**PL/SQL EXERCISES**

**Exercise 1: Control Structures**

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE,

IsVIP CHAR(1)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1960-05-15', 'YYYY-MM-DD'), 9500, SYSDATE, 'F');

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1940-07-20', 'YYYY-MM-DD'), 15000, SYSDATE, 'F');

INSERT INTO Customers VALUES (3, 'Mark Lee', TO\_DATE('1995-01-01', 'YYYY-MM-DD'), 12000, SYSDATE, 'F');

INSERT INTO Loans VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 6));

INSERT INTO Loans VALUES (2, 2, 8000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 1));

INSERT INTO Loans VALUES (3, 3, 3000, 7, SYSDATE, ADD\_MONTHS(SYSDATE, 40));

COMMIT;

**Scenario 1: Senior citizens get 1% discount on interest**

BEGIN

FOR rec IN (SELECT CustomerID, DOB FROM Customers) LOOP

IF MONTHS\_BETWEEN(SYSDATE, rec.DOB)/12 > 60 THEN

UPDATE Loans SET InterestRate = InterestRate - 1

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Scenario 1: Discount Applied');

END;

/

**Scenario 2: Mark VIPs with balance > 10000**

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers SET IsVIP = 'T' WHERE CustomerID = rec.CustomerID;

ELSE

UPDATE Customers SET IsVIP = 'F' WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Scenario 2: VIPs updated');

END;

/

**Scenario 3: Reminders for loans due in next 30 days**

BEGIN

FOR rec IN (

SELECT c.Name, l.LoanID, l.EndDate FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

