**PL/SQL PROGRAMMING**

**CREATION OF TABLES:**

--customer table creation

create table customers (

customer\_id number primary key,

name varchar2(100),

age number,

balance number,

is\_vip char(1) default 'N');

--loans table creation

create table loans(

loan\_id number primary key,

customer\_id number references customers(customer\_id),

loan\_amount number,

interest\_rate number(5,2),

due\_date date );

--accounts table creation

create table accounts (

account\_id number primary key,

customer\_id number references customers(customer\_id),

account\_type varchar2(20), -- e.g., 'Savings', 'Current'

balance number(10, 2)

);

--employees table creation

create table employees (

emp\_id number primary key,

name varchar2(100),

department\_id number,

salary number(10, 2),

performance\_rating number(2) -- optional if you want to filter on it

);

**INSERTION OF DATA IN TABLES:**

--insertion of data into customer table

insert into customers values (1, 'Ravi Kumar', 65, 8000, 'N');

insert into customers values (2, 'Meena Sharma', 45, 15000,'N');

insert into customers values (3, 'Suresh Yadav', 70, 12000, 'N');

insert into customers values (4, 'Anjali Verma', 59, 9500, 'N');

insert into customers values (5, 'Vijay Rao', 61, 11000, 'N');

--insertion of data into loans table

insert into loans values (101, 1, 500000, 8.50, SYSDATE + 10);

insert into loans values (102, 2, 300000, 9.00, SYSDATE + 45);

insert into loans values (103, 3, 750000, 8.00, SYSDATE + 5);

insert into loans values (104, 4, 200000, 9.25, SYSDATE - 1);

insert into loans values (105, 5, 1000000, 7.75, SYSDATE + 25);

--insertion of data into accounts table

insert into accounts (account\_id, customer\_id, account\_type, balance)

values (1001, 1, 'Savings', 10000.00);

insert into accounts (account\_id, customer\_id, account\_type, balance)

values (1002, 2, 'Savings', 25000.00);

insert into accounts (account\_id, customer\_id, account\_type, balance)

values (1003, 3, 'Current', 15000.00);

insert into accounts (account\_id, customer\_id, account\_type, balance)

values (1004, 4, 'Savings', 0.00);

insert into accounts (account\_id, customer\_id, account\_type, balance)

values (1005, 5, 'Current', 5000.00);

insert into accounts (account\_id, customer\_id, account\_type, balance)

values (1006, 2, 'Savings', 30000.00);

COMMIT;

--insertion of data into employees data

-- EMPLOYEES

insert into employees (emp\_id, name, department\_id, salary, performance\_rating)

values (1, 'Ravi Kumar', 101, 45000, 8);

insert into employees (emp\_id, name, department\_id, salary, performance\_rating)

values (2, 'Meena Sharma', 101, 52000, 9);

insert into employees (emp\_id, name, department\_id, salary, performance\_rating)

values (3, 'Suresh Yadav', 102, 60000, 7);

insert into employees (emp\_id, name, department\_id, salary, performance\_rating)

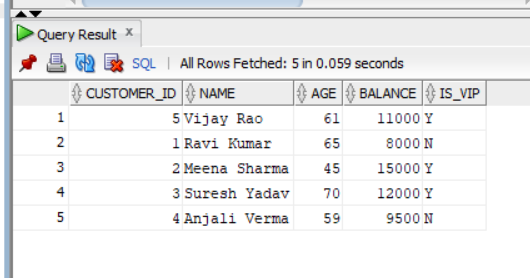
values (4, 'Anjali Verma', 103, 70000, 10);

insert into employees (emp\_id, name, department\_id, salary, performance\_rating)

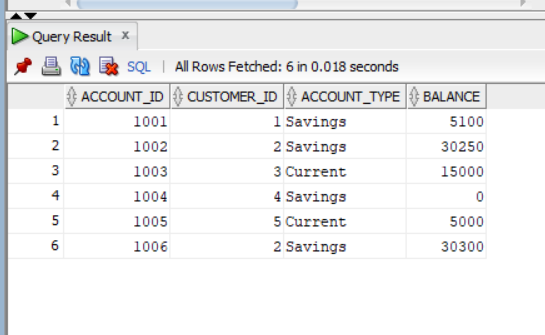
values (5, 'Vijay Rao', 101, 55000, 6);

COMMIT;

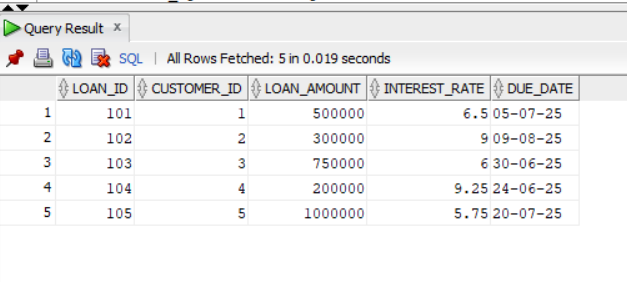
**Customer Table:**

****

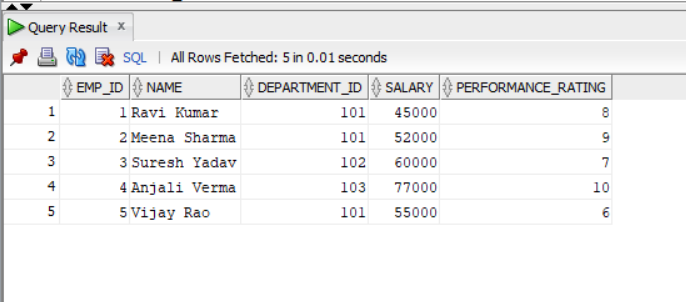
**Accounts Table:**

****

**Loans Table:**

****

**Employees Table:**

****

**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.

--Enable Output

SET SERVEROUTPUT ON;

BEGIN

FOR loan\_rec IN (SELECT loan\_id, customer\_id FROM loans) LOOP

DECLARE

cust\_age customers.age%TYPE;

BEGIN

SELECT age INTO cust\_age

FROM customers

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

IF cust\_age > 60 THEN

UPDATE loans

SET interest\_rate = interest\_rate - 1

WHERE loan\_id = loan\_rec.loan\_id;

DBMS\_OUTPUT.put\_line('Discount applied to Loan ID: ' || loan\_rec.loan\_id);

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.put\_line('Customer not found for Loan ID: ' || loan\_rec.loan\_id);

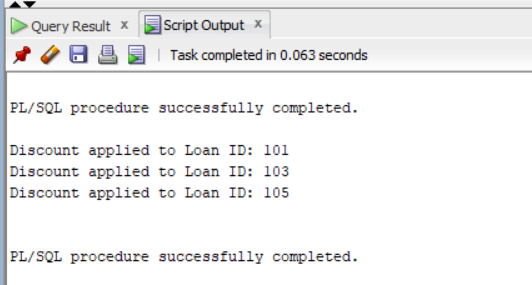
END;

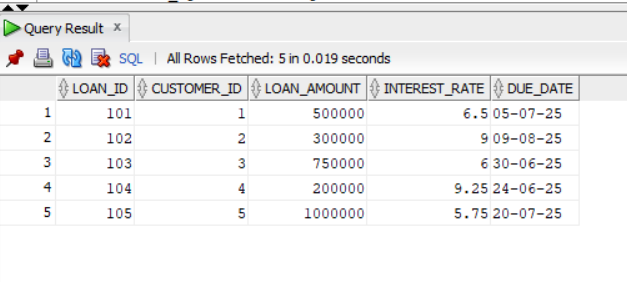
END LOOP;

COMMIT;

END;

/



****

**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 cust IN (SELECT customer\_id, balance FROM customers) LOOP

IF cust.balance > 10000 THEN

UPDATE customers

SET is\_vip = 'Y'

WHERE customer\_id = cust.customer\_id;

DBMS\_OUTPUT.put\_line('Customer ID ' || cust.customer\_id || ' promoted to VIP.');

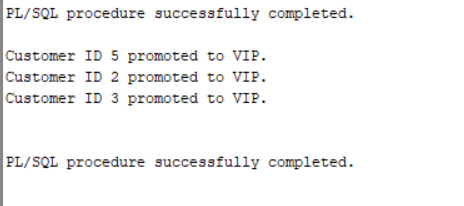
END IF;

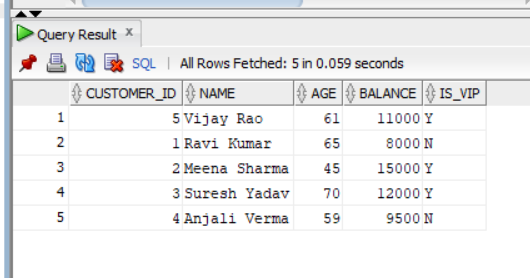
END LOOP;

COMMIT;

END;

/



****

**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.

BEGIN

FOR l IN (

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

FROM loans l

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

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

) LOOP

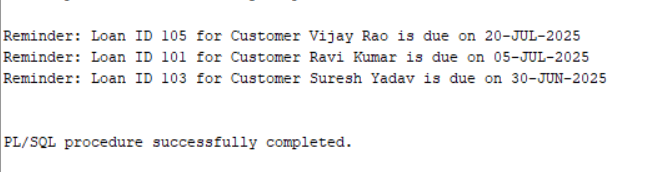
DBMS\_OUTPUT.put\_line('Reminder: Loan ID ' || l.loan\_id || ' for Customer ' || l.name ||

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

END LOOP;

END;

/



**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.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR acc IN (

SELECT account\_id, balance

FROM accounts

WHERE LOWER(account\_type) = 'savings'

) LOOP

UPDATE accounts

SET balance = balance + (balance \* 0.01)

WHERE account\_id = acc.account\_id;

DBMS\_OUTPUT.put\_line('Interest applied to Account ID: ' || acc.account\_id);

END LOOP;

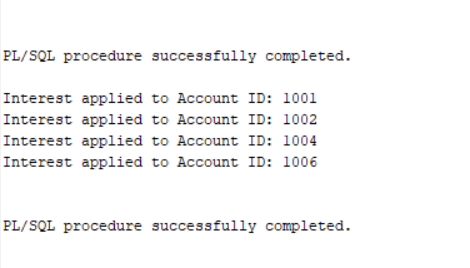
COMMIT;

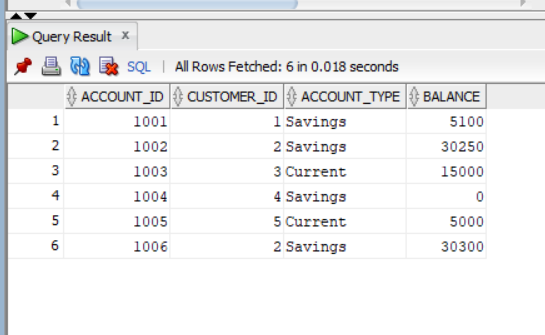
END;

/

**Calling procedure:**

EXEC ProcessMonthlyInterest;



****

**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.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept\_id NUMBER,

p\_bonus\_percent NUMBER -- e.g., pass 10 for 10%

) AS

BEGIN

FOR emp IN (

SELECT emp\_id, salary

FROM employees

WHERE department\_id = p\_dept\_id

) LOOP

UPDATE employees

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

WHERE emp\_id = emp.emp\_id;

DBMS\_OUTPUT.put\_line('Bonus applied to Emp ID: ' || emp.emp\_id);

END LOOP;

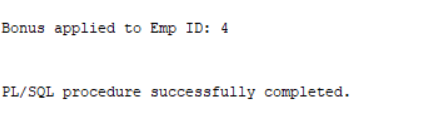
COMMIT;

END;

/

**Calling procedure:**

EXEC UpdateEmployeeBonus(103,10);



**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 (

p\_from\_acc\_id NUMBER,

p\_to\_acc\_id NUMBER,

p\_amount NUMBER

) AS

v\_balance NUMBER;

BEGIN

-- Get balance of source account

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = p\_from\_acc\_id

FOR UPDATE;

IF v\_balance < p\_amount THEN

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

END IF;

-- Deduct from source

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_acc\_id;

-- Add to destination

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_acc\_id;

DBMS\_OUTPUT.put\_line('Transferred ' || p\_amount || ' from ' || p\_from\_acc\_id || ' to ' || p\_to\_acc\_id);

COMMIT;

END;

/

**Calling procedure:**

exec TransferFunds(1001, 1002, 5000);

