**PL/SQL Programming**

*CREATE TABLE Customers (*

*CustomerID NUMBER PRIMARY KEY,*

*Name VARCHAR2(100),*

*DOB DATE,*

*Balance NUMBER,*

*LastModified DATE*

*);*

*CREATE TABLE Accounts (*

*AccountID NUMBER PRIMARY KEY,*

*CustomerID NUMBER,*

*AccountType VARCHAR2(20),*

*Balance NUMBER,*

*LastModified DATE,*

*FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)*

*);*

*CREATE TABLE Transactions (*

*TransactionID NUMBER PRIMARY KEY,*

*AccountID NUMBER,*

*TransactionDate DATE,*

*Amount NUMBER,*

*TransactionType VARCHAR2(10),*

*FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID));*

*CREATE TABLE Loans (*

*LoanID NUMBER PRIMARY KEY,*

*CustomerID NUMBER,*

*LoanAmount NUMBER,*

*InterestRate NUMBER,*

*StartDate DATE,*

*EndDate DATE,*

*FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID));*

*CREATE TABLE Employees (*

*EmployeeID NUMBER PRIMARY KEY,*

*Name VARCHAR2(100),*

*Position VARCHAR2(50),*

*Salary NUMBER,*

*Department VARCHAR2(50),*

*HireDate DATE*

*);*

Example Scripts for Sample Data Insertion

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

*VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);*

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

*VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);*

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

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

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

*VALUES (2, 2, 'Checking', 1500, SYSDATE);*

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

*VALUES (1, 1, SYSDATE, 200, 'Deposit');*

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

*VALUES (2, 2, SYSDATE, 300, 'Withdrawal');*

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

*VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));*

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

*VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));*

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

*VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));*

**Exercise 1:**

**Scenario 1:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

IsVIP VARCHAR2(5) DEFAULT 'FALSE'

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

InterestRate NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Salary NUMBER

);

CREATE TABLE ErrorLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

ErrorMessage VARCHAR2(4000),

ErrorDate DATE

);

-- Customers

INSERT INTO Customers VALUES (1, 'Amit', 65, 15000, 'FALSE');

INSERT INTO Customers VALUES (2, 'Neha', 45, 8000, 'FALSE');

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

-- Loans

INSERT INTO Loans VALUES (101, 1, 10.5, SYSDATE + 20);

INSERT INTO Loans VALUES (102, 2, 11.0, SYSDATE + 40);

INSERT INTO Loans VALUES (103, 3, 12.0, SYSDATE + 10);

-- Accounts

INSERT INTO Accounts VALUES (201, 1, 10000);

INSERT INTO Accounts VALUES (202, 2, 5000);

-- Employees

INSERT INTO Employees VALUES (301, 'Suresh', 30000);

DELETE FROM Employees WHERE EmployeeID = 302;

INSERT INTO Employees VALUES (302, 'Meena', 45000);

COMMIT;

BEGIN

FOR rec IN (

SELECT CustomerID, Name, Age FROM Customers

WHERE Age > 60

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Eligible for discount: ' || rec.Name ||

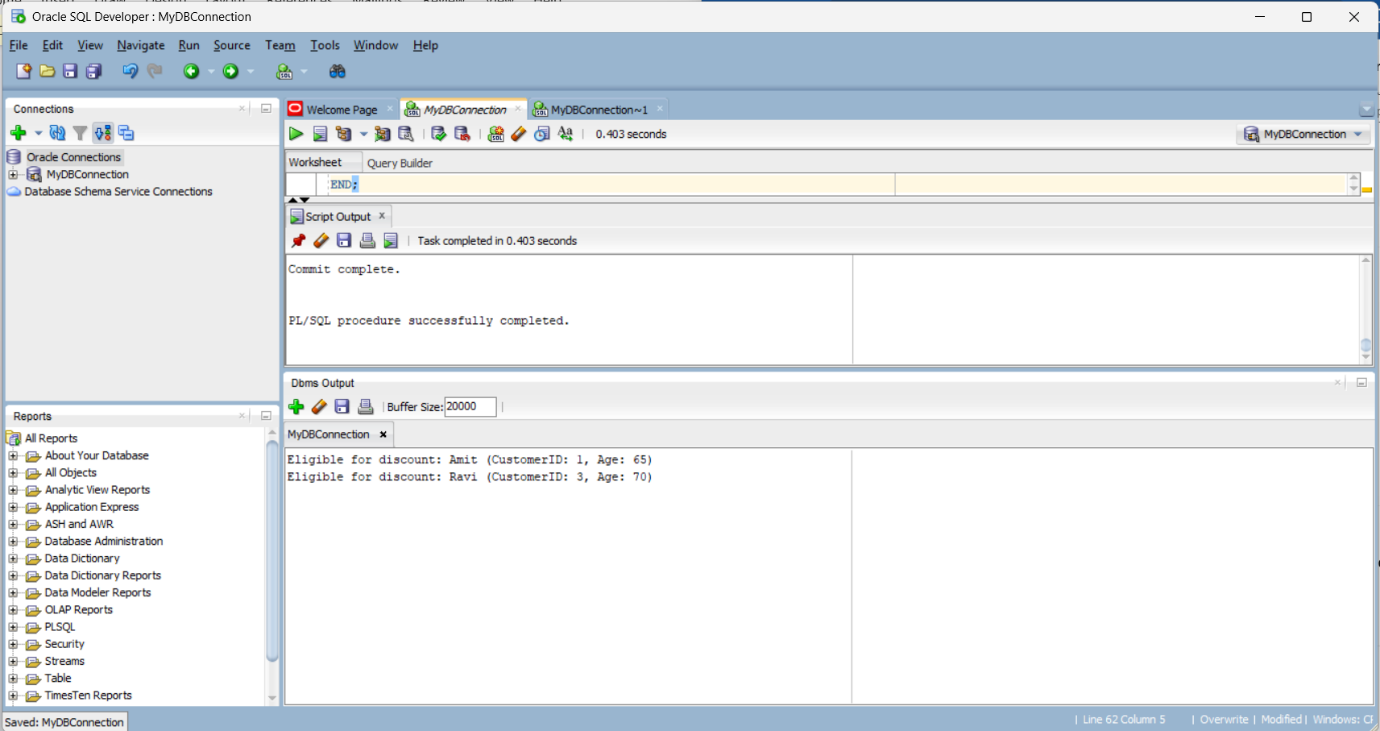
' (CustomerID: ' || rec.CustomerID ||

', Age: ' || rec.Age || ')');

END LOOP;

END;

**Output:**



**Scenario 2:**

BEGIN

FOR rec IN (

SELECT CustomerID, Name, Balance

FROM Customers

WHERE Balance > 10000

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Eligible for VIP: ' || rec.Name ||

