**Sistem de gestiune al bazei de date pentru cafeneaua**

Brew Haven

[🧋](https://emojipedia.org/hot-beverage)

**CUPRINS**

1. Prezentați pe scurt baza de date (utilitatea ei). – [**Pg. 4**](#Utilitate)

2. Realizați diagrama entitate-relație (ERD): entitățile, relațiile și atributele trebuie definite în limba română (vezi curs SGBD, model de diagrama entitate-relație; nu se va accepta alt format).  [**Pg. - 5**](#erd)

3. Pornind de la diagrama entitate-relație realizați diagrama conceptuală a modelului propus, integrând toate atributele necesare: entitățile, relațiile și atributele trebuie definite în limba română. [**Pg - 6**](#Conceptuala)

4. Implementați în Oracle diagrama conceptuală realizată: definiți toate tabelele, adăugând toate constrângerile de integritate necesare (chei primare, cheile externe etc). [**Pg - 7**](#Crearea_tabelelor)

5. Adăugați informații coerente în tabelele create (minim 5 înregistrări pentru fiecare entitate independentă; minim 10 înregistrări pentru fiecare tabelă asociativă). [**Pg - 11**](#Inserarea)

6. Formulați în limbaj natural o problemă pe care să o rezolvați folosind un subprogram stocat independent care să utilizeze toate cele 3 tipuri de colecții studiate. Apelați subprogramul. [**Pg - 23**](#sase)

7. Formulați în limbaj natural o problemă pe care să o rezolvați folosind un subprogram stocat independent care să utilizeze 2 tipuri diferite de cursoare studiate, unul dintre acestea fiind cursor parametrizat, dependent de celălalt cursor. Apelați subprogramul. [**Pg - 30**](#sapte)

8. Formulați în limbaj natural o problemă pe care să o rezolvați folosind un subprogram stocat independent de tip funcție care să utilizeze într-o singură comandă SQL 3 dintre tabelele create. Tratați toate excepțiile care pot apărea, incluzând excepțiile predefinite NO\_DATA\_FOUND și TOO\_MANY\_ROWS. Apelați subprogramul astfel încât să evidențiați toate cazurile tratate. [**Pg - 35**](#opt)

9. Formulați în limbaj natural o problemă pe care să o rezolvați folosind un subprogram stocat independent de tip procedură care să aibă minim 2 parametri și să utilizeze într-o singură comandă SQL 5 dintre tabelele create. Definiți minim 2 excepții proprii, altele decât cele predefinite la nivel de sistem. Apelați subprogramul astfel încât să evidențiați toate cazurile definite și tratate.  [**Pg - 38**](#noua)

10. Definiți un trigger de tip LMD la nivel de comandă. Declanșați trigger-ul. [**Pg - 45**](#zece)

11. Definiți un trigger de tip LMD la nivel de linie. Declanșați trigger-ul. [**Pg - 49**](#unspe)

12. Definiți un trigger de tip LDD. Declanșați trigger-ul. [**Pg - 54**](#doispe)

13. Formulați în limbaj natural o problemă pe care să o rezolvați folosind un pachet care să includă tipuri de date complexe și obiecte necesare unui flux de acțiuni integrate, specifice bazei de date definite (minim 2 tipuri de date, minim 2 funcții, minim 2 proceduri). [**Pg - 58**](#treispe)

Introducere

TEMA:

Cafeneaua "BREW HEAVEN" este un loc linistit si bine organizat, unde clientii pot savura bauturi variate si produse delicioase, totul intr-o atmosfera relaxanta, cu un sistem eficient de comenzi si bonusuri pentru angajati

INFRASTRUCTURA:

-Versiunea : Oracle Database 21c Express Edition Release 21.0.0.0.0 – Production Version 21.3.0.0.0

-Nu folosesc o masina virtuala

-Am lucrat pe: Microsoft windows 11 Pro | Versiunea 10.0.22631 Build 22631

-RAM alocat : 4GB

1. Utilitatea bazei de date:

Baza de date este creata pentru a gestiona comenzile dintr-o cafenea.

Tabelul Bauturi stocheaza detalii despre bauturile disponibile, iar Produse contine informatii despre produsele servite alaturi, cum ar fi gustarile.

Clientii sunt inregistrati in tabelul Clienti, iar angajatii sunt evidentiati in tabelul Angajati, impreuna cu date despre salariu si rol.

Prezenta angajatilor la munca este urmarita prin tabelul Zile\_lucrate.

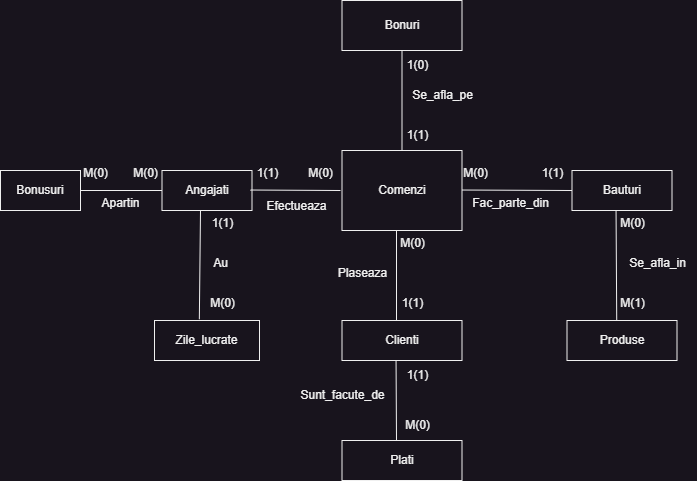
Comenzile plasate de clienti sunt salvate in tabelul Comenzi, care retine si angajatul responsabil. Platile sunt inregistrate in tabelul Plati, cu detalii despre metoda si suma platita.

Legatura dintre bauturi si produse este facuta prin tabelul Bauturi\_Produse, iar bonusurile oferite angajatilor sunt trecute in tabelul Bonusuri\_Angajati.

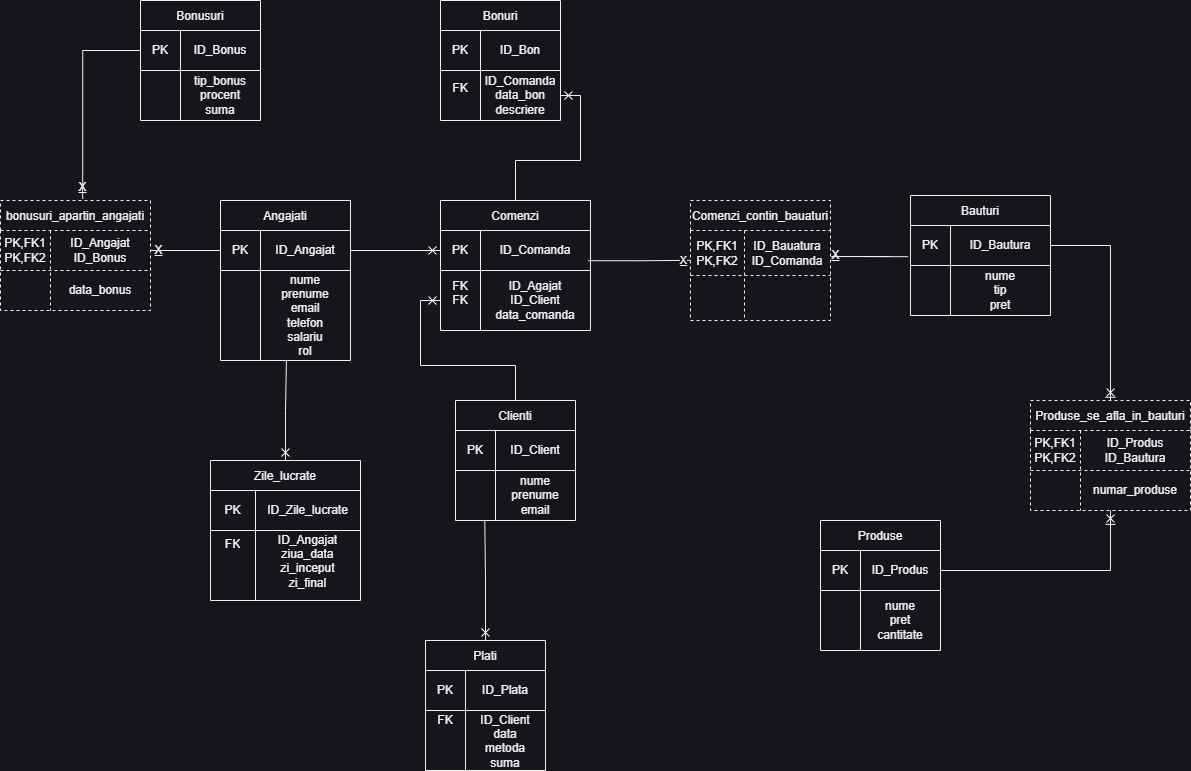
Fiecare comanda are un bon asociat, salvat in tabelul Bonuri, iar bauturile din fiecare comanda sunt evidentiate in tabelul Comenzi\_contin\_bauturi.

Aceasta schema ajuta la organizarea comenzilor, produselor si activitatii angajatilor in cafenea.

2. DIAGRAMA ERD



3.DIAGRAMA CONCEPTUALA



4. **Crearea tabelelor:**

CREATE TABLE Bonusuri

(

ID\_Bonus INT PRIMARY KEY,

tip\_bonus VARCHAR(255),

procent NUMBER(10,2),

suma NUMBER(4)

);

CREATE TABLE Bauturi (

ID\_Bautura INT PRIMARY KEY,

nume VARCHAR(255) UNIQUE,

tip VARCHAR(255) NOT NULL,

pret DECIMAL(10, 2) NOT NULL

);

CREATE TABLE Clienti (

ID\_Clienti INT PRIMARY KEY,

nume VARCHAR(255) UNIQUE,

prenume VARCHAR(255) UNIQUE,

email VARCHAR(255) UNIQUE

);

CREATE TABLE Plati (

ID\_Plata INT PRIMARY KEY,

data DATE NOT NULL,

metoda VARCHAR(255) NOT NULL,

suma DECIMAL(10, 2) NOT NULL,

ID\_Client INT NOT NULL,

FOREIGN KEY (ID\_Client) REFERENCES Clienti(ID\_Clienti)

);

CREATE TABLE Angajati (

ID\_Angajat INT PRIMARY KEY,

rol VARCHAR(255) NOT NULL,

nume VARCHAR(255) UNIQUE,

prenume VARCHAR(255) UNIQUE,

email VARCHAR(255) UNIQUE ,

telefon VARCHAR(255) UNIQUE,

salariu NUMBER(4)

);

CREATE TABLE Zile\_lucrate (

ID\_Zile\_lucrate INT PRIMARY KEY,

ziua\_data DATE NOT NULL,

zi\_inceput timestamp NOT NULL ,

zi\_final timestamp NOT NULL ,

ID\_Angajat INT,

FOREIGN KEY(ID\_Angajat) REFERENCES Angajati(ID\_Angajat)

);

CREATE TABLE Produse (

ID\_Produs INT PRIMARY KEY,

nume VARCHAR(255)UNIQUE ,

pret DECIMAL(10, 2)NOT NULL,

cantitate INT NOT NULL

);

CREATE TABLE Comenzi (

ID\_Comanda INT PRIMARY KEY,

ID\_Angajat INT,

ID\_Client INT,

data\_comanda DATE,

FOREIGN KEY (ID\_Angajat) REFERENCES Angajati(ID\_Angajat),

FOREIGN KEY (ID\_Client) REFERENCES Clienti(ID\_Clienti)

);

CREATE TABLE Bauturi\_Produse (

numar\_produse INT,

ID\_Bautura INT,

ID\_Produs INT,

PRIMARY KEY (ID\_Bautura, ID\_Produs),

FOREIGN KEY (ID\_Bautura) REFERENCES Bauturi(ID\_Bautura),

FOREIGN KEY (ID\_Produs) REFERENCES Produse(ID\_Produs)

);

CREATE TABLE Bonusuri\_Angajati (

data\_bonus DATE NOT NULL,

ID\_Bonus INT,

ID\_Angajat INT,

PRIMARY KEY (ID\_Bonus,ID\_Angajat),

FOREIGN KEY (ID\_Bonus) REFERENCES Bonusuri(ID\_Bonus),

FOREIGN KEY (ID\_Angajat) REFERENCES Angajati(ID\_Angajat)

);

CREATE TABLE Comenzi\_contin\_bauturi

(

ID\_Comanda,

ID\_Bautura,

PRIMARY KEY(ID\_Comanda,ID\_Bautura),

FOREIGN KEY (ID\_Comanda) REFERENCES Comenzi(ID\_Comanda),

FOREIGN KEY (ID\_Bautura) REFERENCES Bauturi(ID\_Bautura)

);

CREATE TABLE Bonuri (

ID\_Bon INT PRIMARY KEY,

Data\_Bon DATE NOT NULL,

descriere VARCHAR(4000),

ID\_Comanda INT UNIQUE NOT NULL,

FOREIGN KEY(ID\_Comanda) REFERENCES Comenzi(ID\_Comanda)

);

5. **Inserarea datelor in tabele:**

INSERT INTO Bonusuri (ID\_Bonus, tip\_bonus, procent, suma) VALUES (1, 'Performance Bonus', 10.50, 500);

INSERT INTO Bonusuri (ID\_Bonus, tip\_bonus, procent, suma) VALUES (2, 'Loyalty Bonus', 5.25, 250);

INSERT INTO Bonusuri (ID\_Bonus, tip\_bonus, procent, suma) VALUES (3, 'Sales Bonus', 15.75, 750);

INSERT INTO Bonusuri (ID\_Bonus, tip\_bonus, procent, suma) VALUES (4, 'Team Achievement', 12.30, 600);

INSERT INTO Bonusuri (ID\_Bonus, tip\_bonus, procent, suma) VALUES (5, 'Customer Satisfaction', 8.20, 400);

INSERT INTO Angajati (ID\_Angajat, rol, nume, prenume, email, telefon, salariu) VALUES (1, 'Manager', 'Popescu', 'Ion', 'ion.popescu@coffeeshop.ro', '0712345678', 5000);

INSERT INTO Angajati (ID\_Angajat, rol, prenume, nume, email, telefon, salariu) VALUES (2, 'Barista', 'Maria', 'Ionescu', 'maria.ionescu@coffeeshop.ro', '0723456789', 3500);

INSERT INTO Angajati (ID\_Angajat, rol, prenume, nume, email, telefon, salariu) VALUES (3, 'Waiter', 'Dan', 'Cristea', 'dan.cristea@coffeeshop.ro', '0734567890', 3200);

INSERT INTO Angajati (ID\_Angajat, rol, prenume, nume, email, telefon, salariu) VALUES (4, 'Cashier', 'Elena', 'Stancu', 'elena.stancu@coffeeshop.ro', '0745678901', 3300);

INSERT INTO Angajati (ID\_Angajat, rol, prenume, nume, email, telefon, salariu) VALUES (5, 'Chef', 'Andreea', 'Moldovan', 'andreea.moldovan@coffeeshop.ro', '0756789012', 4000);

INSERT INTO Angajati (ID\_Angajat, rol, prenume, nume, email, telefon, salariu) VALUES (6, 'Barista', 'Alex', 'Dragomir', 'alex.dragomir@coffeeshop.ro', '0767890123', 3600);

INSERT INTO Angajati (ID\_Angajat, rol, prenume, nume, email, telefon, salariu) VALUES (7, 'Waiter', 'Mihai', 'Popa', 'mihai.popa@coffeeshop.ro', '0778901234', 3250);

INSERT INTO Angajati (ID\_Angajat, rol, prenume, nume, email, telefon, salariu) VALUES (8, 'Waiter', 'Matei', 'Ivan', 'matei.ivan@coffeeshop.ro', '0778261234', 3700);

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (1, 'Radu', 'Cristina', 'cristina.radu@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (2, 'Marin', 'George', 'george.marin@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (3, 'Florea', 'Ana', 'ana.florea@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (4, 'Stoica', 'Mihai', 'mihai.stoica@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (5, 'Tanase', 'Elena', 'elena.tanase@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (6, 'Nistor', 'Adrian', 'adrian.nistor@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (7, 'Dobre', 'Iulia', 'iulia.dobre@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (8, 'Petre', 'Razvan', 'razvan.petre@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (9, 'Ionescu', 'Maria', 'maria.ionescu@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (10, 'Popa', 'Andrei', 'andrei.popa@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (11, 'Dumitru', 'Ana-maria', 'ana.dumitru@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (12, 'Stoica', 'Ioana', 'ioana.stoica@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (13, 'Vasilescu', 'Iosif', 'mihai.vasilescu@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (14, 'Marinescu', 'Mgadalena', 'elena.marinescu@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (15, 'Tudor', 'Alexandru', 'alexandru.tudor@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (16, 'Georgescu', 'Cristi', 'cristina.georgescu@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (17, 'Iliescu', 'Radu', 'radu.iliescu@email.com');

INSERT INTO Clienti (ID\_Clienti, nume, PRENUME, email) VALUES (18, 'Preda', 'Adriana', 'adriana.preda@email.com');

INSERT INTO Bauturi (ID\_Bautura, nume, tip, pret) VALUES (1, 'Espresso', 'Cafea', 7.50);

INSERT INTO Bauturi (ID\_Bautura, nume, tip, pret) VALUES (2, 'Cappuccino', 'Cafea', 9.75);

INSERT INTO Bauturi (ID\_Bautura, nume, tip, pret) VALUES (3, 'Latte', 'Cafea', 10.25);

INSERT INTO Bauturi (ID\_Bautura, nume, tip, pret) VALUES (4, 'Ceai Verde', 'Ceai', 6.50);

INSERT INTO Bauturi (ID\_Bautura, nume, tip, pret) VALUES (5, 'Smoothie Fructe', 'Bautura Racoritoare', 12.00);

INSERT INTO Bauturi (ID\_Bautura, nume, tip, pret) VALUES (6, 'Americano', 'Cafea', 8.25);

INSERT INTO Bauturi (ID\_Bautura, nume, tip, pret) VALUES (7, 'Ceai Negru', 'Ceai', 6.75);

INSERT INTO Produse (ID\_Produs, nume, pret, cantitate) VALUES (1, 'Cafea', 10.00, 50);

INSERT INTO Produse (ID\_Produs, nume, pret, cantitate) VALUES (2, 'Zahar', 12.50, 40);

INSERT INTO Produse (ID\_Produs, nume, pret, cantitate) VALUES (3, 'Ceai Negru', 13.75, 35);

INSERT INTO Produse (ID\_Produs, nume, pret, cantitate) VALUES (4, 'Ciocolata ', 14.00, 30);

INSERT INTO Produse (ID\_Produs, nume, pret, cantitate) VALUES (5, 'Ceai Verde', 9.00, 60);

INSERT INTO Produse (ID\_Produs, nume, pret, cantitate) VALUES (6, 'Scortisoara', 8.25, 30);

INSERT INTO Produse (ID\_Produs, nume, pret, cantitate) VALUES (7, 'Sirop vanilie', 12.75, 25);

INSERT INTO Produse (ID\_Produs, nume, pret, cantitate) VALUES (8, 'Sirop caramel', 22.00, 15);

INSERT INTO Produse (ID\_Produs, nume, pret, cantitate) VALUES (9, 'Lapte', 10.50, 40);

INSERT INTO Produse (ID\_Produs, nume, pret, cantitate) VALUES (10, 'Lapte de cocos', 14.25, 18);

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (1, 2, 1, TO\_DATE('2024-01-15', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (2, 6, 2, TO\_DATE('2024-01-16', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (3, 3, 3, TO\_DATE('2024-01-17', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (4, 4, 4, TO\_DATE('2024-01-18', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (5, 2, 5, TO\_DATE('2024-01-19', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (6, 6, 6, TO\_DATE('2024-01-20', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (7, 3, 7, TO\_DATE('2024-01-21', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (8, 4, 8, TO\_DATE('2024-01-22', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (9, 2, 5, TO\_DATE('2024-01-25', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (10, 3, 7, TO\_DATE('2024-01-28', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (11, 4, 8, TO\_DATE('2024-02-05', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (12, 1, 6, TO\_DATE('2024-02-10', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (13, 5, 9, TO\_DATE('2024-02-15', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (14, 2, 10, TO\_DATE('2024-02-20', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (15, 4, 11, TO\_DATE('2024-02-25', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (16, 3, 12, TO\_DATE('2024-03-02', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (17, 1, 7, TO\_DATE('2024-03-07', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (18, 5, 8, TO\_DATE('2024-03-12', 'YYYY-MM-DD'));

--pt too many rows

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (101, 1, 1,TO\_DATE('2024-03-07', 'YYYY-MM-DD'));

INSERT INTO Comenzi (ID\_Comanda, ID\_Angajat, ID\_Client, data\_comanda) VALUES (102, 1, 1, TO\_DATE('2024-01-20', 'YYYY-MM-DD'));

INSERT INTO Bonuri (ID\_Bon, Data\_Bon, ID\_Comanda) VALUES (1, TO\_DATE('2024-01-15', 'YYYY-MM-DD'), 1);

INSERT INTO Bonuri (ID\_Bon, Data\_Bon, ID\_Comanda) VALUES (2, TO\_DATE('2024-01-16', 'YYYY-MM-DD'), 2);

INSERT INTO Bonuri (ID\_Bon, Data\_Bon, ID\_Comanda) VALUES (3, TO\_DATE('2024-01-17', 'YYYY-MM-DD'), 3);

INSERT INTO Bonuri (ID\_Bon, Data\_Bon, ID\_Comanda) VALUES (4, TO\_DATE('2024-01-18', 'YYYY-MM-DD'), 4);

INSERT INTO Bonuri (ID\_Bon, Data\_Bon, ID\_Comanda) VALUES (5, TO\_DATE('2024-01-19', 'YYYY-MM-DD'), 5);

INSERT INTO Bonuri (ID\_Bon, Data\_Bon, ID\_Comanda) VALUES (6, TO\_DATE('2024-01-20', 'YYYY-MM-DD'), 6);

INSERT INTO Bonuri (ID\_Bon, Data\_Bon, ID\_Comanda) VALUES (7, TO\_DATE('2024-01-21', 'YYYY-MM-DD'), 7);

INSERT INTO Bonuri (ID\_Bon, Data\_Bon, ID\_Comanda) VALUES (8, TO\_DATE('2024-01-22', 'YYYY-MM-DD'), 8);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (2, 1, 2);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (1, 2, 1);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (3, 3, 3);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (2, 4, 5);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (1, 5, 3);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (2, 6, 4);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (3, 7, 6);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (1, 1, 5);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (2, 2, 6);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (3, 3, 2);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (1, 4, 4);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (2, 5, 1);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (3, 6, 3);

INSERT INTO Bauturi\_Produse (numar\_produse, ID\_Bautura, ID\_Produs) VALUES (1, 7, 5);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-15', 'YYYY-MM-DD'), 1, 1);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-16', 'YYYY-MM-DD'), 2, 2);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-17', 'YYYY-MM-DD'), 3, 3);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-18', 'YYYY-MM-DD'), 4, 4);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-19', 'YYYY-MM-DD'), 5, 5);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-20', 'YYYY-MM-DD'), 1, 6);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-21', 'YYYY-MM-DD'), 2, 7);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-22', 'YYYY-MM-DD'), 3, 1);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-23', 'YYYY-MM-DD'), 4, 2);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-24', 'YYYY-MM-DD'), 5, 3);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-25', 'YYYY-MM-DD'), 1, 4);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-26', 'YYYY-MM-DD'), 2, 5);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-27', 'YYYY-MM-DD'), 3, 6);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-28', 'YYYY-MM-DD'), 4, 7);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-29', 'YYYY-MM-DD'), 5, 1);

INSERT INTO Bonusuri\_Angajati (data\_bonus, ID\_Bonus, ID\_Angajat) VALUES (TO\_DATE('2024-01-30', 'YYYY-MM-DD'), 1, 2);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat) VALUES

(1, TO\_DATE('2024-01-15', 'YYYY-MM-DD'),

TO\_TIMESTAMP('2024-01-15 08:00:00', 'YYYY-MM-DD HH24:MI:SS'),

TO\_TIMESTAMP('2024-01-15 16:00:00', 'YYYY-MM-DD HH24:MI:SS'), 1);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat) VALUES

(2, TO\_DATE('2024-01-15', 'YYYY-MM-DD'),

TO\_TIMESTAMP('2024-01-15 09:00:00', 'YYYY-MM-DD HH24:MI:SS'),

TO\_TIMESTAMP('2024-01-15 17:00:00', 'YYYY-MM-DD HH24:MI:SS'), 2);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat) VALUES

(3, TO\_DATE('2024-01-16', 'YYYY-MM-DD'),

TO\_TIMESTAMP('2024-01-16 10:00:00', 'YYYY-MM-DD HH24:MI:SS'),

TO\_TIMESTAMP('2024-01-16 18:00:00', 'YYYY-MM-DD HH24:MI:SS'), 3);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat) VALUES

(4, TO\_DATE('2024-01-16', 'YYYY-MM-DD'),

TO\_TIMESTAMP('2024-01-16 08:30:00', 'YYYY-MM-DD HH24:MI:SS'),

TO\_TIMESTAMP('2024-01-16 16:30:00', 'YYYY-MM-DD HH24:MI:SS'), 4);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat) VALUES

(5, TO\_DATE('2024-01-17', 'YYYY-MM-DD'),

TO\_TIMESTAMP('2024-01-17 07:45:00', 'YYYY-MM-DD HH24:MI:SS'),

TO\_TIMESTAMP('2024-01-17 15:45:00', 'YYYY-MM-DD HH24:MI:SS'), 5);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat) VALUES

(6, TO\_DATE('2024-01-17', 'YYYY-MM-DD'),

TO\_TIMESTAMP('2024-01-17 09:15:00', 'YYYY-MM-DD HH24:MI:SS'),

TO\_TIMESTAMP('2024-01-17 17:15:00', 'YYYY-MM-DD HH24:MI:SS'), 6);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat) VALUES

(7, TO\_DATE('2024-01-18', 'YYYY-MM-DD'),

TO\_TIMESTAMP('2024-01-18 08:00:00', 'YYYY-MM-DD HH24:MI:SS'),

TO\_TIMESTAMP('2024-01-18 16:00:00', 'YYYY-MM-DD HH24:MI:SS'), 7);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (8, TO\_DATE('2024-01-19', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-19 08:30:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-19 16:30:00', 'YYYY-MM-DD HH24:MI:SS'), 4);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (9, TO\_DATE('2024-01-19', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-19 08:30:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-19 16:30:00', 'YYYY-MM-DD HH24:MI:SS'), 1);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (10, TO\_DATE('2024-01-20', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-20 09:00:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-20 17:00:00', 'YYYY-MM-DD HH24:MI:SS'), 2);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (11, TO\_DATE('2024-01-21', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-21 07:30:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-21 15:30:00', 'YYYY-MM-DD HH24:MI:SS'), 3);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (12, TO\_DATE('2024-01-22', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-22 10:00:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-22 18:00:00', 'YYYY-MM-DD HH24:MI:SS'), 4);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (13, TO\_DATE('2024-01-23', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-23 08:00:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-23 16:00:00', 'YYYY-MM-DD HH24:MI:SS'), 5);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (14, TO\_DATE('2024-01-24', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-24 07:00:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-24 15:00:00', 'YYYY-MM-DD HH24:MI:SS'), 6);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (15, TO\_DATE('2024-01-25', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-25 09:30:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-25 17:30:00', 'YYYY-MM-DD HH24:MI:SS'), 7);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (16, TO\_DATE('2024-01-26', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-26 08:30:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-26 16:30:00', 'YYYY-MM-DD HH24:MI:SS'), 1);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (17, TO\_DATE('2024-01-27', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-27 07:30:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-27 15:30:00', 'YYYY-MM-DD HH24:MI:SS'), 2);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (18, TO\_DATE('2024-01-28', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-28 09:00:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-28 17:00:00', 'YYYY-MM-DD HH24:MI:SS'), 3);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (19, TO\_DATE('2024-01-29', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-29 08:00:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-29 16:00:00', 'YYYY-MM-DD HH24:MI:SS'), 4);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (20, TO\_DATE('2024-01-30', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-01-30 07:00:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-01-30 15:00:00', 'YYYY-MM-DD HH24:MI:SS'), 4);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (21, TO\_DATE('2024-02-01', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-02-01 09:30:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-02-01 17:30:00', 'YYYY-MM-DD HH24:MI:SS'), 6);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (22, TO\_DATE('2024-02-02', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-02-02 08:30:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-02-02 16:30:00', 'YYYY-MM-DD HH24:MI:SS'), 4);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (23, TO\_DATE('2024-02-03', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-02-03 09:00:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-02-03 17:00:00', 'YYYY-MM-DD HH24:MI:SS'), 1);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (24, TO\_DATE('2024-02-04', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-02-04 07:30:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-02-04 15:30:00', 'YYYY-MM-DD HH24:MI:SS'), 2);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (25, TO\_DATE('2024-02-05', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-02-05 08:00:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-02-05 16:00:00', 'YYYY-MM-DD HH24:MI:SS'), 3);

INSERT INTO Zile\_lucrate (ID\_Zile\_lucrate, ziua\_data, zi\_inceput, zi\_final, ID\_Angajat)

VALUES (26, TO\_DATE('2024-02-06', 'YYYY-MM-DD'), TO\_TIMESTAMP('2024-02-06 10:00:00', 'YYYY-MM-DD HH24:MI:SS'), TO\_TIMESTAMP('2024-02-06 18:00:00', 'YYYY-MM-DD HH24:MI:SS'), 4);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (1, TO\_DATE('2024-01-15', 'YYYY-MM-DD'), 'Card', 45.50, 1);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (2, TO\_DATE('2024-01-16', 'YYYY-MM-DD'), 'Numerar', 35.25, 2);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (3, TO\_DATE('2024-01-17', 'YYYY-MM-DD'), 'Card', 62.75, 3);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (4, TO\_DATE('2024-01-18', 'YYYY-MM-DD'), 'Numerar', 28.50, 4);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (5, TO\_DATE('2024-01-19', 'YYYY-MM-DD'), 'Card', 55.00, 5);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (6, TO\_DATE('2024-01-20', 'YYYY-MM-DD'), 'Numerar', 42.75, 6);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (7, TO\_DATE('2024-01-21', 'YYYY-MM-DD'), 'Card', 38.25, 7);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (8, TO\_DATE('2024-01-22', 'YYYY-MM-DD'), 'Numerar', 52.50, 8);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (9, TO\_DATE('2024-01-23', 'YYYY-MM-DD'), 'Card', 120.00, 8);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (10, TO\_DATE('2024-01-24', 'YYYY-MM-DD'), 'Numerar', 75.25, 6);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (11, TO\_DATE('2024-01-25', 'YYYY-MM-DD'), 'Card', 95.00, 6);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (12, TO\_DATE('2024-01-26', 'YYYY-MM-DD'), 'Numerar', 45.90, 6);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (13, TO\_DATE('2024-01-27', 'YYYY-MM-DD'), 'Card', 60.75, 4);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (14, TO\_DATE('2024-01-28', 'YYYY-MM-DD'), 'Transfer Bancar', 150.30, 3);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (15, TO\_DATE('2024-01-29', 'YYYY-MM-DD'), 'Card', 210.50, 2);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (16, TO\_DATE('2024-01-30', 'YYYY-MM-DD'), 'Numerar', 35.75, 1);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (17, TO\_DATE('2024-02-01', 'YYYY-MM-DD'), 'Card', 87.90, 8);

INSERT INTO Plati (ID\_Plata, data, metoda, suma, ID\_Client) VALUES (18, TO\_DATE('2024-02-02', 'YYYY-MM-DD'), 'Transfer Bancar', 100.00, 6);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (1, 1);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (1, 2);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (1, 3);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (2, 1);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (2, 4);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (2, 5);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (3, 2);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (3, 6);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (3, 7);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (4, 1);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (4, 3);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (4, 5);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (5, 2);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (5, 4);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (5, 6);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (6, 1);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (6, 7);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (7, 3);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (7, 5);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (8, 6);

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura) VALUES (12, 1);

6. Cerinta problemei :

Scrieti un program PL/SQL numit contributie care sa determine pentru fiecare angajat:

-Numele si prenumele acestuia si ziua din saptamana in care fiecare angajat a avut cele mai multe comenzi,iar in caz ca acesta nu are comenzi sa se afiseze: ‘Numele angajatului’ NU ARE COMENZI.

-Venitul total generat prin vanzarea bauturilor, folosind pretul bauturilor vandute in comenzile procesate de acestia, cat si lista tuturor bauturilor vandute de fiecare angajat, iar in caz ca venitul generat a fost 0 in loc de lista de bauturi sa se afiseze ‘Nu a vandut nicio bautura’

-Folositi toate cele 3 tipuri de colecții studiate

CREATE OR REPLACE TYPE zile\_vector AS VARRAY(8) OF VARCHAR2(20);

/

CREATE OR REPLACE TYPE frecventa AS VARRAY(8) OF NUMBER(2);

/

CREATE OR REPLACE TYPE lista\_bauturi IS TABLE OF VARCHAR2(50); --Aici am folosit nested table

/

CREATE OR REPLACE PROCEDURE contributie

AS

--De acesti 2 VARRAYS ma folosesc pentru a creea un vector de frecventa pentru zilele sptamanii

zile zile\_vector := zile\_vector('Luni', 'Marti', 'Miercuri', 'Joi', 'Vineri', 'Sambata', 'Duminica');

zile\_frecventa frecventa := frecventa(0,0,0,0,0,0,0);

TYPE ang IS RECORD

(

nume angajati.nume%TYPE,

prenume angajati.prenume%TYPE,

venit\_generat bauturi.pret%TYPE,

l\_bauturi lista\_bauturi ,

data\_comanda comenzi.data\_comanda%TYPE

);

--De acest INDEX TABLE ma folosesc pentru a stoca date despre fiecare angajat comform cerintei

TYPE t\_clasament\_angajati IS TABLE OF ang INDEX BY PLS\_INTEGER ; --Aici am folosit tabel indexat

clasament\_angajati t\_clasament\_angajati := t\_clasament\_angajati();

CURSOR nume\_prenume IS

SELECT nume,prenume

FROM angajati;

--Acest cursor selecteaza date din 4 tabele pentru a parcurge mai usor datele necesare angajatilor pe care le introduc

--ulterior in clasament\_angajati care are ca tip de date RECORD-ul ang

CURSOR date\_bauturi\_angajat IS

SELECT

A.nume AS Nume\_Angajat,

A.prenume AS Prenume\_Angajat,

B.nume AS Nume\_Bautura,

B.pret AS Pret\_Bautura,

C.data\_comanda AS Data\_comanda

FROM

Comenzi C

INNER JOIN

Comenzi\_contin\_bauturi CB ON C.ID\_Comanda = CB.ID\_Comanda

INNER JOIN

Bauturi B ON CB.ID\_Bautura = B.ID\_Bautura

INNER JOIN

Angajati A ON C.ID\_Angajat = A.ID\_Angajat;

CURSOR data\_ziua IS

SELECT

A.nume AS Nume\_Angajat,

C.data\_comanda AS Data\_comanda

FROM angajati A

INNER JOIN

comenzi C ON C.ID\_Angajat = A.ID\_Angajat;

CURSOR angaj IS

SELECT nume,prenume

FROM angajati;

--Variabile

idx INT :=0; --Il folosesc pentru initializarea tabelului indexat clasament\_angajati

ziua VARCHAR(50);

cnt NUMBER(2);

idx1 INT; --Il folosesc pentru vectorul de frecventa

valoare\_max NUMBER := 0;

index\_max NUMBER := 0;

BEGIN

--Initializez feicare angajat

FOR n IN nume\_prenume LOOP

idx := idx +1;

clasament\_angajati(idx).nume := n.nume;

clasament\_angajati(idx).prenume := n.prenume;

clasament\_angajati(idx).venit\_generat := 0;

clasament\_angajati(idx).l\_bauturi := lista\_bauturi();

END LOOP;

--Calculez pentru fiecare venitul generat din bauturile vandute

--Introduc in NESTED TABLE-ul l\_bauturi , numele tuturor bauturilor vandute pentru a le afisa mai incolo

FOR i IN 1..clasament\_angajati.COUNT LOOP

FOR d IN date\_bauturi\_angajat LOOP

IF d.Nume\_Angajat = clasament\_angajati(i).nume THEN

clasament\_angajati(i).venit\_generat := clasament\_angajati(i).venit\_generat + d.Pret\_Bautura;

clasament\_angajati(i).l\_bauturi.EXTEND;

clasament\_angajati(I).l\_bauturi(clasament\_angajati(i).l\_bauturi.COUNT) := d.Nume\_Bautura;

END IF;

END LOOP;

END LOOP;

--In acest loop calculez vectorul de frecventa pentru zille saptamanii

FOR a IN angaj LOOP

zile\_frecventa := frecventa(0,0,0,0,0,0,0); -- initializez de fiecare data cu 0 toate pozitiile

valoare\_max := 0;

index\_max := 0;

FOR d IN data\_ziua LOOP

IF d.Nume\_Angajat = a.nume THEN

ziua := trim(to\_char(d.Data\_comanda, 'DAY')); -- am folosit trim pentru a ma asigura ca nu au spatii zilele saptamanii

idx1 := CASE --Acest case il folosesc pentru a determina ce pozitie a vectorului "zile frecventa" sa incrementez

WHEN ziua = 'MONDAY' THEN 1

WHEN ziua = 'TUESDAY' THEN 2

WHEN ziua = 'WEDNESDAY' THEN 3

WHEN ziua = 'THURSDAY' THEN 4

WHEN ziua = 'FRIDAY' THEN 5

WHEN ziua= 'SATURDAY' THEN 6

WHEN ziua = 'SUNDAY' THEN 7

ELSE NULL

END;

zile\_frecventa(idx1) := zile\_frecventa(idx1) +1;

END IF;

END LOOP;

FOR i IN 1..zile\_frecventa.COUNT LOOP

IF zile\_frecventa(i) > valoare\_max THEN

valoare\_max := zile\_frecventa(i);

index\_max := i;

END IF;

END LOOP;

IF index\_max > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Ziua din saptamana cu cele mai multe comenzi pentru ' || a.nume || ' ' || a.prenume || ' : ' || zile(index\_max));

ELSE

DBMS\_OUTPUT.PUT\_LINE( a.nume || ' ' || a.prenume || ' NU ARE COMENZI');

END IF;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('---------------#########---------------');

DBMS\_OUTPUT.PUT\_LINE(' ');

--Aici afisez pe rand toate datele obtinute despre angajati din index table ul clasament angajati

FOR i IN 1..clasament\_angajati.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('Angajat: ' || clasament\_angajati(i).nume || ' ' || clasament\_angajati(i).prenume);

DBMS\_OUTPUT.PUT\_LINE('Venit Generat: ' || clasament\_angajati(i).venit\_generat || ' RON');

IF clasament\_angajati(i).l\_bauturi.COUNT > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Bauturi Vandute:');

FOR j IN 1..clasament\_angajati(i).l\_bauturi.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('- ' || clasament\_angajati(i).l\_bauturi(j));

END LOOP;

ELSE

DBMS\_OUTPUT.PUT\_LINE(' Nu a vandut nicio bautura');

END IF;

DBMS\_OUTPUT.PUT\_LINE('---------------------------------------');

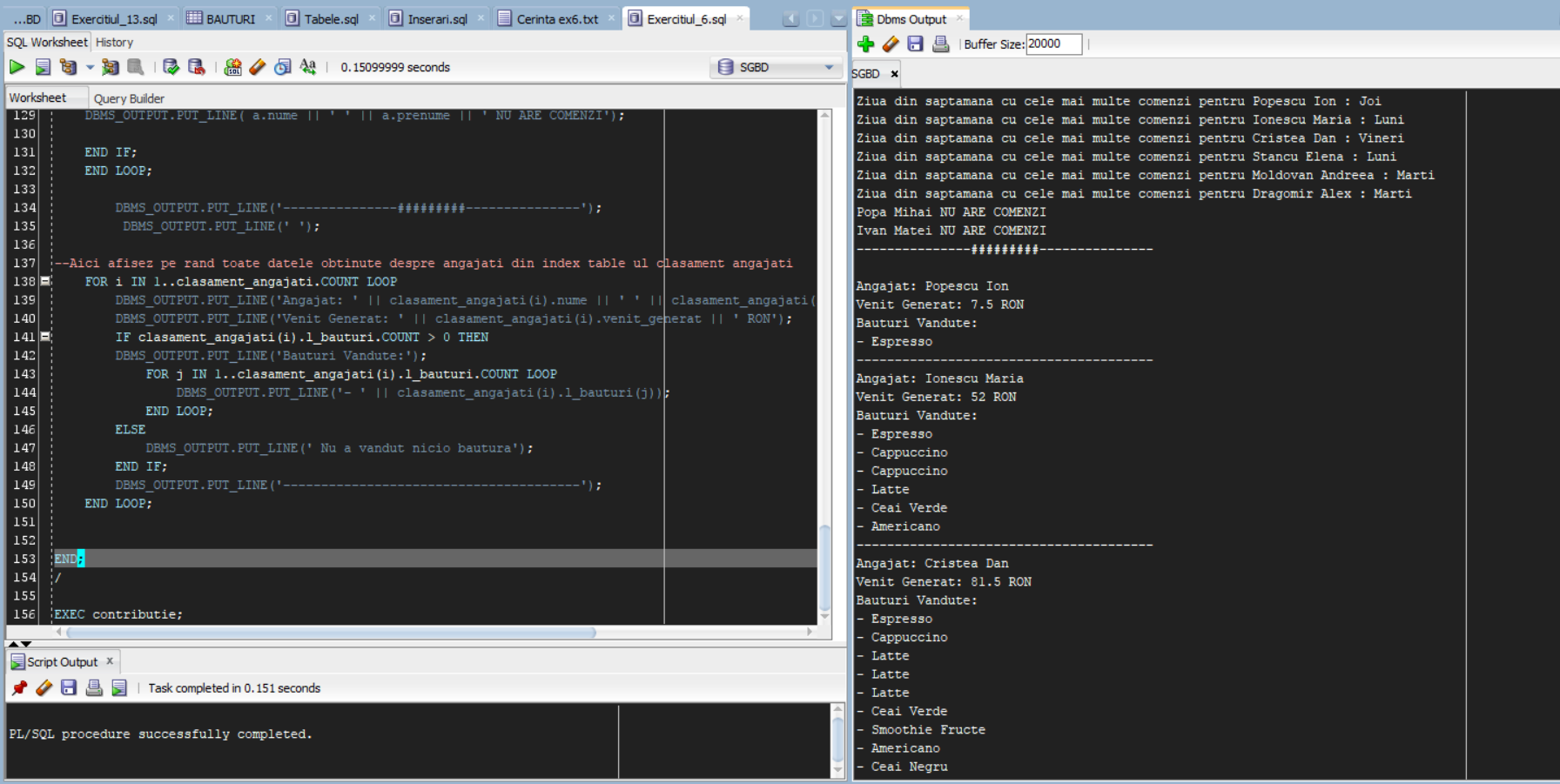
END LOOP;

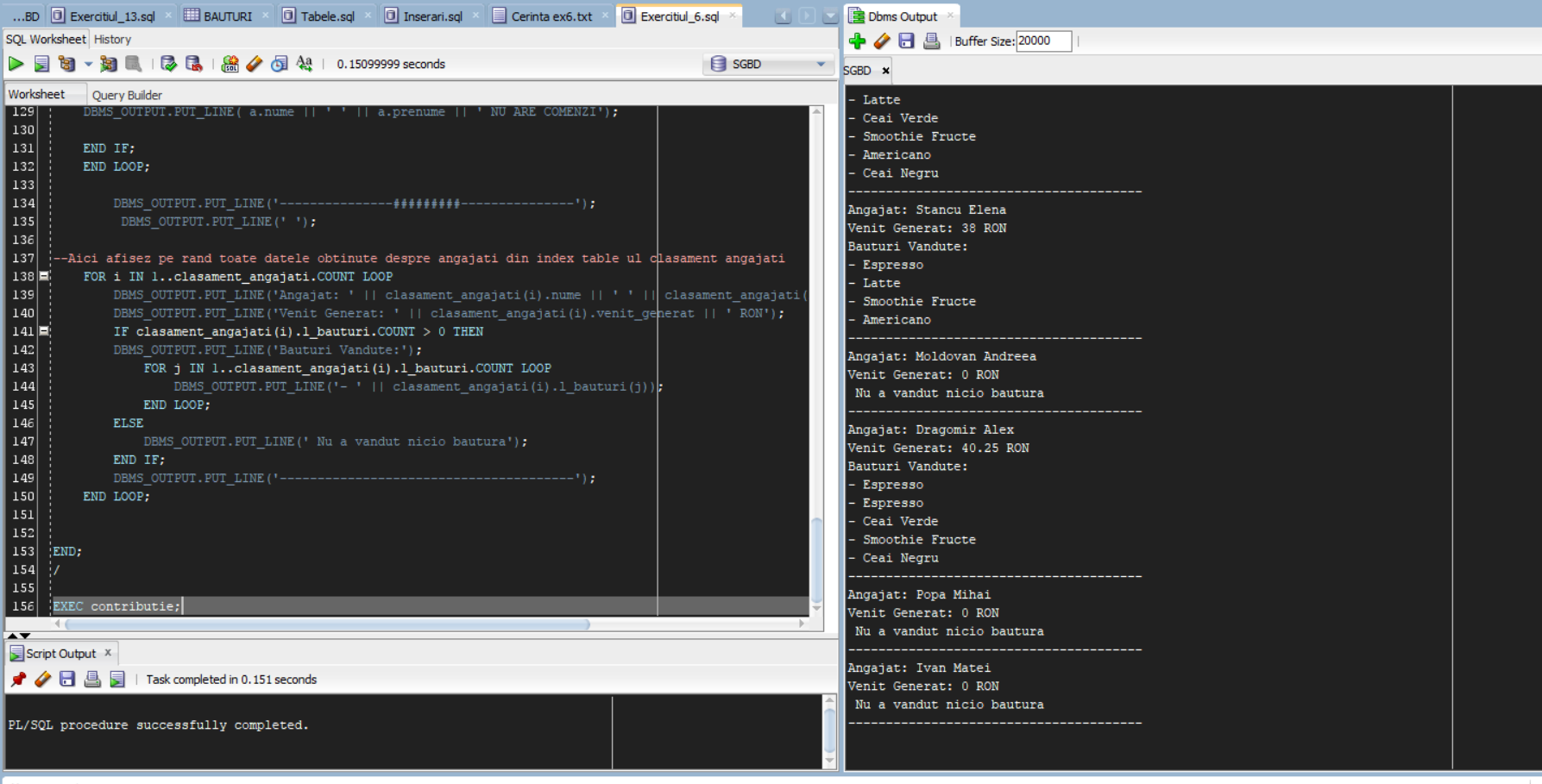
END;

/

EXEC contributie;

Screenshoturi:





7. Cerinta problemei:  
Scrieti un program PL/SQL numit raport\_clienti, care sa afiseze un raport detaliat pentru fiecare client.

Pentru fiecare client care are plati inregistrate, sa afiseze:

Numele si prenumele clientului.

Bauturile comandate si de cate ori au comandat aceasta bautura

Metoda de plata folosita.

Suma totala platita de client, afisata in lei.

Daca un client nu are plati inregistrate, sa afiseze un mesaj specific:”Client: [nume] [prenume] NU ARE PLATI INREGISTRATE”.

CREATE OR REPLACE PROCEDURE raport\_clienti

AS

CURSOR cli IS

SELECT id\_clienti,nume,prenume

FROM CLIENTI;

CURSOR date\_clienti IS -- Acest cursor il folosesc pentru a determina date despre fiecare client conform cerintei

SELECT

C.nume AS Nume\_client,

C.prenume AS Prenume\_client,

C.id\_clienti AS id\_client,

SUM(P.suma) AS Suma\_platita,

P.metoda AS Metoda\_plata

FROM clienti C

LEFT JOIN

Plati P ON P.ID\_client = C.ID\_Clienti

GROUP BY C.nume, C.prenume, C.id\_clienti, P.metoda;

CURSOR date\_bauturi(p\_client INT) IS --Acest cursor il folosesc pentru a determina bauturile si numarul acestora comandate de fiecare client

SELECT

B.nume AS Nume\_bautura,

COUNT(\*) AS Numar\_bauturi

FROM comenzi C

LEFT JOIN

comenzi\_contin\_bauturi CC ON CC.ID\_Comanda = C.ID\_Comanda

LEFT JOIN

bauturi B ON B.ID\_Bautura = CC.ID\_Bautura

WHERE ID\_Client = p\_client

GROUP BY nume;

--Variabile

ok NUMBER(2);

METODA VARCHAR(50);

TYPE ang IS RECORD(

Nume\_client clienti.nume%TYPE,

Prenume\_client clienti.prenume%TYPE,

id\_client clienti.id\_clienti%TYPE,

Total\_suma\_platita NUMBER(10, 2),

Metoda\_plata plati.metoda%TYPE

);

TYPE tabel\_clienti IS TABLE OF ang;

tabel\_date\_clienti tabel\_clienti;

BEGIN

OPEN date\_clienti; --Introduc in tabel toate informatiile angajatilor din cursorul date\_clienti pe cate o pozitie

FETCH date\_clienti BULK COLLECT INTO tabel\_date\_clienti;

CLOSE date\_clienti;

FOR j in cli LOOP

ok := 0;

FOR i IN tabel\_date\_clienti.FIRST .. tabel\_date\_clienti.LAST LOOP

metoda := tabel\_date\_clienti(i).Metoda\_plata;

IF j.id\_clienti = tabel\_date\_clienti(i).id\_client AND tabel\_date\_clienti(i).Metoda\_plata IS NOT NULL THEN --Identific clientul si daca acesta are macar o plata facut , in caz afirmativ afisez

IF ok = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Client: ' || tabel\_date\_clienti(i).Nume\_client || ' ' || tabel\_date\_clienti(i).Prenume\_client);

FOR d IN date\_bauturi(tabel\_date\_clienti(i).id\_client) LOOP

DBMS\_OUTPUT.PUT\_LINE('A comandat bautura ' || d.Nume\_bautura || ' de ' || d.Numar\_bauturi || ' ori');

END LOOP;

ok := 1;

END IF;

DBMS\_OUTPUT.PUT\_LINE('Metoda: ' || tabel\_date\_clienti(i).Metoda\_plata);

DBMS\_OUTPUT.PUT\_LINE(' Total suma plătită: ' || tabel\_date\_clienti(i).Total\_suma\_platita || ' lei');

END IF;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('---------------------------------------');

IF metoda IS NULL THEN

DBMS\_OUTPUT.PUT\_LINE('Client: ' || j.nume || ' ' || j.prenume || ' NU ARE PLATI INREGISTRATE');

END IF;

END LOOP;

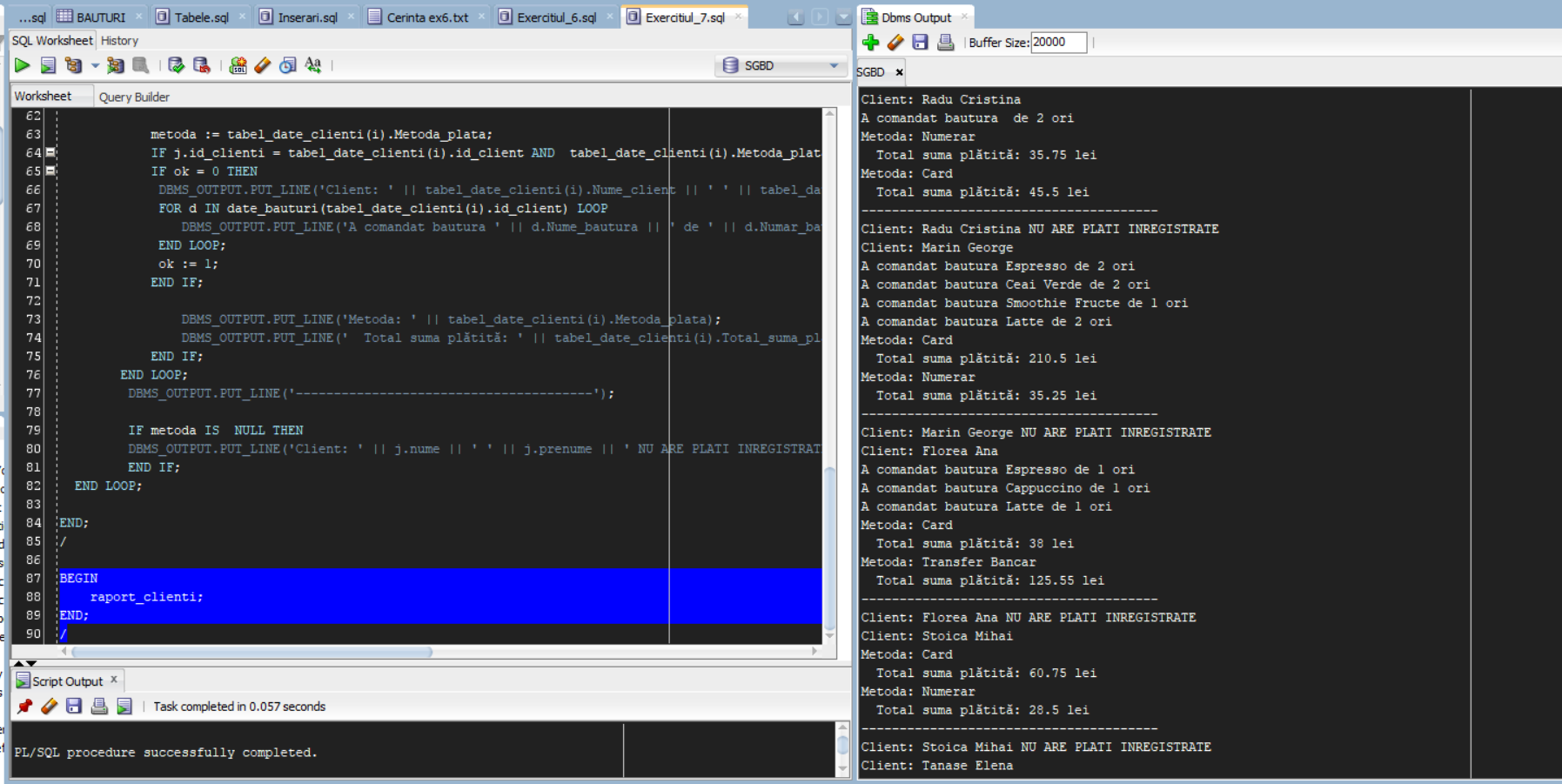
END;

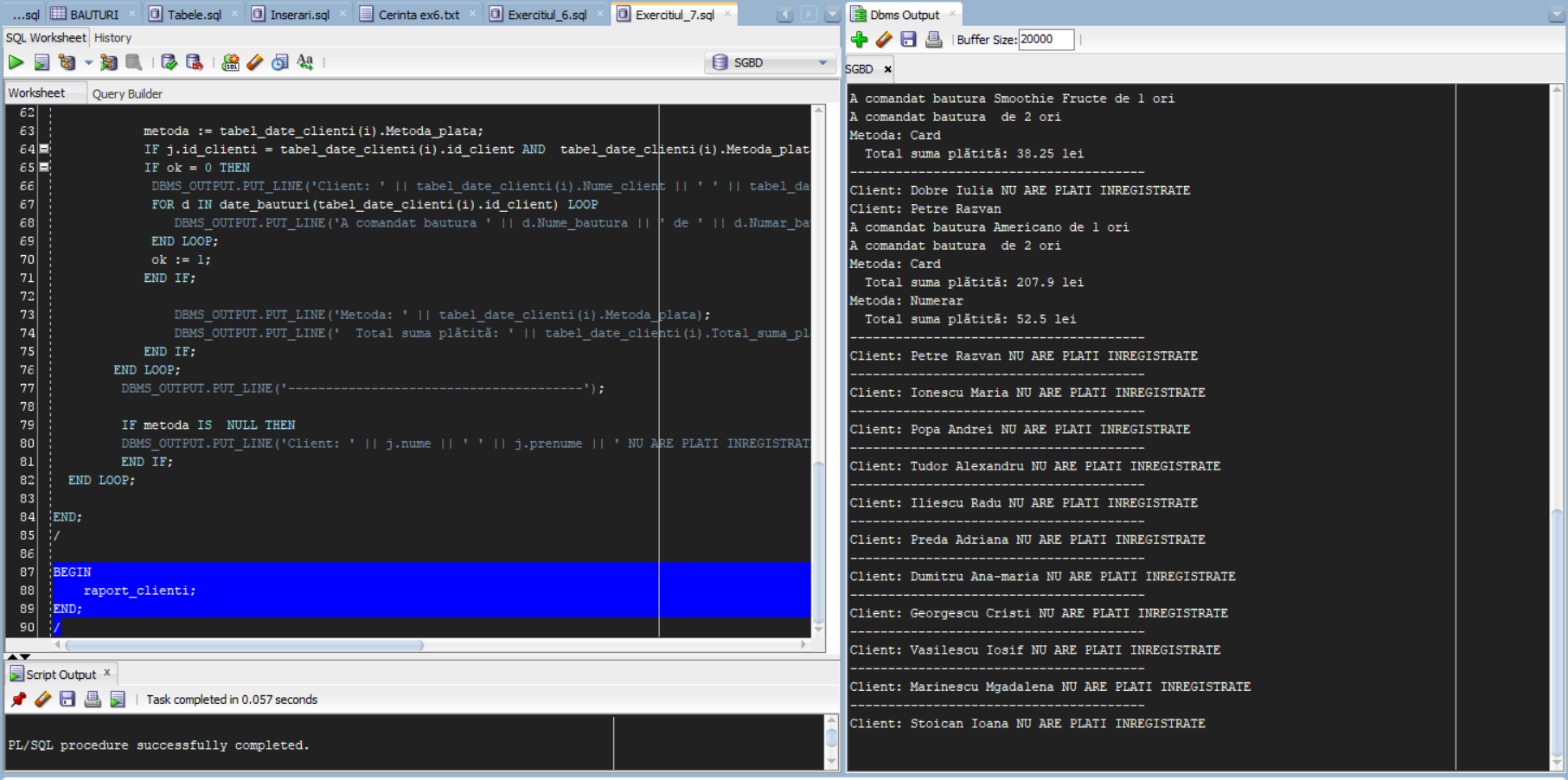
/

BEGIN

raport\_clienti;

END;

/  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
Screenshoturi:  
  




8.Cerinta problemei:  
Scrie o functie PL/SQL numita Situatie\_client, care primeste ca parametru un ID de client si returneaza un mesaj de tip VARCHAR2. Functia trebuie sa calculeze suma totala platita de acel client si sa obtina numele angajatului care a procesat cea mai recenta comanda a clientului.

Daca nu exista date pentru clientul respectiv, functia trebuie sa returneze un mesaj de eroare specific NO\_DATA\_FOUND.

Daca interogarea returneaza mai multe randuri, trebuie gestionata eroarea TOO\_MANY\_ROWS.

CREATE OR REPLACE FUNCTION Situatie\_client

(p\_ID\_Client IN Clienti.ID\_Clienti%TYPE) RETURN VARCHAR2

IS

--Variabile

v\_total\_suma Plati.suma%TYPE := 0;

v\_cel\_mai\_recent\_angajat VARCHAR2(255);

v\_rezultat VARCHAR2(1000);

BEGIN

--Aici selectez suma totala si angajatul care a procesat comanda( cel mai recent)

SELECT SUM(p.suma), a.nume || ' ' || a.prenume

INTO v\_total\_suma, v\_cel\_mai\_recent\_angajat

FROM Plati p

JOIN Comenzi c ON p.ID\_Client = c.ID\_Client

JOIN Angajati a ON c.ID\_Angajat = a.ID\_Angajat

WHERE p.ID\_Client = p\_ID\_Client

GROUP BY a.nume, a.prenume

ORDER BY MAX(c.data\_comanda) DESC;

v\_rezultat := 'Suma totala platita: ' || v\_total\_suma || ' RON. Angajatul cel mai recent implicat: ' || v\_cel\_mai\_recent\_angajat;

RETURN v\_rezultat;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'EROARE: Nu exista inregistrari pentru clientul cu ID-ul ' || p\_ID\_Client || '.';

WHEN TOO\_MANY\_ROWS THEN

RETURN 'EROARE: Interogarea a returnat prea multe randuri pentru clientul cu ID-ul ' || p\_ID\_Client || '.';

WHEN OTHERS THEN

RETURN 'ALTA EROARE: ' || SQLERRM;

END;

/

DECLARE

v\_result VARCHAR2(1000);

BEGIN

-- Caz bun

v\_result := Situatie\_client(2);

DBMS\_OUTPUT.PUT\_LINE(v\_result);

-- Caz NO\_DATA\_FOUND

v\_result := Situatie\_client(999);

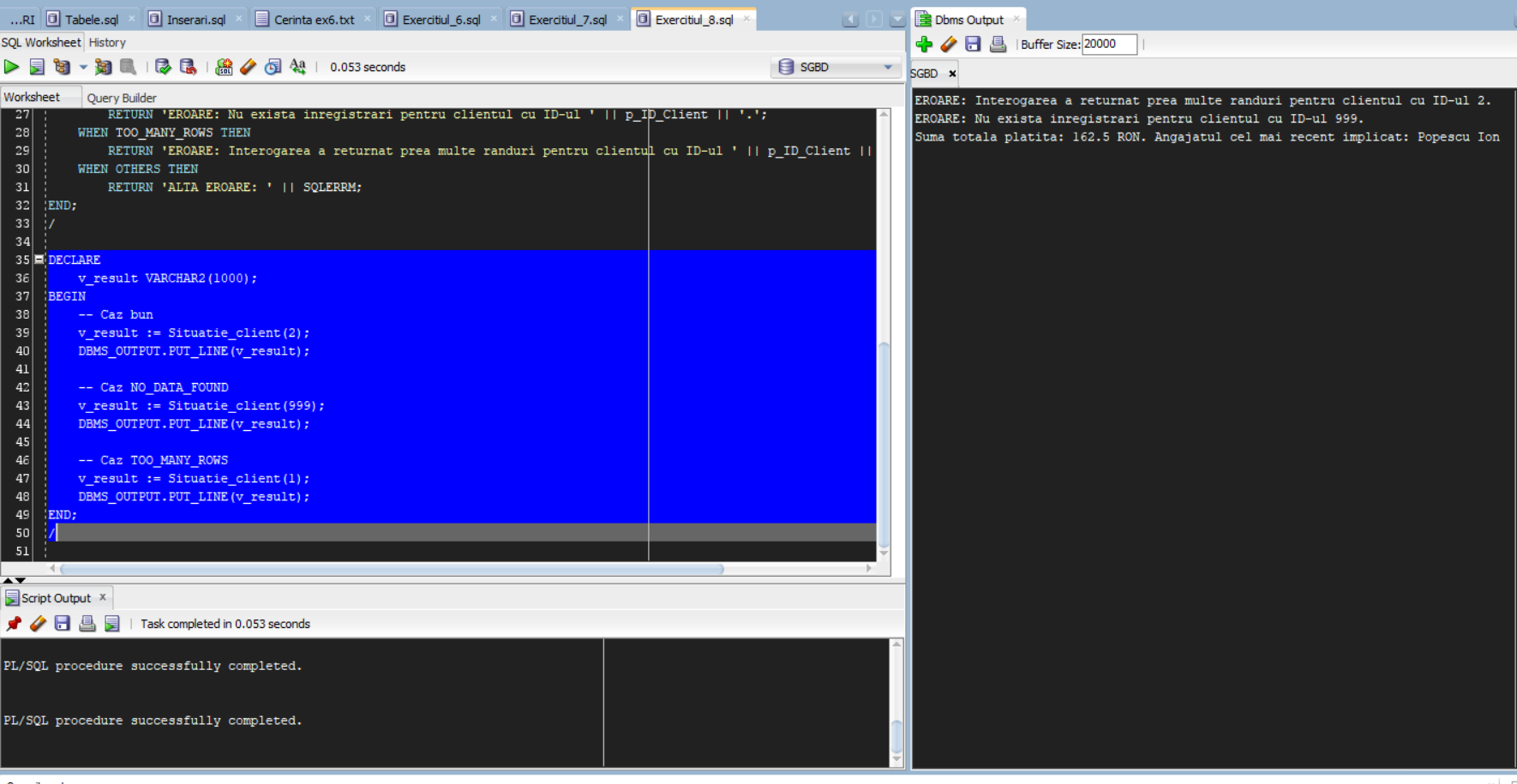
DBMS\_OUTPUT.PUT\_LINE(v\_result);

-- Caz TOO\_MANY\_ROWS

v\_result := Situatie\_client(1);

DBMS\_OUTPUT.PUT\_LINE(v\_result);

END;

/  
  
  
SCREENSHOTURI:  


9.Cerinta problemei:  
Scrie o procedura PL/SQL numita alocare\_bonus\_angajati, care primeste ca parametri o perioada de timp (data\_start si data\_final) si un prag minim de vanzari (valoare\_prag). Procedura trebuie sa afiseze pentru fiecare angajat care a depasit acest prag in perioada specificata, suma totala a vanzarilor realizate si bonusurile acordate.

Bonusurile sunt acordate in functie de tipul lor, iar salariul fiecarui angajat se majoreaza procentual in functie de aceste bonusuri.

Procedura trebuie sa gestioneze urmatoarele exceptii:

1. niciun\_angajat – daca nu exista niciun angajat care sa fi realizat vanzari in perioada specificata.
2. prag\_nedepasit\_de\_nimeni – daca niciun angajat nu a depasit pragul minim specificat.

CREATE OR REPLACE PROCEDURE alocare\_bonus\_angajati(

data\_start IN DATE,

data\_final IN DATE,

valoare\_prag IN NUMBER

)AS

niciun\_angajat EXCEPTION;

prag\_nedepasit\_de\_nimeni EXCEPTION;

TYPE ang IS RECORD

(

valoare\_totala NUMBER(10,2),

nume\_angajat VARCHAR(50),

id\_angajat NUMBER

);

TYPE bon IS RECORD

(

id\_angajat INT,

tip\_bonus VARCHAR(50),

procent NUMBER(10,2)

);

--Nested tables pentru stocarea datelor din crusoare si gestionarea acestora

TYPE t\_date\_angajati IS TABLE OF ang;

TYPE t\_angajati\_bonusuri IS TABLE OF bon;

date\_angajati t\_date\_angajati;

angajati\_bonusuri t\_angajati\_bonusuri;

CURSOR verif\_angajati IS -- Acest cursor l am folosit pur si simlu pentru verificarea exceptiei

SELECT

A.id\_angajat

FROM ANGAJATI A

LEFT JOIN COMENZI C ON C.ID\_angajat = A.id\_angajat

WHERE C.data\_comanda BETWEEN data\_start AND data\_final;

CURSOR conditie\_ang IS -- Cursor pentru a obtine datele relevante depsre angajati ex: suma totala cu care a contribuit fiecare in perioada specificata

SELECT

SUM(B.pret) AS valoare\_totala,

A.nume || ' ' || A.prenume AS Nume\_angajat,

A.id\_angajat

FROM ANGAJATI A

LEFT JOIN COMENZI C ON C.ID\_angajat = A.id\_angajat

LEFT JOIN COMENZI\_CONTIN\_BAUTURI CC ON CC.ID\_comanda = C.ID\_comanda

LEFT JOIN BAUTURI B ON B.ID\_bautura = CC.ID\_bautura

WHERE C.data\_comanda BETWEEN data\_start AND data\_final

GROUP BY A.ID\_angajat, A.nume, A.prenume

HAVING SUM(B.pret) > valoare\_prag;

CURSOR bonus IS --Cursor pentru a vedea ce bonusuri au fiecare angajat

SELECT

A.id\_angajat,

tip\_bonus,

procent

FROM BONUSURI B

JOIN BONUSURI\_ANGAJATI BA ON BA.id\_bonus = B.id\_bonus

JOIN ANGAJATI A ON A.id\_angajat = BA.id\_angajat;

--Variabile

v\_conditie\_ang conditie\_ang%ROWTYPE;

v\_verif\_ang verif\_angajati%ROWTYPE;

BEGIN

OPEN verif\_angajati; --Aici inserez o linie din cursor ca sa vad daca exista macar un angajat

FETCH verif\_angajati INTO v\_verif\_ang;

IF verif\_angajati%NOTFOUND THEN

RAISE niciun\_angajat;

END IF;

CLOSE verif\_angajati; -- Aici inserez o linie din cursor dsa vad daca macar cineva a depasit pragul minim dintre angajati existenti in perioada specificata

OPEN conditie\_ang;

FETCH conditie\_ang INTO v\_conditie\_ang;

IF conditie\_ang%NOTFOUND THEN

RAISE prag\_nedepasit\_de\_nimeni;

END IF;

CLOSE conditie\_ang;

OPEN conditie\_ang; --Introduc in tabel toate informatiile angajatilor din cursorul date\_clienti pe cate o pozitie

FETCH conditie\_ang BULK COLLECT INTO date\_angajati;

CLOSE conditie\_ang;

OPEN bonus;

FETCH bonus BULK COLLECT INTO angajati\_bonusuri;

CLOSE bonus;

FOR i IN 1 .. date\_angajati.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('Angajatul ' || date\_angajati(i).nume\_angajat || ' cu vanzari in valoare de ' || date\_angajati(i).valoare\_totala || 'RON, in perioada ' || TO\_CHAR(data\_start, 'DD-MM-YYYY') ||

' - ' || TO\_CHAR(data\_final, 'DD-MM-YYYY') );

FOR j IN 1 .. angajati\_bonusuri.COUNT LOOP--aici parcurg nested table-ul angajati\_bonusri care este de tip bon

IF angajati\_bonusuri(j).id\_angajat = date\_angajati(i).id\_angajat THEN --Daca angajatul are un anumit bonus si indeplineste conditiile impuse acesta va primi masura cuvenita

IF angajati\_bonusuri(j).tip\_bonus = 'Performance Bonus' THEN

UPDATE ANGAJATI

SET salariu = salariu + salariu \* angajati\_bonusuri(j).procent/100

WHERE id\_angajat = angajati\_bonusuri(j).id\_angajat;

DBMS\_OUTPUT.PUT\_LINE(' - A primit o marire de ' || angajati\_bonusuri(j).procent || '% pentru bonusul Performance Bonus.');

ELSIF angajati\_bonusuri(j).tip\_bonus = 'Customer Satisfaction' THEN

UPDATE ANGAJATI

SET salariu = salariu + salariu \* angajati\_bonusuri(j).procent/100

WHERE id\_angajat = angajati\_bonusuri(j).id\_angajat;

DBMS\_OUTPUT.PUT\_LINE(' - A primit o marire de ' || angajati\_bonusuri(j).procent || '% pentru bonusul Customer Satisfaction.');

ELSIF angajati\_bonusuri(j).tip\_bonus = 'Sales Bonus' THEN

UPDATE ANGAJATI

SET salariu = salariu + salariu \* angajati\_bonusuri(j).procent/100

WHERE id\_angajat = angajati\_bonusuri(j).id\_angajat;

DBMS\_OUTPUT.PUT\_LINE(' - A primit o marire de ' || angajati\_bonusuri(j).procent || '% pentru bonusul Sales Bonus.');

ELSIF angajati\_bonusuri(j).tip\_bonus = 'Team Achievement' THEN

UPDATE ANGAJATI

SET salariu = salariu + salariu \* angajati\_bonusuri(j).procent/100

WHERE id\_angajat = angajati\_bonusuri(j).id\_angajat;

DBMS\_OUTPUT.PUT\_LINE(' - A primit o marire de ' || angajati\_bonusuri(j).procent || '% pentru bonusul Team Achievement.');

END IF;

END IF;

END LOOP;

END LOOP;

EXCEPTION

WHEN niciun\_angajat THEN

DBMS\_OUTPUT.PUT\_LINE('Eroare: Nu exista angajati in perioada specificata.');

WHEN prag\_nedepasit\_de\_nimeni THEN

DBMS\_OUTPUT.PUT\_LINE('Eroare: Nu exista angajati care au depasit pragul minim in perioada specificata.');

END;

/

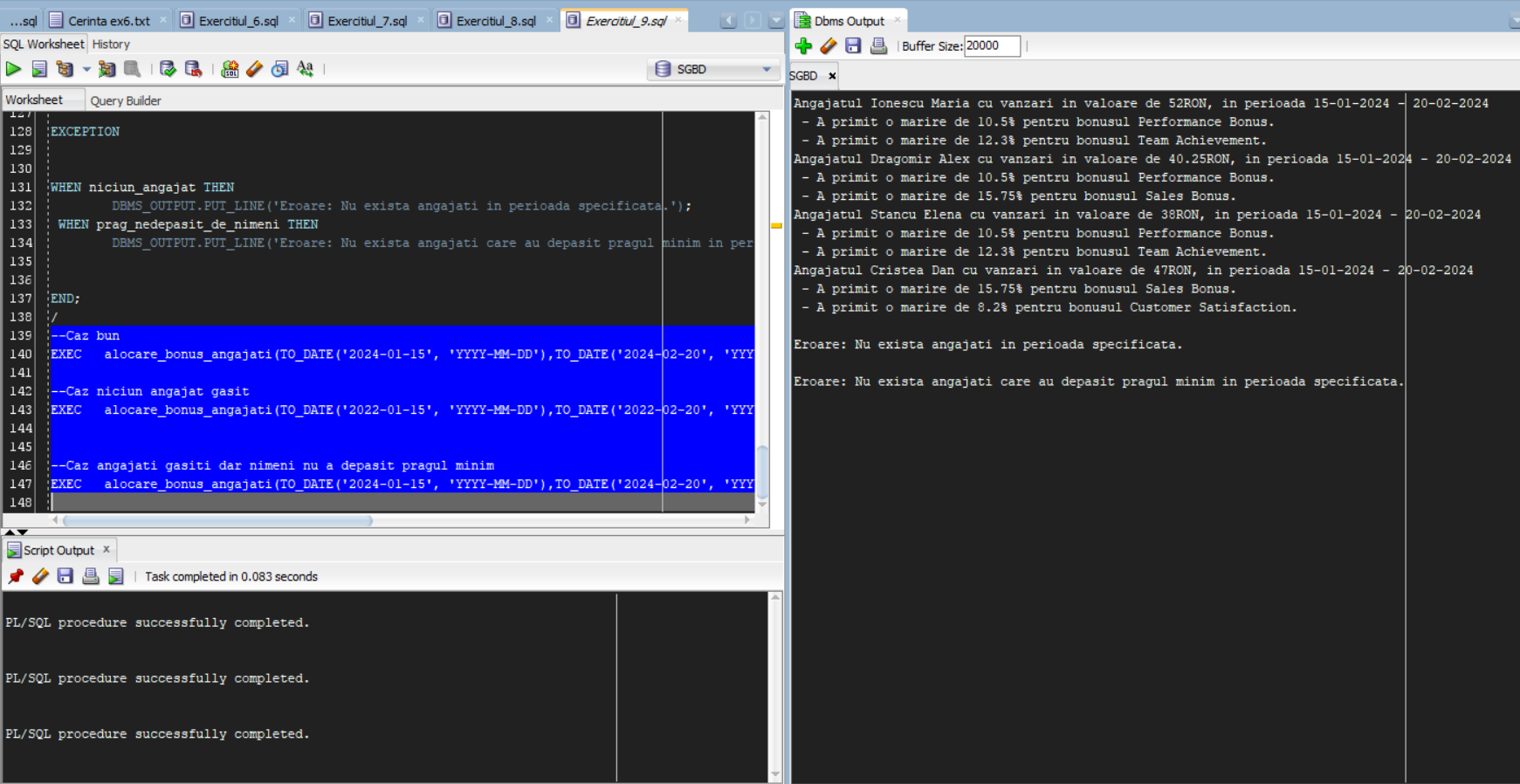
--Caz bun

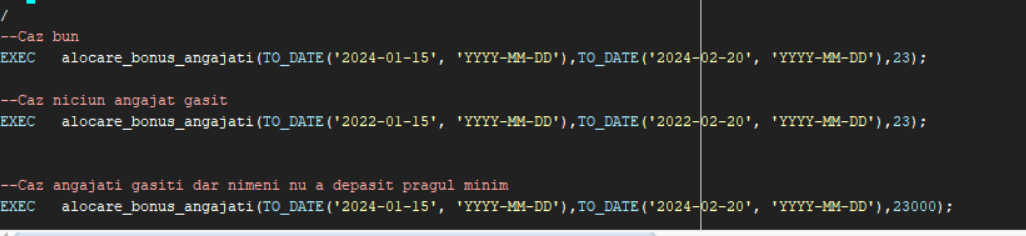
EXEC alocare\_bonus\_angajati(TO\_DATE('2024-01-15', 'YYYY-MM-DD'),TO\_DATE('2024-02-20', 'YYYY-MM-DD'),23);

--Caz niciun angajat gasit

EXEC alocare\_bonus\_angajati(TO\_DATE('2022-01-15', 'YYYY-MM-DD'),TO\_DATE('2022-02-20', 'YYYY-MM-DD'),23);

--Caz angajati gasiti dar nimeni nu a depasit pragul minim

EXEC alocare\_bonus\_angajati(TO\_DATE('2024-01-15', 'YYYY-MM-DD'),TO\_DATE('2024-02-20', 'YYYY-MM-DD'),23000);  




10.Cerinta problemei:  
Creeaza un tabel Statistici\_Bauturi care contine urmatoarele coloane: id\_statistici (cheie primara), numar\_total\_bauturi (numarul total al bauturilor) si pret\_mediu (pretul mediu al bauturilor).

creeaza un trigger date\_bauturi care se declanseaza **dupa o operatie de inserare** in tabela Bauturi. Triggerul trebuie sa calculeze urmatoarele statistici:

1. **Pretul total al bauturilor** – daca depaseste limita de 1000 per total a tuturor bauturilor, declanseaza o eroare folosind
2. **Numarul total de bauturi** si **pretul mediu** al bauturilor – actualizeaza tabela Statistici\_Bauturi cu valorile calculate.

CREATE TABLE Statistici\_Bauturi (

id\_statistici INT PRIMARY KEY,

numar\_total\_bauturi INT,

pret\_mediu DECIMAL(10, 2)

);

CREATE OR REPLACE TRIGGER date\_bauturi

AFTER INSERT ON Bauturi

DECLARE

v\_total\_pret DECIMAL(10, 2);

v\_count INT;

v\_avg\_pret DECIMAL(10, 2);

BEGIN

-- Pretul total al bauturilor

SELECT SUM(pret) INTO v\_total\_pret FROM Bauturi;

-- Restrictie pret total

IF v\_total\_pret > 1000 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Pretul total al bauturilor depaseste limita de 1000');

END IF;

--Nr total bauturi si pret mediu

SELECT COUNT(\*), AVG(pret)

INTO v\_count, v\_avg\_pret

FROM Bauturi;

UPDATE Statistici\_Bauturi

SET numar\_total\_bauturi = v\_count,

pret\_mediu = v\_avg\_pret

WHERE id\_statistici = 1;

END;

/

DECLARE

exceptie\_unicitate EXCEPTION;

PRAGMA EXCEPTION\_INIT(exceptie\_unicitate, -1);

BEGIN

INSERT INTO Bauturi (ID\_Bautura, nume, tip, pret) VALUES (9, 'Suc coacaze', 'Fresh', 18);

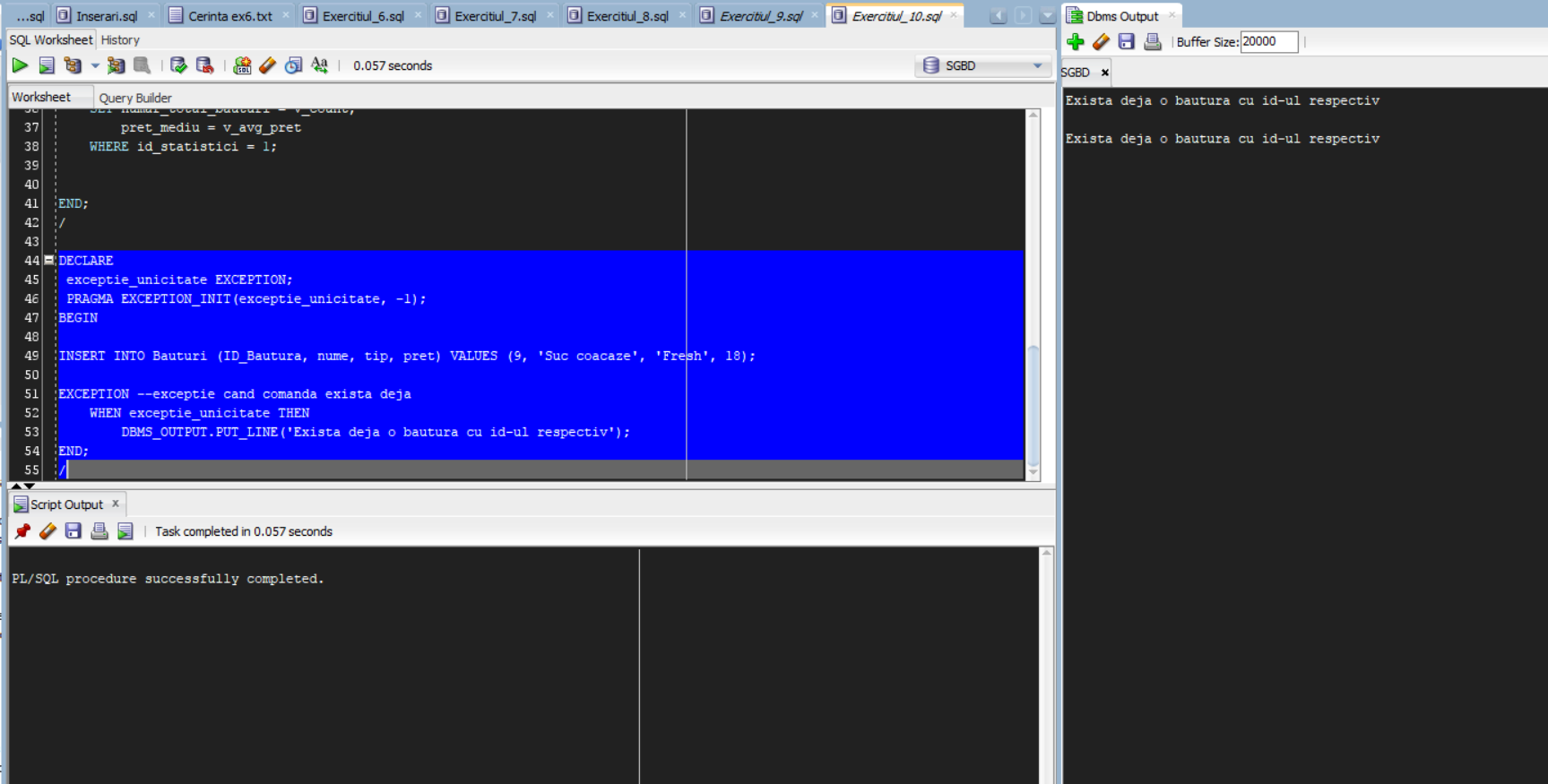
EXCEPTION --exceptie cand comanda exista deja

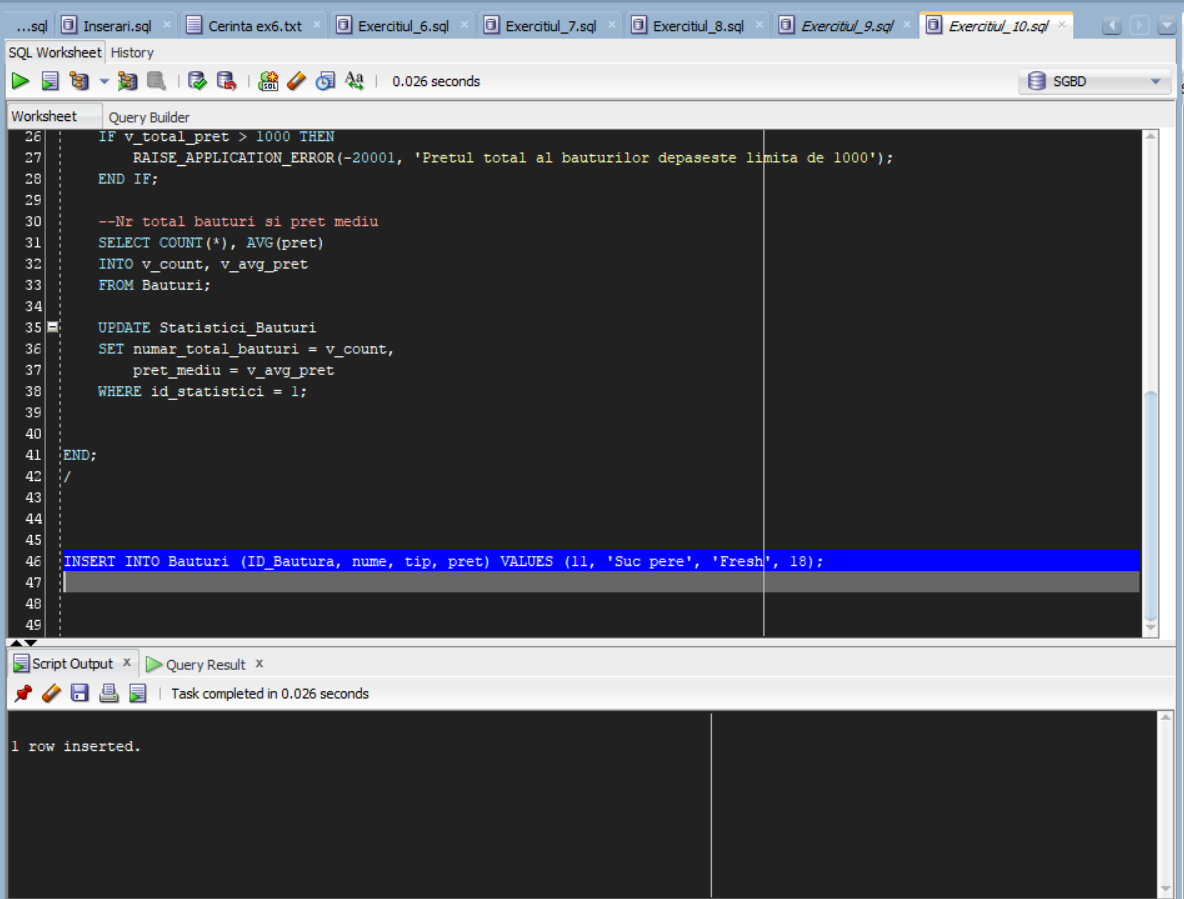
WHEN exceptie\_unicitate THEN

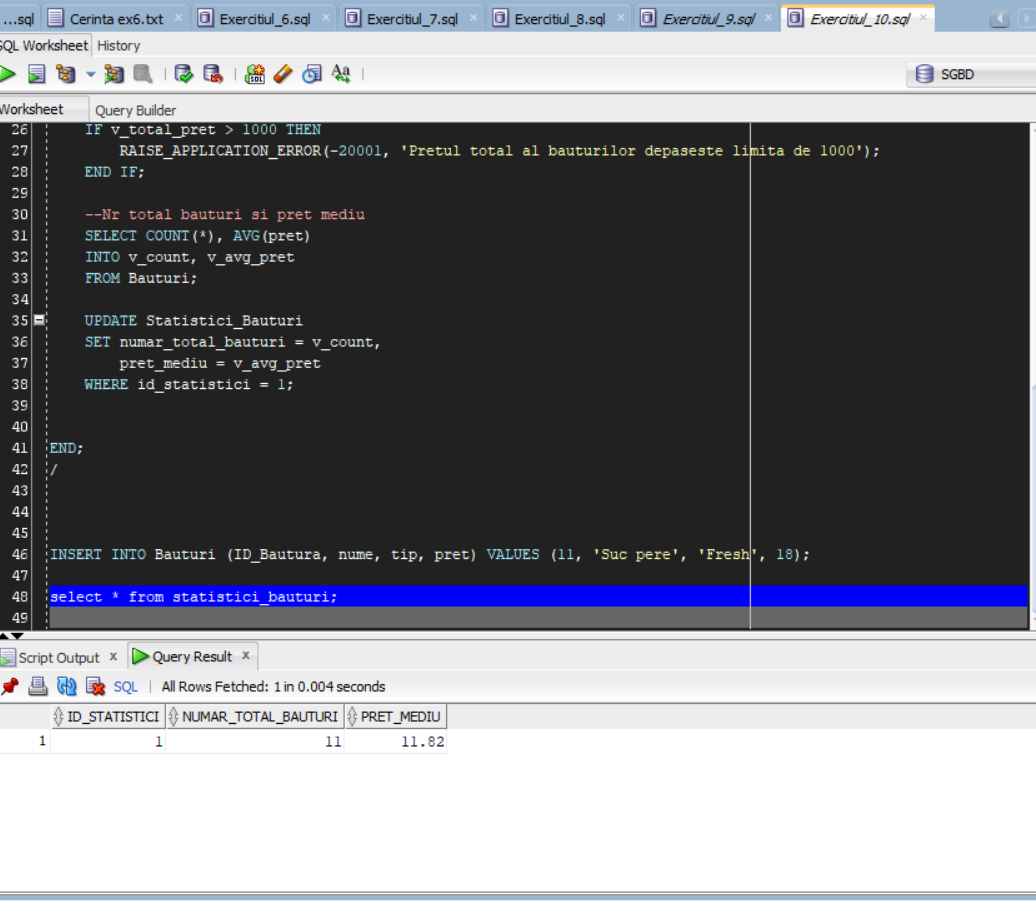
DBMS\_OUTPUT.PUT\_LINE('Exista deja o bautura cu id-ul respectiv');

END;

/







11.

Cerinta problemei:

Creeaza un trigger PL/SQL denumit plati\_clienti\_comanda, care se declanseaza dupa o operatie de **UPDATE** sau **DELETE** asupra tabelei Comenzi. Acest trigger trebuie sa actualizeze suma totala cheltuita de client in tabela Clienti, in functie de modificarile efectuate asupra comenzilor.

1. **In cazul unui UPDATE**:

Verifica daca a fost modificat ID\_Client sau ID\_Comanda.

Calculeaza suma totala a bauturilor din comanda veche si comanda noua.

Actualizeaza suma totala cheltuita de clientul vechi prin scaderea sumei vechi.

Actualizeaza suma totala cheltuita de clientul nou prin adaugarea sumei noi si scaderea sumei vechi (daca exista).

1. **In cazul unui DELETE**:

Calculeaza suma totala a bauturilor din comanda stearsa.

Actualizeaza suma totala cheltuita de client prin scaderea sumei respective.

Verifica daca clientul mai are comenzi. Daca nu mai are, sterge clientul din tabelul Clienti.

CREATE OR REPLACE TRIGGER plati\_clienti\_comanda

AFTER UPDATE OR DELETE ON Comenzi

FOR EACH ROW

DECLARE

v\_suma\_veche NUMBER(10, 2);

v\_suma\_noua NUMBER(10, 2);

v\_comenzi\_count INT;

BEGIN

-- In cazul unui UPDATE

IF UPDATING AND (:OLD.ID\_Client != :NEW.ID\_Client OR :OLD.ID\_Comanda != :NEW.ID\_Comanda) THEN

DBMS\_OUTPUT.PUT\_LINE('UPDATE detectat: Comanda ' || :OLD.ID\_Comanda || ' a fost modificata.');

-- Suma comenzii vechi

SELECT SUM(b.pret)

INTO v\_suma\_veche

FROM Comenzi\_contin\_bauturi cb

JOIN Bauturi b ON cb.ID\_Bautura = b.ID\_Bautura

WHERE cb.ID\_Comanda = :OLD.ID\_Comanda;

-- Suma comenzii noi

SELECT SUM(b.pret)

INTO v\_suma\_noua

FROM Comenzi\_contin\_bauturi cb

JOIN Bauturi b ON cb.ID\_Bautura = b.ID\_Bautura

WHERE cb.ID\_Comanda = :NEW.ID\_Comanda;

-- Actualizeaza suma clientului vechi

UPDATE Clienti

SET suma\_totala = suma\_totala - NVL(v\_suma\_veche, 0)

WHERE ID\_Clienti = :OLD.ID\_Client;

DBMS\_OUTPUT.PUT\_LINE('Clientul vechi cu id-ul ' || :OLD.ID\_Client || ' a fost actualizat: suma scazuta cu ' || NVL(v\_suma\_veche, 0) || ' RON.');

-- Actualizeaza suma clientului nou

UPDATE Clienti

SET suma\_totala = suma\_totala - NVL(v\_suma\_veche, 0) + NVL(v\_suma\_noua, 0)

WHERE ID\_Clienti = :NEW.ID\_Client;

DBMS\_OUTPUT.PUT\_LINE('Clientul nou cu id-ul ' || :NEW.ID\_Client || ' a fost actualizat: suma adaugata cu ' || NVL(v\_suma\_noua, 0) || ' RON.');

END IF;

-- In cazul unui DELETE

IF DELETING THEN

DBMS\_OUTPUT.PUT\_LINE('DELETE detectat: Comanda ' || :OLD.ID\_Comanda || ' a fost stearsa.');

-- Suma comenzii vechi

SELECT SUM(b.pret)

INTO v\_suma\_veche

FROM Comenzi\_contin\_bauturi cb

JOIN Bauturi b ON cb.ID\_Bautura = b.ID\_Bautura

WHERE cb.ID\_Comanda = :OLD.ID\_Comanda;

-- Actualizeaza suma clientului

UPDATE Clienti

SET suma\_totala = suma\_totala - NVL(v\_suma\_veche, 0)

WHERE ID\_Clienti = :OLD.ID\_Client;

DBMS\_OUTPUT.PUT\_LINE('Clientul cu id-ul ' || :OLD.ID\_Client || ' a fost actualizat: suma scazuta cu ' || NVL(v\_suma\_veche, 0) || ' RON.');

-- Verifica daca mai exista comenzi pentru client

SELECT COUNT(\*) INTO v\_comenzi\_count

FROM Comenzi

WHERE ID\_Client = :OLD.ID\_Client;

IF v\_comenzi\_count = 0 THEN

DELETE FROM Clienti

WHERE ID\_Clienti = :OLD.ID\_Client;

DBMS\_OUTPUT.PUT\_LINE('Clientul cu id-ul ' || :OLD.ID\_Client || ' a fost sters deoarece nu mai are comenzi.');

END IF;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Eroare: Nicio comanda gasita pentru client.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Alta eroare: ' || SQLERRM);

END;

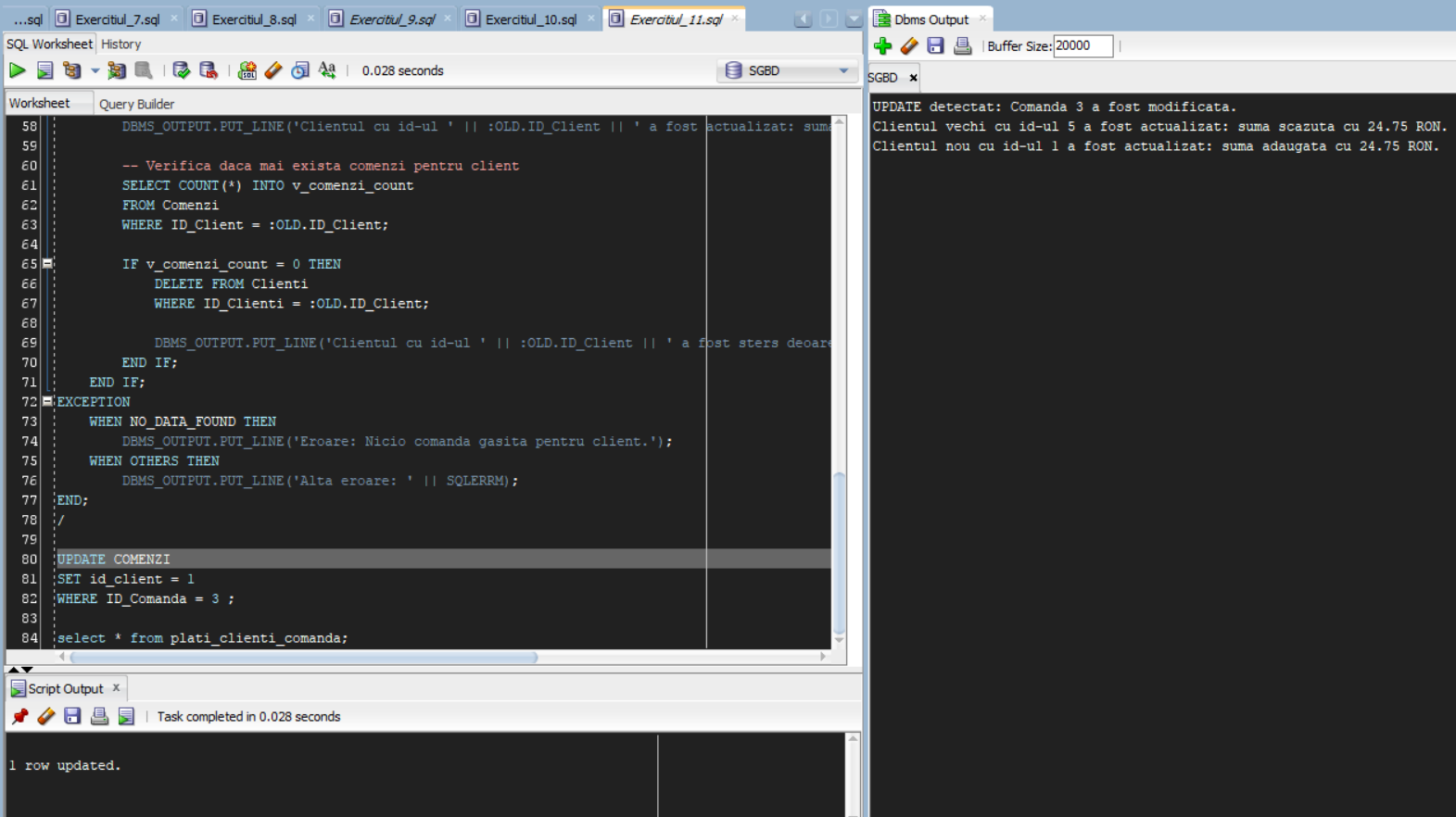
/

UPDATE COMENZI

SET id\_client = 1

WHERE ID\_Comanda = 3 ;

SCREENSHOTURI:



12.

Cerinta problemei:

Creeaza un trigger numit Monitorizare\_Activitati\_tabel care se activeaza automat dupa orice operatie de tip **CREATE**, **ALTER** sau **DROP** asupra obiectelor din schema utilizatorului. Triggerul trebuie sa respecte urmatoarele cerinte:

Daca operatia este de tip **CREATE** si obiectul creat este un tabel, atunci triggerul trebuie sa verifice daca numele tabelului incepe cu prefixul ‘ TEST\_ ‘, daca nu, operatia trebuie sa fie blocata

Pentru fiecare operatie (**CREATE**, **ALTER**, **DROP**) asupra oricarui tip de obiect triggerul trebuie sa insereze un eveniment intr-un tabel de log denumit Log\_Activitati, care contine urmatoarele informatii:

**Data evenimentului** – data si ora la care s-a efectuat operatia.

**Numele utilizatorului** – utilizatorul care a efectuat operatia.

**Tipul operatiei** – CREATE, ALTER sau DROP.

**Tipul obiectului** – TABLE, VIEW, INDEX, etc.

**Numele obiectului** – numele obiectului asupra caruia s-a efectuat operatia.

**Descrierea operatiei** – un text care sa includa utilizatorul, operatia, tipul si numele obiectului.

/\*CREATE TABLE Log\_Activitati (

id\_eveniment INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

data\_eveniment TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

user\_name VARCHAR2(50),

tip\_operatie VARCHAR2(50),

tip\_obiect VARCHAR2(50),

nume\_obiect VARCHAR2(255),

descriere\_operatie VARCHAR2(500)

);

\*/

CREATE OR REPLACE TRIGGER Monitorizare\_Activitati\_tabel

AFTER CREATE OR ALTER OR DROP ON SCHEMA

DECLARE

descriere VARCHAR2(500);

BEGIN

--Aici am creat o conditie ca orice nou tabel creat de utilizator sa inceapa cu test\_

IF ORA\_SYSEVENT = 'CREATE' AND ORA\_DICT\_OBJ\_TYPE = 'TABLE' THEN

IF NOT ORA\_DICT\_OBJ\_NAME LIKE 'TEST\_%' THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Numele tabelului trebuie să înceapă cu TEST\_!!!');

END IF;

END IF;

--am pus toate datele intr-o singura variabile

descriere := 'Utilizatorul ' || USER || ' a efectuat operația ' || ORA\_SYSEVENT ||' asupra obiectului de tip ' || ORA\_DICT\_OBJ\_TYPE || ' numit ' || ORA\_DICT\_OBJ\_NAME || '.';

INSERT INTO Log\_Activitati(data\_eveniment, user\_name, tip\_operatie, tip\_obiect, nume\_obiect,descriere\_operatie)

VALUES (

SYSDATE,

USER,

ORA\_SYSEVENT,

ORA\_DICT\_OBJ\_TYPE,

ORA\_DICT\_OBJ\_NAME,

descriere

);

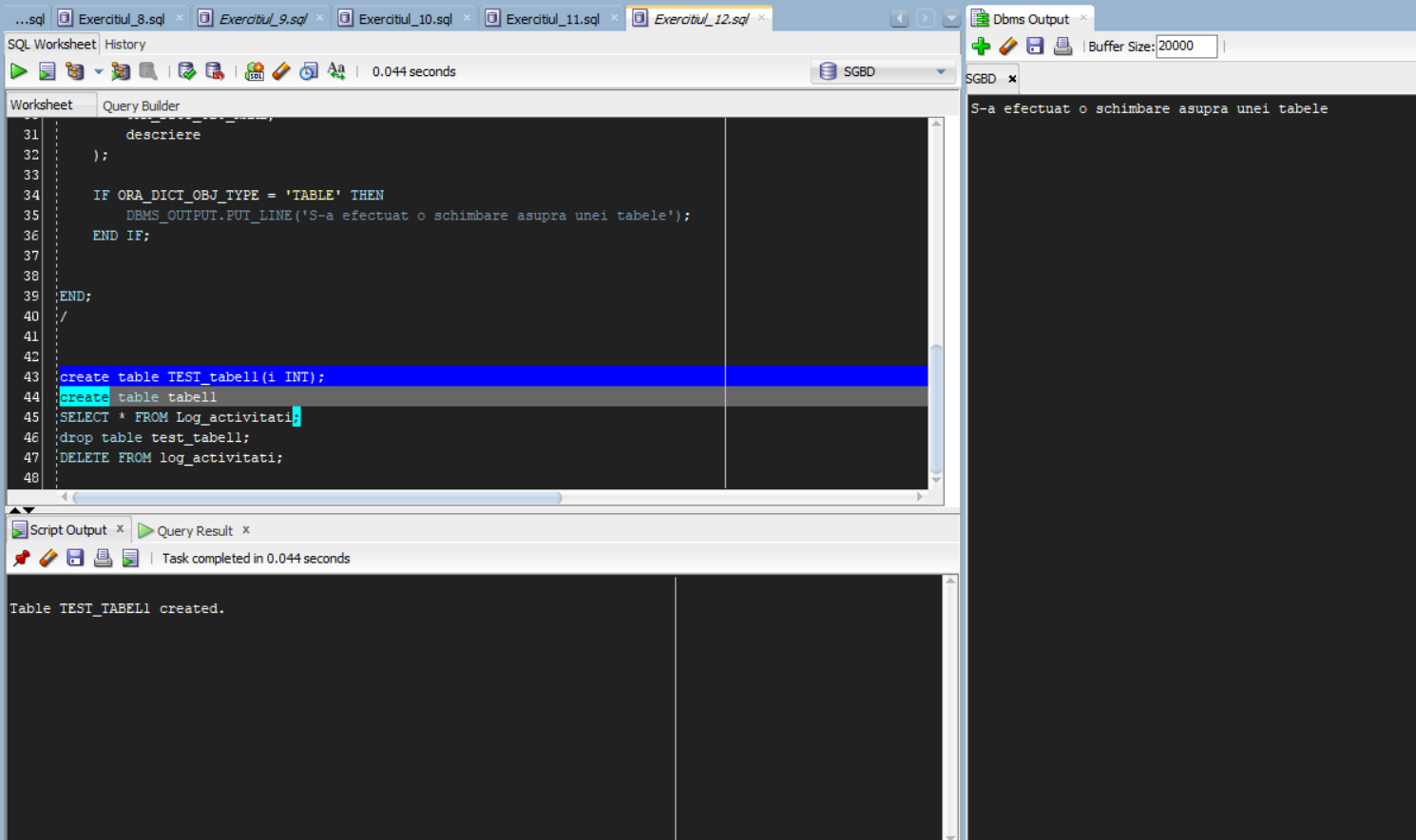
IF ORA\_DICT\_OBJ\_TYPE = 'TABLE' THEN

DBMS\_OUTPUT.PUT\_LINE('S-a efectuat o schimbare asupra unei tabele');

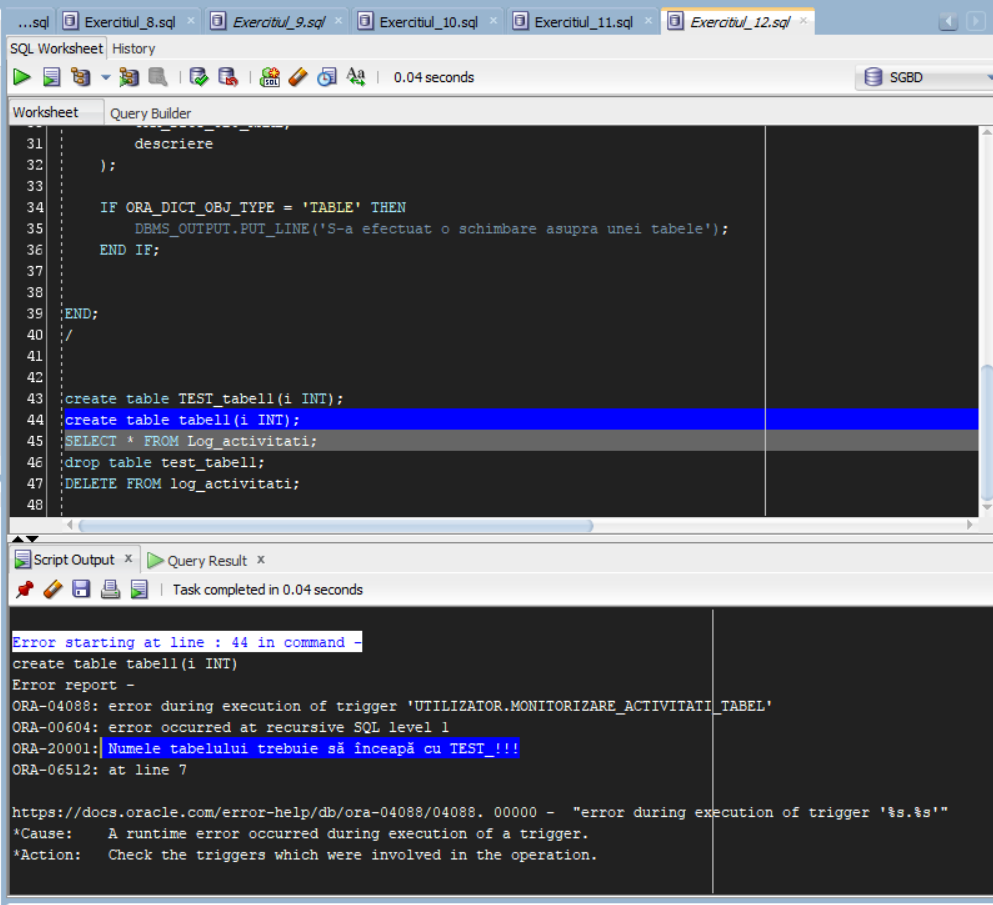
END IF;

END;

/







13.

Cerinta problemei:

Scrie un program PL/SQL care sa gestioneze procesul de creare si afisare a bonurilor fiscale pe baza comenzilor dintr-o baza de date. Programul trebuie sa utilizeze un pachet PL/SQL definit anterior, care ofera functionalitati pentru calcularea pretului unei comenzi, generarea bonurilor fiscale si compararea comenzilor in functie de valoare in RON a acestora.

1.Sa se citeasca de la tastatura datele necesare pentru o comanda si sa se insereze in tabelele respective datele, iar apoi sa se creeze un bon fiscal in baza de date cu datele inserate.

2.Sa se afiseze comanda inserata sub forma unui bon fiscal , iar dupa sa se afiseze comanda cu valoarea cea mai mare dintre toate cele existente ,inclusiv cea adaugata de tine.

3.De asemenea sa se rezolve exceptia de unicitate de unei comenzi in cazul in care utilizatorul insereaza o comanda cu acelasi id, sa se afiseze un mesaj corespunzator.

**PACHETUL:**

CREATE OR REPLACE PACKAGE gestionare\_bonusuri\_comenzi AS

-- Tipuri de date complexe

TYPE bon\_date IS RECORD(

client\_nume VARCHAR2(255),

client\_prenume VARCHAR2(255),

angajat\_nume VARCHAR2(255),

angajat\_prenume VARCHAR2(255),

bauturi VARCHAR2(4000),

suma\_plata NUMBER(10, 2),

data\_comanda DATE

);

CURSOR nume\_bauturi IS

SELECT

C.id\_comanda,

C.data\_comanda,

B.nume

FROM COMENZI C

JOIN COMENZI\_CONTIN\_BAUTURI ccb ON ccb.ID\_COMANDA = C.ID\_COMANDA

JOIN BAUTURI B ON ccb.ID\_BAUTURA = B.ID\_BAUTURA

GROUP BY B.nume,C.id\_comanda,C.data\_comanda;

TYPE IduriComanda IS TABLE OF INT;

TYPE tabel\_bon IS TABLE OF bon\_date;

--Functii

FUNCTION CalculeazaPretComanda(idul\_comenzii IN INT) RETURN NUMBER;

FUNCTION ComparaBonuri(data\_start IN DATE, data\_final IN DATE) RETURN BON\_DATE;

--Proceduri

PROCEDURE GenereazaBonFiscal(idul\_comenzii IN INT);

PROCEDURE AfisareBonuri (iduri IN IduriComanda);

END gestionare\_bonusuri\_comenzi;

/

CREATE OR REPLACE PACKAGE BODY gestionare\_bonusuri\_comenzi AS

FUNCTION ComparaBonuri(data\_start IN DATE, data\_final IN DATE) RETURN BON\_DATE IS

bonuri tabel\_bon := tabel\_bon();

bon\_cel\_mai\_scump bon\_date;

bon\_client\_frecvent bon\_date;

max\_suma NUMBER(10, 2) := 0;

client\_max\_bonuri VARCHAR2(255);

ok NUMBER(2) := 0;

bautura VARCHAR(400);

BEGIN

FOR bon\_rec IN ( --aici selectez toate datele care se afla pe un bon si le pun in nested table de tipul bon\_date in ordine descrescatoare dupa pretul comenzii

SELECT cl.nume AS client\_nume,

cl.prenume AS client\_prenume,

ag.nume AS angajat\_nume,

ag.prenume AS angajat\_prenume,

SUM(b.pret) AS suma\_plata,

c.data\_comanda

FROM Comenzi c

JOIN Clienti cl ON c.ID\_Client = cl.ID\_Clienti

JOIN Angajati ag ON c.ID\_Angajat = ag.ID\_Angajat

JOIN Comenzi\_contin\_bauturi ccb ON c.ID\_Comanda = ccb.ID\_Comanda

JOIN Bauturi b ON ccb.ID\_Bautura = b.ID\_Bautura

GROUP BY cl.nume, cl.prenume, ag.nume, ag.prenume, c.data\_comanda

ORDER BY SUM(b.pret) DESC

) LOOP

IF ( bon\_rec.DATA\_COMANDA >= data\_start AND bon\_rec.DATA\_COMANDA <= data\_final ) THEN -- Daca

bonuri.EXTEND(1);

bonuri(bonuri.COUNT) := bon\_date(

client\_nume => bon\_rec.client\_nume,

client\_prenume => bon\_rec.client\_prenume,

angajat\_nume => bon\_rec.angajat\_nume,

angajat\_prenume => bon\_rec.angajat\_prenume,

suma\_plata => bon\_rec.suma\_plata,

data\_comanda => bon\_rec.data\_comanda);

FOR baut IN nume\_bauturi LOOP -- aici mai adaug si separat bauturile

IF baut.data\_comanda = bon\_rec.data\_comanda THEN

OK := 1;

bautura := bautura || baut.nume || ', ';

END IF;

END LOOP;

IF OK = 0 THEN -- daca nu exista nicio bautura se va pune un mesaj coresp

bautura := 'Nicio bautura';

end if;

bonuri(bonuri.COUNT).bauturi := bautura;

END IF;

END LOOP;

IF bonuri.COUNT > 0 THEN -- Daca exista cel putin un bon altfel eroare

RETURN bonuri(1);

ELSE

RAISE\_APPLICATION\_ERROR(-20001, 'Niciun bon');

END IF;

END ComparaBonuri;

--##############################################################################--

FUNCTION CalculeazaPretComanda(idul\_comenzii IN INT) RETURN NUMBER IS

pret\_total NUMBER(10, 2) := 0;

BEGIN

SELECT SUM(b.pret) --Aiici pretul bauturii

INTO pret\_total

FROM Comenzi\_contin\_bauturi ccb

JOIN Bauturi b ON ccb.ID\_Bautura = b.ID\_Bautura

WHERE ccb.ID\_Comanda = idul\_comenzii;

IF pret\_total IS NULL THEN

pret\_total := 0;

END IF;

DBMS\_OUTPUT.PUT\_LINE('Pretul total pentru comanda ' || idul\_comenzii || ' este: ' || pret\_total);

RETURN pret\_total;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

RAISE;

END CalculeazaPretComanda;

--##############################################################################--

PROCEDURE GenereazaBonFiscal(idul\_comenzii IN INT) IS

id\_bon INT;

descriere bonuri.descriere%TYPE;

bon\_descriere bon\_date;

OK NUMBER(2) := 0;

LINE\_SEPARATOR VARCHAR2(100) := '----------------------------------------';

LINE\_SEPARATOR2 VARCHAR2(100) := '#######################################';

BEGIN

id\_bon := BONURI\_SEQ.NEXTVAL;

SELECT cl.nume AS client\_nume, --aici extrag toate datele pentru bon

cl.prenume AS client\_prenume,

ag.nume AS angajat\_nume,

ag.prenume AS angajat\_prenume,

c.data\_comanda

INTO bon\_descriere.client\_nume,

bon\_descriere.client\_prenume,

bon\_descriere.angajat\_nume,

bon\_descriere.angajat\_prenume,

bon\_descriere.data\_comanda

FROM Comenzi c

JOIN Clienti cl ON c.ID\_Client = cl.ID\_Clienti

JOIN Angajati ag ON c.ID\_Angajat = ag.ID\_Angajat

LEFT JOIN Comenzi\_contin\_bauturi ccb ON c.ID\_Comanda = ccb.ID\_Comanda

LEFT JOIN Bauturi b ON ccb.ID\_Bautura = b.ID\_Bautura

WHERE c.ID\_Comanda = idul\_comenzii

GROUP BY cl.nume, cl.prenume, ag.nume, ag.prenume, c.data\_comanda;

FOR baut IN nume\_bauturi LOOP --Iar aici pun bauturile

IF baut.id\_comanda = idul\_comenzii THEN

OK := 1;

bon\_descriere.bauturi := bon\_descriere.bauturi || baut.nume || ', ';

END IF;

END LOOP;

IF OK = 0 THEN

bon\_descriere.bauturi := 'Nicio bautura';

end if;

bon\_descriere.suma\_plata := CalculeazaPretComanda(idul\_comenzii);

descriere := CHR(10) || LINE\_SEPARATOR2 || CHR(10) || -- aici formez bonul

' BON FISCAL (Comanda '|| idul\_comenzii || ')' || CHR(10) ||

LINE\_SEPARATOR || CHR(10) ||

'Client: ' || bon\_descriere.client\_nume || ' ' || bon\_descriere.client\_prenume || CHR(10) ||

'Angajat: ' || bon\_descriere.angajat\_nume || ' ' || bon\_descriere.angajat\_prenume || CHR(10) ||

'Data: ' || TO\_CHAR(bon\_descriere.data\_comanda, 'DD-MM-YYYY') || CHR(10) ||

LINE\_SEPARATOR || CHR(10) ||

'Bauturi:' || CHR(10) ||

bon\_descriere.bauturi || CHR(10) ||

LINE\_SEPARATOR || CHR(10) ||

'TOTAL: ' || TO\_CHAR(bon\_descriere.suma\_plata, '999,990.00') || ' RON' || CHR(10) ||

LINE\_SEPARATOR2;

INSERT INTO Bonuri (ID\_Bon, Data\_Bon, ID\_Comanda,DESCRIERE)VALUES (id\_bon, bon\_descriere.data\_comanda, idul\_comenzii, descriere);

-- Se poate adauga si logica pentru actualizarea stocurilor, daca e cazul.

END GenereazaBonFiscal;

--#########################################################---------

PROCEDURE AfisareBonuri (iduri IN IduriComanda) IS

descriere\_afis bonuri.descriere%TYPE;

BEGIN

FOR i IN iduri.FIRST..iduri.LAST LOOP

BEGIN

SELECT descriere

INTO descriere\_afis

FROM BONURI

WHERE iduri(i) = ID\_Comanda;

DBMS\_OUTPUT.PUT\_LINE(descriere\_afis);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN --eroare ca nu gaseste comanda dupa id

RAISE\_APPLICATION\_ERROR( -20001,'Comanda cu ID-UL ' || iduri(i) || ' nu a fost gasita.');

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20002,'Eroare la afisarea comenzii ' || iduri(i) || ': ' || SQLERRM

);

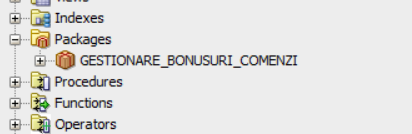
END;

END LOOP;

END AfisareBonuri;

END gestionare\_bonusuri\_comenzi;

/



Problema:

DECLARE

iduri\_comenzi gestionare\_bonusuri\_comenzi.IduriComanda;

bon gestionare\_bonusuri\_comenzi.bon\_date;

v\_client\_id INT;

v\_angajat\_id INT;

v\_bautura\_id INT;

v\_comanda\_id INT;

v\_data\_comanda DATE := SYSDATE;

exceptie\_unicitate EXCEPTION;

PRAGMA EXCEPTION\_INIT(exceptie\_unicitate, -1);

BEGIN

v\_client\_id := &ID\_CLIENT;

v\_angajat\_id := &ID\_ANGAJAT;

v\_comanda\_id := &ID\_COMANDA;

v\_bautura\_id := &ID\_BAUTURA;

BEGIN --inserez datele mai intai in comenzi si dupa in comenzi contin bauturi

INSERT INTO Comenzi (ID\_Comanda, ID\_Client, ID\_Angajat, Data\_Comanda)VALUES (v\_comanda\_id, v\_client\_id, v\_angajat\_id, v\_data\_comanda);

END;

BEGIN

INSERT INTO Comenzi\_contin\_bauturi (ID\_Comanda, ID\_Bautura)VALUES (v\_comanda\_id, v\_bautura\_id);

END;

--Aici generez bonul fiscal

gestionare\_bonusuri\_comenzi.GenereazaBonFiscal(v\_comanda\_id);

DBMS\_OUTPUT.PUT\_LINE('Bonul fiscal pentru comanda ' || v\_comanda\_id || ' a fost generat.');

--Aici il afisez

iduri\_comenzi := gestionare\_bonusuri\_comenzi.IduriComanda(v\_comanda\_id);

gestionare\_bonusuri\_comenzi.AFISAREBONURI(iduri\_comenzi);

--Aici obtin comanda cu cel mai mare pret dintre doua date specificate

bon := gestionare\_bonusuri\_comenzi.ComparaBonuri(TO\_DATE('15-01-2024','DD-MM-YYYY'),SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Bonul cu cea mai mare suma: ' || bon.client\_nume || ', Angajat: ' || bon.client\_prenume || ' '

|| bon.angajat\_nume || ' ' || bon.angajat\_prenume || ' ' || bon.bauturi || ' In valoare de '

|| bon.suma\_plata || ', in data de ' || bon.data\_comanda);

EXCEPTION --exceptie cand comanda exista deja

WHEN exceptie\_unicitate THEN

DBMS\_OUTPUT.PUT\_LINE('COMANDA CU ID-UL ' || v\_comanda\_id || ' EXISTA DEJA!!');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Alta eroare' || SQLERRM);

END;

/

