**MODULE 2 – PL/SQL PROGRAMMING**

**Exercise 1: Control Structures**

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

**Code**

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

dob DATE

);

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

interest\_rate NUMBER(5,2),

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

INSERT INTO customers VALUES (1, TO\_DATE('1950-03-15', 'YYYY-MM-DD'));

INSERT INTO customers VALUES (2, TO\_DATE('1985-06-25', 'YYYY-MM-DD'));

INSERT INTO loans VALUES (101, 1, 9.5);

INSERT INTO loans VALUES (102, 2, 8.5);

COMMIT;

SET SERVEROUTPUT ON;

DECLARE

v\_age NUMBER;

CURSOR cur\_discount IS

SELECT c.customer\_id, c.dob, l.loan\_id, l.interest\_rate

FROM customers c

JOIN loans l ON c.customer\_id = l.customer\_id;

BEGIN

FOR rec IN cur\_discount LOOP

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, rec.dob) / 12);

IF v\_age > 60 THEN

UPDATE loans

SET interest\_rate = rec.interest\_rate \* 0.99

WHERE loan\_id = rec.loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to Loan ID: ' || rec.loan\_id ||

' | Age: ' || v\_age ||

' | New Interest: ' || (rec.interest\_rate \* 0.99));

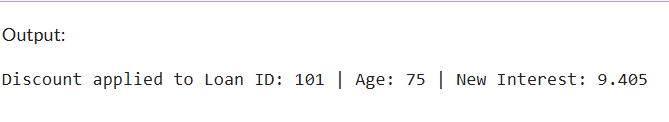
END IF;

END LOOP;

COMMIT;

END;

/



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

**Code**

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

customer\_name VARCHAR2(100),

balance NUMBER(10, 2),

isvip VARCHAR2(5) DEFAULT 'FALSE'

);

INSERT INTO customers VALUES (1, 'Uma', 10500.00, 'FALSE');

INSERT INTO customers VALUES (2, 'Arun', 12000.00, 'FALSE');

INSERT INTO customers VALUES (3, 'Vigna', 8000.00, 'FALSE');

INSERT INTO customers VALUES (4, 'Anna', 15000.00, 'FALSE');

INSERT INTO customers VALUES (5, 'Emma', 1000.00, 'FALSE');

COMMIT;

DECLARE

v\_balance NUMBER;

CURSOR cur\_customers IS

SELECT customer\_id, balance

FROM customers;

BEGIN

FOR rec IN cur\_customers LOOP

v\_balance := rec.balance;

IF v\_balance > 10000 THEN

UPDATE customers

SET isvip = 'TRUE'

WHERE customer\_id = rec.customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || rec.customer\_id ||

' promoted to VIP | Balance: ' || v\_balance);

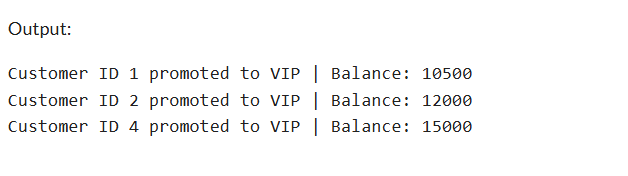
END IF;

END LOOP;

COMMIT;

END;

/

****

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

**Code**

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

customer\_name VARCHAR2(100)

);

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

due\_date DATE,

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

INSERT INTO customers VALUES (1, 'Akash');

INSERT INTO customers VALUES (2, 'kiran');

INSERT INTO customers VALUES (3, 'Jannet');

INSERT INTO loans VALUES (201, 1, SYSDATE + 15);

INSERT INTO loans VALUES (202, 2, SYSDATE + 40);

INSERT INTO loans VALUES (203, 3, SYSDATE + 5);

COMMIT;

SET SERVEROUTPUT ON;

DECLARE

CURSOR cur\_due\_loans IS

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

FROM customers c

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

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

BEGIN

FOR rec IN cur\_due\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.loan\_id ||

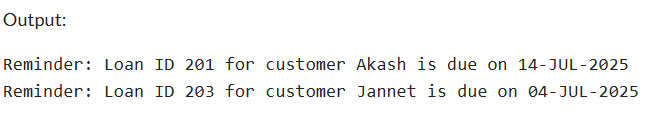
' for customer ' || rec.customer\_name ||

' is due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY'));

END LOOP;

END;

/



**Exercise 3: Stored Procedures**

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

**Code**

SET SERVEROUTPUT ON;

CREATE TABLE accounts (

account\_id NUMBER PRIMARY KEY,

account\_type VARCHAR2(20),

balance NUMBER(10, 2)

);

INSERT INTO accounts VALUES (101, 'savings', 10000);

INSERT INTO accounts VALUES (102, 'current', 20000);

INSERT INTO accounts VALUES (103, 'savings', 5000);

INSERT INTO accounts VALUES (104, 'savings', 15000);

INSERT INTO accounts VALUES (105, 'current', 8000);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE accounts

SET balance = balance \* 1.01

WHERE LOWER(account\_type) = 'savings';

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' savings accounts updated.');

COMMIT;

END;

/

BEGIN

ProcessMonthlyInterest;

END;

/

BEGIN

FOR rec IN (SELECT account\_id, account\_type, balance FROM accounts ORDER BY account\_id) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.account\_id ||

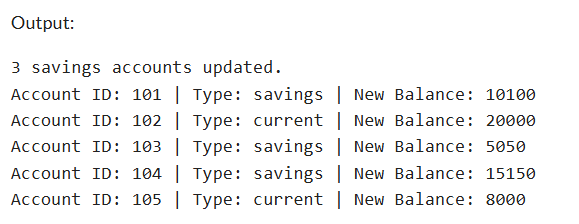
' | Type: ' || rec.account\_type ||

' | New Balance: ' || rec.balance);

END LOOP;

END;

/



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

**Code**

SET SERVEROUTPUT ON;

CREATE TABLE employees (

employee\_id NUMBER PRIMARY KEY,

employee\_name VARCHAR2(100),

department VARCHAR2(50),

salary NUMBER(10, 2)

);

INSERT INTO employees VALUES (1, 'Alice', 'HR', 50000);

INSERT INTO employees VALUES (2, 'Bob', 'IT', 60000);

INSERT INTO employees VALUES (3, 'Charlie', 'HR', 55000);

INSERT INTO employees VALUES (4, 'David', 'Finance', 70000);

INSERT INTO employees VALUES (5, 'Eve', 'IT', 65000);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE employees

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

WHERE LOWER(department) = LOWER(p\_department);

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' employees in ' || p\_department ||

' department received a bonus of ' || p\_bonus\_percent || '%.');

COMMIT;

END;

/

BEGIN

UpdateEmployeeBonus('IT', 10);

END;

/

BEGIN

FOR rec IN (SELECT \* FROM employees ORDER BY employee\_id) LOOP

DBMS\_OUTPUT.PUT\_LINE('ID: ' || rec.employee\_id ||

' | Name: ' || rec.employee\_name ||

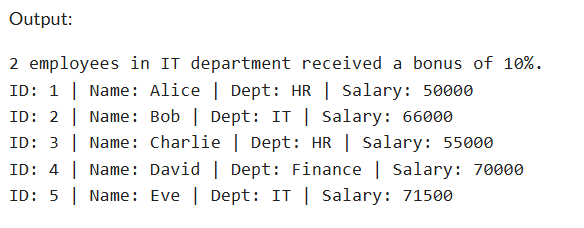
' | Dept: ' || rec.department ||

' | Salary: ' || rec.salary);

END LOOP;

END;

/

****

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

**Code**

SET SERVEROUTPUT ON;

CREATE TABLE accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

balance NUMBER(10, 2)

);

INSERT INTO accounts VALUES (1001, 1, 5000);

INSERT INTO accounts VALUES (1002, 1, 3000);

INSERT INTO accounts VALUES (1003, 2, 7000);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_source\_account IN NUMBER,

p\_target\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_source\_balance NUMBER;

BEGIN

SELECT balance INTO v\_source\_balance

FROM accounts

WHERE account\_id = p\_source\_account;

IF v\_source\_balance < p\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance in source account.');

ELSE

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_source\_account;

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_target\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ' || p\_amount ||

' transferred from account ' || p\_source\_account ||

' to account ' || p\_target\_account);

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: One or both account IDs not found.');

WHEN OTHERS THEN

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

END;

/

BEGIN

TransferFunds(1001, 1002, 2000);

END;

/

BEGIN

FOR rec IN (SELECT account\_id, balance FROM accounts ORDER BY account\_id) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.account\_id ||

' | Balance: ' || rec.balance);

END LOOP;

END;

/

