**EXERCISE 1: CONTROL STRUCTURES**

-- STEP 1: Create required tables

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE loans CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE customers CASCADE CONSTRAINTS';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

age NUMBER,

interest\_rate NUMBER,

balance NUMBER,

isvip VARCHAR2(5)

);

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

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

due\_date DATE

);

-- STEP 2: Insert sample data

INSERT INTO customers VALUES (1, 'Alice', 65, 5.5, 12000, 'FALSE');

INSERT INTO customers VALUES (2, 'Bob', 45, 6.2, 8000, 'FALSE');

INSERT INTO customers VALUES (3, 'Charlie', 70, 7.0, 15000, 'FALSE');

INSERT INTO loans VALUES (101, 1, SYSDATE + 10);

INSERT INTO loans VALUES (102, 2, SYSDATE + 40);

INSERT INTO loans VALUES (103, 3, SYSDATE + 5);

COMMIT;

-- SCENARIO 1: Apply 1% discount to interest rate if age > 60

BEGIN

FOR rec IN (SELECT \* FROM customers WHERE age > 60) LOOP

UPDATE customers

SET interest\_rate = interest\_rate - 1

WHERE customer\_id = rec.customer\_id;

END LOOP;

COMMIT;

END;

/

-- SCENARIO 2: Set VIP flag if balance > 10000

BEGIN

FOR rec IN (SELECT \* FROM customers WHERE balance > 10000) LOOP

UPDATE customers

SET isvip = 'TRUE'

WHERE customer\_id = rec.customer\_id;

END LOOP;

COMMIT;

END;

/

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

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (

SELECT c.name, l.due\_date

FROM customers c

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

WHERE l.due\_date <= SYSDATE + 30

) LOOP

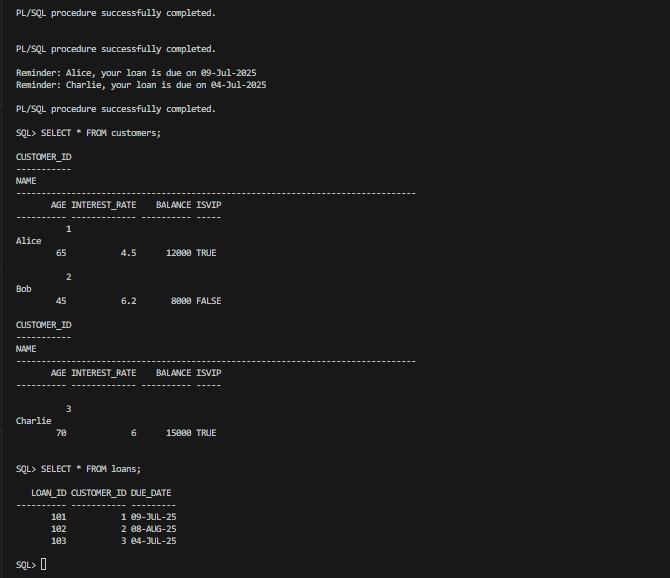
DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || rec.name || ', your loan is due on ' || TO\_CHAR(rec.due\_date, 'DD-Mon-YYYY'));

END LOOP;

END;

/

OUTPUT



**EXERCISE 2:STORED PROCEDURES**

-- STEP 1: Drop if exists

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE employees CASCADE CONSTRAINTS';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- STEP 2: Create tables

CREATE TABLE accounts (

account\_id NUMBER PRIMARY KEY,

customer\_name VARCHAR2(100),

balance NUMBER,

account\_type VARCHAR2(20)

);

CREATE TABLE employees (

emp\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

department VARCHAR2(50),

salary NUMBER

);

-- STEP 3: Insert data

INSERT INTO accounts VALUES (1, 'Alice', 10000, 'savings');

INSERT INTO accounts VALUES (2, 'Bob', 5000, 'savings');

INSERT INTO accounts VALUES (3, 'Charlie', 7000, 'checking');

