**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:**

**BEGIN**

**FOR cust\_rec IN (**

**SELECT CustomerID, DOB**

**FROM Customers**

**) LOOP**

**IF (MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB)/12) > 60 THEN**

**UPDATE Loans**

**SET InterestRate = InterestRate - 1**

**WHERE CustomerID = cust\_rec.CustomerID;**

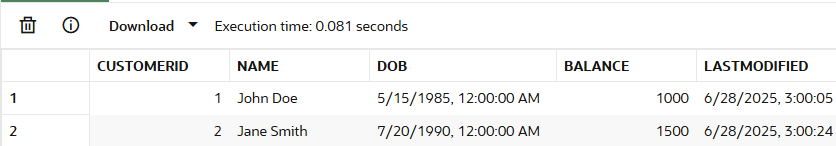
**END IF;**

**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.

**CODE:**

*Adding IsVIP Column in Customers*

ALTER TABLE Customers ADD (IsVIP VARCHAR2(5));

*Looping query to set value for IsVip*

BEGIN

  FOR cust\_rec IN (

    SELECT CustomerID, Balance

    FROM Customers

  ) LOOP

    IF cust\_rec.Balance > 10000 THEN

      UPDATE Customers

      SET IsVIP = 'TRUE'

      WHERE CustomerID = cust\_rec.CustomerID;

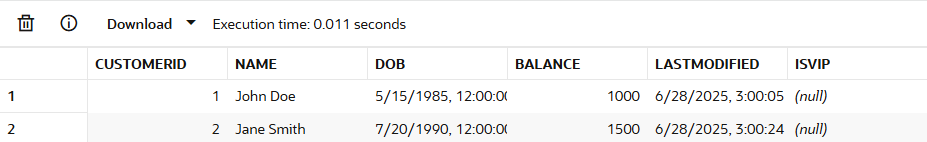
    END IF;

  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.

**CODE:**

**DECLARE**

  v\_name Customers.Name%TYPE;

**BEGIN**

**FOR loan\_rec IN** (

    SELECT CustomerID, EndDate

    FROM Loans

    WHERE EndDate <= SYSDATE + 30

  ) **LOOP**

    SELECT Name INTO v\_name

    FROM Customers

    WHERE CustomerID = loan\_rec.CustomerID;

    DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || v\_name ||

                         ', your loan is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-Mon-YYYY'));

**END LOOP;**

**END;**

**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

* + **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

**CODE:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_acc\_id IN Accounts.AccountID%TYPE,

p\_to\_acc\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

) IS

v\_balance Accounts.Balance%TYPE;

**BEGIN**

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_acc\_id;

IF v\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_acc\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_acc\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer completed successfully.');

**EXCEPTION**

WHEN OTHERS THEN

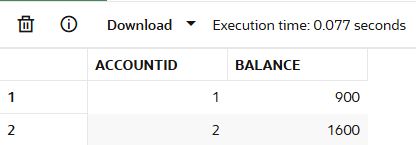
ROLLBACK;

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

**END;**

**OUTPUT:**

*EXEC SafeTransferFunds(1, 2, 100)*

*;*****

**Scenario 2:** Manage errors when updating employee salaries.

* + **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

**CODE:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_emp\_id IN Employees.EmployeeID%TYPE,

p\_percent IN NUMBER

) IS

**BEGIN**

-- Attempt to update salary

UPDATE Employees

SET Salary = Salary + (Salary \* p\_percent / 100)

WHERE EmployeeID = p\_emp\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee not found.');

END IF;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully.');

**EXCEPTION**

WHEN OTHERS THEN

ROLLBACK;

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

**END;**

**OUTPUT:**

*EXEC UpdateSalary(1, 10);*



**Scenario 3:** Ensure data integrity when adding a new customer.

* + **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

**CODE:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_cust\_id IN Customers.CustomerID%TYPE,

p\_name IN Customers.Name%TYPE,

p\_dob IN Customers.DOB%TYPE,

p\_balance IN Customers.Balance%TYPE

) IS

**BEGIN**

-- Insert new customer

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_cust\_id, p\_name, p\_dob, p\_balance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully.');

**EXCEPTION**

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_cust\_id || ' already exists.');

WHEN OTHERS THEN

ROLLBACK;

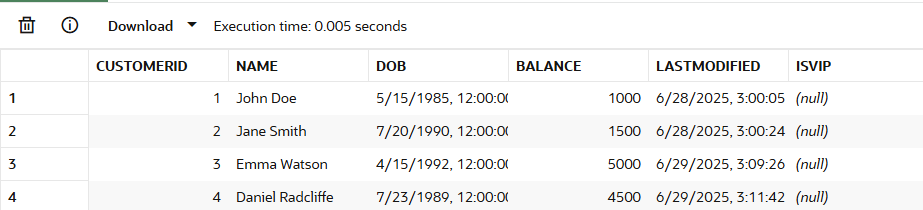
DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

**END;**

**OUPUT:**

*EXEC AddNewCustomer(3, 'Emma Watson', TO\_DATE('1992-04-15', 'YYYY-MM-DD'), 5000);*

*EXEC AddNewCustomer(4, 'Daniel Radcliffe', TO\_DATE('1989-07-23', 'YYYY-MM-DD'), 4500);*



**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:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

**BEGIN**

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

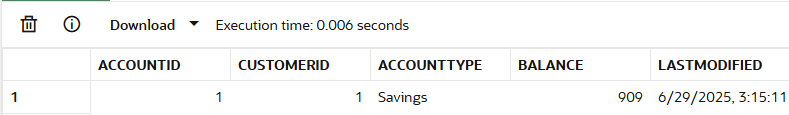
WHERE AccountType = 'Savings';

COMMIT;

**END;**

**OUTPUT:**

*EXEC ProcessMonthlyInterest;*



**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:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN Employees.Department%TYPE,

p\_bonus\_pct IN NUMBER

) IS

**BEGIN**

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_pct / 100)

WHERE Department = p\_department;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('No employees found in department: ' || p\_department);

ELSE

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' employee(s) received bonus in department: ' || p\_department);

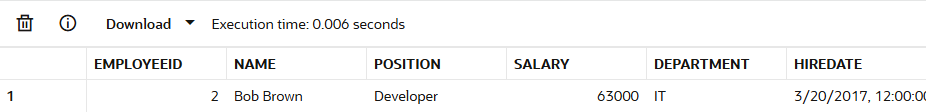
END IF;

**COMMIT;**

**END;**

**OUTPUT:**

*EXEC UpdateEmployeeBonus('IT', 5);*



**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:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_acc\_id IN Accounts.AccountID%TYPE,

p\_to\_acc\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

) IS

v\_balance Accounts.Balance%TYPE;

**BEGIN**

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_acc\_id;

IF v\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_acc\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_acc\_id;

**COMMIT;**

DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: One or both account IDs are invalid.');

WHEN OTHERS THEN

**ROLLBACK;**

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

**END;**

**OUTPUT:**

*EXEC TransferFunds(1, 2, 200)*

*;*