# Week-2

**PL/SQL programming**

**Exercise 1- Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

* + **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

-- Create table

CREATE TABLE Customers (

CustomerID INT,

Name VARCHAR(100),

Age INT,

Balance NUMBER,

InterestRate NUMBER,

IsVIP VARCHAR(5)

);

-- Insert data

INSERT INTO Customers VALUES (1, 'John', 65, 15000, 5.5, 'FALSE');

INSERT INTO Customers VALUES (2, 'Alice', 45, 9000, 6.0, 'FALSE');

-- PL/SQL block to apply discount

DECLARE

CURSOR cust\_cursor IS

SELECT CustomerID, InterestRate, Age FROM Customers;

BEGIN

FOR cust IN cust\_cursor LOOP

IF cust.Age > 60 THEN

UPDATE Customers

SET InterestRate = InterestRate - (InterestRate \* 0.01)

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

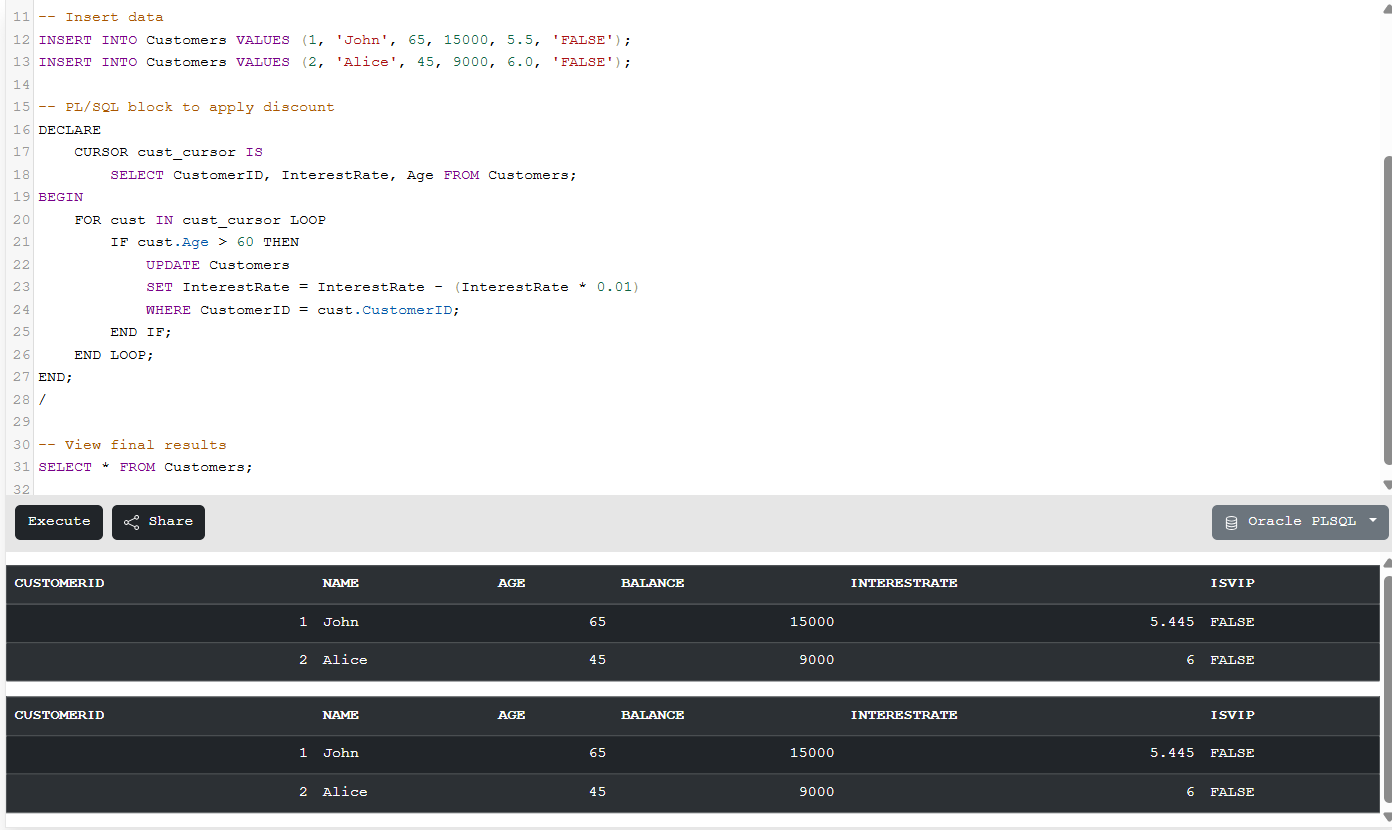
END;

