**PL/SQL**

**Exercise-1: Control Structures**

**Creating Customers Table:**

CREATE TABLE Customers (

cid NUMBER PRIMARY KEY,

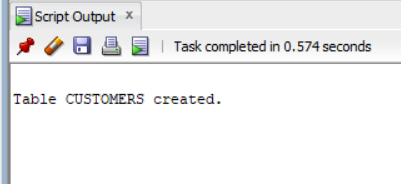
name VARCHAR2(50),

age NUMBER,

bal NUMBER(10,2),

isvip VARCHAR2(5) DEFAULT 'FALSE'

);



**Inserting values into table:**

INSERT INTO Customers (cid, name, age, bal) VALUES (1, 'Dev', 65, 15000.00);

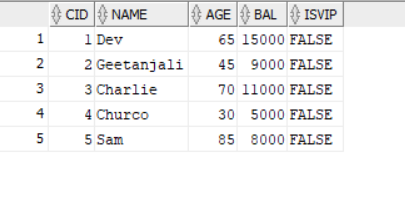
INSERT INTO Customers (cid, name, age, bal) VALUES (2, 'Geetanjali', 45, 9000.00);

INSERT INTO Customers (cid, name, age, bal) VALUES (3, 'Charlie', 70, 11000.00);

INSERT INTO Customers (cid, name, age, bal) VALUES (4, 'Churco', 30, 5000.00);

INSERT INTO Customers (cid, name, age, bal) VALUES (5, 'Sam', 85, 8000.00);

SELECT \* FROM Customers;



**Creating Loan table:**

CREATE TABLE loans (

lid NUMBER PRIMARY KEY,

cid NUMBER,

InterestRate NUMBER(5,2),

DueDate DATE,

FOREIGN KEY (cid) REFERENCES Customers(cid)

);

INSERT INTO loans (lid, cid, InterestRate, DueDate)

VALUES (101, 1, 7.5, SYSDATE + 15);

INSERT INTO loans (lid, cid, InterestRate, DueDate)

VALUES (102, 2, 8.0, SYSDATE + 45);

INSERT INTO loans (lid, cid, InterestRate, DueDate)

VALUES (103, 3, 6.5, SYSDATE + 5);

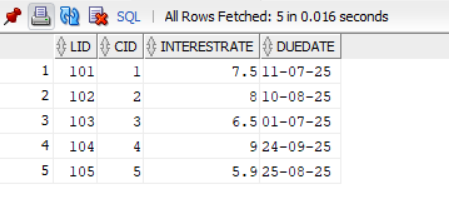
INSERT INTO loans (lid, cid, InterestRate, DueDate)

VALUES (104, 4, 9.0, SYSDATE + 90);

INSERT INTO loans (lid, cid, InterestRate, DueDate)

VALUES (105, 5, 5.9, SYSDATE + 60);

SELECT \* FROM loans;



**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 IN (

SELECT c.cid, c.name, c.age, l.lid, l.InterestRate

FROM Customers c

JOIN loans l ON c.cid = l.cid

WHERE c.age > 60

)

LOOP

UPDATE loans

SET InterestRate = InterestRate - 1

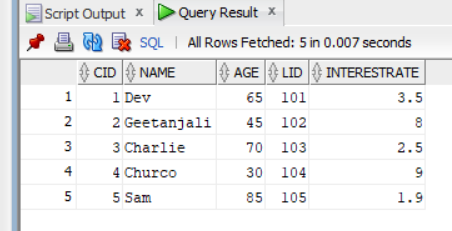
WHERE lid = cust.lid;

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 cust IN (SELECT cid, name, bal FROM Customers WHERE bal > 10000)

LOOP

UPDATE Customers

SET isvip = 'TRUE'

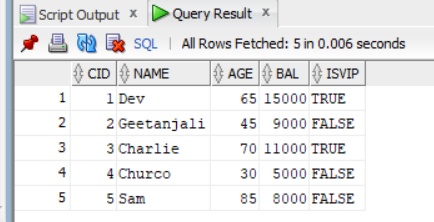
WHERE cid = cust.cid;

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.

