**MODULE 3 PL/SQL EXERCISES**

**Exercise 1: Control Structures**

CREATE TABLE CUSTOMERS (

CUST\_ID NUMBER PRIMARY KEY,

NAME VARCHAR2(100),

AGE NUMBER,

BALANCE NUMBER,

ISVIP VARCHAR2(5)

);

CREATE TABLE LOANS (

LOAN\_ID NUMBER PRIMARY KEY,

CUST\_ID NUMBER,

INTEREST\_RATE NUMBER,

DUE\_DATE DATE,

FOREIGN KEY (CUST\_ID) REFERENCES CUSTOMERS(CUST\_ID)

);

BEGIN

-- Scenario 1: Apply 1% discount for customers above 60

FOR cust IN (SELECT \* FROM CUSTOMERS) LOOP

IF cust.AGE > 60 THEN

UPDATE LOANS

SET INTEREST\_RATE = INTEREST\_RATE - 1

WHERE CUST\_ID = cust.CUST\_ID;

END IF;

END LOOP;

-- Scenario 2: Mark customers with balance > 10000 as VIP

FOR cust IN (SELECT \* FROM CUSTOMERS) LOOP

IF cust.BALANCE > 10000 THEN

UPDATE CUSTOMERS

SET ISVIP = 'TRUE'

WHERE CUST\_ID = cust.CUST\_ID;

END IF;

END LOOP;

-- Scenario 3: Print reminders for loans due in next 30 days

FOR loan\_rec IN (

SELECT L.LOAN\_ID, L.CUST\_ID, L.DUE\_DATE, C.NAME

FROM LOANS L

JOIN CUSTOMERS C ON L.CUST\_ID = C.CUST\_ID

WHERE L.DUE\_DATE <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_rec.LOAN\_ID ||

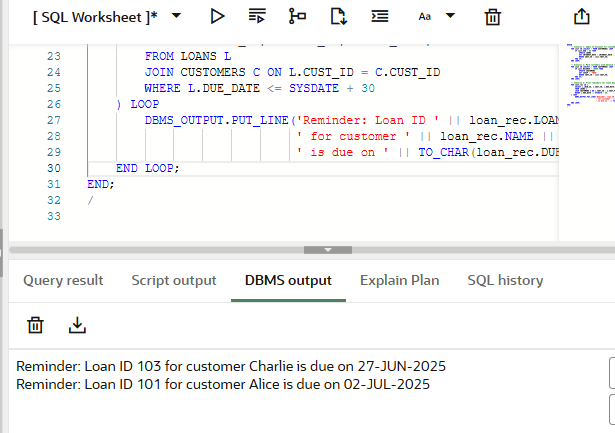
' for customer ' || loan\_rec.NAME ||

' is due on ' || TO\_CHAR(loan\_rec.DUE\_DATE, 'DD-MON-YYYY'));

END LOOP;

END;

/



**Exercise 2: Error Handling**

CREATE TABLE ACCOUNTS (

ACC\_ID NUMBER PRIMARY KEY,

BALANCE NUMBER

);

CREATE TABLE EMPLOYEES (

EMP\_ID NUMBER PRIMARY KEY,

NAME VARCHAR2(100),

SALARY NUMBER

);

CREATE TABLE CUSTOMERS (

CUST\_ID NUMBER PRIMARY KEY,

NAME VARCHAR2(100),

AGE NUMBER,

BALANCE NUMBER

);

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

from\_acc\_id IN NUMBER,

to\_acc\_id IN NUMBER,

amount IN NUMBER

) AS

from\_balance ACCOUNTS.BALANCE%TYPE;

BEGIN

SELECT BALANCE INTO from\_balance FROM ACCOUNTS WHERE ACC\_ID = from\_acc\_id;

IF from\_balance < amount THEN

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

END IF;

UPDATE ACCOUNTS SET BALANCE = BALANCE - amount WHERE ACC\_ID = from\_acc\_id;

UPDATE ACCOUNTS SET BALANCE = BALANCE + amount WHERE ACC\_ID = to\_acc\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

CREATE OR REPLACE PROCEDURE UpdateSalary (

emp\_id IN NUMBER,

percent IN NUMBER

) AS

BEGIN

UPDATE EMPLOYEES

SET SALARY = SALARY + (SALARY \* percent / 100)

WHERE EMP\_ID = emp\_id;

IF SQL%ROWCOUNT = 0 THEN

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

END IF;

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_age IN NUMBER,

p\_balance IN NUMBER

) AS

BEGIN

INSERT INTO CUSTOMERS VALUES (p\_id, p\_name, p\_age, p\_balance);

COMMIT;

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END;

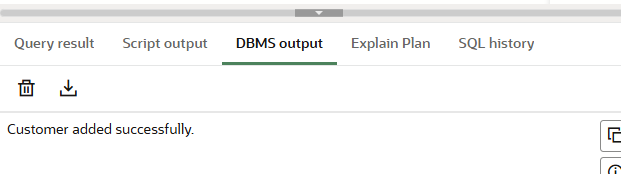
/

EXEC SafeTransferFunds(1, 2, 1000);

EXEC UpdateSalary(101, 10);

EXEC AddNewCustomer(1, 'David', 30, 5000);

EXEC AddNewCustomer(3, 'Eva', 25, 3000);



**Exercise 3: Stored Procedures**

CREATE TABLE SAVINGS\_ACCOUNTS (

ACC\_ID NUMBER PRIMARY KEY,

BALANCE NUMBER

);

CREATE TABLE EMPLOYEES (

EMP\_ID NUMBER PRIMARY KEY,

NAME VARCHAR2(100),

DEPT VARCHAR2(50),

SALARY NUMBER

);

CREATE TABLE ACCOUNTS (

ACC\_ID NUMBER PRIMARY KEY,

BALANCE NUMBER

);

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE SAVINGS\_ACCOUNTS

SET BALANCE = BALANCE + (BALANCE \* 0.01);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all savings accounts.');

EXCEPTION

WHEN OTHERS THEN

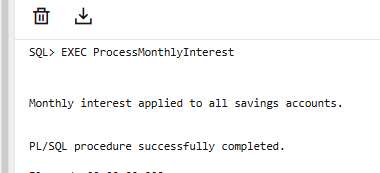
ROLLBACK;

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

END;

/

EXEC ProcessMonthlyInterest;



CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    dept\_name IN VARCHAR2,

    bonus\_percent IN NUMBER

) AS

BEGIN

    UPDATE EMPLOYEES

    SET SALARY = SALARY + (SALARY \* bonus\_percent / 100)

    WHERE DEPT = dept\_name;

    IF SQL%ROWCOUNT = 0 THEN

        RAISE\_APPLICATION\_ERROR(-20002, 'No employees found in department ' || dept\_name);

    END IF;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Bonus applied to department: ' || dept\_name);

EXCEPTION

    WHEN OTHERS THEN

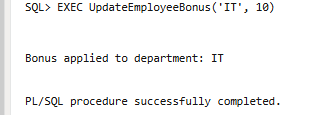
        ROLLBACK;

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

END;

/

EXEC UpdateEmployeeBonus('IT', 10);



CREATE OR REPLACE PROCEDURE TransferFunds (

    from\_acc IN NUMBER,

    to\_acc IN NUMBER,

    amount IN NUMBER

) AS

    from\_balance ACCOUNTS.BALANCE%TYPE;

BEGIN

    SELECT BALANCE INTO from\_balance FROM ACCOUNTS WHERE ACC\_ID = from\_acc;

    IF from\_balance < amount THEN

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

    END IF;

    UPDATE ACCOUNTS SET BALANCE = BALANCE - amount WHERE ACC\_ID = from\_acc;

    UPDATE ACCOUNTS SET BALANCE = BALANCE + amount WHERE ACC\_ID = to\_acc;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully.');

EXCEPTION

    WHEN OTHERS THEN

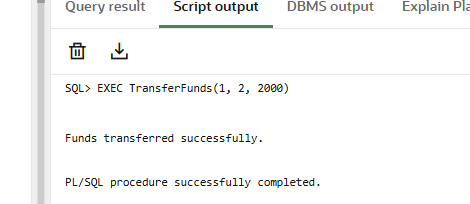
        ROLLBACK;

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

END;

/

EXEC TransferFunds(1, 2, 2000);



**Exercise 6: Cursors**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100)

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER REFERENCES Customers(CustomerID),

Balance NUMBER(12,2)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

CustomerID NUMBER REFERENCES Customers(CustomerID),

Amount NUMBER(10,2),

TransactionDate DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER REFERENCES Customers(CustomerID),

Amount NUMBER(12,2),

InterestRate NUMBER(5,2)

);

DECLARE

CURSOR GenerateMonthlyStatements IS

SELECT c.CustomerID, c.Name, t.Amount, t.TransactionDate

FROM Customers c

JOIN Transactions t ON c.CustomerID = t.CustomerID

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

v\_cust\_id Customers.CustomerID%TYPE;

v\_name Customers.Name%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_date Transactions.TransactionDate%TYPE;

BEGIN

OPEN GenerateMonthlyStatements;

LOOP

FETCH GenerateMonthlyStatements INTO v\_cust\_id, v\_name, v\_amount, v\_date;

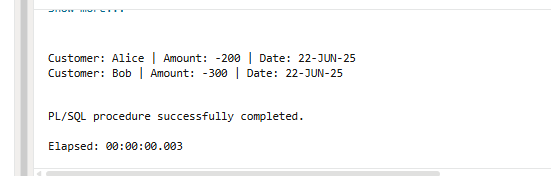
EXIT WHEN GenerateMonthlyStatements%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || v\_name || ' | Amount: ' || v\_amount || ' | Date: ' || v\_date);

END LOOP;

CLOSE GenerateMonthlyStatements;

END;



DECLARE

    CURSOR ApplyAnnualFee IS

        SELECT AccountID, Balance FROM Accounts;

    v\_account\_id Accounts.AccountID%TYPE;

    v\_balance Accounts.Balance%TYPE;

    annual\_fee CONSTANT NUMBER := 500;

