Sisteme Avansate de Baze de Date
Temă de casă

Cerințe:

1. (1p)
   a. Să se dea un exemplu de atribut repetitiv (multivaloare) al unei entități în modelul entitate-legătură.

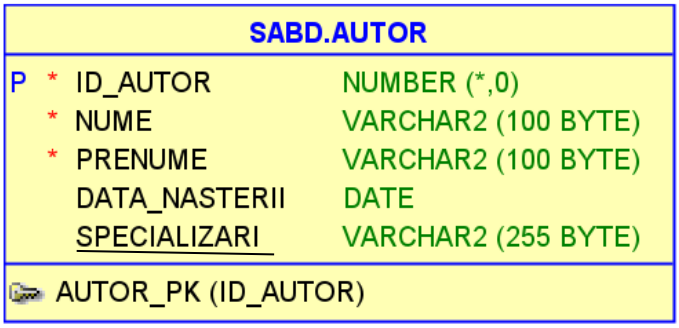


Figura 1.1.

În figura 1.1 se observă, că un autor poate avea mai multe specializări (de exemplu: ”ficțiune”, ”istorie”, ”științe sociale”), așadar atributul multivaloare care satisface cerințele este ”specializări”.

   b. Să se dea un exemplu de atribut repetitiv (multivaloare) al unei relații mulți-la-mulți în modelul entitate-legătură

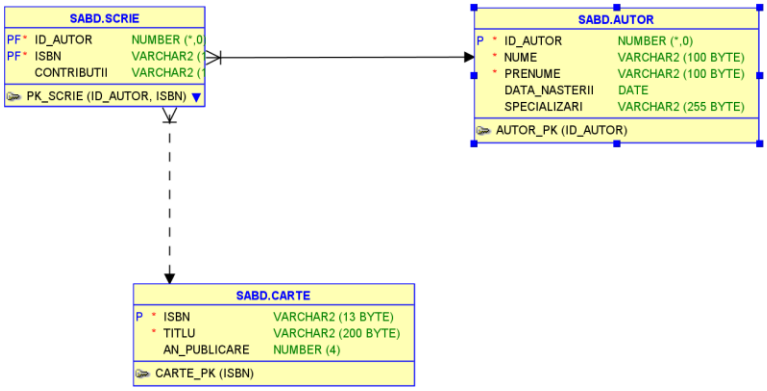


Figura 1.2.

Atributul multivaloare care satisface cerințele subpunctului b) și se observă în figura 1.2, îl constituie „contribuții”, deoarece un autor poate avea mai multe contribuții la aceeași carte (de exemplu: autor principal, coautor, editor), iar mai mulți autori pot contribui la aceeași carte, fiecare cu una sau mai multe contribuții. Această situație este modelată prin relația SCRIE dintre entitățile AUTOR și CARTE, relație de tip many-to-many, în care atributul contribuții este multivaloare.

c. Să se arate cum se transformă atributele de mai sus la crearea design-ului logic al unei baze de date relaționale.

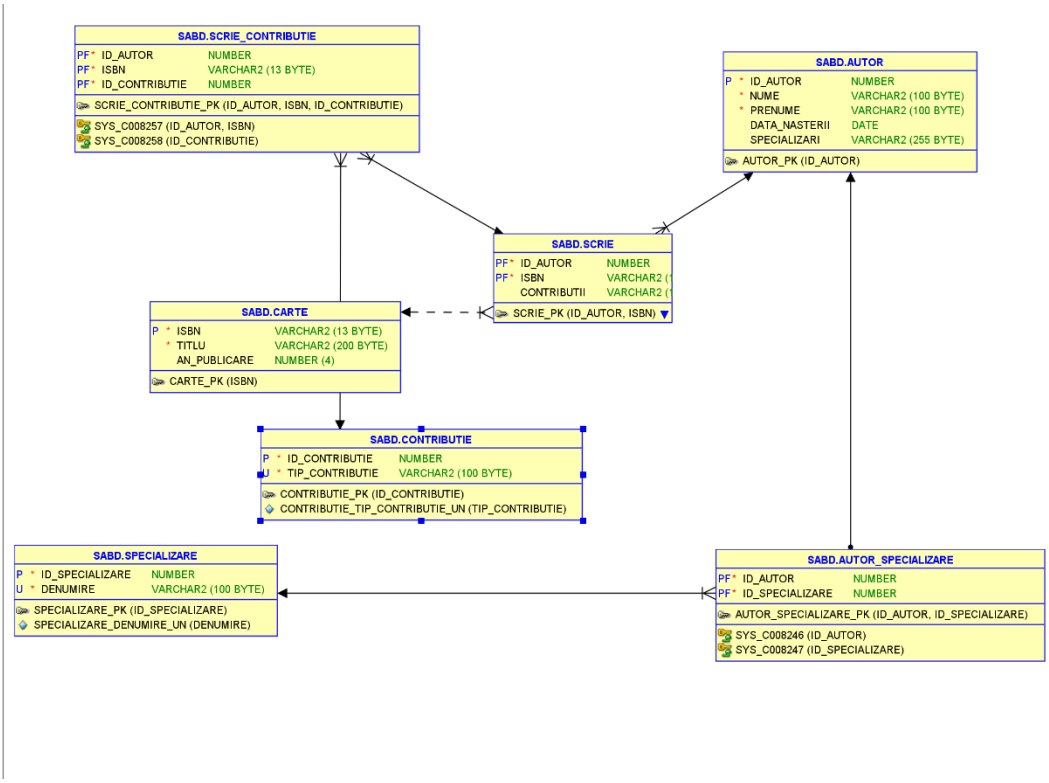


Figura 1.3.

Pentru a rezolva problema cauzată de atributul SPECIALIZARI (care conținea valori multiple separate) am creat un tabel numit "Specializare" în care am mutat denumirile specializărilor, atribuindu-le un identificator unic. Relația dintre Autori și Specializare fiind M-M (un autor poate avea mai multe specializări, iar o specializare poate fi asociată mai multor autori). Având în vedere că avem relația M-M, aceasta a trebuit împărțită în 2 relații 1-M + un tabel de legătură numit "Autor_Specializare".

Ca să rezolv problema atributului CONTRIBUTII (care apare în tabelul de legătură SCRIE și poate conține mai multe tipuri de contribuții pentru aceeași pereche autor-carte) am creat un tabel numit "Contributie" care conține tipurile posibile de contribuții. Relația dintre SCRIE și CONTRIBUTIE este 1-M (o asociere autor-carte poate implica mai multe tipuri de contribuții). Pentru a gestiona această relație am creat tabelul de legătură "Scrie_Contributie".

2. (0.5p)

   a. Să se dea un exemplu de relație de tip 3 (între mai mult de două entități) în modelul entitate-legătură.