BEGIN

FOR rec IN (

SELECT c.cid, c.Name, l.lid, l.DueDate

FROM loans l

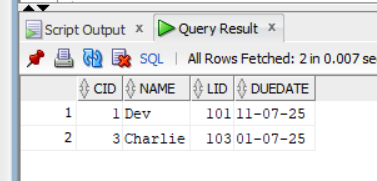
JOIN customers c ON c.cid = l.cid

WHERE l.DueDate <= SYSDATE + 30

)

END;

Output:



**Exercise-3: Stored Procedures**

**Creating tables for employees and accounts:**

CREATE TABLE accounts (

accId NUMBER PRIMARY KEY,

cid NUMBER,

AccountType VARCHAR2(20),

bal NUMBER(12,2)

);

CREATE TABLE employees (

eid NUMBER PRIMARY KEY,

name VARCHAR2(50),

department VARCHAR2(30),

sal NUMBER(10,2)

);

Inserting data intlo tables:

INSERT ALL

INTO accounts (accId, cid, AccountType, bal) VALUES (101, 1, 'Savings', 10000.00)

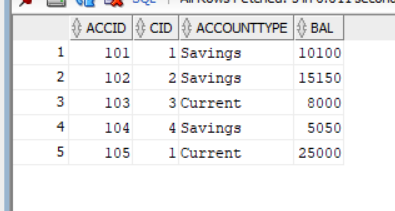
INTO accounts (accId, cid, AccountType, bal) VALUES (102, 2, 'Savings', 15000.00)

INTO accounts (accId, cid, AccountType, bal) VALUES (103, 3, 'Current', 8000.00)

INTO accounts (accId, cid, AccountType, bal) VALUES (104, 4, 'Savings', 5000.00)

INTO accounts (accId, cid, AccountType, bal) VALUES (105, 1, 'Current', 25000.00);

SELECT \* FROM dual;



INSERT ALL

INTO employees (eid, name, department, sal) VALUES (1, 'Alice', 'HR', 40000.00)

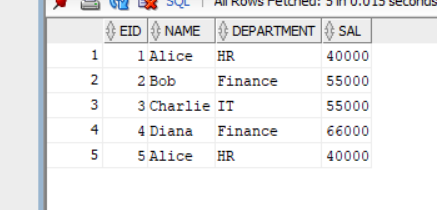
INTO employees (eid, name, department, sal) VALUES (2, 'Bob', 'Finance', 50000.00)

INTO employees (eid, name, department, sal) VALUES (3, 'Charlie', 'IT', 55000.00)

INTO employees (eid, name, department, sal) VALUES (4, 'Diana', 'Finance', 60000.00)

INTO employees (eid, name, department, sal) VALUES (5, 'Alice', 'HR', 40000.00);

SELECT \* FROM dual;



**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 MonthWiseIntrest IS

BEGIN

UPDATE accounts

SET bal = bal + (bal \* 0.01)

WHERE AccountType = 'Savings';

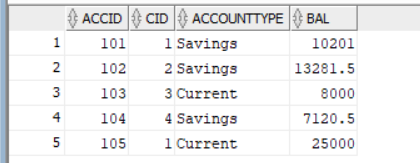
COMMIT;

END;

Exec MonthWiseIntrest;

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.

CREATE OR REPLACE PROCEDURE EmployeeBonusUpdating (

depName IN VARCHAR2,

bonusPercentage IN NUMBER

) AS

BEGIN

UPDATE employees

SET sal = sal + (sal \* bonusPercentage / 100)

WHERE department = depName;

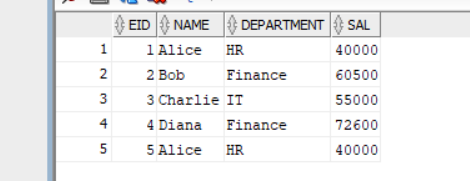
COMMIT;

END;

Exec EmployeeBonusUpdating('Finance', 10);

SELECT \* FROM employees;

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.

CREATE OR REPLACE PROCEDURE FundingTransfer (

FromAcc IN NUMBER,

ToAcc IN NUMBER,

amt IN NUMBER

) AS

bal\_insufficient EXCEPTION;

BEGIN

DECLARE

fromBal NUMBER;

BEGIN

SELECT bal INTO fromBal FROM accounts WHERE accId = FromAcc FOR UPDATE;

IF fromBal < amt THEN

RAISE bal\_insufficient;

END IF;

UPDATE accounts

SET bal = bal - amt

WHERE accId = FromAcc;

UPDATE accounts

SET bal = bal + amt

WHERE accId = ToAcc;

COMMIT;

END;

END;

Exec FundingTransfer(102, 104, 2000);

END;

SELECT \* FROM accounts;

Output:

