**Week 2 PL/SQL Assignment**

**Exercise 1: Control Structures**

**Scenario 1:**

DECLARE

CURSOR cust\_cursor IS

SELECT CustomerID, DOB

FROM Customers;

cust\_age NUMBER;

cust\_id NUMBER;

loan\_id NUMBER;

new\_interest\_rate NUMBER;

BEGIN

FOR cust\_rec IN cust\_cursor LOOP

cust\_id := cust\_rec.CustomerID;

cust\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB) / 12);

IF cust\_age > 60 THEN

FOR loan\_rec IN (

SELECT LoanID, InterestRate

FROM Loans

WHERE CustomerID = cust\_id

) LOOP

loan\_id := loan\_rec.LoanID;

new\_interest\_rate := loan\_rec.InterestRate - 1;

UPDATE Loans

SET InterestRate = new\_interest\_rate

WHERE LoanID = loan\_id;

END LOOP;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 2:**

DECLARE

CURSOR cust\_cursor IS

SELECT CustomerID, Balance

FROM Customers;

cust\_id NUMBER;

BEGIN

FOR cust\_rec IN cust\_cursor LOOP

cust\_id := cust\_rec.CustomerID;

IF cust\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'T'

WHERE CustomerID = cust\_id;

ELSE

UPDATE Customers

SET IsVIP = 'F'

WHERE CustomerID = cust\_id;

END IF;

END LOOP;

COMMIT;

END;

/

**Scenario 3:**

DECLARE

CURSOR loan\_cursor IS

SELECT Loans.LoanID, Loans.CustomerID, Customers.Name, Loans.EndDate

FROM Loans

JOIN Customers ON Loans.CustomerID = Customers.CustomerID

WHERE Loans.EndDate <= SYSDATE + 30;

BEGIN

FOR loan\_rec IN loan\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || loan\_rec.Name || ', your loan (ID: ' || loan\_rec.LoanID || ') is due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD') || '.');

END LOOP;

END;

/

**Exercise 2:**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id NUMBER,

p\_to\_account\_id NUMBER,

p\_amount NUMBER

)

IS

v\_from\_balance NUMBER;

v\_to\_balance NUMBER;

BEGIN

-- Check if the source account has sufficient funds

SELECT Balance INTO v\_from\_balance FROM Accounts WHERE AccountID = p\_from\_account\_id;

IF v\_from\_balance < p\_amount THEN

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

END IF;

-- Deduct amount from source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

-- Add amount to destination account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

-- Log the error message

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

END SafeTransferFunds;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id NUMBER,

p\_percentage NUMBER

)

IS

v\_current\_salary NUMBER;

BEGIN

-- Check if the employee exists and get the current salary

SELECT Salary INTO v\_current\_salary FROM Employees WHERE EmployeeID = p\_employee\_id;

-- Update the salary by the given percentage

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Handle the exception if employee ID does not exist

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

WHEN OTHERS THEN

-- Handle other exceptions

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

END UpdateSalary;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

)

IS

BEGIN

-- Insert the new customer

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

-- Handle the exception if customer ID already exists

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

WHEN OTHERS THEN

-- Handle other exceptions

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

END AddNewCustomer;

/

**Exercise 3:**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

-- Update the balance of all savings accounts by applying 1% interest

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

END ProcessMonthlyInterest;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department VARCHAR2,

p\_bonus\_percentage NUMBER

)

IS

BEGIN

-- Update the salary of employees in the given department by adding the bonus percentage

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Handle other exceptions

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

END UpdateEmployeeBonus;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id NUMBER,

p\_to\_account\_id NUMBER,

p\_amount NUMBER

)

IS

v\_from\_balance NUMBER;

BEGIN

-- Check if the source account has sufficient funds

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id;

IF v\_from\_balance < p\_amount THEN

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

END IF;

-- Deduct amount from source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

-- Add amount to destination account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

-- Log the error message

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

END TransferFunds;

/

**Exercise 4:**

**Scenario 1:**

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob DATE

) RETURN NUMBER

IS

v\_age NUMBER;

BEGIN

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

RETURN v\_age;

END CalculateAge;

/

**Scenario 2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_loan\_duration\_years NUMBER

) RETURN NUMBER

IS

v\_monthly\_interest\_rate NUMBER;

v\_number\_of\_payments NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

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

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

IF v\_monthly\_interest\_rate = 0 THEN

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

ELSE

v\_monthly\_installment := p\_loan\_amount \* v\_monthly\_interest\_rate /

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

END IF;

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

**Scenario 3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN

IS

v\_balance NUMBER;

BEGIN

-- Fetch the current balance for the specified account

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

-- Check if the balance is sufficient

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

WHEN OTHERS THEN

RETURN FALSE;

END HasSufficientBalance;

/

**Exercise 5:**

**Scenario 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

/

**Scenario 2:**

CREATE TABLE AuditLog (

AuditID NUMBER PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

LogDate DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

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

VALUES (

: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 NUMBER;

BEGIN

-- Fetch the current balance for the specified account

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

-- Ensure deposits are positive

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

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

END IF;

-- Ensure withdrawals do not exceed the balance

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_balance THEN

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

END IF;

END CheckTransactionRules;

/

**Exercise 6:**

**Scenario 1:**

DECLARE

CURSOR transaction\_cursor IS

SELECT c.CustomerID, c.Name, t.TransactionID, 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);

v\_customer\_id Customers.CustomerID%TYPE;

v\_customer\_name Customers.Name%TYPE;

v\_transaction\_id Transactions.TransactionID%TYPE;

v\_transaction\_date Transactions.TransactionDate%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_transaction\_type Transactions.TransactionType%TYPE;

BEGIN

OPEN transaction\_cursor;

LOOP

FETCH transaction\_cursor INTO v\_customer\_id, v\_customer\_name, v\_transaction\_id, v\_transaction\_date, v\_amount, v\_transaction\_type;

EXIT WHEN transaction\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id);

DBMS\_OUTPUT.PUT\_LINE('Customer Name: ' || v\_customer\_name);

DBMS\_OUTPUT.PUT\_LINE('Transaction ID: ' || v\_transaction\_id);

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

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

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

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

END LOOP;

CLOSE transaction\_cursor;

END;

/

**Scenario 2:**

DECLARE

CURSOR account\_cursor IS

SELECT AccountID, Balance

FROM Accounts;

v\_account\_id Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_annual\_fee CONSTANT NUMBER := 50; -- Example annual fee

BEGIN

OPEN account\_cursor;

LOOP

FETCH account\_cursor INTO v\_account\_id, v\_balance;

EXIT WHEN account\_cursor%NOTFOUND;

-- Deduct the annual fee from the balance

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee

WHERE AccountID = v\_account\_id;

-- Print the update statement

DBMS\_OUTPUT.PUT\_LINE('Annual fee of ' || v\_annual\_fee || ' applied to Account ID: ' || v\_account\_id);

END LOOP;

CLOSE account\_cursor;

COMMIT;

END;

/

**Scenario 3:**

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate

FROM Loans;

v\_loan\_id Loans.LoanID%TYPE;

v\_interest\_rate Loans.InterestRate%TYPE;

v\_new\_interest\_rate CONSTANT NUMBER := 5.5; -- Example new interest rate

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_loan\_id, v\_interest\_rate;

EXIT WHEN loan\_cursor%NOTFOUND;

-- Update the interest rate based on the new policy

UPDATE Loans

SET InterestRate = v\_new\_interest\_rate

WHERE LoanID = v\_loan\_id;

-- Print the update statement

DBMS\_OUTPUT.PUT\_LINE('Interest rate for Loan ID: ' || v\_loan\_id || ' updated to ' || v\_new\_interest\_rate);

END LOOP;

CLOSE loan\_cursor;

COMMIT;

END;

/

**Exercise 7:**

**Scenario 1:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

);

PROCEDURE UpdateCustomerDetails(

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

);

FUNCTION GetCustomerBalance(

p\_customer\_id NUMBER

) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance 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

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

WHEN OTHERS THEN

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

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails(

p\_customer\_id 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\_customer\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customer\_id || ' does not exist.');

WHEN OTHERS THEN

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

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(

p\_customer\_id 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

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customer\_id || ' does not exist.');

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireNewEmployee(

p\_employee\_id NUMBER,

p\_name VARCHAR2,

p\_position VARCHAR2,

p\_salary NUMBER,

p\_department VARCHAR2,

p\_hire\_date DATE

);

PROCEDURE UpdateEmployeeDetails(

p\_employee\_id NUMBER,

p\_name VARCHAR2,

p\_position VARCHAR2,

p\_salary NUMBER,

p\_department VARCHAR2

);

FUNCTION CalculateAnnualSalary(

p\_employee\_id NUMBER

) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireNewEmployee(

p\_employee\_id NUMBER,

p\_name VARCHAR2,

p\_position VARCHAR2,

p\_salary NUMBER,

p\_department VARCHAR2,

p\_hire\_date 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

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

WHEN OTHERS THEN

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

END HireNewEmployee;

PROCEDURE UpdateEmployeeDetails(

p\_employee\_id 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\_employee\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

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

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(

p\_employee\_id NUMBER

) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

RETURN v\_salary \* 12; -- Assuming monthly salary is stored

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount(

p\_account\_id NUMBER,

p\_customer\_id NUMBER,

p\_account\_type VARCHAR2,

p\_balance NUMBER

);

PROCEDURE CloseAccount(

p\_account\_id NUMBER

);

FUNCTION GetTotalBalance(

p\_customer\_id NUMBER

) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenNewAccount(

p\_account\_id NUMBER,

p\_customer\_id NUMBER,

p\_account\_type VARCHAR2,

p\_balance 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

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

WHEN OTHERS THEN

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

END OpenNewAccount;

PROCEDURE CloseAccount(

p\_account\_id NUMBER

) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_account\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

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

END CloseAccount;

FUNCTION GetTotalBalance(

p\_customer\_id 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

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customer\_id || ' does not have any accounts.');

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END GetTotalBalance;

END AccountOperations;

/