/

-- View final results

SELECT \* FROM Customers;

**Output:**

****

**Scenario 2:** A customer can be promoted to VIP status based on their balance.

* + **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

-- Create table

CREATE TABLE Customers (

CustomerID INT,

Name VARCHAR(100),

Age INT,

Balance NUMBER,

InterestRate NUMBER,

IsVIP VARCHAR(5)

);

-- Insert data

INSERT INTO Customers VALUES (1, 'John', 65, 15000, 5.5, 'FALSE');

INSERT INTO Customers VALUES (2, 'Alice', 45, 9000, 6.0, 'FALSE');

-- PL/SQL block to apply discount

DECLARE

CURSOR cust\_cursor IS

SELECT CustomerID, Balance FROM Customers;

BEGIN

FOR cust IN cust\_cursor LOOP

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

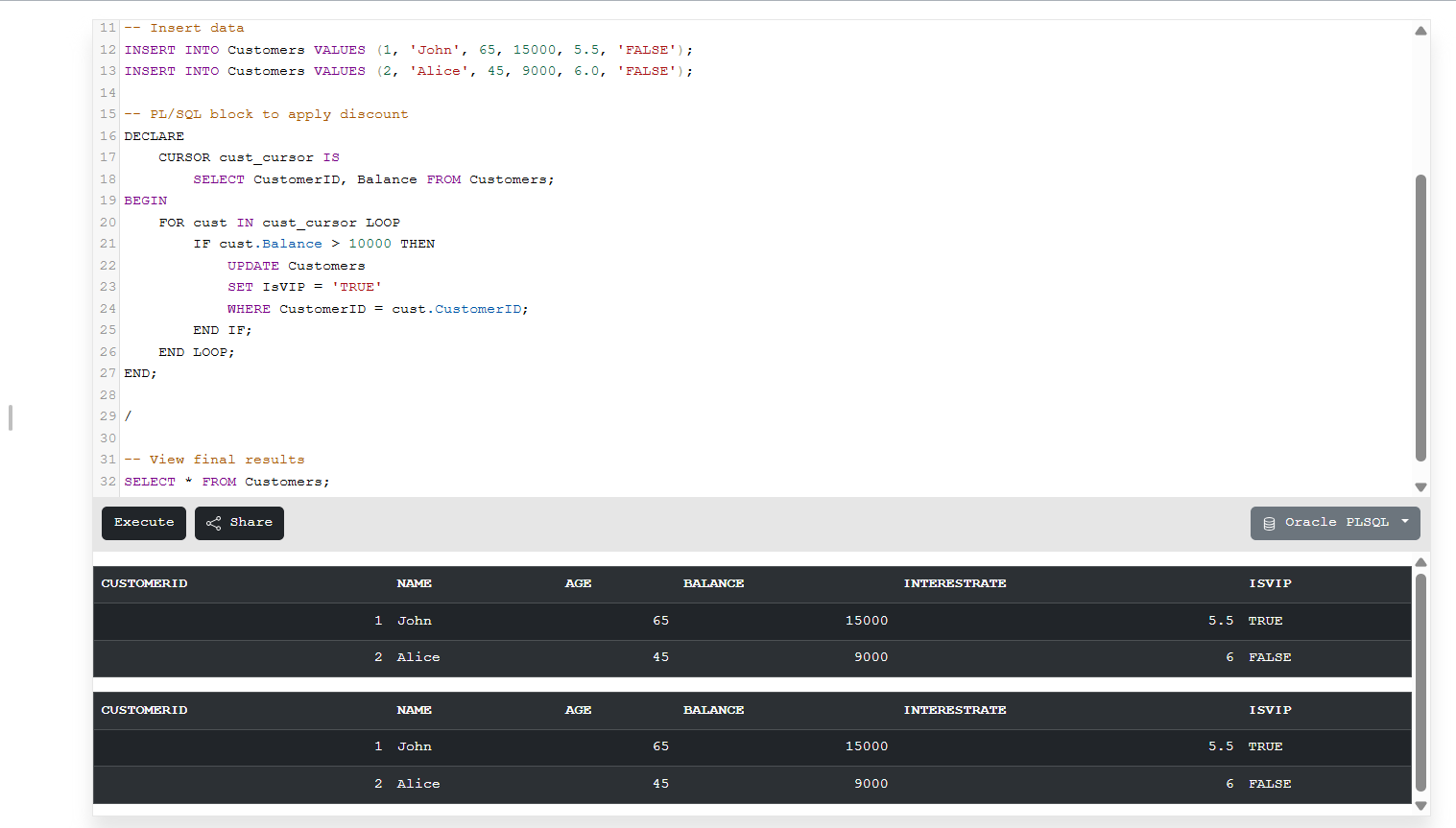
END;

/

-- View final results

SELECT \* FROM Customers;

**Output:**



**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

-- Create Loans table

CREATE TABLE Loans (

LoanID INT,

CustomerID INT,

DueDate DATE

);

-- Insert sample loan data

INSERT INTO Loans VALUES (101, 1, SYSDATE + 10);

INSERT INTO Loans VALUES (102, 2, SYSDATE + 45);

-- PL/SQL block to send reminders for loans due in next 30 days

DECLARE

CURSOR loan\_cursor IS

SELECT CustomerID, DueDate FROM Loans

WHERE DueDate <= SYSDATE + 30;

BEGIN

FOR loan IN loan\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan for customer ' || loan.CustomerID || ' is due on ' || loan.DueDate);

END LOOP;

END;

/

**Output:**



**Exercise 3- Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

* + **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

-- Step 1: Create table

CREATE TABLE SavingsAccounts (

AccountID INT,

CustomerName VARCHAR(100),

Balance NUMBER

);

-- Step 2: Insert sample data

INSERT INTO SavingsAccounts VALUES (101, 'John', 10000);

INSERT INTO SavingsAccounts VALUES (102, 'Alice', 5000);

-- Step 3: Create procedure

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE SavingsAccounts

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

END;

/

-- Step 4: Execute procedure

BEGIN

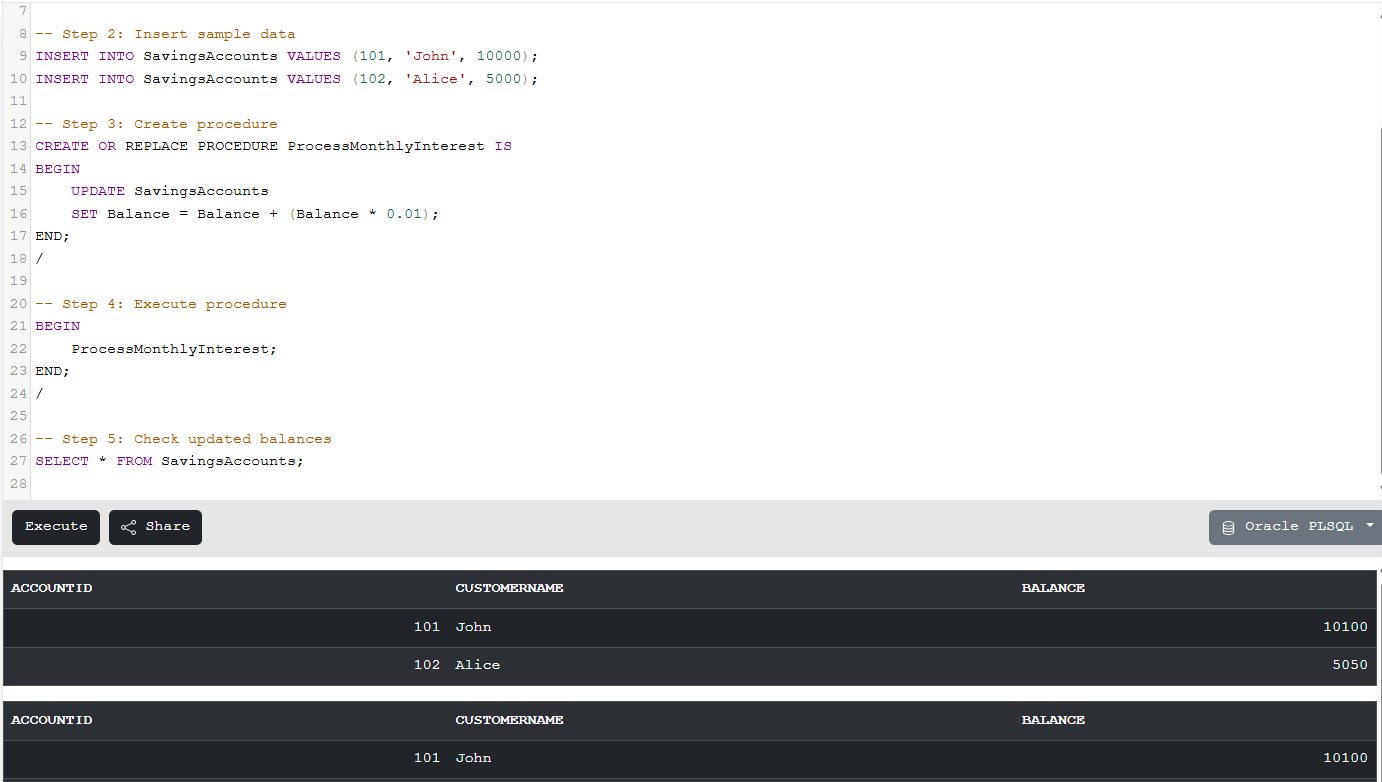
ProcessMonthlyInterest;

END;

/

-- Step 5: Check updated balances

SELECT \* FROM SavingsAccounts;

**Output:**

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

* + **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

-- Step 1: Create table

CREATE TABLE Employees (

EmployeeID INT,

Name VARCHAR(100),

Salary NUMBER,

DepartmentID INT

);

-- Step 2: Insert sample data

INSERT INTO Employees VALUES (1, 'John', 50000, 101);

INSERT INTO Employees VALUES (2, 'Alice', 60000, 102);

INSERT INTO Employees VALUES (3, 'Bob', 55000, 101);

-- Step 3: Create procedure

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(dept\_id NUMBER, bonus\_pct NUMBER) IS

BEGIN

UPDATE Employees

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

WHERE DepartmentID = dept\_id;

END;

