Name : Arshiya Tabassum A

Superset ID : 6424209

**WEEK 2**

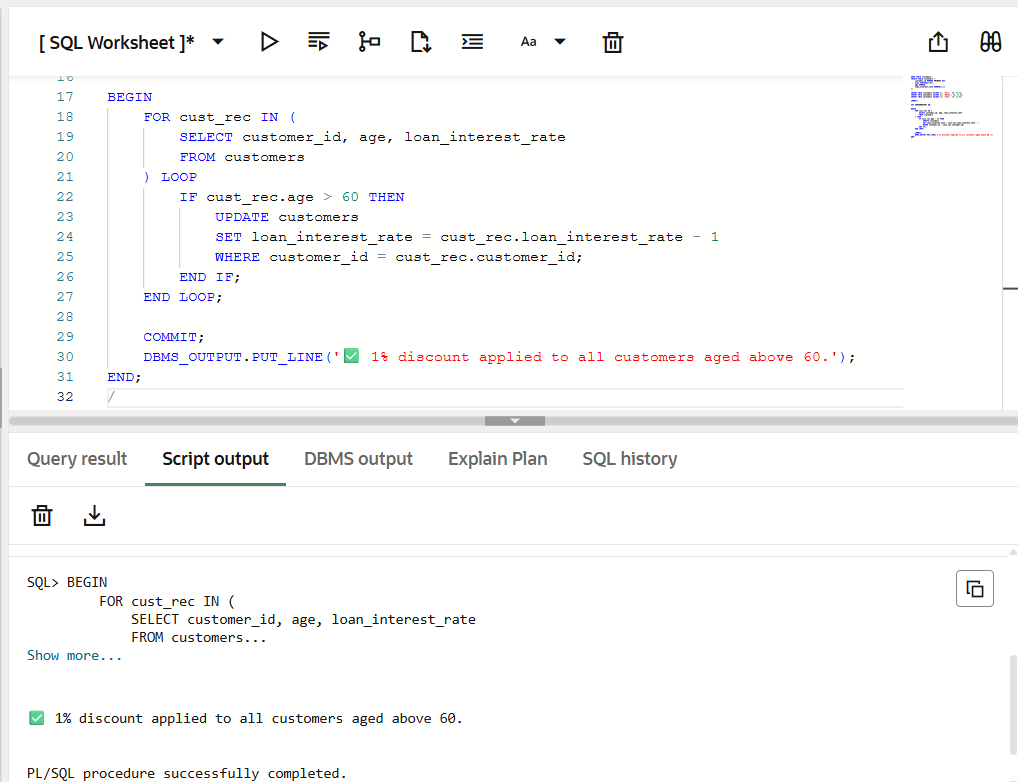
**PL SQL HANDS ON**

**Module 3**

**Exercise 1: Control Structures**

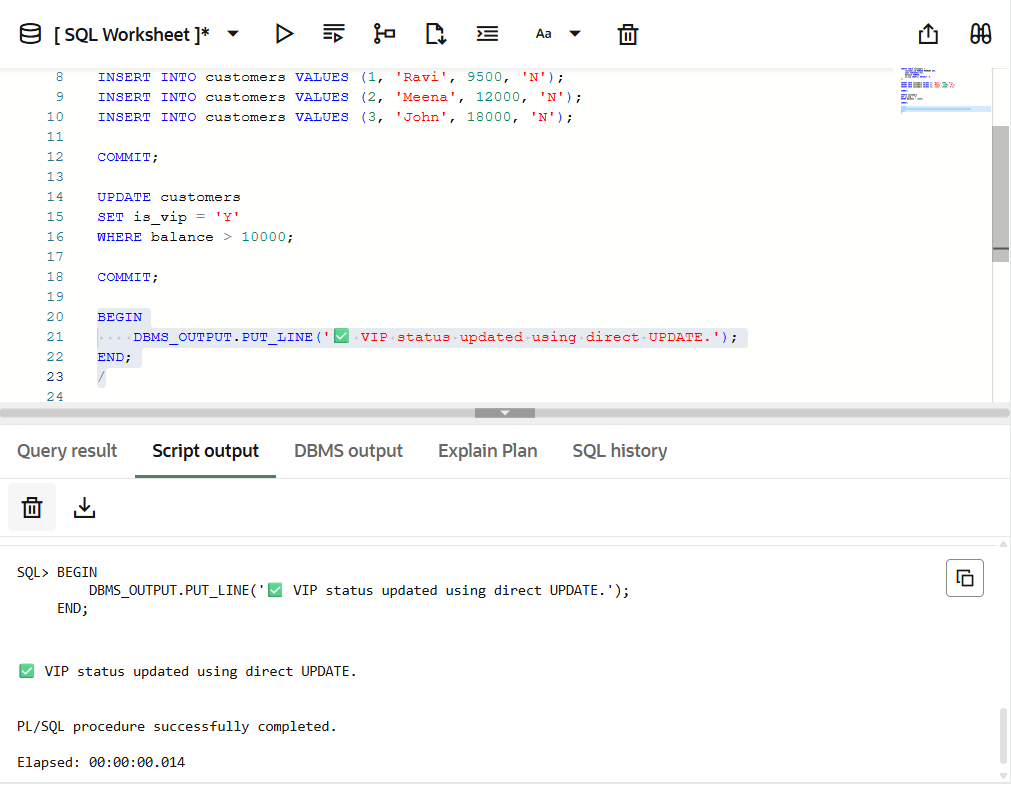
Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.

