**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

**Scenario 2:** Manage errors when updating employee salaries.

**Scenario 3:** Ensure data integrity when adding a new customer.

**Code:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Loans CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Employees CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE ErrorLog CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP SEQUENCE Transactions\_seq';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

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

);

CREATE TABLE ErrorLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

ErrorMessage VARCHAR2(4000),

ErrorDate DATE DEFAULT SYSDATE

);

CREATE SEQUENCE Transactions\_seq START WITH 3 INCREMENT BY 1;

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

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

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

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

COMMIT;

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_from\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id

FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_to\_account\_id;

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

VALUES (Transactions\_seq.NEXTVAL, p\_from\_account\_id, SYSDATE, p\_amount, 'Debit');

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

VALUES (Transactions\_seq.NEXTVAL, p\_to\_account\_id, SYSDATE, p\_amount, 'Credit');

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage) VALUES ('SafeTransferFunds Error: ' || SQLERRM);

END;

/

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_emp\_id IN NUMBER,

p\_percent IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_emp\_id;

IF SQL%ROWCOUNT = 0 THEN

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

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage) VALUES ('UpdateSalary Error: ' || SQLERRM);

END;

/

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) AS

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

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage) VALUES ('AddNewCustomer Error: Duplicate ID');

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage) VALUES ('AddNewCustomer Error: ' || SQLERRM);

END;

/

BEGIN

SafeTransferFunds(1, 2, 200);

END;

/

BEGIN

UpdateSalary(1, 10);

END;

/

BEGIN

AddNewCustomer(3, 'Ravi Kumar', TO\_DATE('1992-01-01', 'YYYY-MM-DD'), 2000);

END;

/

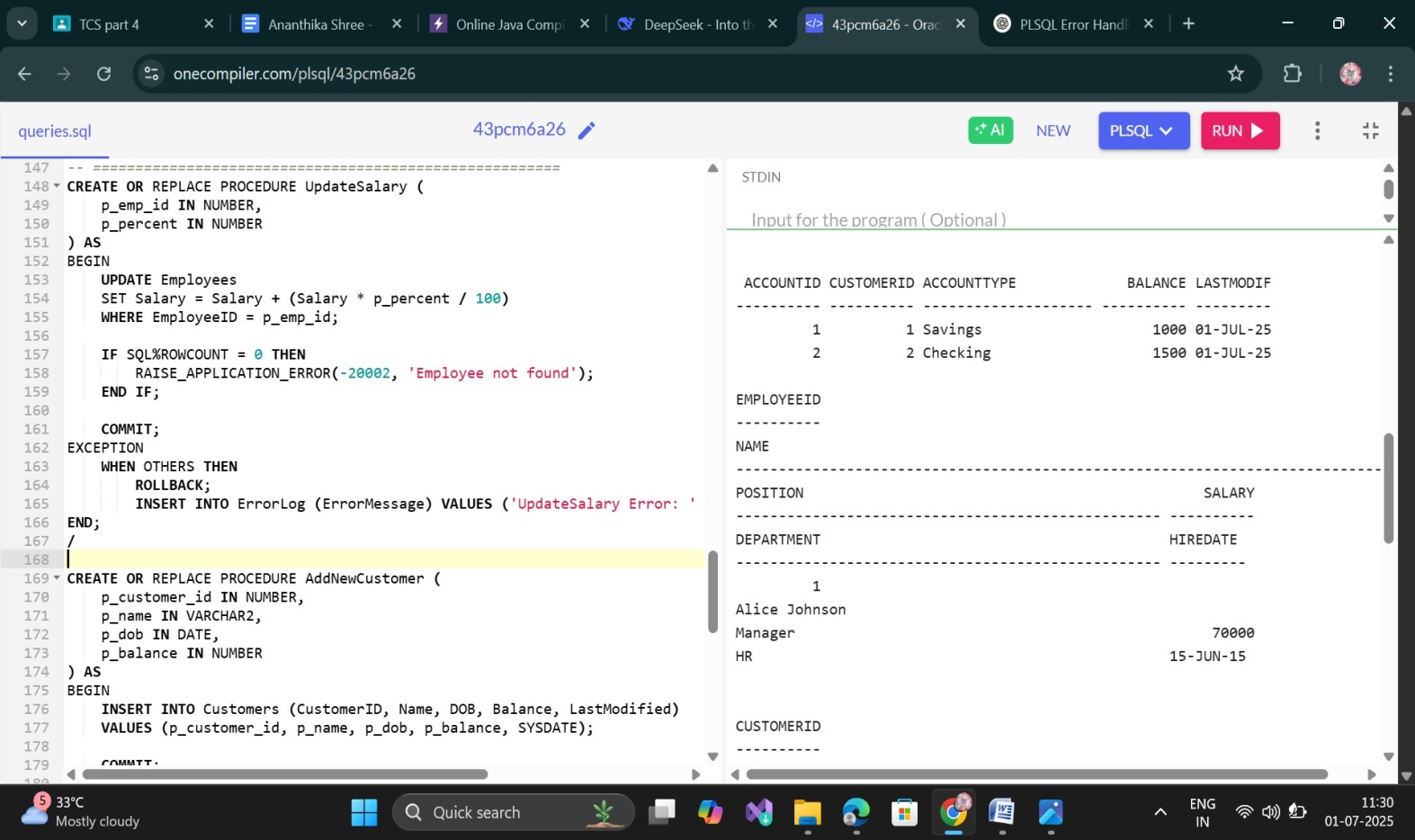
SELECT \* FROM ErrorLog;

SELECT \* FROM Accounts;

SELECT \* FROM Employees WHERE EmployeeID = 1;

SELECT \* FROM Customers;

Output:



**Exercise 4: Functions**

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

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

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

**Code:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

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)

);

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

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

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

COMMIT;

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;

/

SELECT Name, DOB, CalculateAge(DOB) AS Age FROM Customers;

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_amount NUMBER,

p\_annual\_rate NUMBER,

p\_years NUMBER

)

RETURN NUMBER

IS

monthly\_rate NUMBER;

months NUMBER;

emi NUMBER;

BEGIN

IF p\_annual\_rate = 0 THEN

RETURN ROUND(p\_amount / (p\_years \* 12), 2);

END IF;

monthly\_rate := p\_annual\_rate / 12 / 100;

months := p\_years \* 12;

emi := p\_amount \* monthly\_rate \* POWER(1 + monthly\_rate, months) /

(POWER(1 + monthly\_rate, months) - 1);

RETURN ROUND(emi, 2);

END;

/

SELECT CalculateMonthlyInstallment(50000, 6, 5) AS EMI FROM dual;

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;

RETURN CASE WHEN v\_balance >= p\_amount THEN TRUE ELSE FALSE END;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

WHEN OTHERS THEN

RETURN FALSE;

END;

/

SET SERVEROUTPUT ON;

DECLARE

result BOOLEAN;

BEGIN

result := HasSufficientBalance(1, 300);

IF result THEN

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

ELSE

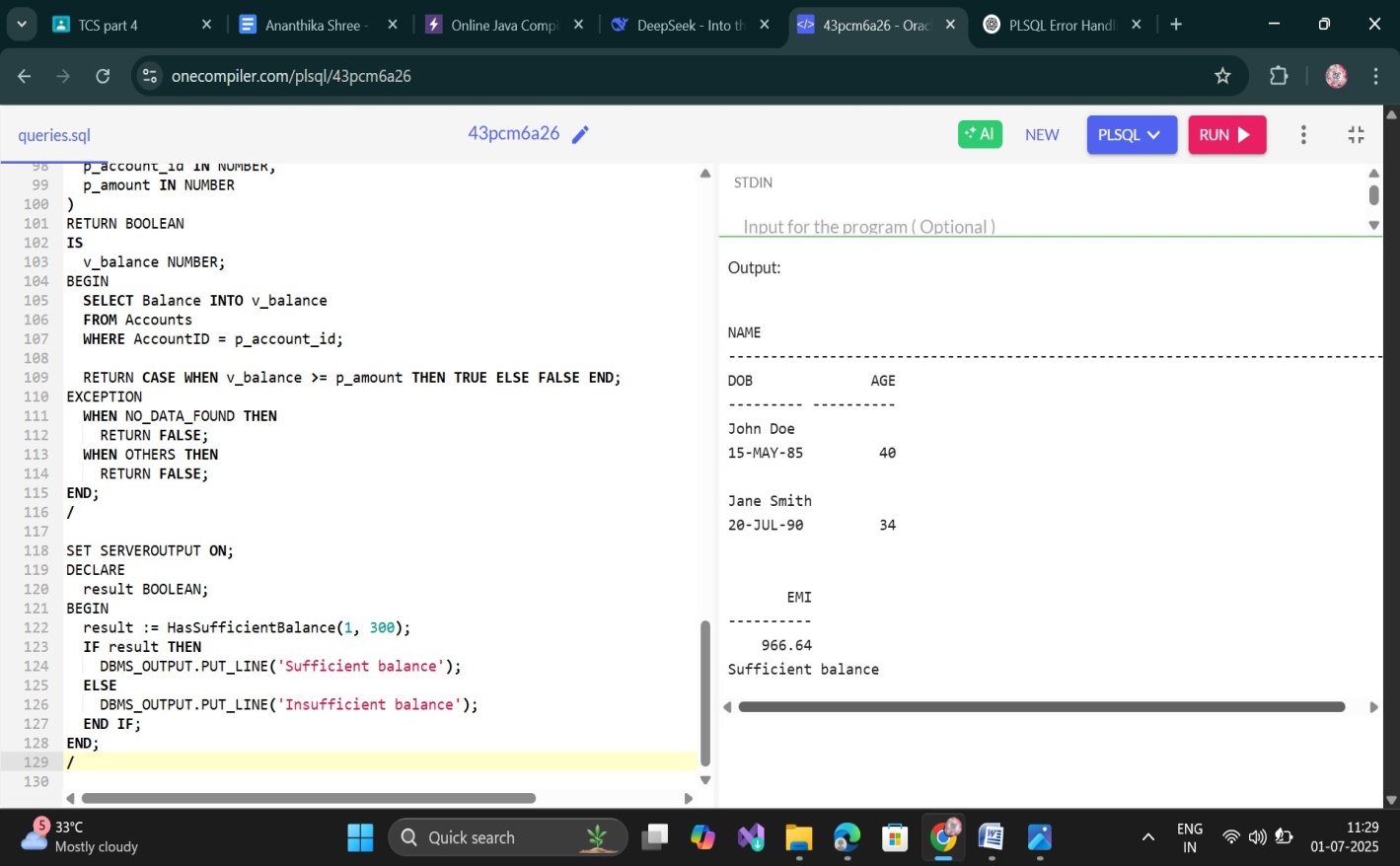
DBMS\_OUTPUT.PUT\_LINE('Insufficient balance');

END IF;

END;

/

Output:



**Exercise 5: Triggers**

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

**Scenario 2:** Maintain an audit log for all transactions.

**Scenario 3:**Enforce business rules on deposits and withdrawals.

**Code:**

BEGIN

EXECUTE IMMEDIATE 'DROP TRIGGER UpdateCustomerLastModified';

EXECUTE IMMEDIATE 'DROP TRIGGER LogTransaction';

EXECUTE IMMEDIATE 'DROP TRIGGER CheckTransactionRules';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE AuditLog CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Transactions CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Employees CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

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 AuditLog (

AuditID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

ActionType VARCHAR2(20),

ActionDate DATE,

AccountID NUMBER,

Amount NUMBER,

TransactionType VARCHAR2(10)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

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

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

COMMIT;

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

UPDATE Customers SET Balance = 1200 WHERE CustomerID = 1;

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (ActionType, ActionDate, AccountID, Amount, TransactionType)

VALUES ('INSERT', SYSDATE, :NEW.AccountID, :NEW.Amount, :NEW.TransactionType);

END;

/

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' THEN

IF :NEW.Amount > v\_balance THEN

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

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

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

END IF;

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'Invalid Transaction Type');

END IF;

END;

/

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

VALUES (1, 1, SYSDATE, 300, 'Deposit');

SET SERVEROUTPUT ON;

BEGIN

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

VALUES (2, 1, SYSDATE, 5000, 'Withdrawal');

EXCEPTION

WHEN OTHERS THEN

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

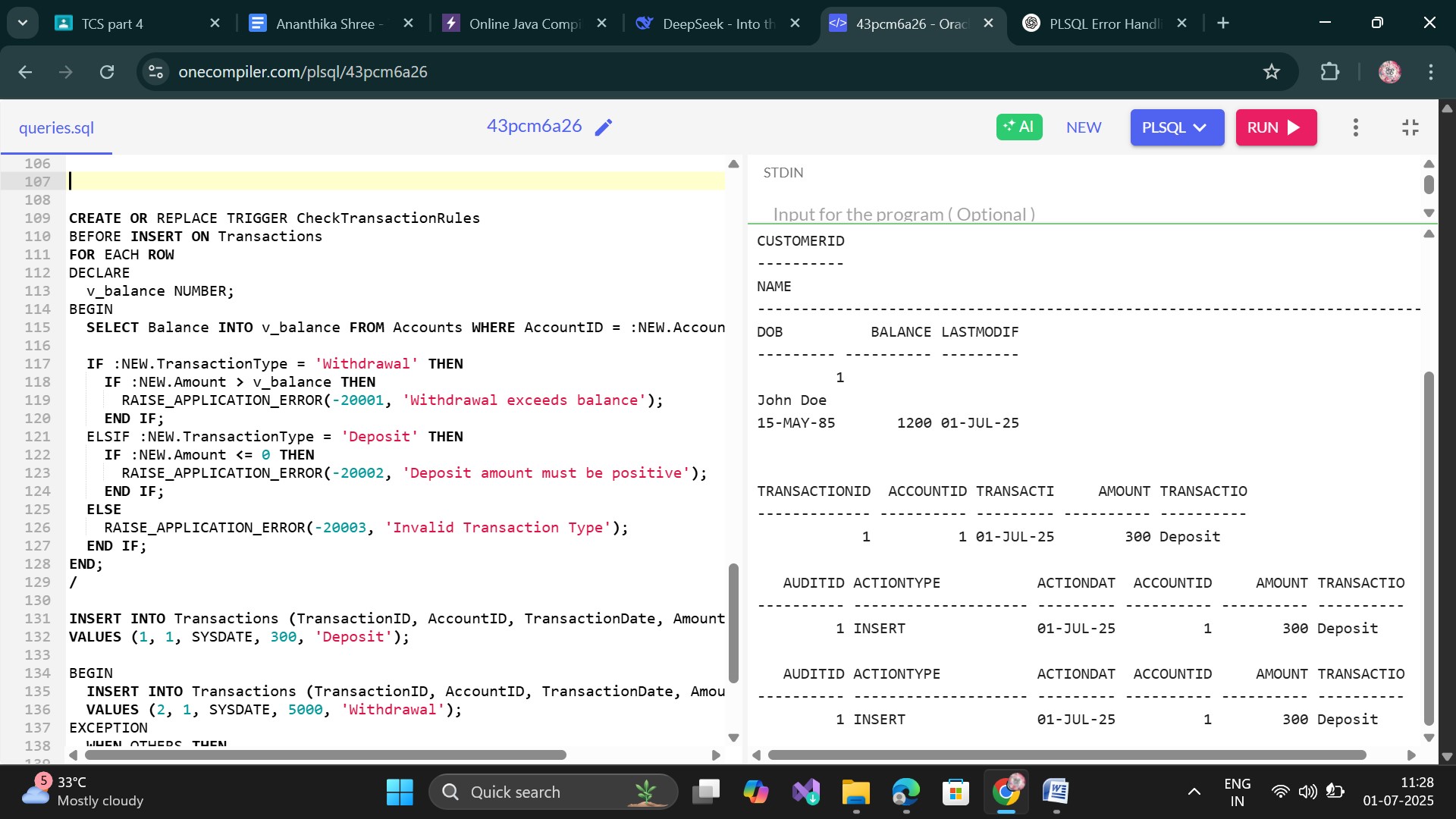
END;

/

SELECT \* FROM Customers;

SELECT \* FROM Transactions;

SELECT \* FROM AuditLog;

**Output:**

**Exercise 6: Cursors**

**Scenario 1:** Generate monthly statements for all customers.

**Scenario 2:** Apply annual fee to all accounts.

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

**Code:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Loans CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Transactions CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

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

);

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

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

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

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

COMMIT;

SET SERVEROUTPUT ON;

DECLARE

CURSOR trans\_cursor IS

SELECT c.CustomerID, c.Name, t.AccountID, 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 TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM')

ORDER BY c.CustomerID, t.TransactionDate;

v\_cust\_id Customers.CustomerID%TYPE;

v\_name Customers.Name%TYPE;

v\_acc\_id Accounts.AccountID%TYPE;

v\_date Transactions.TransactionDate%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_type Transactions.TransactionType%TYPE;

BEGIN

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

OPEN trans\_cursor;

LOOP

FETCH trans\_cursor INTO v\_cust\_id, v\_name, v\_acc\_id, v\_date, v\_amount, v\_type;

EXIT WHEN trans\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || v\_name || ' (ID: ' || v\_cust\_id || ')');

DBMS\_OUTPUT.PUT\_LINE(' Account: ' || v\_acc\_id || ', Date: ' || TO\_CHAR(v\_date, 'DD-Mon-YYYY') ||

', Amount: ' || v\_amount || ', Type: ' || v\_type);

END LOOP;

CLOSE trans\_cursor;

END;

/

DECLARE

CURSOR acc\_cursor IS

SELECT AccountID, Balance FROM Accounts;

v\_acc\_id Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_fee CONSTANT NUMBER := 100;

BEGIN

OPEN acc\_cursor;

LOOP

FETCH acc\_cursor INTO v\_acc\_id, v\_balance;

EXIT WHEN acc\_cursor%NOTFOUND;

IF v\_balance >= v\_fee THEN

UPDATE Accounts

SET Balance = Balance - v\_fee,

LastModified = SYSDATE

WHERE AccountID = v\_acc\_id;

DBMS\_OUTPUT.PUT\_LINE('Fee of ' || v\_fee || ' applied to Account ' || v\_acc\_id);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in Account ' || v\_acc\_id);

END IF;

END LOOP;

CLOSE acc\_cursor;

COMMIT;

END;

/

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate FROM Loans;

v\_loan\_id Loans.LoanID%TYPE;

v\_old\_rate Loans.InterestRate%TYPE;

v\_new\_rate NUMBER;

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_loan\_id, v\_old\_rate;

EXIT WHEN loan\_cursor%NOTFOUND;

IF v\_old\_rate < 6 THEN

v\_new\_rate := v\_old\_rate + 0.5;

ELSE

v\_new\_rate := v\_old\_rate + 0.25;

END IF;

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE LoanID = v\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Loan ' || v\_loan\_id || ': Interest updated from ' || v\_old\_rate || ' to ' || v\_new\_rate);

END LOOP;

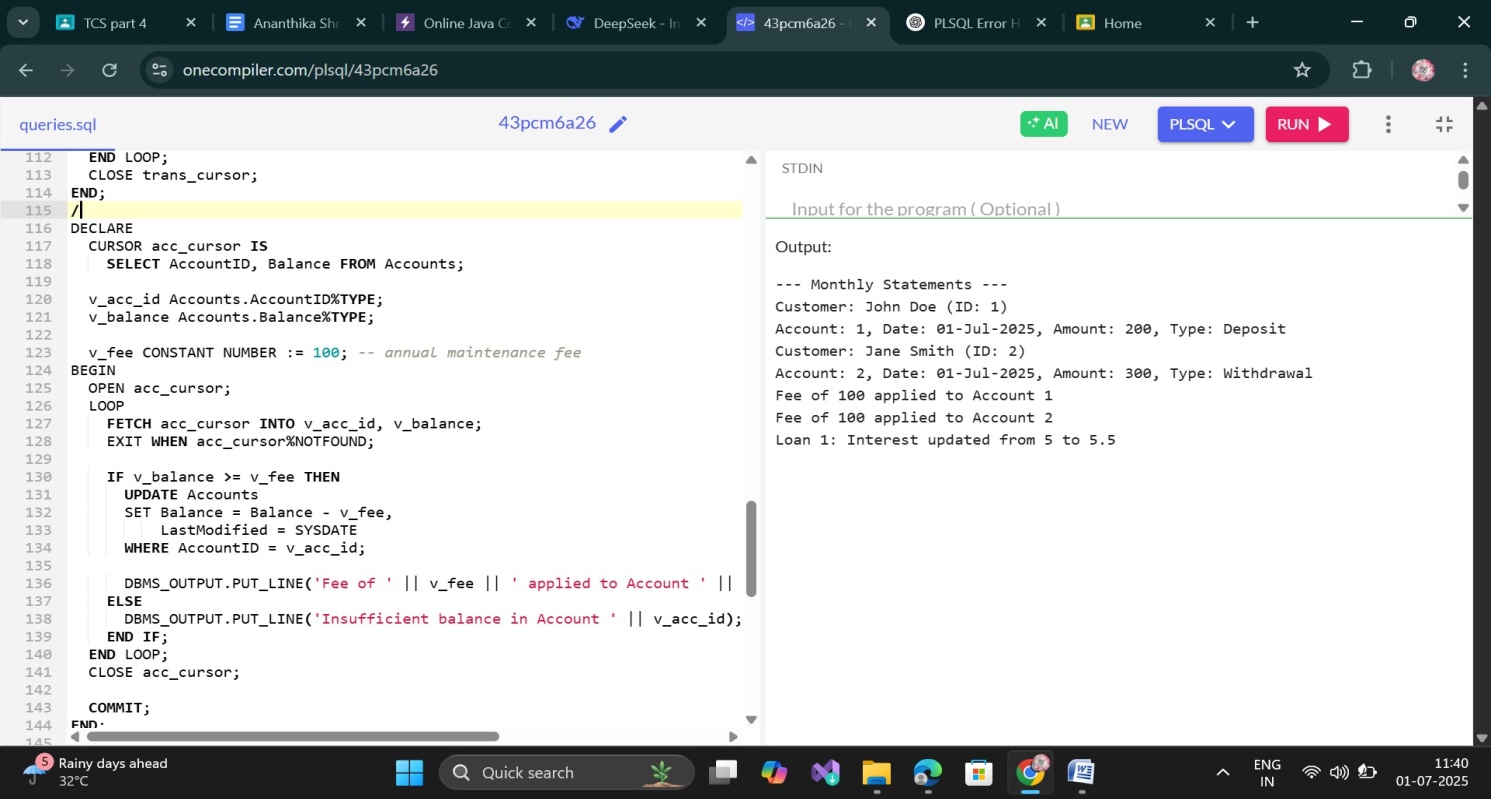
CLOSE loan\_cursor;

COMMIT;

END;

/

Output:



**Exercise 7: Packages**

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

**Scenario 2:** Create a package to manage employee data.

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

**Code:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Loans CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Transactions CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

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

);

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

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

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

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

COMMIT;

CREATE OR REPLACE PACKAGE CustomerManagement IS

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 GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

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);

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;

END;

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

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

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE EmployeeManagement IS

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

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

FUNCTION CalculateAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

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

BEGIN

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

VALUES (p\_id, p\_name, p\_position, p\_salary, p\_dept, p\_hiredate);

END;

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

BEGIN

UPDATE Employees

SET Salary = p\_salary

WHERE EmployeeID = p\_id;

END;

FUNCTION CalculateAnnualSalary(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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE AccountOperations IS

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 AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

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

BEGIN

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

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

END;

PROCEDURE CloseAccount(p\_accid NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_accid;

END;

FUNCTION GetTotalBalance(p\_custid NUMBER) RETURN NUMBER IS

v\_total NUMBER := 0;

BEGIN

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

RETURN NVL(v\_total, 0);

END;

END AccountOperations;

/

SET SERVEROUTPUT ON;

BEGIN

CustomerManagement.AddCustomer(3, 'Ravi Kumar', TO\_DATE('1992-01-01','YYYY-MM-DD'), 2000);

CustomerManagement.UpdateCustomer(3, 'Ravi K.', 2500);

DBMS\_OUTPUT.PUT\_LINE('Customer 3 Balance: ' || CustomerManagement.GetCustomerBalance(3));

END;

/

BEGIN

EmployeeManagement.HireEmployee(3, 'Kiran Kumar', 'Analyst', 50000, 'Finance', TO\_DATE('2019-01-10','YYYY-MM-DD'));

EmployeeManagement.UpdateEmployee(3, 55000);

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || EmployeeManagement.CalculateAnnualSalary(3));

END;

/

BEGIN

AccountOperations.OpenAccount(3, 3, 'Savings', 5000);

DBMS\_OUTPUT.PUT\_LINE('Total Balance for Customer 3: ' || AccountOperations.GetTotalBalance(3));

AccountOperations.CloseAccount(3);

END;

/

Output:

