WEEK-2

**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

CODE:

-- Cleanup (optional)

BEGIN

    EXECUTE IMMEDIATE 'DROP TABLE loans CASCADE CONSTRAINTS';

    EXECUTE IMMEDIATE 'DROP TABLE customers CASCADE CONSTRAINTS';

EXCEPTION

    WHEN OTHERS THEN NULL;

END;

/

-- Create tables

CREATE TABLE customers (

    customer\_id NUMBER PRIMARY KEY,

    name VARCHAR2(50),

    date\_of\_birth DATE,

    loan\_interest\_rate NUMBER(5,2),

    balance NUMBER(10,2)

);

CREATE TABLE loans (

    loan\_id NUMBER PRIMARY KEY,

    customer\_id NUMBER REFERENCES customers(customer\_id),

    due\_date DATE,

    amount NUMBER(10,2)

);

-- Insert sample data

INSERT ALL

    INTO customers VALUES (1, 'John Smith', DATE '1950-07-15', 5.25, 7500.00)

    INTO customers VALUES (2, 'Emma Davis', DATE '1962-11-22', 4.75, 12500.00)

    INTO customers VALUES (3, 'Robert Chen', DATE '1948-03-04', 6.50, 8500.00)

    INTO customers VALUES (4, 'Sarah Johnson', DATE '1975-09-18', 5.75, 15500.00)

    INTO customers VALUES (5, 'Michael Brown', DATE '1955-12-30', 7.25, 9500.00)

    INTO loans VALUES (101, 1, SYSDATE + 15, 10000)

    INTO loans VALUES (102, 2, SYSDATE + 45, 20000)

    INTO loans VALUES (103, 3, SYSDATE + 10, 15000)

    INTO loans VALUES (104, 4, SYSDATE + 60, 25000)

    INTO loans VALUES (105, 5, SYSDATE + 25, 18000)

SELECT \* FROM dual;

COMMIT;

-- Scenario 1: Apply discount to seniors

SET SERVEROUTPUT ON;

DECLARE

    CURSOR c\_seniors IS

        SELECT customer\_id, loan\_interest\_rate

        FROM customers

        WHERE TRUNC(MONTHS\_BETWEEN(SYSDATE, date\_of\_birth)/12) > 60;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('Applying senior discounts...');

    FOR senior IN c\_seniors LOOP

        UPDATE customers

        SET loan\_interest\_rate = senior.loan\_interest\_rate - 1

        WHERE customer\_id = senior.customer\_id;

        DBMS\_OUTPUT.PUT\_LINE('- Discount applied to customer ' || senior.customer\_id ||

                            ': Rate reduced from ' || senior.loan\_interest\_rate || '% to ' ||

                            (senior.loan\_interest\_rate - 1) || '%');

    END LOOP;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Senior discount processing complete.');

END;

/

-- Verification

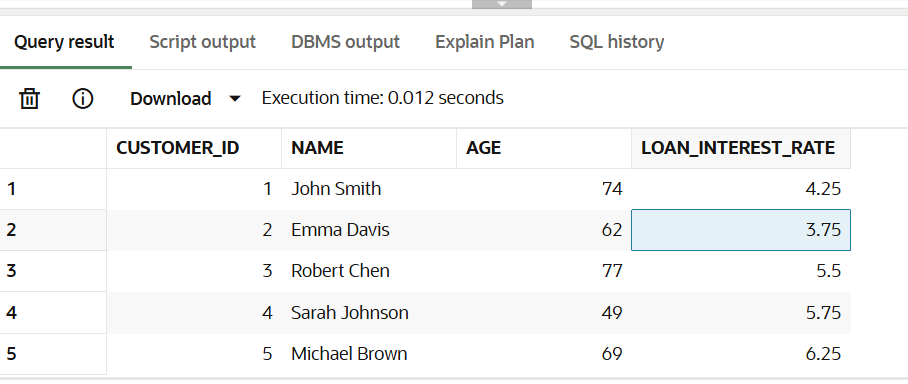
SELECT customer\_id, name,

       TRUNC(MONTHS\_BETWEEN(SYSDATE, date\_of\_birth)/12) AS age,

       loan\_interest\_rate

FROM customers;

RESULT:



**Scenario 2:** A customer can be promoted to VIP status based on their balance.

**Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

CODE:

-- Cleanup and setup for Scenario 2

BEGIN

    EXECUTE IMMEDIATE 'DROP TABLE customers CASCADE CONSTRAINTS';

EXCEPTION

    WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE customers (

    customer\_id NUMBER PRIMARY KEY,

    name VARCHAR2(50),

    date\_of\_birth DATE,

    loan\_interest\_rate NUMBER(5,2),

    balance NUMBER(10,2),

    is\_vip CHAR(1) DEFAULT 'N'  -- Corrected to include is\_vip column

);

INSERT INTO customers VALUES (1, 'John Smith', DATE '1950-07-15', 5.25, 7500.00, 'N');

INSERT INTO customers VALUES (2, 'Emma Davis', DATE '1962-11-22', 4.75, 12500.00, 'N');

INSERT INTO customers VALUES (3, 'Robert Chen', DATE '1948-03-04', 6.50, 8500.00, 'N');

INSERT INTO customers VALUES (4, 'Sarah Johnson', DATE '1975-09-18', 5.75, 15500.00, 'N');

INSERT INTO customers VALUES (5, 'Michael Brown', DATE '1955-12-30', 7.25, 9500.00, 'N');

COMMIT;

-- Scenario 2: Promote VIP customers

SET SERVEROUTPUT ON;

DECLARE

    CURSOR c\_vip IS

        SELECT customer\_id, name, balance

        FROM customers

        WHERE balance > 10000;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('Promoting VIP customers...');

    FOR vip IN c\_vip LOOP

        UPDATE customers

        SET is\_vip = 'Y'

        WHERE customer\_id = vip.customer\_id;

        DBMS\_OUTPUT.PUT\_LINE('- Promoted ' || vip.name || ' (ID:' || vip.customer\_id ||

                            ') to VIP. Balance: $' || vip.balance);

    END LOOP;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('VIP promotion complete.');

END;

/

-- Verification

SELECT customer\_id, name, balance, is\_vip

FROM customers;

RESULT:

Promoting VIP customers...

- Promoted Emma Davis (ID:2) to VIP. Balance: $12500

- Promoted Sarah Johnson (ID:4) to VIP. Balance: $15500

VIP promotion complete.

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

CODE

-- Scenario 3: Loan reminders

SET SERVEROUTPUT ON;

DECLARE

    CURSOR c\_loans IS

        SELECT c.name AS customer\_name, l.loan\_id, l.due\_date

        FROM loans l

        JOIN customers c ON l.customer\_id = c.customer\_id

        WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

        ORDER BY l.due\_date;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('Sending loan reminders...');

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

    FOR loan IN c\_loans LOOP

        DBMS\_OUTPUT.PUT\_LINE(

            'Reminder for: ' || loan.customer\_name ||

            ' | Loan #' || loan.loan\_id ||

            ' | Due: ' || TO\_CHAR(loan.due\_date, 'FMDay, DD Month YYYY') ||

            ' (' || TRUNC(loan.due\_date - SYSDATE) || ' days remaining)'

        );

    END LOOP;

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

    DBMS\_OUTPUT.PUT\_LINE('Reminders sent successfully.');

END;

/

REUSLUT:

Sending loan reminders...

================================

Reminder for: Robert Chen | Loan #103 | Due: Thursday, 10 July 2025 (9 days remaining)

Reminder for: John Smith | Loan #101 | Due: Tuesday, 15 July 2025 (14 days remaining)

Reminder for: Michael Brown | Loan #105 | Due: Friday, 25 July 2025 (24 days remaining)

================================

Reminders sent successfully.

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.001

**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

**Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

Code:

-- Cleanup

BEGIN

    EXECUTE IMMEDIATE 'DROP PROCEDURE ProcessMonthlyInterest';

    EXECUTE IMMEDIATE 'DROP TABLE savings\_accounts CASCADE CONSTRAINTS';

EXCEPTION

    WHEN OTHERS THEN NULL;

END;

/

-- Create table

CREATE TABLE savings\_accounts (

    account\_id NUMBER PRIMARY KEY,

    customer\_id NUMBER,

    balance NUMBER(15,2),

    interest\_rate NUMBER(5,2) DEFAULT 1.0

);

-- Insert data

INSERT INTO savings\_accounts VALUES (101, 1, 5000.00, 1.0);

INSERT INTO savings\_accounts VALUES (102, 2, 12500.00, 1.0);

INSERT INTO savings\_accounts VALUES (103, 3, 8500.00, 1.0);

COMMIT;

-- Create procedure

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

    UPDATE savings\_accounts

    SET balance = balance \* (1 + interest\_rate/100);

    DBMS\_OUTPUT.PUT\_LINE('Applied 1% monthly interest to ' || SQL%ROWCOUNT || ' savings accounts');

EXCEPTION

    WHEN OTHERS THEN

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

        RAISE;

END;

/

-- Test

SET SERVEROUTPUT ON;

DECLARE

    CURSOR c\_acc IS SELECT account\_id, balance FROM savings\_accounts ORDER BY account\_id;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== INITIAL BALANCES ===');

    FOR acc IN c\_acc LOOP

        DBMS\_OUTPUT.PUT\_LINE('Account ' || acc.account\_id || ': $' || acc.balance);

    END LOOP;

    ProcessMonthlyInterest();

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '=== UPDATED BALANCES ===');

    FOR acc IN c\_acc LOOP

        DBMS\_OUTPUT.PUT\_LINE('Account ' || acc.account\_id || ': $' ||

            (SELECT balance FROM savings\_accounts WHERE account\_id = acc.account\_id));

    END LOOP;

END;

/

Result:

=== INITIAL BALANCES ===

Account 101: $5000.00

Account 102: $12500.00

Account 103: $8500.00

Applied 1% monthly interest to 3 savings accounts

=== UPDATED BALANCES ===

Account 101: $5050.00

Account 102: $12625.00

Account 103: $8585.00

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

**Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

Code:

-- Cleanup

BEGIN

    EXECUTE IMMEDIATE 'DROP PROCEDURE UpdateEmployeeBonus';

    EXECUTE IMMEDIATE 'DROP TABLE employees CASCADE CONSTRAINTS';

EXCEPTION

    WHEN OTHERS THEN NULL;

END;

/

-- Create table

CREATE TABLE employees (

    employee\_id NUMBER PRIMARY KEY,

    name VARCHAR2(50),

    department\_id NUMBER,

    salary NUMBER(10,2),

    bonus\_percentage NUMBER(5,2)

);

-- Insert data

INSERT INTO employees VALUES (1, 'John Smith', 10, 5000.00, NULL);

INSERT INTO employees VALUES (2, 'Emma Davis', 10, 6500.00, NULL);

INSERT INTO employees VALUES (3, 'Robert Chen', 20, 7200.00, NULL);

INSERT INTO employees VALUES (4, 'Sarah Johnson', 20, 5800.00, NULL);

COMMIT;

-- Create procedure

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

    p\_dept\_id IN NUMBER,

    p\_bonus\_percent IN NUMBER

) IS

BEGIN

    UPDATE employees

    SET bonus\_percentage = p\_bonus\_percent,

        salary = salary \* (1 + p\_bonus\_percent/100)

    WHERE department\_id = p\_dept\_id;

    DBMS\_OUTPUT.PUT\_LINE('Applied ' || p\_bonus\_percent || '% bonus to ' || SQL%ROWCOUNT || ' employees');

EXCEPTION

    WHEN OTHERS THEN

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

        RAISE;

END;

/

-- Test

SET SERVEROUTPUT ON;

DECLARE

    v\_dept\_id NUMBER := 10;

    v\_bonus NUMBER := 5;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== INITIAL SALARIES (DEPT 10) ===');

    FOR e IN (SELECT name, salary FROM employees WHERE department\_id = v\_dept\_id) LOOP

        DBMS\_OUTPUT.PUT\_LINE(e.name || ': $' || e.salary);

    END LOOP;

    UpdateEmployeeBonus(v\_dept\_id, v\_bonus);

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '=== UPDATED SALARIES ===');

    FOR e IN (SELECT name, salary FROM employees WHERE department\_id = v\_dept\_id) LOOP

        DBMS\_OUTPUT.PUT\_LINE(e.name || ': $' || e.salary);

    END LOOP;

END;

/

Result:

=== INITIAL SALARIES (DEPT 10) ===

John Smith: $5000.00

Emma Davis: $6500.00

Applied 5% bonus to 2 employees

=== UPDATED SALARIES ===

John Smith: $5250.00

Emma Davis: $6825.00

**Scenario 3:** Customers should be able to transfer funds between their accounts.

**Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

Code:

-- Cleanup

BEGIN

    EXECUTE IMMEDIATE 'DROP PROCEDURE TransferFunds';

    EXECUTE IMMEDIATE 'DROP TABLE accounts CASCADE CONSTRAINTS';

EXCEPTION

    WHEN OTHERS THEN NULL;

END;

/

-- Create table

CREATE TABLE accounts (

    account\_id NUMBER PRIMARY KEY,

    customer\_id NUMBER,

    account\_type VARCHAR2(20),

    balance NUMBER(15,2)

);

-- Insert data

INSERT INTO accounts VALUES (1001, 1, 'CHECKING', 5000.00);

INSERT INTO accounts VALUES (1002, 1, 'SAVINGS', 12500.00);

INSERT INTO accounts VALUES (2001, 2, 'CHECKING', 8500.00);

COMMIT;

-- Create procedure

CREATE OR REPLACE PROCEDURE TransferFunds(

    p\_from\_account IN NUMBER,

    p\_to\_account IN NUMBER,

    p\_amount IN NUMBER

) IS

    v\_from\_balance NUMBER;

BEGIN

    -- Check account exists

    SELECT COUNT(\*) INTO v\_from\_balance FROM accounts WHERE account\_id = p\_from\_account;

    IF v\_from\_balance = 0 THEN

        RAISE\_APPLICATION\_ERROR(-20001, 'Source account not found');

    END IF;

    -- Check sufficient funds

    SELECT balance INTO v\_from\_balance FROM accounts WHERE account\_id = p\_from\_account FOR UPDATE;

    IF v\_from\_balance < p\_amount THEN

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

    END IF;

    -- Perform transfer

    UPDATE accounts SET balance = balance - p\_amount WHERE account\_id = p\_from\_account;

    UPDATE accounts SET balance = balance + p\_amount WHERE account\_id = p\_to\_account;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('$' || p\_amount || ' transferred from #' || p\_from\_account || ' to #' || p\_to\_account);

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

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

        RAISE;

END;

/

-- Test

SET SERVEROUTPUT ON;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('=== INITIAL BALANCES ===');

    DBMS\_OUTPUT.PUT\_LINE('Account 1001: $' || (SELECT balance FROM accounts WHERE account\_id = 1001));

    DBMS\_OUTPUT.PUT\_LINE('Account 1002: $' || (SELECT balance FROM accounts WHERE account\_id = 1002));

    -- Successful transfer

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Testing valid transfer...');

    TransferFunds(1002, 1001, 1500);

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '=== AFTER TRANSFER ===');

    DBMS\_OUTPUT.PUT\_LINE('Account 1001: $' || (SELECT balance FROM accounts WHERE account\_id = 1001));

    DBMS\_OUTPUT.PUT\_LINE('Account 1002: $' || (SELECT balance FROM accounts WHERE account\_id = 1002));

    -- Insufficient funds

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Testing insufficient funds...');

    TransferFunds(1001, 1002, 10000);

EXCEPTION

    WHEN OTHERS THEN

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

END;

/

Result:

=== INITIAL BALANCES ===

Account 1001: $5000.00

Account 1002: $12500.00

Testing valid transfer...

$1500 transferred from #1002 to #1001

=== AFTER TRANSFER ===

Account 1001: $6500.00

Account 1002: $11000.00

Testing insufficient funds...

Transfer failed: ORA-20002: Insufficient funds

ORA-06512: at "SYSTEM.TRANSFERFUNDS", line 22

Expected error caught: ORA-20002: Insufficient funds