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.

****

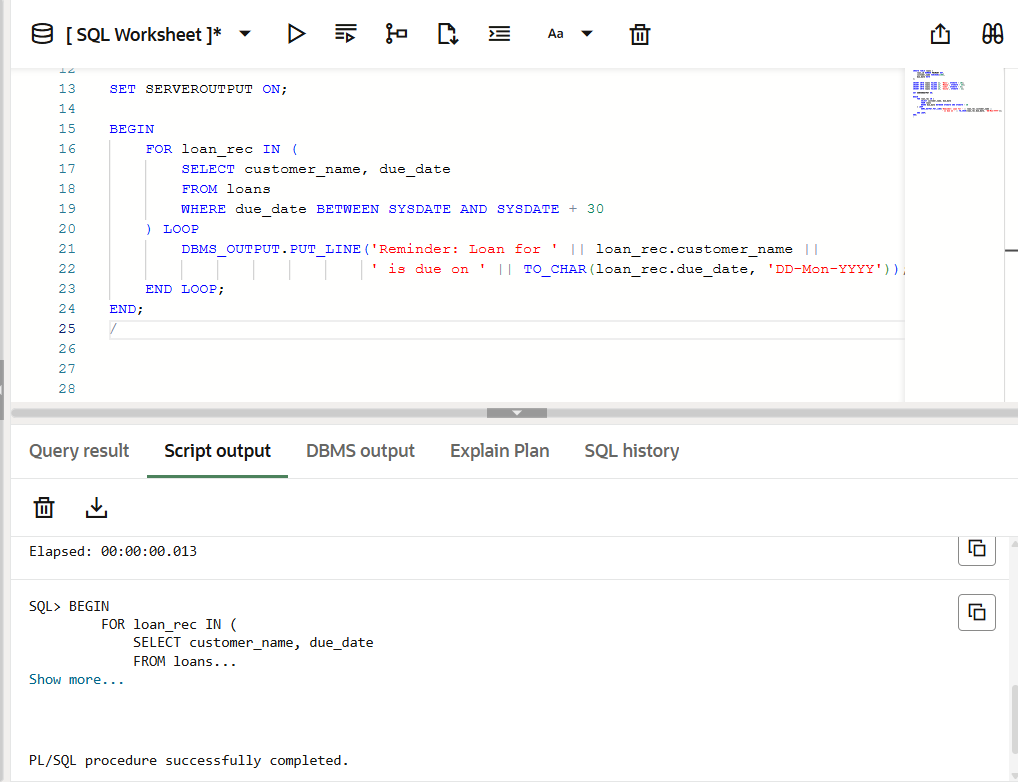
Scenario 2: A customer can be promoted to VIP status based on their balance.

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.



**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 TABLE accounts (

    account\_id NUMBER PRIMARY KEY,

    account\_holder VARCHAR2(100),

    balance NUMBER

);

INSERT INTO accounts VALUES (1, 'Ravi', 5000);

INSERT INTO accounts VALUES (2, 'Meena', 7000);

COMMIT;

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

    from\_account\_id IN NUMBER,

    to\_account\_id IN NUMBER,

    amount IN NUMBER

) IS

    from\_balance NUMBER;

    insufficient\_funds EXCEPTION;

BEGIN

    SELECT balance INTO from\_balance

    FROM accounts

    WHERE account\_id = from\_account\_id;

    IF from\_balance < amount THEN

        RAISE 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;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || amount || ' successful from Account ' ||

                         from\_account\_id || ' to Account ' || to\_account\_id);

EXCEPTION

    WHEN insufficient\_funds THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance in Account ' || from\_account\_id);

    WHEN OTHERS THEN

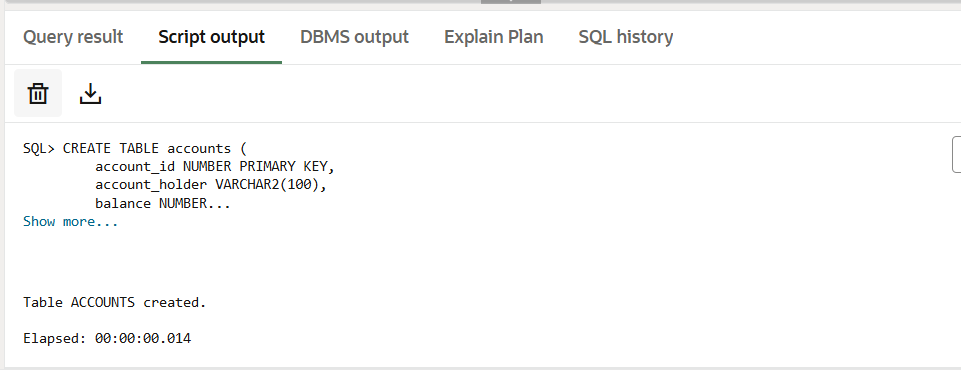
        ROLLBACK;

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

END;

/

**OUTPUT :**



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 TABLE employees (

    employee\_id NUMBER PRIMARY KEY,

    employee\_name VARCHAR2(100),

    salary NUMBER);

INSERT INTO employees VALUES (101, 'Ravi', 40000);

INSERT INTO employees VALUES (102, 'Meena', 55000);

INSERT INTO employees VALUES (103, 'John', 30000);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateSalary (

    p\_employee\_id IN NUMBER,

    p\_percentage IN NUMBER

)

IS v\_current\_salary NUMBER;

BEGIN

    SELECT salary INTO v\_current\_salary

    FROM employees

    WHERE employee\_id = p\_employee\_id;

    UPDATE employees

    SET salary = salary + (v\_current\_salary \* p\_percentage / 100)

    WHERE employee\_id = p\_employee\_id;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully for Employee ID: ' || p\_employee\_id);

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_employee\_id || ' not found.');

    WHEN OTHERS THEN

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

END;

/

BEGIN

    UpdateSalary(101, 10);

END;

/

**OUTPUT :**



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 TABLE customers (

    customer\_id NUMBER PRIMARY KEY,

    customer\_name VARCHAR2(100),

    email VARCHAR2(100)

);

INSERT INTO customers VALUES (1, 'Ravi', 'ravi@email.com');

INSERT INTO customers VALUES (2, 'Meena', 'meena@email.com');

COMMIT;

CREATE OR REPLACE PROCEDURE AddNewCustomer (

    p\_customer\_id IN NUMBER,

    p\_customer\_name IN VARCHAR2,

    p\_email IN VARCHAR2

)

IS BEGIN

    INSERT INTO customers (customer\_id, customer\_name, email)

    VALUES (p\_customer\_id, p\_customer\_name, p\_email);

    COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('Customer added successfully: ' || p\_customer\_name);

EXCEPTION

    WHEN DUP\_VAL\_ON\_INDEX THEN

        DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID ' || p\_customer\_id || ' already exists. Insertion skipped.');

    WHEN OTHERS THEN

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

END;

/

BEGIN

    AddNewCustomer(3, 'John', 'john@email.com');

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.

**CODE :**

CREATE TABLE accounts (

    account\_id NUMBER PRIMARY KEY,

    account\_holder VARCHAR2(100),

    balance NUMBER,

    account\_type VARCHAR2(20)

);

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

INSERT INTO accounts VALUES (2, 'Meena', 15000, 'current');

INSERT INTO accounts VALUES (3, 'John', 20000, 'savings');

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

    UPDATE accounts

    SET balance = balance + (balance \* 0.01)

    WHERE account\_type = 'savings';

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all savings accounts.');

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

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

END;

/

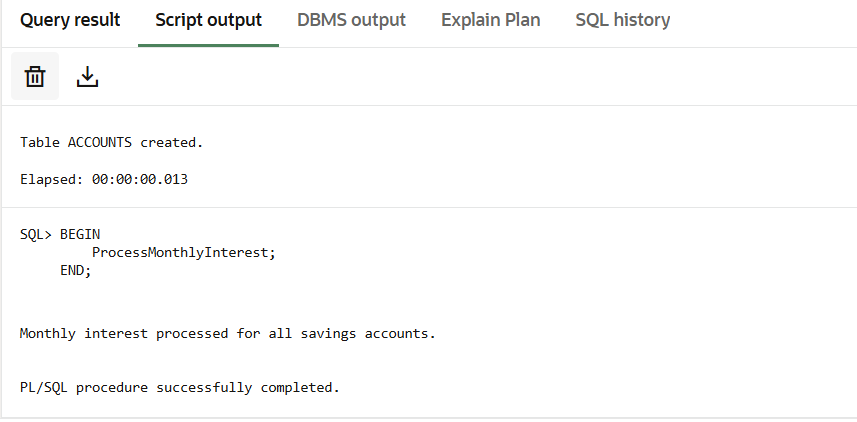
BEGIN

    ProcessMonthlyInterest;

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.

**CODE :**SELECT table\_name FROM user\_tables WHERE table\_name = 'EMPLOYEES';

CREATE TABLE employees (

    employee\_id NUMBER PRIMARY KEY,

    employee\_name VARCHAR2(100),

    salary NUMBER,

    department VARCHAR2(50)

);

DROP TABLE employees;

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

INSERT INTO employees VALUES (102, 'Meena', 55000, 'Finance');

INSERT INTO employees VALUES (103, 'John', 30000, 'HR');

INSERT INTO employees VALUES (104, 'Anita', 60000, 'IT');

COMMIT;

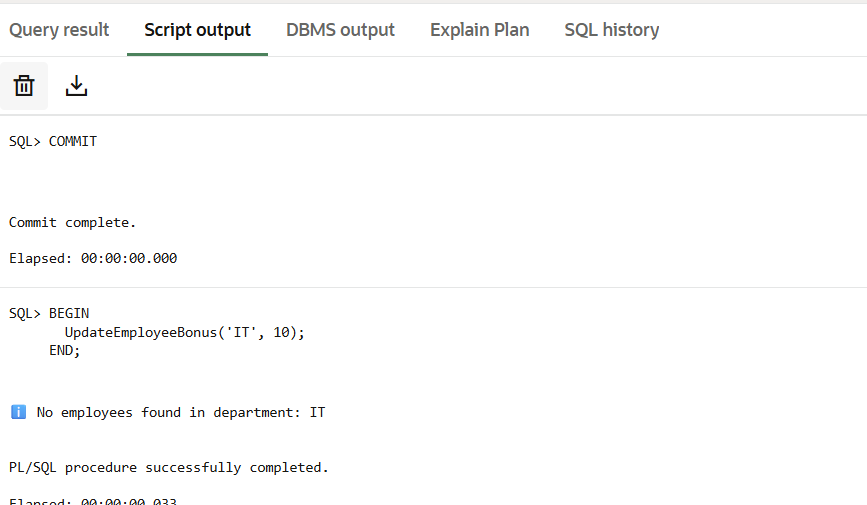
BEGIN

  UpdateEmployeeBonus('IT', 10);

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.

