**WEEK\_2**

**PL/SQL Control Structures & Stored Procedures**

**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.

Program: CREATE TABLE CUSTOMERS (

CUSTOMER\_ID NUMBER PRIMARY KEY,

AGE NUMBER,

LOAN\_INTEREST NUMBER

);

SELECT table\_name FROM user\_tables WHERE table\_name = 'CUSTOMERS';

DESC CUSTOMERS;

DROP TABLE CUSTOMERS;

CREATE TABLE CUSTOMERS (

CUSTOMER\_ID NUMBER PRIMARY KEY,

AGE NUMBER,

LOAN\_INTEREST NUMBER

);

INSERT INTO CUSTOMERS VALUES (1, 65, 9.5);

INSERT INTO CUSTOMERS VALUES (2, 45, 10.0);

INSERT INTO CUSTOMERS VALUES (3, 70, 8.0);

INSERT INTO CUSTOMERS VALUES (4, 61, 7.5);

COMMIT;

SET SERVEROUTPUT ON;

DECLARE

CURSOR cust\_cursor IS

SELECT CUSTOMER\_ID, AGE, LOAN\_INTEREST FROM CUSTOMERS;

BEGIN

FOR cust\_rec IN cust\_cursor LOOP

IF cust\_rec.AGE > 60 THEN

UPDATE CUSTOMERS

SET LOAN\_INTEREST = cust\_rec.LOAN\_INTEREST - 1

WHERE CUSTOMER\_ID = cust\_rec.CUSTOMER\_ID;

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to customer ID: ' || cust\_rec.CUSTOMER\_ID);

END IF;

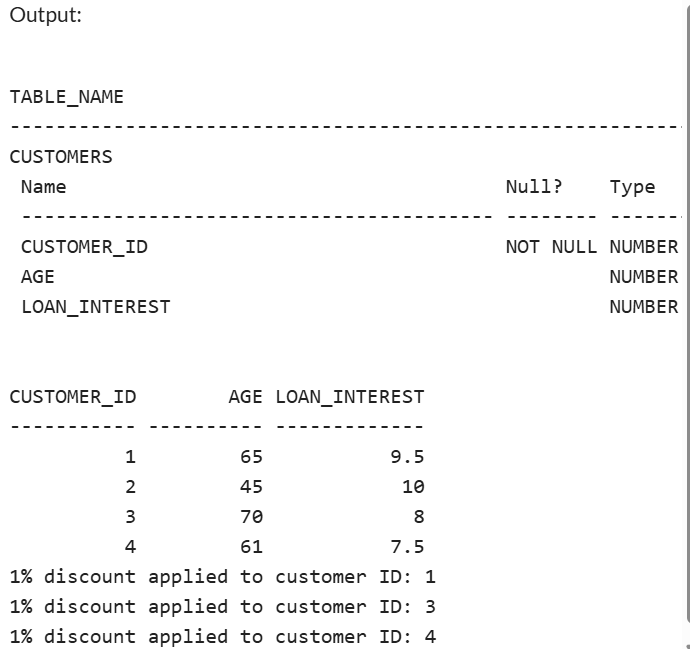
END LOOP;

COMMIT;

END;

/

**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.

**Program:**

CREATE TABLE CUSTOMERS (

CUSTOMER\_ID NUMBER PRIMARY KEY,

BALANCE NUMBER(10, 2),

ISVIP VARCHAR2(5) DEFAULT 'FALSE'

);

INSERT INTO CUSTOMERS (CUSTOMER\_ID, BALANCE) VALUES (1, 9500);

INSERT INTO CUSTOMERS (CUSTOMER\_ID, BALANCE) VALUES (2, 15000);

INSERT INTO CUSTOMERS (CUSTOMER\_ID, BALANCE) VALUES (3, 3000);

INSERT INTO CUSTOMERS (CUSTOMER\_ID, BALANCE) VALUES (4, 25000);

INSERT INTO CUSTOMERS (CUSTOMER\_ID, BALANCE) VALUES (5, 10500);

COMMIT;

SET SERVEROUTPUT ON;

BEGIN

FOR cust IN (SELECT CUSTOMER\_ID, BALANCE FROM CUSTOMERS) LOOP

IF cust.BALANCE > 10000 THEN

UPDATE CUSTOMERS

SET ISVIP = 'TRUE'

WHERE CUSTOMER\_ID = cust.CUSTOMER\_ID;

ELSE

UPDATE CUSTOMERS

SET ISVIP = 'FALSE'

WHERE CUSTOMER\_ID = cust.CUSTOMER\_ID;

END IF;

END LOOP;

COMMIT;

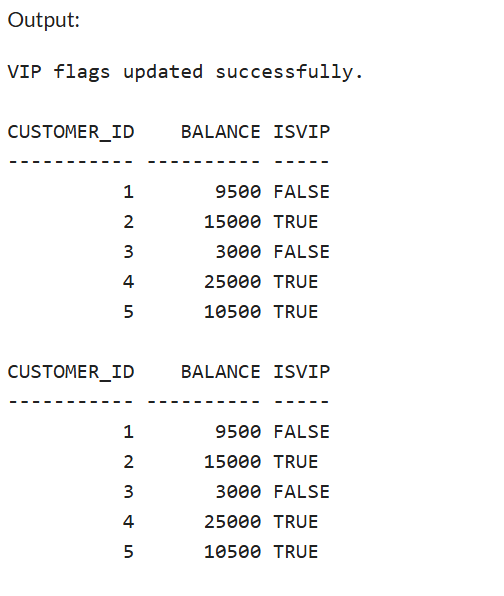
DBMS\_OUTPUT.PUT\_LINE('VIP flags updated successfully.');

END;

/

SELECT CUSTOMER\_ID, BALANCE, ISVIP 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.

Program:

SET SERVEROUTPUT ON;

SET DEFINE OFF;

-- Create the table

CREATE TABLE LOANS (

LOAN\_ID NUMBER PRIMARY KEY,

CUSTOMER\_ID NUMBER,

DUE\_DATE DATE,

AMOUNT NUMBER(10, 2)

);

-- Insert sample data

INSERT INTO LOANS VALUES (1, 101, TO\_DATE('2025-07-10', 'YYYY-MM-DD'), 5000);

INSERT INTO LOANS VALUES (2, 102, TO\_DATE('2025-08-15', 'YYYY-MM-DD'), 7000);

INSERT INTO LOANS VALUES (3, 103, TO\_DATE('2025-07-05', 'YYYY-MM-DD'), 8000);

INSERT INTO LOANS VALUES (4, 104, TO\_DATE('2025-06-20', 'YYYY-MM-DD'), 6000);

COMMIT;

SELECT \* FROM LOANS;

-- PL/SQL block to send reminders

BEGIN

FOR loan\_rec IN (

SELECT LOAN\_ID, CUSTOMER\_ID, DUE\_DATE, AMOUNT

FROM LOANS

WHERE DUE\_DATE BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Customer ' || loan\_rec.CUSTOMER\_ID ||

', your loan ID ' || loan\_rec.LOAN\_ID ||

' of amount $' || loan\_rec.AMOUNT ||

' is due on ' || TO\_CHAR(loan\_rec.DUE\_DATE, 'DD-Mon-YYYY')

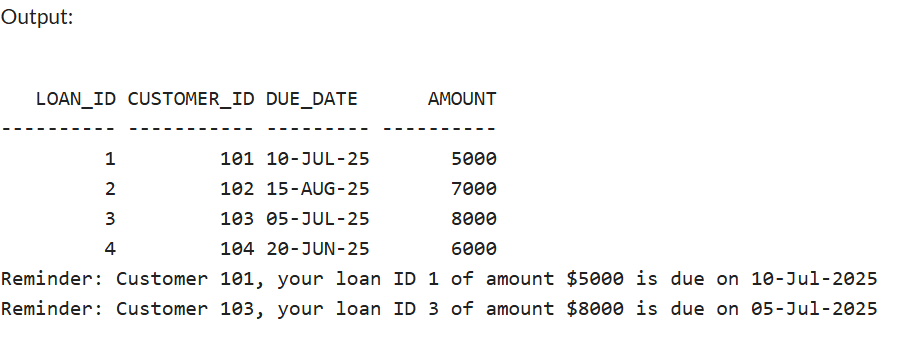
);

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.

PROGRAM:

-- Drop tables first if re-running

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Accounts table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Transactions table

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

-- Loans table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Employees table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- Sample Data Insertion

-- Customers

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);

-- Accounts

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

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

-- Transactions

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

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

-- Loans

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

-- Employees

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;

-- Procedure: ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

UPDATE Accounts

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

LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

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

END;

/

-- Enable DBMS output

SET SERVEROUTPUT ON;

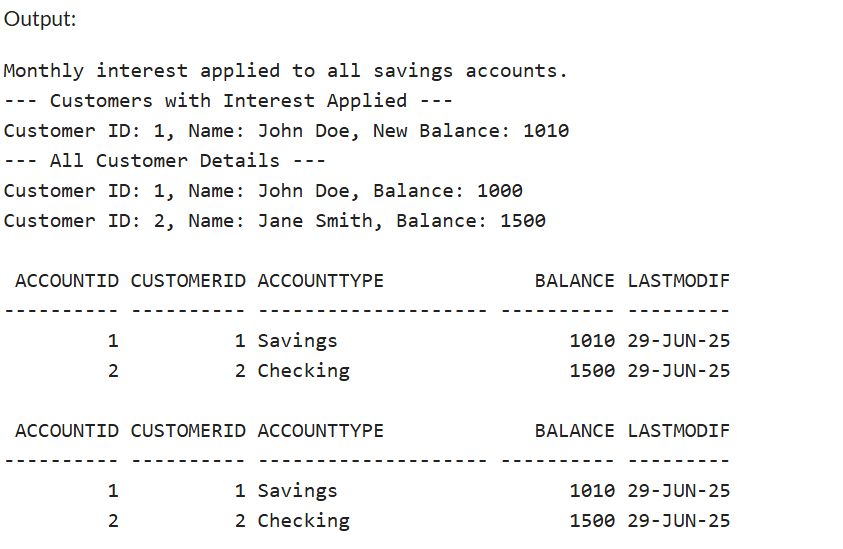
-- Execute the procedure

EXEC ProcessMonthlyInterest;

-- Check updated balances

SELECT \* FROM Accounts;

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.

**PROGRAM:**

-- Drop tables first if re-running

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Accounts table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Transactions table

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

-- Loans table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Employees table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- Sample Data Insertion

-- Customers

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);

-- Accounts

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

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

-- Transactions

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

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

-- Loans

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

-- Employees

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;

-- Procedure: ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

UPDATE Accounts

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

LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

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

DBMS\_OUTPUT.PUT\_LINE('--- Customers with Interest Applied ---');

FOR rec IN (

SELECT c.CustomerID, c.Name, a.Balance

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

WHERE a.AccountType = 'Savings'

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || rec.CustomerID || ', Name: ' || rec.Name || ', New Balance: ' || rec.Balance);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('--- All Customer Details ---');

FOR cust IN (

SELECT \* FROM Customers

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || cust.CustomerID || ', Name: ' || cust.Name || ', Balance: ' || cust.Balance);

END LOOP;

END;

/

-- Procedure: UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

)

IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_bonus\_percent || '% applied to employees in department: ' || p\_department);

DBMS\_OUTPUT.PUT\_LINE('--- Updated Employee Details ---');

FOR emp IN (

SELECT \* FROM Employees WHERE Department = p\_department

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || emp.EmployeeID || ', Name: ' || emp.Name || ', New Salary: ' || emp.Salary);

END LOOP;

END;

/

-- Enable DBMS output

SET SERVEROUTPUT ON;

