**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.

**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.

**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.

Solution:

**Queries:**

-- INIT database

CREATE TABLE CUSTOMERS (

customer\_id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,

customer\_name VARCHAR2(100),

age NUMBER,

balance NUMBER(10, 2),

is\_vip VARCHAR2(5) DEFAULT 'FALSE'

);

CREATE TABLE LOANS (

loan\_id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,

customer\_id NUMBER,

loan\_amount NUMBER(10, 2),

interest\_rate NUMBER(5, 2),

due\_date DATE,

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

);

INSERT INTO CUSTOMERS (customer\_name, age, balance) VALUES ('Customer A', 65, 8000.00);

INSERT INTO CUSTOMERS (customer\_name, age, balance) VALUES ('Customer B', 55, 12000.00);

INSERT INTO CUSTOMERS (customer\_name, age, balance) VALUES ('Customer C', 70, 5000.00);

INSERT INTO CUSTOMERS (customer\_name, age, balance) VALUES ('Customer D', 30, 25000.00);

INSERT INTO CUSTOMERS (customer\_name, age, balance) VALUES ('Customer E', 45, 9500.00);

INSERT INTO CUSTOMERS (customer\_name, age, balance) VALUES ('Customer F', 62, 11000.00);

INSERT INTO LOANS (customer\_id, loan\_amount, interest\_rate, due\_date) VALUES (1, 50000.00, 5.00, SYSDATE + 15);

INSERT INTO LOANS (customer\_id, loan\_amount, interest\_rate, due\_date) VALUES (2, 20000.00, 6.50, SYSDATE + 45);

INSERT INTO LOANS (customer\_id, loan\_amount, interest\_rate, due\_date) VALUES (3, 75000.00, 5.25, SYSDATE + 25);

INSERT INTO LOANS (customer\_id, loan\_amount, interest\_rate, due\_date) VALUES (4, 10000.00, 7.00, SYSDATE + 5);

INSERT INTO LOANS (customer\_id, loan\_amount, interest\_rate, due\_date) VALUES (5, 30000.00, 6.00, SYSDATE + 90);

INSERT INTO LOANS (customer\_id, loan\_amount, interest\_rate, due\_date) VALUES (6, 40000.00, 5.50, SYSDATE + 20);

INSERT INTO LOANS (customer\_id, loan\_amount, interest\_rate, due\_date) VALUES (1, 15000.00, 5.10, SYSDATE + 35);

INSERT INTO LOANS (customer\_id, loan\_amount, interest\_rate, due\_date) VALUES (3, 25000.00, 5.30, SYSDATE + 10);

COMMIT;

DECLARE

CURSOR c\_customers\_over\_60 IS

SELECT

c.customer\_id,

c.age,

l.loan\_id,

l.interest\_rate

FROM

CUSTOMERS c

JOIN

LOANS l ON c.customer\_id = l.customer\_id

WHERE

c.age > 60;

v\_discount\_rate CONSTANT NUMBER := 0.01;

BEGIN

FOR rec IN c\_customers\_over\_60 LOOP

UPDATE LOANS

SET interest\_rate = rec.interest\_rate - v\_discount\_rate

WHERE loan\_id = rec.loan\_id;

DBMS\_OUTPUT.PUT\_LINE(

'Updated Loan ID ' || rec.loan\_id ||

' for Customer ID ' || rec.customer\_id ||

' (Age: ' || rec.age || '): Old Rate = ' || TO\_CHAR(rec.interest\_rate, '99.99') || '%' ||

', New Rate = ' || TO\_CHAR(rec.interest\_rate - v\_discount\_rate, '99.99') || '%'

);

END LOOP;

COMMIT;

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

END;

/

DECLARE

CURSOR c\_potential\_vips IS

SELECT

customer\_id,

balance,

is\_vip

FROM

CUSTOMERS

WHERE

balance > 10000.00

AND is\_vip = 'FALSE';

BEGIN

FOR rec IN c\_potential\_vips LOOP

UPDATE CUSTOMERS

SET is\_vip = 'TRUE'

WHERE customer\_id = rec.customer\_id;

DBMS\_OUTPUT.PUT\_LINE(

'Promoted Customer ID ' || rec.customer\_id ||

' (Balance: $' || TO\_CHAR(rec.balance, '999,999.00') || ') to VIP status.'

);

END LOOP;

COMMIT;

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

END;

/

DECLARE

CURSOR c\_due\_loans IS

SELECT

c.customer\_name,

l.loan\_amount,

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;

v\_current\_date DATE := SYSDATE;

BEGIN

FOR rec IN c\_due\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE(

'REMINDER: Dear ' || rec.customer\_name ||

', your loan of $' || TO\_CHAR(rec.loan\_amount, '999,999.00') ||

' is due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY') || '. Please ensure timely payment.'

);

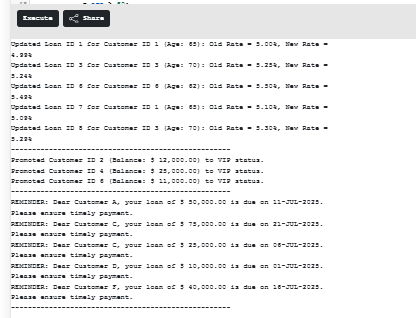
END LOOP;

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

END;

/

**Output:**



**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.

**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.

**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.

**Solution:**

**Queries:**

-- INIT database

CREATE TABLE SAVINGS\_ACCOUNTS (

account\_id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,

customer\_name VARCHAR2(100),

balance NUMBER(10, 2)

);

CREATE TABLE EMPLOYEES (

employee\_id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,

employee\_name VARCHAR2(100),

department\_id NUMBER,

salary NUMBER(10, 2)

);

CREATE TABLE BANK\_ACCOUNTS (

account\_id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,

customer\_name VARCHAR2(100),

balance NUMBER(10, 2)

);

INSERT INTO SAVINGS\_ACCOUNTS (customer\_name, balance) VALUES ('Customer A', 1000.00);

INSERT INTO SAVINGS\_ACCOUNTS (customer\_name, balance) VALUES ('Customer B', 5000.50);

INSERT INTO SAVINGS\_ACCOUNTS (customer\_name, balance) VALUES ('Customer C', 250.75);

INSERT INTO SAVINGS\_ACCOUNTS (customer\_name, balance) VALUES ('Customer D', 12000.00);

INSERT INTO EMPLOYEES (employee\_name, department\_id, salary) VALUES ('Employee 1', 10, 50000.00);

INSERT INTO EMPLOYEES (employee\_name, department\_id, salary) VALUES ('Employee 2', 20, 60000.00);

INSERT INTO EMPLOYEES (employee\_name, department\_id, salary) VALUES ('Employee 3', 10, 55000.00);

INSERT INTO EMPLOYEES (employee\_name, department\_id, salary) VALUES ('Employee 4', 30, 70000.00);

INSERT INTO EMPLOYEES (employee\_name, department\_id, salary) VALUES ('Employee 5', 20, 62000.00);

INSERT INTO BANK\_ACCOUNTS (customer\_name, balance) VALUES ('Customer P', 1500.00);

INSERT INTO BANK\_ACCOUNTS (customer\_name, balance) VALUES ('Customer Q', 3000.00);

INSERT INTO BANK\_ACCOUNTS (customer\_name, balance) VALUES ('Customer R', 500.00);

INSERT INTO BANK\_ACCOUNTS (customer\_name, balance) VALUES ('Customer S', 8000.00);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

v\_interest\_rate CONSTANT NUMBER := 0.01;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Before interest processing:');

FOR rec IN (SELECT account\_id, customer\_name, balance FROM SAVINGS\_ACCOUNTS ORDER BY account\_id) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.account\_id || ', Customer: ' || rec.customer\_name || ', Balance: $' || TO\_CHAR(rec.balance, '999,999.00'));

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('--- Applying Monthly Interest ---');

UPDATE SAVINGS\_ACCOUNTS

SET balance = balance \* (1 + v\_interest\_rate);

DBMS\_OUTPUT.PUT\_LINE('After interest processing:');

FOR rec IN (SELECT account\_id, customer\_name, balance FROM SAVINGS\_ACCOUNTS ORDER BY account\_id) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.account\_id || ', Customer: ' || rec.customer\_name || ', New Balance: $' || TO\_CHAR(rec.balance, '999,999.00'));

END LOOP;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END ProcessMonthlyInterest;

/

BEGIN

ProcessMonthlyInterest;

END;

/

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department\_id IN NUMBER,

p\_bonus\_percentage IN NUMBER

)

