**Week\_2\_Pl\_Sql**

**Exercise 1: Control Structures:**

**Scenario 1:**

BEGIN

    FOR cus IN (

        SELECT CUSTOMERID, DOB

        FROM CUSTOMERS

    ) LOOP

        IF Months\_Between(SYSDATE, cus.DOB)/12 < 60 THEN

            UPDATE LOANS

            SET INTERESTRATE = INTERESTRATE - 1

            WHERE CUSTOMERID = cus.CUSTOMERID;

            DBMS\_OUTPUT.PUT\_LINE('Updated interest rate for Customer ID: ' || cus.CustomerID);

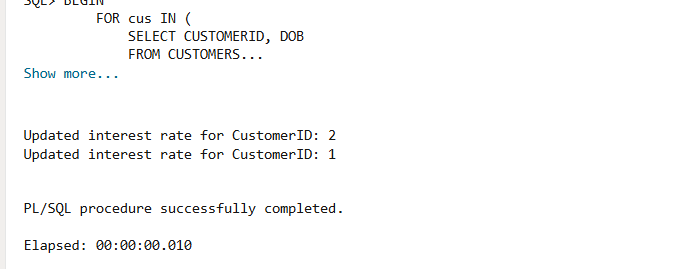
        END IF;

    END LOOP;

END;

/

**Output:**

****

**Scenario 2:**

BEGIN

   FOR rec IN (SELECT CustomerID, Balance FROM Customers)

   LOOP

      IF rec.Balance > 10000 THEN

         UPDATE Customers

         SET IsVIP = 'TRUE'

         WHERE CustomerID = rec.CustomerID;

         DBMS\_OUTPUT.PUT\_LINE('CustomerID ' || rec.CustomerID || ' promoted to VIP.');

      END IF;

   END LOOP;

   COMMIT;

END;

/

**Output:**

A computer screen shot of a computer code

AI-generated content may be incorrect.

**Scenario 1:**

BEGIN

   FOR rec IN (

      SELECT LoanID, CustomerID, EndDate

      FROM Loans

      WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30

   ) LOOP

      DBMS\_OUTPUT.PUT\_LINE(

         'Reminder: Loan ID ' || rec.LoanID ||

         ' for Customer ID ' || rec.CustomerID ||

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

      );

   END LOOP;

END;

/

**Output:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Exercise 2: Error Handling:**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

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\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

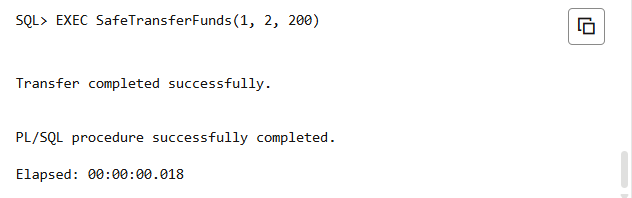
ROLLBACK;

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

END;

/

**Output:**

****

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary(

    p\_employee\_id IN NUMBER,

    p\_percentage  IN NUMBER

)

IS

BEGIN

    UPDATE Employees

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

    WHERE EmployeeID = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

        RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID does not exist.');

    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:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

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;

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

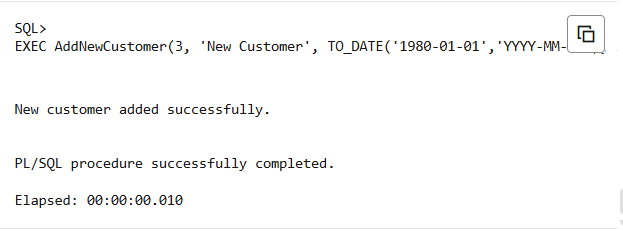
WHEN OTHERS THEN

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

END;

/

**Output:**

****

**Exercise 3: Stored Procedures:**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all Savings accounts.');

END;

/

EXEC ProcessMonthlyInterest;

