PL/SQL:

**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.

CREATE TABLE customers (customer\_id NUMBER PRIMARY KEY, name VARCHAR2(100), age NUMBER, loan\_interest\_rate NUMBER(5,2), balance NUMBER, isvip VARCHAR2(5));

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');

COMMIT;

BEGIN

FOR cust IN (SELECT customer\_id, age, loan\_interest\_rate FROM customers) LOOP

IF cust.age > 60 THEN

UPDATE customers

SET loan\_interest\_rate = loan\_interest\_rate - 0.01

WHERE customer\_id = cust.customer\_id;

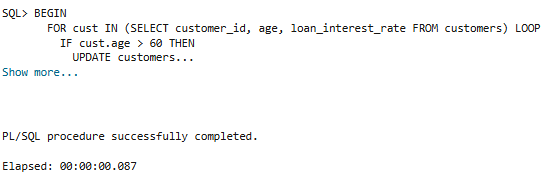
END IF;

END LOOP;

COMMIT;

END;

**Output:**



**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.

BEGIN

FOR cust IN (SELECT customer\_id, balance FROM customers) LOOP

IF cust.balance > 10000 THEN

UPDATE customers SET IsVIP = 'TRUE'

WHERE customer\_id = cust.customer\_id;

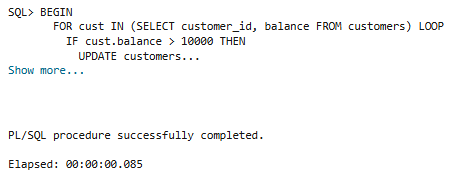
END IF;

END LOOP;

COMMIT;

END;

**Output:**



**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.

CREATE TABLE loans (loan\_id NUMBER PRIMARY KEY, customer\_id NUMBER, due\_date DATE);

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

INSERT INTO loans VALUES (2, 102, SYSDATE + 5);

INSERT INTO loans VALUES (3, 103, SYSDATE + 40);

COMMIT;

BEGIN

FOR loan IN (

SELECT customer\_id, due\_date

FROM loans

WHERE due\_date <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Loan for customer ' || loan.customer\_id ||

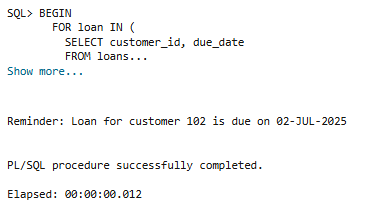
' is due on ' || TO\_CHAR(loan.due\_date, 'DD-MON-YYYY')

);

END LOOP;

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.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (SELECT account\_id, balance FROM savings\_accounts) LOOP

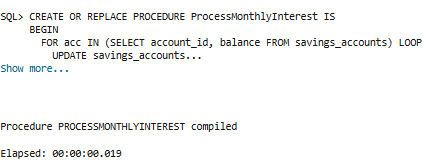
UPDATE savings\_accounts SET balance = balance + (balance \* 0.01) WHERE account\_id= acc.account\_id;

END LOOP;

COMMIT;

END;

**Output:**

****

**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.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_id IN NUMBER,

bonus\_percent IN NUMBER) IS

BEGIN

UPDATE employees

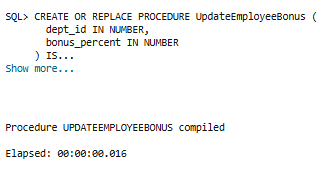
SET salary = salary + (salary \* (bonus\_percent / 100))

WHERE department\_id = dept\_id;

COMMIT;

END;

**Output:**

****

**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.

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_account IN NUMBER,

to\_account IN NUMBER,

amount IN NUMBER) IS

from\_balance NUMBER;

BEGIN

SELECT balance INTO from\_balance

FROM accounts

WHERE account\_id = from\_account

FOR UPDATE;

IF from\_balance >= amount THEN

UPDATE accounts

SET balance = balance - amount

WHERE account\_id = from\_account;

UPDATE accounts

SET balance = balance + amount

WHERE account\_id = to\_account;

COMMIT;

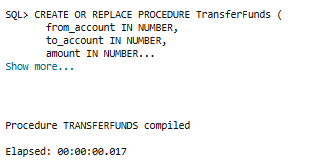
ELSE

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

END IF;

END;

**Output:**

****