IS

v\_updated\_count NUMBER := 0;

BEGIN

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

DBMS\_OUTPUT.PUT\_LINE('Before bonus update for Department ' || p\_department\_id || ':');

FOR rec IN (SELECT employee\_id, employee\_name, salary FROM EMPLOYEES WHERE department\_id = p\_department\_id ORDER BY employee\_id) LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || rec.employee\_id || ', Name: ' || rec.employee\_name || ', Salary: $' || TO\_CHAR(rec.salary, '999,999.00'));

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('--- Applying ' || p\_bonus\_percentage || '% Bonus to Department ' || p\_department\_id || ' ---');

UPDATE EMPLOYEES

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

WHERE department\_id = p\_department\_id;

v\_updated\_count := SQL%ROWCOUNT;

DBMS\_OUTPUT.PUT\_LINE('After bonus update for Department ' || p\_department\_id || ':');

FOR rec IN (SELECT employee\_id, employee\_name, salary FROM EMPLOYEES WHERE department\_id = p\_department\_id ORDER BY employee\_id) LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || rec.employee\_id || ', Name: ' || rec.employee\_name || ', New Salary: $' || TO\_CHAR(rec.salary, '999,999.00'));

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(v\_updated\_count || ' employee(s) in Department ' || p\_department\_id || ' had their salaries updated.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END UpdateEmployeeBonus;

/

BEGIN

UpdateEmployeeBonus(p\_department\_id => 10, p\_bonus\_percentage => 5);

END;

/

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_source\_account\_id IN NUMBER,

p\_destination\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_source\_balance NUMBER(10, 2);

e\_insufficient\_funds EXCEPTION;

BEGIN

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

DBMS\_OUTPUT.PUT\_LINE('Before transfer:');

FOR rec IN (SELECT account\_id, customer\_name, balance FROM BANK\_ACCOUNTS WHERE account\_id IN (p\_source\_account\_id, p\_destination\_account\_id) ORDER BY account\_id) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.account\_id || ', Customer: ' || rec.customer\_name || ', Balance: $' || TO\_CHAR(rec.balance, '999,999.00'));

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('--- Attempting to transfer $' || TO\_CHAR(p\_amount, '999,999.00') ||

' from Account ' || p\_source\_account\_id || ' to Account ' || p\_destination\_account\_id || ' ---');

SELECT balance INTO v\_source\_balance

FROM BANK\_ACCOUNTS

WHERE account\_id = p\_source\_account\_id

FOR UPDATE OF balance;

IF v\_source\_balance < p\_amount THEN

RAISE e\_insufficient\_funds;

END IF;

UPDATE BANK\_ACCOUNTS

SET balance = balance - p\_amount

WHERE account\_id = p\_source\_account\_id;

UPDATE BANK\_ACCOUNTS

SET balance = balance + p\_amount

WHERE account\_id = p\_destination\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer of $' || TO\_CHAR(p\_amount, '999,999.00') ||

' from Account ' || p\_source\_account\_id ||

' to Account ' || p\_destination\_account\_id || ' completed successfully.');

DBMS\_OUTPUT.PUT\_LINE('After transfer:');

FOR rec IN (SELECT account\_id, customer\_name, balance FROM BANK\_ACCOUNTS WHERE account\_id IN (p\_source\_account\_id, p\_destination\_account\_id) ORDER BY account\_id) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.account\_id || ', Customer: ' || rec.customer\_name || ', New Balance: $' || TO\_CHAR(rec.balance, '999,999.00'));

END LOOP;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: One or both accounts not found.');

WHEN e\_insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient funds in source account ' || p\_source\_account\_id || '. Available: $' || TO\_CHAR(v\_source\_balance, '999,999.00') || ', Requested: $' || TO\_CHAR(p\_amount, '999,999.00'));

WHEN OTHERS THEN

ROLLBACK;

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

END TransferFunds;

/

BEGIN

TransferFunds(p\_source\_account\_id => 1, p\_destination\_account\_id => 2, p\_amount => 500.00);

END;

/

BEGIN

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

TransferFunds(p\_source\_account\_id => 3, p\_destination\_account\_id => 4, p\_amount => 1000.00);

END;

/

BEGIN

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

TransferFunds(p\_source\_account\_id => 99, p\_destination\_account\_id => 1, p\_amount => 100.00);

END;

/

**Output Screenshot:**

