**Week 2 Assignment- PL/SQL**

Creation of Tables:

CREATE TABLE Clients (

ClientID NUMBER PRIMARY KEY,

FullName VARCHAR2(100),

DateOfBirth DATE,

AccountBalance NUMBER,

LastUpdate DATE

);

CREATE TABLE BankAccounts (

BankAccountID NUMBER PRIMARY KEY,

ClientID NUMBER,

TypeOfAccount VARCHAR2(20),

AccountBalance NUMBER,

LastUpdate DATE,

FOREIGN KEY (ClientID) REFERENCES Clients(ClientID)

);

CREATE TABLE FinancialTransactions (

TransactionID NUMBER PRIMARY KEY,

BankAccountID NUMBER,

DateOfTransaction DATE,

TransactionAmount NUMBER,

TypeOfTransaction VARCHAR2(10),

FOREIGN KEY (BankAccountID) REFERENCES BankAccounts(BankAccountID)

);

CREATE TABLE PersonalLoans (

LoanID NUMBER PRIMARY KEY,

ClientID NUMBER,

AmountLoaned NUMBER,

LoanInterestRate NUMBER,

LoanStartDate DATE,

LoanEndDate DATE,

FOREIGN KEY (ClientID) REFERENCES Clients(ClientID)

);

CREATE TABLE StaffMembers (

StaffID NUMBER PRIMARY KEY,

FullName VARCHAR2(100),

JobTitle VARCHAR2(50),

MonthlySalary NUMBER,

DepartmentName VARCHAR2(50),

DateHired DATE

);

Insertion of Values in their respective tables:

INSERT INTO Clients (ClientID, FullName, DateOfBirth, AccountBalance, LastUpdate)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Clients (ClientID, FullName, DateOfBirth, AccountBalance, LastUpdate)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO BankAccounts (BankAccountID, ClientID, TypeOfAccount, AccountBalance, LastUpdate)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO BankAccounts (BankAccountID, ClientID, TypeOfAccount, AccountBalance, LastUpdate)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO FinancialTransactions (TransactionID, BankAccountID, DateOfTransaction, TransactionAmount, TypeOfTransaction)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO FinancialTransactions (TransactionID, BankAccountID, DateOfTransaction, TransactionAmount, TypeOfTransaction)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO PersonalLoans (LoanID, ClientID, AmountLoaned, LoanInterestRate, LoanStartDate, LoanEndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO StaffMembers (StaffID, FullName, JobTitle, MonthlySalary, DepartmentName, DateHired)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO StaffMembers (StaffID, FullName, JobTitle, MonthlySalary, DepartmentName, DateHired)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

Exercise 1: Control Structures

Scenario 1:

DECLARE

v\_client\_id NUMBER;

v\_dob DATE;

v\_client\_age NUMBER;

BEGIN

FOR rec IN (SELECT ClientID, DateOfBirth FROM Clients) LOOP

v\_client\_id := rec.ClientID;

v\_dob := rec.DateOfBirth;

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

IF v\_client\_age > 60 THEN

UPDATE PersonalLoans

SET LoanInterestRate = LoanInterestRate - 1

WHERE ClientID = v\_client\_id;

END IF;

END LOOP;

END;

Scenario 2:

BEGIN

FOR rec IN (SELECT ClientID, AccountBalance FROM Clients) LOOP

IF rec.AccountBalance > 10000 THEN

UPDATE Clients

SET IsVIP = 'TRUE'

WHERE ClientID = rec.ClientID;

END IF;

END LOOP;

END;

Scenario 3:

BEGIN

FOR rec IN (SELECT ClientID, LoanID, LoanEndDate FROM PersonalLoans WHERE LoanEndDate <= SYSDATE + 30) LOOP

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

END LOOP;

END;

Exercise 2: Error Handling

Scenario 1:

CREATE OR REPLACE PROCEDURE SafeTransfer (

p\_from\_account NUMBER,

p\_to\_account NUMBER,

p\_transfer\_amount NUMBER

) IS

e\_insufficient\_funds EXCEPTION;

v\_current\_balance NUMBER;

BEGIN

SELECT AccountBalance INTO v\_current\_balance FROM BankAccounts WHERE BankAccountID = p\_from\_account;

IF v\_current\_balance < p\_transfer\_amount THEN

RAISE e\_insufficient\_funds;

END IF;

UPDATE BankAccounts

SET AccountBalance = AccountBalance - p\_transfer\_amount

WHERE BankAccountID = p\_from\_account;

UPDATE BankAccounts

SET AccountBalance = AccountBalance + p\_transfer\_amount

WHERE BankAccountID = 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 UpdateEmployeePay (

p\_staff\_id NUMBER,

p\_increase\_percentage NUMBER

) IS

e\_staff\_not\_found EXCEPTION;

BEGIN

UPDATE StaffMembers

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

WHERE StaffID = p\_staff\_id;

IF SQL%NOTFOUND THEN

RAISE e\_staff\_not\_found;

END IF;

COMMIT;

EXCEPTION

WHEN e\_staff\_not\_found THEN

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

WHEN OTHERS THEN

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

ROLLBACK;

END;

Scenario 3:

CREATE OR REPLACE PROCEDURE AddClient (

p\_client\_id NUMBER,

p\_full\_name VARCHAR2,

p\_date\_of\_birth DATE,

p\_initial\_balance NUMBER

) IS

e\_client\_exists EXCEPTION;

BEGIN

INSERT INTO Clients (ClientID, FullName, DateOfBirth, AccountBalance, LastUpdate)

VALUES (p\_client\_id, p\_full\_name, p\_date\_of\_birth, p\_initial\_balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

RAISE e\_client\_exists;

WHEN e\_client\_exists THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Client ID ' || p\_client\_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 BankAccounts

SET AccountBalance = AccountBalance \* 1.01

WHERE TypeOfAccount = 'Savings';

COMMIT;

END;

Scenario 2:

CREATE OR REPLACE PROCEDURE UpdateBonus (

p\_department\_name VARCHAR2,

p\_bonus\_percentage NUMBER

) IS

BEGIN

UPDATE StaffMembers

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

WHERE DepartmentName = p\_department\_name;

COMMIT;

END;

Scenario 3:

CREATE OR REPLACE PROCEDURE TransferMoney (

p\_from\_account NUMBER,

p\_to\_account NUMBER,

p\_transfer\_amount NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT AccountBalance INTO v\_balance FROM BankAccounts WHERE BankAccountID = p\_from\_account;

IF v\_balance < p\_transfer\_amount THEN

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

RETURN;

END IF;

UPDATE BankAccounts

SET AccountBalance = AccountBalance - p\_transfer\_amount

WHERE BankAccountID = p\_from\_account;

UPDATE BankAccounts

SET AccountBalance = AccountBalance + p\_transfer\_amount

WHERE BankAccountID = p\_to\_account;

COMMIT;

END;

Exercise 4: Functions

Scenario 1:

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

v\_years\_old NUMBER;

BEGIN

SELECT FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_birthdate) / 12) INTO v\_years\_old FROM dual;

RETURN v\_years\_old;

END;

Scenario 2:

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_principal\_amount NUMBER,

p\_annual\_rate NUMBER,

p\_loan\_term\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_payment NUMBER;

v\_monthly\_interest\_rate NUMBER;

v\_total\_months NUMBER;

BEGIN

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

v\_total\_months := p\_loan\_term\_years \* 12;

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

RETURN v\_monthly\_payment;

END;

Scenario 3:

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_acc\_id NUMBER,

p\_withdrawal\_amount NUMBER

) RETURN BOOLEAN IS

v\_current\_balance NUMBER;

BEGIN

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

IF v\_current\_balance >= p\_withdrawal\_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.LastUpdated := SYSDATE;

END;

Scenario 2:

CREATE TABLE TransactionAuditLog (

AuditLogID NUMBER PRIMARY KEY,

TransID NUMBER,

AccID NUMBER,

TransDate DATE,

TransAmount NUMBER,

TransType VARCHAR2(10)

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO TransactionAuditLog (AuditLogID, TransID, AccID, TransDate, TransAmount, TransType)

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\_current\_balance NUMBER;

BEGIN

IF :NEW.TransType = 'Deposit' AND :NEW.TransAmount <= 0 THEN

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

ELSIF :NEW.TransType = 'Withdrawal' THEN

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

IF v\_current\_balance < :NEW.TransAmount THEN

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

END IF;

END IF;

END;

Exercise 6: Cursors

Scenario 1:

DECLARE

CURSOR c\_current\_month\_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\_record c\_current\_month\_transactions%ROWTYPE;

BEGIN

OPEN c\_current\_month\_transactions;

LOOP

FETCH c\_current\_month\_transactions INTO v\_transaction\_record;

EXIT WHEN c\_current\_month\_transactions%NOTFOUND;

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

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

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

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

END LOOP;

CLOSE c\_current\_month\_transactions;

END;

Scenario 2:

DECLARE

CURSOR c\_all\_accounts IS

SELECT \* FROM Accounts;

v\_account\_record c\_all\_accounts%ROWTYPE;

BEGIN

OPEN c\_all\_accounts;

LOOP

FETCH c\_all\_accounts INTO v\_account\_record;

EXIT WHEN c\_all\_accounts%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - 20

WHERE AccountID = v\_account\_record.AccountID;

END LOOP;

CLOSE c\_all\_accounts;

COMMIT;

END;

Scenario 3:

DECLARE

CURSOR c\_all\_loans IS

SELECT \* FROM Loans;

v\_loan\_record c\_all\_loans%ROWTYPE;

BEGIN

OPEN c\_all\_loans;

LOOP

FETCH c\_all\_loans INTO v\_loan\_record;

EXIT WHEN c\_all\_loans%NOTFOUND;

UPDATE Loans

SET InterestRate = InterestRate + 0.5

WHERE LoanID = v\_loan\_record.LoanID;

END LOOP;

CLOSE c\_all\_loans;

COMMIT;

END;

Exercise 7: Packages

Scenario 1:

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(p\_id NUMBER, p\_full\_name VARCHAR2, p\_birthdate DATE, p\_initial\_balance NUMBER);

PROCEDURE ModifyCustomerDetails(p\_id NUMBER, p\_full\_name VARCHAR2, p\_birthdate DATE, p\_balance NUMBER);

FUNCTION RetrieveCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_id NUMBER, p\_full\_name VARCHAR2, p\_birthdate DATE, p\_initial\_balance NUMBER) IS

BEGIN

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

VALUES (p\_id, p\_full\_name, p\_birthdate, p\_initial\_balance, SYSDATE);

COMMIT;

END;

PROCEDURE ModifyCustomerDetails(p\_id NUMBER, p\_full\_name VARCHAR2, p\_birthdate DATE, p\_balance NUMBER) IS

BEGIN

UPDATE Customers

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

WHERE CustomerID = p\_id;

COMMIT;

END;

FUNCTION RetrieveCustomerBalance(p\_id NUMBER) RETURN NUMBER IS

v\_customer\_balance NUMBER;

BEGIN

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

RETURN v\_customer\_balance;

END;

END CustomerManagement;

/

Scenario 2:

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireNewEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_job\_title VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2, p\_hire\_date DATE);

PROCEDURE UpdateEmployeeInfo(p\_id NUMBER, p\_name VARCHAR2, p\_job\_title VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2);

FUNCTION CalculateYearlySalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireNewEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_job\_title 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\_job\_title, p\_salary, p\_department, p\_hire\_date);

COMMIT;

END;

PROCEDURE UpdateEmployeeInfo(p\_id NUMBER, p\_name VARCHAR2, p\_job\_title VARCHAR2, p\_salary NUMBER, p\_department VARCHAR2) IS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_id;

COMMIT;

END;

FUNCTION CalculateYearlySalary(p\_id NUMBER) RETURN NUMBER IS

v\_monthly\_salary NUMBER;

BEGIN

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

RETURN v\_monthly\_salary \* 12;

END;

END EmployeeManagement;

/

Scenario 3:

CREATE OR REPLACE PACKAGE AccountOperations AS

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

PROCEDURE RemoveAccount(p\_id NUMBER);

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

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE CreateAccount(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 RemoveAccount(p\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_id;

COMMIT;

END;

FUNCTION ComputeTotalBalance(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;

/