v\_balance < :NEW.Amount THEN

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

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.');

END IF;

END CheckTransactionRules;

/

**Exercise 6: Cursors**

**Scenario 1:**

DECLARE

CURSOR cur\_transactions IS

SELECT c.CustomerID, c.Name, a.AccountID, a.AccountType, 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 EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE)

ORDER BY c.CustomerID, a.AccountID, t.TransactionDate;

TYPE transaction\_record IS RECORD (

CustomerID Customers.CustomerID%TYPE,

Name Customers.Name%TYPE,

AccountID Accounts.AccountID%TYPE,

AccountType Accounts.AccountType%TYPE,

TransactionDate Transactions.TransactionDate%TYPE,

Amount Transactions.Amount%TYPE,

TransactionType Transactions.TransactionType%TYPE

);

v\_transaction transaction\_record;

BEGIN

OPEN cur\_transactions;

LOOP

FETCH cur\_transactions INTO v\_transaction;

EXIT WHEN cur\_transactions%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_transaction.CustomerID);

DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_transaction.Name);

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || v\_transaction.AccountID);

DBMS\_OUTPUT.PUT\_LINE('Account Type: ' || v\_transaction.AccountType);

DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || v\_transaction.TransactionDate);

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

DBMS\_OUTPUT.PUT\_LINE('Transaction Type: ' || v\_transaction.TransactionType);

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

END LOOP;

CLOSE cur\_transactions;

END;

/

**Scenario 2:**

DECLARE

CURSOR cur\_accounts IS

SELECT AccountID, Balance

FROM Accounts;

v\_fee NUMBER := 50; -- Annual maintenance fee

v\_account\_balance Accounts.Balance%TYPE;

BEGIN

OPEN cur\_accounts;

LOOP

FETCH cur\_accounts INTO v\_account\_balance;

EXIT WHEN cur\_accounts%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - v\_fee

WHERE AccountID = v\_account\_balance.AccountID;

END LOOP;

CLOSE cur\_accounts;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Annual maintenance fee applied to all accounts.');

END;

/

**Scenario 3:**

DECLARE

CURSOR cur\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

v\_new\_interest\_rate NUMBER;

v\_loan\_id Loans.LoanID%TYPE;

v\_interest\_rate\_increase NUMBER := 0.5 / 100;

BEGIN

OPEN cur\_loans;

LOOP

FETCH cur\_loans INTO v\_loan\_id, v\_new\_interest\_rate;

EXIT WHEN cur\_loans%NOTFOUND;

UPDATE Loans

SET InterestRate = v\_new\_interest\_rate + v\_interest\_rate\_increase

WHERE LoanID = v\_loan\_id;

END LOOP;

CLOSE cur\_loans;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Interest rates updated for all loans based on the new policy.');

END;

/

**Exercise 7: Packages**

**Scenario 1:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

);

PROCEDURE UpdateCustomerDetails(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

);

FUNCTION GetCustomerBalance(

p\_customer\_id IN NUMBER

) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Customer ID already exists.');

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails(

p\_customer\_id 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\_customer\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Customer ID not found.');

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(

p\_customer\_id IN NUMBER

) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Customers

WHERE CustomerID = p\_customer\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireNewEmployee(

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_position IN VARCHAR2,

p\_salary IN NUMBER,

p\_department IN VARCHAR2,

p\_hire\_date IN DATE

);

PROCEDURE UpdateEmployeeDetails(

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_position IN VARCHAR2,

p\_salary IN NUMBER,

p\_department IN VARCHAR2

);

FUNCTION CalculateAnnualSalary(

p\_employee\_id IN NUMBER

) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireNewEmployee(

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_position IN VARCHAR2,

p\_salary IN NUMBER,

p\_department IN VARCHAR2,

p\_hire\_date 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\_hire\_date);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Employee ID already exists.');

END HireNewEmployee;

PROCEDURE UpdateEmployeeDetails(

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_position IN VARCHAR2,

p\_salary IN NUMBER,

p\_department IN VARCHAR2

) IS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID not found.');

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(

p\_employee\_id IN NUMBER

) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

RETURN v\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount(

p\_account\_id IN NUMBER,

p\_customer\_id IN NUMBER,

p\_account\_type IN VARCHAR2,

p\_balance IN NUMBER

);

PROCEDURE CloseAccount(

p\_account\_id IN NUMBER

);

FUNCTION GetTotalBalance(

p\_customer\_id IN NUMBER

) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenNewAccount(

p\_account\_id IN NUMBER,

p\_customer\_id IN NUMBER,

p\_account\_type IN VARCHAR2,

p\_balance IN NUMBER

) IS

BEGIN

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

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Account ID already exists.');

END OpenNewAccount;

PROCEDURE CloseAccount(

p\_account\_id IN NUMBER

) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_account\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Account ID not found.');

END CloseAccount;

FUNCTION GetTotalBalance(

p\_customer\_id IN NUMBER

) 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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END GetTotalBalance;

END AccountOperations;

/