**Week 2- PL/SQL PROGRAMS**

**Exercise 1: Control Structures**

**//Firstly created Customer Table**

CREATE TABLE customers (customer\_id NUMBER, name VARCHAR2(50), age NUMBER, balance NUMBER, loan\_interest\_rate NUMBER, IsVIP VARCHAR2(5));

**//Input entered in Customer Table**

INSERT INTO customers VALUES (1, 'Ravi', 45, 9000, 7.5, 'FALSE');

INSERT INTO customers VALUES (2, 'Meena', 61, 12000, 8.0, 'FALSE');

INSERT INTO customers VALUES (3, 'Rajesh', 70, 15000, 7.0, 'FALSE');

**//Secondly created Loans Table**

CREATE TABLE loans (loan\_id NUMBER, customer\_id NUMBER, due\_date DATE);

**//Input entered in Loans Table**

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

INSERT INTO loans VALUES (102, 2, SYSDATE + 20);

INSERT INTO loans VALUES (103, 3, SYSDATE + 50);

**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.**

BEGIN

FOR cust\_rec IN (SELECT customer\_id FROM customers WHERE age > 60) LOOP

UPDATE customers

SET loan\_interest\_rate = loan\_interest\_rate - 1

WHERE customer\_id = cust\_rec.customer\_id;

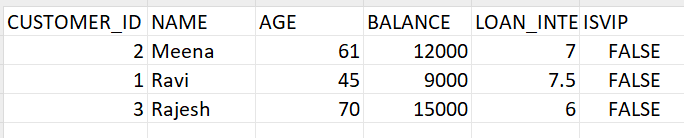
END LOOP;

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.**

BEGIN

FOR vip\_rec IN (SELECT customer\_id FROM customers WHERE balance > 10000) LOOP

UPDATE customers

SET IsVIP = 'TRUE'

WHERE customer\_id = vip\_rec.customer\_id;

END LOOP;

COMMIT;

END;

/

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

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

BEGIN

FOR loan\_rec IN (

SELECT l.loan\_id, c.name, l.due\_date

FROM loans l

JOIN customers c ON l.customer\_id = c.customer\_id

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || loan\_rec.name ||

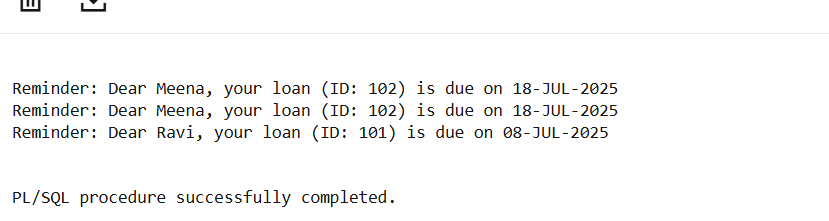
', your loan (ID: ' || loan\_rec.loan\_id ||

') is due on ' || TO\_CHAR(loan\_rec.due\_date, 'DD-MON-YYYY'));

END LOOP;

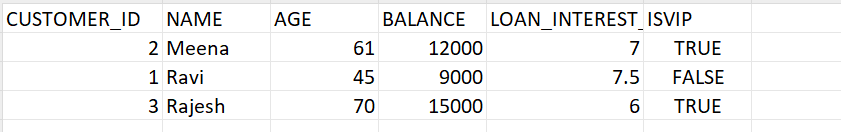
END;

/



**OUTPUT:**

Select \* from Customers;



**Exercise 3: Stored Procedures**

**//Firstly create accounts table**

CREATE TABLE accounts ( account\_id NUMBER PRIMARY KEY, account\_type VARCHAR2(20), customer\_name VARCHAR2(50), balance NUMBER);

**// Insert data in accounts**

INSERT INTO accounts VALUES (101, 'savings', 'Ravi', 10000);

INSERT INTO accounts VALUES (102, 'savings', 'Meena', 15000);

INSERT INTO accounts VALUES (103, 'current', 'Rajesh', 20000);

INSERT INTO accounts VALUES (104, 'savings', 'Preethi', 5000);

**-//Secondly create Employees table**

CREATE TABLE employees ( emp\_id NUMBER PRIMARY KEY, emp\_name VARCHAR2(50), department VARCHAR2(30), salary NUMBER

);

**// Insert data into employess**

INSERT INTO employees VALUES (1, 'John', 'Sales', 50000);

INSERT INTO employees VALUES (2, 'Alice', 'Sales', 55000);

INSERT INTO employees VALUES (3, 'Bob', 'HR', 48000);

**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.**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE accounts

SET balance = balance \* 1.01

WHERE account\_type = 'savings';

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

END;

/

**//Execute Procedure**

BEGIN

ProcessMonthlyInterest;

END;

/

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

**o 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.**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept\_name IN VARCHAR2,

bonus\_percent IN NUMBER

) AS

BEGIN

UPDATE employees

SET salary = salary + (salary \* bonus\_percent / 100)

WHERE department = dept\_name;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to department: ' || dept\_name);

END;

/

**//Execute Procedure: 10% bonus to 'Sales' dept**

BEGIN

UpdateEmployeeBonus('Sales', 10);

END;

/

**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.**

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_acct IN NUMBER,

to\_acct IN NUMBER,

amt IN NUMBER

) AS

insufficient\_balance EXCEPTION;

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance FROM accounts WHERE account\_id = from\_acct;

IF v\_balance < amt THEN

RAISE insufficient\_balance;

END IF;

UPDATE accounts

SET balance = balance - amt

WHERE account\_id = from\_acct;

UPDATE accounts

SET balance = balance + amt

WHERE account\_id = to\_acct;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ' || amt || ' transferred from ' || from\_acct || ' to ' || to\_acct);

EXCEPTION

WHEN insufficient\_balance THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient balance in account ' || from\_acct);

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: One of the account IDs is invalid.');

WHEN OTHERS THEN

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

END;

/

**//Execute a fund transfer: 3000 from account 101 to 103**

BEGIN

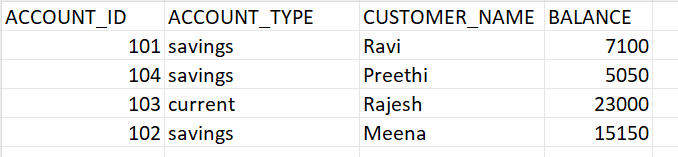
TransferFunds(101, 103, 3000);

END;

/

**Output:**

Select \* from Accounts;



Select \* from Employees;