**CODE :**

DROP TABLE accounts;

CREATE TABLE accounts (

    account\_id NUMBER PRIMARY KEY,

    customer\_name VARCHAR2(100),

    balance NUMBER

);

INSERT INTO accounts VALUES (1, 'Ravi', 10000);

INSERT INTO accounts VALUES (2, 'Meena', 5000);

INSERT INTO accounts VALUES (3, 'John', 3000);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

    p\_from\_account IN NUMBER,

    p\_to\_account IN NUMBER,

    p\_amount IN NUMBER

)

IS v\_balance NUMBER;

BEGIN

    SELECT balance INTO v\_balance

    FROM accounts

    WHERE account\_id = p\_from\_account

    FOR UPDATE;

    IF v\_balance < p\_amount THEN

        DBMS\_OUTPUT.PUT\_LINE(' Transfer failed: Insufficient funds in source account.');

        RETURN;

    END IF;

    UPDATE accounts

    SET balance = balance - p\_amount

    WHERE account\_id = p\_from\_account;

    UPDATE accounts

    SET balance = balance + p\_amount

    WHERE account\_id = p\_to\_account;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ₹' || p\_amount || ' transferred from Account ' || p\_from\_account || ' to Account ' || p\_to\_account);

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        DBMS\_OUTPUT.PUT\_LINE('One of the accounts does not exist.');

        ROLLBACK;

    WHEN OTHERS THEN

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

        ROLLBACK;

END;

/

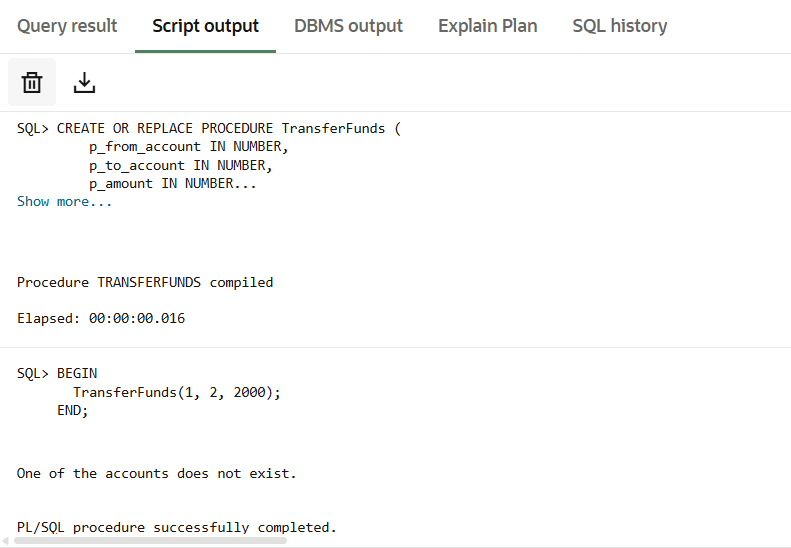
BEGIN

  TransferFunds(1, 2, 2000);

END;

/

**OUTPUT :**

****

**Exercise 4: Functions**

Scenario 1: Calculate the age of customers for eligibility checks.

Question: Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

**CODE :**

CREATE OR REPLACE FUNCTION CalculateAge (

    p\_dob DATE

)

RETURN NUMBER

IS  v\_age NUMBER;

BEGIN

    v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

    RETURN v\_age;

END;

/

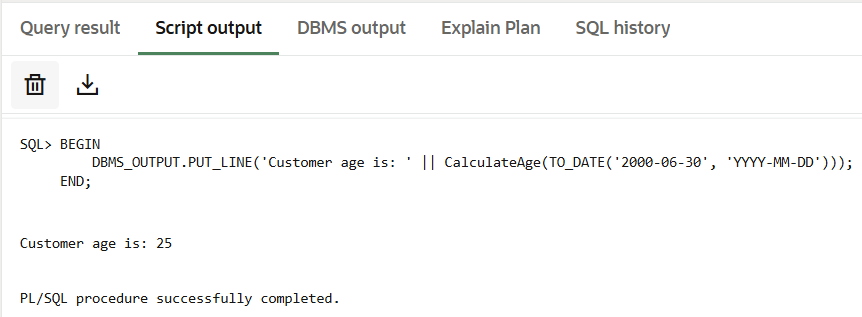
BEGIN

    DBMS\_OUTPUT.PUT\_LINE('Customer age is: ' || CalculateAge(TO\_DATE('2000-06-30', 'YYYY-MM-DD')));

END;

/

**OUTPUT :**

****

Scenario 2: The bank needs to compute the monthly installment for a loan.

