**WEEK – 02**

**PL/SQL programming**

**Superset ID: 6262264**

**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.

**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.

**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.

**Solution :**

**STEP – 1 : PostgreSQL Table Creation and Data Insertion Example :**

// Drop tables if they already exist

DROP TABLE IF EXISTS loans;

DROP TABLE IF EXISTS customers;

//Create Customers table

CREATE TABLE customers (

customerid SERIAL PRIMARY KEY,

name TEXT,

age INTEGER,

balance NUMERIC,

isvip TEXT DEFAULT 'N'

);

//Show Customers table structure without column\_default

SELECT

column\_name,

data\_type,

is\_nullable

FROM

information\_schema.columns

WHERE

table\_name = 'customers'

ORDER BY ordinal\_position;

//Create Loans table

CREATE TABLE loans (

loanid SERIAL PRIMARY KEY,

customerid INTEGER REFERENCES customers(customerid),

interestrate NUMERIC,

duedate DATE

);

-- Show Loans table structure without column\_default

SELECT

column\_name,

data\_type,

is\_nullable

FROM

information\_schema.columns

WHERE

table\_name = 'loans'

ORDER BY ordinal\_position;

-- Insert sample customers

INSERT INTO customers (name, age, balance) VALUES

('Alice', 65, 15000),

('Bob', 45, 8000),

('Charlie', 72, 12000),

('Diana', 30, 5000);

-- Insert sample loans

INSERT INTO loans (customerid, interestrate, duedate) VALUES

(1, 7.5, CURRENT\_DATE + INTERVAL '20 days'),

(2, 6.0, CURRENT\_DATE + INTERVAL '10 days'),

(3, 5.5, CURRENT\_DATE + INTERVAL '25 days'),

(4, 8.0, CURRENT\_DATE + INTERVAL '40 days');

**Scenario 1 : PL/SQL Block to Apply 1% Discount to Loans of Customers Over 60**

// Apply 1% discount to loan interest rates for customers above 60

DO $$

DECLARE

rec RECORD;

BEGIN

FOR rec IN

SELECT c.customerid, c.name, l.loanid, l.interestrate

FROM customers c

JOIN loans l ON c.customerid = l.customerid

WHERE c.age > 60

LOOP

UPDATE loans

SET interestrate = rec.interestrate - 1

WHERE loanid = rec.loanid;

RAISE INFO '%: 1%% discount offered.', rec.name;

END LOOP;

END;

$$ LANGUAGE plpgsql;

**Scenario 2: Promote customers with balance > 10,000 to VIP**

DO $$

DECLARE

rec RECORD;

BEGIN

FOR rec IN

SELECT customerid, name

FROM customers

WHERE balance > 10000

LOOP

UPDATE customers

SET isvip = 'Y'

WHERE customerid = rec.customerid;

RAISE INFO '%: Promoted to VIP.', rec.name;

END LOOP;

END;

$$ LANGUAGE plpgsql;

**Scenario 3 : Send reminders for loans due within the next 30 days**

DO $$

DECLARE

rec RECORD;

due\_date\_text TEXT;

BEGIN

FOR rec IN

SELECT l.loanid, c.name, l.duedate

FROM loans l

JOIN customers c ON c.customerid = l.customerid

WHERE l.duedate BETWEEN CURRENT\_DATE AND CURRENT\_DATE + INTERVAL '30 days'

LOOP

due\_date\_text := to\_char(rec.duedate, 'DD-Mon-YYYY');

RAISE INFO '%: Loan due on %.', rec.name, due\_date\_text;

END LOOP;

END;

$$ LANGUAGE plpgsql;