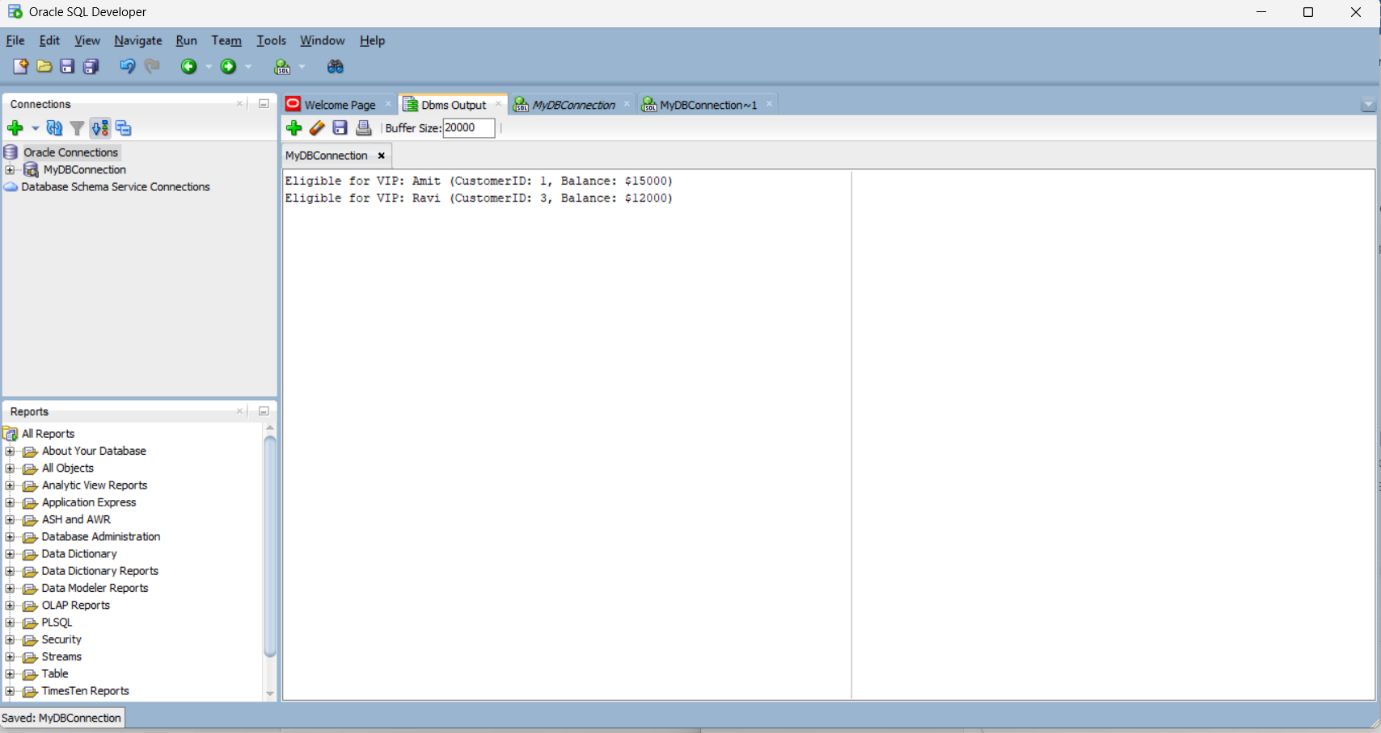
' (CustomerID: ' || rec.CustomerID ||

', Balance: $' || rec.Balance || ')');

END LOOP;

END;

**Output:**

****

**Scenario 3:**

BEGIN

FOR rec IN (

SELECT L.LoanID, L.CustomerID, C.Name, L.DueDate

FROM Loans L

JOIN Customers C ON L.CustomerID = C.CustomerID

WHERE L.DueDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID ||

' for Customer ' || rec.Name ||

' (CustomerID: ' || rec.CustomerID ||

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

END LOOP;

END;

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise 2:**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

v\_error\_msg VARCHAR2(4000);

BEGIN

-- Step 1: Check source balance

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id;

IF v\_balance < p\_amount THEN

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

END IF;

-- Step 2: Transfer

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 successful.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

v\_error\_msg := SUBSTR('Transfer Error: ' || SQLERRM, 1, 4000);

-- ✅ Safe column names

INSERT INTO ErrorLog(ERRORMESSAGE, ERRORDATE)

VALUES (v\_error\_msg,SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Error during transfer: '||v\_error\_msg);

END;

/

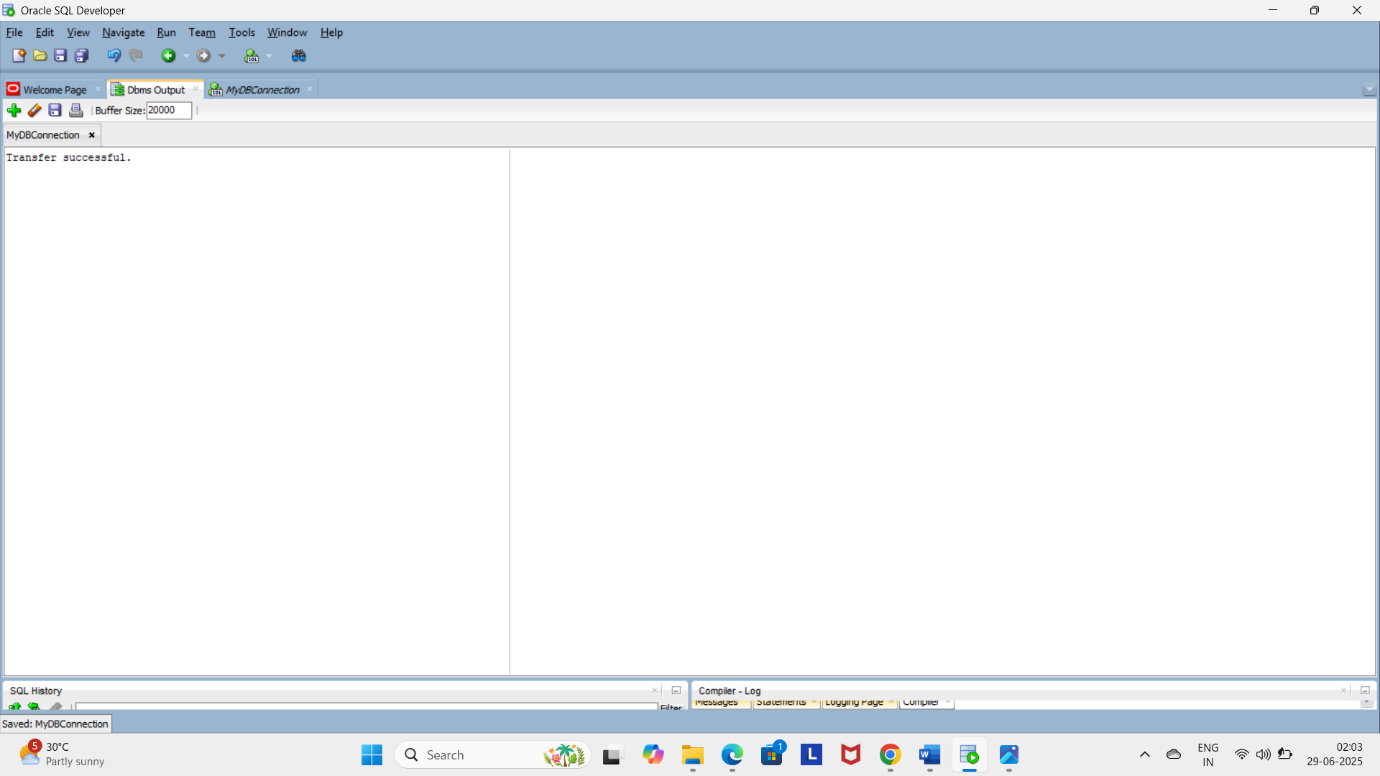
BEGIN

SafeTransferFunds(201, 202, 500);

END;

/

**Output:**

****

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

v\_error\_msg VARCHAR2(4000);

BEGIN

-- Update the employee's salary

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

-- Check if employee exists

IF SQL%ROWCOUNT = 0 THEN

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

END IF;

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

v\_error\_msg := SUBSTR('Salary Update Error: ' || SQLERRM, 1, 4000);

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES (v\_error\_msg, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Error during salary update: ' || v\_error\_msg);

END;

/

SET SERVEROUTPUT ON;

-- Successful case

BEGIN

UpdateSalary(301, 10); -- Increase salary of Suresh by 10%

END;

/

-- Failure case (ID not found)

BEGIN

UpdateSalary(201);