/

-- Step 4: Execute procedure

BEGIN

UpdateEmployeeBonus(101, 10); -- 10% bonus for dept 101

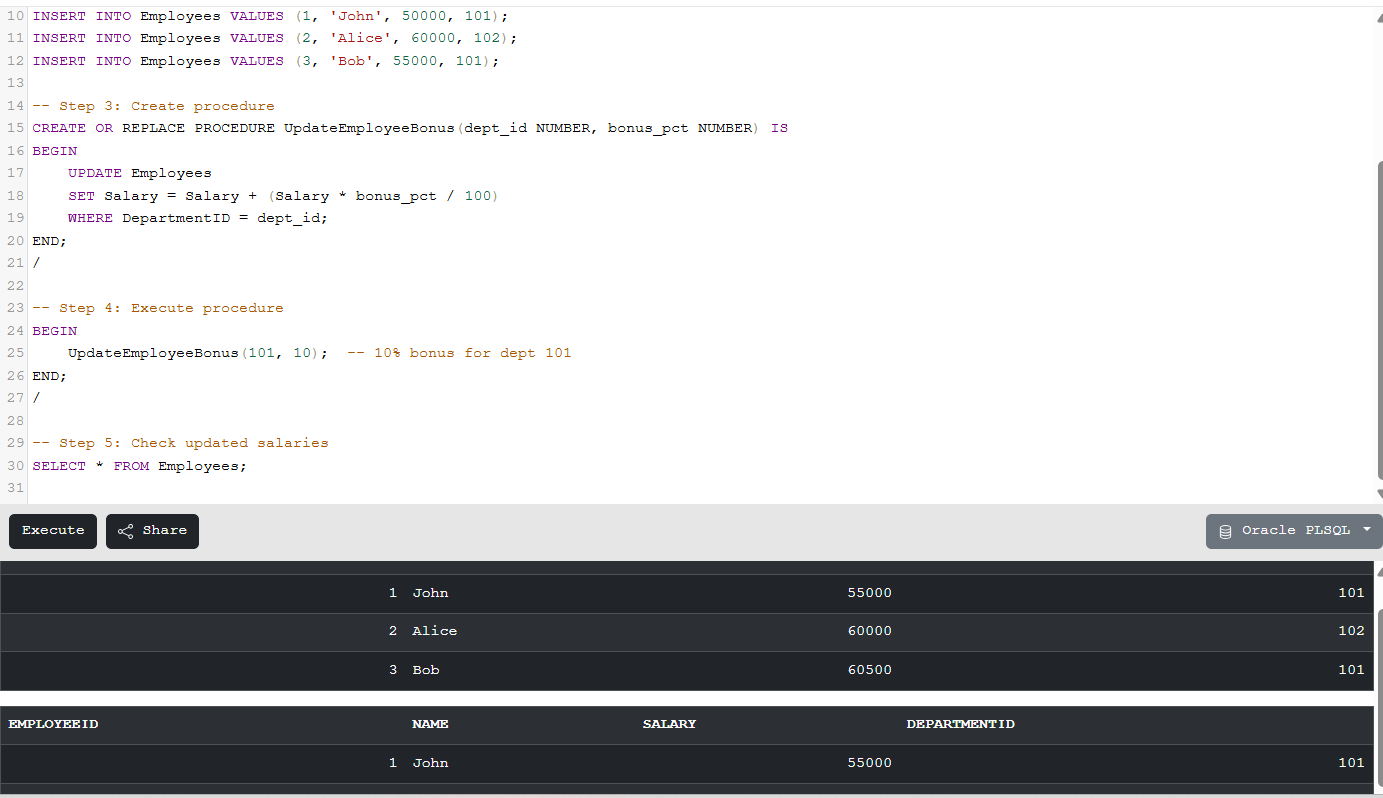
END;

/

-- Step 5: Check updated salaries

SELECT \* FROM Employees;

**Output:**



**Scenario 3:** Customers should be able to transfer funds between their accounts.

* + **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

-- Step 1: Create table

CREATE TABLE Accounts (

AccountID INT,

AccountHolder VARCHAR(100),

Balance NUMBER

);

-- Step 2: Insert sample data

INSERT INTO Accounts VALUES (201, 'John', 8000);

INSERT INTO Accounts VALUES (202, 'Alice', 3000);

-- Step 3: Create procedure

CREATE OR REPLACE PROCEDURE TransferFunds(from\_acct NUMBER, to\_acct NUMBER, amount NUMBER) IS

from\_balance NUMBER;

BEGIN

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

IF from\_balance >= amount THEN

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_acct;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_acct;

ELSE

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

END IF;

END;

/

-- Step 4: Execute procedure

BEGIN

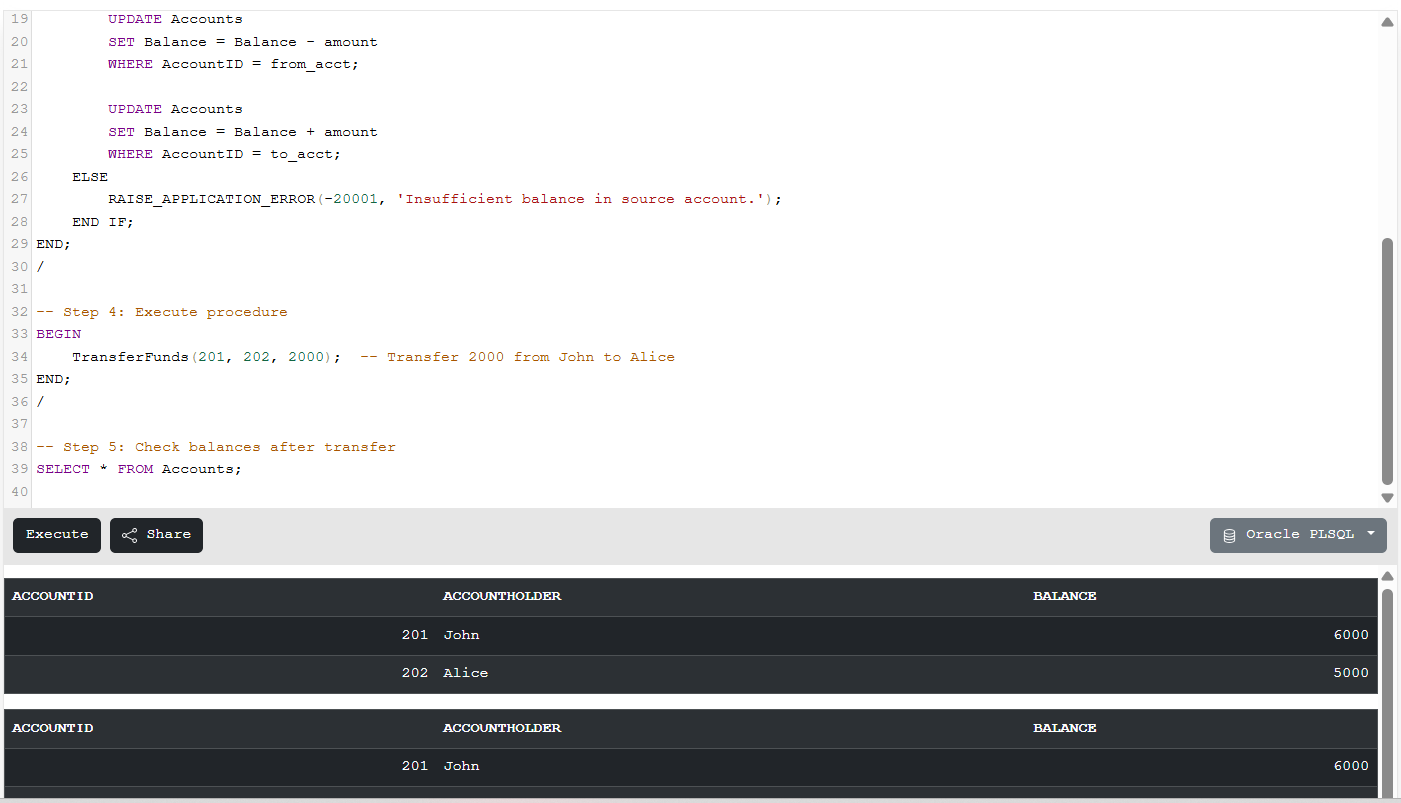
TransferFunds(201, 202, 2000); -- Transfer 2000 from John to Alice

END;

/

-- Step 5: Check balances after transfer

SELECT \* FROM Accounts;

**Output:**