**WEEK – 2**

**PL-SQL**

**(Mandatory Hands-On Exercises)**

**Exercise 1: Control Structures**

**Scenario 1: Discount for customers above 60**

**Code:**

DECLARE

TYPE customer\_type IS RECORD (

customer\_id NUMBER,

age NUMBER,

loan\_id NUMBER,

interest\_rate NUMBER

);

TYPE customer\_table IS TABLE OF customer\_type INDEX BY BINARY\_INTEGER;

v\_customers customer\_table;

v\_discount\_rate NUMBER := 1.0; -- 1% discount

BEGIN

v\_customers(1).customer\_id := 1;

v\_customers(1).age := 65;

v\_customers(1).loan\_id := 101;

v\_customers(1).interest\_rate := 5.0;

v\_customers(2).customer\_id := 2;

v\_customers(2).age := 55;

v\_customers(2).loan\_id := 102;

v\_customers(2).interest\_rate := 4.5;

v\_customers(3).customer\_id := 3;

v\_customers(3).age := 70;

v\_customers(3).loan\_id := 103;

v\_customers(3).interest\_rate := 6.0;

FOR i IN 1..v\_customers.COUNT LOOP

IF v\_customers(i).age > 60 THEN

DBMS\_OUTPUT.PUT\_LINE('Customer ' || v\_customers(i).customer\_id ||

': Old rate ' || v\_customers(i).interest\_rate || '%');

v\_customers(i).interest\_rate := v\_customers(i).interest\_rate - v\_discount\_rate;

DBMS\_OUTPUT.PUT\_LINE('Applied discount. New rate: ' ||

v\_customers(i).interest\_rate || '%');

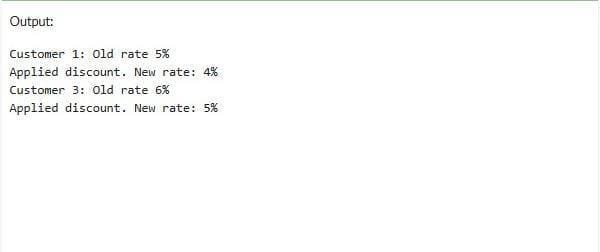
END IF;

END LOOP;

END;

/

**Output:**

****

Scenario 2: VIP promotion based on balance:

Code:

DECLARE

TYPE customer\_type IS RECORD (

customer\_id NUMBER,

balance NUMBER,

is\_vip VARCHAR2(5)

);

TYPE customer\_table IS TABLE OF customer\_type INDEX BY BINARY\_INTEGER;

v\_customers customer\_table;

v\_vip\_threshold NUMBER := 10000;

BEGIN

v\_customers(1).customer\_id := 1;

v\_customers(1).balance := 5000;

v\_customers(1).is\_vip := 'FALSE';

v\_customers(2).customer\_id := 2;

v\_customers(2).balance := 15000;

v\_customers(2).is\_vip := 'FALSE';

v\_customers(3).customer\_id := 3;

v\_customers(3).balance := 8000;

v\_customers(3).is\_vip := 'FALSE';

FOR i IN 1..v\_customers.COUNT LOOP

IF v\_customers(i).balance > v\_vip\_threshold THEN

v\_customers(i).is\_vip := 'TRUE';

DBMS\_OUTPUT.PUT\_LINE('Promoted customer ' || v\_customers(i).customer\_id ||

' to VIP (Balance: $' || v\_customers(i).balance || ')');

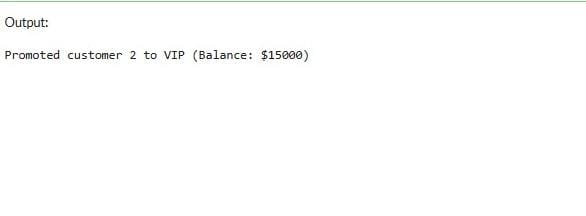
END IF;

END LOOP;

END;

/

Output:



Scenario 3: Loan reminders

Code:

DECLARE

TYPE loan\_type IS RECORD (

loan\_id NUMBER,

due\_date DATE,

customer\_name VARCHAR2(100),

email VARCHAR2(100)

);

TYPE loan\_table IS TABLE OF loan\_type INDEX BY BINARY\_INTEGER;

v\_loans loan\_table;

v\_today DATE := SYSDATE;

BEGIN

v\_loans(1).loan\_id := 101;

v\_loans(1).due\_date := v\_today + 15;

v\_loans(1).customer\_name := 'John Smith';

v\_loans(1).email := 'john@example.com';

v\_loans(2).loan\_id := 102;

v\_loans(2).due\_date := v\_today + 45;

v\_loans(2).customer\_name := 'Mary Johnson';

v\_loans(2).email := 'mary@example.com';

v\_loans(3).loan\_id := 103;

v\_loans(3).due\_date := v\_today + 10;

v\_loans(3).customer\_name := 'Robert Brown';

v\_loans(3).email := 'robert@example.com';

DBMS\_OUTPUT.PUT\_LINE('Sending loan due reminders...');

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

FOR i IN 1..v\_loans.COUNT LOOP

IF v\_loans(i).due\_date BETWEEN v\_today AND v\_today + 30 THEN

DBMS\_OUTPUT.PUT\_LINE('Dear ' || v\_loans(i).customer\_name || ',');

DBMS\_OUTPUT.PUT\_LINE('Your loan (ID: ' || v\_loans(i).loan\_id || ') is due in ' ||

(v\_loans(i).due\_date - v\_today) || ' days.');

DBMS\_OUTPUT.PUT\_LINE('Email: ' || v\_loans(i).email);

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

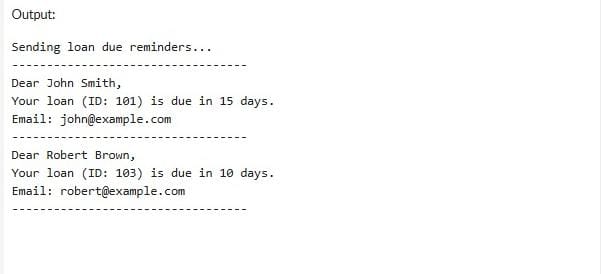
END IF;

END LOOP;

END;

/

Output:



**Exercise 3: Stored Procedures**

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

Code :

-- Drop and create Accounts table

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

AccountType VARCHAR2(20),

Balance NUMBER(10,2)

);

INSERT INTO Accounts VALUES (1, 'Savings', 1000.00);

INSERT INTO Accounts VALUES (2, 'Savings', 2000.00);

INSERT INTO Accounts VALUES (3, 'Checking', 3000.00);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

END;

/

BEGIN

ProcessMonthlyInterest;

END;

/

SELECT \* FROM Accounts;

Output:



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

Code:

-- Drop and create Employees table

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

EmpID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Department VARCHAR2(50),

Salary NUMBER(10,2)

);

INSERT INTO Employees VALUES (1, 'Alice', 'HR', 50000.00);

INSERT INTO Employees VALUES (2, 'Bob', 'IT', 60000.00);

INSERT INTO Employees VALUES (3, 'Charlie', 'IT', 70000.00);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept IN VARCHAR2,

bonusPercent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonusPercent / 100)

WHERE Department = dept;

END;

/

BEGIN

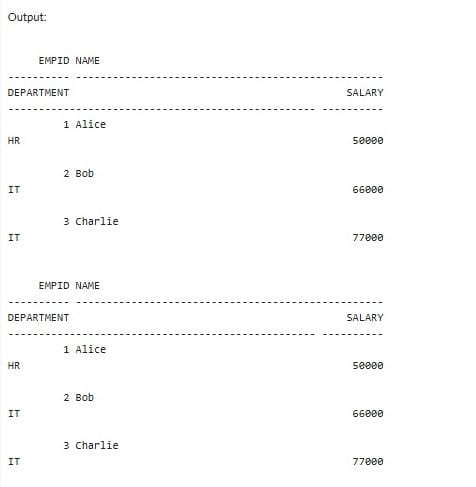
UpdateEmployeeBonus('IT', 10);

END;

/

SELECT \* FROM Employees;

Output:



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

Code:

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE BankAccounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE BankAccounts (

AccountNumber NUMBER PRIMARY KEY,

CustomerName VARCHAR2(50),

Balance NUMBER(10,2)

);

INSERT INTO BankAccounts VALUES (101, 'John Doe', 5000.00);

INSERT INTO BankAccounts VALUES (102, 'Jane Smith', 3000.00);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

sourceAcc IN NUMBER,

destAcc IN NUMBER,

amount IN NUMBER

) IS

sourceBalance NUMBER;

BEGIN

SELECT Balance INTO sourceBalance

FROM BankAccounts

WHERE AccountNumber = sourceAcc;

IF sourceBalance >= amount THEN

UPDATE BankAccounts

SET Balance = Balance - amount

WHERE AccountNumber = sourceAcc;

UPDATE BankAccounts

SET Balance = Balance + amount

WHERE AccountNumber = destAcc;

ELSE

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

END IF;

END;

/

BEGIN

TransferFunds(101, 102, 1000);

END;

/

SELECT \* FROM BankAccounts;

Output:



**(Non – Mandatory exercises)**

**Exercise 2: Error Handling**

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

Code :

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE BankAccounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE BankAccounts (

AccountID NUMBER PRIMARY KEY,

Balance NUMBER

);

INSERT INTO BankAccounts VALUES (1, 5000);

INSERT INTO BankAccounts VALUES (2, 2000);

COMMIT;

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

fromAcc IN NUMBER,

toAcc IN NUMBER,

amt IN NUMBER

) IS

BEGIN

DECLARE

curr\_balance NUMBER;

BEGIN

SELECT Balance INTO curr\_balance FROM BankAccounts WHERE AccountID = fromAcc;

IF curr\_balance < amt THEN

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

END IF;

UPDATE BankAccounts SET Balance = Balance - amt WHERE AccountID = fromAcc;

UPDATE BankAccounts SET Balance = Balance + amt WHERE AccountID = toAcc;

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

END;

/

BEGIN

SafeTransferFunds(1, 2, 1000);

SafeTransferFunds(1, 2, 10000);

END;

/

Output:



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

Code:

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

EmpID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Salary NUMBER

);

INSERT INTO Employees VALUES (101, 'Alice', 30000);

INSERT INTO Employees VALUES (102, 'Bob', 40000);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateSalary (

emp\_id IN NUMBER,

percent IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE EmpID = emp\_id;

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;

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

END;

/

BEGIN

UpdateSalary(101, 10);

UpdateSalary(999, 5);

END;

/

Output:



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

Code:

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100)

);

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

COMMIT;

CREATE OR REPLACE PROCEDURE AddNewCustomer (

cid IN NUMBER,

cname IN VARCHAR2

) IS

BEGIN

INSERT INTO Customers VALUES (cid, cname);

COMMIT;

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END;

/

BEGIN

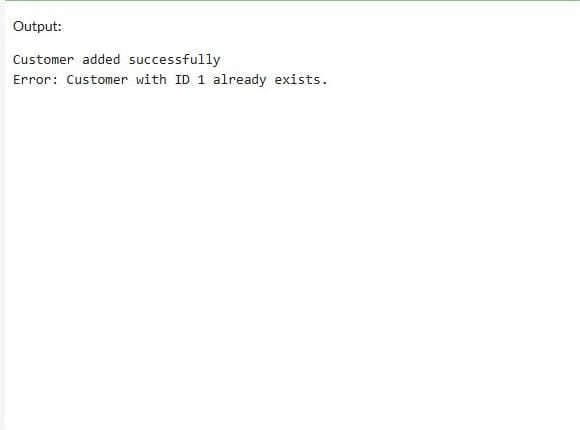
AddNewCustomer(2, 'Krishna');

AddNewCustomer(1, 'Duplicate');

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 Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

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 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'));

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(custID IN NUMBER, name IN VARCHAR2, dob IN DATE, balance IN NUMBER);

PROCEDURE UpdateCustomer(custID IN NUMBER, name IN VARCHAR2, dob IN DATE);

FUNCTION GetCustomerBalance(custID IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

PROCEDURE AddCustomer(custID IN NUMBER, name IN VARCHAR2, dob IN DATE, balance IN NUMBER) IS

BEGIN

INSERT INTO Customers VALUES (custID, name, dob, balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

END;

PROCEDURE UpdateCustomer(custID IN NUMBER, name IN VARCHAR2, dob IN DATE) IS

BEGIN

UPDATE Customers SET Name = name, DOB = dob, LastModified = SYSDATE

WHERE CustomerID = custID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Customer not found');

ELSE

COMMIT;

END IF;

END;

FUNCTION GetCustomerBalance(custID IN NUMBER) RETURN NUMBER IS

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Customers WHERE CustomerID = custID;

RETURN bal;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(empID IN NUMBER, name IN VARCHAR2, position IN VARCHAR2,

salary IN NUMBER, dept IN VARCHAR2, hireDate IN DATE);

PROCEDURE UpdateEmployee(empID IN NUMBER, position IN VARCHAR2, salary IN NUMBER);

FUNCTION AnnualSalary(empID IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(empID IN NUMBER, name IN VARCHAR2, position IN VARCHAR2,

salary IN NUMBER, dept IN VARCHAR2, hireDate IN DATE) IS

BEGIN

INSERT INTO Employees VALUES (empID, name, position, salary, dept, hireDate);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

END;

PROCEDURE UpdateEmployee(empID IN NUMBER, position IN VARCHAR2, salary IN NUMBER) IS

BEGIN

UPDATE Employees SET Position = position, Salary = salary WHERE EmployeeID = empID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Employee not found');

ELSE

COMMIT;

END IF;

END;

FUNCTION AnnualSalary(empID IN NUMBER) RETURN NUMBER IS

monthly NUMBER;

BEGIN

SELECT Salary INTO monthly FROM Employees WHERE EmployeeID = empID;

RETURN monthly \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(accID IN NUMBER, custID IN NUMBER, accType IN VARCHAR2, balance IN NUMBER);

PROCEDURE CloseAccount(accID IN NUMBER);

FUNCTION GetTotalBalance(custID IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(accID IN NUMBER, custID IN NUMBER, accType IN VARCHAR2, balance IN NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (accID, custID, accType, balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

END;

PROCEDURE CloseAccount(accID IN NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = accID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Account not found');

ELSE

COMMIT;

END IF;

END;

FUNCTION GetTotalBalance(custID IN NUMBER) RETURN NUMBER IS

total NUMBER;

BEGIN

SELECT SUM(Balance) INTO total FROM Accounts WHERE CustomerID = custID;

RETURN total;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END;

END AccountOperations;

/

BEGIN

CustomerManagement.AddCustomer(3, 'Ravi', TO\_DATE('1992-11-11', 'YYYY-MM-DD'), 2500);

EmployeeManagement.HireEmployee(3, 'Charlie', 'Analyst', 50000, 'Finance', SYSDATE);

AccountOperations.OpenAccount(3, 3, 'Savings', 2500);

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

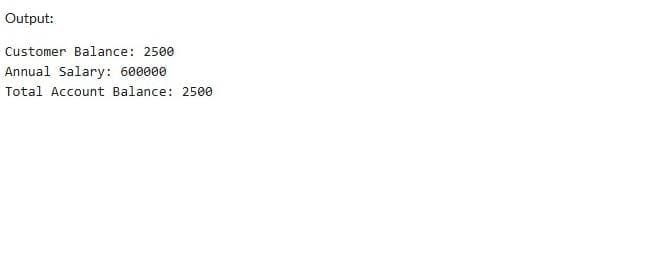
DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || EmployeeManagement.AnnualSalary(3));

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

END;

/

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:

-- Enable DBMS Output

SET SERVEROUTPUT ON;

-------------------------------

-- Scenario 1: CalculateAge

-------------------------------

CREATE OR REPLACE FUNCTION CalculateAge(dob DATE)

RETURN NUMBER

IS

age NUMBER;

BEGIN

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

RETURN age;

END;

/

-- Test Scenario 1

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Scenario 1 - Age is: ' || CalculateAge(DATE '1990-06-15'));

END;

/

-------------------------------

-- Scenario 2: CalculateMonthlyInstallment

-------------------------------

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

loan\_amount NUMBER,

annual\_interest\_rate NUMBER,

duration\_years NUMBER

)

RETURN NUMBER

IS

monthly\_interest\_rate NUMBER;

months NUMBER;

installment NUMBER;

BEGIN

monthly\_interest\_rate := annual\_interest\_rate / 12 / 100;

months := duration\_years \* 12;

IF monthly\_interest\_rate = 0 THEN

installment := loan\_amount / months;

ELSE

installment := loan\_amount \* monthly\_interest\_rate /

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

END IF;

RETURN ROUND(installment, 2);

END;

/

-- Test Scenario 2

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Scenario 2 - Monthly Installment: ' ||

CalculateMonthlyInstallment(100000, 10, 5));

END;

/

-------------------------------

-- Scenario 3: HasSufficientBalance

-------------------------------

-- Create a sample table

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

account\_id NUMBER PRIMARY KEY,

balance NUMBER

);

-- Insert test data

INSERT INTO Accounts VALUES (101, 5000);

INSERT INTO Accounts VALUES (102, 15000);

COMMIT;

-- Create Function

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_account\_id NUMBER, p\_amount NUMBER)

RETURN BOOLEAN

IS

acc\_balance NUMBER;

BEGIN

SELECT balance INTO acc\_balance FROM Accounts WHERE account\_id = p\_account\_id;

IF acc\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

-- Test Scenario 3

DECLARE

result BOOLEAN;

