**Week-2 PLSQL Exercises**

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:**

CREATE TABLE customers (

  customer\_id NUMBER PRIMARY KEY,

  name VARCHAR2(100),

  age NUMBER,

  balance NUMBER(10, 2),

  isVIP VARCHAR2(5) DEFAULT 'FALSE'

);

CREATE TABLE loans (

  loan\_id NUMBER PRIMARY KEY,

  customer\_id NUMBER,

  interest\_rate NUMBER(5, 2),

  due\_date DATE,

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

);

-- Insert values

INSERT INTO customers VALUES (1, 'Arjun', 65, 15000, 'FALSE');

INSERT INTO customers VALUES (2, 'Priya', 40, 9000, 'FALSE');

INSERT INTO customers VALUES (3, 'Ravi', 70, 12000, 'FALSE');

INSERT INTO loans VALUES (101, 1, 9.5, SYSDATE + 10);

INSERT INTO loans VALUES (102, 2, 8.0, SYSDATE + 40);

INSERT INTO loans VALUES (103, 3, 10.0, SYSDATE + 5);

COMMIT;

-- PL/SQL block to update interest rates and show output

BEGIN

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

    UPDATE loans

    SET interest\_rate = interest\_rate - 1

    WHERE customer\_id = rec.customer\_id;

    DBMS\_OUTPUT.PUT\_LINE('Discount applied to customer ID: ' || rec.customer\_id);

  END LOOP;

END;

/

**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:**

BEGIN

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

    UPDATE customers

    SET isVIP = 'TRUE'

    WHERE customer\_id = rec.customer\_id;

  END LOOP;

  DBMS\_OUTPUT.PUT\_LINE('VIP status updated.');

END;

/

**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:**

BEGIN

  FOR rec IN (

    SELECT c.name, l.due\_date

    FROM loans l

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

    WHERE l.due\_date <= SYSDATE + 30

  ) LOOP

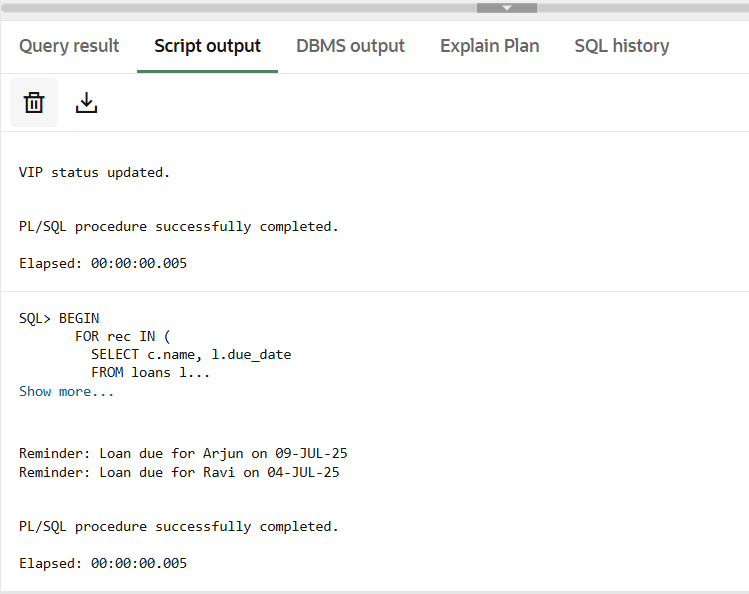
    DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan due for ' || rec.name || ' on ' || rec.due\_date);

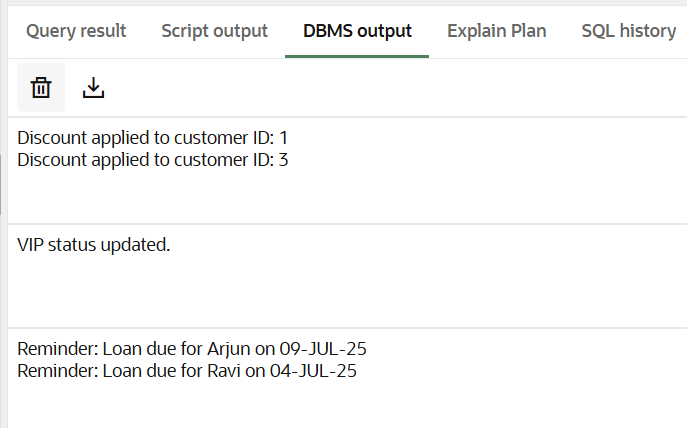
  END LOOP;

END;

/

**Output:**





1. **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:**

-- Table for savings accounts

CREATE TABLE savings\_accounts (

    account\_id NUMBER PRIMARY KEY,

    customer\_id NUMBER,

    balance NUMBER(10, 2)

);

-- Sample data

INSERT INTO savings\_accounts VALUES (1, 101, 10000);

INSERT INTO savings\_accounts VALUES (2, 102, 5000);

INSERT INTO savings\_accounts VALUES (3, 103, 20000);

COMMIT;

-- Procedure to apply 1% interest

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;

END;

/

**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:**

-- Execute the procedure

BEGIN

  ProcessMonthlyInterest;

END;

/

-- Employee table

CREATE TABLE employees (

    emp\_id NUMBER PRIMARY KEY,

    name VARCHAR2(100),

    department VARCHAR2(50),

    salary NUMBER(10, 2)

);

-- Sample data

INSERT INTO employees VALUES (1, 'Arun', 'HR', 40000);

INSERT INTO employees VALUES (2, 'Priya', 'Sales', 45000);

INSERT INTO employees VALUES (3, 'Ravi', 'Sales', 48000);

COMMIT;

-- Procedure to update bonus by department

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

  dept\_name IN VARCHAR2,

  bonus\_percent IN NUMBER

) IS

BEGIN

  UPDATE employees

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

  WHERE department = dept\_name;

END;

/

**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:**

BEGIN

  UpdateEmployeeBonus('Sales', 10);

END;

/

-- Account table

CREATE TABLE accounts (

    account\_id NUMBER PRIMARY KEY,

    customer\_id NUMBER,

    balance NUMBER(10, 2)

);

-- Sample data

INSERT INTO accounts VALUES (100, 1, 5000);

INSERT INTO accounts VALUES (200, 2, 3000);

COMMIT;

-- Procedure to transfer funds

CREATE OR REPLACE PROCEDURE TransferFunds(

  from\_account IN NUMBER,

  to\_account IN NUMBER,

  amount IN NUMBER

) IS

  insufficient\_balance EXCEPTION;

BEGIN

  -- Check balance

  DECLARE

    available\_balance NUMBER;

  BEGIN

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

    IF available\_balance < amount THEN

      RAISE insufficient\_balance;

    END IF;

  END;

  -- Deduct from sender

  UPDATE accounts

  SET balance = balance - amount

  WHERE account\_id = from\_account;

  -- Credit to receiver

  UPDATE accounts

  SET balance = balance + amount

  WHERE account\_id = to\_account;

EXCEPTION

  WHEN insufficient\_balance THEN

    DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in account ' || from\_account);

END;

/

-- Execute the procedure

BEGIN

  TransferFunds(100, 200, 1000);

END;

/

**Output:**