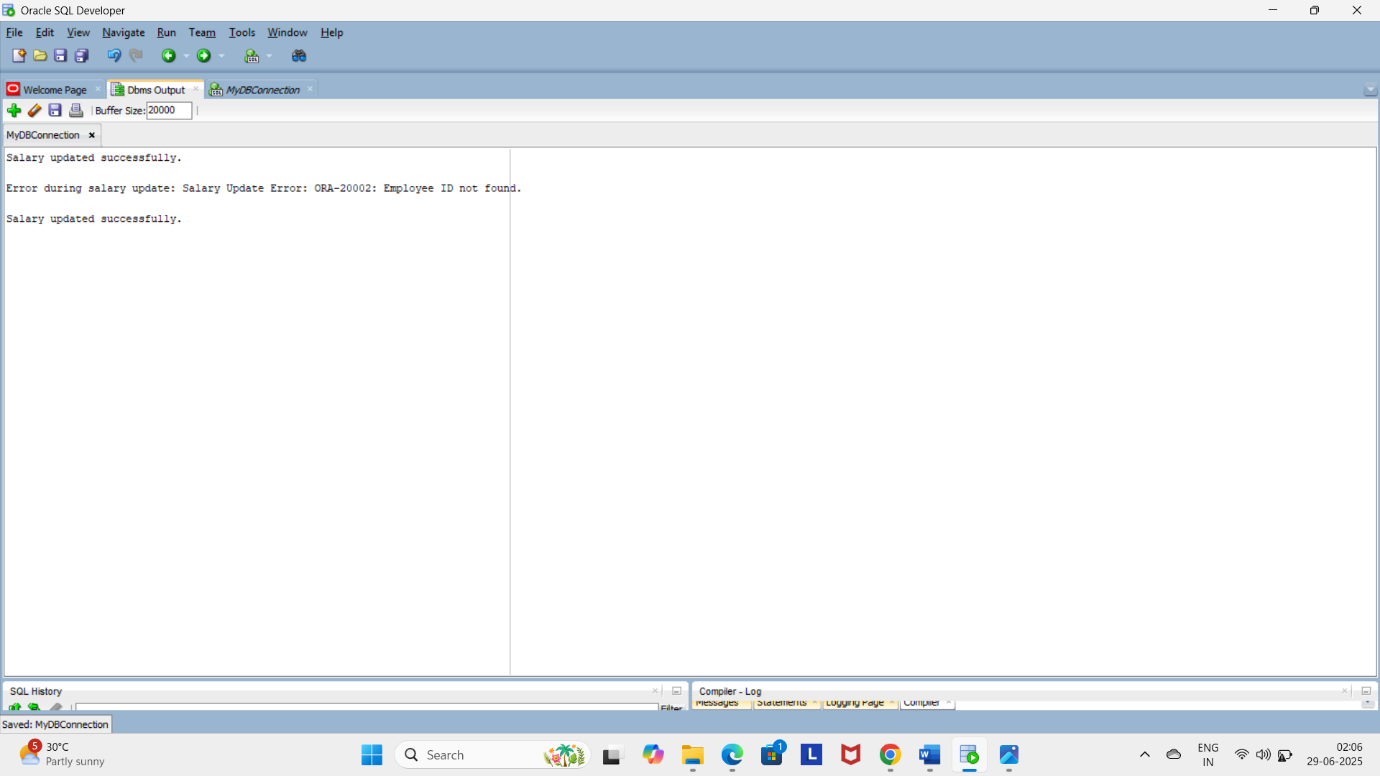
END;

/

-- Check the log

SELECT \* FROM ErrorLog ORDER BY LogID DESC;

**Output:**

****

**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_age IN NUMBER,

p\_balance IN NUMBER

) AS

v\_error\_msg VARCHAR2(4000);

BEGIN

INSERT INTO Customers (CustomerID, Name, Age, Balance)

VALUES (p\_customer\_id, p\_name, p\_age, p\_balance);

COMMIT;

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

v\_error\_msg := 'Duplicate Customer ID: ' || TO\_CHAR(p\_customer\_id);

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES (v\_error\_msg, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Error: ' || v\_error\_msg);

WHEN OTHERS THEN

ROLLBACK;

v\_error\_msg := SUBSTR('Add Customer Error: ' || SQLERRM, 1, 4000);

INSERT INTO ErrorLog (ErrorMessage, ErrorDate)

VALUES (v\_error\_msg, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Error during customer addition: ' || v\_error\_msg);

END;

/

SET SERVEROUTPUT ON;

-- Successful case

BEGIN

AddNewCustomer(4, 'Pooja', 30, 6000);

END;

/

-- Duplicate ID case (already exists)

BEGIN

AddNewCustomer(1, 'Amit', 65, 15000);

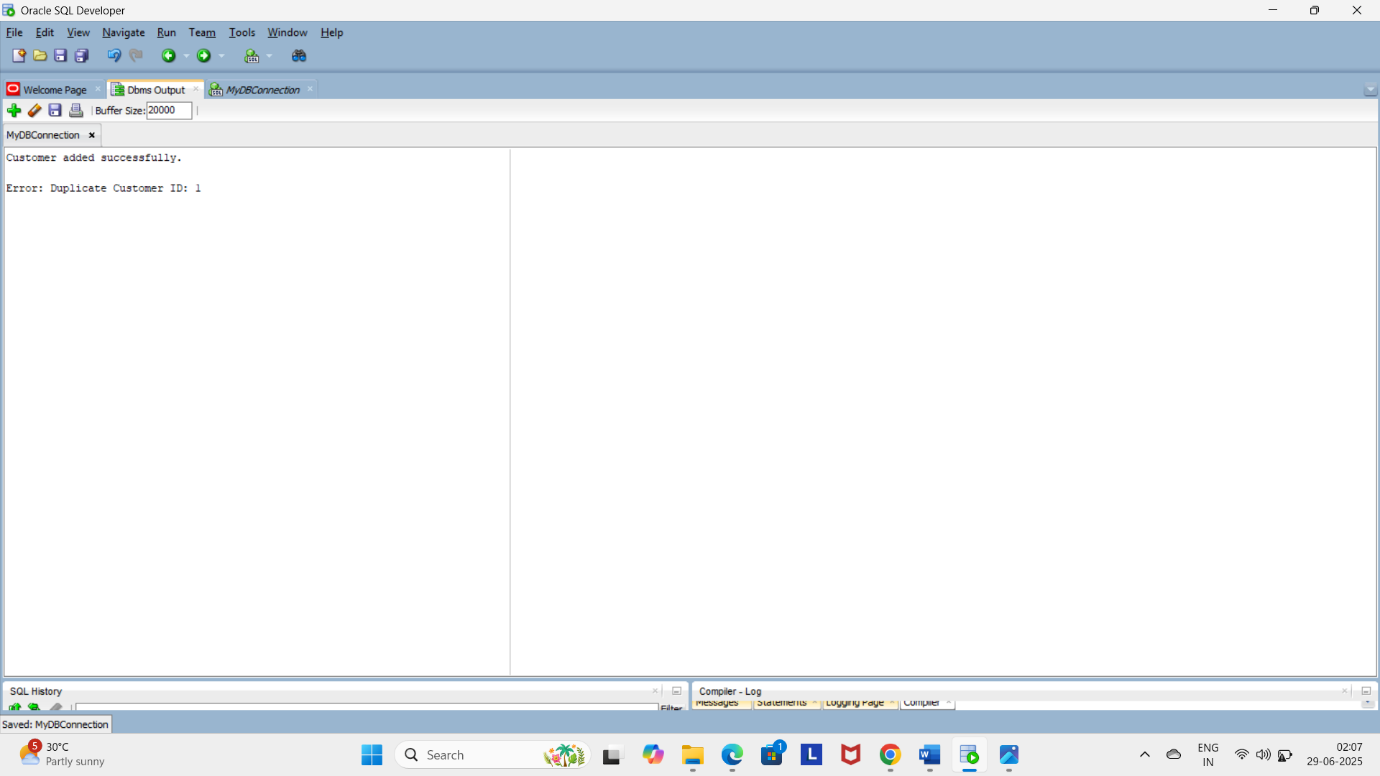
END;

/

-- Check error log

SELECT \* FROM ErrorLog ORDER BY LogID DESC;

**Output:**

****

**Exercise 3:**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

v\_err\_msg VARCHAR2(4000);

BEGIN

-- Apply 1% interest to all accounts

UPDATE Accounts

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

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

v\_err\_msg := SUBSTR('Interest Process Error: ' || SQLERRM, 1, 4000);

-- Safely insert into log

INSERT INTO ErrorLog("ERRORMESSAGE", "ERRORDATE")

VALUES (v\_err\_msg, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Error during interest processing: ' || v\_err\_msg);

END;

/

BEGIN

ProcessMonthlyInterest;

END;

**A screenshot of a computer

AI-generated content may be incorrect.**/ Output:

**Scenario 3:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_pct IN NUMBER

) AS

v\_error\_msg VARCHAR2(4000);

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'No employees found in department ' || p\_department);

END IF;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to department: ' || p\_department);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

v\_error\_msg := SUBSTR('Bonus Update Error: ' || SQLERRM, 1, 4000);

INSERT INTO ErrorLog(ErrorMessage, ErrorDate)

VALUES (v\_error\_msg, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE(v\_error\_msg);

END;

/

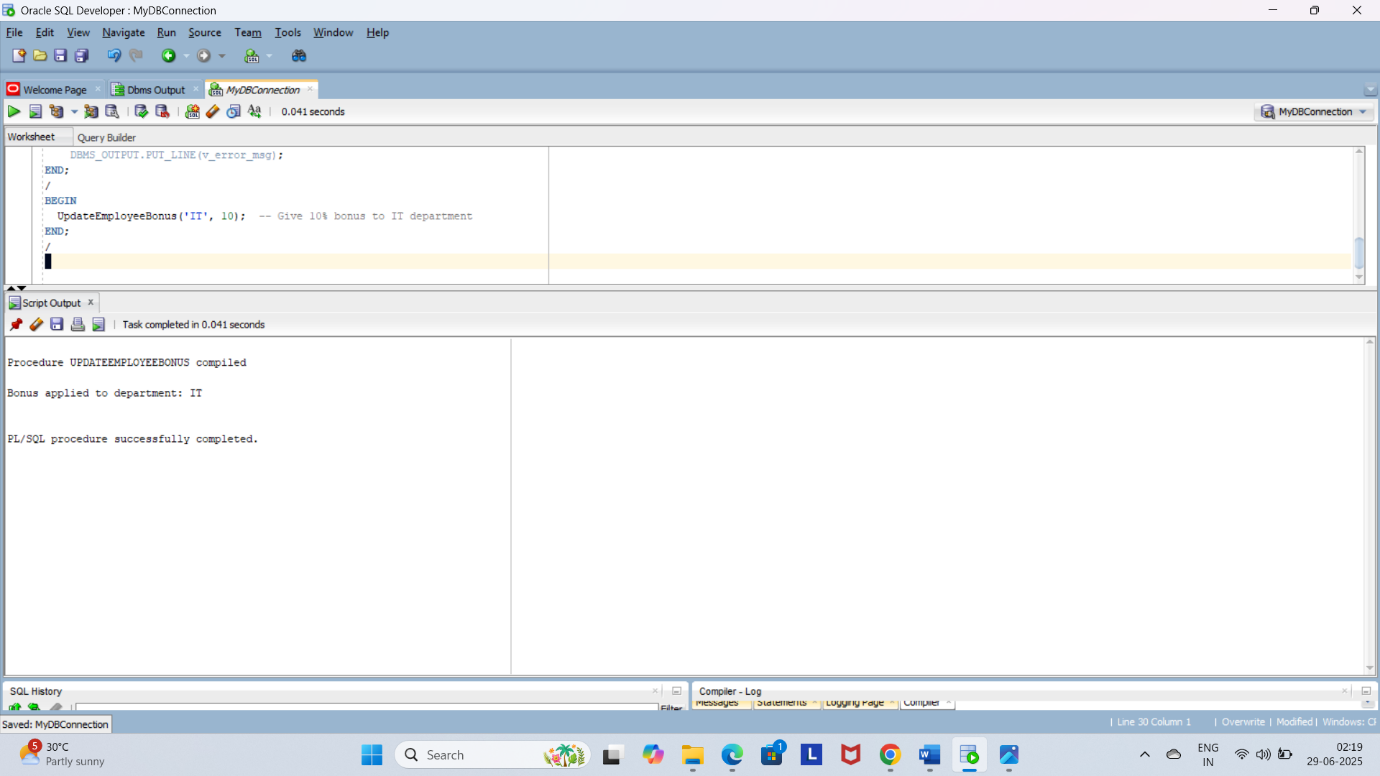
BEGIN

UpdateEmployeeBonus('IT', 10); -- Give 10% bonus to IT department

END;

/

**Output:**

****

**Exercise 4:**

**Scenario 1:**

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

RETURN NUMBER

IS

v\_age NUMBER;

BEGIN

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

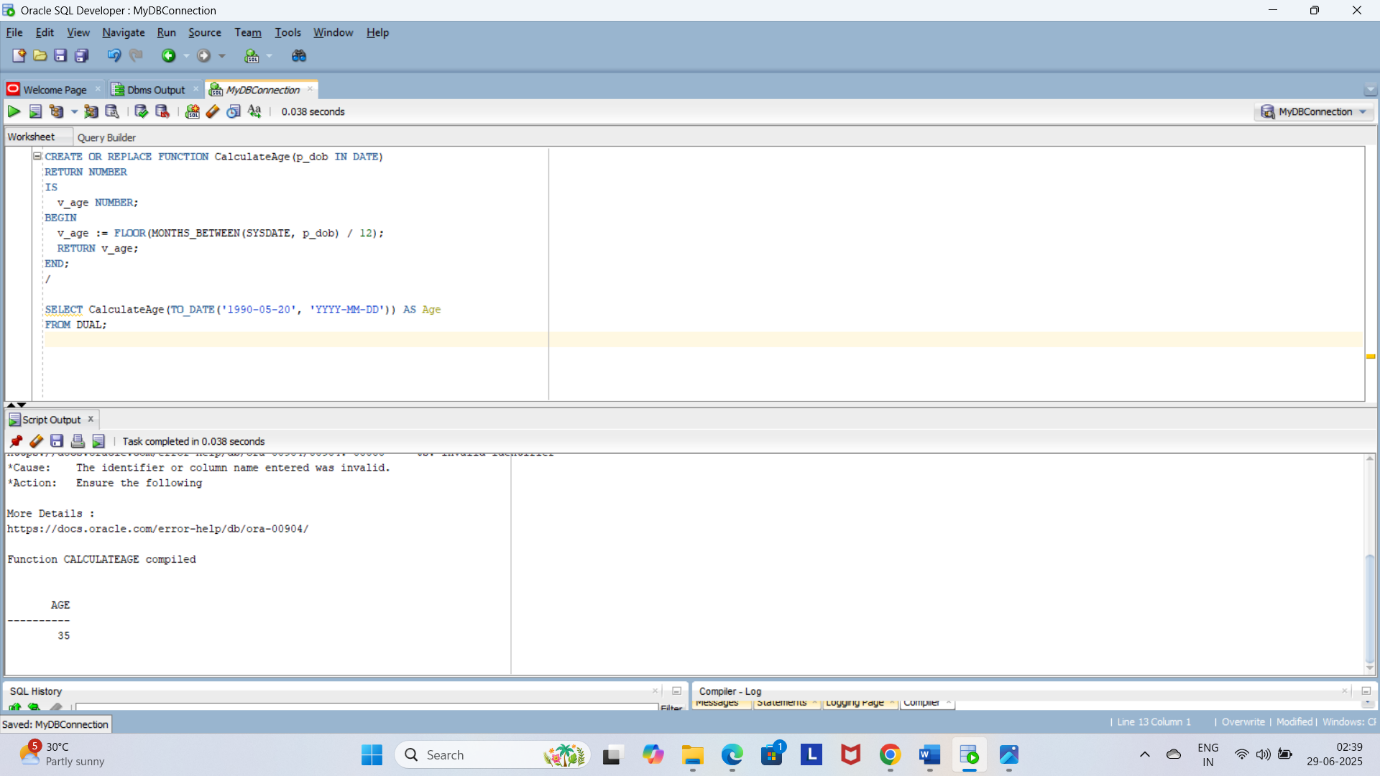
RETURN v\_age;

END;

/

SELECT CalculateAge(TO\_DATE('1990-05-20', 'YYYY-MM-DD')) AS Age

FROM DUAL;



**Scenario 2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loan\_amount IN NUMBER,

p\_annual\_interest\_rate IN NUMBER,

p\_loan\_years IN NUMBER

) RETURN NUMBER

IS

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

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

v\_emi NUMBER;

BEGIN

IF v\_monthly\_rate = 0 THEN

v\_emi := p\_loan\_amount / v\_months;

ELSE

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

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

END IF;

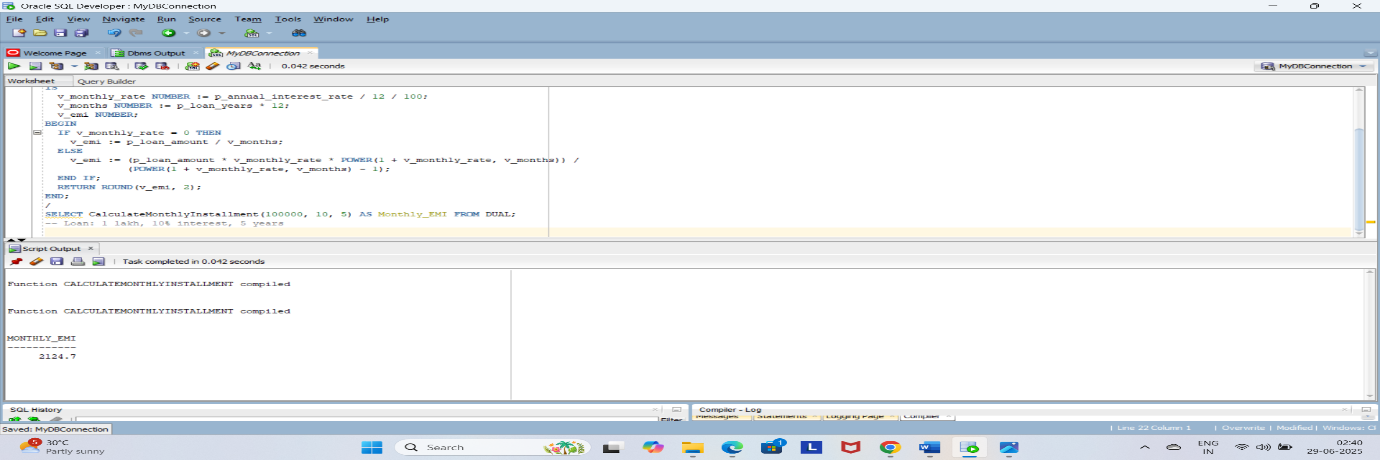
RETURN ROUND(v\_emi, 2);

END;

/

SELECT CalculateMonthlyInstallment(100000, 10, 5) AS Monthly\_EMI FROM DUAL;

-- Loan: 1 lakh, 10% interest, 5 years

**Output:**

**Scenario 3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_account\_id IN NUMBER,

p\_amount IN NUMBER

) RETURN NUMBER

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

IF v\_balance >= p\_amount THEN

RETURN 1;

ELSE

RETURN 0;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

RETURN 0;

END;

/

SELECT HasSufficientBalance(201, 5000) AS IsEligible FROM DUAL;

**Output: A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise 5:**

**Scenario 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

SELECT CustomerID, Name, LastModified

FROM Customers

WHERE CustomerID = 1;

UPDATE Customers

SET Name = 'Amit Kumar'

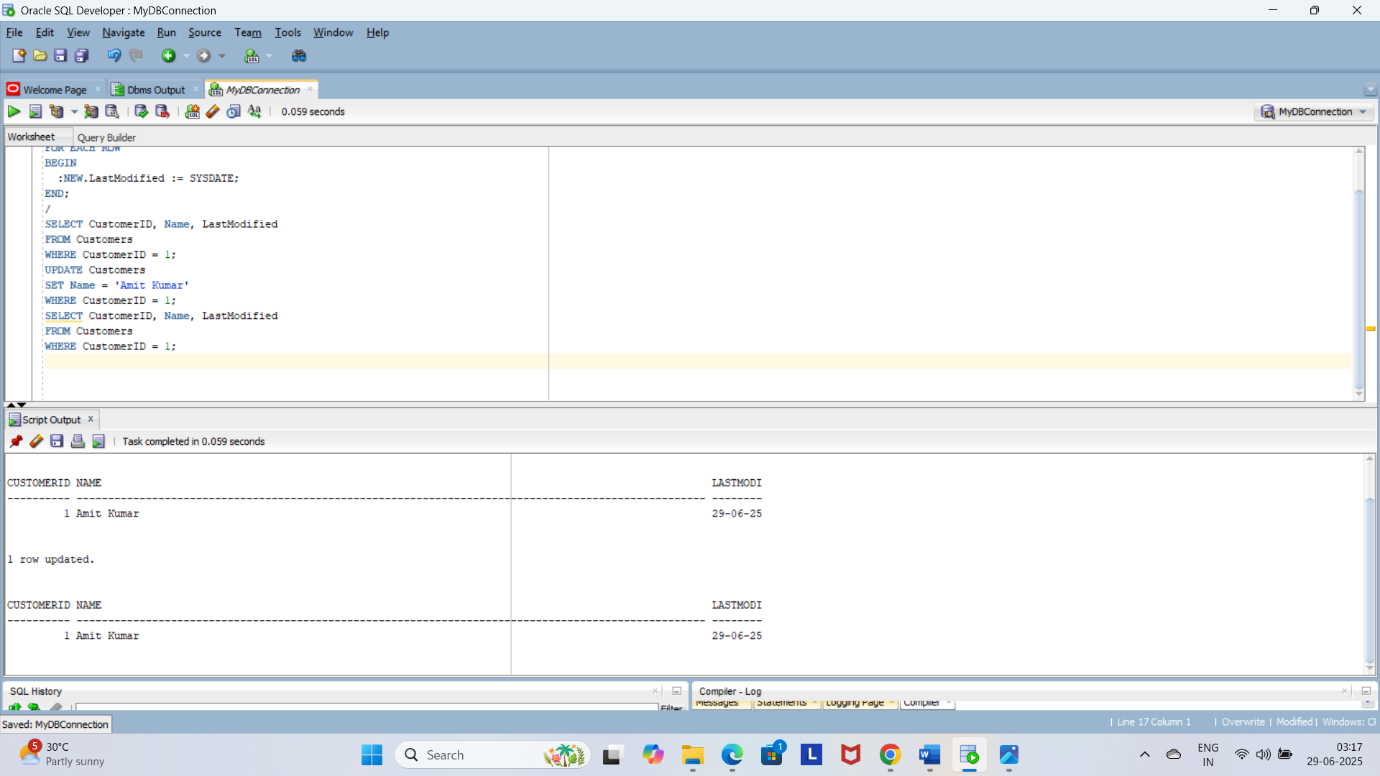
WHERE CustomerID = 1;

SELECT CustomerID, Name, LastModified

FROM Customers

WHERE CustomerID = 1;

**Output:**

****

**Scenario 2:**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

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

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

END;

/

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

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

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

END;