BEGIN

    OPEN ApplyAnnualFee;

    LOOP

        FETCH ApplyAnnualFee INTO v\_account\_id, v\_balance;

        EXIT WHEN ApplyAnnualFee%NOTFOUND;

        UPDATE Accounts

        SET Balance = Balance - annual\_fee

        WHERE AccountID = v\_account\_id;

    END LOOP;

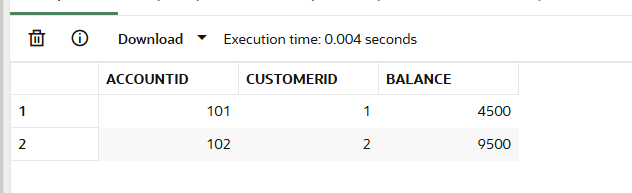
    CLOSE ApplyAnnualFee;

    COMMIT;

END;

/

SELECT \* FROM Accounts;



DECLARE

    CURSOR UpdateLoanInterestRates IS

        SELECT LoanID, Amount FROM Loans;

    v\_loan\_id Loans.LoanID%TYPE;

    v\_amount Loans.Amount%TYPE;

BEGIN

    OPEN UpdateLoanInterestRates;

    LOOP

        FETCH UpdateLoanInterestRates INTO v\_loan\_id, v\_amount;

        EXIT WHEN UpdateLoanInterestRates%NOTFOUND;

        IF v\_amount >= 15000 THEN

            UPDATE Loans SET InterestRate = 7.0 WHERE LoanID = v\_loan\_id;

        ELSE

            UPDATE Loans SET InterestRate = 6.5 WHERE LoanID = v\_loan\_id;

        END IF;

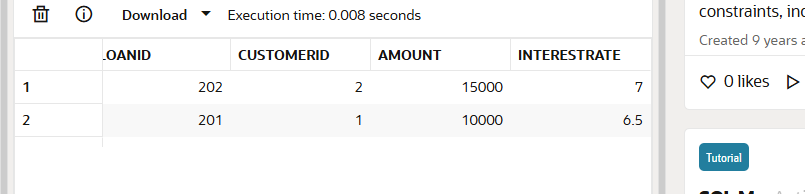
    END LOOP;

    CLOSE UpdateLoanInterestRates;

    COMMIT;

END;

SELECT \* FROM Loans;



**Exercise 4: Functions**

CREATE TABLE Customers (

CustomerID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

Name VARCHAR2(100),

DateOfBirth DATE

);

CREATE TABLE Loans (

LoanID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER, -- Annual %

DurationYears NUMBER

);

CREATE TABLE Accounts (

AccountID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

CREATE OR REPLACE FUNCTION CalculateAge(dob DATE)

RETURN NUMBER

IS

age NUMBER;

BEGIN

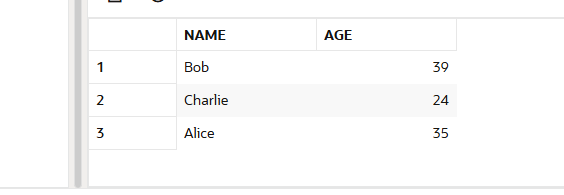
age := FLOOR(MONTHS\_BETWEEN(SYSDATE, dob) / 12);

RETURN age;

END;

/

SELECT Name, CalculateAge(DateOfBirth) AS Age FROM Customers;



CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

    amount NUMBER, rate NUMBER, years NUMBER

) RETURN NUMBER

IS

    r NUMBER := rate / 12 / 100;

    n NUMBER := years \* 12;

    emi NUMBER;

BEGIN

    emi := (amount \* r \* POWER(1 + r, n)) / (POWER(1 + r, n) - 1);

    RETURN ROUND(emi, 2);

END;

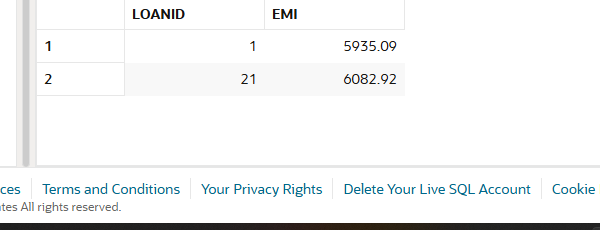
/

SELECT

    LoanID,

    CalculateMonthlyInstallment(LoanAmount, InterestRate, DurationYears) AS EMI

FROM Loans;



CREATE OR REPLACE FUNCTION HasSufficientBalance(acc\_id NUMBER, amt NUMBER)

RETURN CHAR

IS

    bal NUMBER;

BEGIN

    SELECT Balance INTO bal FROM Accounts WHERE AccountID = acc\_id;

    IF bal >= amt THEN

        RETURN 'Y';

    ELSE

        RETURN 'N';

    END IF;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN 'N';

END;

/

SELECT

    AccountID,

    HasSufficientBalance(AccountID, 1000) AS IsEligible

FROM Accounts;

