**Proiect SGBD**

**Baza de Date a aplicației Chess.com**

Realizat de Dobrin Ionuț-Alexandru

Grupa 234

Anul universitar 2023-2024

**Cuprins**

1. Prezentarea pe scurt a bazei de date (utilitatea ei)

2. Realizați diagrama entitate-relație

3. Realizați diagrama conceptuală + entități, relații și atribute

4. Implementarea în Oracle a diagramei conceptuale

5. Adăugarea informațiilor coerente în tabelele create

6. Rezolvarea unei probleme pe care o rezolv folosind un subprogram stocat independent care să utilizeze toate cele 3 tipuri de colecții + apelarea subprogramului

7. Rezolvarea unei probleme pe care o rezolv folosind un subprogram stocat independent care să utilizeze 2 tipuri diferite de cursoare, unul dintre acestea fiind cursor parametrizat, dependent de celălalt cursor + apelarea subprogramului

8. Rezolvarea unei probleme pe care o rezolv folosind un subprogram stocat independent de tip funcție care să utilizeze într-o singură comandă SQL 3 dintre tabelele definite + definirea a minim 2 excepții proprii + apelarea subprogramul astfel încât să evidențiez toate cazurile definite și tratate

9. Rezolvarea unei probleme pe care o rezolv folosind un subprogram stocat independent de tip procedură care să utilizeze într-o singură comandă SQL 5 dintre tabelele definite + tratarea tuturor excepțiilor care pot apărea, incluzând excepțiile NO\_DATA\_FOUND și TOO\_MANY\_ROWS + apelarea subprogramului astfel încât să evidențiez toate cazurile tratate

10. Definirea unui trigger de tip LMD la nivel de comandă + declanșarea trigger-ului

11. Definirea unui trigger de tip LMD la nivel de linie + declanșarea trigger-ului

12. Definirea unui trigger de tip LDD + declanșarea trigger-ului

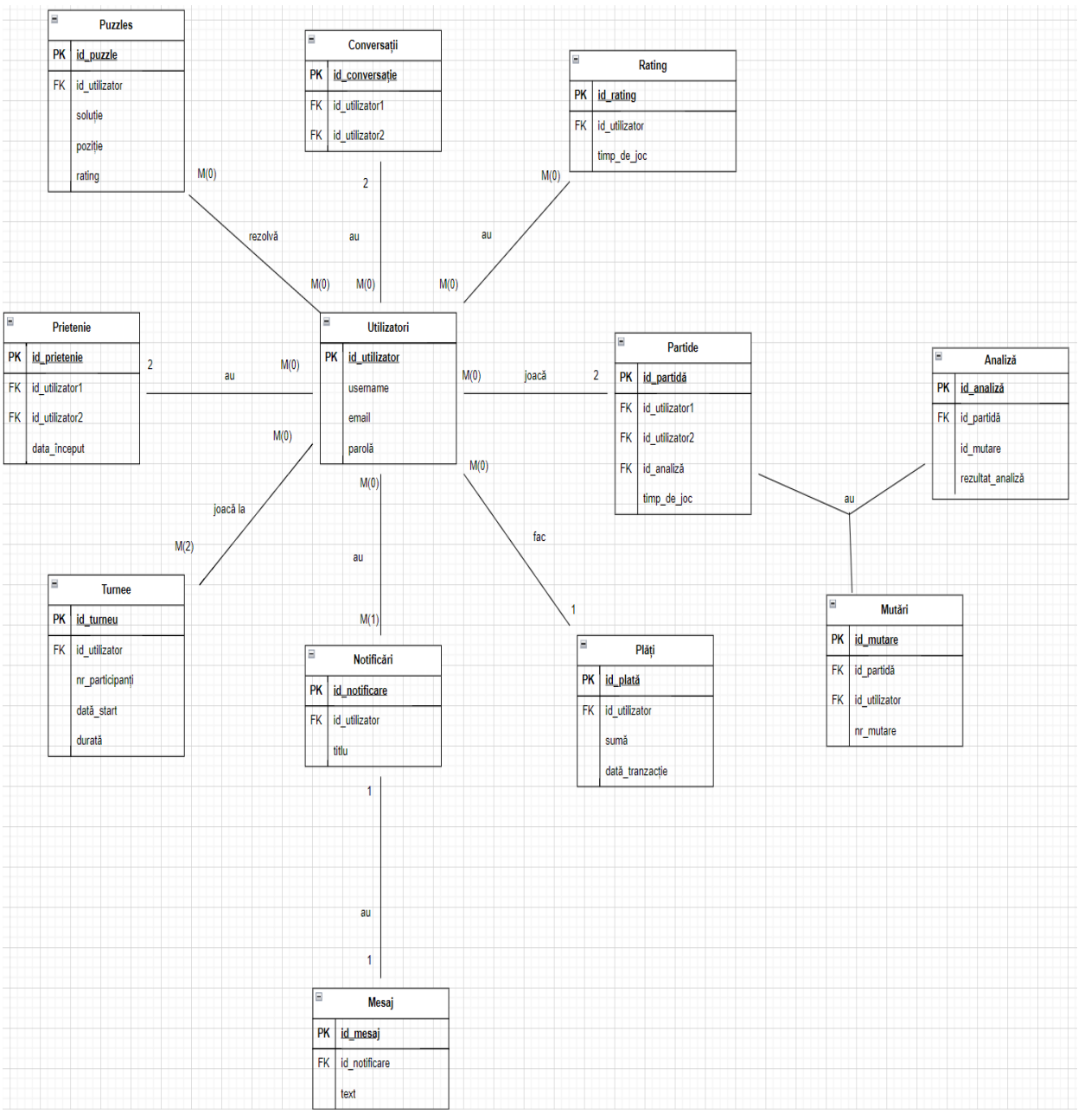
13. Definirea unui pachet care să conțină toate obiectele definite în cadrul proiectului

14. Definirea unui 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)

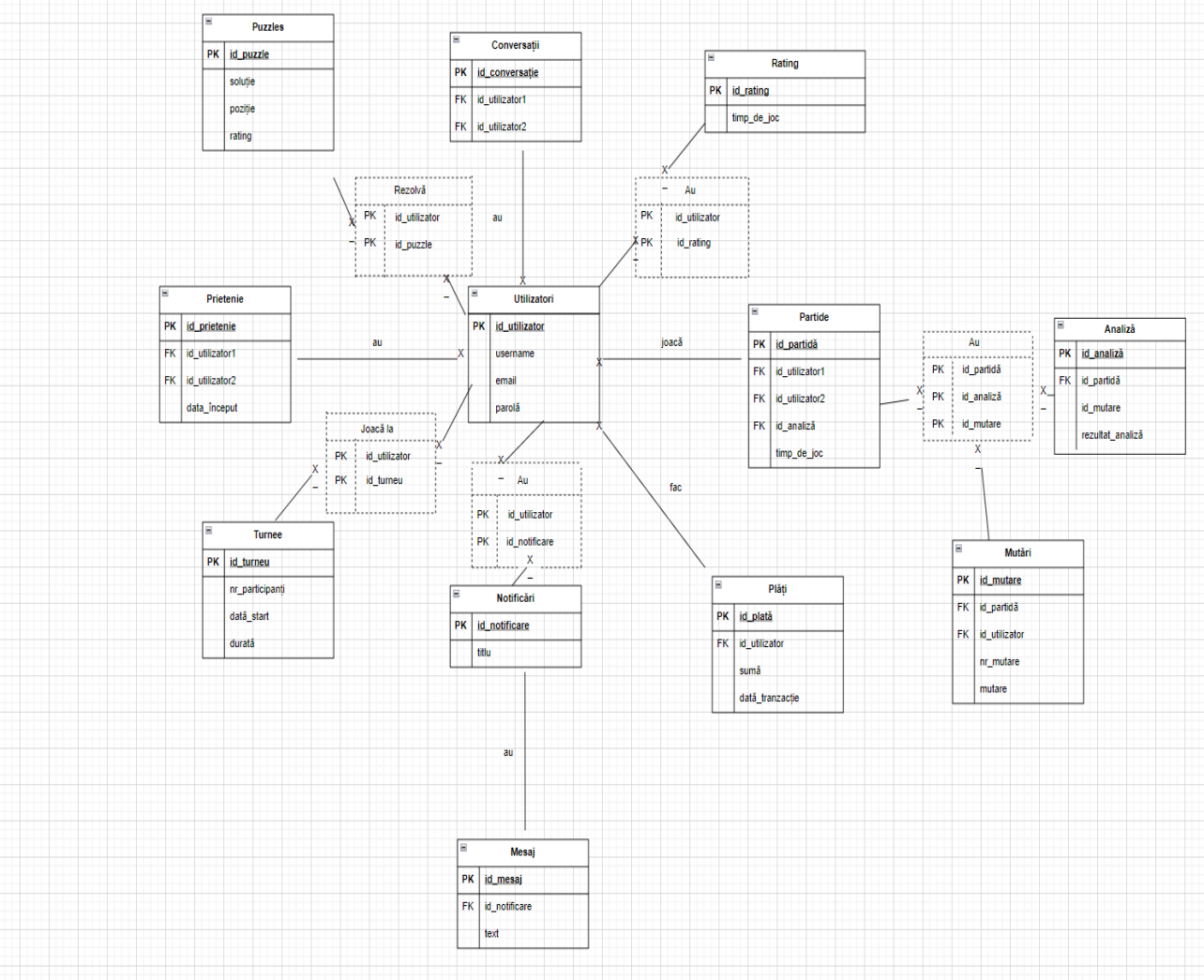
**1. Prezentarea pe scurt a bazei de date (utilitatea ei)**

În acest proiect voi prezenta cum arată implementarea “back-endului” aplicației Chess.com (https://www.chess.com) , dezvoltând baza de date din spatele aplicației. Până nu mult timp în urmă, bazele de date pentru șah erau reprezentate de cărți. Așa se studia șahul și așa se “stocau” partidele memorabile jucate de foștii campioni mondiali și Mari Maeștri. În schimb, în secolul 21, odată cu apariția tehnologiei, s-a făcut ușor-ușor tranziția către baze de date online, unde pot fi accesate toate partidele celebre și nu numai. Orice partidă jucată online sau la un turneu oficial, poate fi introdusă într-o bază de date, la care au acces toți jucătorii interesați. Datorită frumuseții jocului, dar și a popularității acestuia, odată cu inițializarea primelor baze de date, au apărut și primele aplicații pe care utilizatorii pot să joace partide, să învețe făcând puzzle-uri, analizând mutările partidelor, să participe la turnee oficiale online, dar și să simtă un sentiment de competitivitate, orice partidă fiind cotată și având în consecință schimbarea rating-ului între 2 utilizatori. De asemenea, aplicația poate fi teoretic categorizată și sub forma unei aplicații sociale, deoarece un utilizator își poate face prieteni, poate să aibă conversații cu aceștia, dar poate să și primească notificări de exemplu pentru turnee sau poate să se uite la streamer-ul lor favorit cum se joacă. Nu în ultimul rând, pentru mai multe beneficii, un utilizator poate să opteze la un abonament plătit pentru a putea accesa tot ce poate să ofere aplicația.

**2. Realizați diagrama entitate-relație**



**3. Realizați diagrama conceptuală + entități, relații și atribute**



***Entități:***

Pentru modelul de date al aplicației Chess.com, structurile UTILIZATORI, RATING, CONVERSAȚII, PUZZLES, PRIETENIE, TURNEE, NOTIFICĂRI, MESAJ, PLĂȚI, PARTIDE, MUTĂRI, ANALIZĂ reprezintă entități.

*UTILIZATORI* = Persoane care își fac cont pe aplicație. Cheie primară id\_utilizator.

*RATING* = Se alocă un punctaj (mai exact elo

(https://en.wikipedia.org/wiki/Elo\_rating\_system )) fiecărui jucător pentru orice stil de joc (https://en.wikipedia.org/wiki/Time\_control ), în funcție de rezultatele pe care le are în partidele pe care le joacă. Cheie primară id\_rating.

*CONVERSAȚII* = O discuție între doi utilizatori. Cheie primară id\_conversație. *PUZZLES* = Un utilizator se poate antrena, rezolvând unele poziții de natură tactică. Cheie primară id\_puzzle.

*PRIETENIE* = Relația dintre un utilizator și un prieten de-al lui. Cheie primară id\_prietenie.

*TURNEE* = Sunt organizate pe platformă la o anumită dată, ce durează un timp specificat partide între utilizatori, la final fiind un clasament în ordine descrescătoare în funcție de câte puncte de joc a câștigat fiecare utilizator. Cheie primară id\_turneu.

*NOTIFICĂRI* = Un mesaj pe care un utilizator îl primește în inbox. Cheie primară id\_notificare.

*MESAJ* = Textul dintr-o notificare. Cheie primară id\_mesaj.

*PLĂȚI* = O sumă de bani pe care un utilizator o plătește. Cheie primară id\_plată. *PARTIDE* = Totalitatea jocurilor de șah jucate în istorie. Cheie primară id\_partidă. *MUTĂRI* = Totalitatea mișcărilor făcute într-o partidă. Cheie primară id\_mutare. *ANALIZĂ* = Totalitatea rezultatelor partidei posibile în urma unor alte mutări decât cele făcute în partidă. Cheie primară id\_analiză.

***Relații:***

*UTILIZATORII au RATING* = relație dintre UTILIZATOR și RATING reflectând legătura dintre acestea (Ce rating are fiecare utilizator). Ea are cardinalitate minimă 0:0 și cardinalitate maximă m:n.

*UTILIZATORII au CONVERSAȚII* = relație dintre UTILIZATOR și CONVERSAȚII reflectând legătura dintre acestea (Ce conversații are fiecare utilizator). Ea are cardinalitate minimă 0:2 și cardinalitate maximă m:2.

*UTILIZATORII rezolvă PUZZLES*= relație dintre UTILIZATOR și PUZZLES reflectând legătura dintre acestea (Ce puzzle a rezolvat fiecare utilizator). Ea are cardinalitate minimă 0:0 și cardinalitate maximă m:n.

*UTILIZATORII au* *PRIETENIE*= relație dintre UTILIZATOR și PRIETENIE reflectând legătura dintre acestea (Ce prietenie are fiecare utilizator). Ea are cardinalitate minimă 0:2 și cardinalitate maximă m:2.

*UTILIZATORII joacă la TURNEE* = relație dintre UTILIZATOR și TURNEE reflectând legătura dintre acestea (Ce turnee joacă fiecare utilizator). Ea are cardinalitate minimă 0:2 și cardinalitate maximă m:n.

*UTILIZATORII au NOTIFICĂRI*= relație dintre UTILIZATOR și NOTIFICĂRI reflectând legătura dintre acestea (Ce notificări are fiecare utilizator). Ea are cardinalitate minimă 0:1 și cardinalitate maximă m:n.

*NOTIFICĂRI au MESAJ*= relație dintre NOTIFICĂRI și MESAJ reflectând legătura dintre acestea (Ce mesaj e în fiecare Notificare). Ea are cardinalitatea 1:1.

*UTILIZATORII fac PLĂȚI* = relație dintre UTILIZATOR și PLATĂ reflectând legătura dintre acestea (Ce utilizator a făcut ce plată). Ea are cardinalitate minimă 0:1 și cardinalitate maximă m:1.

*UTILIZATORII joacă PARTIDE* = relație dintre UTILIZATOR și PARTIDĂ reflectând legătura dintre acestea (Ce partide a jucată de către fiecare utilizator). Ea are cardinalitate minimă 0:2 și cardinalitate maximă m:2.

*PARTIDE au ANALIZĂ și MUTĂRI* = relație dintre PARTIDE, ANALIZĂ și MUTĂRI reflectând legătura dintre acestea (Ce mutări și ce analize are fiecare partidă).

***Atribute:***

- Entitatea independentă UTILIZATORI are ca atribute:

id\_utilizator = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul unic destinat automat fiecărui utilizator în momentul în care se creează un cont în aplicație.

username = variabliă de tip caracter de lungime maximă 25, care reprezintă nickname-ul fiecărui utilizator.

email = variabliă de tip caracter de lungime maximă 100, care reprezintă email-ul utilizatorului.

parolă = variabliă de tip caracter de lungime maximă 25, care reprezintă parola pe care utilizatorul o alege pentru a se loga în cont.

- Entitatea independentă RATING are ca atribute:

id\_rating = variabliă de tip întreg de lungime maximă 4, care reprezintă “elo”-ul fiecărui utilizator.

timp\_de\_joc = variabliă de tip caracter de lungime maximă 10, care reprezintă stilul de joc la care este destinat rating-ul respectiv ce va putea avea următoarele valori: clasic, rapid, blitz, bullet, daily.

- Entitatea independentă PUZZLES are ca atribute:

id\_puzzle = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul unui puzzle.

soluție = variabliă de tip caracter de lungime maximă 250, care reprezintă mutările corecte pentru un puzzle

poziție = variabliă de tip caracter de lungime maximă 250, care reprezintă pisele și ce câmpuri ocupă pentru a asambla poziția

rating = variabliă de tip întreg de lungime maximă 4, care reprezintă “elo”-ul unui puzzle

- Entitatea independentă PRIETENIE are ca atribute:

id\_prietenie = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul unei prietenii.

id\_utilizator1 = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul unic destinat automat fiecărui utilizator în momentul în care se creează un cont în aplicație, reprezentând un prieten din relația de prietenie.

id\_utilizator2 = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul unic destinat automat fiecărui utilizator în momentul în care se creează un cont în aplicație, reprezentând celălalt prieten din relația de prietenie.

data\_început = variabliă de tip dată, care reprezintă data din momentul în care s-a creat o prietenie

- Entitatea independentă TURNEE are ca atribute:

id\_turneu = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul unui turneu.

nr\_participanți = variabliă de tip întreg de lungime maximă 4, care reprezintă numărul de partcipanți la turneu

dată\_start = variabliă de tip întreg dată, care reprezintă data de început a turneului durată = variabliă de tip întreg de lungime maximă 2, care reprezintă numărul de zile în care se desfășoară turneul

- Entitatea independentă NOTIFICĂRI are ca atribute:

id\_notificare = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul notificării titlu = variabliă de tip caracter de lungime maximă 20, care reprezintă titlul unei notificări

- Entitatea independentă MESAJ are ca atribute:

id\_mesaj = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul mesajului unei notificări

id\_notificare = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul notificării la care vrem să atașăm un text

text = variabliă de tip caracter de lungime maximă 10000, care reprezintă text-ul propriu-zis al unei notificări

- Entitatea independentă PLĂȚI are ca atribute:

id\_plată = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul unei plăți.

id\_utilizator = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul utilizatorului care a efectuat o plată.

sumă = variabliă de tip întreg de lungime maximă 3, care reprezintă suma de bani în dolari pe care a achitat-o.

dată\_tranzacție = variabliă de tip dată, care reprezintă data la care s-a achitat plata.

- Entitatea independentă PARTIDE are ca atribute:

id\_partidă = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul unei partide.

id\_utilizator1 = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul primului jucător care a jucat respectiva partidă.

id\_utilizator2 = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul celui de-al doilea jucător care a jucat respectiva partidă.

id\_analiză = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul analizei din spatele partidei

timp\_de\_joc = variabliă de tip caracter de lungime maximă 10, care reprezintă stilul de joc la care este destinat rating-ul respectiv ce va putea avea următoarele valori: clasic, rapid, blitz, bullet, daily.

- Entitatea independentă ANALIZĂ are ca atribute:

id\_analiză = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul unei analizei de-al unei partide

id\_partidă = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul partidei asupra căreia se desfășoara analiza

id\_mutare = variabliă de tip întreg de lungime maximă 9, care reprezintă mutarea cu care analizăm o partidă

rezultat\_analiză = variabliă de tip caracter de lungime maximă 3, care reprezintă rezultatul partidei și va avea următoarele valori: 1-0 când albul câștigă, 0-1 când negrul câștigă și ½ când rezultatul este remiză.

- Entitatea independentă MUTĂRI are ca atribute:

id\_mutare = variabliă de tip caracter de lungime maximă 5, care reprezintă mutarea făcută, exemplu e4 (pion se mută pe câmpul e4), Nf6 (din engleză Night = cal se mută pe câmpul f6), 0-0-0 (rocada mare) etc.

id\_utilizator = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul utilizatorului care face o mutare

id\_partidă = variabliă de tip întreg de lungime maximă 9, care reprezintă id-ul partidei

nr\_mutare = variabliă de tip întreg de lungime maximă 4, care reprezintă nr-ul mutării (exemplu la mutarea 1 sau 100 sau 1550)

**4. Implementarea în Oracle a diagramei conceptuale +**

**5. Adăugarea informațiilor coerente în tabelele create**

--Cerintele 4 si 5 din proiect:

--Inserarea datelor in tabela Utilizatori

CREATE TABLE Utilizatori

(

id\_utilizator int NOT NULL,

username varchar2(25) NOT NULL UNIQUE,

email varchar2(100) NOT NULL UNIQUE,

parola varchar2(25) NOT NULL,

PRIMARY KEY(id\_utilizator)

);

INSERT INTO Utilizatori

VALUES (1, 'ionut', 'ionut@gmail.com', 'ionut');

INSERT INTO Utilizatori

VALUES (2, 'alex', 'ale@gmail.com', 'pal');

INSERT INTO Utilizatori

VALUES (3, 'maremaestru', 'maremaestru@yahoo.com', 'GM');

INSERT INTO Utilizatori

VALUES (4, 'queen', 'qel@gmail.com', 'iamqueen');

INSERT INTO Utilizatori

VALUES (5, 'd4player', 'plic@gmail.com', 'londonsystem');

DESCRIBE Utilizatori;

SELECT \*

FROM Utilizatori;

--Inserarea datelor in tabela Utilizatori

CREATE TABLE Rating

(

id\_rating int NOT NULL,

timp\_de\_joc varchar2(10) NOT NULL,

PRIMARY KEY(id\_rating)

);

INSERT INTO Rating

VALUES (1500, 'clasic');

INSERT INTO Rating

VALUES (2500, 'rapid');

INSERT INTO Rating

VALUES (250, 'bullet');

INSERT INTO Rating

VALUES (1750, 'blitz');

INSERT INTO Rating

VALUES (1867, 'blitz');

INSERT INTO Rating

VALUES (2000, 'blitz');

INSERT INTO Rating

VALUES (3175, 'blitz');

DESCRIBE Rating;

SELECT \*

FROM Rating;

--Inserarea datelor in tabela Conversatii

CREATE TABLE Conversatii

(

id\_conversatie int NOT NULL,

id\_utilizator1 int NOT NULL,

id\_utilizator2 int NOT NULL,

PRIMARY KEY(id\_conversatie),

FOREIGN KEY(id\_utilizator1) REFERENCES Utilizatori(id\_utilizator),

FOREIGN KEY(id\_utilizator2) REFERENCES Utilizatori(id\_utilizator)

);

INSERT INTO Conversatii

VALUES (1, 1, 2);

INSERT INTO Conversatii

VALUES (2, 1, 3);

INSERT INTO Conversatii

VALUES (3, 1, 4);

INSERT INTO Conversatii

VALUES (4, 1, 5);

INSERT INTO Conversatii

VALUES (5, 2, 3);

DESCRIBE Conversatii;

SELECT \*

FROM Conversatii;

--Inserarea datelor in tabela Puzzles

CREATE TABLE Puzzles

(

id\_puzzle int NOT NULL,

solutie varchar2(250) NOT NULL,

pozitie varchar2(250) NOT NULL,

rating int NOT NULL,

PRIMARY KEY(id\_puzzle)

);

INSERT INTO Puzzles

VALUES (1, 'Ra8', 'Kd6 Ra1 Kd8', 100);

INSERT INTO Puzzles

VALUES (2, 'Qd7', 'Kd6 Qc6 Kd8 Rc8 Re8', 150);

INSERT INTO Puzzles

VALUES (3, 'Qa8 Rc8 Qc8', 'Kh6 Qa2 Rc3 Kh8', 200);

INSERT INTO Puzzles

VALUES (4, 'Kg7 h4 Kf6 h3 Ke7 h2 c7 Kb7 Kd7 h1=Q c8=Q', 'Kh8 c6 Ka6 h5', 2500);

INSERT INTO Puzzles

VALUES (5, 'Kc6 Kc8 d6 Kd8 d7 Ke7 Kc7 Ke6 d8=Q', 'Kd6 d5 Kd8', 1000);

DESCRIBE Puzzles;

SELECT \*

FROM Puzzles;

--Inserarea datelor in tabela Prietenie

CREATE TABLE Prietenie

(

id\_prietenie int NOT NULL,

id\_utilizator1 int NOT NULL,

id\_utilizator2 int NOT NULL,

data\_inceput date,

PRIMARY KEY(id\_prietenie),

FOREIGN KEY(id\_utilizator1) REFERENCES Utilizatori(id\_utilizator),

FOREIGN KEY(id\_utilizator2) REFERENCES Utilizatori(id\_utilizator)

);

INSERT INTO Prietenie

VALUES (1, 1, 4, to\_date('31-03-2023', 'dd-mm-yyyy'));

INSERT INTO Prietenie

VALUES (2, 1, 5, to\_date('30-04-2023', 'dd-mm-yyyy'));

INSERT INTO Prietenie

VALUES (3, 2, 3, to\_date('19-05-2023', 'dd-mm-yyyy'));

INSERT INTO Prietenie

VALUES (4, 2, 4, to\_date('30-06-2022', 'dd-mm-yyyy'));

INSERT INTO Prietenie

VALUES (5, 3, 5, to\_date('18-08-2022', 'dd-mm-yyyy'));

DESCRIBE Prietenie;

SELECT \*

FROM Prietenie;

--Inserarea datelor in tabela Turnee

CREATE TABLE Turnee

(

id\_turneu int NOT NULL,

nr\_participanti int NOT NULL,

data\_start date NOT NULL,

durata int NOT NULL,

PRIMARY KEY(id\_turneu)

);

INSERT INTO Turnee

VALUES (1, 5, to\_date('31-03-2023', 'dd-mm-yyyy'), 7);

INSERT INTO Turnee

VALUES (2, 3, to\_date('01-04-2023', 'dd-mm-yyyy'), 3);

INSERT INTO Turnee

VALUES (3, 4, to\_date('31-01-2023', 'dd-mm-yyyy'), 12);

INSERT INTO Turnee

VALUES (4, 2, to\_date('28-02-2023', 'dd-mm-yyyy'), 8);

INSERT INTO Turnee

VALUES (5, 5, to\_date('29-02-2020', 'dd-mm-yyyy'), 15);

DESCRIBE Turnee;

SELECT \*

FROM Turnee;

--Inserarea datelor in tabela Notificari

CREATE TABLE Notificari

(

id\_notificare int NOT NULL,

titlu varchar2(20) NOT NULL,

PRIMARY KEY(id\_notificare)

);

INSERT INTO Notificari

VALUES (1, 'Turneu in 2 zile');

INSERT INTO Notificari

VALUES (2, 'Turneu Titled');

INSERT INTO Notificari

VALUES (3, 'Cerere prietenie');

INSERT INTO Notificari

VALUES (4, 'Felicitari!');

INSERT INTO Notificari

VALUES (5, 'Turneu in 5 zile');

DESCRIBE Notificari;

SELECT \*

FROM Notificari;

--Inserarea datelor in tabela Mesaj

CREATE TABLE Mesaj

(

id\_mesaj int NOT NULL,

id\_notificare int NOT NULL,

text varchar2(1000) NOT NULL,

PRIMARY KEY(id\_mesaj),

FOREIGN KEY(id\_notificare) REFERENCES Notificari(id\_notificare)

);

INSERT INTO Mesaj

VALUES (1, 1, 'Inregistreaza-te acum la turneul "Titled Tuesday" care incepe in 2 zile!');

INSERT INTO Mesaj

VALUES (2, 2, 'Turneu in care participa maestri incep in curand!');

INSERT INTO Mesaj

VALUES (3, 3, 'Ai primit o cerere de prietenie de la d4player!');

INSERT INTO Mesaj

VALUES (4, 4, 'Ai reusit sa obtii un elo record. Felicitari!');

INSERT INTO Mesaj

VALUES (5, 5, 'Au mai ramas doar 5 zile in care te poti inscrie la turneul nostru!');

DESCRIBE Mesaj;

SELECT \*

FROM Mesaj;

--Inserarea datelor in tabela Plati

CREATE TABLE Plati

(

id\_plata int NOT NULL,

id\_utilizator int NOT NULL,

suma int NOT NULL,

data\_tranzactie date NOT NULL,

PRIMARY KEY(id\_plata),

FOREIGN KEY(id\_utilizator) REFERENCES Utilizatori(id\_utilizator)

);

INSERT INTO Plati

VALUES (1, 1, 15, to\_date('01-01-2023', 'dd-mm-yyyy'));

INSERT INTO Plati

VALUES (2, 1, 20, to\_date('01-02-2023', 'dd-mm-yyyy'));

INSERT INTO Plati

VALUES (3, 2, 15, to\_date('01-03-2023', 'dd-mm-yyyy'));

INSERT INTO Plati

VALUES (4, 4, 15, to\_date('31-01-2023', 'dd-mm-yyyy'));

INSERT INTO Plati

VALUES (5, 1, 10, to\_date('01-05-2023', 'dd-mm-yyyy'));

DESCRIBE Plati;

SELECT \*

FROM Plati;

--Inserarea datelor in tabela Partide

CREATE TABLE Partide

(

id\_partida int NOT NULL,

id\_utilizator1 int NOT NULL,

id\_utilizator2 int NOT NULL,

id\_analiza int NOT NULL,

timp\_de\_joc varchar2(10) NOT NULL,

PRIMARY KEY(id\_partida),

FOREIGN KEY(id\_utilizator1) REFERENCES Utilizatori(id\_utilizator),

FOREIGN KEY(id\_utilizator2) REFERENCES Utilizatori(id\_utilizator)

);

INSERT INTO Partide

VALUES (1, 1, 2, 1, 'blitz');

INSERT INTO Partide

VALUES (2, 1, 3, 2, 'bullet');

INSERT INTO Partide

VALUES (3, 1, 4, 3, 'clasic');

INSERT INTO Partide

VALUES (4, 1, 5, 4, 'blitz');

INSERT INTO Partide

VALUES (5, 2, 3, 5, 'blitz');

DESCRIBE Partide;

SELECT \*

FROM Partide;

--Inserarea datelor in tabela Analiza

CREATE TABLE Analiza

(

id\_analiza int NOT NULL,

id\_partida int NOT NULL,

id\_mutare int NOT NULL,

rezultat\_analiza varchar2(3) NOT NULL,

PRIMARY KEY(id\_analiza),

FOREIGN KEY(id\_partida) REFERENCES Partide(id\_partida)

);

INSERT INTO Analiza

VALUES (1, 1, 1, '1-0');

INSERT INTO Analiza

VALUES (2, 2, 2, '1/2');

INSERT INTO Analiza

VALUES (3, 3, 4, '1-0');

INSERT INTO Analiza

VALUES (4, 5, 3, '0-1');

INSERT INTO Analiza

VALUES (5, 4, 5, '1/2');

DESCRIBE Analiza;

SELECT \*

FROM Analiza;

--Inserarea datelor in tabela Mut?ri

CREATE TABLE Mutari

(

id\_mutare int NOT NULL,

id\_partida int NOT NULL,

id\_utilizator int NOT NULL,

nr\_mutare int NOT NULL,

mutare varchar2(5) NOT NULL,

PRIMARY KEY(id\_mutare),

FOREIGN KEY(id\_partida) REFERENCES Partide(id\_partida),

FOREIGN KEY(id\_utilizator) REFERENCES Utilizatori(id\_utilizator)

);

INSERT INTO Mutari

VALUES (1, 1, 1, 1, 'e4');

INSERT INTO Mutari

VALUES (2, 2, 1, 1, 'e5');

INSERT INTO Mutari

VALUES (3, 3, 4, 2, 'Nf3');

INSERT INTO Mutari

VALUES (4, 4, 5, 3, 'Nf6');

INSERT INTO Mutari

VALUES (5, 5, 2, 10, 'O-O-O');

DESCRIBE Mutari;

SELECT \*

FROM Mutari;

--Inserarea datelor in tabela asociativa UtilizatoriRating

CREATE TABLE UtilizatoriRating

(

id\_utilizator int NOT NULL,

id\_rating int NOT NULL,

PRIMARY KEY(id\_utilizator, id\_rating),

FOREIGN KEY(id\_utilizator) REFERENCES Utilizatori(id\_utilizator),

FOREIGN KEY(id\_rating) REFERENCES Rating(id\_rating)

);

INSERT INTO UtilizatoriRating

VALUES (1, 1500);

INSERT INTO UtilizatoriRating

VALUES (1, 2500);

INSERT INTO UtilizatoriRating

VALUES (2, 2500);

INSERT INTO UtilizatoriRating

VALUES (1, 250);

INSERT INTO UtilizatoriRating

VALUES (5, 1500);

INSERT INTO UtilizatoriRating

VALUES (3, 1867);

INSERT INTO UtilizatoriRating

VALUES (3, 3175);

INSERT INTO UtilizatoriRating

VALUES (4, 2000);

INSERT INTO UtilizatoriRating

VALUES (4, 1750);

INSERT INTO UtilizatoriRating

VALUES (5, 3175);

DESCRIBE UtilizatoriRating;

SELECT \*

FROM UtilizatoriRating;

--Inserarea datelor in tabela asociativa UtilizatoriRating

CREATE TABLE UtilizatoriPuzzles

(

id\_utilizator int NOT NULL,

id\_puzzle int NOT NULL,

PRIMARY KEY(id\_utilizator, id\_puzzle),

FOREIGN KEY(id\_utilizator) REFERENCES Utilizatori(id\_utilizator),

FOREIGN KEY(id\_puzzle) REFERENCES Puzzles(id\_puzzle)

);

INSERT INTO UtilizatoriPuzzles

VALUES (1, 1);

INSERT INTO UtilizatoriPuzzles

VALUES (1, 2);

INSERT INTO UtilizatoriPuzzles

VALUES (2, 1);

INSERT INTO UtilizatoriPuzzles

VALUES (1, 3);

INSERT INTO UtilizatoriPuzzles

VALUES (5, 4);

INSERT INTO UtilizatoriPuzzles

VALUES (3, 4);

INSERT INTO UtilizatoriPuzzles

VALUES (3, 5);

INSERT INTO UtilizatoriPuzzles

VALUES (4, 5);

INSERT INTO UtilizatoriPuzzles

VALUES (4, 1);

INSERT INTO UtilizatoriPuzzles

VALUES (5, 2);

DESCRIBE UtilizatoriPuzzles;

SELECT \*

FROM UtilizatoriPuzzles;

--Inserarea datelor in tabela asociativa UtilizatoriPrietenie

CREATE TABLE UtilizatoriTurnee

(

id\_utilizator int NOT NULL,

id\_turneu int NOT NULL,

PRIMARY KEY(id\_utilizator, id\_turneu),

FOREIGN KEY(id\_utilizator) REFERENCES Utilizatori(id\_utilizator),

FOREIGN KEY(id\_turneu) REFERENCES Turnee(id\_turneu)

);

INSERT INTO UtilizatoriTurnee

VALUES (1, 1);

INSERT INTO UtilizatoriTurnee

VALUES (1, 2);

INSERT INTO UtilizatoriTurnee

VALUES (1, 3);

INSERT INTO UtilizatoriTurnee

VALUES (1, 4);

INSERT INTO UtilizatoriTurnee

VALUES (1, 5);

INSERT INTO UtilizatoriTurnee

VALUES (2, 1);

INSERT INTO UtilizatoriTurnee

VALUES (2, 2);

INSERT INTO UtilizatoriTurnee

VALUES (2, 3);

INSERT INTO UtilizatoriTurnee

VALUES (3, 1);

INSERT INTO UtilizatoriTurnee

VALUES (3, 2);

INSERT INTO UtilizatoriTurnee

VALUES (3, 3);

INSERT INTO UtilizatoriTurnee

VALUES (3, 5);

INSERT INTO UtilizatoriTurnee

VALUES (4, 2);

INSERT INTO UtilizatoriTurnee

VALUES (4, 4);

INSERT INTO UtilizatoriTurnee

VALUES (5, 1);

INSERT INTO UtilizatoriTurnee

VALUES (5, 2);

INSERT INTO UtilizatoriTurnee

VALUES (5, 3);

INSERT INTO UtilizatoriTurnee

VALUES (5, 4);

INSERT INTO UtilizatoriTurnee

VALUES (5, 5);

DESCRIBE UtilizatoriTurnee;

SELECT \*

FROM UtilizatoriTurnee;

--Inserarea datelor in tabela asociativa PartideInfo

CREATE TABLE PartideInfo

(

id\_partida int NOT NULL,

id\_analiza int NOT NULL,

id\_mutare int NOT NULL,

PRIMARY KEY(id\_partida, id\_analiza, id\_mutare),

FOREIGN KEY(id\_partida) REFERENCES Partide(id\_partida),

FOREIGN KEY(id\_analiza) REFERENCES Analiza(id\_analiza),

FOREIGN KEY(id\_mutare) REFERENCES Mutari(id\_mutare)

);

INSERT INTO PartideInfo

VALUES (1, 1, 1);

INSERT INTO PartideInfo

VALUES (1, 1, 2);

INSERT INTO PartideInfo

VALUES (2, 1, 1);

INSERT INTO PartideInfo

VALUES (2, 1, 2);

INSERT INTO PartideInfo

VALUES (3, 1, 4);

INSERT INTO PartideInfo

VALUES (3, 2, 5);

INSERT INTO PartideInfo

VALUES (4, 4, 3);

INSERT INTO PartideInfo

VALUES (4, 3, 3);

INSERT INTO PartideInfo

VALUES (5, 5, 5);

INSERT INTO PartideInfo

VALUES (1, 1, 3);

DESCRIBE PartideInfo;

SELECT \*

FROM PartideInfo;

--cream secventa seq ce va fi folosita pentru inserarea in tabel

CREATE SEQUENCE seq

START WITH 1

INCREMENT BY 1

MINVALUE 1

MAXVALUE 5;

----Inserarea datelor in tabela asociativa UtilizatoriNotificari

CREATE TABLE UtilizatoriNotificari

(

id\_utilizator int NOT NULL,

id\_notificare int NOT NULL,

PRIMARY KEY(id\_utilizator, id\_notificare),

FOREIGN KEY(id\_utilizator) REFERENCES Utilizatori(id\_utilizator),

FOREIGN KEY(id\_notificare) REFERENCES Notificari(id\_notificare)

);

DESCRIBE UtilizatoriNotificari;

INSERT INTO UtilizatoriNotificari VALUES (seq.nextval, 2);

INSERT INTO UtilizatoriNotificari VALUES (seq.nextval, 1);

INSERT INTO UtilizatoriNotificari VALUES (seq.nextval, 3);

INSERT INTO UtilizatoriNotificari VALUES (seq.nextval, 5);

INSERT INTO UtilizatoriNotificari VALUES (seq.nextval, 4);

**6. Rezolvarea unei probleme pe care o rezolv folosind un subprogram stocat independent care să utilizeze toate cele 3 tipuri de colecții + apelarea subprogramului**

Cerință:

Dezvolți un subprogram stocat independent care să calculeze și să afișeze statistici legate de rezolvarea puzzle-urilor.

*Statisticile* pe care subprogramul trebuie să le calculeze includ:

1. Numărul total de puzzle-uri rezolvate de utilizator.
2. Media rating-ului puzzle-urilor rezolvate de utilizator.
3. Cel mai dificil puzzle rezolvat de utilizator (cel cu rating-ul maxim).
4. Lista de puzzle-uri rezolvate de utilizator, împreună cu rating-ul lor.

Lista de puzzle-uri de utilizator, împreună cu rating-ul lor se va afișa de 3 ori, o dată pentru fiecare tip de colecție.

--Exercitiul 6

CREATE OR REPLACE PROCEDURE StatisticiPuzzle(

p\_utilizator\_id Utilizatori.id\_utilizator%TYPE

) AS

TYPE PuzzleStatType IS RECORD (

id\_puzzle Puzzles.id\_puzzle%TYPE,

rating Puzzles.rating%TYPE

);

TYPE PuzzleStatList IS TABLE OF PuzzleStatType; --nested table

TYPE PuzzleStatVArray IS VARRAY(100) OF PuzzleStatType; --varray

TYPE PuzzleStatIndexBy IS TABLE OF PuzzleStatType INDEX BY PLS\_INTEGER; --index by table

v\_numar\_total\_puzzleuri UtilizatoriPuzzles.id\_puzzle%TYPE := 0;

v\_suma\_rating NUMBER := 0;

v\_max\_rating Puzzles.rating%TYPE := NULL;

--initializare pentru cele 3 colectii

v\_puzzleuriList PuzzleStatList := PuzzleStatList();

v\_puzzleuriVArray PuzzleStatVArray;

v\_puzzleuriIndexBy PuzzleStatIndexBy;

BEGIN

v\_puzzleuriList := PuzzleStatList();

v\_puzzleuriVArray := PuzzleStatVArray();

v\_puzzleuriIndexBy := PuzzleStatIndexBy();

FOR puzzle\_rec IN (

SELECT up.id\_puzzle, p.rating

FROM UtilizatoriPuzzles up

JOIN Puzzles p ON up.id\_puzzle = p.id\_puzzle

WHERE up.id\_utilizator = p\_utilizator\_id

)

LOOP

v\_numar\_total\_puzzleuri := v\_numar\_total\_puzzleuri + 1;

v\_suma\_rating := v\_suma\_rating + puzzle\_rec.rating;

IF v\_max\_rating IS NULL OR puzzle\_rec.rating > v\_max\_rating THEN

v\_max\_rating := puzzle\_rec.rating;

END IF;

--populare

v\_puzzleuriList.EXTEND;

v\_puzzleuriList(v\_numar\_total\_puzzleuri).id\_puzzle := puzzle\_rec.id\_puzzle;

v\_puzzleuriList(v\_numar\_total\_puzzleuri).rating := puzzle\_rec.rating;

v\_puzzleuriVArray.EXTEND;

v\_puzzleuriVArray(v\_puzzleuriVArray.LAST) := puzzle\_rec;

v\_puzzleuriIndexBy(v\_numar\_total\_puzzleuri) := puzzle\_rec;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Numărul total de puzzle-uri rezolvate: ' || v\_numar\_total\_puzzleuri);

IF v\_numar\_total\_puzzleuri > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Media rating-ului puzzle-urilor rezolvate: ' || ROUND(v\_suma\_rating / v\_numar\_total\_puzzleuri, 2));

DBMS\_OUTPUT.PUT\_LINE('Cel mai dificil puzzle rezolvat (cel cu rating-ul maxim): ' || v\_max\_rating);

DBMS\_OUTPUT.PUT\_LINE('Lista de puzzle-uri rezolvate de utilizator, împreună cu rating-ul lor (nested table):');

FOR i IN 1..v\_numar\_total\_puzzleuri LOOP

DBMS\_OUTPUT.PUT\_LINE('ID Puzzle: ' || v\_puzzleuriList(i).id\_puzzle || ', Rating: ' || v\_puzzleuriList(i).rating);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Lista de puzzle-uri rezolvate de utilizator, împreună cu rating-ul lor (VARRAY):');

FOR i IN 1..v\_puzzleuriVArray.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('ID Puzzle: ' || v\_puzzleuriVArray(i).id\_puzzle || ', Rating: ' || v\_puzzleuriVArray(i).rating);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Lista de puzzle-uri rezolvate de utilizator, împreună cu rating-ul lor (Index-by table):');

FOR i IN v\_puzzleuriIndexBy.FIRST..v\_puzzleuriIndexBy.LAST LOOP

DBMS\_OUTPUT.PUT\_LINE('ID Puzzle: ' || v\_puzzleuriIndexBy(i).id\_puzzle || ', Rating: ' || v\_puzzleuriIndexBy(i).rating);

END LOOP;

ELSE

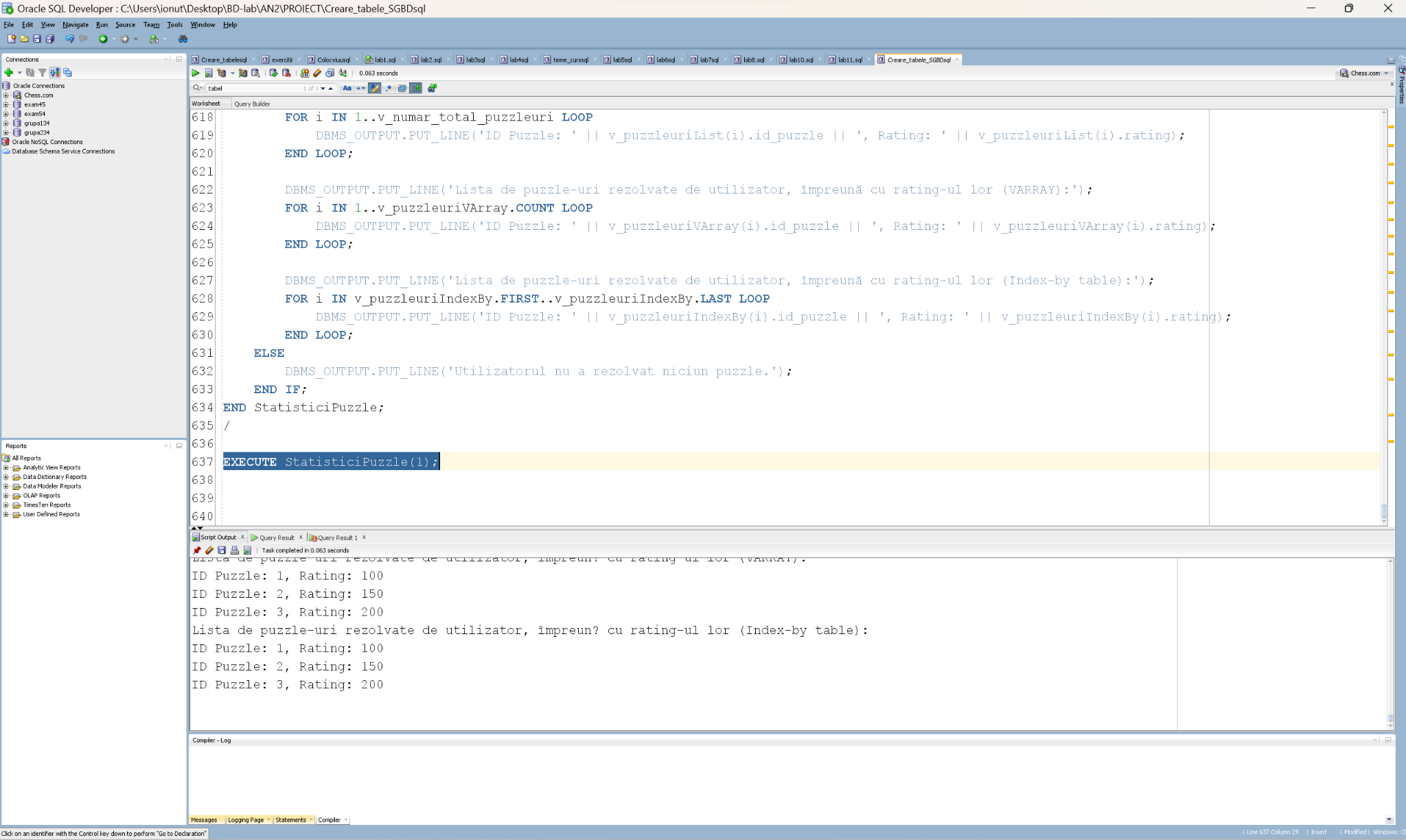
DBMS\_OUTPUT.PUT\_LINE('Utilizatorul nu a rezolvat niciun puzzle.');

END IF;

END StatisticiPuzzle;

/

EXECUTE StatisticiPuzzle(1);



**7. Rezolvarea unei probleme pe care o rezolv folosind un subprogram stocat independent care să utilizeze 2 tipuri diferite de cursoare, unul dintre acestea fiind cursor parametrizat, dependent de celălalt cursor + apelarea subprogramului**

Cerință:

Scrieți acum un subprogram cu 2 parametri, un id\_utilizator și un rating, care afișează toate informațiile din toate puzzle-urile care sunt rezolvate de id\_utilizator și au rating-ul mai mare decât rating. Acest subprogram trebuie să conțină 2 cursoare, un ciclu cursor care stochează toate puzzle-urile care au rating mai mare decât rating-ul dat ca parametru, și un cursor parametrizat care depinde de ciclu cursor care să aibă toate puzzle-urile care să fie rezolvate de utilizator. Să se și calculeze puzzle-urile care sunt și în cursorul parametrizat, și în ciclul cursor, astfel încăt să se rezolve cerința pusă inițială. Apelați acest subprogram.

--Exercitiul 7

CREATE OR REPLACE PROCEDURE PuzzleByRating(

p\_user\_id NUMBER,

p\_rating NUMBER

) AS

-- Declaram un cursor parametrizat

CURSOR parametrized\_cursor (p\_user\_id NUMBER) IS

SELECT p.id\_puzzle, p.solutie, p.pozitie, p.rating

FROM Puzzles p

JOIN UtilizatoriPuzzles up ON up.id\_puzzle = p.id\_puzzle

WHERE up.id\_utilizator = p\_user\_id;

-- Declaram un ciclu cursor

CURSOR loop\_cursor IS

SELECT id\_puzzle, solutie, pozitie, rating

FROM Puzzles

WHERE rating > p\_rating;

-- Declaram variabilele pentru citirea rezultatelor din cursorul parametrizat

v\_id\_puzzle\_param Puzzles.id\_puzzle%TYPE;

v\_solutie\_param Puzzles.solutie%TYPE;

v\_pozitie\_param Puzzles.pozitie%TYPE;

v\_rating\_param Puzzles.rating%TYPE;

-- Declaram variabilele pentru citirea rezultatelor din ciclu cursor

v\_id\_puzzle\_loop Puzzles.id\_puzzle%TYPE;

v\_solutie\_loop Puzzles.solutie%TYPE;

v\_pozitie\_loop Puzzles.pozitie%TYPE;

v\_rating\_loop Puzzles.rating%TYPE;

-- Declaram o variabila pentru a stoca ID-urile puzzle-urilor din intersectie

TYPE puzzle\_id\_list IS TABLE OF Puzzles.id\_puzzle%TYPE;

v\_intersected\_ids puzzle\_id\_list := puzzle\_id\_list();

BEGIN

-- Deschidem ciclu cursor și iterăm prin el

OPEN loop\_cursor;

LOOP

FETCH loop\_cursor INTO v\_id\_puzzle\_loop, v\_solutie\_loop, v\_pozitie\_loop, v\_rating\_loop;

EXIT WHEN loop\_cursor%NOTFOUND;

-- Afișăm informațiile din ciclu cursor

DBMS\_OUTPUT.PUT\_LINE('ID\_Puzzle (Ciclu): ' || v\_id\_puzzle\_loop || ', Solutie: ' || v\_solutie\_loop ||

', Pozitie: ' || v\_pozitie\_loop || ', Rating: ' || v\_rating\_loop);

-- Deschidem cursorul parametrizat și iterăm prin el

OPEN parametrized\_cursor(p\_user\_id);

LOOP

FETCH parametrized\_cursor INTO v\_id\_puzzle\_param, v\_solutie\_param, v\_pozitie\_param, v\_rating\_param;

EXIT WHEN parametrized\_cursor%NOTFOUND;

-- Afișăm informațiile din cursorul parametrizat

DBMS\_OUTPUT.PUT\_LINE('ID\_Puzzle (Parametrizat): ' || v\_id\_puzzle\_param || ', Solutie: ' || v\_solutie\_param ||

', Pozitie: ' || v\_pozitie\_param || ', Rating: ' || v\_rating\_param);

-- Adăugăm ID-ul puzzle-ului în intersecție dacă se găsește în ambele cursore

IF v\_id\_puzzle\_param = v\_id\_puzzle\_loop THEN

v\_intersected\_ids.EXTEND;

v\_intersected\_ids(v\_intersected\_ids.LAST) := v\_id\_puzzle\_param;

END IF;

END LOOP;

-- Închidem cursorul parametrizat înainte de a trece la următoarea iterație a ciclu cursorului

CLOSE parametrized\_cursor;

END LOOP;

-- Închidem ciclu cursorul

CLOSE loop\_cursor;

-- Afișăm puzzle-urile care se găsesc în ambele cursore

IF v\_intersected\_ids.COUNT > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Puzzle-urile din ambele cursore:');

FOR i IN 1..v\_intersected\_ids.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('ID\_Puzzle: ' || v\_intersected\_ids(i));

END LOOP;

ELSE

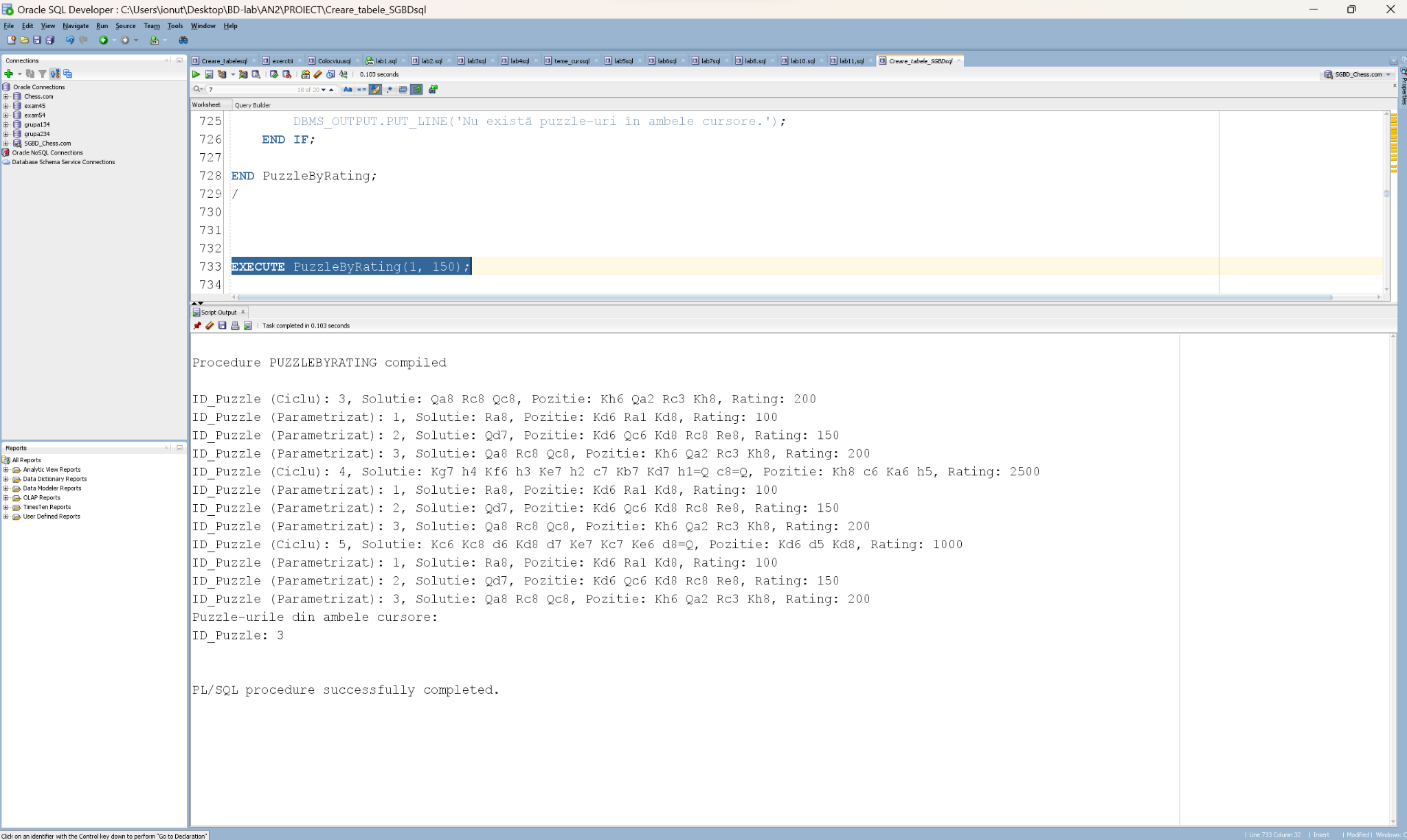
DBMS\_OUTPUT.PUT\_LINE('Nu există puzzle-uri în ambele cursore.');

END IF;

END PuzzleByRating;

/

EXECUTE PuzzleByRating(1, 150);



**8. Rezolvarea unei probleme pe care o rezolv folosind un subprogram stocat independent de tip funcție care să utilizeze într-o singură comandă SQL 3 dintre tabelele definite + definirea a minim 2 excepții proprii + apelarea subprogramul astfel încât să evidențiez toate cazurile definite și tratate**

7 tabele!!

Cerință:

Selectați username-ul utilizatorilor care au primit o notificare cu un text în care se menționează că participă maeștri la un turneu ce va începe în curând și titlul notificării are cuvântul cheie *Titled* și care a făcut o plată de 15$ și care a inițiat o conversație cu utilizatorul cu id-ul 2 și care a inițiat o conversatie cu utilizatorul cu numărul 4. Cele 2 ***excepții*** vor fi când utilizatorul dat ca parametru nu există în baza de date și când nu există astfel de utilizatori care să respecte toate condițiile.

--Exercitiul 8

CREATE OR REPLACE FUNCTION GetUsernamesForNotification(

p\_id\_utilizator NUMBER

)

RETURN SYS\_REFCURSOR

IS

-- Declaram variabila pentru cursorul de rezultate

user\_cursor SYS\_REFCURSOR;

-- Declaram variabila pentru a stoca rezultatul pentru exceptia 1

v\_user\_count NUMBER;

-- Declaram variabile pentru tratarea exceptiilor

custom\_exception1 EXCEPTION;

custom\_exception2 EXCEPTION;

BEGIN

-- Verificam daca utilizatorul exista in Utilizatori

SELECT COUNT(\*)

INTO v\_user\_count

FROM Utilizatori

WHERE id\_utilizator = p\_id\_utilizator;

IF v\_user\_count = 0 THEN

-- Daca nu exista, ridicam o exceptie

RAISE custom\_exception1;

END IF;

-- Deschidem cursorul de rezultate

OPEN user\_cursor FOR

SELECT DISTINCT u.username

FROM Utilizatori u

LEFT OUTER JOIN UtilizatoriNotificari UN ON u.id\_utilizator = UN.id\_utilizator

RIGHT OUTER JOIN Plati P ON u.id\_utilizator = P.id\_utilizator

FULL OUTER JOIN Conversatii C ON u.id\_utilizator = C.id\_utilizator1

LEFT OUTER JOIN Prietenie PR ON u.id\_utilizator = PR.id\_utilizator1

WHERE

u.id\_utilizator = p\_id\_utilizator

AND u.id\_utilizator IN (

SELECT DISTINCT UN.id\_utilizator

FROM UtilizatoriNotificari UN

LEFT OUTER JOIN Notificari N ON UN.id\_notificare = N.id\_notificare

LEFT OUTER JOIN Mesaj M ON UN.id\_notificare = M.id\_notificare

WHERE N.titlu LIKE '%Titled%'

AND M.text LIKE '%maestri%'

)

AND (P.suma = 15 OR P.suma IS NULL)

AND (C.id\_utilizator2 = 2 OR C.id\_utilizator2 IS NULL)

AND (PR.id\_utilizator2 = 4 OR PR.id\_utilizator2 IS NULL);

-- Verificam daca cursorul a fost deschis corect

IF user\_cursor%ISOPEN AND user\_cursor%NOTFOUND THEN

-- Ridicam o exceptie pentru niciun rezultat gasit

RAISE custom\_exception2;

END IF;

-- Returnam cursorul de rezultate

RETURN user\_cursor;

EXCEPTION

WHEN custom\_exception1 THEN

-- Tratam exceptia pentru utilizatorul care nu exista

DBMS\_OUTPUT.PUT\_LINE('Utilizatorul nu exista in Baza de date.');

RETURN NULL; -- Adaugam aceasta instructiune pentru a trata exceptia si a evita eroarea ORA-06503

WHEN custom\_exception2 THEN

-- Tratam exceptia pentru cursorul care nu a gasit rezultate

DBMS\_OUTPUT.PUT\_LINE('Nu s-au gasit rezultate pentru utilizatorul dat.');

CLOSE user\_cursor; -- Inchidem cursorul in caz de exceptie

RETURN NULL; -- Adaugam aceasta instructiune pentru a trata exceptia si a evita eroarea ORA-06503

WHEN OTHERS THEN

-- Tratam alte exceptii neasteptate

DBMS\_OUTPUT.PUT\_LINE('O eroare neasteptata a aparut.');

CLOSE user\_cursor; -- Inchidem cursorul in caz de exceptie

RETURN NULL; -- Adaugam aceasta instructiune pentru a trata exceptia si a evita eroarea ORA-06503

END GetUsernamesForNotification;

/

DECLARE

v\_result\_cursor SYS\_REFCURSOR;

v\_username VARCHAR2(100);

BEGIN

-- Apel corect

v\_result\_cursor := GetUsernamesForNotification(1);

-- Verificam daca cursorul este deschis

IF v\_result\_cursor%ISOPEN THEN

-- Parcurgem cursorul in bucla LOOP

LOOP

FETCH v\_result\_cursor INTO v\_username;

EXIT WHEN v\_result\_cursor%NOTFOUND;

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

END LOOP;

-- Inchidem cursorul

CLOSE v\_result\_cursor;

END IF;

END;

/

DECLARE

v\_result\_cursor SYS\_REFCURSOR;

v\_username VARCHAR2(100);

BEGIN

-- Apel incorect

v\_result\_cursor := GetUsernamesForNotification(2);

-- Verificam daca cursorul este deschis

IF v\_result\_cursor%ISOPEN THEN

-- Parcurgem cursorul in bucla LOOP

LOOP

FETCH v\_result\_cursor INTO v\_username;

EXIT WHEN v\_result\_cursor%NOTFOUND;

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

END LOOP;

CLOSE v\_result\_cursor;

END IF;

END;

/

DECLARE

v\_result\_cursor SYS\_REFCURSOR;

v\_username VARCHAR2(100);

BEGIN

-- Apel incorect

v\_result\_cursor := GetUsernamesForNotification(6);

-- Verificam daca cursorul este deschis

IF v\_result\_cursor%ISOPEN THEN

-- Parcurgem cursorul in bucla LOOP

LOOP

FETCH v\_result\_cursor INTO v\_username;

EXIT WHEN v\_result\_cursor%NOTFOUND;

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

END LOOP;

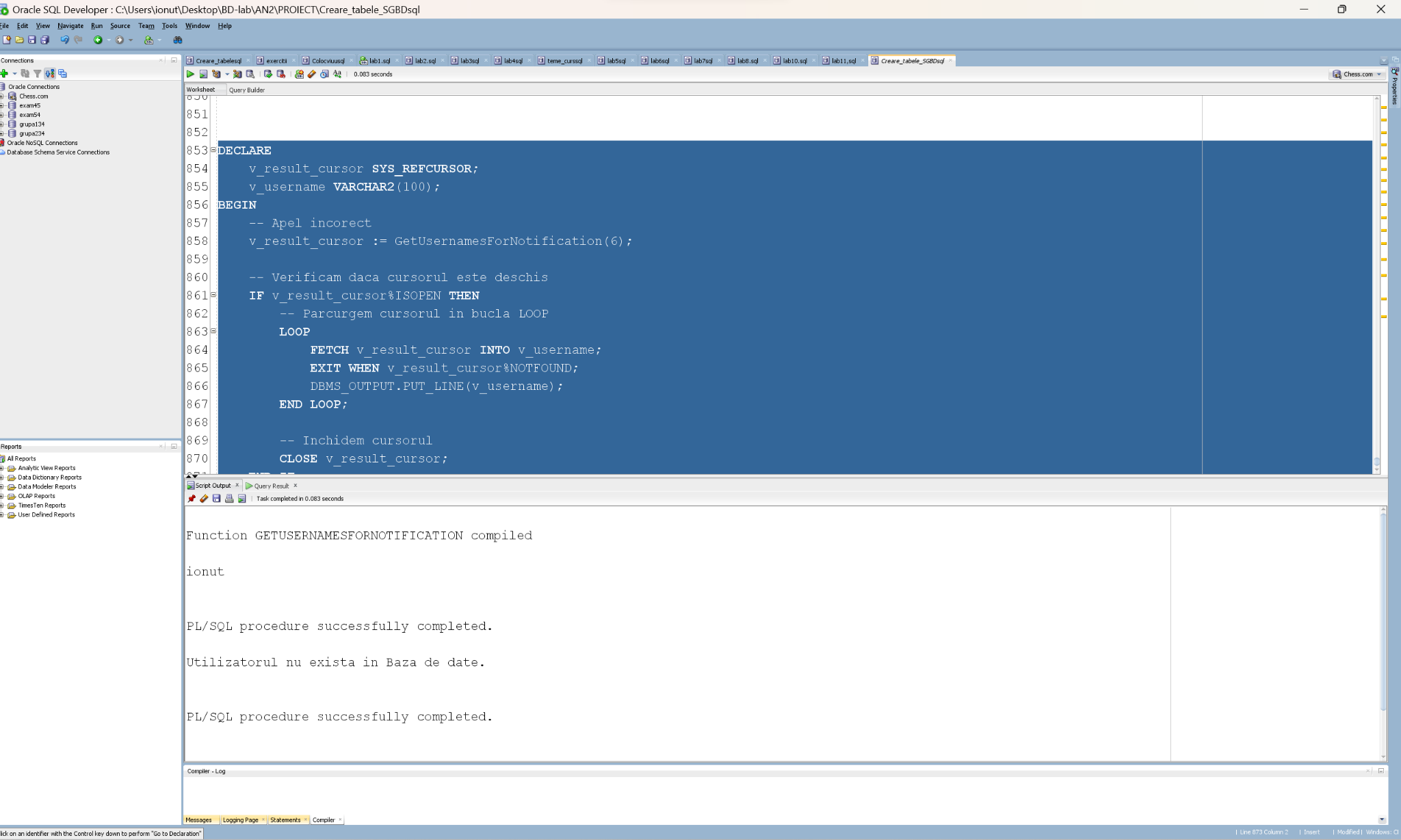
-- Inchidem cursorul

CLOSE v\_result\_cursor;

END IF;

END;

/



**9. Rezolvarea unei probleme pe care o rezolv folosind un subprogram stocat independent de tip procedură care să utilizeze într-o singură comandă SQL 5 dintre tabelele definite + tratarea tuturor excepțiilor care pot apărea, incluzând excepțiile NO\_DATA\_FOUND și TOO\_MANY\_ROWS + apelarea subprogramului astfel încât să evidențiez toate cazurile tratate**

Cerință:  
  
Să se scrie un subprogram stocat independent de tip procedură care primește ca parametru un id\_utilizator și să verifice următoarele condiții pentru acesta: Dacă există încercări ale acestui utilizator de a rezolva măcar un puzzle care în poziția de start are un turn pe a1 (în componenta atributului poziție e secvența Ra1) și care are lungimea username-ului 4 sau 5 și care a jucat în măcar un turneu care a început în 2023 și care a inițiat o prietenie în martie aprilie sau iunie. Dacă toate aceste condiții sunt adevărate, să se afișeze un mesaj corespunzător și să se afișeze ce plată a efectuat acest utilizator, iar dacă a efectuat mai multe plăți, să se intre pe excepția TOO MANY ROWS și dacă utilizatorul dat ca parametru nu respectă toate condițiile, atunci să se intre pe excepția NO DATA FOUND.

--Cerinta 9

CREATE OR REPLACE PROCEDURE CheckUserCriteria(

p\_id\_utilizator UtilizatoriTurnee.id\_utilizator%TYPE

)

AS

v\_user\_id UtilizatoriTurnee.id\_utilizator%TYPE;

v\_payment\_id Plati.id\_plata%TYPE;

BEGIN

SELECT DISTINCT(id\_utilizator)

INTO v\_user\_id

FROM UtilizatoriTurnee

WHERE id\_utilizator = p\_id\_utilizator

AND id\_turneu IN

(SELECT id\_turneu

FROM Turnee

WHERE data\_start >= TO\_DATE('01-JAN-2023', 'DD-MON-YYYY'))

AND id\_utilizator IN

(SELECT id\_utilizator

FROM Utilizatori

WHERE (LENGTH(username) = 4 OR LENGTH(username) = 5)

AND id\_utilizator IN

(SELECT id\_utilizator

FROM UtilizatoriPUZZLES

WHERE id\_puzzle =

(SELECT id\_puzzle

FROM Puzzles

WHERE Pozitie LIKE '%Ra1%')))

AND id\_utilizator IN

(SELECT DISTINCT(id\_utilizator1)

FROM Prietenie

WHERE EXTRACT (MONTH FROM data\_inceput) IN (3, 4, 6));

DBMS\_OUTPUT.PUT\_LINE('Utilizatorul cu ID ' || v\_user\_id || ' corespunde criteriilor.');

SELECT id\_plata

INTO v\_payment\_id

FROM Plati

WHERE p\_id\_utilizator = id\_utilizator;

DBMS\_OUTPUT.PUT\_LINE('ID Plata: ' || v\_payment\_id);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Utilizatorul cu ID ' || p\_id\_utilizator || ' nu corespunde criteriilor.');

WHEN TOO\_MANY\_ROWS THEN

DBMS\_OUTPUT.PUT\_LINE('Prea multe înregistrări întoarse pentru utilizator.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('O eroare neașteptată a apărut: ' || SQLERRM);

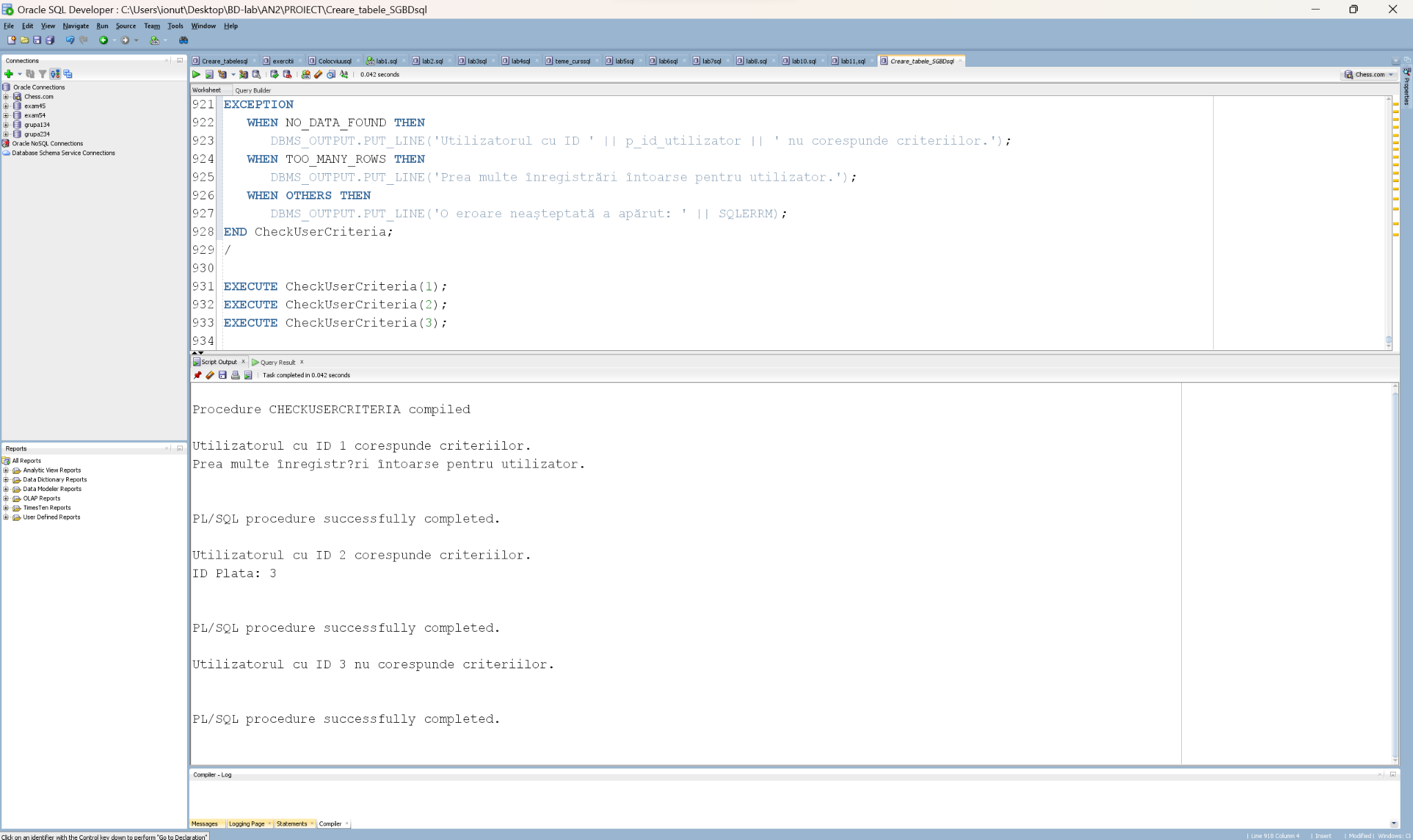
END CheckUserCriteria;

/

EXECUTE CheckUserCriteria(1);

EXECUTE CheckUserCriteria(2);

EXECUTE CheckUserCriteria(3);



**10. Definirea unui trigger de tip LMD la nivel de comandă + declanșarea trigger-ului**

Cerință:

Când aplicația Chess.com se updatează, niciun nou utilizator nu poate să-și creeze un cont. De aceea ar fi o idee bună să creăm un trigger care împiedică orice utilizator să-și facă cont. Definiți un trigger de tip LMD la nivel de comandă care împiedică inserarea oricărui nou utilizator și declanșați triggerul.

-- Definirea trigger-ului

CREATE OR REPLACE TRIGGER limit\_users

BEFORE INSERT ON Utilizatori

DECLARE

v\_user\_count NUMBER;

BEGIN

-- Verificăm numărul curent de utilizatori

SELECT COUNT(\*) INTO v\_user\_count FROM Utilizatori;

-- Ridicăm o excepție dacă numărul de utilizatori depășește limita curentă

IF v\_user\_count > 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Nu se pot adăuga noi utilizatori. Limita a fost atinsă.');

END IF;

END;

/

-- Declanșarea trigger-ului

BEGIN

-- Acest bloc încearcă să adauge un utilizator și va declanșa trigger-ul

INSERT INTO Utilizatori VALUES (6, 'newuser', 'newuser@mail.com', 'newpassword');

EXCEPTION

WHEN OTHERS THEN

-- Capturăm și afișăm mesajul de eroare generat de trigger

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

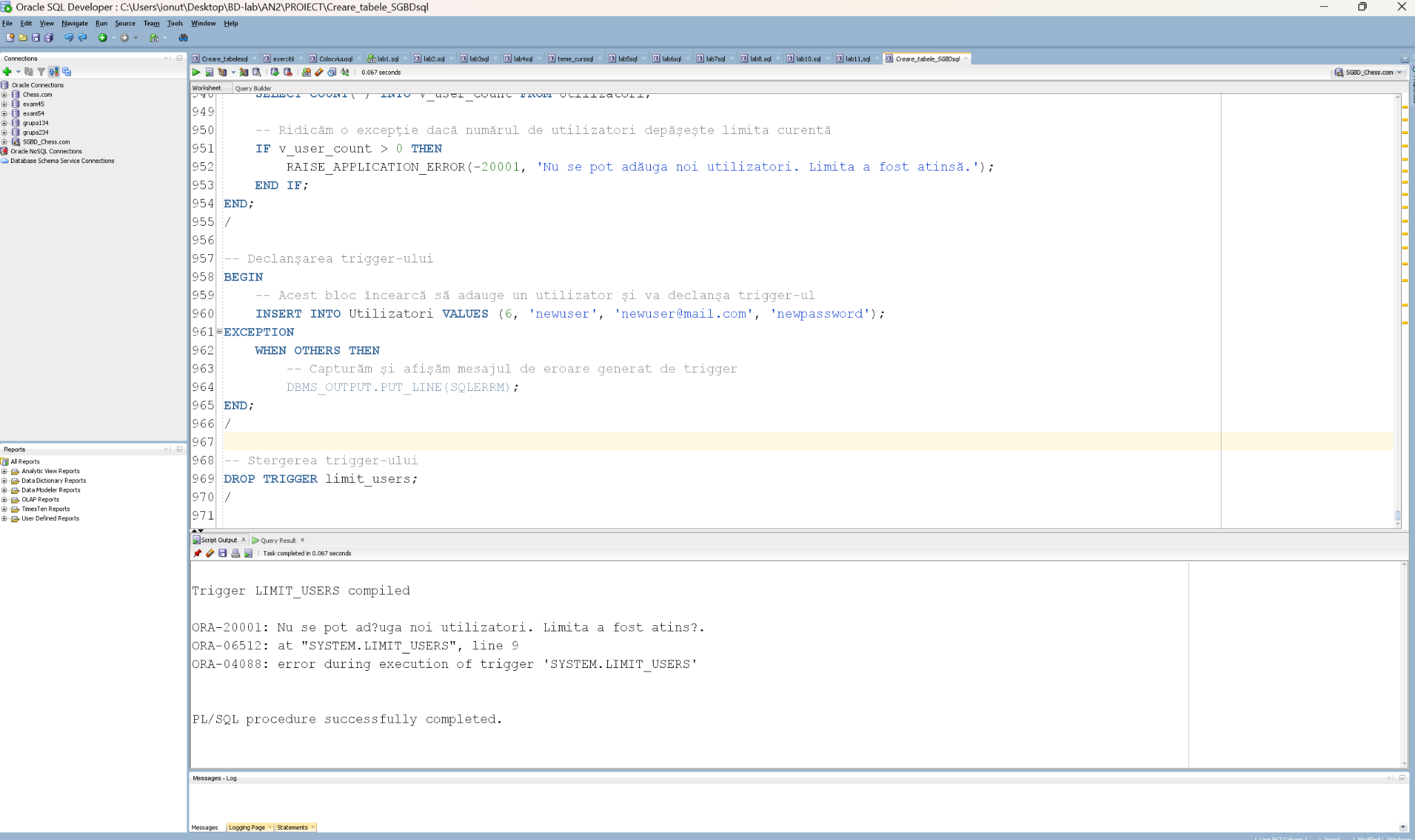
END;

/

-- Stergerea trigger-ului

DROP TRIGGER limit\_users;

/



**11. Definirea unui trigger de tip LMD la nivel de linie + declanșarea trigger-ului**

Cerință:

Implementați un trigger de tip LMD care să asigure integritatea datelor în tabela UtilizatoriNotificari. Trigger-ul trebuie să prevină inserarea unui utilizator în această tabelă care nu exista în tabela Utilizatori sau inserarea unei notificări care nu există în tabela Notificari. De asemenea, trigger-ul trebuie să prevină ștergerea unui utilizator sau a unei notificări din tabela UtilizatoriNotificari, în cazul în care există înregistrări asociate în tabela Utilizatori sau Notificari.

--Cerinta 11

CREATE OR REPLACE TRIGGER UtilizatoriNotificari\_Trigger

BEFORE INSERT OR DELETE ON UtilizatoriNotificari

FOR EACH ROW

DECLARE

v\_utilizator\_count NUMBER;

v\_notificare\_count NUMBER;

BEGIN

IF INSERTING THEN

-- Verifica daca utilizatorul exista in tabela Utilizatori

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

FROM Utilizatori

WHERE id\_utilizator = :NEW.id\_utilizator;

IF v\_utilizator\_count = 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Utilizatorul nu exista in tabela Utilizatori.');

END IF;

-- Verifica daca notificarea exista in tabela Notificari

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

FROM Notificari

WHERE id\_notificare = :NEW.id\_notificare;

IF v\_notificare\_count = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Notificarea nu exista in tabela Notificari.');

END IF;

ELSIF DELETING THEN

-- Verifica daca exista inregistrari asociate in tabela UtilizatoriNotificari

IF :OLD.id\_utilizator IS NOT NULL AND :OLD.id\_notificare IS NOT NULL THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Nu poti sterge un utilizator sau notificare care are inregistrari asociate in tabela UtilizatoriNotificari.');

END IF;

END IF;

END;

/

-- Operatiune de insert care nu e corecta din pricina faptului ca utilizatorul 6 nu e in Utilizatori

INSERT INTO UtilizatoriNotificari VALUES (6, 2);

-- Operatiune de insert care nu e corecta din pricina faptului ca notificarea 6 nu e in Notificari

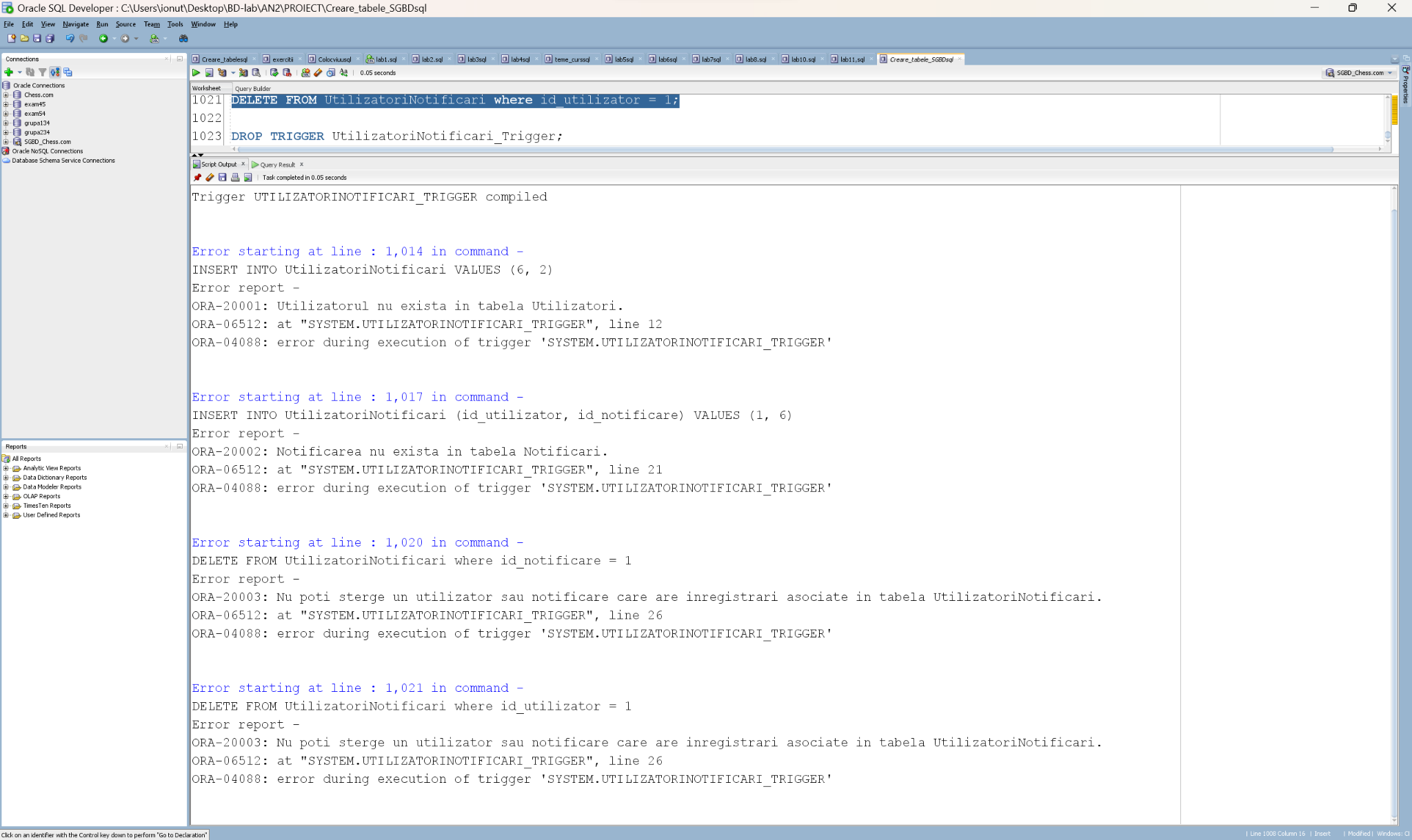
INSERT INTO UtilizatoriNotificari (id\_utilizator, id\_notificare) VALUES (1, 6);

--Nu putem da DELETE din cauza Trigger-ului

DELETE FROM UtilizatoriNotificari where id\_notificare = 1;

DELETE FROM UtilizatoriNotificari where id\_utilizator = 1;

DROP TRIGGER UtilizatoriNotificari\_Trigger;



**12. Definirea unui trigger de tip LDD + declanșarea trigger-ului**

Cerință:

Deoarece nu este niciun angajat online la sediul Chess.com între orele 19:40 și 20:10, baza de date este vulnerabilă atacurilor unor Hackeri! De aceea, ar fi o idee bună să avem un trigger care să insereze într-o tabelă nouă toate acțiunile pe care le face oricine (căci nefiind nimeni online sigur e un hacker) pe baza noastră de date din interiorul acestui program.

CREATE TABLE modifications\_history (

utilizator VARCHAR(30),

nume\_bd VARCHAR(50),

eveniment VARCHAR(20),

nume\_obiect VARCHAR(30),

data DATE

);

CREATE OR REPLACE TRIGGER modifications

BEFORE CREATE OR DROP OR ALTER ON SCHEMA

BEGIN

-- Obținem ora și minutele curente

DECLARE

v\_current\_hour INT;

v\_current\_minute INT;

BEGIN

SELECT TO\_NUMBER(TO\_CHAR(SYSDATE, 'HH24')), TO\_NUMBER(TO\_CHAR(SYSDATE, 'MI'))

INTO v\_current\_hour, v\_current\_minute FROM dual;

-- Verificăm dacă ora și minutele curente sunt în intervalul 19:40 - 20:10

IF (v\_current\_hour = 19 AND v\_current\_minute >= 40) OR

(v\_current\_hour = 20 AND v\_current\_minute <= 10) THEN

-- Executăm acțiunea de inserare doar în acest interval

INSERT INTO modifications\_history VALUES (SYS.LOGIN\_USER, SYS.DATABASE\_NAME, SYS.SYSEVENT, SYS.DICTIONARY\_OBJ\_NAME, SYSDATE);

END IF;

END;

END;

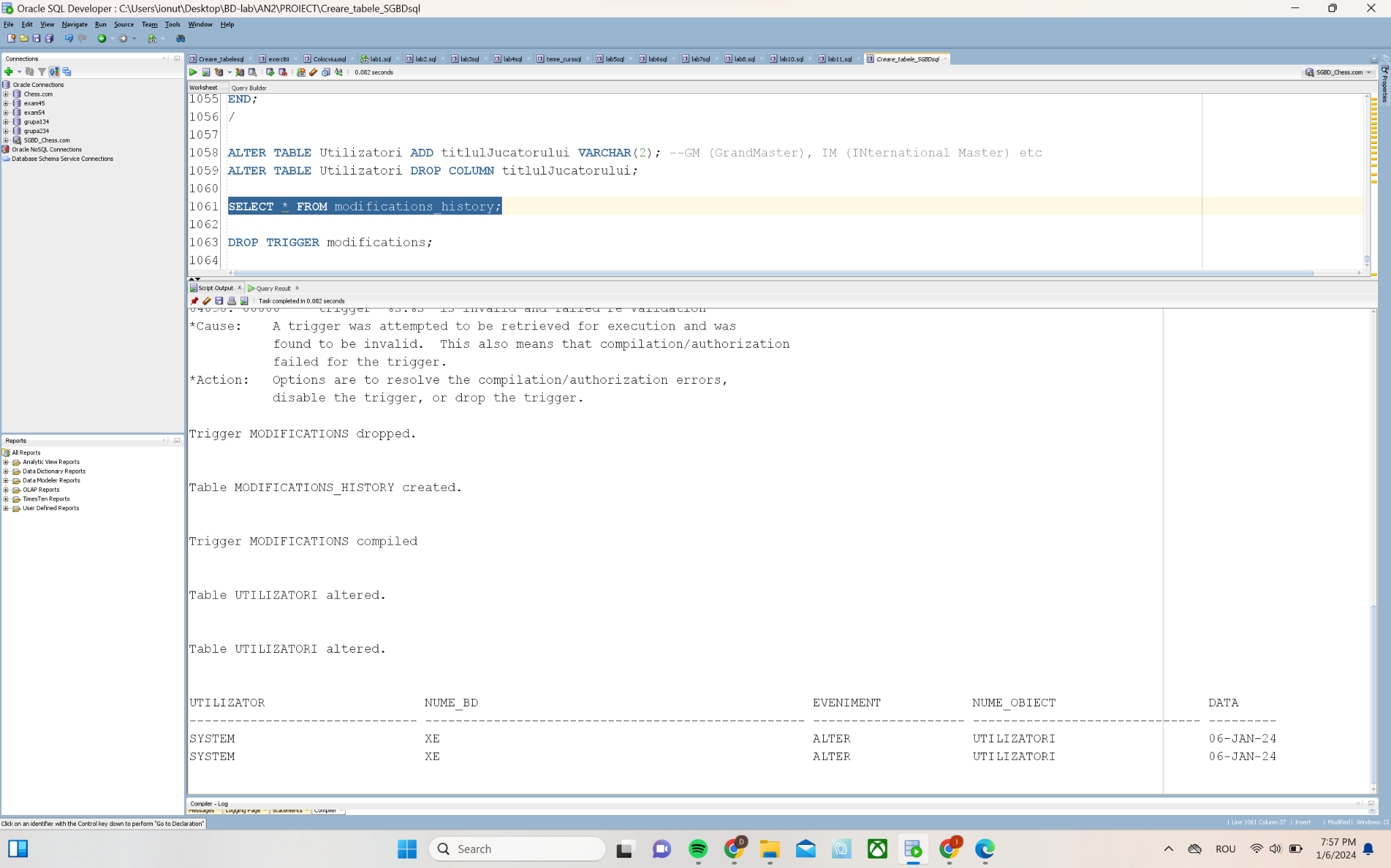
/

ALTER TABLE Utilizatori ADD titlulJucatorului VARCHAR(2); --GM (GrandMaster), IM (INternational Master) etc

ALTER TABLE Utilizatori DROP COLUMN titlulJucatorului;

SELECT \* FROM modifications\_history;

DROP TRIGGER modifications;



**13. Definirea unui pachet care să conțină toate obiectele definite în cadrul proiectului**

--Cerinta 13

CREATE OR REPLACE PACKAGE cerinta13 AS

PROCEDURE StatisticiPuzzle(p\_utilizator\_id Utilizatori.id\_utilizator%TYPE);

PROCEDURE PuzzleByRating(p\_user\_id NUMBER, p\_rating NUMBER);

FUNCTION GetUsernamesForNotification(p\_id\_utilizator NUMBER) RETURN SYS\_REFCURSOR;

PROCEDURE CheckUserCriteria(p\_id\_utilizator UtilizatoriTurnee.id\_utilizator%TYPE);

END cerinta13;

/

CREATE OR REPLACE PACKAGE BODY cerinta13 AS

--6

PROCEDURE StatisticiPuzzle(

p\_utilizator\_id Utilizatori.id\_utilizator%TYPE

) AS

TYPE PuzzleStatType IS RECORD (

id\_puzzle Puzzles.id\_puzzle%TYPE,

rating Puzzles.rating%TYPE

);

TYPE PuzzleStatList IS TABLE OF PuzzleStatType; --nested table

TYPE PuzzleStatVArray IS VARRAY(100) OF PuzzleStatType; --varray

TYPE PuzzleStatIndexBy IS TABLE OF PuzzleStatType INDEX BY PLS\_INTEGER; --index by table

v\_numar\_total\_puzzleuri UtilizatoriPuzzles.id\_puzzle%TYPE := 0;

v\_suma\_rating NUMBER := 0;

v\_max\_rating Puzzles.rating%TYPE := NULL;

--initializare pentru cele 3 colectii

v\_puzzleuriList PuzzleStatList := PuzzleStatList();

v\_puzzleuriVArray PuzzleStatVArray;

v\_puzzleuriIndexBy PuzzleStatIndexBy;

BEGIN

v\_puzzleuriList := PuzzleStatList();

v\_puzzleuriVArray := PuzzleStatVArray();

v\_puzzleuriIndexBy := PuzzleStatIndexBy();

FOR puzzle\_rec IN (

SELECT up.id\_puzzle, p.rating

FROM UtilizatoriPuzzles up

JOIN Puzzles p ON up.id\_puzzle = p.id\_puzzle

WHERE up.id\_utilizator = p\_utilizator\_id

)

LOOP

v\_numar\_total\_puzzleuri := v\_numar\_total\_puzzleuri + 1;

v\_suma\_rating := v\_suma\_rating + puzzle\_rec.rating;

IF v\_max\_rating IS NULL OR puzzle\_rec.rating > v\_max\_rating THEN

v\_max\_rating := puzzle\_rec.rating;

END IF;

--populare

v\_puzzleuriList.EXTEND;

v\_puzzleuriList(v\_numar\_total\_puzzleuri).id\_puzzle := puzzle\_rec.id\_puzzle;

v\_puzzleuriList(v\_numar\_total\_puzzleuri).rating := puzzle\_rec.rating;

v\_puzzleuriVArray.EXTEND;

v\_puzzleuriVArray(v\_puzzleuriVArray.LAST) := puzzle\_rec;

v\_puzzleuriIndexBy(v\_numar\_total\_puzzleuri) := puzzle\_rec;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Num?rul total de puzzle-uri rezolvate: ' || v\_numar\_total\_puzzleuri);

IF v\_numar\_total\_puzzleuri > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Media rating-ului puzzle-urilor rezolvate: ' || ROUND(v\_suma\_rating / v\_numar\_total\_puzzleuri, 2));

DBMS\_OUTPUT.PUT\_LINE('Cel mai dificil puzzle rezolvat (cel cu rating-ul maxim): ' || v\_max\_rating);

DBMS\_OUTPUT.PUT\_LINE('Lista de puzzle-uri rezolvate de utilizator, împreun? cu rating-ul lor (nested table):');

FOR i IN 1..v\_numar\_total\_puzzleuri LOOP

DBMS\_OUTPUT.PUT\_LINE('ID Puzzle: ' || v\_puzzleuriList(i).id\_puzzle || ', Rating: ' || v\_puzzleuriList(i).rating);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Lista de puzzle-uri rezolvate de utilizator, împreun? cu rating-ul lor (VARRAY):');

FOR i IN 1..v\_puzzleuriVArray.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('ID Puzzle: ' || v\_puzzleuriVArray(i).id\_puzzle || ', Rating: ' || v\_puzzleuriVArray(i).rating);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Lista de puzzle-uri rezolvate de utilizator, împreun? cu rating-ul lor (Index-by table):');

FOR i IN v\_puzzleuriIndexBy.FIRST..v\_puzzleuriIndexBy.LAST LOOP

DBMS\_OUTPUT.PUT\_LINE('ID Puzzle: ' || v\_puzzleuriIndexBy(i).id\_puzzle || ', Rating: ' || v\_puzzleuriIndexBy(i).rating);

END LOOP;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Utilizatorul nu a rezolvat niciun puzzle.');

END IF;

END StatisticiPuzzle;

--7

PROCEDURE PuzzleByRating(

p\_user\_id NUMBER,

p\_rating NUMBER

) AS

-- Declaram un cursor parametrizat

CURSOR parametrized\_cursor (p\_user\_id NUMBER) IS

SELECT p.id\_puzzle, p.solutie, p.pozitie, p.rating

FROM Puzzles p

JOIN UtilizatoriPuzzles up ON up.id\_puzzle = p.id\_puzzle

WHERE up.id\_utilizator = p\_user\_id;

-- Declaram un cursor dinamic

TYPE dynamic\_cursor IS REF CURSOR;

-- Declaram un tip de tabel pentru a stoca ID-urile puzzle-urilor din cursorul dinamic

TYPE puzzle\_id\_list IS TABLE OF Puzzles.id\_puzzle%TYPE;

-- Declaram variabilele pentru citirea rezultatelor din cursorul dinamic

v\_id\_puzzle Puzzles.id\_puzzle%TYPE;

v\_solutie Puzzles.solutie%TYPE;

v\_pozitie Puzzles.pozitie%TYPE;

v\_rating Puzzles.rating%TYPE;

-- Declaram o variabila pentru a stoca ID-urile puzzle-urilor din intersectie

v\_param\_ids puzzle\_id\_list := puzzle\_id\_list();

-- Declaram o variabila pentru a stoca ID-urile puzzle-urilor din intersectie

v\_intersected\_ids puzzle\_id\_list := puzzle\_id\_list();

BEGIN

-- Iteram prin cursorul parametrizat si stocam ID-urile in variabila v\_param\_ids

FOR param\_rec IN parametrized\_cursor(p\_user\_id) LOOP

v\_param\_ids.EXTEND;

v\_param\_ids(v\_param\_ids.LAST) := param\_rec.id\_puzzle;

DBMS\_OUTPUT.PUT\_LINE('ID\_Puzzle: ' || param\_rec.id\_puzzle || ', Solutie: ' || param\_rec.solutie ||

', Pozitie: ' || param\_rec.pozitie || ', Rating: ' || param\_rec.rating);

END LOOP;

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

-- Deschidem cursorul dinamic pentru a selecta puzzle-urile cu rating-ul mai mare decat p\_rating

DECLARE

dynamic\_cursor SYS\_REFCURSOR;

BEGIN

OPEN dynamic\_cursor FOR 'SELECT id\_puzzle, solutie, pozitie, rating FROM Puzzles WHERE rating > :1' USING p\_rating;

-- Iteram prin cursorul dinamic si stocam ID-urile in variabila v\_intersected\_ids daca se afla deja in v\_param\_ids

LOOP

FETCH dynamic\_cursor INTO v\_id\_puzzle, v\_solutie, v\_pozitie, v\_rating;

EXIT WHEN dynamic\_cursor%NOTFOUND;

IF v\_id\_puzzle MEMBER OF v\_param\_ids THEN

v\_intersected\_ids.EXTEND;

v\_intersected\_ids(v\_intersected\_ids.LAST) := v\_id\_puzzle;

DBMS\_OUTPUT.PUT\_LINE('ID\_Puzzle: ' || v\_id\_puzzle || ', Solutie: ' || v\_solutie ||

', Pozitie: ' || v\_pozitie || ', Rating: ' || v\_rating);

END IF;

END LOOP;

-- Inchidem cursorul dinamic

CLOSE dynamic\_cursor;

END;

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

-- Iteram si prin intersectie si afisam informatiile

FOR i IN 1..v\_intersected\_ids.COUNT LOOP

SELECT id\_puzzle, solutie, pozitie, rating

INTO v\_id\_puzzle, v\_solutie, v\_pozitie, v\_rating

FROM Puzzles

WHERE id\_puzzle = v\_intersected\_ids(i);

DBMS\_OUTPUT.PUT\_LINE('ID\_Puzzle: ' || v\_id\_puzzle || ', Solutie: ' || v\_solutie ||

', Pozitie: ' || v\_pozitie || ', Rating: ' || v\_rating);

END LOOP;

END PuzzleByRating;

--8

FUNCTION GetUsernamesForNotification(

p\_id\_utilizator NUMBER

)

RETURN SYS\_REFCURSOR

IS

-- Declaram variabila pentru cursorul de rezultate

user\_cursor SYS\_REFCURSOR;

-- Declaram variabila pentru a stoca rezultatul pentru exceptia 1

v\_user\_count NUMBER;

-- Declaram variabile pentru tratarea exceptiilor

custom\_exception1 EXCEPTION;

custom\_exception2 EXCEPTION;

BEGIN

-- Verificam daca utilizatorul exista in Utilizatori

SELECT COUNT(\*)

INTO v\_user\_count

FROM Utilizatori

WHERE id\_utilizator = p\_id\_utilizator;

IF v\_user\_count = 0 THEN

-- Daca nu exista, ridicam o exceptie

RAISE custom\_exception1;

END IF;

-- Deschidem cursorul de rezultate

OPEN user\_cursor FOR

SELECT DISTINCT u.username

FROM Utilizatori u

LEFT OUTER JOIN UtilizatoriNotificari UN ON u.id\_utilizator = UN.id\_utilizator

RIGHT OUTER JOIN Plati P ON u.id\_utilizator = P.id\_utilizator

FULL OUTER JOIN Conversatii C ON u.id\_utilizator = C.id\_utilizator1

LEFT OUTER JOIN Prietenie PR ON u.id\_utilizator = PR.id\_utilizator1

WHERE

u.id\_utilizator = p\_id\_utilizator

AND u.id\_utilizator IN (

SELECT DISTINCT UN.id\_utilizator

FROM UtilizatoriNotificari UN

LEFT OUTER JOIN Notificari N ON UN.id\_notificare = N.id\_notificare

LEFT OUTER JOIN Mesaj M ON UN.id\_notificare = M.id\_notificare

WHERE N.titlu LIKE '%Titled%'

AND M.text LIKE '%maestri%'

)

AND (P.suma = 15 OR P.suma IS NULL)

AND (C.id\_utilizator2 = 2 OR C.id\_utilizator2 IS NULL)

AND (PR.id\_utilizator2 = 4 OR PR.id\_utilizator2 IS NULL);

-- Verificam daca cursorul a fost deschis corect

IF user\_cursor%ISOPEN AND user\_cursor%NOTFOUND THEN

-- Ridicam o exceptie pentru niciun rezultat gasit

RAISE custom\_exception2;

END IF;

-- Returnam cursorul de rezultate

RETURN user\_cursor;

EXCEPTION

WHEN custom\_exception1 THEN

-- Tratam exceptia pentru utilizatorul care nu exista

DBMS\_OUTPUT.PUT\_LINE('Utilizatorul nu exista in Baza de date.');

RETURN NULL; -- Adaugam aceasta instructiune pentru a trata exceptia si a evita eroarea ORA-06503

WHEN custom\_exception2 THEN

-- Tratam exceptia pentru cursorul care nu a gasit rezultate

DBMS\_OUTPUT.PUT\_LINE('Nu s-au gasit rezultate pentru utilizatorul dat.');

CLOSE user\_cursor; -- Inchidem cursorul in caz de exceptie

RETURN NULL; -- Adaugam aceasta instructiune pentru a trata exceptia si a evita eroarea ORA-06503

WHEN OTHERS THEN

-- Tratam alte exceptii neasteptate

DBMS\_OUTPUT.PUT\_LINE('O eroare neasteptata a aparut.');

CLOSE user\_cursor; -- Inchidem cursorul in caz de exceptie

RETURN NULL; -- Adaugam aceasta instructiune pentru a trata exceptia si a evita eroarea ORA-06503

END GetUsernamesForNotification;

--9

PROCEDURE CheckUserCriteria(

p\_id\_utilizator UtilizatoriTurnee.id\_utilizator%TYPE

)

AS

v\_user\_id UtilizatoriTurnee.id\_utilizator%TYPE;

v\_payment\_id Plati.id\_plata%TYPE;

BEGIN

SELECT DISTINCT(id\_utilizator)

INTO v\_user\_id

FROM UtilizatoriTurnee

WHERE id\_utilizator = p\_id\_utilizator

AND id\_turneu IN

(SELECT id\_turneu

FROM Turnee

WHERE data\_start >= TO\_DATE('01-JAN-2023', 'DD-MON-YYYY'))

AND id\_utilizator IN

(SELECT id\_utilizator

FROM Utilizatori

WHERE (LENGTH(username) = 4 OR LENGTH(username) = 5)

AND id\_utilizator IN

(SELECT id\_utilizator

FROM UtilizatoriPUZZLES

WHERE id\_puzzle =

(SELECT id\_puzzle

FROM Puzzles

WHERE Pozitie LIKE '%Ra1%')))

AND id\_utilizator IN

(SELECT DISTINCT(id\_utilizator1)

FROM Prietenie

WHERE EXTRACT (MONTH FROM data\_inceput) IN (3, 4, 6));

DBMS\_OUTPUT.PUT\_LINE('Utilizatorul cu ID ' || v\_user\_id || ' corespunde criteriilor.');

SELECT id\_plata

INTO v\_payment\_id

FROM Plati

WHERE p\_id\_utilizator = id\_utilizator;

DBMS\_OUTPUT.PUT\_LINE('ID Plata: ' || v\_payment\_id);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Utilizatorul cu ID ' || p\_id\_utilizator || ' nu corespunde criteriilor.');

WHEN TOO\_MANY\_ROWS THEN

DBMS\_OUTPUT.PUT\_LINE('Prea multe înregistr?ri întoarse pentru utilizator.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('O eroare nea?teptat? a ap?rut: ' || SQLERRM);

END CheckUserCriteria;

END cerinta13;

/

--Verificare ex 6

EXECUTE cerinta13.StatisticiPuzzle(1);

--Verificare ex 7

EXECUTE cerinta13.PuzzleByRating(1, 150);

--Verificare ex 8

DECLARE

v\_result\_cursor SYS\_REFCURSOR;

v\_username VARCHAR2(100);

BEGIN

-- Apel corect

v\_result\_cursor := cerinta13.GetUsernamesForNotification(1);

-- Verificam daca cursorul este deschis

IF v\_result\_cursor%ISOPEN THEN

-- Parcurgem cursorul in bucla LOOP

LOOP

FETCH v\_result\_cursor INTO v\_username;

EXIT WHEN v\_result\_cursor%NOTFOUND;

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

END LOOP;

-- Inchidem cursorul

CLOSE v\_result\_cursor;

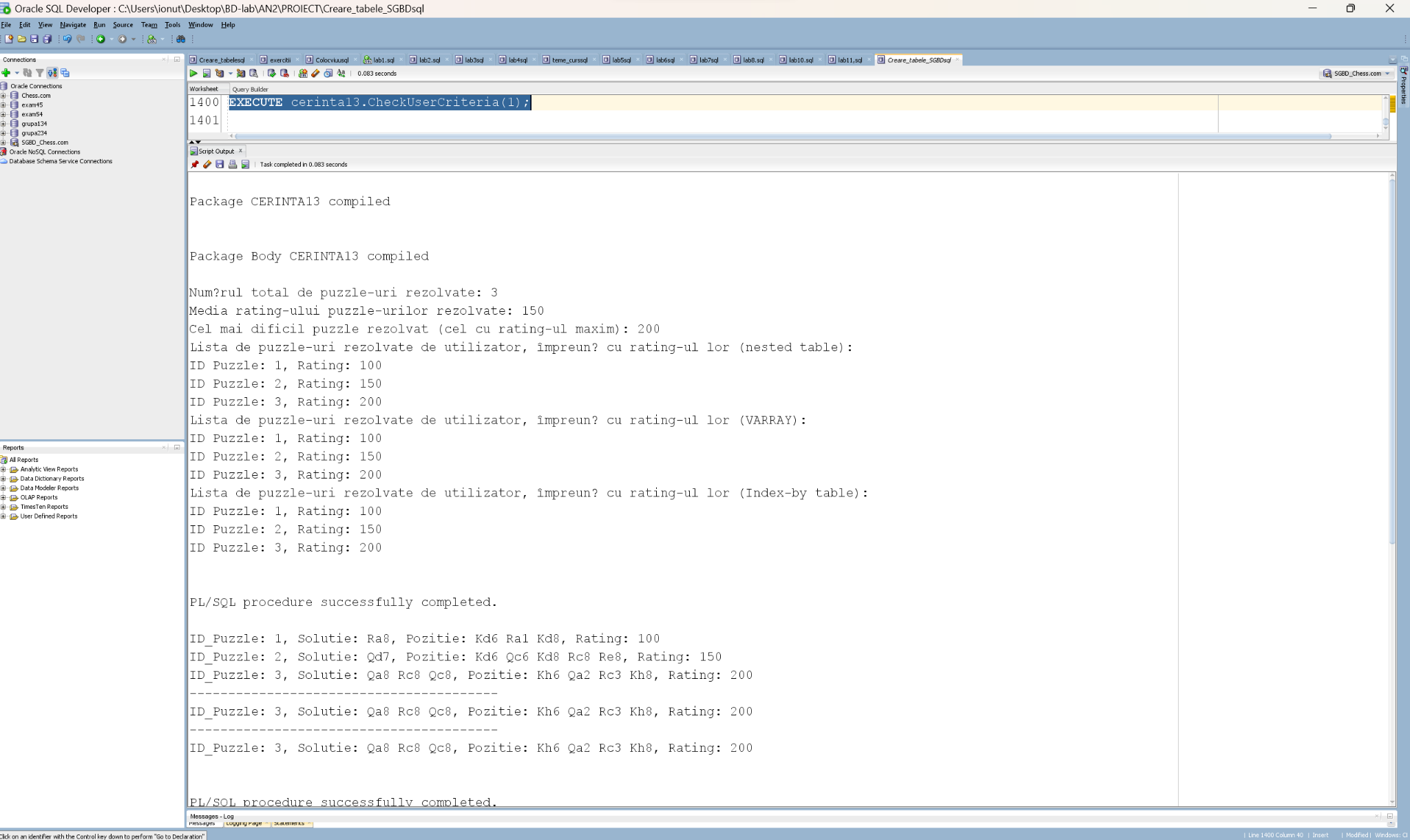
END IF;

END;

/

--Verificare ex 9

EXECUTE cerinta13.CheckUserCriteria(1);

****

**14. Definirea unui 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)**

--Cerinta 14

CREATE OR REPLACE PACKAGE cerinta14 AS

-- Tipuri de date

TYPE PuzzleStatType IS RECORD (

id\_puzzle Puzzles.id\_puzzle%TYPE,

rating Puzzles.rating%TYPE

);

TYPE StringList IS TABLE OF VARCHAR2(100) INDEX BY PLS\_INTEGER;

--TYPE VARRAY

FUNCTION GetUsernamesForNotification(p\_id\_utilizator NUMBER) RETURN StringList;

FUNCTION ProcessUser(p\_id\_utilizator Utilizatori.id\_utilizator%TYPE) RETURN VARCHAR2;

-- Proceduri

PROCEDURE StatisticiPuzzle(p\_utilizator\_id Utilizatori.id\_utilizator%TYPE);

PROCEDURE CheckUserCriteria(p\_id\_utilizator UtilizatoriTurnee.id\_utilizator%TYPE);

END cerinta14;

/

-- Implementarea pachetului

CREATE OR REPLACE PACKAGE BODY cerinta14 AS

FUNCTION GetUsernamesForNotification(

p\_id\_utilizator NUMBER

)

RETURN StringList

IS

v\_usernames StringList := StringList();

v\_user\_count NUMBER;

custom\_exception1 EXCEPTION;

custom\_exception2 EXCEPTION;

BEGIN

SELECT COUNT(\*)

INTO v\_user\_count

FROM Utilizatori

WHERE id\_utilizator = p\_id\_utilizator;

IF v\_user\_count = 0 THEN

RAISE custom\_exception1;

END IF;

-- Preluăm datele în colecția de șiruri

SELECT DISTINCT u.username

BULK COLLECT INTO v\_usernames

FROM Utilizatori u

LEFT OUTER JOIN UtilizatoriNotificari UN ON u.id\_utilizator = UN.id\_utilizator

RIGHT OUTER JOIN Plati P ON u.id\_utilizator = P.id\_utilizator

FULL OUTER JOIN Conversatii C ON u.id\_utilizator = C.id\_utilizator1

LEFT OUTER JOIN Prietenie PR ON u.id\_utilizator = PR.id\_utilizator1

WHERE

u.id\_utilizator = p\_id\_utilizator

AND u.id\_utilizator IN (

SELECT DISTINCT UN.id\_utilizator

FROM UtilizatoriNotificari UN

LEFT OUTER JOIN Notificari N ON UN.id\_notificare = N.id\_notificare

LEFT OUTER JOIN Mesaj M ON UN.id\_notificare = M.id\_notificare

WHERE N.titlu LIKE '%Titled%'

AND M.text LIKE '%maestri%'

)

AND (P.suma = 15 OR P.suma IS NULL)

AND (C.id\_utilizator2 = 2 OR C.id\_utilizator2 IS NULL)

AND (PR.id\_utilizator2 = 4 OR PR.id\_utilizator2 IS NULL);

IF v\_usernames.COUNT = 0 THEN

RAISE custom\_exception2;

END IF;

-- Returnăm colecția de șiruri

RETURN v\_usernames;

EXCEPTION

WHEN custom\_exception1 THEN

DBMS\_OUTPUT.PUT\_LINE('Utilizatorul nu există în Baza de date.');

RETURN v\_usernames; -- Returnăm colecția goală în caz de excepție

WHEN custom\_exception2 THEN

DBMS\_OUTPUT.PUT\_LINE('Nu s-au găsit rezultate pentru utilizatorul dat.');

RETURN v\_usernames; -- Returnăm colecția goală în caz de excepție

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('O eroare neașteptată a apărut.');

RETURN v\_usernames; -- Returnăm colecția goală în caz de excepție

END GetUsernamesForNotification;

--FUNCTION2

FUNCTION ProcessUser(p\_id\_utilizator Utilizatori.id\_utilizator%TYPE) RETURN VARCHAR2 IS

v\_result VARCHAR2(200);

v\_usernames StringList;

BEGIN

-- Apelam functia pentru notificari

v\_result := 'Notificari pentru utilizatorul cu ID ' || p\_id\_utilizator || ': ';

-- Preluam datele in colectie

v\_usernames := GetUsernamesForNotification(p\_id\_utilizator);

FOR idx IN v\_usernames.FIRST..v\_usernames.LAST

LOOP

v\_result := v\_result || v\_usernames(idx) || ', ';

END LOOP;

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

-- Apelam procedura pentru statistici puzzle

StatisticiPuzzle(p\_id\_utilizator);

-- Apelam procedura pentru criterii utilizator

CheckUserCriteria(p\_id\_utilizator);

RETURN 'Procesarea utilizatorului a fost finalizata.';

END ProcessUser;

PROCEDURE StatisticiPuzzle(p\_utilizator\_id Utilizatori.id\_utilizator%TYPE) IS

TYPE PuzzleStatList IS TABLE OF PuzzleStatType INDEX BY PLS\_INTEGER;

v\_puzzleuriList PuzzleStatList;

v\_numar\_total\_puzzleuri UtilizatoriPuzzles.id\_puzzle%TYPE := 0;

v\_suma\_rating NUMBER := 0;

v\_max\_rating Puzzles.rating%TYPE := NULL;

BEGIN

FOR puzzle\_rec IN (

SELECT up.id\_puzzle, p.rating

FROM UtilizatoriPuzzles up

JOIN Puzzles p ON up.id\_puzzle = p.id\_puzzle

WHERE up.id\_utilizator = p\_utilizator\_id

)

LOOP

v\_numar\_total\_puzzleuri := v\_numar\_total\_puzzleuri + 1;

v\_suma\_rating := v\_suma\_rating + puzzle\_rec.rating;

IF v\_max\_rating IS NULL OR puzzle\_rec.rating > v\_max\_rating THEN

v\_max\_rating := puzzle\_rec.rating;

END IF;

v\_puzzleuriList(v\_numar\_total\_puzzleuri).id\_puzzle := puzzle\_rec.id\_puzzle;

v\_puzzleuriList(v\_numar\_total\_puzzleuri).rating := puzzle\_rec.rating;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Num?rul total de puzzle-uri rezolvate: ' || v\_numar\_total\_puzzleuri);

IF v\_numar\_total\_puzzleuri > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Media rating-ului puzzle-urilor rezolvate: ' || ROUND(v\_suma\_rating / v\_numar\_total\_puzzleuri, 2));

DBMS\_OUTPUT.PUT\_LINE('Cel mai dificil puzzle rezolvat (cel cu rating-ul maxim): ' || v\_max\_rating);

DBMS\_OUTPUT.PUT\_LINE('Lista de puzzle-uri rezolvate de utilizator, împreun? cu rating-ul lor (Index-by table):');

FOR i IN 1..v\_numar\_total\_puzzleuri LOOP

DBMS\_OUTPUT.PUT\_LINE('ID Puzzle: ' || v\_puzzleuriList(i).id\_puzzle || ', Rating: ' || v\_puzzleuriList(i).rating);

END LOOP;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Utilizatorul nu a rezolvat niciun puzzle.');

END IF;

END StatisticiPuzzle;

PROCEDURE CheckUserCriteria(p\_id\_utilizator UtilizatoriTurnee.id\_utilizator%TYPE) IS

v\_user\_id UtilizatoriTurnee.id\_utilizator%TYPE;

v\_payment\_id Plati.id\_plata%TYPE;

BEGIN

SELECT DISTINCT(id\_utilizator)

INTO v\_user\_id

FROM UtilizatoriTurnee

WHERE id\_utilizator = p\_id\_utilizator

AND id\_turneu IN

(SELECT id\_turneu

FROM Turnee

WHERE data\_start >= TO\_DATE('01-JAN-2023', 'DD-MON-YYYY'))

AND id\_utilizator IN

(SELECT id\_utilizator

FROM Utilizatori

WHERE (LENGTH(username) = 4 OR LENGTH(username) = 5)

AND id\_utilizator IN

(SELECT id\_utilizator

FROM UtilizatoriPUZZLES

WHERE id\_puzzle =

(SELECT id\_puzzle

FROM Puzzles

WHERE Pozitie LIKE '%Ra1%')))

AND id\_utilizator IN

(SELECT DISTINCT(id\_utilizator1)

FROM Prietenie

WHERE EXTRACT (MONTH FROM data\_inceput) IN (3, 4, 6));

DBMS\_OUTPUT.PUT\_LINE('Utilizatorul cu ID ' || v\_user\_id || ' corespunde criteriilor.');

SELECT id\_plata

INTO v\_payment\_id

FROM Plati

WHERE p\_id\_utilizator = id\_utilizator;

DBMS\_OUTPUT.PUT\_LINE('ID Plata: ' || v\_payment\_id);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Utilizatorul cu ID ' || p\_id\_utilizator || ' nu corespunde criteriilor.');

WHEN TOO\_MANY\_ROWS THEN

DBMS\_OUTPUT.PUT\_LINE('Prea multe înregistr?ri întoarse pentru utilizator.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('O eroare nea?teptat? a ap?rut: ' || SQLERRM);

END CheckUserCriteria;

END cerinta14;

/

DECLARE

v\_result VARCHAR2(200);

BEGIN

v\_result := cerinta14.ProcessUser(1);

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

END;

/