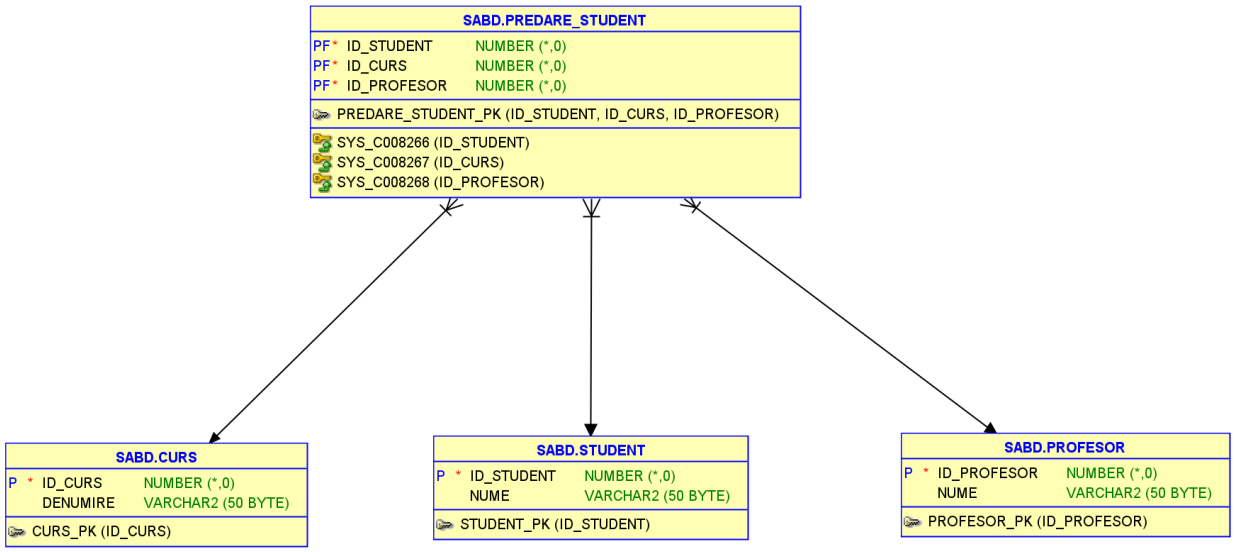


Figura 2.1.

În figura 2.1 analizăm situația în care un profesor predă un anumit curs unui anumit student. Această relație implică simultan toate cele 3 entități și nu poate fi descompusă fără pierdere de informație.

b. Să se dea un exemplu de trei sau mai multe entități care nu formează o relație de tip 3, ci, relația aparentă de tip 3, „se sparge” de fapt în relații mulți-la-mulți (între câte două entități).

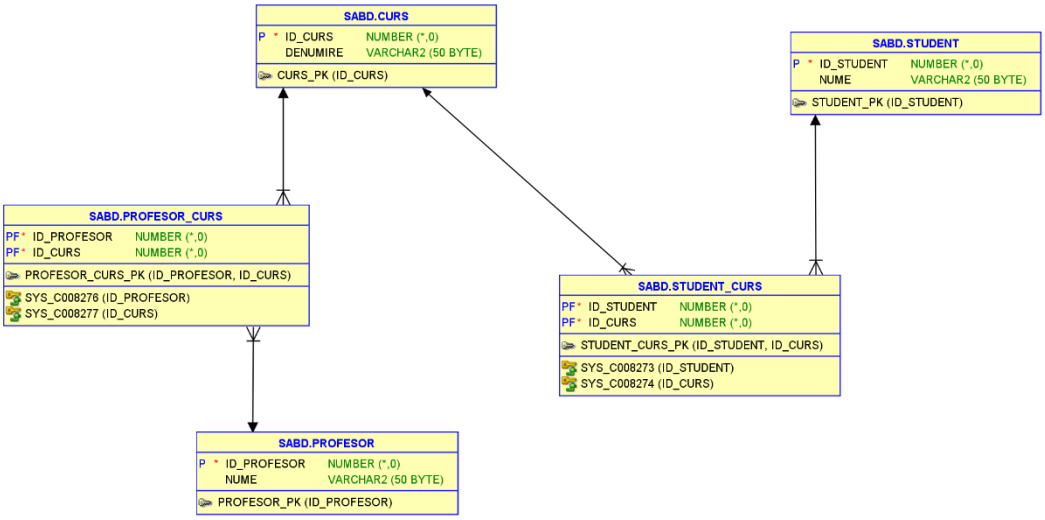


Figura 2.2.

Am modelat o situație în care entitățile Student, Curs și Profesor par să formeze o relație ternară, însă am demonstrat că aceasta „se sparge” în două relații binare independente de tip many-to-many. În loc de o singură tabelă centrală, am creat tabele de joncțiune separate: STUDENT_CURS, care gestionează înscrierile studenților la cursuri, și PROFESOR_CURS, care gestionează alocarea profesorilor la cursuri. Această abordare confirmă că legătura dintre un student și un profesor este mediată exclusiv prin entitatea Curs, neexistând o dependență atomică directă între toate cele trei elemente simultan.

3. (1p)

a. Să se dea un exemplu de tabel relațional care este în FN1, dar nu în FN2. Să se aducă tabelul în FN2.

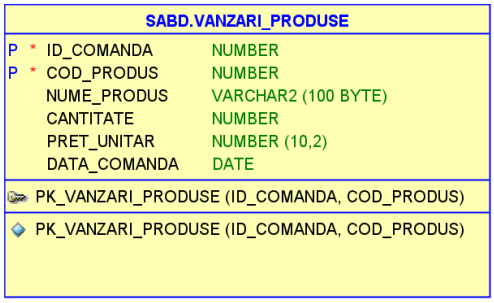


Figura 3.1. Tabelul inițial

Tabelul respectă FN1 deoarece toate atributele sunt atomice și nu există grupuri repetitive. Totuși, acesta nu respectă FN2 deoarece există dependențe parțiale față de cheia primară compusă (#ID_COMANDA, #COD_PRODUS): Atributele NUME_PRODUS și PRET_UNITAR depind doar de COD_PRODUS, nu de întreaga cheie. Ele descriu produsul în sine, indiferent de comanda în care apare acesta.

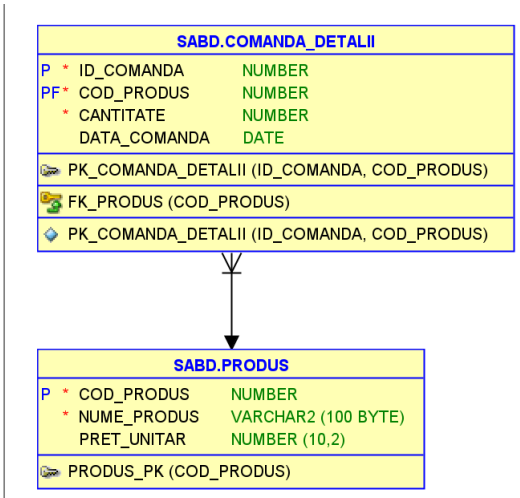


Figura 3.2. Varianta modificată

Pentru a aduce tabelul în **FN2**, am separat datele în două tabele corelate printr-o relație de tip **1-M** (un produs poate apărea în mai multe comenzi). **SABD.PRODUS**: Conține informațiile despre produs (COD_PRODUS, NUME_PRODUS, PRET_UNITAR). **SABD.COMANDA_DETALII**: Conține datele tranzacționale care depind de ambele chei (CANTITATE, DATA_COMANDA).

b. Să se dea un exemplu de tabel relațional care este în FN2, dar nu în FN3. Să se aducă tabelul în FN3.

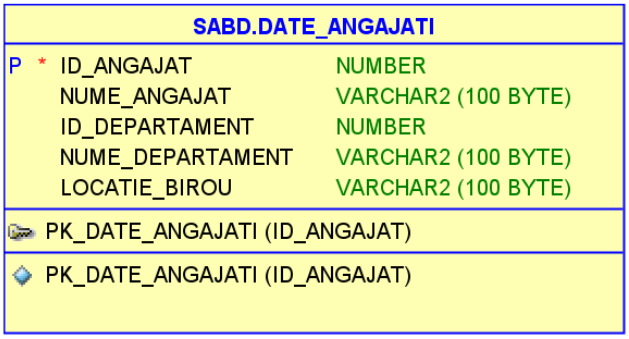


Figura 3.3. Tabelul initial

Tabelul respectă FN2 deoarece cheia primară este formată dintr-un singur atribut (#ID_ANGAJAT), deci nu pot exista dependențe parțiale. Totuși, acesta nu respectă FN3 deoarece există dependențe transitive. Atributele NUME_DEPARTAMENT și LOCATIE_BIROU depind direct de ID_DEPARTAMENT, și doar indirect (tranzitiv) de cheia primară ID_ANGAJAT.

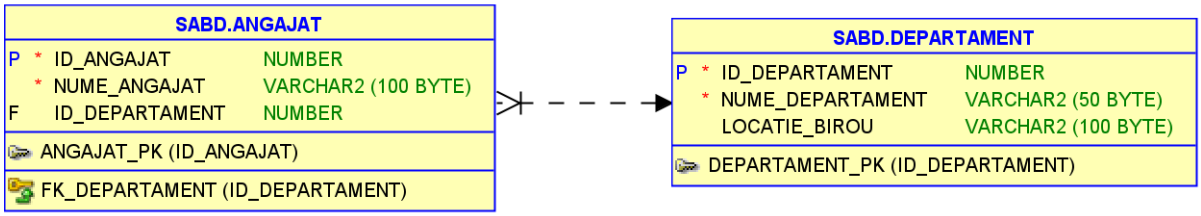


Figura 3.4 Varianta modificată

Pentru a aduce tabelul în **FN3**, am extras atributele care depindeau de ID_DEPARTAMENT într-un tabel separat, eliminând astfel redundanța. **SABD.DEPARTAMENT**: Păstrează detaliile specifice departamentului (ID_DEPARTAMENT, NUME_DEPARTAMENT, LOCATIE_BIROU). **SABD.ANGAJAT**: Rămâne doar cu datele personale și o cheie externă (ID_DEPARTAMENT) către tabelul de referință.

4. (0.5p) Să se dea un exemplu de tabel relațional în care există o dependență multivaloare (multidependență) între atributele (coloanele) sale, care nu este dependență funcțională.

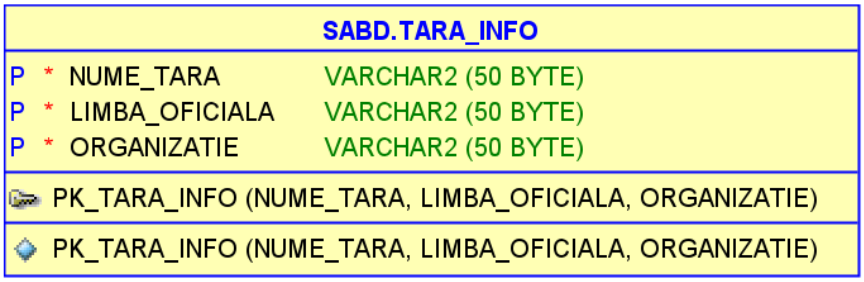


Figura 4.1. Tabelul TARA_INFO

În tabelul TARA_INFO există o dependență multivaloare (Nume_Tara - Limba_Oficiala și Nume_Tara-Organizatie) deoarece pentru o țară dată există mai multe limbi și mai multe organizații care nu au nicio legătură directă între ele. Aceasta nu este o dependență funcțională, deoarece numele țării nu determină o singură valoare unică pentru limbă sau organizație, ci seturi independente de valori. Dacă am dori să adăugăm o limbă nouă pentru o țară (de exemplu, engleza ca limbă de lucru), ar trebui să inserăm câte un rând nou pentru **fiecare** organizație din care țara face parte, generând astfel redundanță și riscul de a avea date inconsistente, situație specifică tabelelor care nu respectă Forma Normală 4.

	NUME_TARA	LIMBA_OFICIALA	ORGANIZATIE
1	Moldova	Romana	ONU
2	Moldova	Romana	UE (Candidat)
3	Romania	Romana	ONU
4	Romania	Romana	UE
5	Ucraina	Ucraineana	ONU
6	Ucraina	Ucraineana	UE (Candidat)

Figura 4.2. Valori pentru exemplu

5. (0.5p) Să se ilustreze printr-un exemplu structura unui index de tip arbore B* și modul în care o interogare SQL folosește acest index.

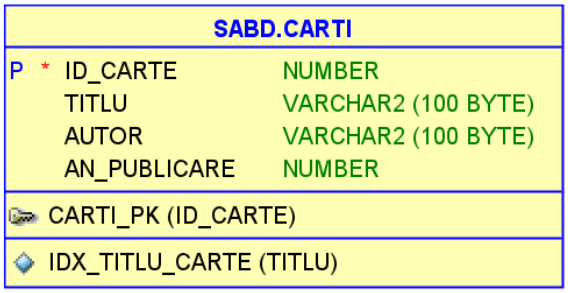


Figura 5.1.

Considerăm tabelul din figura 5.1, care are un index de tip Arbore B* creat pe coloana **TITLU**. Structura indexului este organizată ierarhic pe trei niveluri: Rădăcină, Noduri Intermediare și Noduri Frunză.

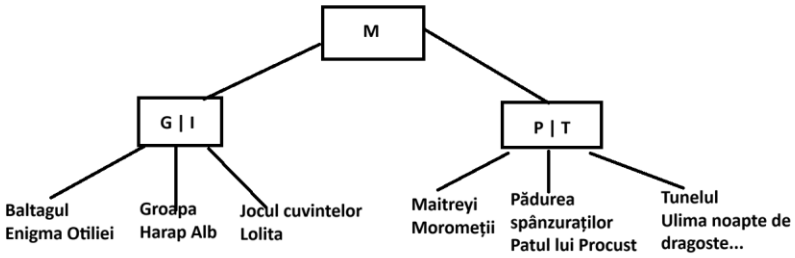


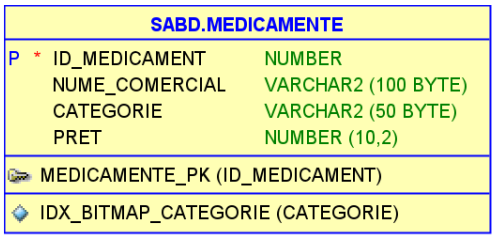
Figura 5.2.

SELECT * FROM SABD.CARTI WHERE TITLU = 'ION';

Pentru a găsi titlul **'ION'** prin intermediul indexului, procesul începe la nivelul rădăcină prin compararea primei litere a operei cu valoarea pivot **M**. Deoarece litera „I” se află alfabetic înaintea lui „M”, sistemul urmează ramura stângă către nodul intermediar ce conține reperele **G | I**. Aici, motorul bazei de date identifică intervalul corespunzător literei căutate și coboară spre nodurile frunză situate la baza ramificației din dreapta a acestui nod.

La nivelul frunză, sistemul verifică secvențial valorile stocate în blocul de date (unde se regăsesc opere precum **"Jocul cuvintelor"**) până la identificarea potrivirii exacte pentru **"ION"**. Odată localizat titlul în index, acesta furnizează adresa fizică a rândului din tabel, numită **ROWID**. Folosind acest pointer, interogarea SQL sare direct la locația de pe disc unde sunt salvate restul detaliilor despre carte, cum ar fi autorul sau anul publicării, realizând astfel căutarea fără a fi necesară scanarea integrală a tabelului **CARTI**.

6. (0.5p) Să se ilustreze printr-un exemplu structura unui index de tip bitmap și modul în care o interogare SQL folosește acest index.