**Output:**

**A white screen with black text

AI-generated content may be incorrect.**

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_department IN VARCHAR2, 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('Salaries updated for department: ' || p\_department);

   ELSE

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

   END IF;

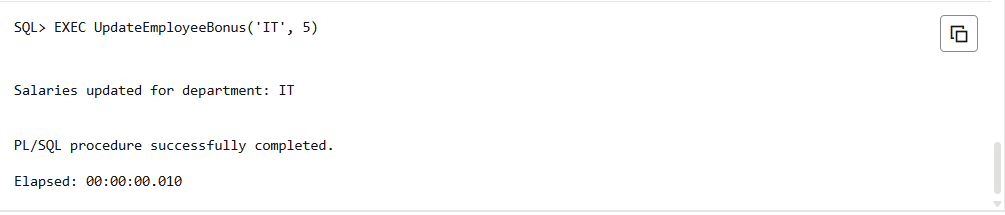
   COMMIT;

END;

/

EXEC UpdateEmployeeBonus('IT', 5);

**Output:**



**Scenario 3:**

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 AccountID = p\_from\_account

   FOR UPDATE;

   IF v\_balance < p\_amount THEN

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

   END IF;

   UPDATE Accounts

   SET Balance = Balance - p\_amount

   WHERE AccountID = p\_from\_account;

   UPDATE Accounts

   SET Balance = Balance + p\_amount

   WHERE AccountID = p\_to\_account;

   COMMIT;

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

EXCEPTION

   WHEN OTHERS THEN

      ROLLBACK;

      DBMS\_OUTPUT.PUT\_LINE('Error transferring funds: ' || SQLERRM);

END;

/

EXEC TransferFunds(1, 2, 300);

**Output:**

**A white background with black dots

AI-generated content may be incorrect.**

**Exercise 4: Functions:**

**Scenario 1:**

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob IN DATE) RETURN NUMBER

IS

    v\_age NUMBER;

BEGIN

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

    RETURN v\_age;

END;

/

SELECT CalculateAge(TO\_DATE('1975-08-11','YYYY-MM-DD')) AS Age FROM dual;

**Output:**

A computer screen shot of a person

AI-generated content may be incorrect.

**Scenario 2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

    p\_loan\_amount   IN NUMBER,

    p\_interest\_rate IN NUMBER,

    p\_years         IN NUMBER

) RETURN NUMBER

IS

    v\_monthly\_rate NUMBER;

    v\_months       NUMBER;

    v\_installment  NUMBER;

BEGIN

    v\_monthly\_rate := p\_interest\_rate / (12 \* 100);

    v\_months := p\_years \* 12;

    v\_installment := p\_loan\_amount \* v\_monthly\_rate / (1 - POWER(1 + v\_monthly\_rate, -v\_months));

    RETURN v\_installment;

END;

/

SELECT CalculateMonthlyInstallment(7000, 8, 3) AS Installment FROM dual;

**Output:**



**Scenario 3:**

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 AccountID = p\_account\_id;

    IF v\_balance >= p\_amount THEN

        RETURN TRUE;

    ELSE

        RETURN FALSE;

    END IF;

END;

/

DECLARE

    v\_result BOOLEAN;

BEGIN

    v\_result := HasSufficientBalance(1, 500);

    IF v\_result THEN

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

    ELSE

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

    END IF;

END;

/

**Output:**

A computer screen shot of a code

AI-generated content may be incorrect.

