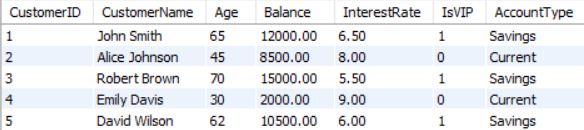
**Week2:**

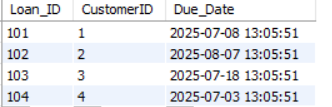
**Ex1: Control Structures**

Database setup:  
In order to implement the required scenarios, a database with Customers and Loans tables is required. So, I have taken two tables Customers and Loans using PL/SQL. I also inserted the values into both the tables.

Customers Table:



Loans Table:



**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.

Code:

BEGIN

    FOR cust IN (

        SELECT CustomerID, InterestRate

        FROM CUSTOMERS

        WHERE Age > 60

    ) LOOP

        UPDATE CUSTOMERS

        SET InterestRate = cust.InterestRate - 1

        WHERE CustomerID = cust.CustomerID;

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

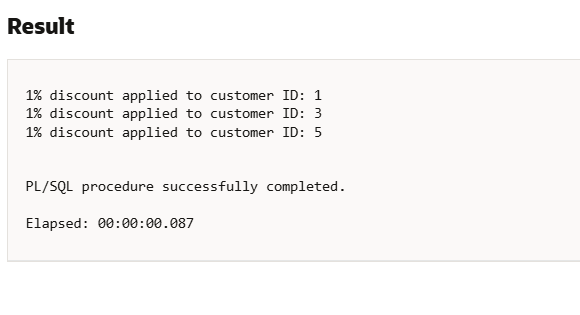
    END LOOP;

    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.

Code:  
BEGIN

FOR cust IN (

SELECT CustomerID, Balance

FROM CUSTOMERS

WHERE Balance > 10000

) LOOP

UPDATE CUSTOMERS

SET IsVIP = 1

WHERE CustomerID = cust.CustomerID;

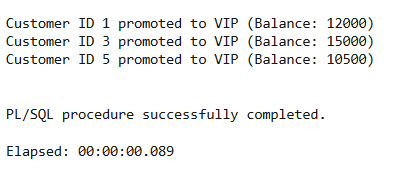
DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || cust.CustomerID || ' promoted to VIP (Balance: ' || cust.Balance || ')');

END LOOP;

COMMIT;

END;

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.

Code:

BEGIN

FOR loan\_rec IN (

SELECT l.Loan\_ID, l.Due\_Date, c.CustomerID, c.CustomerName

FROM LOANS l

JOIN CUSTOMERS c ON l.CustomerID = c.CustomerID

WHERE l.Due\_Date BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_rec.Loan\_ID ||

' for Customer "' || loan\_rec.CustomerName ||

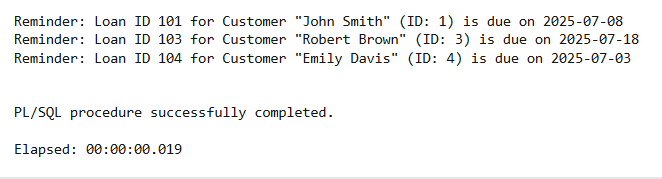
'" (ID: ' || loan\_rec.CustomerID ||

') is due on ' || TO\_CHAR(loan\_rec.Due\_Date, 'YYYY-MM-DD'));

END LOOP;

END;

Output:



**Ex3: 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.

**Code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (

SELECT CustomerID, Balance

FROM CUSTOMERS

WHERE AccountType = 'Savings'

) LOOP

UPDATE CUSTOMERS

SET Balance = Balance + (acc.Balance \* 0.01)

WHERE CustomerID = acc.CustomerID;

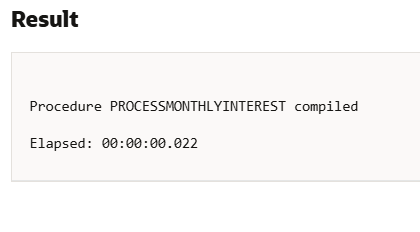
DBMS\_OUTPUT.PUT\_LINE('Interest applied to Savings Account - Customer ID: ' || acc.CustomerID);

END LOOP;

COMMIT;

END;

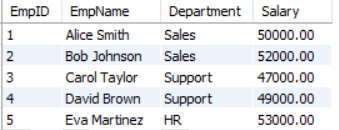
Output:



**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.

Let us take an Employees table for this for better understanding.



**Code:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    p\_department IN VARCHAR2,

    p\_bonus\_pct  IN NUMBER

)

IS

BEGIN

    UPDATE EMPLOYEES

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

    WHERE Department = p\_department;

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

    COMMIT;

END;

/

BEGIN

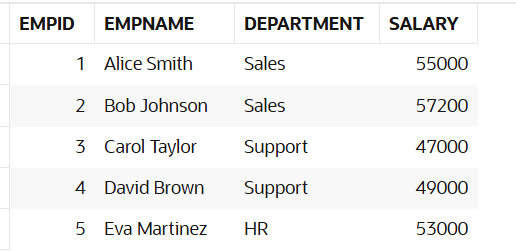
    UpdateEmployeeBonus('Sales', 10);

END;

/

After applying UpdateEmployeeBonus,the table becomes:

**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.

**Code:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_customer\_id IN NUMBER,

p\_to\_customer\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_from\_balance NUMBER;

BEGIN

-- Get sender's balance

SELECT Balance INTO v\_from\_balance

FROM CUSTOMERS

WHERE CustomerID = p\_from\_customer\_id;

-- Check for sufficient funds

IF v\_from\_balance < p\_amount THEN

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

END IF;

-- Deduct from sender

UPDATE CUSTOMERS

SET Balance = Balance - p\_amount

WHERE CustomerID = p\_from\_customer\_id;

-- Add to receiver

UPDATE CUSTOMERS

SET Balance = Balance + p\_amount

WHERE CustomerID = p\_to\_customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_amount ||

' from Customer ID ' || p\_from\_customer\_id ||

' to Customer ID ' || p\_to\_customer\_id);

COMMIT;

END;

/

-- Call the procedure to transfer funds

BEGIN

TransferFunds(1, 2, 2000); -- Transfer 2000 from John Smith to Alice Johnson

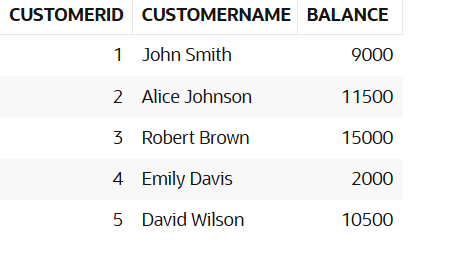
END;

/

-- View final balances

SELECT CustomerID, CustomerName, Balance FROM CUSTOMERS;

Output:



If there is insufficient balance to transfer, it shows this message:

ORA-20001: Insufficient balance for transfer.