BEGIN

result := HasSufficientBalance(101, 3000);

IF result THEN

DBMS\_OUTPUT.PUT\_LINE('Scenario 3 - Sufficient Balance');

ELSE

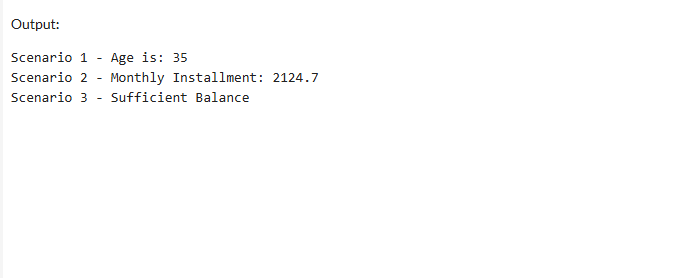
DBMS\_OUTPUT.PUT\_LINE('Scenario 3 - Insufficient Balance');

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:

SET SERVEROUTPUT ON;

----------------------------------------------------------

-- Scenario 1: UpdateCustomerLastModified Trigger

----------------------------------------------------------

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

email VARCHAR2(50),

last\_modified DATE

);

INSERT INTO Customers VALUES (1, 'Ravi', 'ravi@example.com', SYSDATE);

COMMIT;

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.last\_modified := SYSDATE;

END;

/

BEGIN

UPDATE Customers SET email = 'ravi.updated@example.com' WHERE customer\_id = 1;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Scenario 1 - Last Modified:');

FOR c IN (SELECT \* FROM Customers) LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || c.name || ', Last Modified: ' || TO\_CHAR(c.last\_modified, 'DD-Mon-YYYY HH24:MI'));

END LOOP;

END;

/

----------------------------------------------------------

-- Scenario 2: LogTransaction Trigger + AuditLog

----------------------------------------------------------

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions\_Log';

EXECUTE IMMEDIATE 'DROP TABLE AuditLog';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Transactions\_Log (

txn\_id NUMBER PRIMARY KEY,

account\_id NUMBER,

txn\_type VARCHAR2(10),

amount NUMBER

);

CREATE TABLE AuditLog (

audit\_id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

txn\_id NUMBER,

action\_date DATE,

message VARCHAR2(100)

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions\_Log

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (txn\_id, action\_date, message)

VALUES (:NEW.txn\_id, SYSDATE, 'Transaction inserted of type: ' || :NEW.txn\_type);

END;

/

INSERT INTO Transactions\_Log VALUES (1001, 101, 'Deposit', 3000);

COMMIT;

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Scenario 2 - Audit Log:');

FOR r IN (SELECT \* FROM AuditLog) LOOP

DBMS\_OUTPUT.PUT\_LINE('Txn ID: ' || r.txn\_id || ', Message: ' || r.message);

END LOOP;

END;

/

----------------------------------------------------------

-- Scenario 3: CheckTransactionRules Trigger

----------------------------------------------------------

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions\_Business';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

account\_id NUMBER PRIMARY KEY,

balance NUMBER

);

CREATE TABLE Transactions\_Business (

txn\_id NUMBER PRIMARY KEY,

account\_id NUMBER,

txn\_type VARCHAR2(10),

amount NUMBER

);

INSERT INTO Accounts VALUES (101, 5000);

COMMIT;

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions\_Business

FOR EACH ROW

DECLARE

acc\_balance NUMBER;

BEGIN

SELECT balance INTO acc\_balance FROM Accounts WHERE account\_id = :NEW.account\_id;

IF :NEW.txn\_type = 'Withdrawal' AND :NEW.amount > acc\_balance THEN

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

ELSIF :NEW.txn\_type = 'Deposit' AND :NEW.amount <= 0 THEN

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

END IF;

END;

/

INSERT INTO Transactions\_Business VALUES (2001, 101, 'Withdrawal', 1000);

INSERT INTO Transactions\_Business VALUES (2002, 101, 'Deposit', 3000);

BEGIN

INSERT INTO Transactions\_Business VALUES (2003, 101, 'Withdrawal', 10000);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Scenario 3 - Error: ' || SUBSTR(SQLERRM, INSTR(SQLERRM, ':') + 2));

END;

/

BEGIN

INSERT INTO Transactions\_Business VALUES (2004, 101, 'Deposit', -500);