Question: Write a function CalculateMonthlyInstallment that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

**CODE :**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

    p\_loan\_amount       IN NUMBER,

    p\_annual\_interest   IN NUMBER,

    p\_loan\_years        IN NUMBER

)

RETURN NUMBER

IS

    v\_monthly\_rate   NUMBER;

    v\_total\_months   NUMBER;

    v\_emi            NUMBER;

BEGIN

    v\_monthly\_rate := p\_annual\_interest / 12 / 100;

    v\_total\_months := p\_loan\_years \* 12;

    v\_emi := (p\_loan\_amount \* v\_monthly\_rate \* POWER(1 + v\_monthly\_rate, v\_total\_months)) /

             (POWER(1 + v\_monthly\_rate, v\_total\_months) - 1);

    RETURN ROUND(v\_emi, 2);

EXCEPTION

    WHEN ZERO\_DIVIDE THEN

        RETURN 0;

    WHEN OTHERS THEN

        RETURN -1;

END;

/

BEGIN

  DBMS\_OUTPUT.PUT\_LINE(

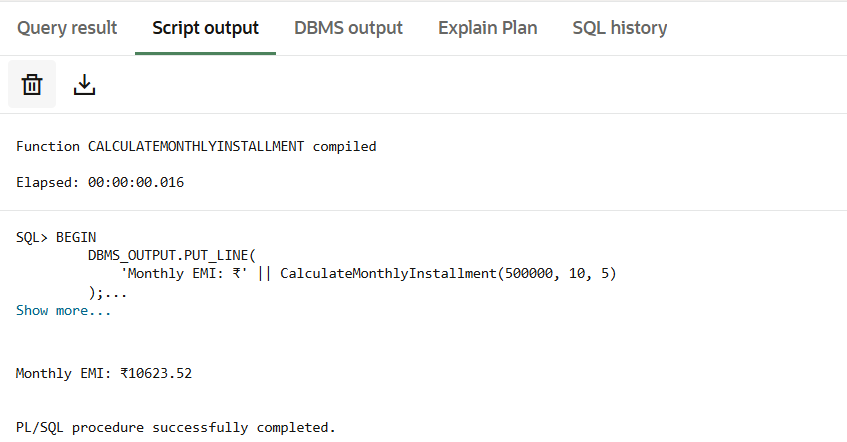
        'Monthly EMI: ₹' || CalculateMonthlyInstallment(500000, 10, 5)

  );

END;

/

**OUTPUT :**

****

Scenario 3: Check if a customer has sufficient balance before making a transaction.

Question: Write a function HasSufficientBalance that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

**CODE :**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

    p\_account\_id IN NUMBER,

    p\_amount     IN NUMBER

)

RETURN BOOLEAN

IS

    v\_balance NUMBER;

BEGIN

    SELECT balance INTO v\_balance

    FROM accounts

    WHERE account\_id = p\_account\_id;

    IF v\_balance >= p\_amount THEN

        RETURN TRUE;

    ELSE

        RETURN FALSE;

    END IF;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN FALSE;

    WHEN OTHERS THEN

        RETURN FALSE;

END;/

BEGIN

    IF HasSufficientBalance(1, 2000) THEN

        DBMS\_OUTPUT.PUT\_LINE('Sufficient balance.');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('Insufficient balance.');

    END IF;

END;/

CREATE TABLE accounts (

    account\_id NUMBER PRIMARY KEY,

    customer\_name VARCHAR2(100),

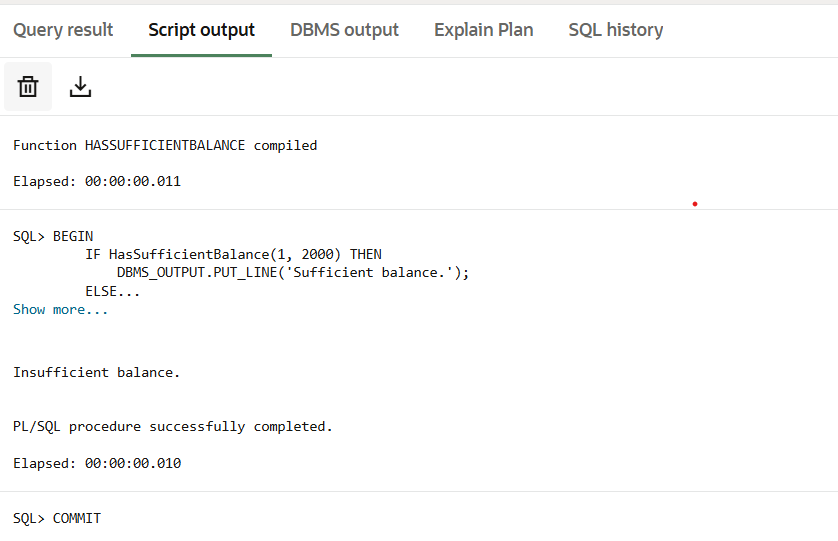
    balance NUMBER);

INSERT INTO accounts VALUES (1, 'Ravi', 10000);

INSERT INTO accounts VALUES (2, 'Meena', 4000);

COMMIT;

**OUTPUT :**

****

**Exercise 5: Triggers**

Scenario 1: Automatically update the last modified date when a customer's record is updated.

