**WEEK 2**

**Module 3 - PL/SQL Programming**

**Exercise 1: Control Structures**

**Inserted Values:**

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

VALUES (1, 'John Doe', TO\_DATE('1950-05-15', 'YYYY-MM-DD'), 12000, SYSDATE);

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

VALUES (2, 'Emily Brown', TO\_DATE('1990-11-30', 'YYYY-MM-DD'), 15000, SYSDATE);

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

VALUES (3, 'Mark Taylor', TO\_DATE('1952-03-12', 'YYYY-MM-DD'), 8000, SYSDATE);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 10000, 6.5, SYSDATE - 60, SYSDATE + 10);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 3, 8000, 6.0, SYSDATE - 100, SYSDATE + 20);

**Scenario 1:**

SET SERVEROUTPUT ON;

BEGIN

FOR cust IN (

SELECT c.CustomerID, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE MONTHS\_BETWEEN(SYSDATE, c.DOB)/12 > 60

) LOOP

UPDATE Loans

SET InterestRate = cust.InterestRate - (cust.InterestRate \* 0.01)

WHERE LoanID = cust.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || cust.CustomerID ||

' | New Interest Rate: ' || TO\_CHAR(cust.InterestRate - (cust.InterestRate \* 0.01), '0.00'));

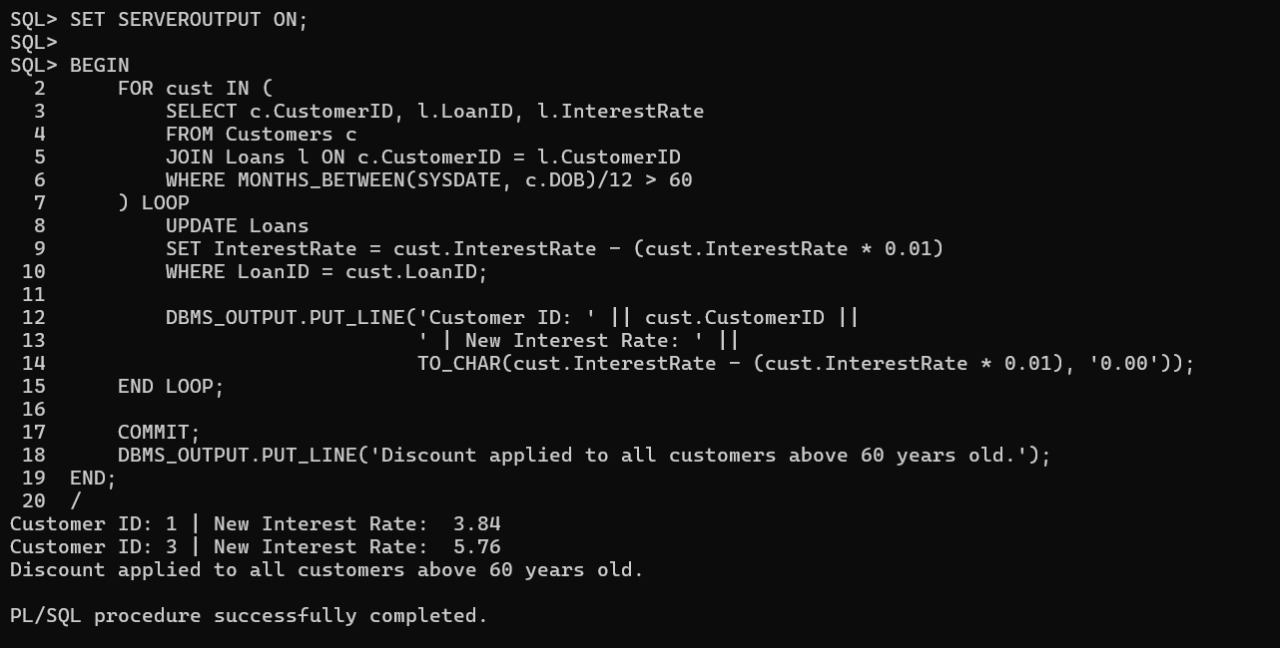
END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to all customers above 60 years old.');

END;

/

**OUTPUT:**

**Scenario 2:**

SET SERVEROUTPUT ON;

BEGIN

FOR cust IN (

SELECT CustomerID, Name FROM Customers WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || cust.CustomerID ||' | Name: ' || cust.Name || ' has been promoted to VIP.');

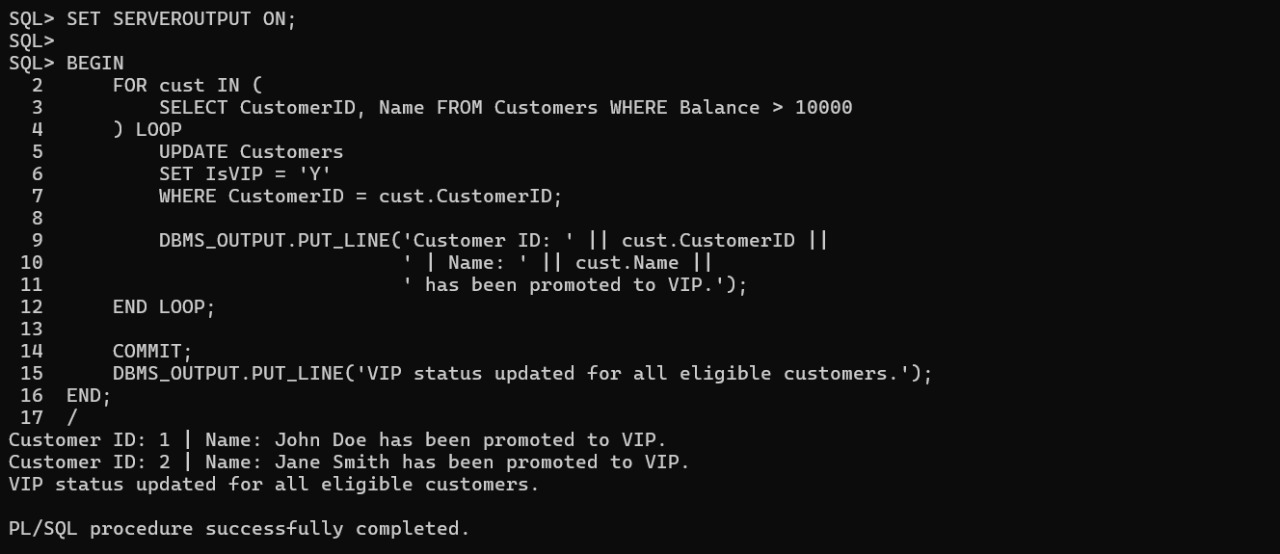
END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('VIP status updated for all eligible customers.');

END;

/

**OUTPUT:**

**Scenario 3:**

SET SERVEROUTPUT ON;

BEGIN

FOR dueLoan IN (

SELECT c.CustomerID, c.Name, l.LoanID, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

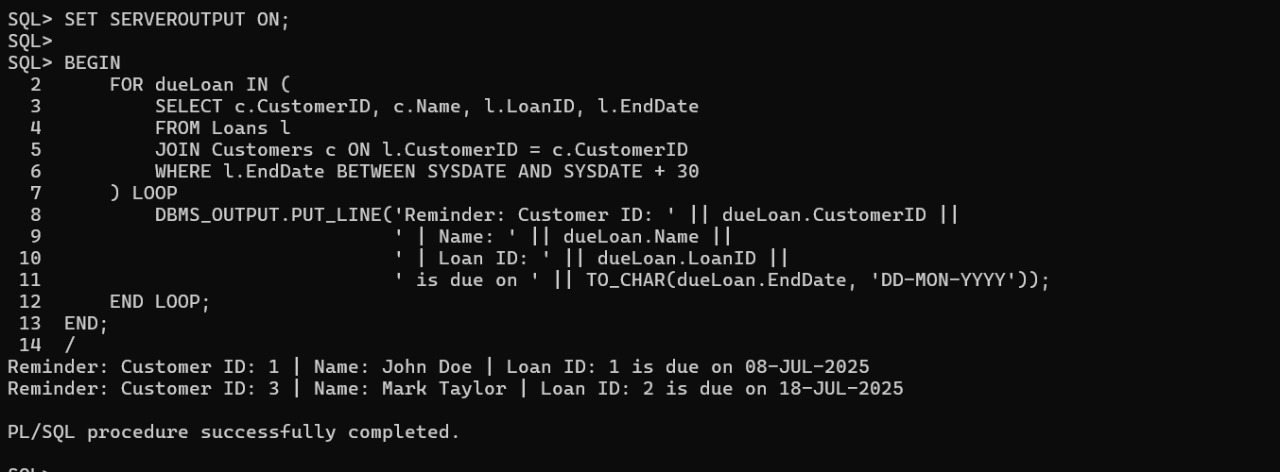
) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ID: ' || dueLoan.CustomerID ||' | Name: '|| dueLoan.Name ||' | Loan ID: ' || dueLoan.LoanID ||' is due on ' || TO\_CHAR(dueLoan.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**OUTPUT:**

**Exercise 2: Error Handling**

**Inserted Values:**  INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1980-01-01', 'YYYY-MM-DD'), 10000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1985-02-15', 'YYYY-MM-DD'), 8000, SYSDATE); INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (101, 1, 'Savings', 5000, SYSDATE);

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

VALUES (102, 2, 'Savings', 3000, SYSDATE); INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (201, 'Alice Johnson', 'Analyst', 60000, 'Finance', TO\_DATE('2015-06-01', 'YYYY-MM-DD'));

**Scenario 1:**

SET SERVEROUTPUT ON

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

from\_acc IN NUMBER,

to\_acc IN NUMBER,

amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = from\_acc;

IF v\_balance < amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in account ' || from\_acc);

END IF;

UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = from\_acc;

UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = to\_acc;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer of $' || amount || ' from Account ' || from\_acc || ' to Account ' || to\_acc || ' completed.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

BEGIN

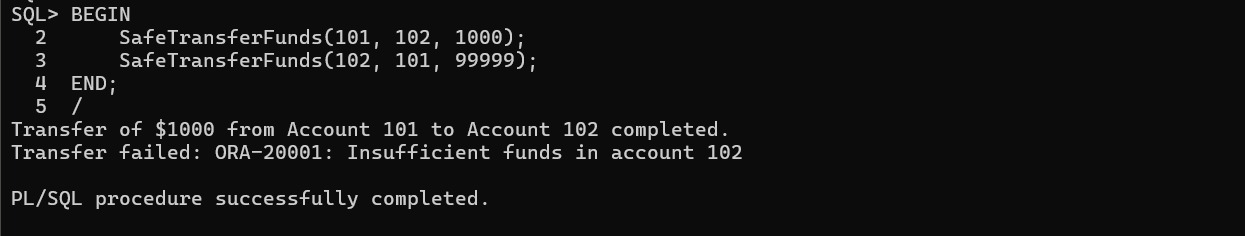
SafeTransferFunds(101, 102, 1000);

SafeTransferFunds(102, 101, 99999);

END;

/

**OUTPUT:**



**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

emp\_id IN NUMBER,

percent IN NUMBER) IS

v\_salary NUMBER;

BEGIN

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

UPDATE Employees

SET Salary = v\_salary + (v\_salary \* percent / 100)

WHERE EmployeeID = emp\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated for Employee ID ' || emp\_id || '. New salary: $' || TO\_CHAR(v\_salary + (v\_salary \* percent / 100), '99999'));

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

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

END;

/

BEGIN

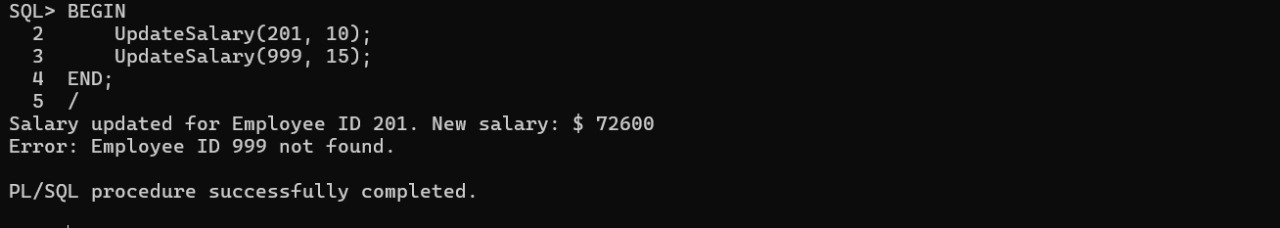
UpdateSalary(201, 10);

UpdateSalary(999, 15);

END;

/

**OUTPUT:**



**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

cust\_id IN NUMBER,

cust\_name IN VARCHAR2,

dob IN DATE,

balance IN NUMBER

) IS

BEGIN

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

VALUES (cust\_id, cust\_name, dob, balance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('New customer added: ID ' || cust\_id || ', Name: ' || cust\_name);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END;

/

BEGIN

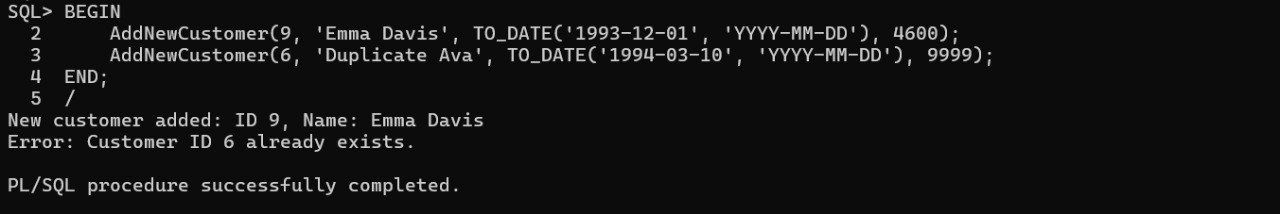
AddNewCustomer(9, 'Emma Davis', TO\_DATE('1993-12-01', 'YYYY-MM-DD'), 4600);

AddNewCustomer(6, 'Duplicate Ava', TO\_DATE('1994-03-10', 'YYYY-MM-DD'), 9999);

END;

/

**OUTPUT:**



**Exercise 3: Stored Procedures**

**Inserted Values:** INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (101, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (102, 2, 'Savings', 500, SYSDATE);

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (201, 'Alice Johnson', 'Manager', 60000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD')); INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (102, 2, 'Savings', 3000, SYSDATE); INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (201, 'Alice Johnson', 'Analyst', 60000, 'Finance', TO\_DATE('2015-06-01', 'YYYY-MM-DD'));

**Scenario 1:**

SET SERVEROUTPUT ON

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType ='Savings')LOOP

UPDATE Accounts

SET Balance = acc.Balance + (acc.Balance \* 0.01)

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || acc.AccountID || ' | New Balance: $' || TO\_CHAR(acc.Balance + (acc.Balance \* 0.01), '9999.99'));

END LOOP;

COMMIT;

END;

/

BEGIN

ProcessMonthlyInterest;

END;

/

**OUTPUT:**



**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept IN VARCHAR2,

bonus\_pct IN NUMBER

) IS

BEGIN

FOR emp IN (SELECT EmployeeID, Salary FROM Employees WHERE Department = dept) LOOP

UPDATE Employees

SET Salary = emp.Salary + (emp.Salary \* bonus\_pct / 100)

WHERE EmployeeID = emp.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || emp.EmployeeID || ' | Bonus Applied: ' || bonus\_pct || '% | New Salary: $' || TO\_CHAR(emp.Salary + (emp.Salary \* bonus\_pct / 100), '99999.99'));

END LOOP;

COMMIT;

END;

/

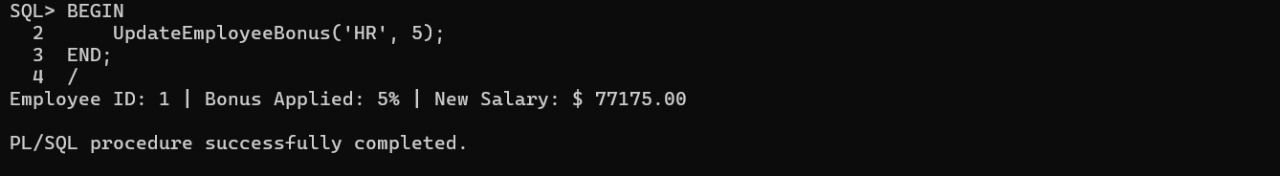
BEGIN

UpdateEmployeeBonus('HR', 5);

END;

/

**OUTPUT:**



**Scenario 3:**

SET SERVEROUTPUT ON

DELETE FROM Accounts WHERE AccountID IN (101, 102);

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

VALUES (101, 1, 'Savings', 1000, SYSDATE);

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

VALUES (102, 2, 'Savings', 100, SYSDATE);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_acc IN NUMBER,

to\_acc IN NUMBER,

amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = from\_acc;

IF v\_balance < amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in account ' || from\_acc);

END IF;

UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = from\_acc;

UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = to\_acc;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transferred $' || amount || ' from Account ' || from\_acc || ' to Account ' || to\_acc);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

BEGIN

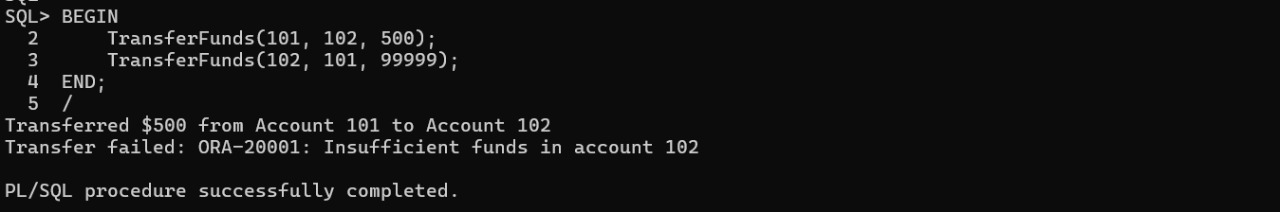
TransferFunds(101, 102, 500);

TransferFunds(102, 101, 99999);

END;

/

**OUTPUT:**



**Exercise 4: Functions**

**Inserted Values:** INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (101, 1, 'Savings', 1000, SYSDATE);

COMMIT;

**Scenario 1:**

SET SERVEROUTPUT ON

CREATE OR REPLACE FUNCTION CalculateAge(dob DATE) RETURN NUMBER IS

age NUMBER;

BEGIN

age := TRUNC(MONTHS\_BETWEEN(SYSDATE, dob) / 12);

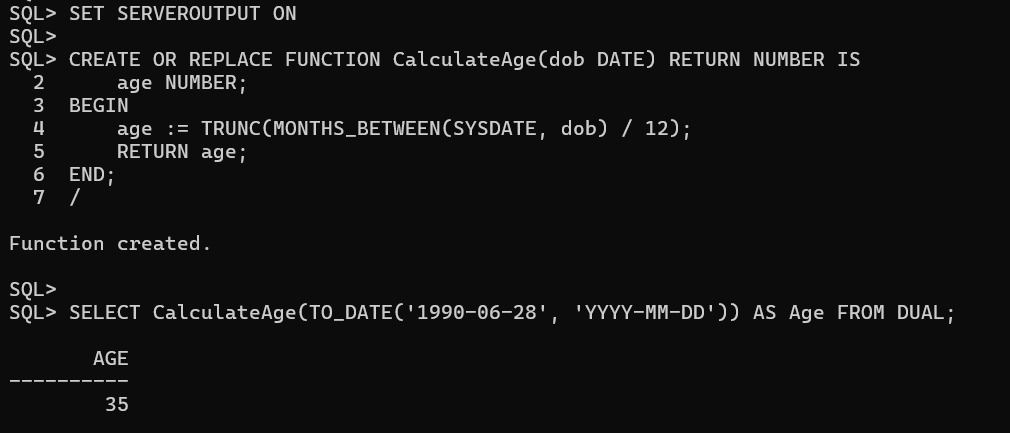
RETURN age;

END;

/

SELECT CalculateAge(TO\_DATE('1990-06-28', 'YYYY-MM-DD')) AS Age FROM DUAL;

**OUTPUT:**



**Scenario 2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

loan\_amount NUMBER,

annual\_rate NUMBER,

duration\_years NUMBER

) RETURN NUMBER IS

r NUMBER;

n NUMBER;

emi NUMBER;

BEGIN

r := annual\_rate / 12 / 100;

n := duration\_years \* 12;

emi := (loan\_amount \* r \* POWER(1 + r, n)) / (POWER(1 + r, n) - 1);

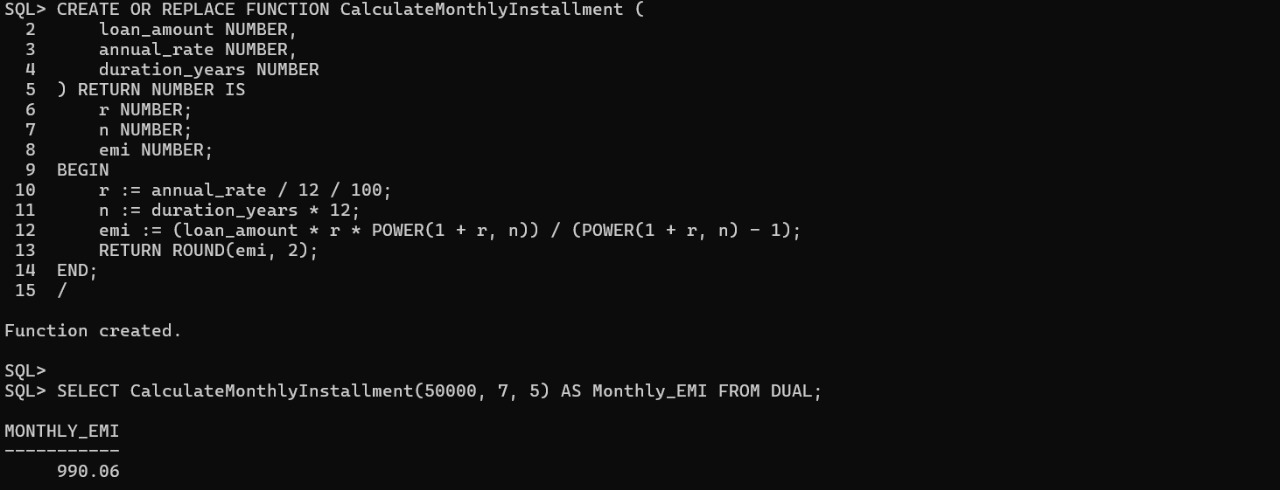
RETURN ROUND(emi, 2);

END;

/

SELECT CalculateMonthlyInstallment(50000, 7, 5) AS Monthly\_EMI FROM DUAL;

**OUTPUT:**



**Scenario 3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

acc\_id NUMBER,

amount NUMBER

) RETURN BOOLEAN IS

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Accounts WHERE AccountID = acc\_id;

IF bal >= amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

DECLARE

result BOOLEAN;

BEGIN

result := HasSufficientBalance(101, 500);

IF result THEN

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

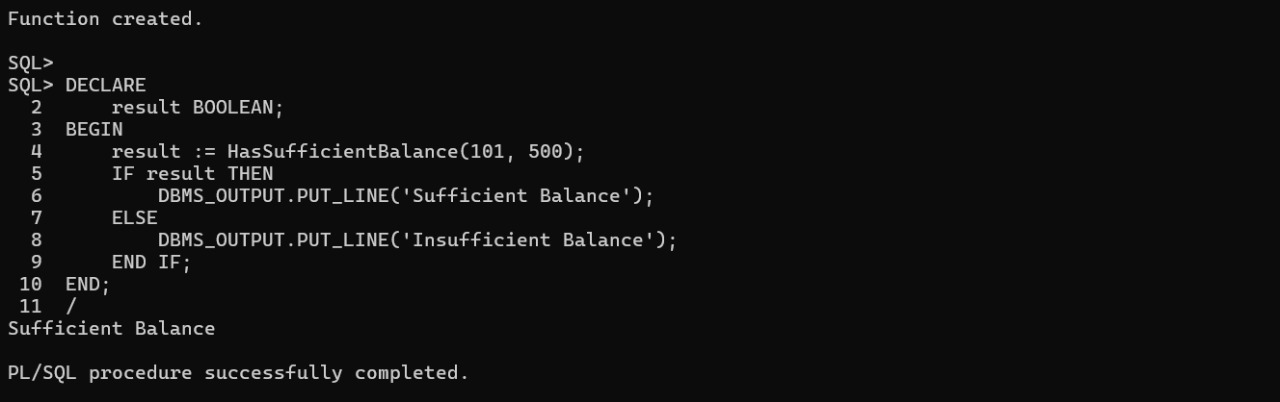
ELSE

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

END IF;

END;

/

**OUTPUT:**

**Exercise 5: Triggers**

**Scenario 1:**

SET SERVEROUTPUT ON

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

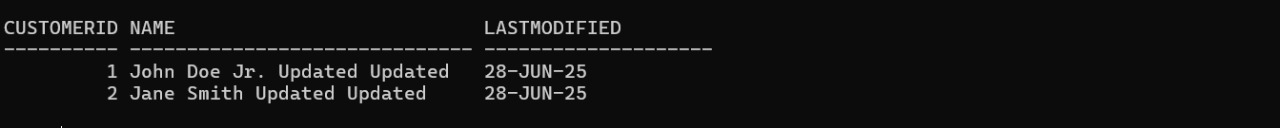
/

UPDATE Customers

SET Name = Name || ' Updated';

SELECT CustomerID, Name, LastModified FROM Customers;

**OUTPUT:**



**Scenario 2:**

CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

Action VARCHAR2(20),

LogDate DATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, Action, LogDate)

VALUES (:NEW.TransactionID, 'INSERT', SYSDATE);

END;

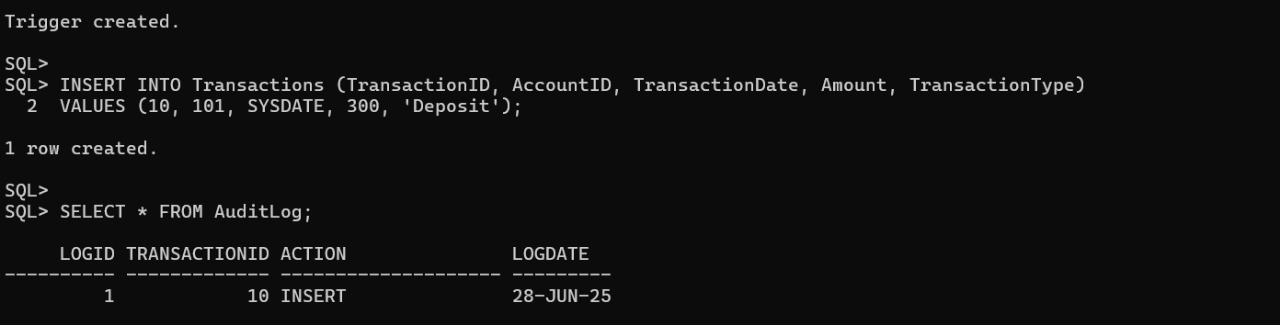
/

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

VALUES (10, 101, SYSDATE, 300, 'Deposit');

SELECT \* FROM AuditLog;

**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;

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

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

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

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

END IF;

END;

/

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

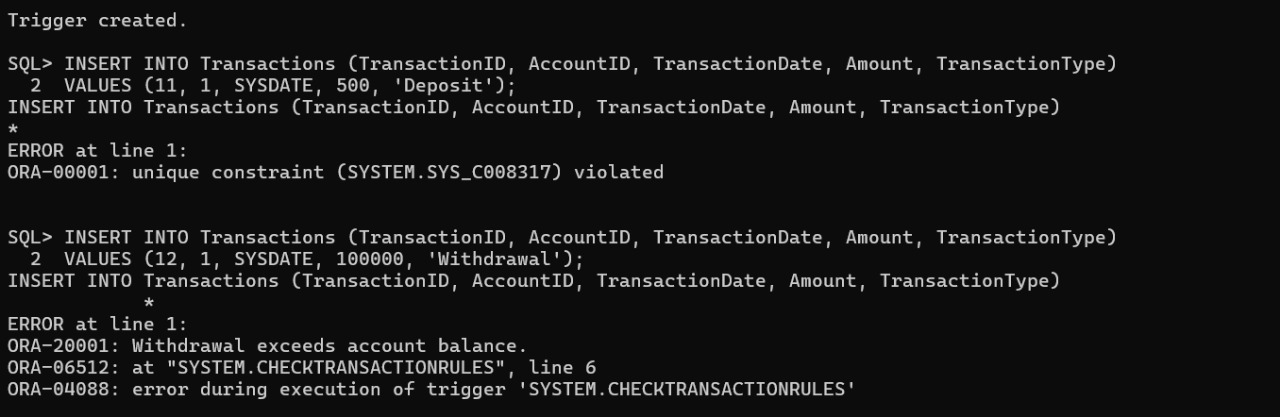
VALUES (11, 1, SYSDATE, 500, 'Deposit')

COMMIT

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

VALUES (12, 1, SYSDATE, 100000, 'Withdrawal')

**OUTPUT:**



**Exercise 6: Cursors**

**Inserted Values:**

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

VALUES (1, 'Alice Jones', TO\_DATE('1980-05-10', 'YYYY-MM-DD'), 3000, SYSDATE);

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

VALUES (2, 'Brian Scott', TO\_DATE('1990-09-22', 'YYYY-MM-DD'), 4500, SYSDATE); INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (201, 1, 'Savings', 2500, SYSDATE);

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

VALUES (202, 2, 'Savings', 4000, SYSDATE);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (401, 1, 10000, 6.5, SYSDATE, ADD\_MONTHS(SYSDATE, 24));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (402, 2, 15000, 7.2, SYSDATE, ADD\_MONTHS(SYSDATE, 36));

COMMIT;

**Scenario 1:**

SET SERVEROUTPUT ON

BEGIN

FOR rec IN (

SELECT c.CustomerID, c.Name, t.TransactionDate, t.Amount, t.TransactionType

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE TO\_CHAR(t.TransactionDate, 'MMYYYY') = TO\_CHAR(SYSDATE, 'MMYYYY')

)

LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || rec.CustomerID || ' | Name: ' || rec.Name ||

' | ' || rec.TransactionType || ' of $' || rec.Amount ||

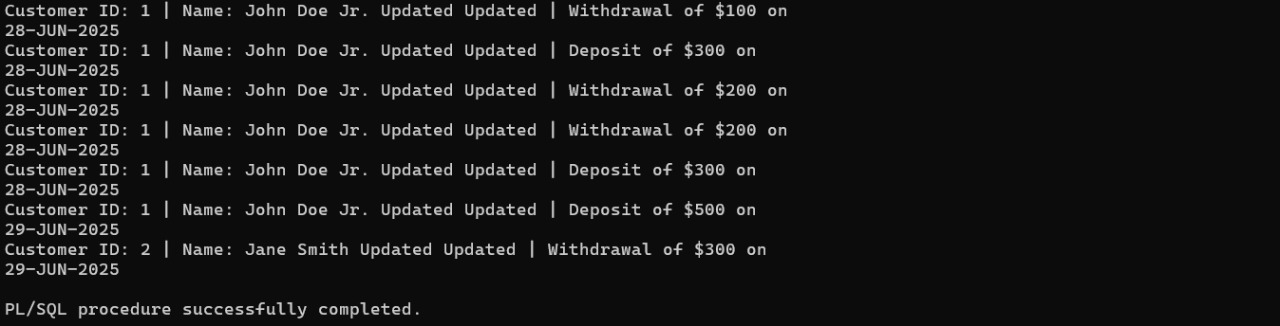
' on ' || TO\_CHAR(rec.TransactionDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**OUTPUT:**



**Scenario 2:**

BEGIN

FOR rec IN (SELECT AccountID, Balance FROM Accounts) LOOP

UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.AccountID || ' | Annual fee deducted. New Balance: $' || (rec.Balance - 100));

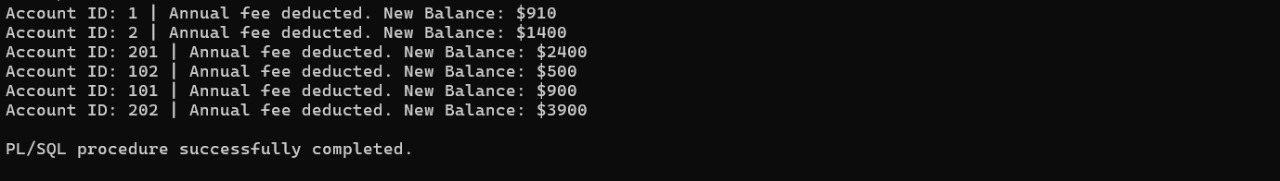
END LOOP;

  COMMIT;

END;

/

**OUTPUT:**



**Scenario 3:**

BEGIN

FOR rec IN (SELECT LoanID, InterestRate FROM Loans) LOOP

UPDATE Loans SET InterestRate = rec.InterestRate + 0.5 WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || rec.LoanID || ' | New Interest Rate: ' || TO\_CHAR(rec.InterestRate + 0.5, '0.00'));

END LOOP;

  COMMIT;

END;

/

**OUTPUT:**



**Exercise 7: Packages**

**Scenario 1:**

SET SERVEROUTPUT ON

CREATE OR REPLACE PACKAGE CustomerManagement AS

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

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

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

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

BEGIN

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

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

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

END;

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

BEGIN

UPDATE Customers SET Name = p\_name, Balance = p\_balance, LastModified = SYSDATE

WHERE CustomerID = p\_id;

DBMS\_OUTPUT.PUT\_LINE('Customer updated: ' || p\_name);

END;

FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

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

RETURN v\_balance;

END;

END;

/

BEGIN

CustomerManagement.AddCustomer(10, 'Lucas West', TO\_DATE('1995-10-05', 'YYYY-MM-DD'), 2000);

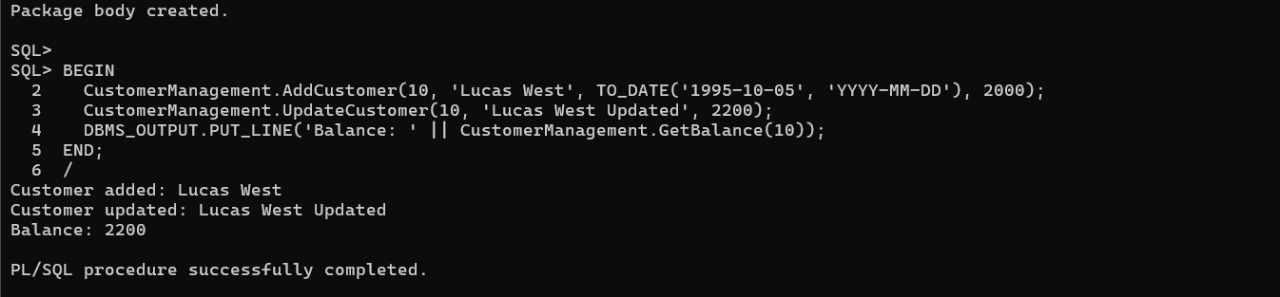
CustomerManagement.UpdateCustomer(10, 'Lucas West Updated', 2200);

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

END;

/

**OUTPUT:**



**Scenario 2:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2, p\_hire DATE);

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER);

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2, p\_hire DATE) IS

BEGIN

INSERT INTO Employees VALUES (p\_id, p\_name, p\_position, p\_salary, p\_dept, p\_hire);

DBMS\_OUTPUT.PUT\_LINE('Employee hired: ' || p\_name);

END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER) IS

BEGIN

UPDATE Employees SET Salary = p\_salary WHERE EmployeeID = p\_id;

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

END;

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

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

RETURN v\_salary \* 12;

END;

END;

/

BEGIN

EmployeeManagement.HireEmployee(10, 'Sophia Grey', 'Analyst', 5000, 'Finance', SYSDATE);

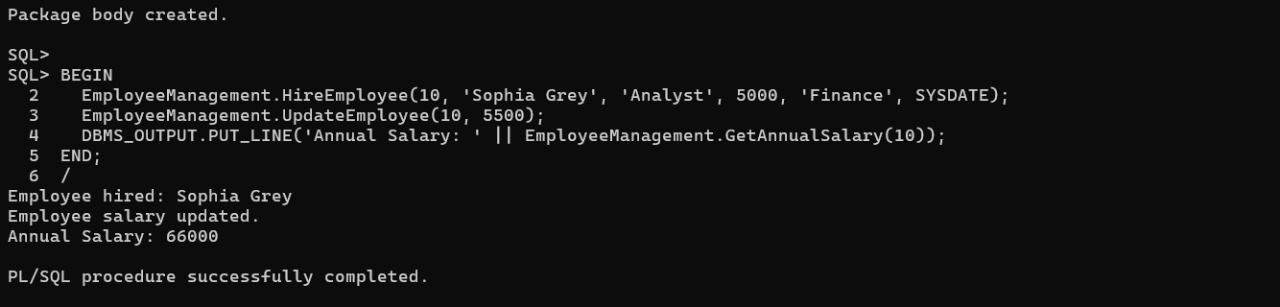
EmployeeManagement.UpdateEmployee(10, 5500);

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

END;

/

**OUTPUT:**



**Scenario 3:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_accid NUMBER);

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_accid NUMBER, p\_custid NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (p\_accid, p\_custid, p\_type, p\_balance, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Account opened: ' || p\_accid);

END;

PROCEDURE CloseAccount(p\_accid NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_accid;

DBMS\_OUTPUT.PUT\_LINE('Account closed: ' || p\_accid);

END;

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER IS

v\_total NUMBER;

BEGIN

SELECT NVL(SUM(Balance), 0) INTO v\_total FROM Accounts WHERE CustomerID = p\_custid;

RETURN v\_total;

END;

END;

/

BEGIN

AccountOperations.OpenAccount(301, 10, 'Savings', 1000);

DBMS\_OUTPUT.PUT\_LINE('Total Balance: ' || AccountOperations.GetTotalBalance(10));

AccountOperations.CloseAccount(301);

END;

/

**OUTPUT:**