-- Execute the procedures

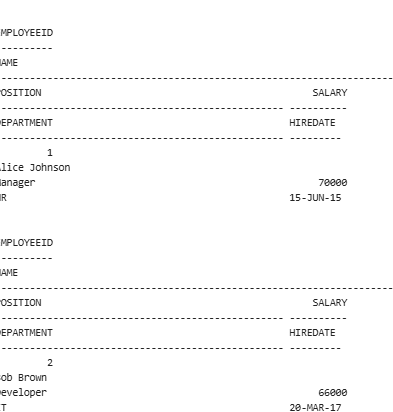
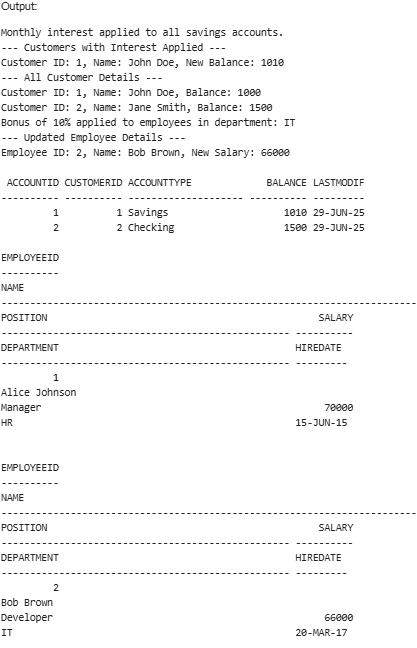
EXEC ProcessMonthlyInterest;

EXEC UpdateEmployeeBonus('IT', 10);

-- Check updated balances and salaries

SELECT \* FROM Accounts;

SELECT \* FROM Employees;



**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.

**PRogram:**

-- Drop tables first if re-running

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Accounts table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Transactions table

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

-- Loans table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Employees table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- Sample Data Insertion

-- Customers

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);

-- Accounts

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

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

-- Transactions

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

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

-- Loans

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

-- Employees

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;

-- Procedure: ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

UPDATE Accounts

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

LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

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

DBMS\_OUTPUT.PUT\_LINE('--- Customers with Interest Applied ---');

FOR rec IN (

SELECT c.CustomerID, c.Name, a.Balance

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

WHERE a.AccountType = 'Savings'

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || rec.CustomerID || ', Name: ' || rec.Name || ', New Balance: ' || rec.Balance);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('--- All Customer Details ---');

FOR cust IN (

SELECT \* FROM Customers

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || cust.CustomerID || ', Name: ' || cust.Name || ', Balance: ' || cust.Balance);

END LOOP;

END;

/

-- Procedure: UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

)

IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_bonus\_percent || '% applied to employees in department: ' || p\_department);

DBMS\_OUTPUT.PUT\_LINE('--- Updated Employee Details ---');

FOR emp IN (

SELECT \* FROM Employees WHERE Department = p\_department

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || emp.EmployeeID || ', Name: ' || emp.Name || ', New Salary: ' || emp.Salary);

END LOOP;

END;

/

-- Procedure: TransferFunds

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account FOR UPDATE;

IF v\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

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

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

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transferred $' || p\_amount || ' from account ' || p\_from\_account || ' to account ' || p\_to\_account);

END;

/

-- Enable DBMS output

SET SERVEROUTPUT ON;

-- Execute the procedures

EXEC ProcessMonthlyInterest;

EXEC UpdateEmployeeBonus('IT', 10);

EXEC TransferFunds(1, 2, 200);

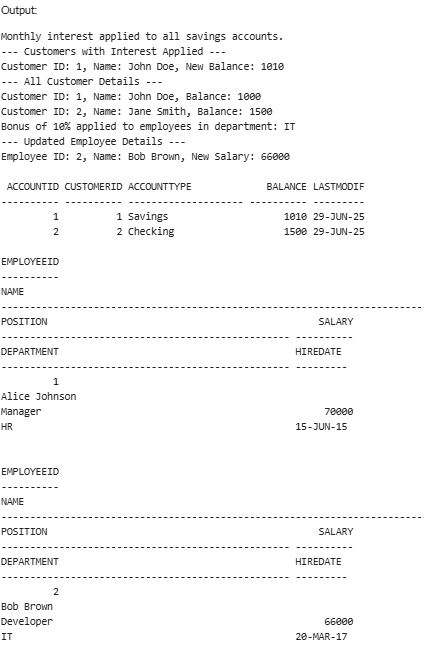
-- Check updated balances and salaries

SELECT \* FROM Accounts;

SELECT \* FROM Employees;

SELECT \* FROM Transactions;

OUTPUT

: 