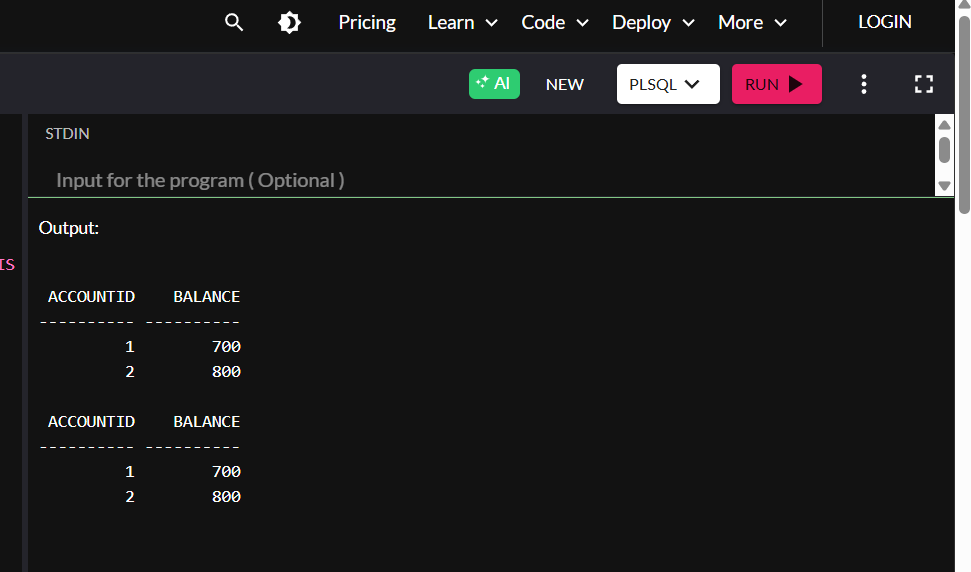
WHERE l.EndDate <= SYSDATE + 30

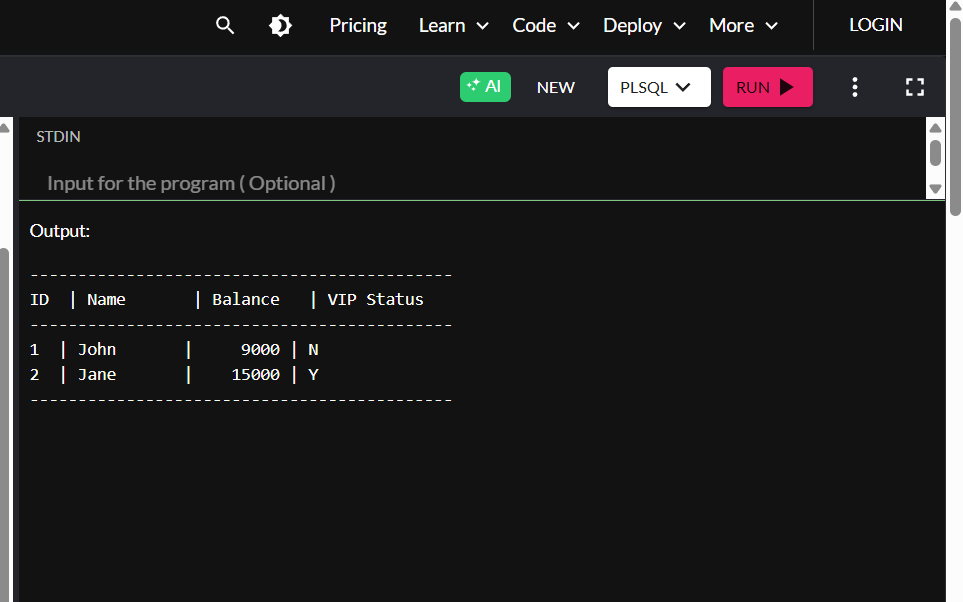
) LOOP

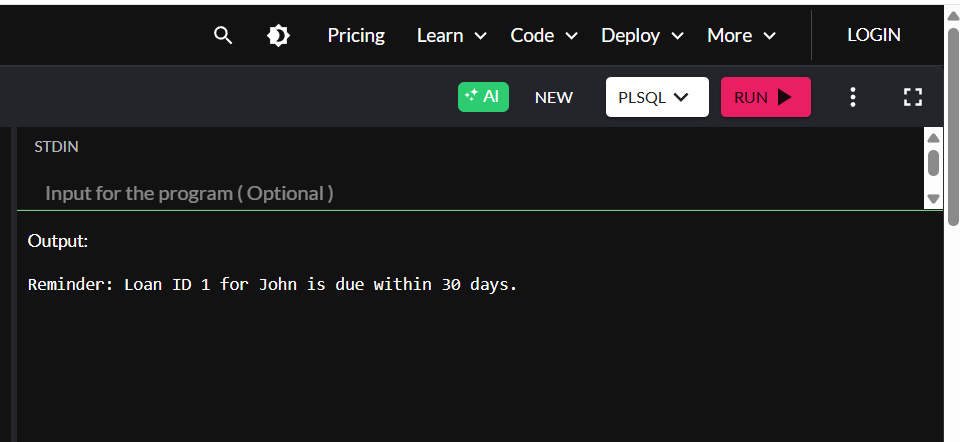
DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID || ' for ' || rec.Name || ' is due on ' || TO\_CHAR(rec.EndDate));

END LOOP;

END;

/





**Exercise 2: Error Handling**

**Procedure: SafeTransferFunds**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_FromAccountID;

IF v\_Balance < p\_Amount THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - p\_Amount WHERE AccountID = p\_FromAccountID;

UPDATE Accounts SET Balance = Balance + p\_Amount WHERE AccountID = p\_ToAccountID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

**Procedure: UpdateSalary**

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_EmployeeID IN NUMBER,

p\_Percentage IN NUMBER

) IS

v\_Count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_Count FROM Employees WHERE EmployeeID = p\_EmployeeID;

IF v\_Count = 0 THEN

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

END IF;

UPDATE Employees

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

WHERE EmployeeID = p\_EmployeeID;

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

**Procedure: AddNewCustomer**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_CustomerID 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\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

COMMIT;