INSERT INTO employees VALUES (101, 'John', 'HR', 40000);

INSERT INTO employees VALUES (102, 'Jane', 'IT', 45000);

INSERT INTO employees VALUES (103, 'Doe', 'IT', 42000);

COMMIT;

-- SCENARIO 1: ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR rec IN (SELECT account\_id, customer\_name, balance FROM accounts WHERE account\_type = 'savings') LOOP

DBMS\_OUTPUT.PUT\_LINE('Before Interest - Account ' || rec.account\_id || ' (' || rec.customer\_name || '): ' || rec.balance);

END LOOP;

UPDATE accounts

SET balance = balance + (balance \* 0.01)

WHERE account\_type = 'savings';

DBMS\_OUTPUT.PUT\_LINE('Monthly interest of 1% applied to all savings accounts.');

FOR rec IN (SELECT account\_id, customer\_name, balance FROM accounts WHERE account\_type = 'savings') LOOP

DBMS\_OUTPUT.PUT\_LINE('After Interest - Account ' || rec.account\_id || ' (' || rec.customer\_name || '): ' || rec.balance);

END LOOP;

COMMIT;

END;

/

-- SCENARIO 2: UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept IN VARCHAR2,

bonus\_percent IN NUMBER

) IS

BEGIN

FOR rec IN (SELECT emp\_id, name, salary FROM employees WHERE department = dept) LOOP

DBMS\_OUTPUT.PUT\_LINE('Before Bonus - ' || rec.name || ': ' || rec.salary);

END LOOP;

UPDATE employees

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

WHERE department = dept;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || bonus\_percent || '% applied to ' || dept || ' department.');

FOR rec IN (SELECT emp\_id, name, salary FROM employees WHERE department = dept) LOOP

DBMS\_OUTPUT.PUT\_LINE('After Bonus - ' || rec.name || ': ' || rec.salary);

END LOOP;

COMMIT;

END;

/

-- SCENARIO 3: TransferFunds

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_account\_id IN NUMBER,

to\_account\_id IN NUMBER,

amount IN NUMBER

) IS

from\_balance NUMBER;

to\_balance NUMBER;

BEGIN

SELECT balance INTO from\_balance FROM accounts WHERE account\_id = from\_account\_id;

SELECT balance INTO to\_balance FROM accounts WHERE account\_id = to\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Before Transfer - From Account ' || from\_account\_id || ': ' || from\_balance);

DBMS\_OUTPUT.PUT\_LINE('Before Transfer - To Account ' || to\_account\_id || ': ' || to\_balance);

IF from\_balance < amount THEN

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

END IF;

UPDATE accounts SET balance = balance - amount WHERE account\_id = from\_account\_id;

UPDATE accounts SET balance = balance + amount WHERE account\_id = to\_account\_id;

SELECT balance INTO from\_balance FROM accounts WHERE account\_id = from\_account\_id;

SELECT balance INTO to\_balance FROM accounts WHERE account\_id = to\_account\_id;

DBMS\_OUTPUT.PUT\_LINE(amount || ' transferred from Account ' || from\_account\_id || ' to Account ' || to\_account\_id);

DBMS\_OUTPUT.PUT\_LINE('After Transfer - From Account ' || from\_account\_id || ': ' || from\_balance);

DBMS\_OUTPUT.PUT\_LINE('After Transfer - To Account ' || to\_account\_id || ': ' || to\_balance);

COMMIT;

END;

/

-- ENABLE OUTPUT

SET SERVEROUTPUT ON;

-- SCENARIO 1 EXECUTION

BEGIN

ProcessMonthlyInterest;

END;

/

-- SCENARIO 2 EXECUTION

BEGIN

UpdateEmployeeBonus('IT', 10);

END;

/

-- SCENARIO 3 EXECUTION

BEGIN

TransferFunds(1, 3, 2000);

END;

/

**OUTPUT:**