**Exercise 5: Triggers:**

**Scenario 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

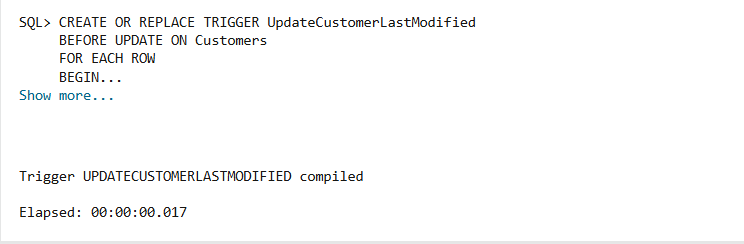
BEGIN

   :NEW.LastModified := SYSDATE;

END;

/

**Output:**



**Scenario 2:**

CREATE TABLE AuditLog (

    AuditID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    TransactionID NUMBER,

    AccountID NUMBER,

    Amount NUMBER,

    TransactionType VARCHAR2(10),

    TransactionDate DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

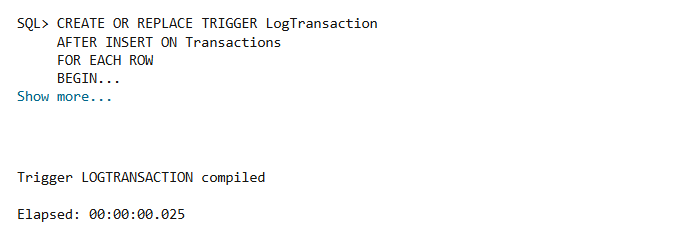
   INSERT INTO AuditLog (TransactionID,AccountID,Amount,TransactionType,TransactionDate)

   VALUES (:NEW.TransactionID,:NEW.AccountID,:NEW.Amount,:NEW.TransactionType,:NEW.TransactionDate);

END;

/

**Output:**



**Scenario 3:**

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

   FOR UPDATE;

   IF :NEW.TransactionType = 'Withdrawal' THEN

      IF :NEW.Amount > v\_balance THEN

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

      END IF;

   ELSIF :NEW.TransactionType = 'Deposit' THEN

      IF :NEW.Amount <= 0 THEN

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

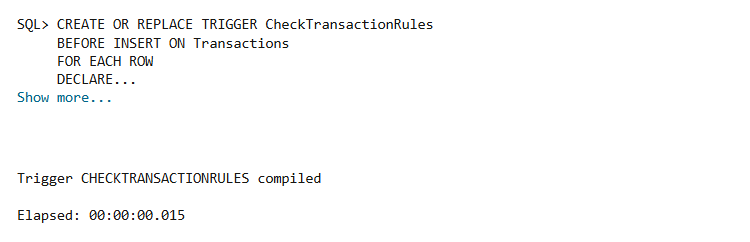
      END IF;

   END IF;

END;

/

**Output:**



**Exercise 6: Cursors:**

**Scenario 1:**

DECLARE

   CURSOR c\_monthly\_txns IS

      SELECT t.TransactionID,t.AccountID,t.TransactionDate,t.Amount,t.TransactionType,a.CustomerID

      FROM Transactions t

      JOIN Accounts a ON t.AccountID = a.AccountID

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

BEGIN

   FOR rec IN c\_monthly\_txns LOOP

      DBMS\_OUTPUT.PUT\_LINE(

         'CustomerID: ' || rec.CustomerID ||

         ', AccountID: ' || rec.AccountID ||

         ', TransactionID: ' || rec.TransactionID ||

         ', Date: ' || TO\_CHAR(rec.TransactionDate, 'DD-MON-YYYY') ||

         ', Type: ' || rec.TransactionType ||

         ', Amount: ' || rec.Amount

      );

   END LOOP;

END;

/

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 2:**

DECLARE

   CURSOR c\_accounts IS

      SELECT AccountID, Balance

      FROM Accounts;

BEGIN

   FOR rec IN c\_accounts LOOP

      UPDATE Accounts

      SET Balance = Balance - 50

      WHERE AccountID = rec.AccountID;

      DBMS\_OUTPUT.PUT\_LINE('Annual fee deducted from AccountID: ' || rec.AccountID ||'. New Balance: ' || (rec.Balance - 50));

   END LOOP;

   COMMIT;

END;

/

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 3:**

DECLARE

   CURSOR c\_loans IS

      SELECT LoanID, InterestRate

      FROM Loans;

   v\_new\_rate NUMBER;

BEGIN

   FOR rec IN c\_loans LOOP

      IF rec.InterestRate < 5 THEN

         v\_new\_rate := rec.InterestRate + 1;

      ELSE

         v\_new\_rate := rec.InterestRate - 0.5;

      END IF;

      UPDATE Loans

      SET InterestRate = v\_new\_rate

      WHERE LoanID = rec.LoanID;

      DBMS\_OUTPUT.PUT\_LINE('LoanID: ' || rec.LoanID || ', Old Rate: ' || rec.InterestRate || ', New Rate: ' || v\_new\_rate );

   END LOOP;

   COMMIT;

END;

/

**Output:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Exercise 7: Packages:**

**Scenario 1:**

CREATE OR REPLACE PACKAGE CustomerManagement IS

   PROCEDURE AddCustomer(p\_customer\_id NUMBER, p\_name VARCHAR2,p\_dob DATE, p\_balance  NUMBER);

   PROCEDURE UpdateCustomer(p\_customer\_id NUMBER, p\_name VARCHAR2,p\_balance NUMBER );

   FUNCTION GetBalance(p\_customer\_id NUMBER ) RETURN NUMBER;

END CustomerManagement;

/

BEGIN

   CustomerManagement.AddCustomer(10, 'New Customer', TO\_DATE('1973-11-21','YYYY-MM-DD'), 2000);

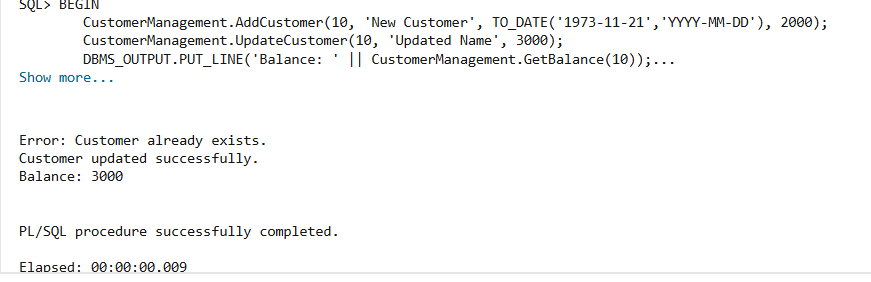
   CustomerManagement.UpdateCustomer(10, 'Updated Name', 3000);

   DBMS\_OUTPUT.PUT\_LINE('Balance: ' || CustomerManagement.GetBalance(10));

END;

/

**Output:**



**Scenario 2:**

CREATE OR REPLACE PACKAGE EmployeeManagement IS

   PROCEDURE HireEmployee(p\_employee\_id NUMBER,p\_name VARCHAR2,p\_position VARCHAR2,p\_salary NUMBER,p\_department VARCHAR2,p\_hiredate DATE);

   PROCEDURE UpdateEmployee(p\_employee\_id NUMBER,p\_salary NUMBER,p\_department VARCHAR2);

   FUNCTION AnnualSalary(

       p\_employee\_id NUMBER

   ) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

   PROCEDURE HireEmployee(

       p\_employee\_id NUMBER,

       p\_name        VARCHAR2,

       p\_position    VARCHAR2,

       p\_salary      NUMBER,

       p\_department  VARCHAR2,

       p\_hiredate    DATE

   ) IS

   BEGIN

       INSERT INTO Employees (

           EmployeeID, Name, Position, Salary, Department, HireDate

       )

       VALUES (

           p\_employee\_id, p\_name, p\_position, p\_salary, p\_department, p\_hiredate

       );

       COMMIT;

       DBMS\_OUTPUT.PUT\_LINE('Employee hired successfully.');

   EXCEPTION

       WHEN DUP\_VAL\_ON\_INDEX THEN

           DBMS\_OUTPUT.PUT\_LINE('Error: Employee already exists.');

   END HireEmployee;

   PROCEDURE UpdateEmployee(

       p\_employee\_id NUMBER,

       p\_salary      NUMBER,

       p\_department  VARCHAR2

   ) IS

   BEGIN

       UPDATE Employees

       SET Salary = p\_salary,

           Department = p\_department

       WHERE EmployeeID = p\_employee\_id;

       IF SQL%ROWCOUNT = 0 THEN

           DBMS\_OUTPUT.PUT\_LINE('Employee not found.');

       ELSE

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

       END IF;

       COMMIT;

   END UpdateEmployee;

   FUNCTION AnnualSalary(

       p\_employee\_id NUMBER

   ) RETURN NUMBER IS

       v\_salary NUMBER;

   BEGIN

       SELECT Salary INTO v\_salary

       FROM Employees

       WHERE EmployeeID = p\_employee\_id;

       RETURN v\_salary \* 12;

   EXCEPTION

       WHEN NO\_DATA\_FOUND THEN

           RETURN NULL;

   END AnnualSalary;

END EmployeeManagement;

/

BEGIN

   EmployeeManagement.HireEmployee(10, 'Sam Smith', 'Analyst', 5000, 'Finance', SYSDATE);

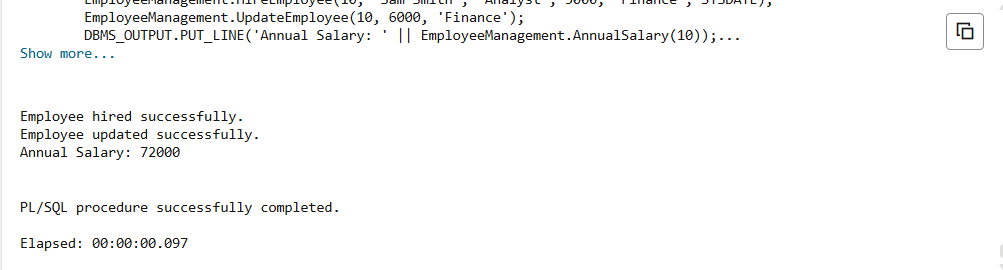
   EmployeeManagement.UpdateEmployee(10, 6000, 'Finance');

   DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || EmployeeManagement.AnnualSalary(10));

END;

/

**Output:**



**Scenario 3:**

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

   PROCEDURE OpenAccount(

       p\_account\_id   NUMBER,

       p\_customer\_id  NUMBER,

       p\_type         VARCHAR2,

       p\_balance      NUMBER

   ) IS

   BEGIN

       INSERT INTO Accounts (

           AccountID, CustomerID, AccountType, Balance, LastModified

       )

       VALUES (

           p\_account\_id, p\_customer\_id, p\_type, p\_balance, SYSDATE

       );

       COMMIT;

       DBMS\_OUTPUT.PUT\_LINE('Account opened successfully.');

   EXCEPTION

       WHEN DUP\_VAL\_ON\_INDEX THEN

           DBMS\_OUTPUT.PUT\_LINE('Error: Account already exists.');

   END OpenAccount;

   PROCEDURE CloseAccount(

       p\_account\_id NUMBER

   ) IS

   BEGIN

       DELETE FROM Accounts

       WHERE AccountID = p\_account\_id;

       IF SQL%ROWCOUNT = 0 THEN

           DBMS\_OUTPUT.PUT\_LINE('Account not found.');

       ELSE

           DBMS\_OUTPUT.PUT\_LINE('Account closed successfully.');

       END IF;

       COMMIT;

   END CloseAccount;

   FUNCTION GetTotalBalance(

       p\_customer\_id 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 GetTotalBalance;

END AccountOperations;

/

BEGIN

   AccountOperations.OpenAccount(20, 1, 'Savings', 1000);

   AccountOperations.CloseAccount(20);

   DBMS\_OUTPUT.PUT\_LINE('Total Balance: ' || AccountOperations.GetTotalBalance(1));

END;

/

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.