	ID_MEDICAMENT	NUME_COMERCIAL	CATEGORIE	PRET
1	1	Paracetamol	Analgezic	15,5
2	2	Amoxicilină	Antibiotic	45
3	3	Aspirină	Analgezic	12
4	4	Vitamina C	Vitamină	25
5	5	Ibuprofen	Analgezic	22

Figura 6.1.

Mulțimea valorilor pe care le poate lua un index bitmap este finită, putând fi reprezentați în valori binare de tipul 10101. Această structură este optimă pentru coloane cu cardinalitate scăzută (unde valorile se repetă frecvent), deoarece căutarea devine mai rapidă, operațiile de filtrare fiind convertite în operații logice la nivel de biți, care sunt mult mai eficiente pentru procesor.

Analgezic	Antibiotic	Vitamină
1	0	0
0	1	0
1	0	0
0	0	1
1	0	0

Figura 6.2.

Astfel, în loc să parcurgă fiecare rând al tabelului MEDICAMENTE, motorul bazei de date procesează vectorul binar asociat categoriei căutate.

7. (0.5p) Să se dea un exemplu de vedere (vizualizare) VIEW_EX astfel încât comanda INSERT INTO VIEW_EX VALUES să producă o eroare. Să se explice cauza erorii.

```
269 CREATE TABLE STOC_MAGAZIN (
270     ID_PRODUS NUMBER PRIMARY KEY,
271     NUME_PRODUS VARCHAR2(100),
272     CATEGORIE VARCHAR2(50),
273     CANTITATE NUMBER,
274     PRET_UNITAR NUMBER
275 );
276
277 INSERT INTO STOC_MAGAZIN VALUES (1, 'Monitor', 'Electronice', 10, 800);
278 INSERT INTO STOC_MAGAZIN VALUES (2, 'Mouse', 'Electronice', 50, 50);
279 INSERT INTO STOC_MAGAZIN VALUES (3, 'Scaun Birou', 'Mobilier', 5, 450);
280
281 CREATE OR REPLACE VIEW VIEW_EX AS
282 SELECT CATEGORIE, SUM(CANTITATE * PRET_UNITAR) AS VALOARE_TOTALA
283 FROM SABD.STOC_MAGAZIN
284 GROUP BY CATEGORIE;
285
286 INSERT INTO VIEW_EX (CATEGORIE, VALOARE_TOTALA)
287 VALUES ('Electrocasnice', 5000);
```

Figura 7.1.

```
View VIEW_EX created.


Error starting at line : 286 in command -
INSERT INTO VIEW_EX (CATEGORIE, VALOARE_TOTALA)
VALUES ('Electrocasnice', 5000)
Error at Command Line : 286 Column : 33
Error report -
SQL Error: ORA-01733: виртуальный столбец здесь недопустим

https://docs.oracle.com/error-help/db/ora-01733/01733. 00000 -  "virtual column not allowed here"
*Cause:    An attempt was made to use an INSERT, UPDATE, or DELETE
           statement on an expression in a view.
*Action:   INSERT, UPDATE, or DELETE data in the base tables,
           instead of the view.

More Details :
https://docs.oracle.com/error-help/db/ora-01733/
```

Figura 7.2. Eroarea apărută

Eroarea apare deoarece vederea **VIEW_EX** este o vedere complexă, care nu are o corespondență directă de tip 1-to-1 cu rândurile din tabelul de bază **STOC_MAGAZIN**. Folosirea clauzei **GROUP BY** împreună cu funcția de agregare **SUM()** face ca această vedere să fie doar pentru citire, deoarece baza de date nu poate determina cum să transforme o valoare agregată, precum **VALOARE_TOTALA**, în valorile individuale ale coloanelor **CANTITATE** și **PRET_UNITAR** necesare pentru inserarea unui rând nou în tabelul fizic.

Pentru ca operația **INSERT** să fie posibilă, vederea ar trebui să fie una simplă, fără funcții de agregare, fără join-uri sau coloane calculate, astfel încât fiecare valoare inserată să poată fi mapată clar și direct pe o coloană din tabelul sursă.

8. (0,5p) Să se ilustreze printr-un exemplu modul în care o vedere (vizualizare) poate fi folosită pentru a asigura securitatea într-o bază de date.

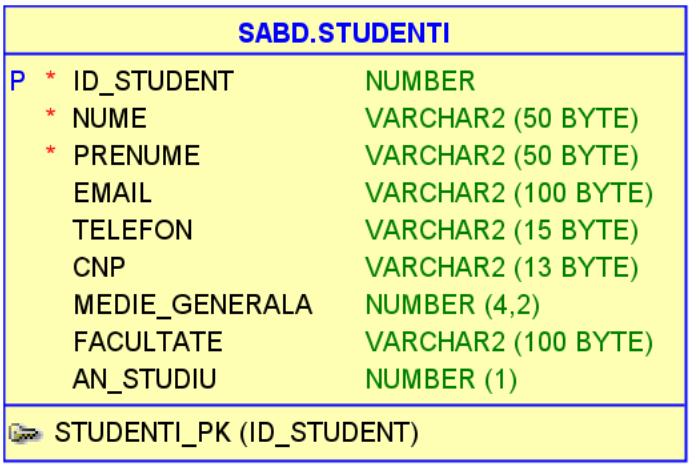


Figura 8.1. Tabelul inițial

```
INSERT INTO STUDENTI VALUES (1, 'Georgescu', 'Ana', 'ana.georgescu@student.ro', '0745123456', '2950315234567', 9.45, 'Informatica', 3);
INSERT INTO STUDENTI VALUES (2, 'Marinescu', 'Bogdan', 'bogdan.marinescu@student.ro', '0756234567', '1920520345678', 7.80, 'Matematica', 2);
INSERT INTO STUDENTI VALUES (3, 'Vasilescu', 'Elena', 'elena.vasilescu@student.ro', '0767345678', '2980710456789', 8.90, 'Informatica', 1);
COMMIT;

--VEDEREA pentru informatii publice
CREATE OR REPLACE VIEW STUDENTI_PUBLIC AS
SELECT
    id_student,
    nume,
    prenume,
    email,
    facultate,
    an_studiu
    --fara date confidentiale
FROM STUDENTI;

--Acordam privilegii pe VEDERE utilizatorilor externi
GRANT SELECT ON STUDENTI_PUBLIC TO utilizator_extern;

--Revocam accesul direct la tabelul STUDENTI
REVOKE SELECT ON STUDENTI FROM utilizator_extern;

SELECT * FROM STUDENTI_PUBLIC;
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0,051 seconds

	ID_STUDENT	NUME	PRENUME	EMAIL	FACULTATE	AN_STUDIU
1	1	Georgescu	Ana	ana.georgescu@student.ro	Informatica	3
2	2	Marinescu	Bogdan	bogdan.marinescu@student.ro	Matematica	2
3	3	Vasilescu	Elena	elena.vasilescu@student.ro	Informatica	1

Figura 8.2. Vizualizare cu un rezultat testat

9. (1p) Să se arate printr-un exemplu în ce condiții este posibil ca două select-uri identice consecutive (fără nici o altă comandă între ele), efectuate în aceeași sesiune de lucru, pe același tabel, pot produce rezultate diferite.

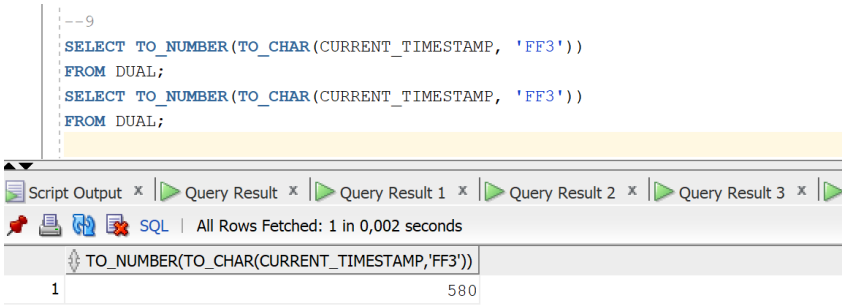


Figura 9.1.

CURRENT_TIMESTAMP este o funcție nedeterministă, cele 2 SELECT-uri sunt evaluate în momente diferite.

10. (0,5p) Să se dea un exemplu care să ilustreze interblocarea.

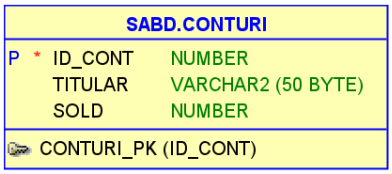


Figura 10.1. Tabelul pentru Conturi

```
--Tranzactia 1
UPDATE CONTURI
SET sold = sold - 100
WHERE id_cont = 1;

UPDATE CONTURI
SET sold = sold + 100
WHERE id_cont = 2;

--Tranzactia 2
UPDATE CONTURI
SET sold = sold - 50
WHERE id_cont = 2;

UPDATE CONTURI
SET sold = sold + 50
WHERE id_cont = 1;
```

Figura 10.2.

11. (0.5p) Să se ilustreze printr-un exemplu utilizarea unui trigger pentru a realiza o constrangere de integritate care nu ar putea fi implementată folosind un CONSTRAINT din definiția unui tabel.

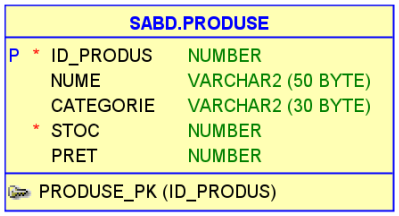
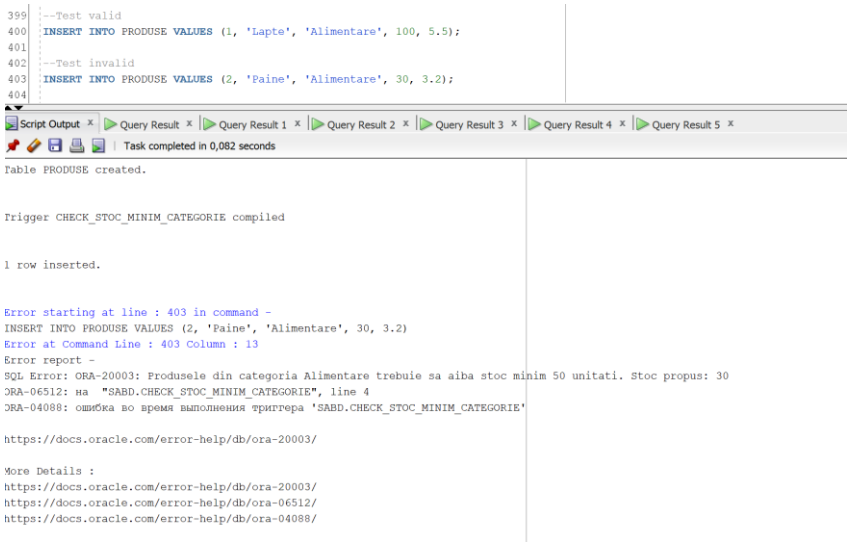


Figura 11.1.

```
CREATE OR REPLACE TRIGGER CHECK_STOC_MINIM_CATEGORIE
BEFORE INSERT OR UPDATE ON PRODUSE
FOR EACH ROW
BEGIN
    -- Produsele alimentare trebuie să aiba stoc minimum 50
    IF :NEW.categorie = 'Alimentare' AND :NEW.stoc < 50 THEN
        RAISE_APPLICATION_ERROR(-20003,
            'Produsele din categoria Alimentare trebuie sa aiba stoc minim 50 unitati. ' ||
            'Stoc propus: ' || :NEW.stoc);
    END IF;

    -- Produsele electronice trebuie să aiba stoc minimum 10
    IF :NEW.categorie = 'Electronice' AND :NEW.stoc < 10 THEN
        RAISE_APPLICATION_ERROR(-20004,
            'Produsele din categoria Electronice trebuie sa aiba stoc minim 10 unitati. ' ||
            'Stoc propus: ' || :NEW.stoc);
    END IF;
END;
```

Figura 11.2. Creare trigger pentru a verifica dacă este îndeplinit stocul minim necesar.



```
--Test valid
INSERT INTO PRODUSE VALUES (1, 'Lapte', 'Alimentare', 100, 5.5);

--Test invalid
INSERT INTO PRODUSE VALUES (2, 'Paine', 'Alimentare', 30, 3.2);
```

Script Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4 | Query Result 5

Task completed in 0,082 seconds

```
Table PRODUSE created.


Trigger CHECK_STOC_MINIM_CATEGORIE compiled


1 row inserted.


Error starting at line : 403 in command -
INSERT INTO PRODUSE VALUES (2, 'Paine', 'Alimentare', 30, 3.2)
Error at Command Line : 403 Column : 13
Error report -
SQL Error: ORA-20003: Produsele din categoria Alimentare trebuie sa aiba stoc minim 50 unitati. Stoc propus: 30
ORA-06512: на  "SABD.CHECK_STOC_MINIM_CATEGORIE", line 4
ORA-04088: ошибка во время выполнения триггера 'SABD.CHECK_STOC_MINIM_CATEGORIE'

https://docs.oracle.com/error-help/db/ora-20003/

More Details :
https://docs.oracle.com/error-help/db/ora-20003/
https://docs.oracle.com/error-help/db/ora-06512/
https://docs.oracle.com/error-help/db/ora-04088/
```

Figura 11.3. Verificare funcționalitate

12. (1p) Să se ilustreze printr-un exemplu de program PL/SQL multi-bloc modul de propagare a excepțiilor. Vor fi ilustrate cel putin situațiile în care o excepție este tratată sau nu în blocul curent și în care controlul programului va fi transmis blocului următor din secvență sau blocului exterior.

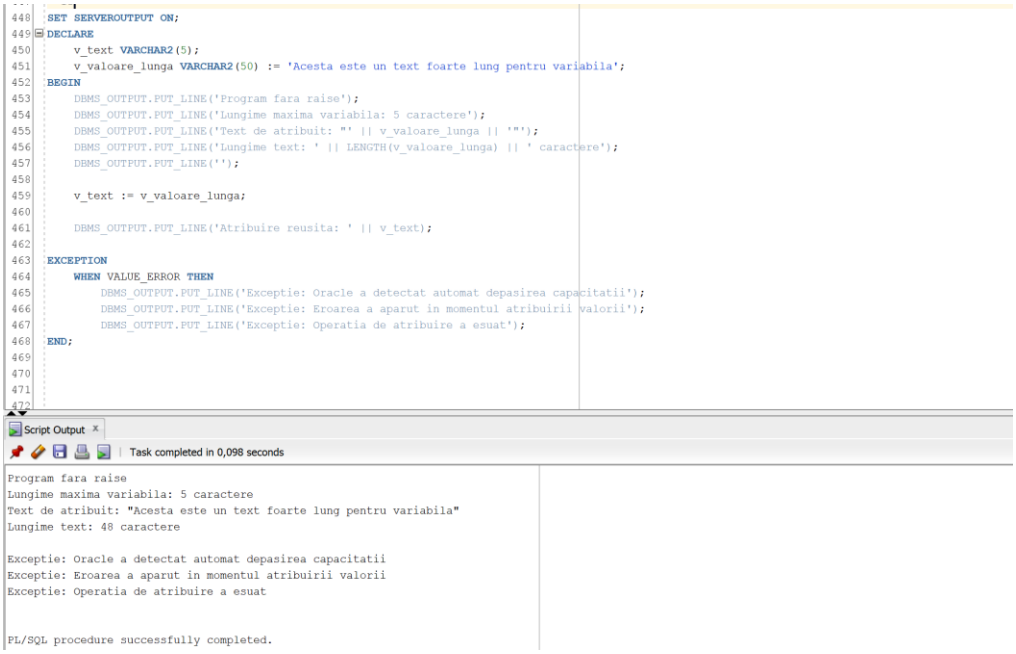
```
SET SERVEROUTPUT ON;
BEGIN
    BEGIN
        DBMS_OUTPUT.PUT_LINE(' Start: Bloc principal');
        BEGIN
            DBMS_OUTPUT.PUT_LINE('Start: Bloc imbricat A');

            RAISE_APPLICATION_ERROR(-20100, 'Exceptie generata in Blocul A');

            DBMS_OUTPUT.PUT_LINE('Sfarsit: Bloc imbricat A');
        EXCEPTION
            WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('Bloc A: Exceptie interceptata si rezolvata');
                DBMS_OUTPUT.PUT_LINE('Cod eroare: ' || SQLCODE);
        END;
        DBMS_OUTPUT.PUT_LINE('');

        BEGIN
            DBMS_OUTPUT.PUT_LINE('Start: Bloc imbricat B');

            RAISE_APPLICATION_ERROR(-20200, 'Exceptie generata in Blocul B');

            DBMS_OUTPUT.PUT_LINE('Sfarsit: Bloc imbricat B');

        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('Bloc B: NO_DATA_FOUND capturata');
        END;
        DBMS_OUTPUT.PUT_LINE('Sfarsit: Bloc principal');

    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Bloc principal: Exceptie primita de la bloc imbricat');
            DBMS_OUTPUT.PUT_LINE('Mesaj: ' || SQLERRM);
    END;
    DBMS_OUTPUT.PUT_LINE('');
    DBMS_OUTPUT.PUT_LINE('Executie finalizata cu succes');
END;
```

Figura 12.1. Cod

```
Start: Bloc principal
Start: Bloc imbricat A
Bloc A: Exceptie interceptata si rezolvata
Cod eroare: -20100

Start: Bloc imbricat B
Bloc principal: Exceptie primita de la bloc imbricat
Mesaj: ORA-20200: Exceptie generata in Blocul B

Executie finalizata cu succes


PL/SQL procedure successfully completed.
```

Figura 12.2. Output

13. (1p) Să se ilustreze prin exemple folosirea instrucțiunii RAISE pentru a ridica atât o excepție predefinită cât și o excepție definită de utilizator. În cazul excepțiilor predefinite, să se explice cum anume folosirea instrucțiunii RAISE schimbă funcționalitatea programului (față de cazul când această instrucțiune nu există).



```
SET SERVEROUTPUT ON;
DECLARE
    v_text VARCHAR2(5);
    v_valoare_lunga VARCHAR2(50) := 'Acesta este un text foarte lung pentru variabila';
BEGIN
    DBMS_OUTPUT.PUT_LINE('Program fara raise');
    DBMS_OUTPUT.PUT_LINE('Lungime maxima variabila: 5 caractere');
    DBMS_OUTPUT.PUT_LINE('Text de atribuit: "' || v_valoare_lunga || '"');
    DBMS_OUTPUT.PUT_LINE('Lungime text: ' || LENGTH(v_valoare_lunga) || ' caractere');
    DBMS_OUTPUT.PUT_LINE('');

    v_text := v_valoare_lunga;

    DBMS_OUTPUT.PUT_LINE('Atribuire reusita: ' || v_text);

EXCEPTION
    WHEN VALUE_ERROR THEN
        DBMS_OUTPUT.PUT_LINE('Exceptie: Oracle a detectat automat depasirea capacitatii');
        DBMS_OUTPUT.PUT_LINE('Exceptie: Eroarea a aparut in momentul atribuirii valorii');
        DBMS_OUTPUT.PUT_LINE('Exceptie: Operatia de atribuire a esuat');
END;
```

Script Output | Task completed in 0,098 seconds

```
Program fara raise
Lungime maxima variabila: 5 caractere
Text de atribuit: "Acesta este un text foarte lung pentru variabila"
Lungime text: 48 caractere

Exceptie: Oracle a detectat automat depasirea capacitatii
Exceptie: Eroarea a aparut in momentul atribuirii valorii
Exceptie: Operatia de atribuire a esuat


PL/SQL procedure successfully completed.
```

Figura 13.1. Program fără RAISE

```
SET SERVEROUTPUT ON;
DECLARE
    v_text VARCHAR2(5);
    v_valoare_lunga VARCHAR2(50) := 'Acesta este un text foarte lung pentru variabila';
BEGIN
    DBMS_OUTPUT.PUT_LINE('Program CU RAISE');
    DBMS_OUTPUT.PUT_LINE('Lungime maxima variabila: 5 caractere');
    DBMS_OUTPUT.PUT_LINE('Text de atribuit: "' || v_valoare_lunga || '"');
    DBMS_OUTPUT.PUT_LINE('Lungime text: ' || LENGTH(v_valoare_lunga) || ' caractere');
    DBMS_OUTPUT.PUT_LINE('');
    DBMS_OUTPUT.PUT_LINE('Se verifica lungimea textului');
    IF LENGTH(v_valoare_lunga) > 5 THEN
        DBMS_OUTPUT.PUT_LINE('DETECTAT: Text prea lung!');
        DBMS_OUTPUT.PUT_LINE('Lungime maxima: 5 caractere');
        DBMS_OUTPUT.PUT_LINE('Lungime curenta: ' || LENGTH(v_valoare_lunga) || ' caractere');
        DBMS_OUTPUT.PUT_LINE('Se ridica exceptia inainte de atribuire');
        RAISE VALUE_ERROR;
    END IF;
    v_text := v_valoare_lunga;
    DBMS_OUTPUT.PUT_LINE('Atribuire reusita: ' || v_text);
EXCEPTION
    WHEN VALUE_ERROR THEN
        DBMS_OUTPUT.PUT_LINE('');
        DBMS_OUTPUT.PUT_LINE('Exceptia a fost ridicata preventiv de programator');
        DBMS_OUTPUT.PUT_LINE('Atribuirea invalida a fost evitata complet');
        DBMS_OUTPUT.PUT_LINE('Variabila v_text ramane nemodificata');
END;
```

Figura 13.2. Program cu RAISE

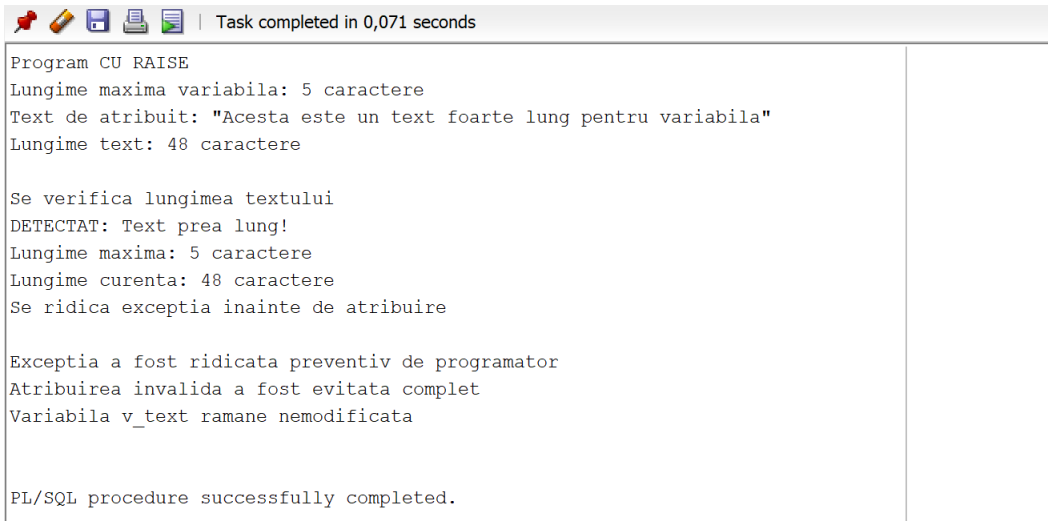


Figura 13.2. Output cu RAISE

Cu RAISE, putem verifica și preveni erorile inainte ca Oracle să încerce operația invalidă, oferind mesaje clare utilizatorului și menținând integritatea datelor.

```
SET SERVEROUTPUT ON;
DECLARE
    ex_email_invalid EXCEPTION;
    v_email VARCHAR2(100) := 'ion.popescu@gmail';  --nu are extensie
    v_nume_utilizator VARCHAR2(50) := 'Popescu Ion';
    v_tip_cont VARCHAR2(20) := 'Premium';
BEGIN
    DBMS_OUTPUT.PUT_LINE('Exceptie definita de utilizator');
    DBMS_OUTPUT.PUT_LINE('Utilizator: ' || v_nume_utilizator);
    DBMS_OUTPUT.PUT_LINE('Tip cont: ' || v_tip_cont);
    DBMS_OUTPUT.PUT_LINE('Email introdus: ' || v_email);
    DBMS_OUTPUT.PUT_LINE('');
    DBMS_OUTPUT.PUT_LINE('Se verifica formatul email-ului');
    IF INSTR(v_email, '@') = 0 THEN
        DBMS_OUTPUT.PUT_LINE('EROARE: Email-ul nu contine caracterul @');
        DBMS_OUTPUT.PUT_LINE('Format asteptat: utilizator@domeniu.extensie');
        RAISE ex_email_invalid;
    END IF;
    DBMS_OUTPUT.PUT_LINE('Verificare @ - OK');
    IF NOT REGEXP_LIKE(v_email, '\.[a-z]{2,}$') THEN
        DBMS_OUTPUT.PUT_LINE('EROARE: Email-ul nu contine extensie valida');
        DBMS_OUTPUT.PUT_LINE('Exemple extensii valide: .com, .ro, .net, .org');
        RAISE ex_email_invalid;
    END IF;
    IF LENGTH(v_email) < 6 THEN
        DBMS_OUTPUT.PUT_LINE('EROARE: Email-ul este prea scurt');
        DBMS_OUTPUT.PUT_LINE('Lungime minima: 6 caractere');
        RAISE ex_email_invalid;
    END IF;
    DBMS_OUTPUT.PUT_LINE('Toate verificarile au trecut cu succes!');
    DBMS_OUTPUT.PUT_LINE('');
    DBMS_OUTPUT.PUT_LINE('Cont creat pentru: ' || v_nume_utilizator);
    DBMS_OUTPUT.PUT_LINE('Email confirmat: ' || v_email);
    DBMS_OUTPUT.PUT_LINE('Tip cont: ' || v_tip_cont);
EXCEPTION
    WHEN ex_email_invalid THEN
        DBMS_OUTPUT.PUT_LINE('');
        DBMS_OUTPUT.PUT_LINE('Exceptie custom tratata');
        DBMS_OUTPUT.PUT_LINE('EROARE: Email invalid pentru utilizatorul ' || v_nume_utilizator);
        DBMS_OUTPUT.PUT_LINE('EMAIL INTRODUS: ' || v_email);
        DBMS_OUTPUT.PUT_LINE('CERINTA: Email-ul trebuie sa aiba formatul: nume@domeniu.extensie');
        DBMS_OUTPUT.PUT_LINE('ACTIUNE: Inregistrare respinsa - va rugam corectati email-ul');
END;
```

Figura 13.3. Excepție definita de utilizator

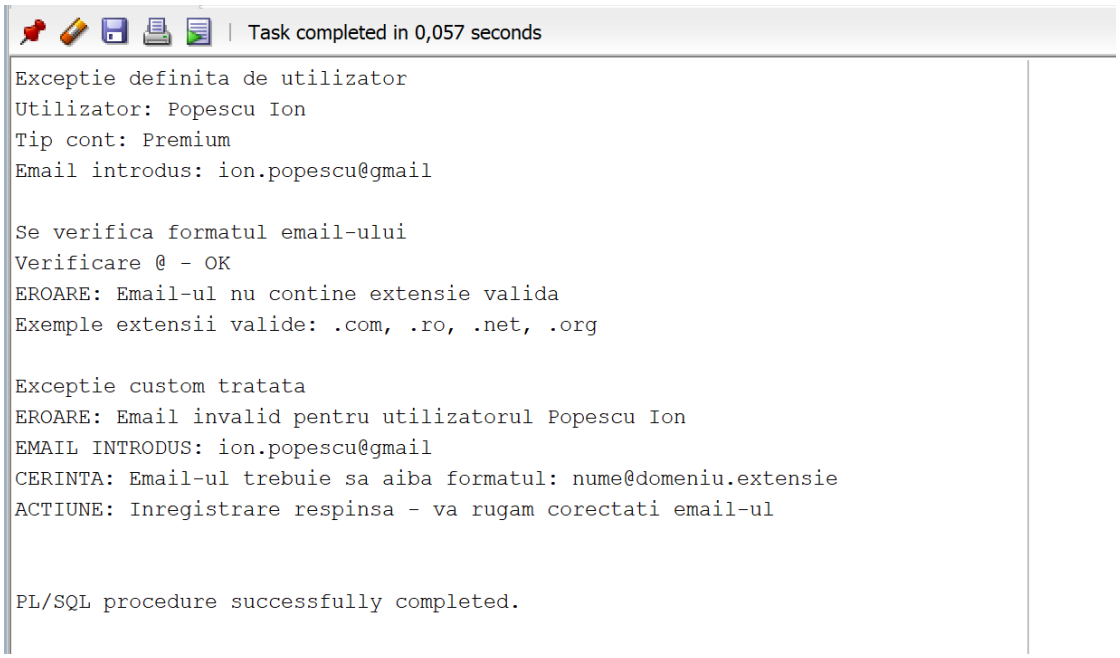


Figura 13.4. Output utilizator

14. (1p) Să se ilustreze printr-un exemplu cum anume folosirea colecțiilor și a instrucțiunilor BULK COLLECT și FORALL îmbunătățesc performanța în comparație cu folosirea cursoarelor. Să se explice diferența de performanță. Ce soluție există atunci când preluarea unei mulțime de rânduri prin BULK COLLECT consumă prea multă memorie și generează o eroare?

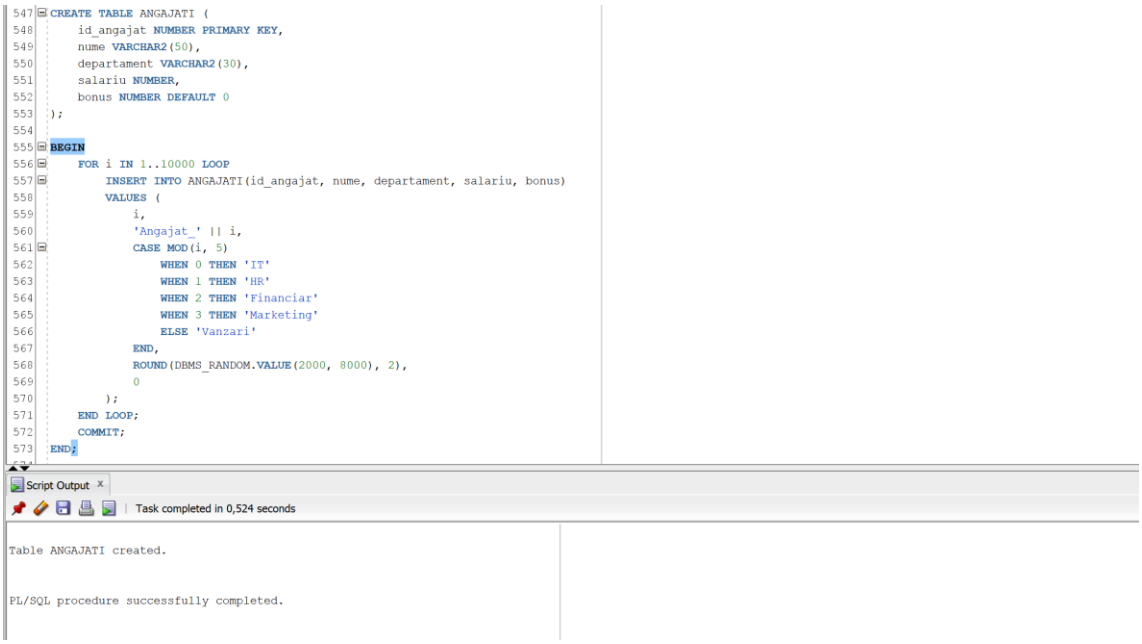


Figura 14.1. Crearea tabelului si popularea acestuia

```
SET SERVEROUTPUT ON;
SET TIMING ON;
DECLARE
    CURSOR c_angajati IS
        SELECT id_angajat, salariu FROM ANGAJATI;

    v_id_angajat ANGAJATI.id_angajat%TYPE;
    v_salariu ANGAJATI.salariu%TYPE;
    v_start_time TIMESTAMP;
    v_end_time TIMESTAMP;
    v_count NUMBER := 0;

BEGIN
    DBMS_OUTPUT.PUT_LINE('Cursor clasic');
    v_start_time := SYSTIMESTAMP;

    FOR rec IN c_angajati LOOP
        UPDATE angajati
        SET bonus = rec.salariu * 0.10
        WHERE id_angajat = rec.id_angajat;
        v_count := v_count + 1;
    END LOOP;
    COMMIT;
    v_end_time := SYSTIMESTAMP;

END;
```

Script Output | Task completed in 0,268 seconds

```
Cursor clasic


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.242
```

Figura 14.2. Cursor clasic

```
SET SERVEROUTPUT ON;
SET TIMING ON;
DECLARE
    TYPE t_id_angajati IS TABLE OF ANGAJATI.id_angajat%TYPE;
    TYPE t_salarii IS TABLE OF ANGAJATI.salariu%TYPE;
    v_id_angajati t_id_angajati;
    v_salarii t_salarii;
    v_bonusuri t_salarii;
    v_start_time TIMESTAMP;
    v_end_time TIMESTAMP;

BEGIN
    DBMS_OUTPUT.PUT_LINE('BULK COLLECT + FORALL');
    v_start_time := SYSTIMESTAMP;
    SELECT id_angajat, salariu
    BULK COLLECT INTO v_id_angajati, v_salarii
    FROM angajati;
    DBMS_OUTPUT.PUT_LINE('Randuri incarcate in memorie: ' || v_id_angajati.COUNT);

    v_bonusuri := t_salarii();
    v_bonusuri.EXTEND(v_salarii.COUNT);

    FOR i IN 1..v_salarii.COUNT LOOP
        v_bonusuri(i) := v_salarii(i) * 0.10;
    END LOOP;
    FORALL i IN v_id_angajati.FIRST .. v_id_angajati.LAST
        UPDATE angajati
        SET bonus = v_bonusuri(i)
        WHERE id_angajat = v_id_angajati(i);
    COMMIT;
    v_end_time := SYSTIMESTAMP;
END;
```

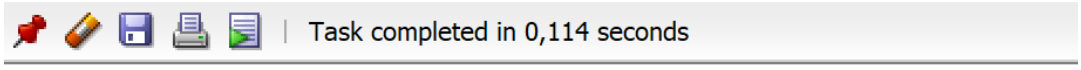
Script Output | Task completed in 0,122 seconds

```
BULK COLLECT + FORALL
Randuri incarcate in memorie: 10000


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.098
```

Figura 14.3. BULK COLLECT + FORALL

```plsql
SET SERVEROUTPUT ON;
SET TIMING ON;
DECLARE
    TYPE t_id_angajati IS TABLE OF ANGAJATI.id_angajat%TYPE;
    TYPE t_salarii IS TABLE OF ANGAJATI.salariu%TYPE;
    v_id_angajati t_id_angajati;
    v_salarii t_salarii;
    v_bonusuri t_salarii;
    CURSOR c_angajati IS
        SELECT id_angajat, salariu FROM ANGAJATI;

    v_start_time TIMESTAMP;
    v_end_time TIMESTAMP;
    v_batch_size CONSTANT NUMBER := 1000;
    v_total_processed NUMBER := 0;
    v_batch_count NUMBER := 0;

BEGIN
    DBMS_OUTPUT.PUT_LINE('BULK COLLECT cu LIMIT');
    v_start_time := SYSTIMESTAMP;
    OPEN c_angajati;
    LOOP
        FETCH c_angajati
        BULK COLLECT INTO v_id_angajati, v_salarii
        LIMIT v_batch_size;
        EXIT WHEN v_id_angajati.COUNT = 0;
        v_batch_count := v_batch_count + 1;
        v_bonusuri := t_salarii();
        v_bonusuri.EXTEND(v_salarii.COUNT);
        FOR i IN 1..v_salarii.COUNT LOOP
            v_bonusuri(i) := v_salarii(i) * 0.10;
        END LOOP;
        FORALL i IN v_id_angajati.FIRST .. v_id_angajati.LAST
            UPDATE angajati
            SET bonus = v_bonusuri(i)
            WHERE id_angajat = v_id_angajati(i);
        v_total_processed := v_total_processed + SQL%ROWCOUNT;
        COMMIT;
    END LOOP;
    CLOSE c_angajati;
    v_end_time := SYSTIMESTAMP;
END;
```

Figura 14.4. BULK COLLECT cu LIMIT




```
BULK COLLECT cu LIMIT


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.078
```

Figura 14.5. Output

BULK COLLECT și FORALL îmbunătățesc dramatic performanța prin reducerea context switches-urilor între PL/SQL și SQL, iar când BULK COLLECT consumă prea multă memorie, soluția este folosirea clauzei LIMIT pentru a procesa datele în batch-uri mici, menținând performanța ridicată și evitând erorile de memorie. Cursorul este lent pentru seturi mai mari de date.