**// Show final Customers table contents**

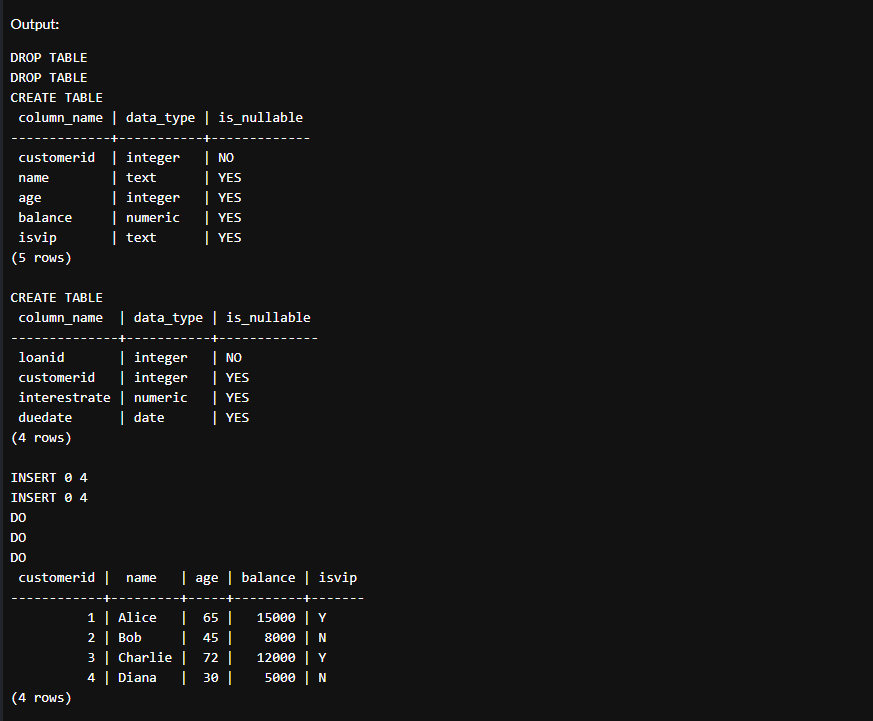
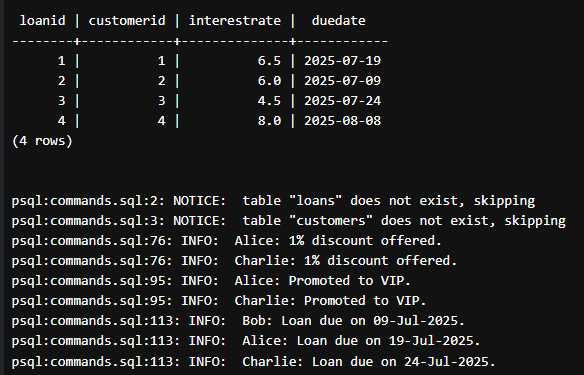
SELECT \* FROM customers ORDER BY customerid;

**// Show final Loans table contents**

SELECT \* FROM loans ORDER BY loanid;

**EXPLANATION :**

This PL/SQL script performs targeted data operations: it incentivizes senior customers by reducing their loan interest rates, elevates high-value clients to VIP status based on balance thresholds, and proactively issues loan due reminders. Through iterative record processing and conditional updates, it showcases dynamic business rule enforcement within a transactional database environment.

**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.

**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.

**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.

**Solution :**

**STEP –1 :** **Database Setup: Creating Tables and Inserting Sample Records**

//Drop tables if they already exist

DROP TABLE IF EXISTS savings\_accounts;

DROP TABLE IF EXISTS employees;

DROP TABLE IF EXISTS accounts;

//Create savings\_accounts table

CREATE TABLE savings\_accounts (

account\_id SERIAL PRIMARY KEY,

customer\_id INTEGER,

balance NUMERIC

);

// Create employees table

CREATE TABLE employees (

employee\_id SERIAL PRIMARY KEY,

name TEXT,

department TEXT,

salary NUMERIC

);

// Create accounts table

CREATE TABLE accounts (

account\_id SERIAL PRIMARY KEY,

customer\_id INTEGER,

balance NUMERIC

);

//Insert sample data into savings\_accounts

INSERT INTO savings\_accounts (customer\_id, balance) VALUES

(1, 1000),

(2, 2000),

(3, 1500);

//Insert sample data into employees

INSERT INTO employees (name, department, salary) VALUES

('Alice', 'Sales', 5000),

('Bob', 'HR', 4500),

('Charlie', 'Sales', 5500),

('Diana', 'IT', 6000);

//Insert sample data into accounts

INSERT INTO accounts (customer\_id, balance) VALUES

(1, 3000),

(2, 4000);

**Scenario 1: ProcessMonthlyInterest**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest()

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE savings\_accounts

SET balance = balance + (balance \* 0.01);

RAISE NOTICE 'Monthly interest processed for all savings accounts.';

END;

$$;

**Scenario 2: UpdateEmployeeBonus**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept TEXT,

bonus\_percent NUMERIC

)

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE employees

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

WHERE department = dept;

RAISE NOTICE 'Bonus of % percent applied to department %.', bonus\_percent, dept;

END;

$$;

**Scenario 3: TransferFunds**

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_account INT,

to\_account INT,

amount NUMERIC

)

LANGUAGE plpgsql

AS $$

DECLARE

from\_balance NUMERIC;

BEGIN

-- Get current balance of source account

SELECT balance INTO from\_balance

FROM accounts

WHERE account\_id = from\_account;

-- Check if sufficient funds

IF from\_balance < amount THEN

RAISE EXCEPTION 'Insufficient balance in source account.';

END IF;

-- Deduct from source

UPDATE accounts

SET balance = balance - amount

WHERE account\_id = from\_account;

-- Add to destination

UPDATE accounts

SET balance = balance + amount

WHERE account\_id = to\_account;

RAISE NOTICE 'Transferred % from account % to account %.', amount, from\_account, to\_account;

END;

$$;

**Test calls to procedures**

**//1. Process monthly interest**

CALL ProcessMonthlyInterest();

**//2. Update bonus for Sales department**

CALL UpdateEmployeeBonus('Sales', 10);

**// 3. Transfer funds from account 1 to account 2**

CALL TransferFunds(1, 2, 500);

**Show final table data to verify**

**//Show savings accounts**

SELECT \* FROM savings\_accounts ORDER BY account\_id;

**// Show employees**

SELECT \* FROM employees ORDER BY employee\_id;

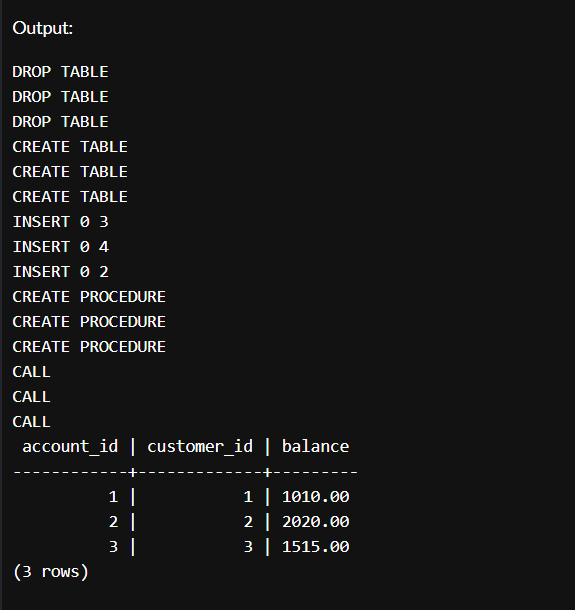
**// Show accounts**

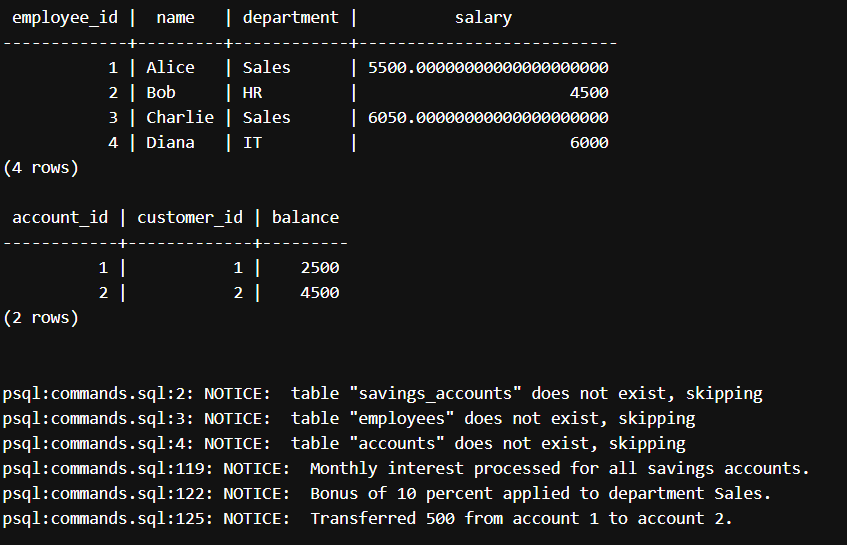
SELECT \* FROM accounts ORDER BY account\_id;

**EXPLANATION :**

This script sets up tables with sample banking data and defines procedures to process monthly interest, update employee bonuses, and transfer funds securely. Each procedure performs targeted updates based on specific business rules. The final queries display the updated records for easy verification.

**OUTPUT:**

****



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_