/

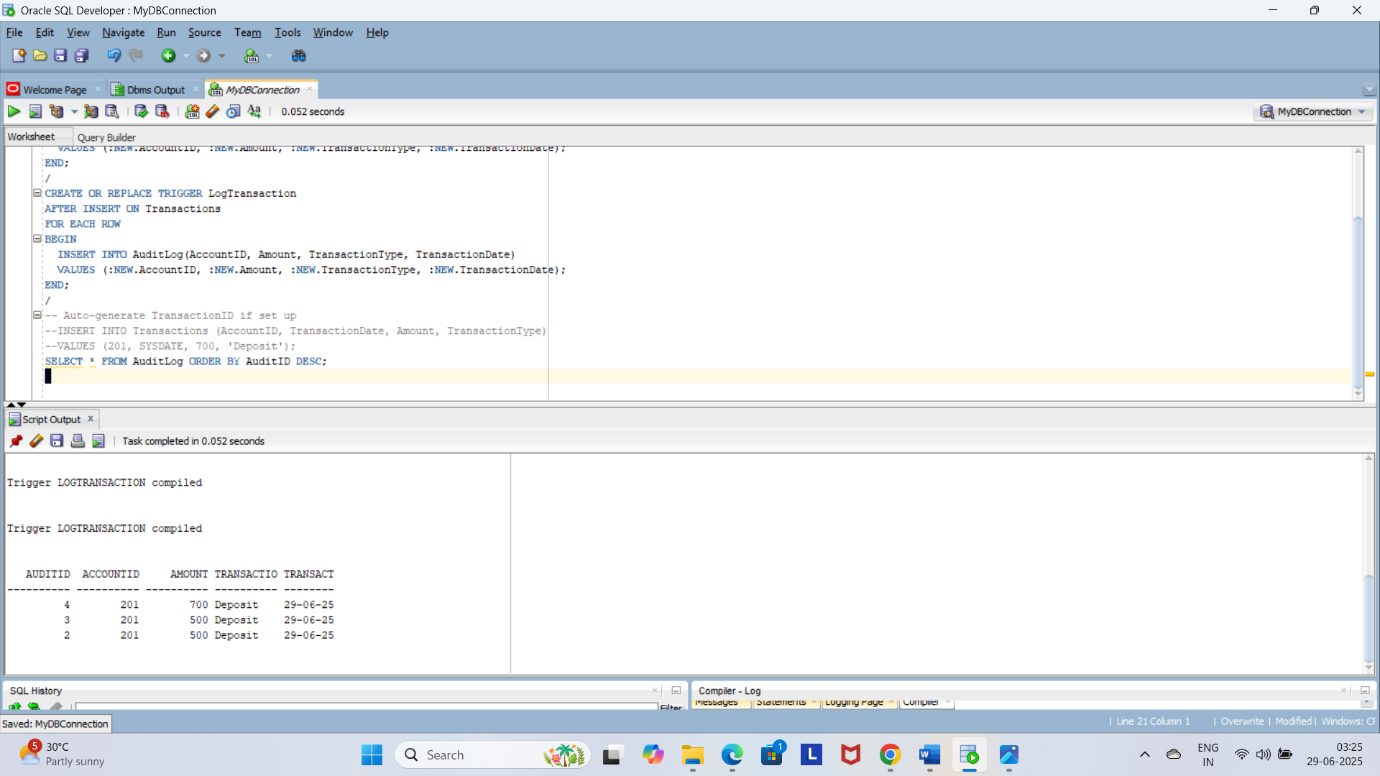
-- Auto-generate TransactionID if set up

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

--VALUES (201, SYSDATE, 700, 'Deposit');

SELECT \* FROM AuditLog ORDER BY AuditID DESC;

**Output:**

****

**Scenario 3:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

-- Get current balance of the account

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' THEN

IF :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance for withdrawal.');

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;

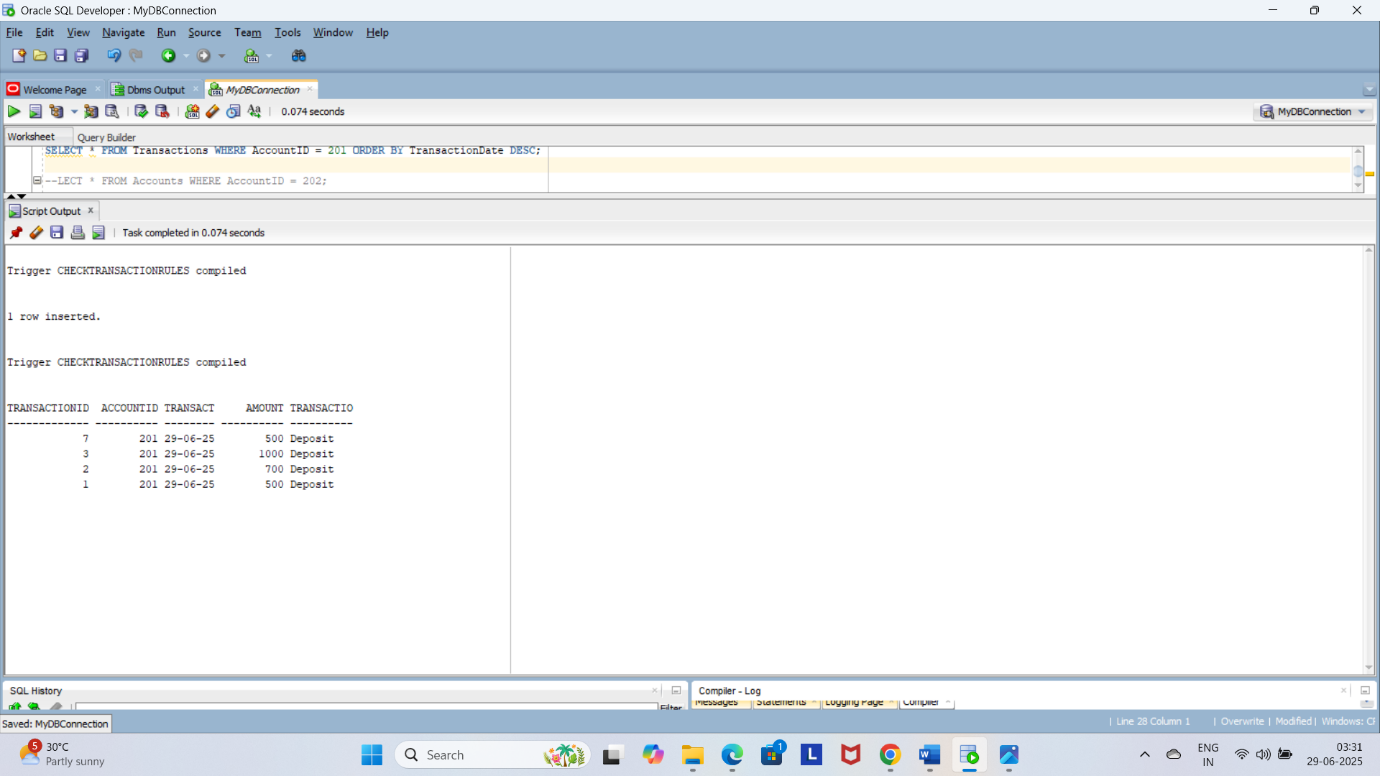
/

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

--VALUES (201,500,'Deposit', SYSDATE);

SELECT \* FROM Transactions WHERE AccountID = 201 ORDER BY TransactionDate DESC;

**Output:**

****

**Exercise 6:**

**Scenario 1:**

DECLARE

CURSOR txn\_cursor IS

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

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')

ORDER BY c.Name;

v\_rec txn\_cursor%ROWTYPE;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Monthly Customer Statements ---');

OPEN txn\_cursor;

LOOP

FETCH txn\_cursor INTO v\_rec;

EXIT WHEN txn\_cursor%NOTFOUND;

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

', Transaction ID: ' || v\_rec.TransactionID ||

', Type: ' || v\_rec.TransactionType ||

', Amount: ' || v\_rec.Amount ||

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

END LOOP;

CLOSE txn\_cursor;

END;

/

SET SERVEROUTPUT ON;

DECLARE

CURSOR txn\_cursor IS

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

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')

ORDER BY c.Name;

v\_rec txn\_cursor%ROWTYPE;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Monthly Customer Statements ---');

OPEN txn\_cursor;

LOOP

FETCH txn\_cursor INTO v\_rec;

EXIT WHEN txn\_cursor%NOTFOUND;

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

', Transaction ID: ' || v\_rec.TransactionID ||

', Type: ' || v\_rec.TransactionType ||

', Amount: ₹' || v\_rec.Amount ||

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

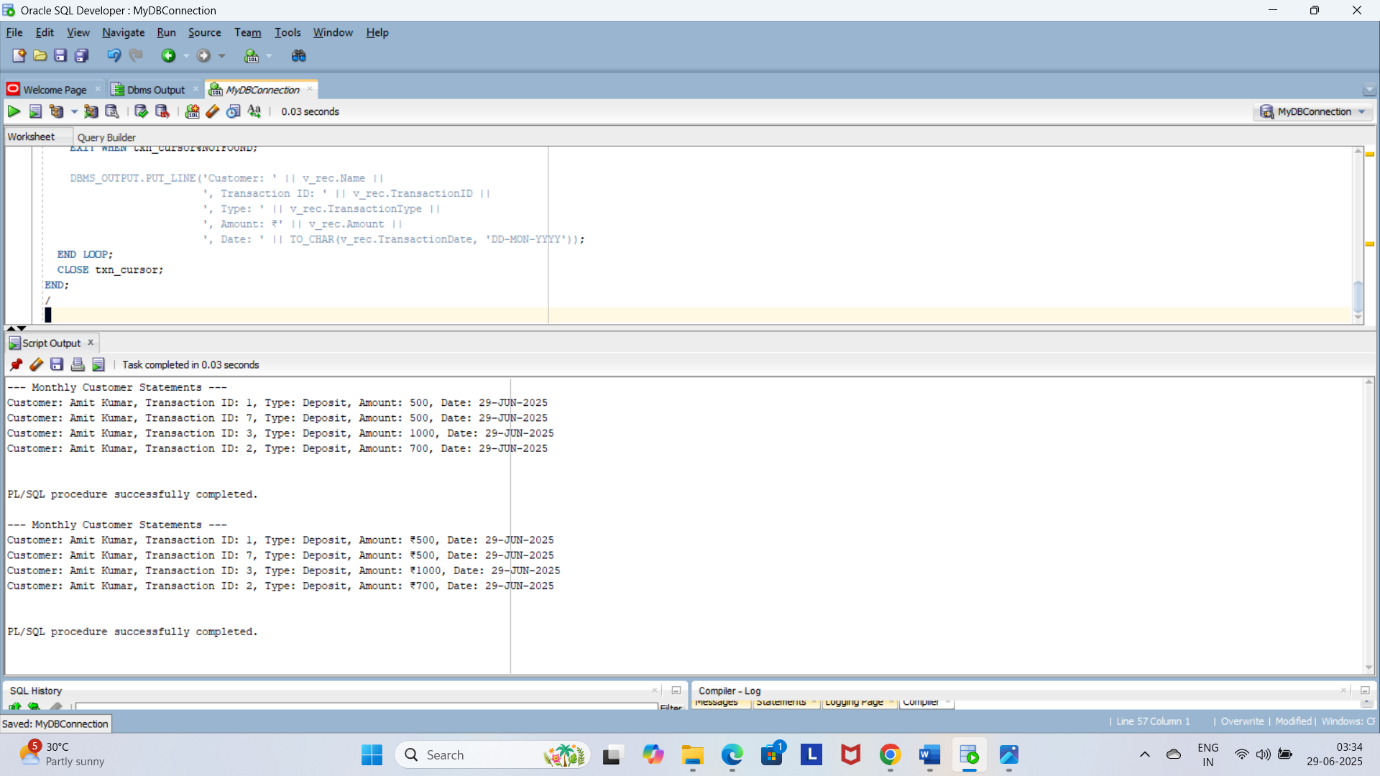
END LOOP;

CLOSE txn\_cursor;

END;

/

**Output:**

****

**Scenario 2:**

SET SERVEROUTPUT ON;

DECLARE

CURSOR acc\_cursor IS

SELECT AccountID, Balance FROM Accounts;

v\_acc acc\_cursor%ROWTYPE;

v\_fee NUMBER := 100;

BEGIN

OPEN acc\_cursor;

LOOP

FETCH acc\_cursor INTO v\_acc;

EXIT WHEN acc\_cursor%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - v\_fee

WHERE AccountID = v\_acc.AccountID;

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

END LOOP;

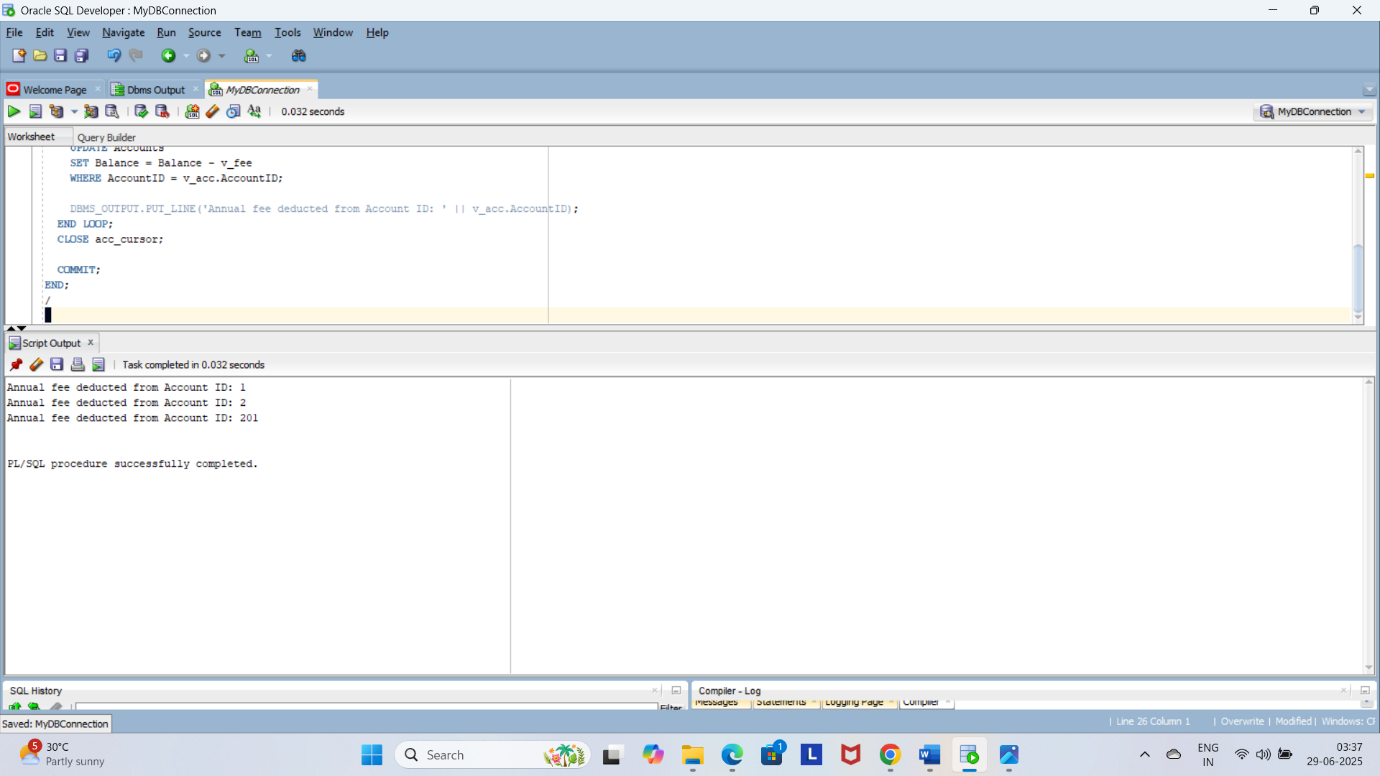
CLOSE acc\_cursor;

COMMIT;

END;

/

**Output:**

****

**Scenario 3:**

SET SERVEROUTPUT ON;

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate FROM Loans;

v\_loan loan\_cursor%ROWTYPE;

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_loan;

EXIT WHEN loan\_cursor%NOTFOUND;

IF v\_loan.InterestRate > 10 THEN

UPDATE Loans

SET InterestRate = InterestRate - 0.5

WHERE LoanID = v\_loan.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Reduced interest for Loan ID: ' || v\_loan.LoanID);

ELSE

UPDATE Loans

SET InterestRate = InterestRate + 0.25

WHERE LoanID = v\_loan.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Increased interest for Loan ID: ' || v\_loan.LoanID);

END IF;

END LOOP;

CLOSE loan\_cursor;

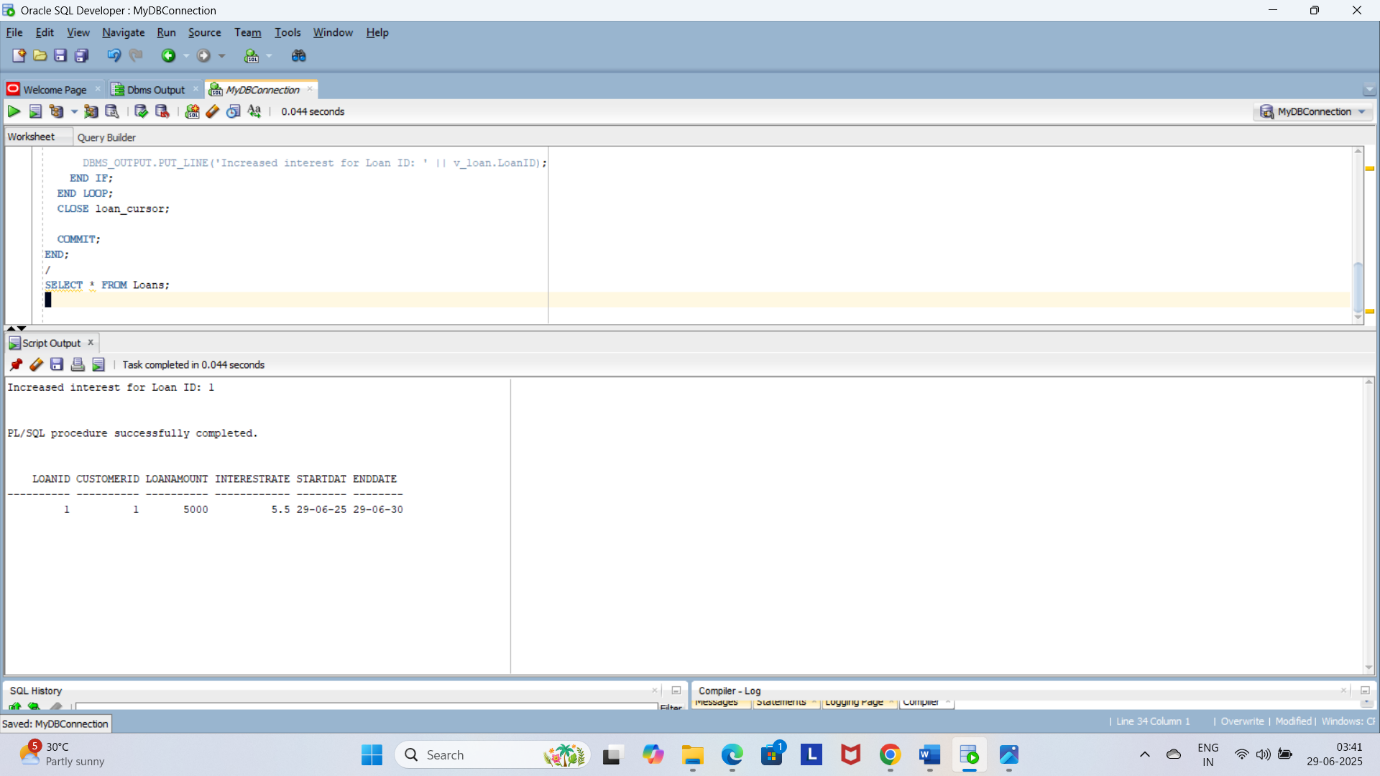
COMMIT;

END;

/

SELECT \* FROM Loans;

**Output:**

****

**Exercise 7:**

**Scenario 1:**

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\_dob DATE

);

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(

p\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

) AS

BEGIN

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

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

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

END AddCustomer;

PROCEDURE UpdateCustomer(

p\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE

) AS

BEGIN

UPDATE Customers

SET Name = p\_name, DOB = p\_dob, LastModified = SYSDATE

WHERE CustomerID = p\_id;

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

END UpdateCustomer;

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER AS

v\_balance NUMBER;

BEGIN

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

RETURN v\_balance;

END GetCustomerBalance;

END CustomerManagement;

/

SET SERVEROUTPUT ON;

BEGIN

CustomerManagement.AddCustomer(1001, 'Zara', TO\_DATE('1996-04-15', 'YYYY-MM-DD'), 9000);

CustomerManagement.UpdateCustomer(1001, 'Zara Khan', TO\_DATE('1996-04-15', 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Balance: ' || CustomerManagement.GetCustomerBalance(1001));

END;

/

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Scenario 2:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(

p\_id NUMBER,

p\_name VARCHAR2,

p\_salary NUMBER

);

PROCEDURE UpdateEmployee(

p\_id NUMBER,

p\_name VARCHAR2,

p\_salary NUMBER

);

FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

--CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

-- PROCEDURE HireEmployee(

-- p\_id NUMBER,

-- p\_name VARCHAR2,

-- p\_salary NUMBER

-- ) AS

-- BEGIN

-- INSERT INTO Employees(EmployeeID, Name, Salary)

-- VALUES (p\_id, p\_name, p\_salary);

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

-- END HireEmployee;

--

-- PROCEDURE UpdateEmployee(

-- p\_id NUMBER,

-- p\_name VARCHAR2,

-- p\_salary NUMBER

-- ) AS

-- BEGIN

-- UPDATE Employees

-- SET Name = p\_name, Salary = p\_salary

-- WHERE EmployeeID = p\_id;

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

-- END UpdateEmployee;

--

-- FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER AS

-- v\_salary NUMBER;

-- BEGIN

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

-- RETURN v\_salary \* 12;

-- END GetAnnualSalary;

--END EmployeeManagement;

--/

SET SERVEROUTPUT ON;

BEGIN

EmployeeManagement.HireEmployee(5001, 'Rohan', 40000);

EmployeeManagement.UpdateEmployee(5001, 'Rohan Mehta', 45000);

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || EmployeeManagement.GetAnnualSalary(5001));

END;

/

**Output:**

**A computer screen shot of a computer

AI-generated content may be incorrect.**

**Scenario 3:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(

p\_account\_id NUMBER,

p\_customer\_id NUMBER,

p\_balance NUMBER

);

PROCEDURE CloseAccount(p\_account\_id NUMBER);

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

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(

p\_account\_id NUMBER,

p\_customer\_id NUMBER,

p\_balance NUMBER

) AS

BEGIN

INSERT INTO Accounts(AccountID, CustomerID, Balance)

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

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

END OpenAccount;

PROCEDURE CloseAccount(p\_account\_id NUMBER) AS

BEGIN

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

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

END CloseAccount;

FUNCTION GetTotalBalance(p\_customer\_id NUMBER) RETURN NUMBER AS

v\_total NUMBER;

BEGIN

SELECT NVL(SUM(Balance), 0)

INTO v\_total

FROM Accounts

WHERE CustomerID = p\_customer\_id;

RETURN v\_total;

END GetTotalBalance;

END AccountOperations;

/

SET SERVEROUTPUT ON;

BEGIN

AccountOperations.OpenAccount(3001, 1001, 7000);

AccountOperations.OpenAccount(3002, 1001, 3000);

DBMS\_OUTPUT.PUT\_LINE('Total Balance: ' || AccountOperations.GetTotalBalance(1001));

AccountOperations.CloseAccount(3002);

END;

/

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**