Question: Write a trigger UpdateCustomerLastModified that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

**CODE :**

DROP TABLE customers;

CREATE TABLE customers (

    customer\_id   NUMBER PRIMARY KEY,

    name          VARCHAR2(100),

    balance       NUMBER,

    last\_modified DATE

);

INSERT INTO customers VALUES (1, 'Ravi', 8000, NULL);

INSERT INTO customers VALUES (2, 'Meena', 12000, NULL);

COMMIT;

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON customers

FOR EACH ROW

BEGIN

    :NEW.last\_modified := SYSDATE;

END;

/

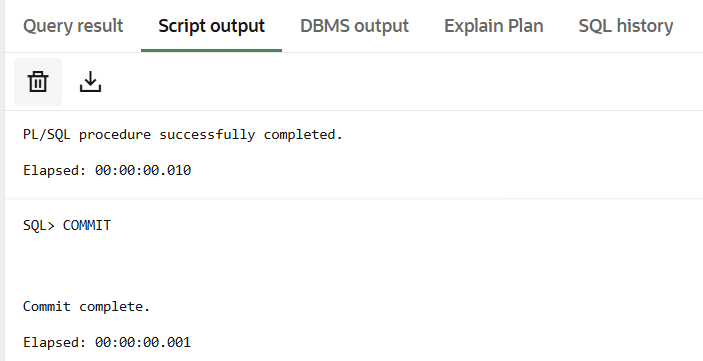
UPDATE customers

SET balance = balance + 1000

WHERE customer\_id = 1;

COMMIT;

**OUTPUT :**

****

Scenario 2: Maintain an audit log for all transactions.

Question: Write a trigger LogTransaction that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

**CODE :**

CREATE TABLE transactions (

    transaction\_id   NUMBER PRIMARY KEY,

    account\_id       NUMBER,

    amount           NUMBER,

    transaction\_type VARCHAR2(20),  -- e.g., 'credit' or 'debit'

    transaction\_date DATE

);

CREATE TABLE audit\_log (

    audit\_id         NUMBER GENERATED BY DEFAULT AS IDENTITY,

    transaction\_id   NUMBER,

    account\_id       NUMBER,

    action           VARCHAR2(20),

    amount           NUMBER,

    log\_time         DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON transactions

FOR EACH ROW

BEGIN

    INSERT INTO audit\_log (

        transaction\_id,

        account\_id,

        action,

        amount,

        log\_time

    ) VALUES (

        :NEW.transaction\_id,

        :NEW.account\_id,

        :NEW.transaction\_type,

        :NEW.amount,

        SYSDATE

    );

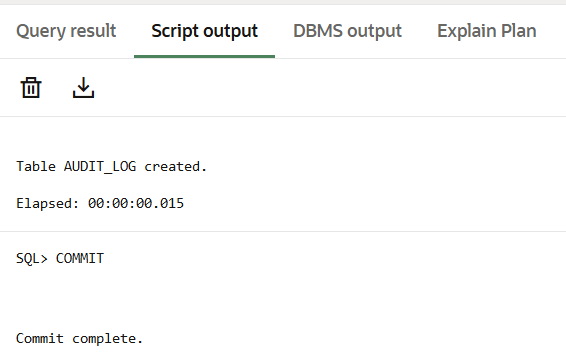
END;

/

INSERT INTO transactions VALUES (1, 101, 5000, 'credit', SYSDATE);

COMMIT;

**OUTPUT :**



Scenario 3: Enforce business rules on deposits and withdrawals.

Question: Write a trigger CheckTransactionRules that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

**CODE :**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

  v\_balance NUMBER;

BEGIN

  SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

  IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_balance THEN

    RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal exceeds available balance.');

  END IF;

  IF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

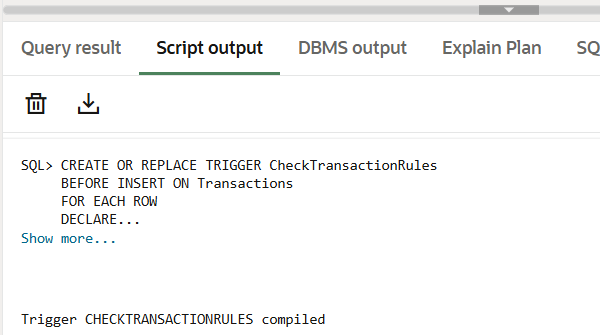
    RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be greater than zero.');

  END IF;

END;

/

**OUTPUT :**

****

**Exercise 6: Cursors**

Scenario 1: Generate monthly statements for all customers.

Question: Write a PL/SQL block using an explicit cursor GenerateMonthlyStatements that retrieves all transactions for the current month and prints a statement for each customer.

**CODE :**

SET SERVEROUTPUT ON;

DECLARE

  CURSOR transaction\_cursor IS

    SELECT t.TransactionID, t.AccountID, t.TransactionDate, t.Amount, t.TransactionType, c.Name

    FROM Transactions t

    JOIN Accounts a ON t.AccountID = a.AccountID

    JOIN Customers c ON a.CustomerID = c.CustomerID

    WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

  v\_transaction transaction\_cursor%ROWTYPE;

BEGIN

  OPEN transaction\_cursor;

  LOOP

    FETCH transaction\_cursor INTO v\_transaction;

    EXIT WHEN transaction\_cursor%NOTFOUND;

    DBMS\_OUTPUT.PUT\_LINE('Customer: ' || v\_transaction.Name ||

                         ' | Date: ' || TO\_CHAR(v\_transaction.TransactionDate, 'YYYY-MM-DD') ||

                         ' | Type: ' || v\_transaction.TransactionType ||

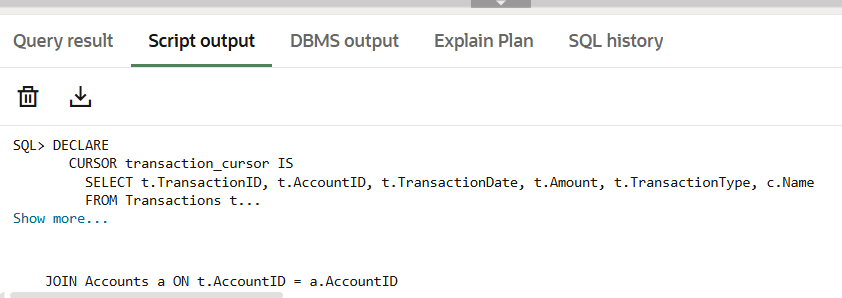
                         ' | Amount: ' || v\_transaction.Amount);

  END LOOP;

  CLOSE transaction\_cursor;

END;

**OUTPUT :**



Scenario 2: Apply annual fee to all accounts.

Questio.n: Write a PL/SQL block using an explicit cursor ApplyAnnualFee that deducts an annual maintenance fee from the balance of all accounts

**CODE :**

DECLARE

  CURSOR account\_cursor IS

    SELECT AccountID, Balance FROM Accounts;

  v\_account account\_cursor%ROWTYPE;

  v\_fee CONSTANT NUMBER := 100; -- Annual fee amount

BEGIN

  OPEN account\_cursor;

  LOOP

    FETCH account\_cursor INTO v\_account;

    EXIT WHEN account\_cursor%NOTFOUND;

    UPDATE Accounts

    SET Balance = Balance - v\_fee,

        LastModified = SYSDATE

    WHERE AccountID = v\_account.AccountID;

    DBMS\_OUTPUT.PUT\_LINE('Annual fee deducted from Account ID: ' || v\_account.AccountID);

  END LOOP;

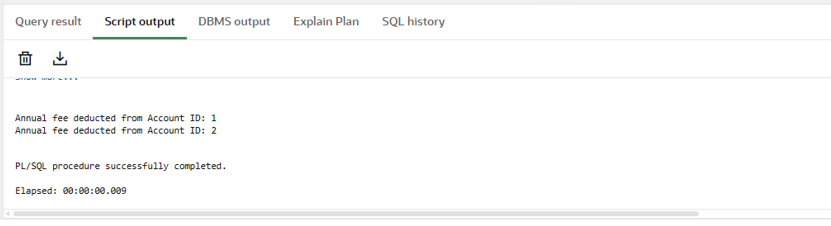
  CLOSE account\_cursor;

  COMMIT;

END;

/

**OUTPUT :**

****

Scenario 3: Update the interest rate for all loans based on a new policy.

Question: Write a PL/SQL block using an explicit cursor UpdateLoanInterestRates that fetches all loans and updates their interest rates based on the new policy.

**CODE :**

DECLARE

  CURSOR loan\_cursor IS

    SELECT LoanID, LoanAmount, InterestRate FROM Loans;

  v\_loan loan\_cursor%ROWTYPE;

  v\_new\_rate NUMBER;

BEGIN

  OPEN loan\_cursor;

  LOOP

    FETCH loan\_cursor INTO v\_loan;

    EXIT WHEN loan\_cursor%NOTFOUND;

    IF v\_loan.LoanAmount >= 10000 THEN

      v\_new\_rate := 6;

    ELSE

      v\_new\_rate := 5;

    END IF;

    UPDATE Loans

    SET InterestRate = v\_new\_rate

    WHERE LoanID = v\_loan.LoanID;

    DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || v\_loan.LoanID ||

                         ' | Old Rate: ' || v\_loan.InterestRate ||

                         ' | New Rate: ' || v\_new\_rate);

  END LOOP;

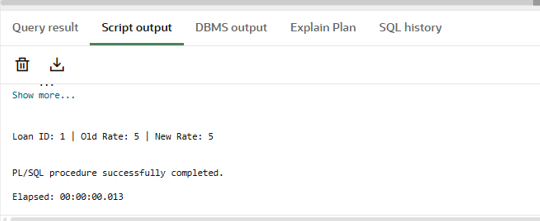
  CLOSE loan\_cursor;

  COMMIT;

END;

/

**OUTPUT:**



**Exercise 7: Packages**

Scenario 1: Group all customer-related procedures and functions into a package.

Question: Create a package CustomerManagement with procedures for adding a new customer, updating customer details, and a function to get customer balance.

**CODE :**

CREATE OR REPLACE PACKAGE CustomerManagement AS

  PROCEDURE AddCustomer(

    p\_customer\_id IN NUMBER,

    p\_name IN VARCHAR2,

    p\_dob IN DATE,

    p\_balance IN NUMBER

  );

  PROCEDURE UpdateCustomerDetails(

    p\_customer\_id IN NUMBER,

    p\_name IN VARCHAR2,

    p\_dob IN DATE

  );

  FUNCTION GetCustomerBalance(p\_customer\_id IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

  PROCEDURE AddCustomer(

    p\_customer\_id IN NUMBER,

    p\_name IN VARCHAR2,

    p\_dob IN DATE,

    p\_balance IN NUMBER

  ) IS

  BEGIN

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

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

    COMMIT;

  EXCEPTION

    WHEN DUP\_VAL\_ON\_INDEX THEN

      DBMS\_OUTPUT.PUT\_LINE('Customer already exists.');

  END;

  PROCEDURE UpdateCustomerDetails(

    p\_customer\_id IN NUMBER,

    p\_name IN VARCHAR2,

    p\_dob IN DATE

  ) IS

  BEGIN

    UPDATE Customers

    SET Name = p\_name,

        DOB = p\_dob,

        LastModified = SYSDATE

    WHERE CustomerID = p\_customer\_id;

    COMMIT;

  END;

  FUNCTION GetCustomerBalance(p\_customer\_id IN NUMBER) RETURN NUMBER IS

    v\_balance NUMBER;

  BEGIN

    SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_customer\_id;

    RETURN v\_balance;

  EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

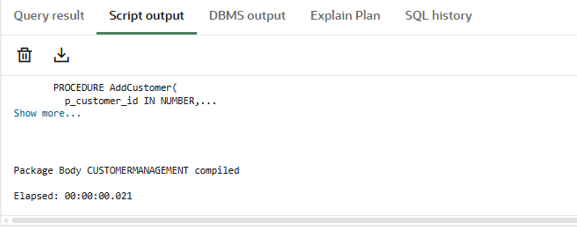
      RETURN 0;

  END;

END CustomerManagement;

/

**OUTPUT:**

****

Scenario 2: Create a package to manage employee data.

Question: Write a package EmployeeManagement with procedures to hire new employees, update employee details, and a function to calculate annual salary**.**

**CODE :**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

  PROCEDURE HireEmployee(

    p\_emp\_id IN NUMBER,

    p\_name IN VARCHAR2,

    p\_position IN VARCHAR2,

    p\_salary IN NUMBER,

    p\_department IN VARCHAR2,

    p\_hire\_date IN DATE

  );

  PROCEDURE UpdateEmployee(

    p\_emp\_id IN NUMBER,

    p\_salary IN NUMBER,

    p\_department IN VARCHAR2

  );

  FUNCTION CalculateAnnualSalary(p\_emp\_id IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

  PROCEDURE HireEmployee(

    p\_emp\_id IN NUMBER,

    p\_name IN VARCHAR2,

    p\_position IN VARCHAR2,

    p\_salary IN NUMBER,

    p\_department IN VARCHAR2,

    p\_hire\_date IN DATE

  ) IS

  BEGIN

    INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

    VALUES (p\_emp\_id, p\_name, p\_position, p\_salary, p\_department, p\_hire\_date);

    COMMIT;

  END;

  PROCEDURE UpdateEmployee(

    p\_emp\_id IN NUMBER,

    p\_salary IN NUMBER,

    p\_department IN VARCHAR2

  ) IS

  BEGIN

    UPDATE Employees

    SET Salary = p\_salary,

        Department = p\_department

    WHERE EmployeeID = p\_emp\_id;

    COMMIT;

  END;

  FUNCTION CalculateAnnualSalary(p\_emp\_id IN NUMBER) RETURN NUMBER IS

    v\_salary NUMBER;

  BEGIN

    SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_emp\_id;

    RETURN v\_salary \* 12;

  EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

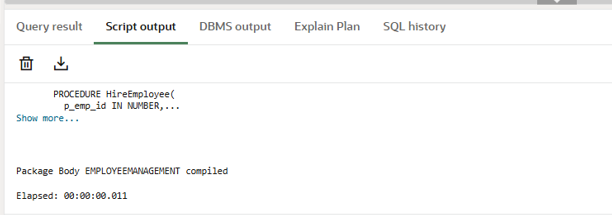
      RETURN 0;

  END;

END EmployeeManagement;

/

**OUTPUT:**

****

Scenario 3: Group all account-related operations into a package.

Question: Create a package AccountOperations with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

**CODE :**

CREATE OR REPLACE PACKAGE AccountOperations AS

  PROCEDURE OpenAccount(

    p\_account\_id IN NUMBER,

    p\_customer\_id IN NUMBER,

    p\_type IN VARCHAR2,

    p\_balance IN NUMBER

  );

  PROCEDURE CloseAccount(p\_account\_id IN NUMBER);

  FUNCTION GetTotalBalance(p\_customer\_id IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

  PROCEDURE OpenAccount(

    p\_account\_id IN NUMBER,

    p\_customer\_id IN NUMBER,

    p\_type IN VARCHAR2,

    p\_balance IN NUMBER

  ) IS

  BEGIN

    INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

    VALUES (p\_account\_id, p\_customer\_id, p\_type, p\_balance, SYSDATE);

    COMMIT;

  END;

  PROCEDURE CloseAccount(p\_account\_id IN NUMBER) IS

  BEGIN

    DELETE FROM Accounts WHERE AccountID = p\_account\_id;

    COMMIT;

  END;

  FUNCTION GetTotalBalance(p\_customer\_id IN NUMBER) RETURN NUMBER IS

    v\_total NUMBER;

  BEGIN

    SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_customer\_id;

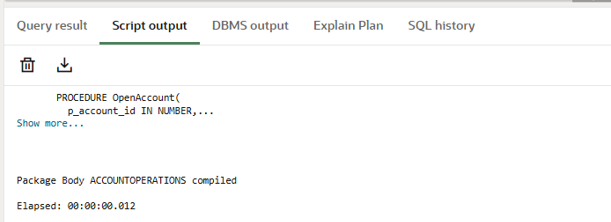
    RETURN NVL(v\_total, 0);

  END;

END AccountOperations;

/

**OUTPUT:**

****