EXCEPTION

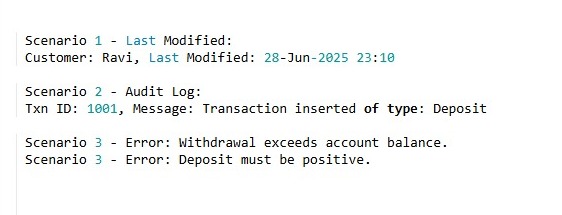
WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Scenario 3 - Error: ' || SUBSTR(SQLERRM, INSTR(SQLERRM, ':') + 2));

END;

/

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:

SET SERVEROUTPUT ON;

------------------------------------------------------------

-- Scenario 1: GenerateMonthlyStatements (Explicit Cursor)

------------------------------------------------------------

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(50)

);

CREATE TABLE Transactions (

txn\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

txn\_date DATE,

txn\_type VARCHAR2(10),

amount NUMBER

);

INSERT INTO Customers VALUES (1, 'Ravi');

INSERT INTO Customers VALUES (2, 'Priya');

INSERT INTO Transactions VALUES (1001, 1, SYSDATE, 'Deposit', 5000);

INSERT INTO Transactions VALUES (1002, 1, SYSDATE, 'Withdrawal', 2000);

INSERT INTO Transactions VALUES (1003, 2, ADD\_MONTHS(SYSDATE, -1), 'Deposit', 3000);

INSERT INTO Transactions VALUES (1004, 2, SYSDATE, 'Deposit', 4000);

COMMIT;

-- Cursor block

DECLARE

CURSOR txn\_cursor IS

SELECT c.name, t.txn\_date, t.txn\_type, t.amount

FROM Customers c

JOIN Transactions t ON c.customer\_id = t.customer\_id

WHERE TO\_CHAR(t.txn\_date, 'MMYYYY') = TO\_CHAR(SYSDATE, 'MMYYYY');

v\_name Customers.name%TYPE;

v\_date DATE;

v\_type VARCHAR2(10);

v\_amount NUMBER;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Scenario 1 - Monthly Statements:');

OPEN txn\_cursor;

LOOP

FETCH txn\_cursor INTO v\_name, v\_date, v\_type, v\_amount;

EXIT WHEN txn\_cursor%NOTFOUND;

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

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

END LOOP;

CLOSE txn\_cursor;

END;

/

------------------------------------------------------------

-- Scenario 2: ApplyAnnualFee (Explicit Cursor)

------------------------------------------------------------

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

account\_id NUMBER PRIMARY KEY,

balance NUMBER

);

INSERT INTO Accounts VALUES (101, 7000);

INSERT INTO Accounts VALUES (102, 15000);

COMMIT;

DECLARE

CURSOR acc\_cursor IS

SELECT account\_id, balance FROM Accounts;

v\_acc\_id Accounts.account\_id%TYPE;

v\_balance Accounts.balance%TYPE;

v\_fee CONSTANT NUMBER := 250;

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Scenario 2 - Applying Annual Fee:');

OPEN acc\_cursor;

LOOP

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

EXIT WHEN acc\_cursor%NOTFOUND;

UPDATE Accounts

SET balance = balance - v\_fee

WHERE account\_id = v\_acc\_id;

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || v\_acc\_id || ', Fee Deducted: ' || v\_fee);

END LOOP;

CLOSE acc\_cursor;

COMMIT;

END;

/

------------------------------------------------------------

-- Scenario 3: UpdateLoanInterestRates (Explicit Cursor)

------------------------------------------------------------

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

interest\_rate NUMBER

);

INSERT INTO Loans VALUES (301, 1, 8.5);

INSERT INTO Loans VALUES (302, 2, 9.5);

COMMIT;

DECLARE

CURSOR loan\_cursor IS

SELECT loan\_id, interest\_rate FROM Loans;

v\_loan\_id Loans.loan\_id%TYPE;

v\_old\_rate Loans.interest\_rate%TYPE;

v\_new\_rate NUMBER;

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Scenario 3 - Updating Loan Interest Rates:');

OPEN loan\_cursor;

LOOP

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

EXIT WHEN loan\_cursor%NOTFOUND;

-- Policy: If interest >= 9%, reduce by 0.5%

IF v\_old\_rate >= 9 THEN

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

ELSE

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

END IF;

UPDATE Loans

SET interest\_rate = v\_new\_rate

WHERE loan\_id = v\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || v\_loan\_id || ', Old Rate: ' || v\_old\_rate ||

', New Rate: ' || v\_new\_rate);

END LOOP;

CLOSE loan\_cursor;

COMMIT;

END;

/

Output:

