**Week 2 Assignment- PL/SQL**

Creation of Tables:

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

);

Insertion of Values in their respective tables:

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'));

Exercise 1: Control Structures

Scenario 1:

DECLARE

v\_customer\_id NUMBER;

v\_dob DATE;

v\_age NUMBER;

BEGIN

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

v\_customer\_id := rec.CustomerID;

v\_dob := rec.DOB;

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

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = v\_customer\_id;

END IF;

END LOOP;

END;

Scenario 2:

BEGIN

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

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

END;

Scenario 3:

BEGIN

FOR rec IN (SELECT CustomerID, LoanID, EndDate FROM Loans WHERE EndDate <= SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID || ' for customer ' || rec.CustomerID || ' is due within 30 days.');

END LOOP;

END;

Exercise 2: Error Handling

Scenario 1:

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account NUMBER,

p\_to\_account NUMBER,

p\_amount NUMBER

) IS

e\_insufficient\_funds EXCEPTION;

v\_balance NUMBER;

BEGIN

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

IF v\_balance < p\_amount THEN

RAISE e\_insufficient\_funds;

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

COMMIT;

EXCEPTION

WHEN e\_insufficient\_funds THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_account);

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END;

Scenario 2:

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id NUMBER,

p\_percentage NUMBER

) IS

e\_employee\_not\_found EXCEPTION;

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

IF SQL%NOTFOUND THEN

RAISE e\_employee\_not\_found;

END IF;

COMMIT;

EXCEPTION

WHEN e\_employee\_not\_found THEN

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

WHEN OTHERS THEN

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

ROLLBACK;

END;

Scenario 3:

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

) IS

e\_customer\_exists 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

RAISE e\_customer\_exists;

WHEN e\_customer\_exists THEN

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

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END;

Exercise 3: Stored Procedures

Scenario 1:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

END;

Scenario 2:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department VARCHAR2,

p\_bonus\_percentage NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

END;

Scenario 3:

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account NUMBER,

p\_to\_account NUMBER,

p\_amount NUMBER

) IS

v\_balance NUMBER;

BEGIN

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

IF v\_balance < p\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_account);

RETURN;

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

COMMIT;

END;

Exercise 4: Functions

Scenario 1: CREATE OR REPLACE FUNCTION CalculateAge (p\_dob DATE) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

SELECT FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12) INTO v\_age FROM dual;

RETURN v\_age;

END;

Scenario 2: CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_loan\_duration NUMBER

) RETURN NUMBER IS

v\_monthly\_installment NUMBER;

v\_monthly\_rate NUMBER;

v\_total\_months NUMBER;

BEGIN

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

v\_total\_months := p\_loan\_duration \* 12;

v\_monthly\_installment := (p\_loan\_amount \* v\_monthly\_rate) / (1 - POWER(1 + v\_monthly\_rate, -v\_total\_months));

RETURN v\_monthly\_installment;

END;

Scenario 3:

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

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

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

END;

Exercise 5: Triggers

Scenario 1:

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

Scenario 2:

**Created AuditLog Table:**

CREATE TABLE TransactionAuditLog (

AuditID NUMBER PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10)

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO TransactionAuditLog (AuditID, TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (TransactionAuditLog\_SEQ.NEXTVAL, :NEW.TransactionID, :NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType);

END;

Scenario 3:

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

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

RAISE\_APPLICATION\_ERROR(-20001, 'Deposit amount must be positive');

ELSIF :NEW.TransactionType = 'Withdrawal' THEN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

IF v\_balance < :NEW.Amount THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient funds for withdrawal');

END IF;

END IF;

END;

Exercise 6: Cursors

Scenario 1: DECLARE

CURSOR c\_transactions IS

SELECT \* FROM Transactions

WHERE EXTRACT(MONTH FROM TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM TransactionDate) = EXTRACT(YEAR FROM SYSDATE);

v\_transaction c\_transactions%ROWTYPE;

BEGIN

OPEN c\_transactions;

LOOP

FETCH c\_transactions INTO v\_transaction;

EXIT WHEN c\_transactions%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('TransactionID: ' || v\_transaction.TransactionID ||

', AccountID: ' || v\_transaction.AccountID ||

', Amount: ' || v\_transaction.Amount ||

', Type: ' || v\_transaction.TransactionType);

END LOOP;

CLOSE c\_transactions;

END;

Scenario 2:

DECLARE

CURSOR c\_accounts IS

SELECT \* FROM Accounts;

v\_account c\_accounts%ROWTYPE;

BEGIN

OPEN c\_accounts;

LOOP

FETCH c\_accounts INTO v\_account;

EXIT WHEN c\_accounts%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - 20

WHERE AccountID = v\_account.AccountID;

END LOOP;

CLOSE c\_accounts;

COMMIT;

END;

Scenario 3:

DECLARE

CURSOR c\_loans IS

SELECT \* FROM Loans;

v\_loan c\_loans%ROWTYPE;

BEGIN

OPEN c\_loans;

LOOP

FETCH c\_loans INTO v\_loan;

EXIT WHEN c\_loans%NOTFOUND;

UPDATE Loans

SET InterestRate = InterestRate + 0.5

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

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

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

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;

END;

PROCEDURE UpdateCustomerDetails(p\_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\_id;

COMMIT;

END;

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

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

RETURN v\_balance;

END;

END CustomerManagement;

/

Scenario 2:

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2, p\_hire\_date DATE);

PROCEDURE UpdateEmployeeDetails(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2);

FUNCTION CalculateAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_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\_id, p\_name, p\_position, p\_salary, p\_department, p\_hire\_date);

COMMIT;

END;

PROCEDURE UpdateEmployeeDetails(p\_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\_id;

COMMIT;

END;

FUNCTION CalculateAnnualSalary(p\_id NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

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

RETURN v\_salary \* 12;

END;

END EmployeeManagement;

/

Scenario 3:

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount(p\_id NUMBER, p\_customer\_id NUMBER, p\_account\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_id NUMBER);

FUNCTION GetTotalBalance(p\_customer\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenNewAccount(p\_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\_id, p\_customer\_id, p\_account\_type, p\_balance, SYSDATE);

COMMIT;

END;

PROCEDURE CloseAccount(p\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_id;

COMMIT;

END;

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;

END;

END AccountOperations;

/