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

-- TEST

BEGIN

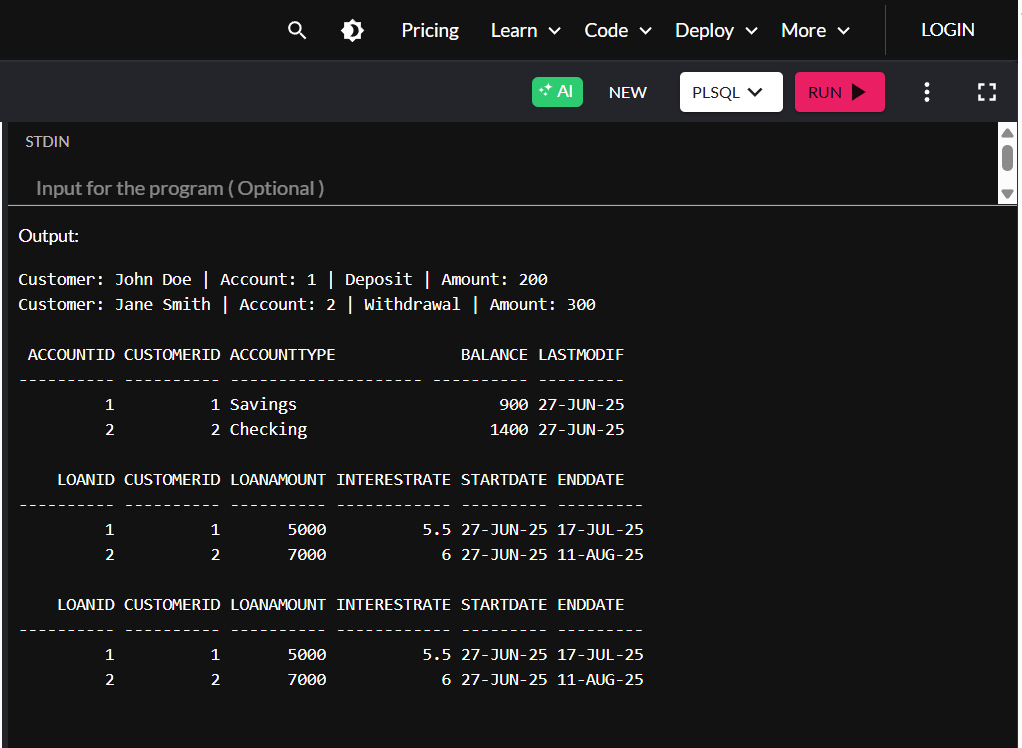
SafeTransferFunds(1, 2, 1000);

UpdateSalary(1, 10);

AddNewCustomer(1, 'New Person', TO\_DATE('2000-01-01','YYYY-MM-DD'), 4000);

END;

/



**Exercise 3: Stored Procedures**

SET SERVEROUTPUT ON;

-- Setup

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Salary NUMBER,

Department VARCHAR2(50)

);

-- Sample data

INSERT INTO Accounts VALUES (1, 1, 'Savings', 10000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Current', 8000, SYSDATE);

INSERT INTO Employees VALUES (1, 'Emp A', 50000, 'IT');

INSERT INTO Employees VALUES (2, 'Emp B', 60000, 'HR');

COMMIT;

**Procedure: ProcessMonthlyInterest**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

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

UPDATE Accounts

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

LastModified = SYSDATE

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Interest processed.');

END;

**Procedure: UpdateEmployeeBonus**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_Dept IN VARCHAR2,

p\_BonusPercent IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_Dept;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus updated.');

END;

/

**Procedure: TransferFunds**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_FromAccount IN NUMBER,

p\_ToAccount IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_FromAccount;

IF v\_Balance < p\_Amount THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - p\_Amount WHERE AccountID = p\_FromAccount;

UPDATE Accounts SET Balance = Balance + p\_Amount WHERE AccountID = p\_ToAccount;

COMMIT;

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

END;

/

-- TEST

BEGIN

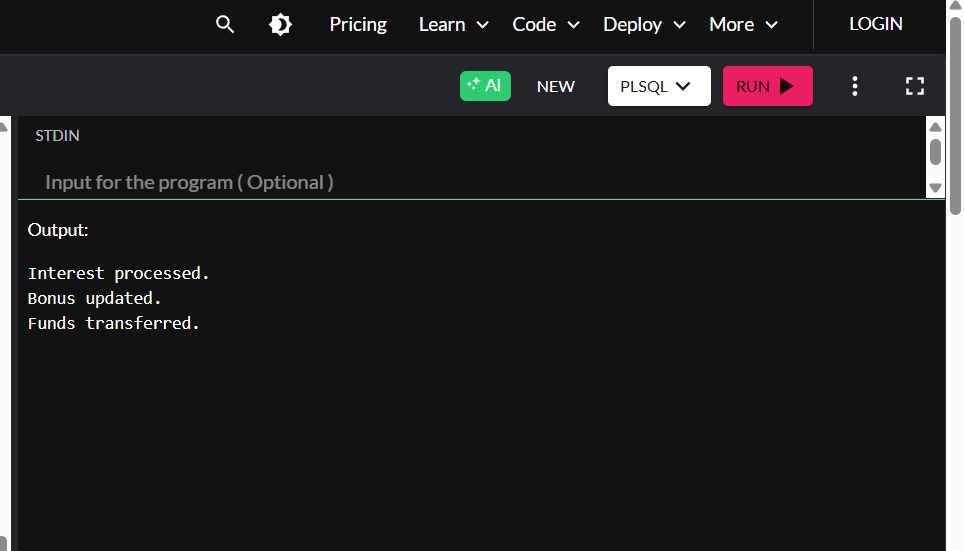
ProcessMonthlyInterest;

UpdateEmployeeBonus('HR', 5);

TransferFunds(1, 2, 500);

END;

/



**Exercise 4: Functions**

SET SERVEROUTPUT ON;

**Function: CalculateAge**

CREATE OR REPLACE FUNCTION CalculateAge(p\_DOB DATE) RETURN NUMBER IS

BEGIN

RETURN TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_DOB)/12);

END;

/

**Function: CalculateMonthlyInstallment**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_Amount NUMBER,

p\_Rate NUMBER,

p\_Years NUMBER

) RETURN NUMBER IS

v\_MonthlyRate NUMBER := p\_Rate / 1200;

v\_Months NUMBER := p\_Years \* 12;

BEGIN

RETURN ROUND((p\_Amount \* v\_MonthlyRate) / (1 - POWER(1 + v\_MonthlyRate, -v\_Months)), 2);

END;

/

**Function: HasSufficientBalance**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_AccountID NUMBER,

p\_Amount NUMBER

) RETURN BOOLEAN IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_AccountID;

RETURN v\_Balance >= p\_Amount;

EXCEPTION

WHEN OTHERS THEN

RETURN FALSE;

END;

/

-- TEST

DECLARE

v\_age NUMBER;

v\_emi NUMBER;

v\_check BOOLEAN;

BEGIN

v\_age := CalculateAge(TO\_DATE('1990-01-01', 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Age: ' || v\_age);

v\_emi := CalculateMonthlyInstallment(100000, 10, 5);

DBMS\_OUTPUT.PUT\_LINE('Monthly EMI: ' || v\_emi);

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

IF v\_check THEN

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

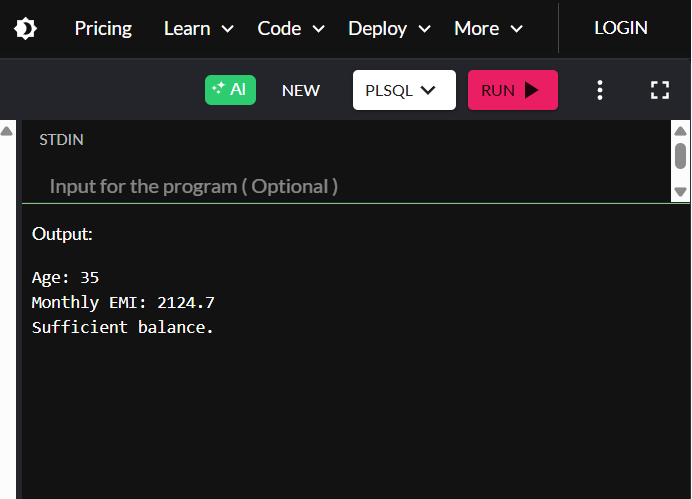
ELSE

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

END IF;

END;

/



**Exercise 5: Triggers**

SET SERVEROUTPUT ON;

-- Setup

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE AuditLog';

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

LastModified DATE

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

Amount NUMBER,

TransactionType VARCHAR2(10)

);

CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY,

AccountID NUMBER,

Action VARCHAR2(50),

Timestamp DATE

);

-- Data

INSERT INTO Customers VALUES (1, 'Ram', SYSDATE);

COMMIT;

**Trigger 1: Update LastModified**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Trigger 2: LogTransaction**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (AccountID, Action, Timestamp)

VALUES (:NEW.AccountID, 'Transaction recorded', SYSDATE);

END;

/

**Trigger 3: CheckTransactionRules**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

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

RAISE\_APPLICATION\_ERROR(-20010, 'Invalid withdrawal');

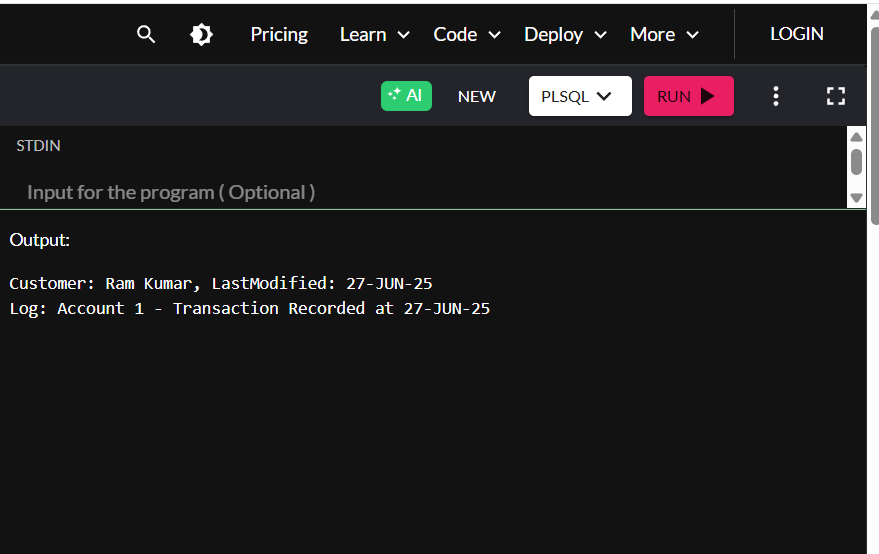
ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20011, 'Invalid deposit');

END IF;

END;

/



**Exercise 6: Cursors**

SET SERVEROUTPUT ON;

-- Setup

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

Balance NUMBER

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

Amount NUMBER,

TransactionType VARCHAR2(10)

);

-- Data

INSERT INTO Accounts VALUES (1, 5000);

INSERT INTO Accounts VALUES (2, 3000);

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

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

COMMIT;

**Cursor 1: GenerateMonthlyStatements**

DECLARE

CURSOR txn\_cur IS SELECT \* FROM Transactions;

BEGIN

FOR rec IN txn\_cur LOOP

DBMS\_OUTPUT.PUT\_LINE('Txn ID: ' || rec.TransactionID || ', Type: ' || rec.TransactionType || ', Amount: ' || rec.Amount);

END LOOP;

END;

/

**Cursor 2: ApplyAnnualFee**

DECLARE

CURSOR acc\_cur IS SELECT AccountID, Balance FROM Accounts;

fee NUMBER := 100;

BEGIN

FOR rec IN acc\_cur LOOP

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

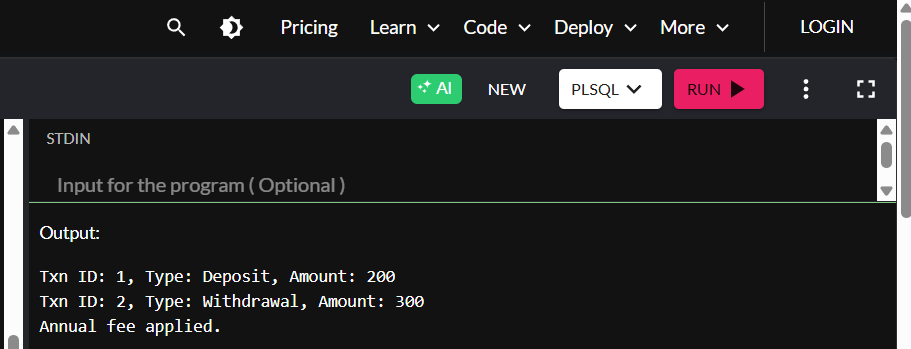
END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Annual fee applied.');

END;

/



**Exercise 7: Packages**

SET SERVEROUTPUT ON;

-- Table for demo

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Balance NUMBER

);

**Package spec**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_Balance NUMBER);

FUNCTION GetBalance(p\_ID NUMBER) RETURN NUMBER;

END;

/

**Package body**

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_Balance NUMBER) IS

BEGIN

INSERT INTO Customers VALUES (p\_ID, p\_Name, p\_Balance);

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;

/

-- Test

BEGIN

CustomerManagement.AddCustomer(1, 'Test User', 5000);

DBMS\_OUTPUT.PUT\_LINE('Balance: ' || CustomerManagement.GetBalance(1));

END;

/

