**Proiect Sisteme de Gestiune**

**a Bazelor de Date**

**Baza de Date pentru**

**Gestiunea unei Universitati**

**Nume:** Kayed Amar

**Grupa:** 243

**Email:** [amar.kayed@s.unibuc.ro](mailto:amar.kayed@s.unibuc.ro)

Anul Universitar 2021 - 2022

**1.**

Exemplele din cadrul acestui proiect au ca scop proiectarea si prezentarea unui model de date ce furnizeaza informatii despre gestiunea studentilor in cadrul unei universitati, concept important in cadrul academic.

Vom prezenta modelul de date, restrictiile pe care trebuie sa le respecte si vom incerca, intr-o maniera didactica, sa construim diagrama E/R corespunzatoare, precum si diagrama conceptuala.

Modelul de date va getiona informatii legate de organizarea si functionarea diferitelor componente ale unei universitati. O universitate reprezinta un cadru academic ce contine mai multe facultati, care la randul lor contin studenti si angajati. Acest model de date este valabil pentru orice tip de universitate.

O universitate este alcatuita din mai multe facultati, fiecare facultate avand mai multe specializari. Pe langa specializari, o facultate are studenti si angajati. Studentii studiaza/invata disciplinele oferite de o specializare.

O specializare este unica in cadrul unei facultati, insa contine discipline care pot fi comune intre mai multe specializari din cadrul **aceleasi** facultati. Exista facultati diferite cu specializari sau discipline asemanatoare, dar nu identice, spre exemplu: atat FMI cat si Automatica predau materia “Baze de Date”, insa aceasta materie se diferentiaza prin cel putin specializarea si profesorul care o preda.

Angajatii lucreaza in cadrul unei facultati, acestia sunt de mai multe tipuri:

* Secretare, caz in care se ocupa cu administrarea facultatii, tot ce tine de partea organizatorica si birocratica a unei facultati.
* Profesori, caz in care se ocupa cu predarea disciplinelor si evaluarea studentilor

Un student poate urmari mai multe specializari diferite, insa nu in cadrul aceleasi facultati. Studentul poate urmari mai multe facultati diferite si in cadrul fiecarei facultati o singura specializare.

Pentru a studia in cadrul unei facultati, un student poate fi cazat la un camin. Indiferent de numarul de facultati la care studentul este inscris, acesta poate ocupa maxim un singur loc la camin(adica se poate caza doar la un singur camin).

La o disciplina participa atat profesori cat si studenti. O disciplina are implicit si o forma de examinare stabilita de profesor si sustinuta de student. Forma de examinare a unei discipline poate fi de mai multe tipuri, insa in diagrama E/R am ales doar sa specificam tipul prin intermediul unui atribut, si nu prin intermediul unor subentitati, intrucat in acest context nu vom pune foarte mult accent pe diferitele tipuri de examinare. Formele de examinare sunt: examen scris, proiect, ascultare orala. Pentru toate formele de examinare trebuie sa se cunoasca nota studentului si data sustinerii.

Facultatile unei universitati ocupa implicit o locatie, aceasta locatie la randul ei apartinand unei tari.

Modelul de date este util, intrucat proiecteaza o structura ordonata in care se pot tine evidenta studentilor inscrisi la diverse facultati, precum si informatiile despre respectivele facultati si ale angajatilor din cadrul acestor facultati.

**Prezentarea constrângerilor (restricții, reguli) impuse asupra modelului.**

In acest context, pentru simplificarea notiunilor, prin specializare ne referim la un program de studiu distinct oferit de o anumita facultate si care poate fi urmarita de studenti ai facultatii respective.

Modelul de date respectă anumite restricţii de funcţionare.

* Facultatea ofera mai multe programe de studii(adica specializari) din care un student poate sa aleaga doar una singura. Studentii pot fi inscrisi la mai multe facultati, insa in cadrul unei facultati pot urma o singura specializare.
* In acest context, prin angajat ne referim la un salariat strict al facultatii(si prin consecinta a universitatii) si nu a oricarei alte organizatii. In acest sens, angajatii pot fi de mai multe tipuri, fiecare tip ocupand un rol esential in buna functionare a facultatii(Ex: profesor, secretara, etc).
* In cazul in care angajatul reprezinta un profesor, acesta poate preda mai multe discipline, disciplinele respective fiind urmarite de studentii inscrisi la facultate.
* Nu exista angajati care sa fie minori(care sa nu aibe varsta de cel putin 18 ani).
* Facultatea are implicit o adresa, care se afla intr-o locatie. Aceasta locatie automat se afla intr-o tara si intr-o regiune.
* Pentru modelul de date s-a considerat faptul ca toate facultatile apartin aceleasi universitati, de aici motivandu-se absenta unei entitati pentru universitati.
* Cu toate ca toate facultatile apartin aceleasi universitati, acestea pot prezenta specializari/discipline asemanatoare iar un student poate fi inscris la mai multe facultati.
* Pentru fiecare disciplina s-a considerat obligatoriu o forma de examinare.
* Toti studentii inscrisi la o facultate trebuie sa urmeze obligatoriu o specializare din cadrul facultatii respective.
* Pentru orice disciplina este obligatoriu ca numarul de ore predate sa fie de cel putin 1, adica sa nu existe discipline care nu sunt predate/parcurse.

**Descrierea entităților, incluzând precizarea cheii primare.**

Pentru modelul de date referitor la gestiunea unei universitati, structurile ANGAJAT, SECRETARA, PROFESOR, DISCIPLINA, EXAMINARE, STUDENT, CAMIN, SPECIALIZARE, FACULTATE, LOCATIE, TARA reprezinta entitati.

Vom prezenta entităţile modelului de date, dând o descriere completă a fiecăreia. De asemenea, pentru fiecare entitate se va preciza cheia primară.

Toate entitatile prezentate sunt independente, cu exceptia entitatii EXAMINARE, aceasta depinzand de entitatea DISCIPLINA(nu poate exista examinare daca nu exista disciplina/materie pentru acest examen).

1. ANGAJAT = persoana fizica/salariat in cadrul unei facultati si, implicit, in cadrul unei universitati. Angajatul se ocupa cu organizarea si gestionarea diferitelor componente care intra buna functionare a unei facultati. Aceasta entitate reprezinta o superclasa pentru entitatile *PROFESOR* si *SECRETARA*, intrucat un angajat in cadrul unei facultati poate fi un profesor sau o secretara, sau niciuna. Cheia primara a acestei superentitati este *cod\_angajat*.
2. SECRETARA = subentitate a entitatii ANGAJAT, aceasta reprezinta reprezinta o persoana fizica care se ocupa cu administrarea unei facultati, cu organizarea si intretinerea activitatilor academice precum si a componentelor birocratice/juridice. Fiind o subentitate, aceasta mosteneste cheia primara a superclasei ANGAJAT, anume atributul *cod\_angajat*.
3. PROFESOR = subentitate a entitatii ANGAJAT, aceasta reprezinta un angajat care se ocupa cu predarea disciplinelor din cadrul unei specializari a facultatii, precum si de evaluarea studentilor inscrisi la facultatea respectiva. Fiind o subentitate, aceasta mosteneste cheia primara a superclasei ANGAJAT, anume atributul *cod\_angajat*.
4. DISCIPLINA = concept academic in cadrul unei facultati, aceasta reprezinta un set de competente menite a fi predate studentilor, urmand ca acestia sa asimilize cunostiinta si sa fie evaluati din respectivele competente. Cheia primara a entitatii este *cod\_disciplina*.
5. EXAMINARE = activitate de evaluare a competentelor dobandite de student in cadrul unei discipline. Aceasta entitate este dependenta de enitatea DISCIPLINA, care contine informatii despre materia pentru care se face examinarea. Cheia primara a entitatii este compusa din *cod\_examinare* si *cod\_disciplina*.
6. STUDENT = persoana fizica, inscrisa in cadrul unei facultati si care urmeaza o specializare a facultatii respective. Aceasta entitate intra in relatie cu disciplinele predate in cadrul facultatii/specializarii, precum si cu profesorul ce preda respectiva disciplina. Cheia primara a acestei entitati este *cod\_student*(initial voiam sa pun numar\_matricol, intrucat acest atribut este unic in cadrul unei universitati, sau cnp).
7. CAMIN = persoana juridica, responsabila cu gazduirea studentilor din cadrul unei facultati. Cheia primara a entitatii este *cod\_camin*.
8. SPECIALIZARE = entitate care defineste programul de studiu din cadrul unei facultati. Aceasta entitate contine informatii legate de domeniul curricular al unei specializari din cadrul unei facultati. Cheia primara a acestei entitati este *cod\_specializare.*
9. FACULTATE = persoana juridica, reprezentand o entitate institutionala responsabila pentru facilitarea unui cadru academic formal. Aceasta entitate contine informatii legate de datele unei facultati si intra in relatii angajati si studenti. Cheia primara a acestei entitati este *cod\_facultate.*
10. LOCATIE = adresa fizica, aceasta entitate contine informatii legate de locatia fizica a unei facultati. Cheia primara a acestei entitati este *cod\_locatie.*
11. TARA = teritoriu geografic care inglobeaza mai multe locatii. Cheia primara a acestei entitati este *cod\_tara.*

**Descrierea relațiilor, incluzând precizarea cardinalității acestora.**

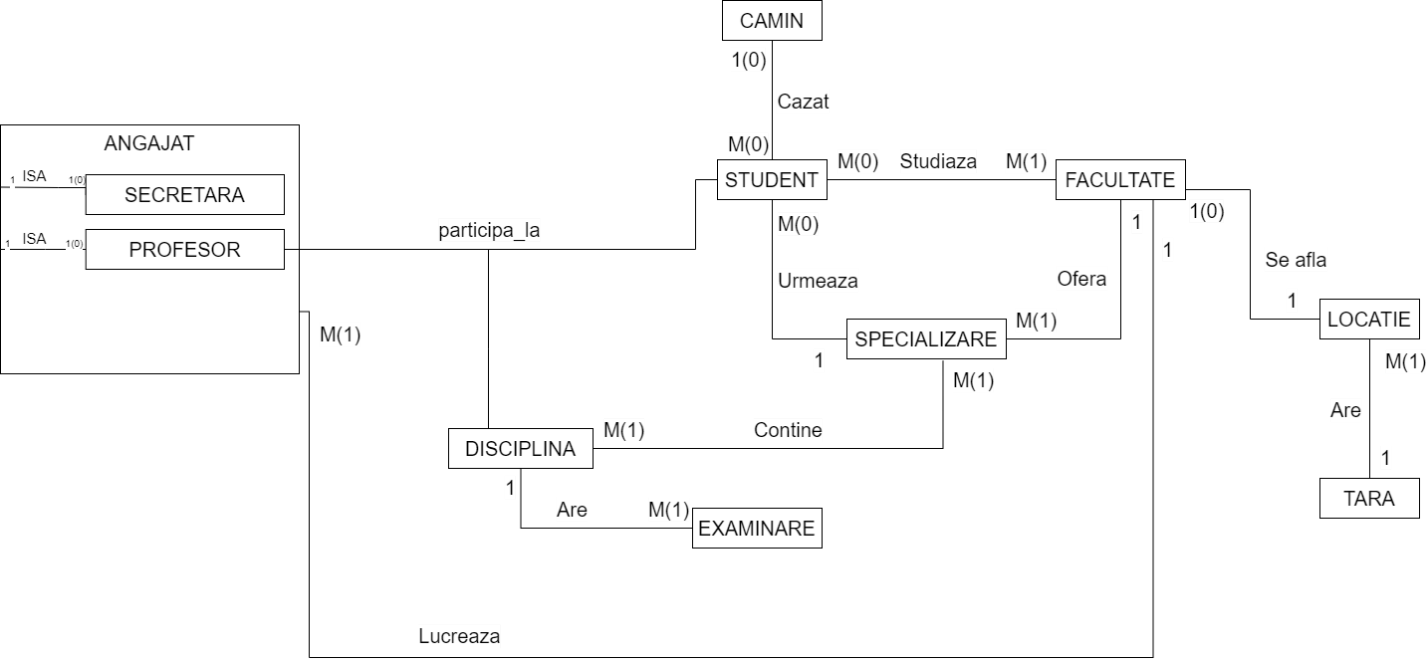
Vom prezenta relaţiile modelului de date, dând o descriere completă a fiecăreia. De fapt, denumirile acestor legături sunt sugestive, reflectând conţinutul acestora şi entităţile pe care le leagă. Pentru fiecare relaţie se va preciza cardinalitatea minimă şi maximă.

* ANGAJAT ISA SECRETARA = relatie de mostenire care leaga superclasa ANGAJAT si subclasa SECRETARA, reflectand legatura dintre acestea(o secretara este un angajat). Ea are cardinalitatea **minima 1:0**(one-to-zero), intrucat o secretara **trebuie sa fie si este** un angajat prin definitie, insa un angajat **nu trebuie** sa fie neaparat o secretara(poate fi si profesor sau orice altceva). De asemenea, relatia are cardinalitatea **maxima 1:1**(one-to-one), intrucat o secretara **poate fi maxim** un singur angajat(nu poti fi o singura secretara si sa ocupi pozitiile a mai multor angajati) si un angajat **poate fi maxim** o singura secretara(nu mai multe concomitent).
* ANGAJAT ISA PROFESOR = relatie de mostenire care leaga superclasa ANGAJAT si subclasa PROFESOR, reflectand legatura dintre acestea(un profesor este un angajat). Ea are cardinalitatea **minima 1:0**(one-to-zero), intrucat un profesor **trebuie sa fie si este** un angajat prin definitie, insa un angajat **nu trebuie** sa fie neaparat un profesor(poate fi si secretara sau orice altceva). De asemenea, relatia are cardinalitatea **maxima 1:1**(one-to-one), intrucat un profesor **poate fi maxim** un singur angajat(nu poti fi un singur profesor si sa ocupi pozitiile a mai multor angajati) si un angajat **poate fi maxim** un singur profesor(nu mai multi concomitent).
* ANGAJAT lucreaza FACULTATE = relatie care leaga superentitatea ANGAJAT si entitatea FACULTATE, reflectand legatura dintre acestea(la ce facultate lucreaza un angajat). Ea are cardinalitatea **minima 1:1**(one-to-one), intrucat un angajat **trebuie** sa lucreze la minim o facultate(altfel nu ar mai fi angajat in contextul universitatii) si o facultate **trebuie** sa contina minim un angajat(altfel s-ar desfiinta facultatea). De asemenea, aceasta relatie are cardinalitatea **maxima 1:n**(one-to-many), intrucat o facultate **poate** avea mai multi angajati in acelasi timp(mai multi profesori si mai multe secretare), insa un angajat **poate** lucra doar la o singura facultate si nu mai multe.
* STUDENT cazat CAMIN = relatie care leaga entitatile STUDENT si CAMIN, reflectand legatura dintre acestea(la ce camin este cazat un student). Ea are cardinalitatea **minima 0:0**(zero-to-zero), intrucat un student **nu trebuie** sa fie cazat neaparat la un camin, iar un camin **nu trebuie** sa aibe neaparat studenti in el(poate sa fie si gol). De asemenea, relatia are cardinalitatea **maxima 1:n**(one-to-many), intrucat un camin **poate** gazdui mai multi studenti, insa un student **poate** fi cazat la maxim un singur camin.
* STUDENT studiaza FACULTATE = relatie care leaga entitatile STUDENT si FACULTATE, reflectand legatura dintre acestea(la ce facultate studiaza un student). Ea are cardinalitatea **minima 1:0**(one-to-zero), intrucat un student **trebuie** sa fie inscris si sa studieze la minim o facultate(altfel nu ar mai fi student), insa o facultate **nu trebuie** neaparat sa aibe vreun student. De asemenea, aceasta relatie are cardinalitatea **maxima n:n**(many-to-many), intrucat un student **poate** fi inscris si sa studieze la mai multe facultati(in viata reala de obicei doar 2 facultati dar optiunea de a urma 3 sau mai multe este posibila), in timp ce o facultate **poate** avea mai multi studenti. Aceasta relatie many-to-many va crea un nou tabel in diagrama conceptuala, tabel care va avea o cheie primara compusa din cheile primare ale celor doua tabele aflate in relatie.
* STUDENT urmeaza SPECIALIZARE = relatie care leaga entitatile STUDENT si SPECIALIZARE, reflectand legatura dintre acestea(ce specializare urmeaza un student). Ea are cardinalitatea **minima 1:0**(one-to-zero), intrucat in cadrul unei facultati un student **trebuie** sa urmeze minim o specializare, insa o specializare **nu trebuie** sa fie urmarita de minim un student. De asemenea, relatia are cardinalitatea **maxima 1:n**(one-to-many), intrucat o specializare **poate** fi urmarita de mai multi studenti in acelasi timp(exista mai multi studenti care urmeaza specializarea de informatica spre exemplu), insa un student **poate** urmari maxim o singura specializare in cadrul unei singure facultati, acesta putand studia in mai multe facultati si pentru fiecare dintre aceste facultati poate urmari o singura specializare.
* FACULTATE ofera SPECIALIZARE = relatie care leaga entitatile FACULTATE si SPECIALIZARE, reflectand legatura dintre acestea(ce specializari ofera o facultate). Ea are cardinalitatea **minima 1:1**(one-to-one), intrucat o facultate **trebuie** sa ofere minim o specializare(adica un program de studiu), iar o specializare **trebuie** sa fie oferita(adica sa apartina) unei singure facultati(nu exista specializari care sa nu fie in cadrul unei facultati). De asemenea, relatia are cardinalitatea **maxima 1:n**(one-to-many), intrucat o facultate **poate** oferi mai multe specializari(cum ar fi FMI-ul, oferind specializare de mate, info si CTI), iar o specializare **poate** fi oferita de o singura facultate(exista, intr-adevar, aceeasi denumire pentru o specializare cum ar fi CTI de la FMI si CTI de la automatica, insa acestea nu sunt exact aceeasi specializare, au materii comune dar si materii diferite, deci nu sunt identice, acestea fiind diferite prin cel putin cheia primara care este unica fiecarei entitati).
* FACULTATE se afla LOCATIE = relatie care leaga entitatile FACULTATE si SPECIALIZARE, reflectand legatura dintre acestea(in ce locatie se afla o facultate). Ea are cardinalitatea **minima 1:0**(one-to-zero), intrucat o facultate **trebuie** sa se afle intr-o locatie, in timp ce intr-o locatie **nu trebuie** sa se afle neaparat o facultate(nu exista orase in care toate cladirile sunt doar facultati). De asemenea, relatia are cardinalitatea **maxima 1:1**(one-to-one), intrucat o facultate se **poate** afla maxim intr-o singura locatie(in acest context facultatile nu au mai multe cladiri separate), in timp ce o locatie **poate** avea maxim o singura facultate.
* LOCATIE are TARA = relatie care leaga entitatile LOCATIE si TARA, reflectand legatura dintre acestea(in ce tara se afla o anumita locatie). Ea are cardinalitatea **minima 1:1**(one-to-one), intrucat o locatie **trebuie** sa se afle intr-o tara, iar o tara **trebuie** sa contina minim o locatie(altfel nu ar mai fi tara si ar fi doar un teritoriu pustiu nelocuit). De asemenea, relatia are cardinalitatea **maxima 1:n**(one-to-many), intrucat o tara **poate** avea mai multe locatii, in timp ce o locatie **poate** sa apartina maxim unei singure tari.
* SPECIALIZARE contine DISCIPLINA = relatie care leaga entitatile SPECIALIZARE si DISCIPLINA, reflectand legatura dintre acestea(ce discipline se studiaza in cadrul unei specializari al unei facultati). Ea are cardinalitatea **minima 1:1**(one-to-one), intrucat o specializare **trebuie** sa contina minimum o disciplina(altfel nu ar mai fi specializare), in timp ce o disciplina **trebuie** sa apartina minim unei specializari. De asemenea, relatia are cardinalitatea **maxima n:n**(many-to-many), intrucat o specializare **poate** sa aibe mai multe discipline, in timp ce aceeasi disciplina **poate** sa apartina mai multor specializari(cum ar fi exemplul de la FMI pentru materia “Baze de Date”, care apartine atat specializarii de informatica, cat si pentru studentii de la specializarea matematica-informatica). Aceasta relatie many-to-many va crea un nou tabel in diagrama conceptuala, tabel care va avea o cheie primara compusa din cheile primare ale celor doua tabele aflate in relatie.
* DISCIPLINA are EXAMINARE = relatie care leaga entitatile DISCIPLINA si EXAMINARE, reflectand legatura dintre acestea(ce forma de examinare are o anumita disciplina). Ea are cardinalitatea **minima 1:1**(one-to-one), intrucat o disciplina **trebuie** sa aibe o forma de examinare(fie ea examen, proiect sau ascultare), in timp ce o examinare **trebuie** sa apartina unei discipline. De asemenea, relatia are cardinalitatea **maxima 1:n**(one-to-many), intrucat o disciplina **poate** avea mai multe examinari(mai multe teste de laborator+proiecte, etc), in timp ce o examinare **poate** sa apartina maxim unei singure discipline(nu se poate da un singur examen pentru doua sau mai multe materii simultan)
* STUDENT participa\_la DISCIPLINA predata\_de PROFESOR = relatie de tip 3 ce leaga entitatile STUDENT, DISCIPLINA si PROFESOR, reflectand ce student participa la disciplina predata de un anumit profesor. Din moment ce este o relatie de tip 3, aceasta se va transforma intr-un tabel care va avea o cheie primara compusa din cele 3 chei primare ale tabelelor aflate in relatie, acest lucru se poate observa si in diagrama conceptuala, denumirea acestui nou tabel nou fiind “PARTICIPA”, intrucat profesorul preda disciplina la care **participa** un student. Datorita acestei relatii de tip 3, acum putem raspunde la intrebarea “la ce disciplina participa un student pentru un anumit profesor”, intrucat o relatie de tip 3 este diferita de 3 relatii de tip 2.

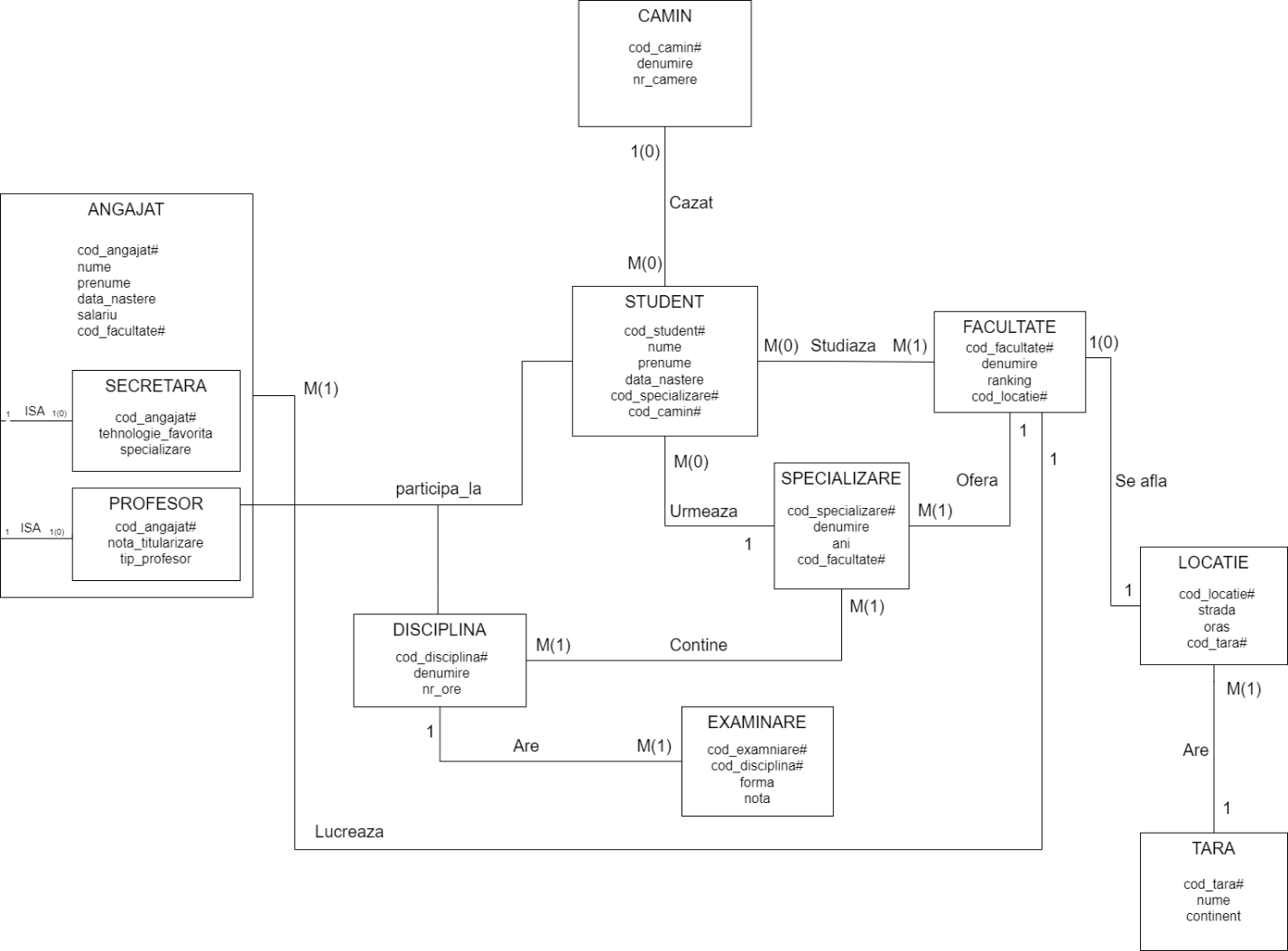
**Descrierea atributelor, incluzând tipul de date și eventualele constrângeri, valori implicite, valori posibile ale atributelor.**

1. Entitatea independenta ANGAJAT are ca atribute:
   * ***cod\_angajat*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul unui angajat, aceasta fiind o cheie primara. Prin urmare, acest atribut nu poate fi NULL. (Ex: 12, 1234, 51)
   * ***nume*** = variabila de tip caracter, de lungime maxima 25, care reprezinta numele angajatului. Acest atribut nu poate fi NULL. (Ex: ‘Popescu’, ‘Mihailescu’)
   * ***prenume*** = variabila de tip caracter, de lungime maxima 25, care reprezinta prenumele angajatului. Acest atribut nu poate fi NULL. (Ex: ‘Mihai’, ‘Andrei’)
   * ***data\_nastere*** = variabila de tip data calendaristica, care reprezinta data nasterii angajatului respectiv. Acest atribut va avea ca valoare implicita data curenta(folosind SYSDATE-ul din SQL). Acest atribut are constrangere de a avea valori posibile care sa fie cu minim 18 ani inainte de data curenta, adica nu exista angajati minori. Acest atribut poate fi si NULL. (Ex: ’17-JUN-87’)
   * ***salariu*** = variabila de tip real, cu 10 cifre in total, dintre care 2 sunt cifre zecimale, care reprezinta salariul lunar al unui angajat.
   * ***cod\_facultate*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul facultatii la care lucreaza angajatul. Atributul reprezinta o cheie externa si trebuie sa corespunda unei valori a cheii primare din tabelul FACULTATE. In acest context, un angajat trebuie sa lucreze la facultate, deci acest atribut nu poate fi null. (Ex: 120, 12340, 510)
2. Subentitatea SECRETARA mosteneste toate atributele superclasei ANGAJAT prin intermediul cheii primare comune dintre aceste doua tabele. Enitatea SECRETARA este dependenta de entitatea ANGAJAT. Prin urmare, subentitatea SECRETARA are atributele:
   * ***cod\_angajat*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul unui angajat, aceasta fiind o cheie primara in tabelul SECRETARA. Cheia primara din SECRETARA poate fi folosita pentru a accesa atributele corespunzatoare din tabelul ANGAJAT. Cheia primara din SECRETARA trebuie sa corespunda unei chei primare din ANGAJAT, ambele neputand fii NULL. (Ex: 12, 1234, 51)
   * ***tehnologie\_favorita*** = variabila de tip caracter, de lungime maxima 25, care reprezinta tehnologia favorita utilizata de secretara pentru a administra facultatea la care este angajata. Aceast atribut poate fi NULL. (Ex: Word, Powerpoint, Excel, etc)
   * ***specializare*** = variabila de tip caracter, de lungime maxima 25, care reprezinta specializarea secretarei. Aceast atribut poate fi NULL. (Ex: ‘Organizare’, ‘Administrare’, ‘Statistica’)
3. Subentitatea PROFESOR mosteneste toate atributele superclasei ANGAJAT, prin intermediul cheii primare comune dintre aceste doua tabele. Entitatea PROFESOR este dependenta de entitatea ANGAJAT. Prin urmare, subeintatea PROFESOR are atributele:
   * ***cod\_angajat*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul unui angajat, aceasta fiind o cheie primara in tabelul PROFESOR. Cheia primara din PROFESOR poate fi folosita pentru a accesa atributele corespunzatoare din tabelul ANGAJAT. Cheia primara din PROFESOR trebuie sa corespunda unei chei primare din ANGAJAT, ambele neputand fii NULL. (Ex: 12, 1234, 51)
   * ***nota\_titularizare*** = variabila de tip numar real, cu 4 cifre, dintre care 2 sunt zecimale, care reprezinta nota obtinuta de profesor in examenul de titularizare. Acest atribut poate fi NULL, intrucat in acest context pot exista profesori fara titularizare. De asemenea, nota maxima este 10.00 (Ex: 9.54, 6.21, 10.00)
   * ***tip\_profesor*** = variabila de tip caracter, de lungime maxima 15, care reprezinta tipul profesorului. Valorile posibile sunt: ‘Cursant’, ‘Seminarist‘, ‘Laborant’. Acest atribut poate fi NULL, caz in care se subintelege ca profesorul este invitat special pentru a tine o ora, iar acesta nu se incadreaza in categoriile anterior mentionate.
4. Entitea independenta DISCIPLINA are ca atribute:
   * ***cod\_disciplina*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul disciplinei, aceasta fiind o cheia primara pentru DISCIPLINA. Prin urmare, acest atribut nu poate fi NULL. (Ex: 10, 20, 30)
   * ***denumire*** = variabila de tip caracter, de lungima maxima 50, care reprezinta denumirea disciplinei. Acest atribut nu poate fi NULL. (Ex: ‘Baze de Date’, ‘Programare Web’)
   * ***nr\_ore*** = variabila de tip intreg, de lungime maxima 3, care reprezinta numarul total de ore predate pentru aceasta disciplina intr-un interval de 2 saptamani. Acest atribut nu poate fi NULL. (Ex: 3, caz in care se face un curs saptamanal si un alt curs odata la doua saptamani)
5. Entitatea dependenta EXAMINARE are ca atribute:
   * ***cod\_examinare*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul examinarii, aceasta facand parte din cheia primara compusa pentru EXAMINARE. Prin urmare, acest atribut nu poate fi NULL. (Ex: 10, 20, 30)
   * ***cod\_disciplina*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul disciplinei(refera cheia primara din DISCIPLINA), aceasta fiind a doua jumatate din cheia primara compusa pentru EXAMINARE. Prin urmare, acest atribut nu poate fi NULL. (Ex: 100, 200, 300). **Cheia primara a tabelului EXAMINARE este compusa din cod\_examinare si cod\_disciplina.**
   * ***forma*** = variabila de tip caracter, de lungime maxima 25, care reprezinta tipul/forma de examinare. Aceast atribut poate fi NULL. (Ex: ‘Examen Scris’, ‘Proiect’)
   * ***nota*** = variabila de tip numar real, de lungime maxima 4, dintre care 2 cifre sunt zecimale, care reprezinta nota obtinuta de student la examinare. Aceast atribut trebuie sa fie mai mic sau egal cu 10 si poate fi NULL.
6. Entitatea independenta STUDENT are ca atribute:
   * ***cod\_student*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul unic al unui student, acest atribut reprezinta o cheie primara si putea fi denumit si “nr\_matricol” sau “cnp”. Prin urmare, acest atribut nu poate fi NULL. (Ex: 10, 20, 30)
   * ***nume*** = variabila de tip caracter, de lungime maxima 25, care reprezinta numele studentului. Acest atribut nu poate fi NULL. (Ex: ‘Popescu’, ‘Mihailescu’)
   * ***prenume*** = variabila de tip caracter, de lungime maxima 25, care reprezinta prenumele studentului. Acest atribut nu poate fi NULL. (Ex: ‘Mihai’, ‘Andrei’)
   * ***data\_nastere*** = variabila de tip data calendaristica, care reprezinta data nasterii studentului respectiv. (Ex: ’17-JUN-87’)
   * ***cod\_specializare*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul specializarii pe care studentul o urmeaza. Acest atribut reprezinta o cheie externa care refera cheia primara din SPECIALIZARE, deci poate fi si NULL.
   * ***cod\_camin*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul caminului la care este cazat studentul, acest atribut reprezinta o cheie externa care refera cheia primara din tabelul CAMIN. Prin urmare, acest atribut poate fi NULL. (Ex: 10, 20, 30)
7. Entitatea independenta CAMIN are ca atribute:
   * ***cod\_camin*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul unic al unui camin, acest atribut reprezinta o cheie primara. Prin urmare, acest atribut nu poate fi NULL. (Ex: 10, 20, 30)
   * ***denumire*** = variabila de tip caracter, de lungime maxima 30, care reprezinta numele caminului. Acest atribut poate fi NULL. (Ex: ‘Grozavesti’, ‘Regie’)
   * ***nr\_camere*** = variabila de tip intreg, de lungime maxima 4, care reprezinta numarul total de camere pe care caminul il are. Acest atribut poate fi si NULL.(Ex: 1024, 10, 502)
8. Entitatea independenta SPECIALIZARE are ca atribute:
   * ***cod\_specializare*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul specializarii. Acest atribut reprezinta o cheie primara, deci nu poate fi NULL.
   * ***denumire*** = variabila de tip caracter, de lungime maxima 30, care reprezinta denumirea specializarii. Acest atribut nu poate fi NULL.
   * ***ani*** = variabila de tip intreg, de lungime maxima 1, care reprezinta numarul de ani necesari pentru a termina specializarea. Acest atribut nu poate fi NULL.
   * ***cod\_facultate*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul facultatii care prezinta aceasta specializare. Acest atribut reprezinta o cheie externa, deci poate fi NULL.
9. Entitatea independenta FACULTATE are ca atribute:
   * ***cod\_facultate*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul facultatii. Acest atribut reprezinta o cheie primara, deci nu poate fi NULL.
   * ***denumire*** = variabila de tip caracter, de lungime maxima 50, care reprezinta denumirea facultatii. Acest atribut nu poate fi NULL.
   * ***ranking*** = variabila de tip intreg, de lungime maxima 4, care reprezinta ranking-ul facultatii respective in top-ul tuturor facultatilor. Acest atribut poate fi NULL.
   * ***cod\_locatie*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul locatiei in care se afla facultatea. Acest atribut este o cheie externa care refera cheia primara din LOCATIE. Acest atribut poate fi NULL.
10. Entitatea independenta LOCATIE are ca atribute:
    * ***cod\_locatie*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul locatiei. Acest atribut reprezinta o cheie primara, deci nu poate fi NULL.
    * ***strada*** = variabila de tip caracter, de lungime maxima 25, care reprezinta numele strazii. Acest atribut poate fi NULL.
    * ***oras*** = variabila de tip caracter, de lungime maxima 25, care reprezinta numele orasului. Acest atribut poate fi NULL.
    * ***cod\_tara*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul tarii in care se afla locatia respectiva. Acest atribut reprezinta o cheie externa care refera cheia primara din TARA.
11. Entitatea independenta TARA are ca atribute:
    * ***cod\_tara*** = variabila de tip intreg, de lungime maxima 5, care reprezinta codul tari. Acest atribut reprezinta o cheie primara, deci nu poate fi NULL.
    * ***nume*** = variabila de tip caracter, de lungime maxima 25, care reprezinta numele tarii. Acest atribut poate fi NULL.
    * ***continent*** = variabila de tip caracter, de lungime maxima 25, care reprezinta numele continentului in care se afla tara. Acest atribut poate fi NULL.
12. Relatia STUDENT participa\_la DISCIPLINA predata\_de PROFESOR are ca atribute: cod\_angajat, cod\_disciplina, cod\_student. Aceste atribute trebuie sa refere cheile primare din PROFESOR, DISCIPLINA, STUDENT. Aceasta relatie formeaza tabelul asociativ PARTICIPA. De asemenea, atributele formeaza o cheie primara compusa pentru tabelul relatiei
13. Relatia STUDENT studiaza FACULTATE are ca atribute: cod\_student, cod\_facultate. Aceste atribute refera cheile primare din tabelele STUDENT si FACULTATE. Aceste atribute formeaza o cheie primara compusa pentru tabelul asociativ STUDIAZA.
14. Relatia SPECIALIZARE contine DISCIPLINA are ca atribute: cod\_disciplina, cod\_specializare. Aceste atribute refera cheile primare din tabelele SPECIALIZARE si DISCIPLINA. Aceste atribute formeaza o cheie primara compusa pentru tabelul asociativ PROGRAMA.

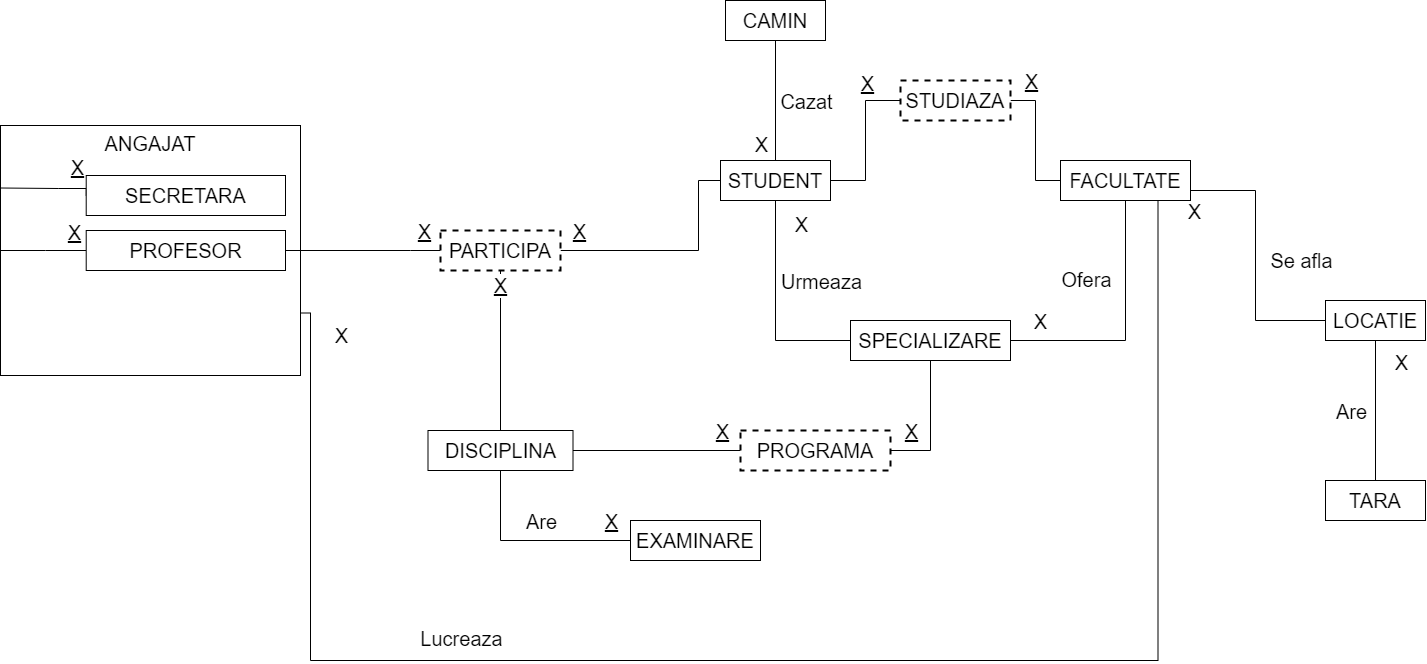
**2.**

****

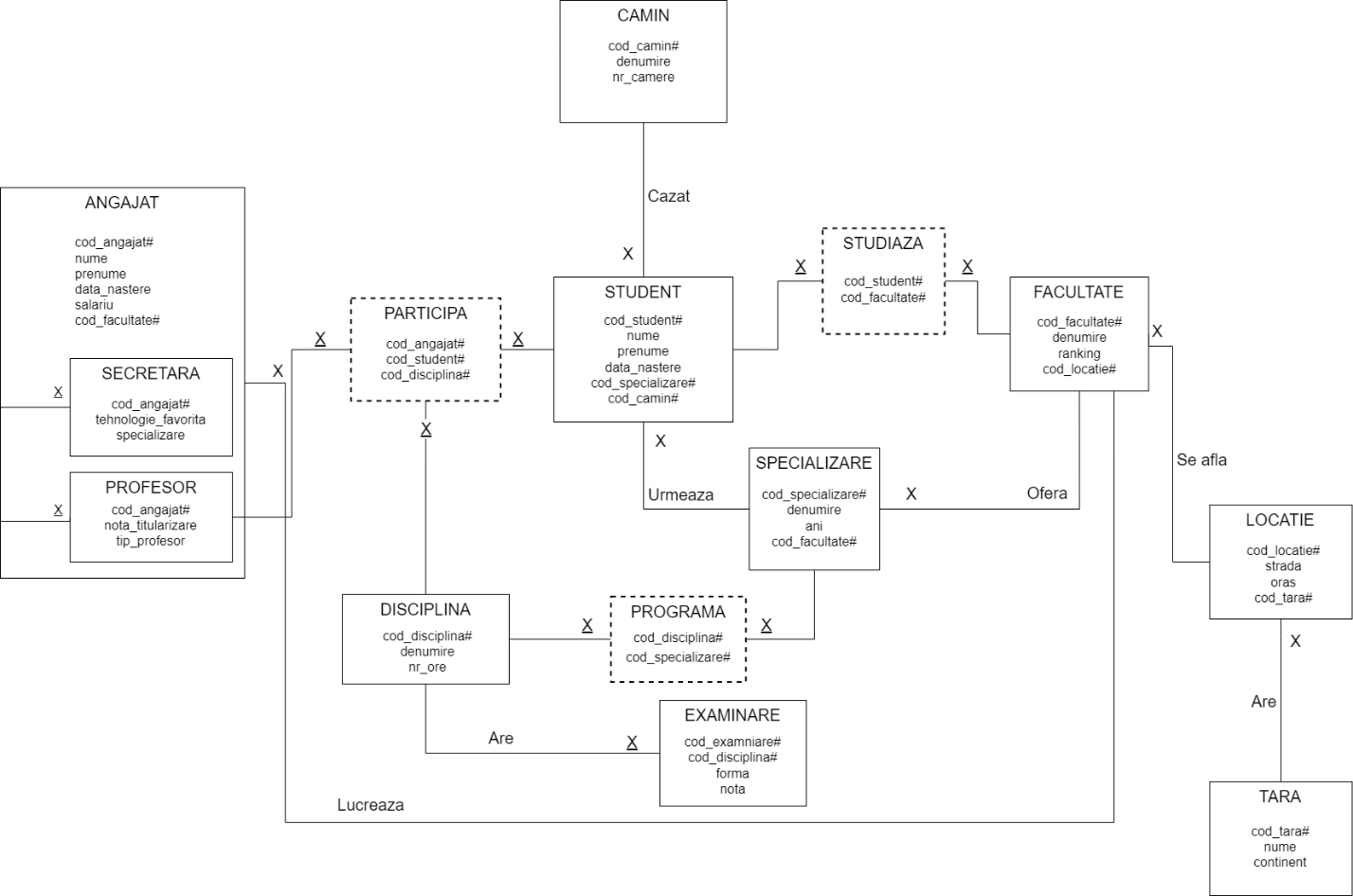
**Diagrama ER cu atribute:**

****

**3.**



**Diagrama Conceptuala cu atribute:**



**4.**

CREATE TABLE tara(

cod\_tara number(5) constraint pk\_tara primary key,

nume varchar2(25),

continent varchar2(25)

);

CREATE TABLE locatie(

cod\_locatie number(5) constraint pk\_locatie primary key,

strada varchar2(25),

oras varchar2(25),

cod\_tara number(5),

constraint fk\_locatie\_tara foreign key(cod\_tara) references tara(cod\_tara)

);

CREATE TABLE facultate(

cod\_facultate NUMBER(5) CONSTRAINT pk\_facultate PRIMARY KEY,

denumire VARCHAR2(50) CONSTRAINT denumire\_facultate NOT NULL,

ranking NUMBER(4),

cod\_locatie number(5) constraint fk\_facultate\_locatie references locatie(cod\_locatie)

);

CREATE TABLE specializare(

cod\_specializare number(5) constraint pk\_specializare primary key,

denumire varchar2(30) constraint denumire\_specializare not null,

ani number(1) constraint ani\_specializare not null,

cod\_facultate number(5) constraint fk\_specializare\_facultate references facultate(cod\_facultate)

);

CREATE TABLE camin(

cod\_camin number(5) constraint pk\_camin primary key,

denumire varchar2(30),

nr\_camere number(4)

);

CREATE TABLE student(

cod\_student number(5) constraint pk\_student primary key,

nume varchar2(25) constraint nume\_student not null,

prenume varchar2(25) constraint prenume\_student not null,

data\_nastere date,

cod\_specializare number(5) constraint fk\_student\_specializare references specializare(cod\_specializare),

cod\_camin number(5) constraint fk\_student\_camin references camin(cod\_camin)

);

CREATE TABLE studiaza(

cod\_student number(5) constraint fk\_studiaza\_student references student(cod\_student),

cod\_facultate number(5) constraint fk\_studiaza\_facultate references facultate(cod\_facultate),

CONSTRAINT pk\_studiaza PRIMARY KEY(cod\_student, cod\_facultate)

);

CREATE TABLE disciplina(

cod\_disciplina NUMBER(5) CONSTRAINT pk\_disciplina PRIMARY KEY,

denumire VARCHAR2(50) CONSTRAINT denumire\_disciplina NOT NULL,

nr\_ore NUMBER(3) CONSTRAINT nr\_ore\_disciplina NOT NULL

);

CREATE TABLE examinare(

cod\_examinare NUMBER(5),

cod\_disciplina NUMBER(5) CONSTRAINT fk\_examinare\_disciplina REFERENCES disciplina(cod\_disciplina),

forma VARCHAR2(25),

nota NUMBER(4,2),

CONSTRAINT pk\_examinare PRIMARY KEY(cod\_examinare, cod\_disciplina)

);

CREATE TABLE programa(

cod\_disciplina number(5) constraint fk\_programa\_disciplina references disciplina(cod\_disciplina),

cod\_specializare number(5) constraint fk\_programa\_specializare references specializare(cod\_specializare),

CONSTRAINT pk\_programa PRIMARY KEY(cod\_disciplina, cod\_specializare)

);

CREATE TABLE angajat(

cod\_angajat NUMBER(5) CONSTRAINT pk\_angajat PRIMARY KEY,

nume VARCHAR2(25) CONSTRAINT nume\_angajat NOT NULL,

prenume VARCHAR2(25) CONSTRAINT prenume\_angajat NOT NULL,

data\_nastere DATE default add\_months(sysdate, -12\*18),

salariu NUMBER(10,2),

cod\_facultate NUMBER(5) CONSTRAINT cod\_facultate\_ang NOT NULL

/\*

Aici practic am incercat sa setez data de nastere default ca fiind ziua de azi - 18 ani(varsta minima pentru angajati este de 18 ani)

Insa am aflat din documentatie ca nu se pot folosi functii sau sysdate in clauza check.

Cum se poate adauga o constrangere pentru a limita intervalul de valori pentru o data in SQL?

CONSTRAINT data\_nastere\_ang\_valida CHECK(data\_nastere <= add\_months(sysdate, -12\*18))

\*/

);

ALTER TABLE angajat

ADD CONSTRAINT fk\_angajat\_facultate FOREIGN KEY(cod\_facultate) REFERENCES facultate(cod\_facultate);

CREATE TABLE secretara(

cod\_angajat NUMBER(5) CONSTRAINT pk\_secretara PRIMARY KEY,

tehnologie\_favorita VARCHAR2(25),

specializare VARCHAR2(25),

CONSTRAINT pk\_secretara\_valid FOREIGN KEY(cod\_angajat) REFERENCES angajat(cod\_angajat)

);

CREATE TABLE profesor(

cod\_angajat NUMBER(5) CONSTRAINT pk\_profesor PRIMARY KEY,

nota\_titularizare NUMBER(4,2),

tip\_profesor VARCHAR2(15),

CONSTRAINT pk\_profesor\_valid FOREIGN KEY(cod\_angajat) REFERENCES angajat(cod\_angajat)

);

CREATE TABLE participa(

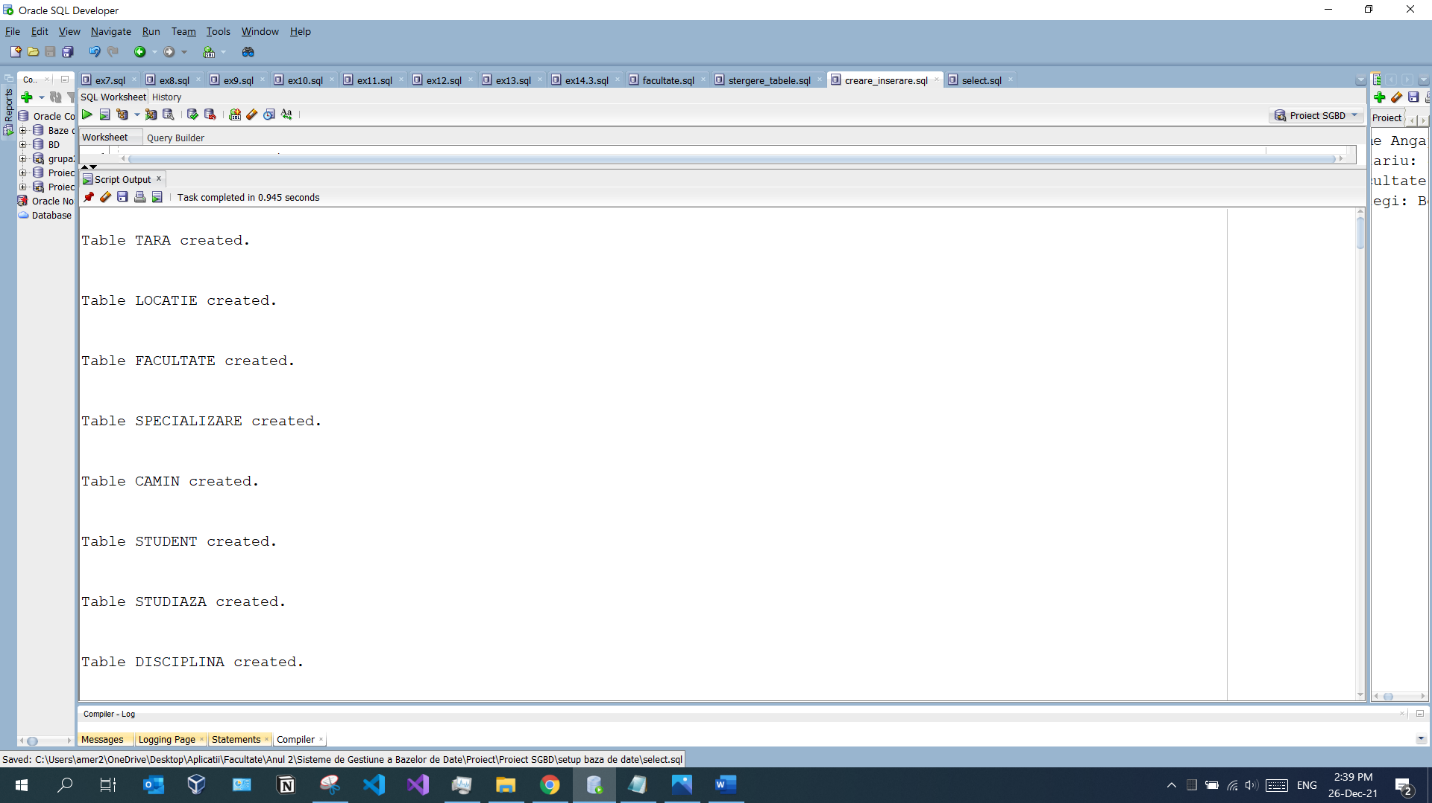
cod\_angajat number(5) constraint fk\_participa\_angajat references angajat(cod\_angajat),

cod\_disciplina number(5) constraint fk\_participa\_disciplina references disciplina(cod\_disciplina),

cod\_student number(5) constraint fk\_participa\_student references student(cod\_student),

CONSTRAINT pk\_participa PRIMARY KEY(cod\_angajat, cod\_disciplina, cod\_student)

);



**5.**

-- Inserari:

INSERT INTO tara

VALUES(1, 'Romania', 'Europa');

INSERT INTO tara

VALUES(2, 'Argentina', 'America de Sud');

INSERT INTO tara

VALUES(3, 'SUA', 'America de Nord');

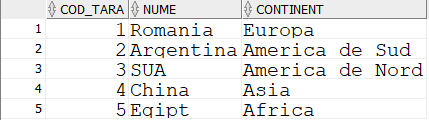
INSERT INTO tara

VALUES(4, 'China', 'Asia');

INSERT INTO tara

VALUES(5, 'Egipt', 'Africa');

commit;



INSERT INTO locatie

VALUES(1, 'Mihai Eminescu', 'Bucuresti', 1);

INSERT INTO locatie

VALUES(2, 'Mihail Kogalniceanu', 'Craiova', 2);

INSERT INTO locatie

VALUES(3, 'George Enescu', 'Timisoara', 3);

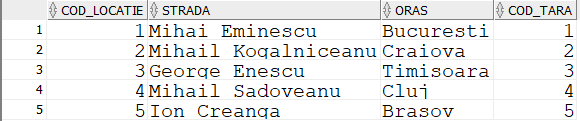
INSERT INTO locatie

VALUES(4, 'Mihail Sadoveanu', 'Cluj', 4);

INSERT INTO locatie

VALUES(5, 'Ion Creanga', 'Brasov', 5);

commit;



INSERT INTO facultate

VALUES(10, 'Facultatea de Matematica si Informatica', 1, 2);

INSERT INTO facultate

VALUES(20, 'Facultatea de Automatica si Calculatoare', 2, 5);

INSERT INTO facultate

VALUES(30, 'Facultatea de Agronomie', 3, 1);

INSERT INTO facultate

VALUES(40, 'Facultatea de Medicina', 4, 3);

INSERT INTO facultate

VALUES(50, 'Facultatea de Constructii', 5, 4);

commit;



INSERT INTO specializare

VALUES(1, 'Informatica', 3, 10);

INSERT INTO specializare

VALUES(2, 'Matematica', 4, 20);

INSERT INTO specializare

VALUES(3, 'Finante', 5, 30);

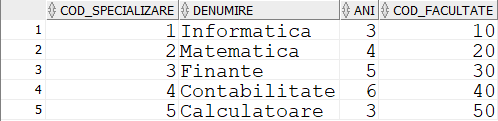
INSERT INTO specializare

VALUES(4, 'Contabilitate', 6, 40);

INSERT INTO specializare

VALUES(5, 'Calculatoare', 3, 50);

commit;



INSERT INTO camin

VALUES(1, 'Grozavesti', 50);

INSERT INTO camin

VALUES(2, 'Politehnica', 600);

INSERT INTO camin

VALUES(3, 'Magic Dorm', 120);

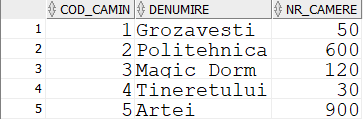
INSERT INTO camin

VALUES(4, 'Tineretului', 30);

INSERT INTO camin

VALUES(5, 'Artei', 900);

commit;



INSERT INTO student

VALUES(11, 'Ionescu', 'Stefan', '05-JAN-02', 1, 1);

INSERT INTO student

VALUES(22, 'Iliescu', 'Mihail', '12-DEC-89', 2, 2);

INSERT INTO student

VALUES(33, 'Georgescu', 'Nicolae', '23-MAY-95', 3, 3);

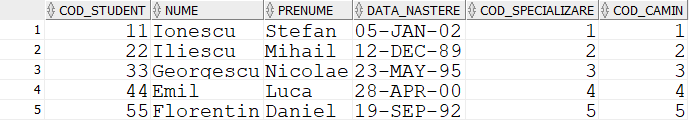
INSERT INTO student

VALUES(44, 'Emil', 'Luca', '28-APR-00', 4, 4);

INSERT INTO student

VALUES(55, 'Florentin', 'Daniel', '19-SEP-92', 5, 5);

commit;



INSERT INTO studiaza

VALUES(11, 10);

INSERT INTO studiaza

VALUES(11, 20);

INSERT INTO studiaza

VALUES(22, 20);

INSERT INTO studiaza

VALUES(22, 30);

INSERT INTO studiaza

VALUES(33, 20);

INSERT INTO studiaza

VALUES(33, 30);

INSERT INTO studiaza

VALUES(33, 40);

INSERT INTO studiaza

VALUES(44, 40);

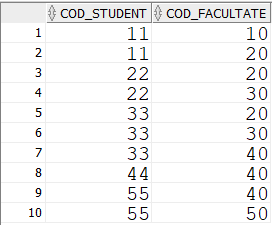
INSERT INTO studiaza

VALUES(55, 40);

INSERT INTO studiaza

VALUES(55, 50);

commit;



INSERT INTO disciplina

VALUES(11, 'Matematica', 5);

INSERT INTO disciplina

VALUES(12, 'Chimie', 6);

INSERT INTO disciplina

VALUES(13, 'Geografie', 2);

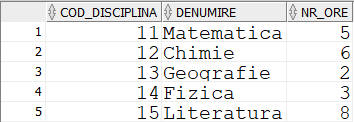
INSERT INTO disciplina

VALUES(14, 'Fizica', 3);

INSERT INTO disciplina

VALUES(15, 'Literatura', '8');

commit;



INSERT INTO examinare

VALUES(1, 11, 'Examen', 10);

INSERT INTO examinare

VALUES(2, 12, 'Proiect', 9.5);

INSERT INTO examinare

VALUES(3, 13, 'Ascultare Orala', 5.77);

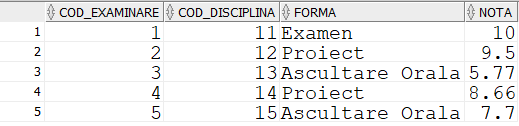
INSERT INTO examinare

VALUES(4, 14, 'Proiect', 8.66);

INSERT INTO examinare

VALUES(5, 15, 'Ascultare Orala', 7.7);

commit;



INSERT INTO programa

VALUES(11, 1);

INSERT INTO programa

VALUES(12, 1);

INSERT INTO programa

VALUES(12, 2);

INSERT INTO programa

VALUES(12, 3);

INSERT INTO programa

VALUES(13, 1);

INSERT INTO programa

VALUES(13, 3);

INSERT INTO programa

VALUES(14, 2);

INSERT INTO programa

VALUES(14, 3);

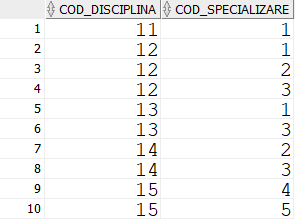
INSERT INTO programa

VALUES(15, 4);

INSERT INTO programa

VALUES(15, 5);

commit;



INSERT INTO angajat

VALUES(1, 'Popescu', 'Ion', '17-JAN-87', 1500, 10);

INSERT INTO angajat

VALUES(2, 'Stefanescu', 'Teodora', '25-DEC-86', 2500, 10);

INSERT INTO angajat

VALUES(3, 'Mihaileascu', 'Andrei', '06-MAY-92', 3800, 10);

INSERT INTO angajat

VALUES(4, 'Popovoci', 'Alexandra', '21-AUG-82', 1800, 20);

INSERT INTO angajat

VALUES(5, 'Ionescu', 'Mihai', '12-SEP-96', 2300, 20);

INSERT INTO angajat

VALUES(6, 'Stanescu', 'Constantin', '19-OCT-89', 3700, 20);

INSERT INTO angajat

VALUES(7, 'Popa', 'Andra', '08-APR-90', 3200, 30);

INSERT INTO angajat

VALUES(8, 'Mihai', 'George', '27-JUL-82', 1900, 30);

INSERT INTO angajat

VALUES(9, 'Marinescu', 'Elena', '19-SEP-77', 4100, 30);

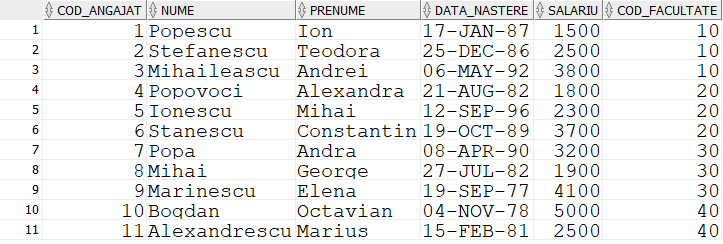
INSERT INTO angajat

VALUES(10, 'Bogdan', 'Octavian', '04-NOV-78', 5000, 40);

INSERT INTO angajat

VALUES(11, 'Alexandrescu', 'Marius', '15-FEB-81', 2500, 40);

commit;



INSERT INTO secretara

VALUES(1, 'Powerpoint', 'Prezentari');

INSERT INTO secretara

VALUES(3, 'Word', 'Birocratie');

INSERT INTO secretara

VALUES(5, 'Excel', 'Organizare');

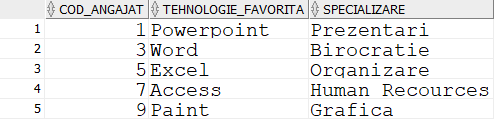
INSERT INTO secretara

VALUES(7, 'Access', 'Human Recources');

INSERT INTO secretara

VALUES(9, 'Paint', 'Grafica');

commit;



INSERT INTO profesor

VALUES(2, 9.25, 'Cursant');

INSERT INTO profesor

VALUES(4, 5.23, 'Seminarist');

INSERT INTO profesor

VALUES(6, 7.5, 'Laborant');

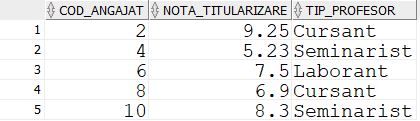
INSERT INTO profesor

VALUES(8, 6.9, 'Cursant');

INSERT INTO profesor

VALUES(10, 8.3, 'Seminarist');

commit;



INSERT INTO participa

VALUES(2, 11, 11);

INSERT INTO participa

VALUES(2, 11, 22);

INSERT INTO participa

VALUES(2, 11, 33);

INSERT INTO participa

VALUES(4, 12, 11);

INSERT INTO participa

VALUES(4, 12, 44);

INSERT INTO participa

VALUES(6, 13, 55);

INSERT INTO participa

VALUES(6, 13, 33);

INSERT INTO participa

VALUES(8, 14, 44);

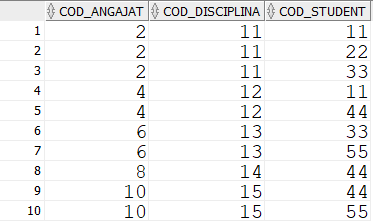
INSERT INTO participa

VALUES(10, 15, 44);

INSERT INTO participa

VALUES(10, 15, 55);

commit;



**6.**

/\*

pentru un id dat ca parametru si incepand cu urmatoarele linii de dupa acest id pana la sfarsitul tabelului

(incepand cu acest id)

pentru fiecare student al carui id este egal cu un numar fibonacci +- o marja de eroare de maxim 1 numar

sa se afiseze numele profesorilor la care acesta invata precum si salariul acestora

afisarea se va face sub forma: nume prenume salariu

pentru fiecare profesor in parte

De asemenea, sa se afiseze lista tuturor numerelor fibonacci mai mici sau egale +- 1 decat id-ul maxim al categoriei

de studenti aflate pe liniile urmatoare din tabel fata de id-ul dat ca parametru

Daca nu exista studenti cu id-uri egale cu numere fibonacci, se va arunca o exceptie "NO\_FIBONACCI\_FOUND"

daca mai mult de jumatate + 1 dintre toti studentii eligibili(care se afla sub id-ul dat ca parametru)

au id-uri egale cu numere fibonacci +- marja de eroare, atunci se va arunca exceptia

"TOO\_MANY\_FIBONACCI"

Spunem ca un numar este egal +-1 cu un numar fibonacci x daca acesta este cuprins in intervalul [x-1, x+1].

Indiferent daca se va arunca o exceptie no\_fibonacci\_found sau too\_many\_fibonacci

id-ul dat, id-ul maxim, precum si numerele fibonacci mai mici ca id-ul maxim vor fi afisate.

In cazul exceptiei no\_data\_found, se va trata direct exceptia, intrucat nu se poate face

nimic fara un id valid.

Rezolvare:

Verificam daca id-ul dat ca parametru este valid, daca nu este valid ne oprim la acest pas

Calculam id-ul maxim luand in considerare doar id-urile care vin imediat dupa id-ul dat ca parametru

Folosim un tablou indexat pentru numerele fibonacci, trebuie sa gasim id-ul maxim pentru ultimul nr fibonacci

Folosim un record cu numele si salariul unui profesor

Folosim un varray de tipul record anterior mentionat pentru a retine detaliile tuturor profesorilor la care invata un student

Folosim un tablou imbricat de varray-uri pentru a contrui o matrice in care fiecare linie i reprezinta lista profesorilor studentului i

Pentru majoritatea operatiilor vom folosi un cursor pentru a parcurge tabelul student

\*/

**-- Cod Exercitiul 6:**

create or replace procedure exercitiul\_6(

id student.cod\_student%type -- codul studentului dat ca parametru

)

is

cursor c is select rownum, cod\_student, nume, prenume from student; -- cursor pentru a parcurge liniile din tabelul student

nr\_linie\_id number; -- numarul liniei parcurse de cursor

id\_student student.cod\_student%type; -- id-ul studentului parcurs de cursor

nume student.nume%type; -- numele studentului parcurs de cursor

prenume student.prenume%type; -- prenumele studentului parcurs de cursor

id\_maxim student.cod\_student%type; -- id-ul de valoare maxima a studentilor aflati pe liniile urmatoare id-ului de student dat ca parametru

type tab\_ind\_fibonacci is table of number index by pls\_integer; -- tablou indexat ce memoreaza numerele fibonacci mai mici +-1 decat id\_maxim

ind tab\_ind\_fibonacci; -- tabloul indexat in care o sa memoram numerele fibonacci calculate

contor number; -- contor folosit pentru a parcurge tabloul indexat anterior declarat

nr\_studenti number; -- numarul total de studenti

sem boolean; -- semafor folosit pentru eficientizarea timpului de executie

no\_fibonacci\_found exception; -- exceptie specifica exercitiului

too\_many\_fibonacci exception; -- exceptie specifica exercitiului

type detalii\_profesor is record( -- tip inregistrare(record) pentru memorarea detaliilor despre un profesor

nume angajat.nume%type,

prenume angajat.prenume%type,

salariu angajat.salariu%type

);

type v\_detalii\_profesor is varray(25) of detalii\_profesor; -- varray de 25 de detalii\_profesor

type tab\_imb\_nume\_profesori is table of v\_detalii\_profesor; -- tablou imbricat al carui elemente sunt varray-uri de detalii\_profesor(matrice)

imb tab\_imb\_nume\_profesori := tab\_imb\_nume\_profesori(); -- imb este practic o matrice de detalii\_profesor

begin

-- verificam daca id-ul dat ca parametru exista de fapt in baza de date

select cod\_student into id\_student from student where cod\_student = id;

-- generam NO\_DATA\_FOUND daca nu gasim id-ul dat ca parametru

-- Daca nu s-a generat eroarea NO\_DATA\_FOUND inseamna ca id-ul exista si il avem memorat in variabila auxiliara id\_student

open c; -- deschidem cursorul pentru a incepe parcurgerea tabelului student

loop -- cat timp nu am parcurs tot tabelul

fetch c into nr\_linie\_id, id\_student, nume, prenume; -- plasam valorile liniei curente in variabilele asociate

exit when c%notfound; -- oprim bucla atunci cand am trecut de ultima linie a tabelului

if id\_student = id then -- daca id-ul curent este egal cu id-ul dat ca parametru

exit; -- inseamna ca putem sa iesim din bucla parcurgerii cursorului

end if;

end loop;

-- am gasit linia unde se afla student-ul cu id-ul dat

-- trebuie sa calculam id-ul maxim dintre cei ramasi

id\_maxim := id\_student; -- initial, id-ul maxim va fi fix id-ul dat ca parametru

loop -- incepand cu linia imediat urmatoare liniei in care se afla id-ul dat ca parametru

exit when c%notfound; -- verificam intai daca id-ul dat ca parametru se afla deja pe ultima linie a tabelului student

fetch c into nr\_linie\_id, id\_student, nume, prenume; -- Daca nu era pe ultima linie, atunci incepem cautarea id-ului maxim

if id\_student > id\_maxim then -- Daca id-ul curent este mai mare decat id-ul maxim calculat pana in acest moment

id\_maxim := id\_student; -- Atunci noul id\_maxim devine id-ul curent

end if;

end loop;

dbms\_output.put\_line('ID-ul dat: ' || id); -- Printam id-ul dat ca parametru

dbms\_output.put\_line('ID-ul maxim: ' || id\_maxim); -- Printam id-ul maxim incepand cu linia urmatoare id-ului curent

close c;

dbms\_output.new\_line; -- Printam o noua linie

-- Acum ca avem id-ul maxim calculat, putem incepe calcularea numerelor fibonacci mai mici +-1 decat id\_maxim

if 1 <= id\_maxim + 1 then -- Daca primul termen fibonacci(1) este mai mic sau egal decat id\_maxim + 1 atunci putem incepe calcularea

ind(1) := 1; -- Primul termen fibonacci este 1

ind(2) := 1; -- Al doilea termen fibonacci este 2

contor := 3; -- Calculul incepe cu al treilea termen fibonacci, deci contor este 3

else -- Daca in schimb id\_maxim + 1 este mai mic decat primul termen fibonacci, atunci nu avem ce termeni fibonacci sa calculam(numerele fibonacci sunt strict pozitive)

raise no\_fibonacci\_found; -- Prin urmare, aruncam exceptia no\_fibonacci\_found, intrucat nu avem numere fibonacci

end if;

-- Daca nu s-a aruncat exceptia, atunci putem continua calcularea numerelor fibonacci

while ind(contor-1) + 1 <= id\_maxim loop -- Cat timp termenul anterior + 1 este mai mic sau egal decat id\_maxim calculam urmatorul termen fibonacci

if ind(contor - 1) + ind(contor - 2) > id\_maxim then -- Daca urmatorul termen fibonacci va fi mai mare strict decat id\_maxim, nu il mai calculam

exit; -- Ci in schimb iesim din bucla

end if;

ind(contor) := ind(contor-1) + ind(contor-2); -- fib(i) = fib(i-1) + fib(i-2);

contor := contor + 1; -- i++;

end loop;

dbms\_output.put\_line('Numerele fibonacci mai mici ca ' || id\_maxim || ':'); -- Incepem afisarea termenilor fibonacci calculati

for i in ind.first..ind.last loop -- Parcurgem tabloul indexat folosind o bucla for

dbms\_output.put(ind(i) || ' '); -- Printam toti termenii fibonacci intr-o singura linie separand prin spatiu

end loop;

dbms\_output.new\_line; -- Afisam 3 linii noi pentru a separa continutul afisat

dbms\_output.new\_line;

dbms\_output.new\_line;

-- Determinam numarul de studenti care se afla dupa id-ul dat, precum si numarul de studenti al caror id este fibonacci

nr\_studenti := 0; -- Initial consideram ca numarul de studenti aflati sub id-ul dat este 0

contor := 0; -- In contor vom memora numarul de id-uri fibonacci, contor este initializat cu 0

open c; -- Deschidem cursorul din nou, acesta se afla inainte de prima linie din tabel

loop -- Cautam sa ajungem la linia unde se afla id-ul dat ca parametru

fetch c into nr\_linie\_id, id\_student, nume, prenume;

exit when id\_student = id; -- Odata ce am ajuns la linia cu id-ul dat ca parametru oprim bucla

end loop;

loop -- Incepem sa numaram studentii aflati sub id-ul dat

nr\_studenti := nr\_studenti + 1; -- Luam in considerare si id-ul dat ca fiind unul elegibil

-- Testam daca id-ul curent este fibonacci

sem := true; -- Consideram sem = true atunci cand id-ul nu este fibonacci

for i in ind.first..ind.last loop -- Parcurgem tabloul de numere fibonacci

if id\_student >= ind(i) - 1 and id\_student <= ind(i) + 1 then -- Daca id-ul curent este egal +-1 fata de termenul fibonacci curent

contor := contor + 1; -- Incrementam contor-ul pentru a creste numarul de id-uri fibonacci

sem := false; -- sem devine false, intrucat stim ca id-ul este fibonacci

end if;

if sem = false then -- Daca am determinat ca id-ul este fibonacci, nu mai este nevoie sa comparam cu restul numerelor fibonacci din tablou

exit; -- Prin urmare, putem iesi din bucla for

end if;

end loop;

fetch c into nr\_linie\_id, id\_student, nume, prenume; -- Trecem la urmatorul id

exit when c%notfound; -- Repetam procesul pentru fiecare id in parte

end loop;

close c; -- Inchidem cursorul odata ce am terminat

/\*

Pana acum avem urmatoarele:

contor = numarul de studenti cu id fibonacci

nr\_studenti = numarul tuturor studentilor

\*/

-- dbms\_output.put\_line(contor);

-- dbms\_output.put\_line(nr\_studenti);

if contor = 0 then -- daca contor = 0 inseaman ca nu avem id-uri fibonacci si aruncam exceptia no\_fibonacci\_found

raise no\_fibonacci\_found;

elsif contor > 0.5 \* nr\_studenti then -- daca avem mai mult de jumatate + 1 dintre toti studentii eligibil cu id-uri fibonacci, intram pe exceptia TOO\_MANY\_FIBONACCI

raise too\_many\_fibonacci;

end if;

-- daca nu am intrat pe exceptia TOO\_MANY\_FIBONACCI, inseamna putem sa

-- afisam toti studentii care se incadreaza in conditia id-ului:

-- Practic nu mai avem nevoie de numarul de id-uri fibonacci, deci putem sa reutilizam variabila contor

open c; -- Deschidem cursorul pentru parcurgerea tabelului student

loop

fetch c into nr\_linie\_id, id\_student, nume, prenume;

exit when id\_student = id; -- Parcurgem toate liniile pana la linia cu id-ul dat ca parametru

end loop;

-- Incepem sa construim matricea de detalii\_profesor din imb

contor := 1; -- Contor ia primul indice al tabloului imb

loop

-- Pentru fiecare id de dupa id-ul parametru, verificam daca este fibonacci si afisam datele

for i in ind.first..ind.last loop

if id\_student >= ind(i) - 1 and id\_student <= ind(i) + 1 then -- Daca id-ul este id fibonacci afisam informatiile

dbms\_output.put\_line('Nr linie: ' || nr\_linie\_id);

dbms\_output.put\_line('ID: ' || id\_student);

dbms\_output.put\_line('Nume Student: ' || nume || ' ' || prenume);

dbms\_output.put('Profesori: ');

imb.extend; -- necesar tablourilor imbricate si varray-urilor

select a.nume, a.prenume, a.salariu -- Selectam toti profesorii studentului cu id-ul curent

bulk collect into imb(contor) -- Stocam toate campurile selectate in imb(contor), care este de tipul varray(25) de detalii\_profesor

from participa p join student s on (s.cod\_student = p.cod\_student)

join profesor pf on (p.cod\_angajat = pf.cod\_angajat)

join angajat a on (p.cod\_angajat = a.cod\_angajat)

where s.cod\_student = id\_student

order by s.nume;

for i in imb(contor).first..imb(contor).last loop -- Parcurgem lista tuturor profesorilor unui student

dbms\_output.put(imb(contor)(i).nume || ' ' || imb(contor)(i).prenume || ' ' || imb(contor)(i).salariu); -- Afisam numele, prenumele si salariul profesorului curent

if i < imb(contor).last then -- Daca nu am ajuns la ultimul profesor

dbms\_output.put(', '); -- Afisam virgula ca separator

else

dbms\_output.put('.'); -- Daca am ajuns la ultimul profesor, punem simbolul punct.

end if;

end loop;

contor := contor + 1; -- Incrementam contor-ul pentru a trece la urmatorul element al lui imb

dbms\_output.new\_line; -- Punem new\_line pentru a se afisa toate dbms\_output.put()-urile executate in for

dbms\_output.new\_line; -- Punem inca un new\_line pentru a pastra afisarea mai spatiata

end if;

end loop;

fetch c into nr\_linie\_id, id\_student, nume, prenume; -- Trecem la urmatorul student

exit when c%notfound; -- Incheiem bucla atunci cand am terminat studentii

end loop;

close c;

ind.delete; -- stergem tabloul indexat

for i in imb.first..imb.last loop

imb(i).delete; -- Stergem fiecare varray existent in imb

end loop;

imb.delete; -- Stergem tabloul imbricat

exception

when no\_data\_found then dbms\_output.put\_line('ID-ul dat nu exista in baza de date'); -- Atunci cand avem no\_data\_found inseamna ca nu exista student cu id-ul dat ca parametru

when no\_fibonacci\_found then dbms\_output.put\_line('Niciun student nu are id fibonacci'); -- Atunci cand nu exista niciun id fibonacci sau daca id\_maxim este mai mic decat primul termen fibonacci

when too\_many\_fibonacci then dbms\_output.put\_line('Mai mult de jumatate + 1 dintre studentii eligibili au id-uri fibonacci'); -- Atunci cand mai mult de jumatate + 1 dintre id-uri sunt fibonacci primim aceasta eroare

-- nu intalnim erori de tipul too\_many\_rows, intrucat toate select-urile pe care le

-- efectuam se fac impreuna cu clauza where cod\_student = id;

-- din moment ce prin definitie cod\_student este cheie primara in tabela student

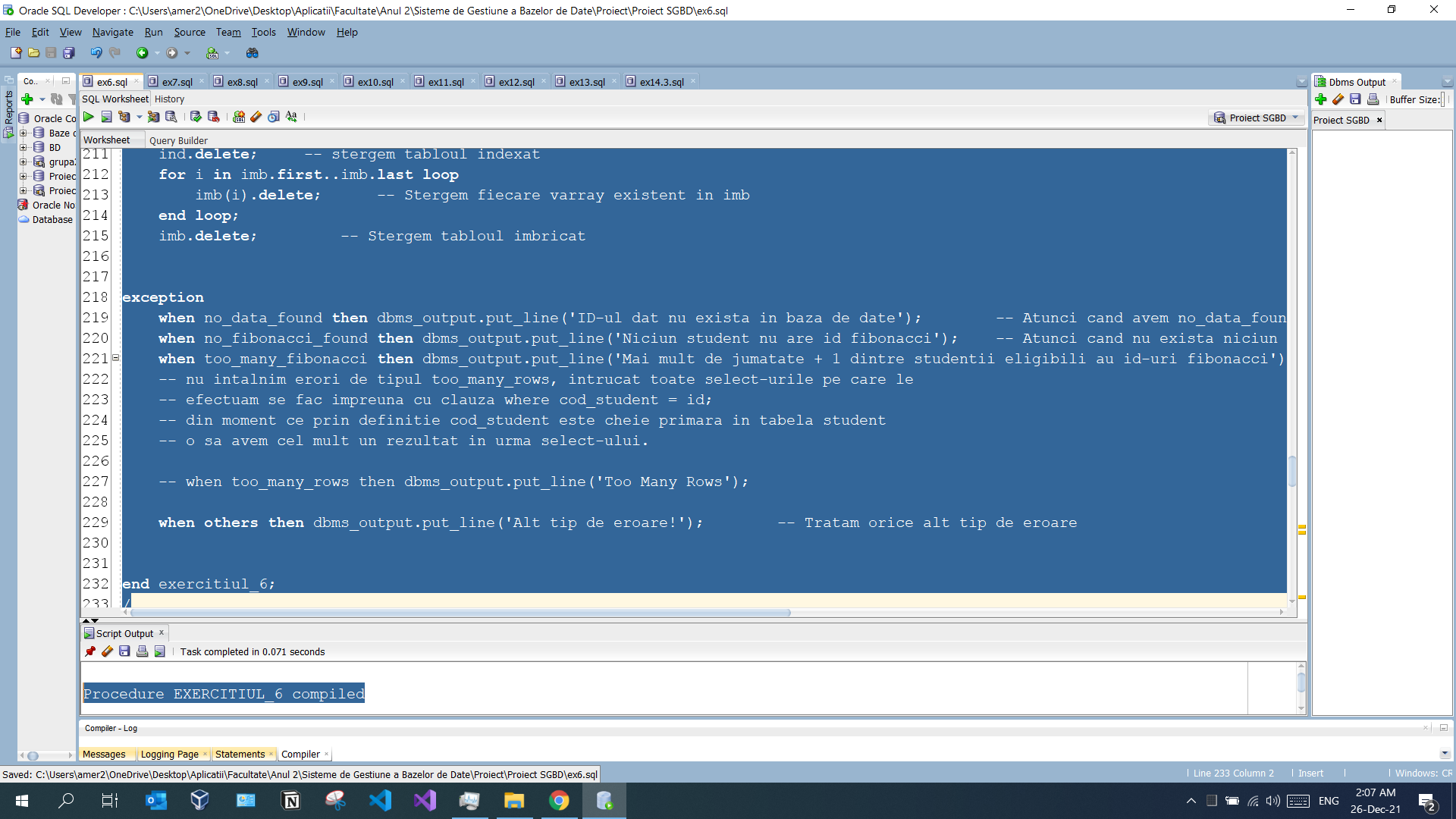
-- o sa avem cel mult un rezultat in urma select-ului.

-- when too\_many\_rows then dbms\_output.put\_line('Too Many Rows');

when others then dbms\_output.put\_line('Alt tip de eroare!'); -- Tratam orice alt tip de eroare

end exercitiul\_6;

/



**-- Rezultate/Explicatii:**

/\*

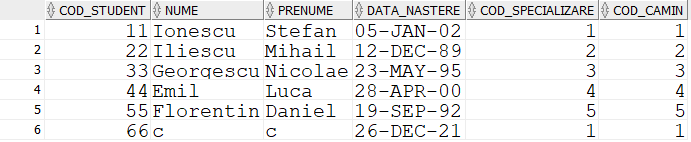
Pentru a vedea mai bine cum se executa programul, vom insera inca o valoare oarecare al carui id nu este numar fibonacci

\*/

insert into student

values(66, 'c', 'c', sysdate, 1, 1);

select \* from student;



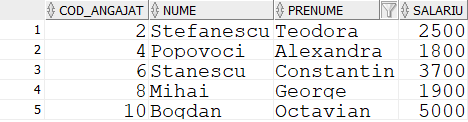
/\*

De asemenea, vom selecta si informatiile despre toti profesorii pentru a le avea la dispozitie.

\*/

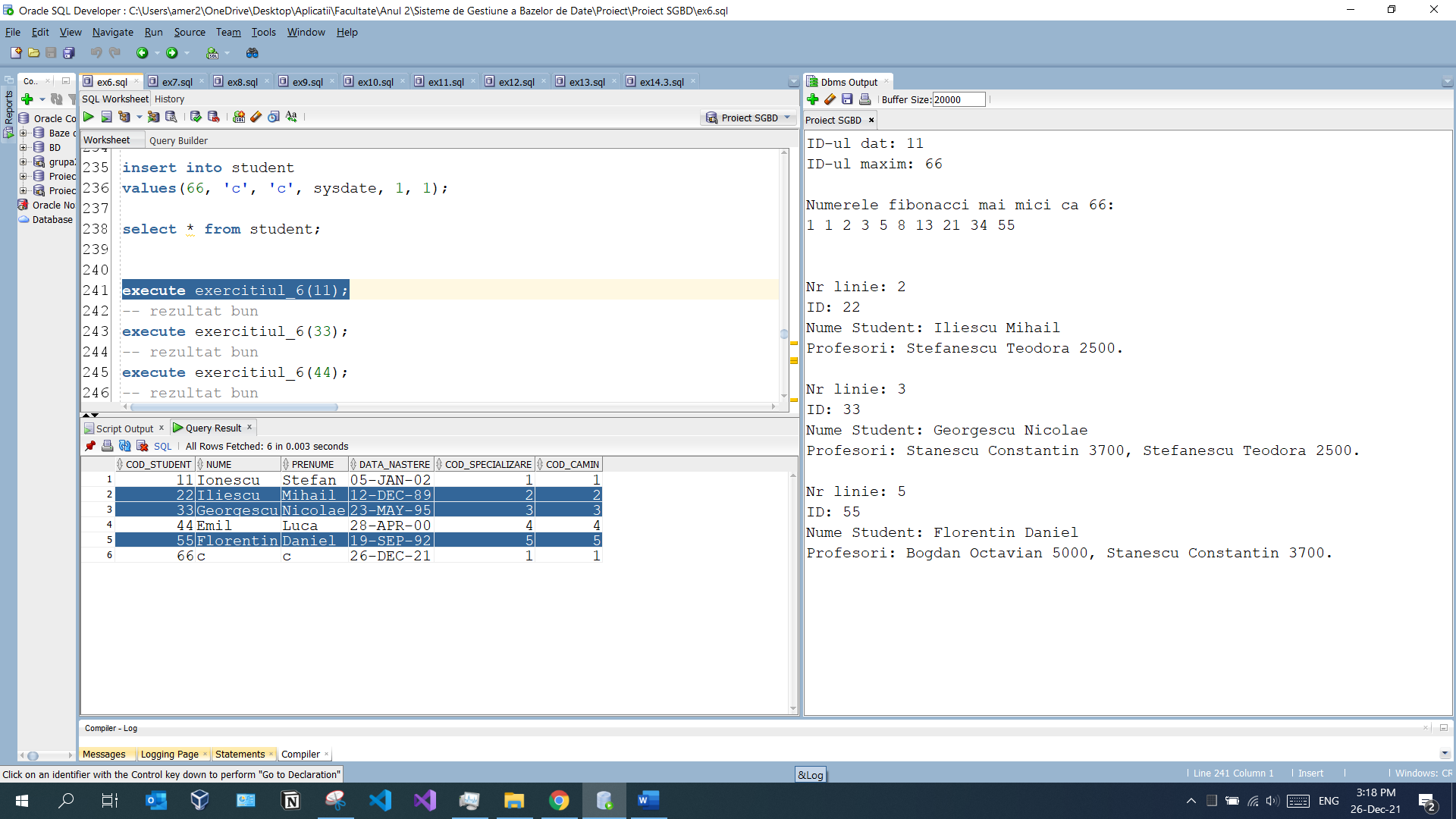
select p.cod\_angajat, nume, prenume, salariu

from angajat a join profesor p on (a.cod\_angajat = p.cod\_angajat);



execute exercitiul\_6(11);

-- rezultat bun



/\*

Id-ul dat ca parametru este 11, acesta este existent in baza de date si deci nu se genereaza exceptia no\_data\_found. Id-urile care se vor lua in considerare in calculele din cadrul procedurii sunt practic toate id-urile incepand de la 11 inclusiv.

Se afiseaza id-ul dat(11), id-ul maxim(66), si numerele fibonacci mai mici +-1 decat 66

In acest caz, din id-urile eligibile avem ca:

11, 44 si 66 sunt id-uri normale(nu sunt fibonacci)

22 este egal prin marja de eroare +1 cu 21 care este numar fibonacci

33 este egal prin marja de eroare -1 cu 34 care este numar fibonacci

55 este numar fibonacci

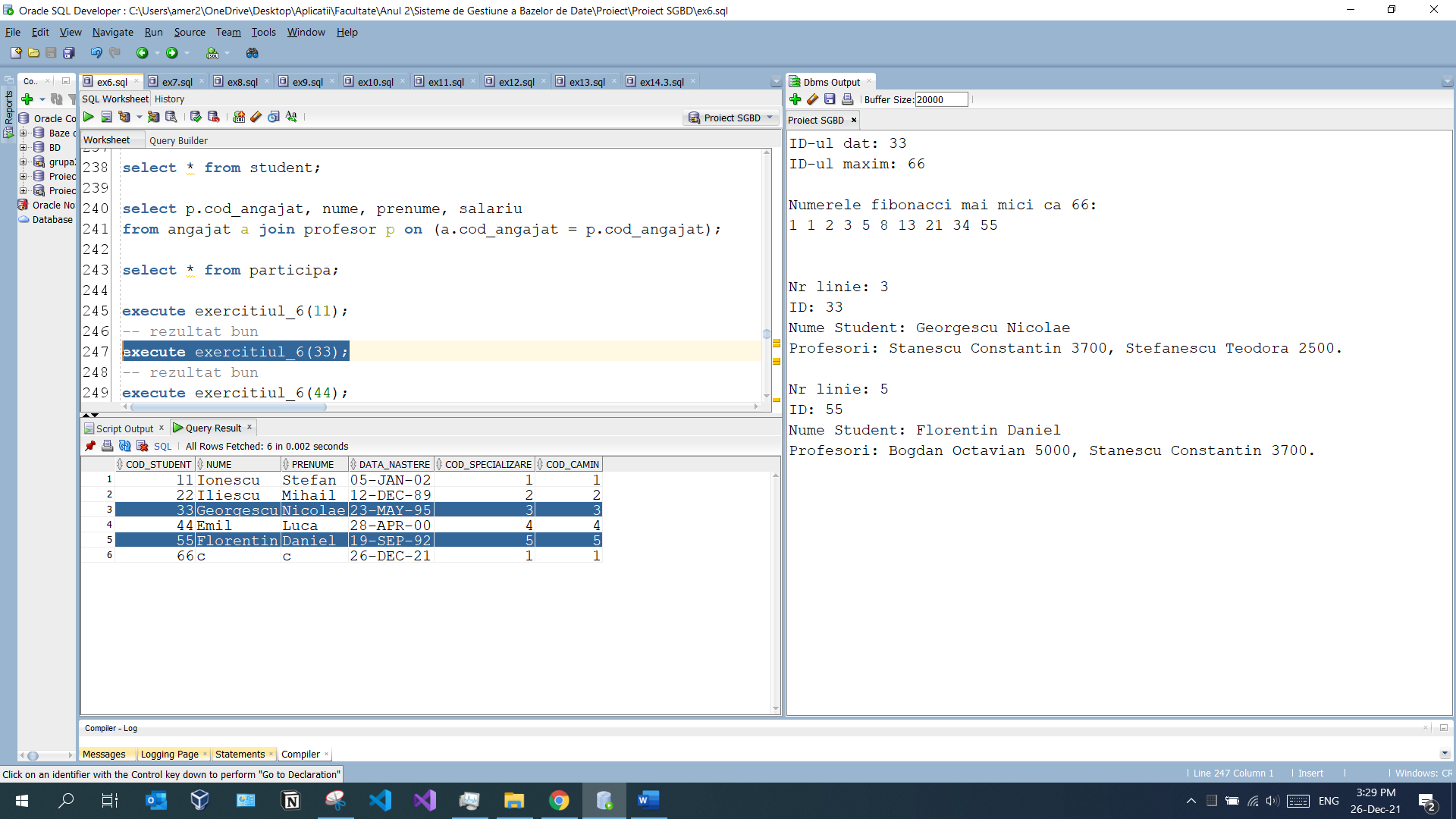
Avem 3 id-uri fibonacci si 3 id-uri normale. Prin urmare, nu avem mai mult de jumatate + 1 din id-urile eligibile ca fiind fibonacci si deci nu se arunca exceptia too\_many\_fibonacci. De asemenea, nu se arunca nici exceptia no\_fibonacci\_found, intrucat am gasit 3 id-uri fibonacci.

Procedura se va executa normal, fara aruncare de exceptii. Pentru fiecare id fibonacci se afiseaza detaliile studentului precum si profesorii acestuia.

\*/

execute exercitiul\_6(33);

-- rezultat bun



/\*

Similar exemplului anterior, de data asta avem ca id dat ca parametru pe 33.

Id-urile ce vin dupa 33 sunt 44, 55 si 66. In acest caz avem ca:

33 este egal prin marja de eroare +1 cu 34 care este numar fibonacci

44 nu este numar fibonacci

55 este numar fibonacci

66 nu este numar fibonacci

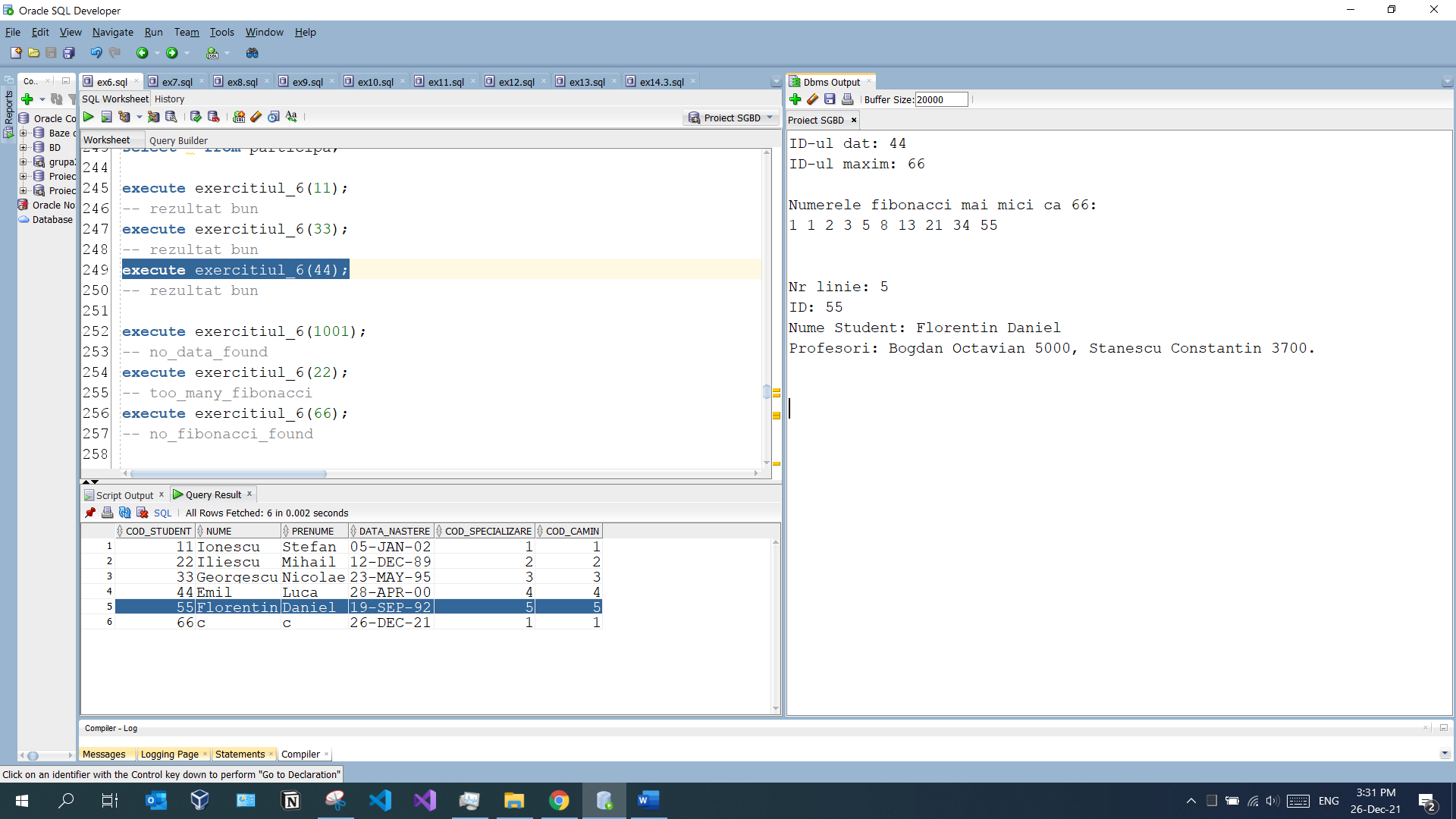
Avem 2 id-uri fibonacci si 2 id-uri care nu sunt fibonacci, deci nu intram pe exceptia too\_many\_fibonacci. Nu intram nici pe exceptiile no\_data\_found sau no\_fibonacci\_found.

Pentru id-urile 33 si 55 se afiseaza informatiile corespunzatoare, intrucat acestea sunt egale +-1 fibonacci.

\*/

execute exercitiul\_6(44);

-- rezultat bun



/\*

Id-ul dat ca parametru este 44, in acest caz avem ca id-uri eligibile:

44 care nu este numar fibonacci

55 care este numar fibonacci

66 care nu este numar fibonacci

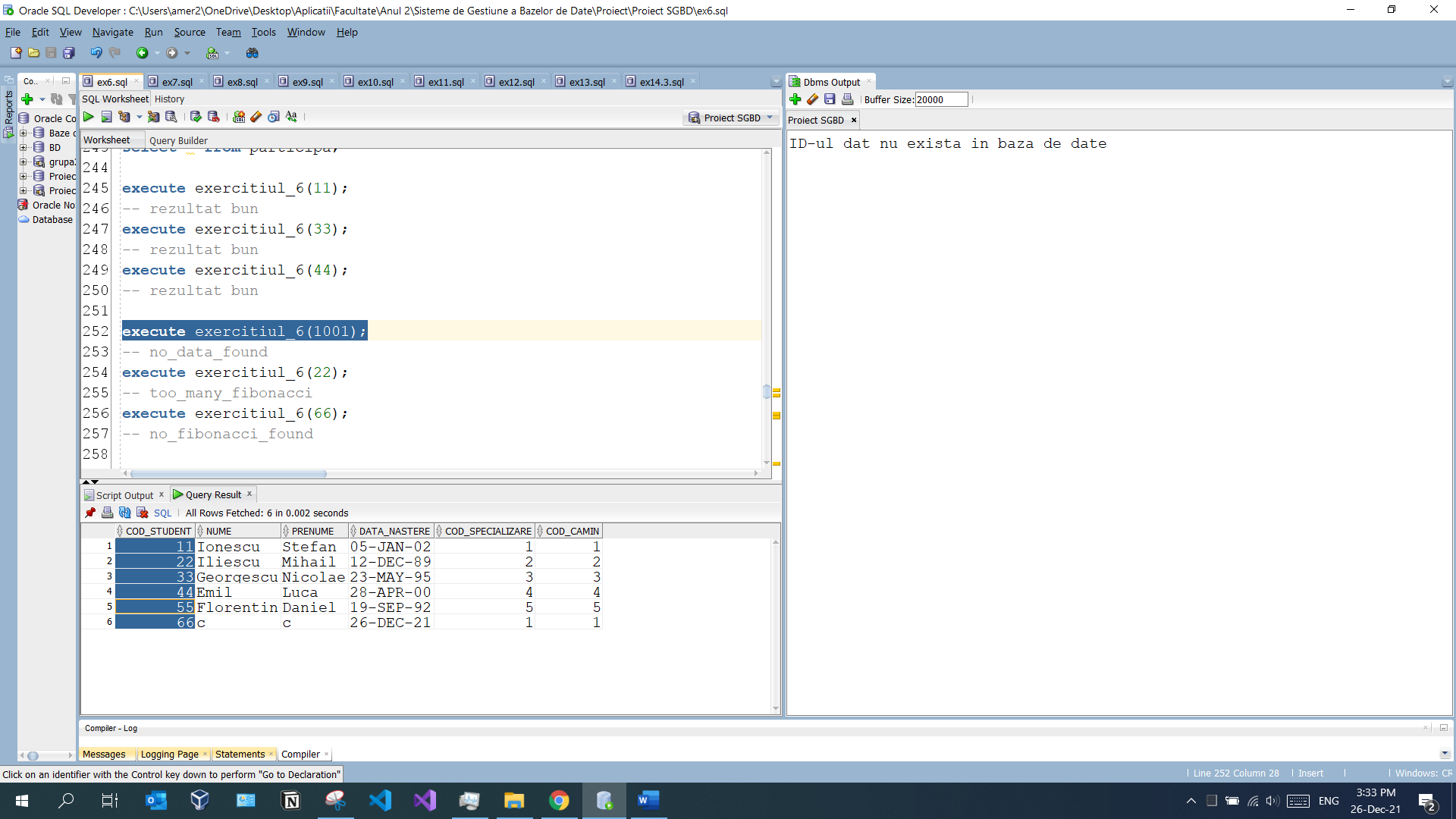
Nu mai intram pe exceptia too\_many\_fibonacci, intrucat avem mai putin de jumatate + 1 dintre id-uri ca fiind numere fibonacci.

In acest caz, se va afisa numele studentului cu id-ul 55, precum si profesorii si salariul profesorilor la care acesta invata.

\*/

execute exercitiul\_6(1001);

-- no\_data\_found



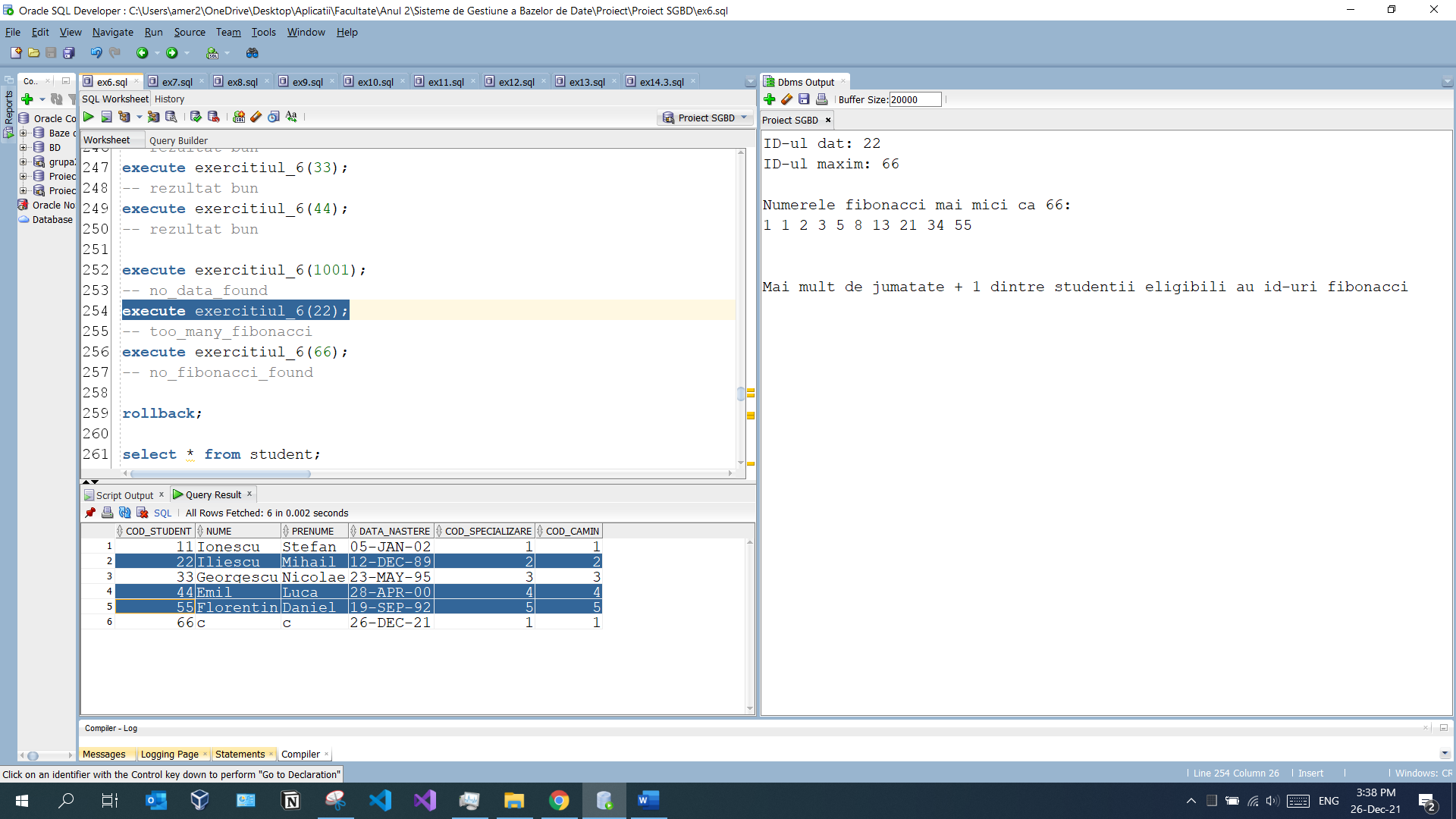
/\*

Id-ul 1001 nu exista in baza de date, se arunca exceptia no\_data\_found. Asa cum apare in query-ul de select din poza, 1001 nu se afla printre id-urile studentilor. Atunci cand se intra pe no\_data\_found, se afiseaza doar un mesaj sugestiv si se opreste executia procedurii, intrucat nu se poate face nimic fara un id valid.

\*/

execute exercitiul\_6(22);

-- too\_many\_fibonacci



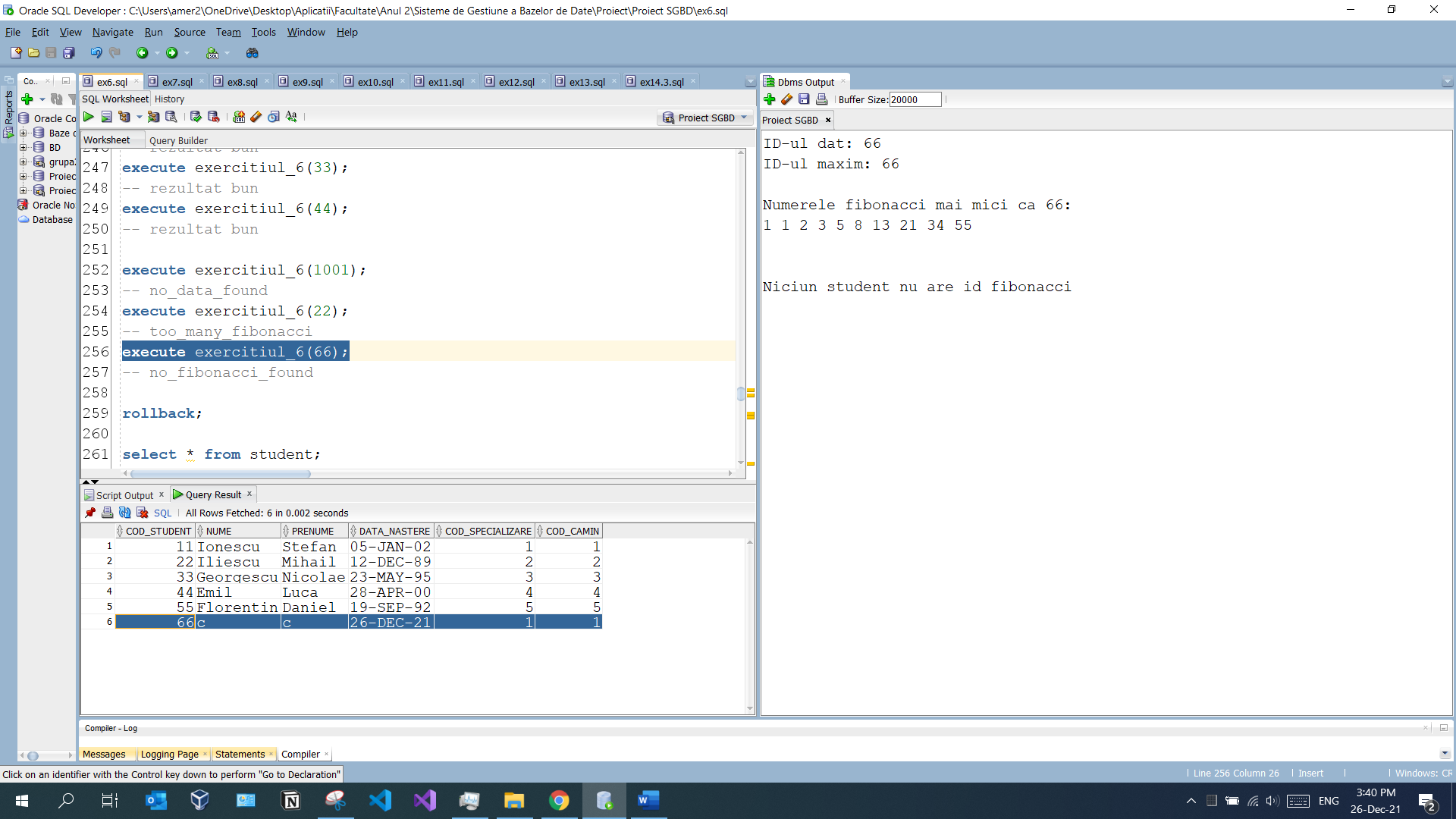
/\*

Pentru id-ul 22 dat ca parametru, o sa avem ca id-uri fibonacci pe 22, 33 si 55, iar ca id-uri normale doar pe 44 si 66. In acest caz, mai mult de jumatate + 1 dintre id-urile eligibile sunt fibonacci si se intra pe exceptia too\_many\_fibonacci.

\*/

execute exercitiul\_6(66);

-- no\_fibonacci\_found



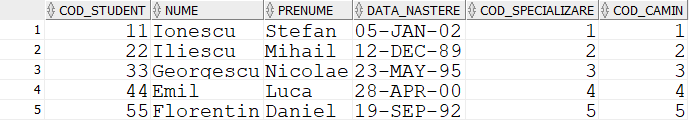
/\*

Singurul id eligibil este 66, acesta fiind pe ultima linie. Cum 66 nu este egal +-1 cu orice numar fibonacci, se intra pe exceptia no\_fibonacci\_found.

\*/

rollback;

-- Se executa rollback pentru eliminarea inserarii facute la inceput strict pentru acest exercitiu



**7.**

/\*

Sa se afiseze toti angajatii al caror salariu se incadreaza intre cei doi parametri a si b, precum si colegii lor de munca la facultate.

\*/

/\*

OBSERVATIE: Inainte de a rula urmatoarele teste este recomandat sa se verifice integritatea datelor din tabelul angajat. Mai exact, este important ca salariile sa fie aceleasi ca cele inserate initial, intrucat acestea ar putea ramane modificate de trigger-ul de la exercitiul 10.

\*/

**-- Cod Exercitiul 7:**

create or replace procedure exercitiul\_7(

a angajat.salariu%type, -- Limita inferioara a intervalului salarial

b angajat.salariu%type -- Limita superioara a intervalului salarial

)

is

-- cursor explicit parametrizat

-- selecteaza numele, prenumele, salariul profesorilor ca se incadreaza in intervalul salarial

-- precum si denumirea facultatii la care acesti profesori predau

cursor c(x angajat.salariu%type, y angajat.salariu%type) is

select nume, prenume, salariu, denumire

from angajat join facultate using(cod\_facultate)

where salariu between x and y;

nume angajat.nume%type; -- variabila pentru numele profesorului

prenume angajat.prenume%type; -- variabila pentru prenumele profesorului

salariu angajat.salariu%type; -- variabila pentru salariul profesorului

fac facultate.denumire%type; -- variabila pentru denumirea facultatii la care preda profesorul

nr\_linii number := 0; -- numarul de iteratii ale cursorului c

salariu\_inexistent exception; -- exceptie pentru cazul in care nu exista salariu din intervalul [a, b]

begin

open c(a, b); -- Deschidem cursorul c pentru parametri a si b

loop

fetch c into nume, prenume, salariu, fac; -- Asignam valorile linie curente variabilelor corespunzatoare

exit when c%notfound; -- Folosim atributul notfound pentru a verifica daca am parcurs toate liniile tabelului. In caz afirmativ iesim din bucla

nr\_linii := nr\_linii + 1; -- La fiecare iteratie a cursorului, incrementam variabila nr\_linii

end loop;

close c; -- Dupa ce am iterat prin toate liniile tabelului inchidem cursorul

if nr\_linii = 0 then -- Daca cursorul c nu a gasit nicio linie din tabel, inseamna ca nu exista un salariu din intervalul [a, b]

raise salariu\_inexistent; -- Prin urmare, se arunca exceptia "salariu\_inexistent"

end if;

-- Daca nu s-a aruncat exceptia "salariu\_inexistent"

open c(a, b); -- Putem incepe afisarea tuturor angajatilor cu salariul din intervalul [a, b]

loop

fetch c into nume, prenume, salariu, fac; -- Extragem informatiile angajatului curent

exit when c%notfound; -- atribut notfound

dbms\_output.put\_line('Nume Angajat: ' || nume || ' ' || prenume); -- Afisam informatiile angajatului

dbms\_output.put\_line('Salariu: ' || salariu);

dbms\_output.put\_line('Facultate: ' || fac);

dbms\_output.put('Colegi: ');

-- Obtinem lista colegilor angajatului curent folosindu-ne de un ciclu cursor in bucla for

for i in (

select nume, prenume

from angajat join facultate using(cod\_facultate) -- ciclu cursor

where denumire = fac

) loop

dbms\_output.put(i.nume || ' ' || i.prenume || ', '); -- Afisam lista tuturor colegilor angajatului curent

end loop;

dbms\_output.new\_line; -- Afisam new\_line pentru a se afisa lista anterior mentionata

dbms\_output.new\_line; -- Afisam inca 2 new\_line-uri pentru spatiere

dbms\_output.new\_line;

end loop;

close c; -- Inchidem cursorul c

exception

-- nu intalnim cazul in care sa avem eroarea no\_data\_found

-- intrucat in cazul cursoarelor, chiar daca nu se gaseste nicio linie in urma

-- select-urilor, cursoarele raman valide, dar goale

-- in cazul care cursoarelor goale, avem exceptia "salariu\_inexistent"

-- when no\_data\_found then dbms\_output.put\_line('Nu au fost gasite date in baza de date');

-- nu intalnim cazul in care sa avem eroarea too\_many\_rows

-- intrucat toate select-urile facute sunt pentru cursoare

-- when too\_many\_rows then dbms\_output.put\_line('Too Many Rows');

when salariu\_inexistent then dbms\_output.put\_line('Nu exista niciun salariu din intervalul [' || a || ', ' || b || ']');

when others then dbms\_output.put\_line('Alt tip de eroare!');

end exercitiul\_7;

/

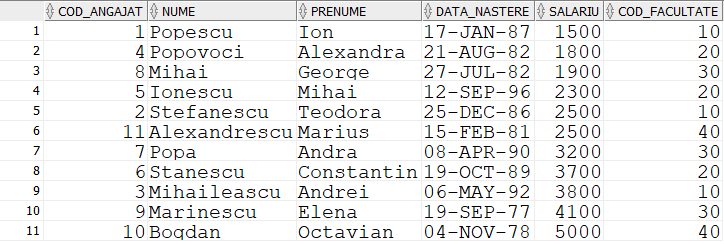
**-- Rezultate/Explicatii:**

/\*

Pentru a vedea mai bine cum se executa programul, vom selecta tabelul tuturor angajatilor si vom sorta dupa salariu.

\*/

select \* from angajat order by salariu;



/\*

De asemenea, vom selecta informatiile din tabelul facultate.

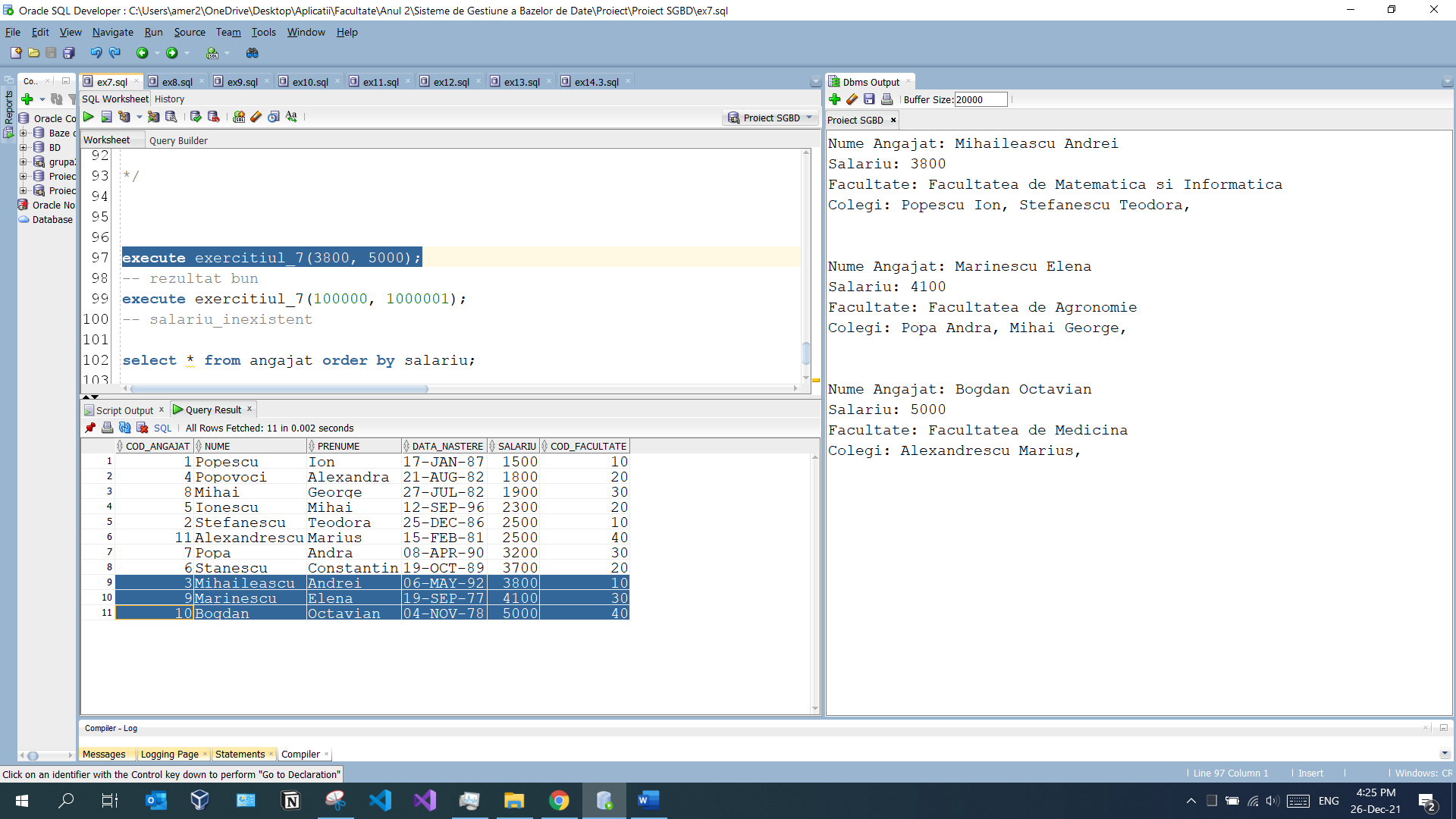
\*/

select \* from facultate;



execute exercitiul\_7(3800, 5000);

-- rezultat bun



/\*

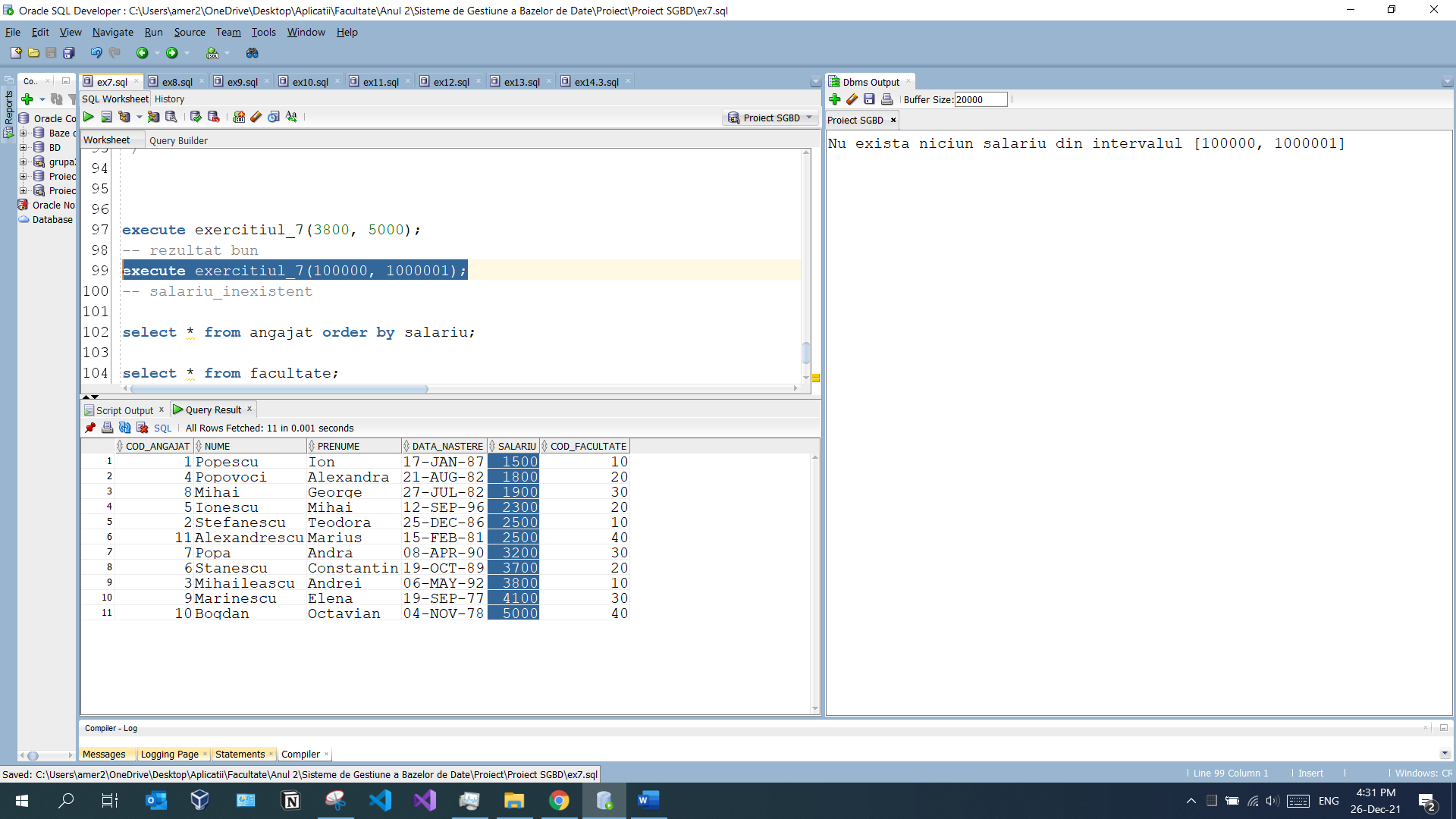
Asa cum reiese din tabelul angajatilor sortati dupa salariu, pentru intervalul salarial [3800, 5000] avem ultimii 3 angajati din tabel, anume: Mihailescu Andrei, Marinescu Elena, Bogdan Octavian.

* Mihailescu Andrei lucreaza la facultatea cu codul 10, anume Facultatea de Matematica si Informatica, alaturi de el lucreaza si Popescu Ion si Stefanescu Teodora.
* Marinescu Elena lucreaza la facultatea cu codul 30, anume Facultatea de Agronomie, alaturi de ea lucreaza Mihai George si Popa Andra.
* Bogdan Octavian lucreaza la facultatea cu codul 40, anume Facultatea de Medicina, alaturi de el lucreaza Alexandrescu Marius.

\*/

execute exercitiul\_7(100000, 1000001);

-- salariu\_inexistent



/\*

Asa cum rezulta din tabelul angajatilor, nu exista salarii din intervalul [100000, 100001]. Prin urmare, se intra pe exceptia “salariu\_inexistent” si se afiseaza un mesaj sugestiv.

\*/

**8.**

/\*

Pentru prenumele si numele unui student date ca parametru se va returna numele tarii in care se afla facultatea la care este inscris studentul.

In cazul in care studentul este inscris la mai multe facultati, se va alege facultatea cu ranking-ul mai mare.

De asemenea, se vor afisa in dbms\_output numele specializarii, numele caminului in care este cazat studentul precum si numele tarii in care se afla facultatea.

Propozitia afisata va avea forma:

Studentul <<nume>> <<prenume>> urmeaza specializarea <<x>> dintr-o facultate din <<y>>,

fiind cazat in caminul <<z>>.

\*/

**-- Cod Exercitiul 8:**

create or replace function exercitiul\_8(

nume\_student in student.nume%type, -- Numele studentului dat ca parametru de intrare de tip IN

prenume\_student in student.prenume%type -- Prenumele studentului dat ca parametru de intrare de tip IN

)

return tara.nume%type -- Tipul de data returnat este numele tarii in care se afla facultatea de ranking maxim a studentului dat ca parametru

is

cod student.cod\_student%type; -- Codul studentului dat ca parametru

nume\_tara tara.nume%type; -- Numele tarii in care afla facultatea studentului

nume\_specializare specializare.denumire%type; -- Numele specializarii la care este inscris studentul

nume\_camin camin.denumire%type; -- Numele caminului studentului

begin

-- putem avea erorile no\_data\_found sau too\_many\_rows

-- no\_data\_found atunci cand nu exista un student cu numele si prenumele dat

-- too\_many\_rows atunci cand avem cel putin doi studenti cu nume si prenume identice

select cod\_student -- selectam codul studentului cu numele si prenumele date

into cod

from student

where initcap(nume) = initcap(nume\_student) and initcap(prenume) = initcap(prenume\_student);

-- Selectarea se face fara case sensitivity, numele studentului putand fii scris cu numere mici sau mari.

-- Daca nu s-a aruncat nicio exceptie, atunci putem cauta numele caminului, numele specializarii si numele tarii

-- Din moment ce selectam informatii din minim 3 tabele diferite,

-- avem de facut join-uri care sa conecteze informatiile astfel incat

-- sa se pastreze corectitudinea datelor.

select c.denumire "Camin", sp.denumire "Specializare", t.nume "Tara"

into nume\_camin, nume\_specializare, nume\_tara

from student s

join camin c on (s.cod\_camin = c.cod\_camin)

join specializare sp on (s.cod\_specializare = sp.cod\_specializare)

join studiaza st on (s.cod\_student = st.cod\_student)

join facultate f on (f.cod\_facultate = st.cod\_facultate)

join locatie l on (l.cod\_locatie = f.cod\_locatie)

join tara t on (l.cod\_tara = t.cod\_tara)

where f.ranking = (

select min(ranking) -- cu cat rankingul este mai mic cu atat facultatea este mai de top

from student s2

join studiaza st2 on (s2.cod\_student = st2.cod\_student)

join facultate f2 on (f2.cod\_facultate = st2.cod\_facultate)

where s2.cod\_student = cod -- Selectam ranking-ul maxim dintre toate facultatiile urmate de studentul dat ca parametru

) and s.cod\_student = cod; -- Selectam detaliile despre facultatea al carei ranking este egal cu ranking-ul maxim dintre toate facultatile urmate de studentul dat ca parametru

-- Afisam detaliile obtinute

dbms\_output.put('Studentul ' || nume\_student || ' ' || prenume\_student);

dbms\_output.put(' urmeaza specializarea ' || nume\_specializare);

dbms\_output.put(' dintr-o facultate din ' || nume\_tara || ',');

if nume\_camin = null then -- Daca studentul nu este cazat la vreun camin, afisam un mesaj sugestiv

dbms\_output.put(' studentul nefiind cazat la vreun camin.');

else

dbms\_output.put(' fiind cazat in caminul ' || nume\_camin || '.');

end if;

dbms\_output.new\_line; -- Pentru dbms\_output.put();

return(nume\_tara); -- Functia returneaza numele tarii in care se afla facultatea de ranking maxim urmata de student

exception

when no\_data\_found then -- Atunci cand nu exista studentul dat ca parametru in baza de date

dbms\_output.put\_line('Nu exista niciun student cu numele de "' || nume\_student || ' ' || prenume\_student || '"');

return(''); -- Returnam sirul vid, intrucat nu putem afla tara in care se afla facultatea

when too\_many\_rows then -- Exista mai multi studenti cu acelasi nume si prenume

dbms\_output.put\_line('Exista mai multi studenti cu numele de "' || nume\_student || ' ' || prenume\_student || '"');

return('');

when others then -- Orice alt tip de eroare

dbms\_output.put\_line('Alt tip de eroare!');

return('');

end exercitiul\_8;

/

**-- Rezultate/Explicatii:**

declare

a tara.nume%type;

begin

a := exercitiul\_8('Mircea', 'Bravo');

a := exercitiul\_8('Ionescu', 'Stefan');

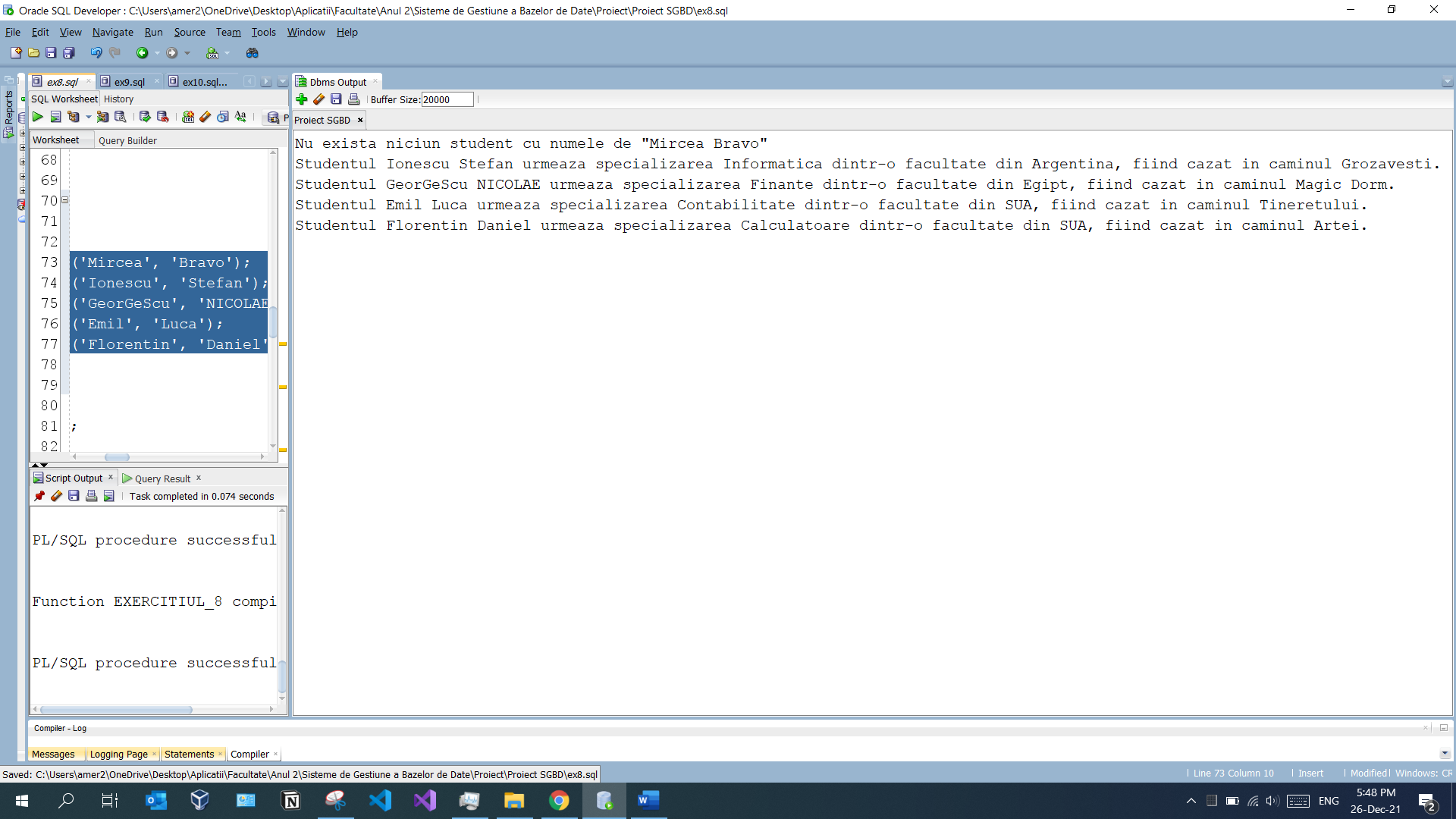
a := exercitiul\_8('GeorGeScu', 'NICOLAE');

a := exercitiul\_8('Emil', 'Luca');

a := exercitiul\_8('Florentin', 'Daniel');

end;

/



/\*

Studentul “Mircea Bravo” nu se regaseste in tabelul student, “GeorGeScu NICOLAE” este de fapt “Georgescu Nicolae” si se regaseste in tabelul student. Restul studentilor se regasesc in tabelul student si deci sunt afisati corespunzator.

\*/

**9.**

/\*

Pentru un ranking al unei facultati dat ca parametru, sa se returneze numele facultatii, impreuna cu nota cea mai mare obtinuta in cadrul facultatii, numele disciplinei, numele studentului si numele profesorului coordonator.

Daca exista mai multi studenti cu nota maxima, se va afisa doar primul in ordine alfabetica dupa nume si prenume.

\*/

**-- Cod Exercitiul 9:**

create or replace procedure exercitiul\_9(

ranking\_facultate facultate.ranking%type -- ranking-ul facultatii dat ca parametru

)

is

cod facultate.cod\_facultate%type; -- Codul facultatii

nume\_facultate facultate.denumire%type; -- Numele facultatii

nota\_maxima examinare.nota%type; -- Nota maxima obtinuta in cadrul facultatii

nume\_disciplina disciplina.denumire%type; -- Numele disciplinei asupra carei s-a obtinut nota maxima

nume\_student student.nume%type; -- Numele studentului care a obtinut nota maxima la disciplina din cadrul facultatii

nume\_profesor angajat.nume%type; -- Numele profesorului coordonator disciplinei cu nota maxima

begin

-- verificam daca codul exista

select cod\_facultate

into cod

from facultate

where ranking = ranking\_facultate;

-- Daca codul nu ar fi exista, s-ar fi generat eroare no\_data\_found si s-ar fi oprit executia procedurii

select \* -- Selectam informatiile necesare din tabele multiple folosind join-uri

into nume\_facultate, nota\_maxima, nume\_disciplina, nume\_student, nume\_profesor

from(

select f.denumire "Facultate",

e.nota "Nota",

d.denumire "Disciplina",

s.nume || ' ' || s.prenume "Student", -- Numele studentului si profesorului sunt complete

a.nume || ' ' || a.prenume "Profesor"

from student s

join participa p on (p.cod\_student = s.cod\_student)

join examinare e on (e.cod\_disciplina = p.cod\_disciplina)

join angajat a on (a.cod\_angajat = p.cod\_angajat)

join profesor pf on (a.cod\_angajat = pf.cod\_angajat)

join disciplina d on (p.cod\_disciplina = d.cod\_disciplina)

join programa pr on (d.cod\_disciplina = pr.cod\_disciplina)

join specializare sp on (pr.cod\_specializare = sp.cod\_specializare)

join facultate f on (f.cod\_facultate = sp.cod\_facultate)

where nota = ( -- Selectam toate detaliile pentru disciplina cu nota maxima

select max(nota) -- Subcerere in care electam nota maxima din cadrul facultatii date

from examinare e

join disciplina d on (e.cod\_disciplina = d.cod\_disciplina)

join programa p on (d.cod\_disciplina = p.cod\_disciplina)

join specializare sp on (sp.cod\_specializare = p.cod\_specializare)

join facultate f on (sp.cod\_facultate = f.cod\_facultate)

where f.cod\_facultate = cod

) and f.cod\_facultate = cod

order by s.nume, s.prenume -- Ordonam selectia dupa nume si prenume

)

where rownum = 1; -- In cazul in care exista mai multi studenti, selectam doar primul ordonat dupa nume si prenume

-- Afisam informatiile obtinute

dbms\_output.put\_line('Facultate: ' || nume\_facultate);

dbms\_output.put\_line('Nota: ' || nota\_maxima);

dbms\_output.put\_line('Disciplina: ' || nume\_disciplina);

dbms\_output.put\_line('Student: ' || nume\_student);

dbms\_output.put\_line('Profesor: ' || nume\_profesor);

exception

when no\_data\_found then dbms\_output.put\_line('Nu exista facultate cu ranking-ul dat');

when too\_many\_rows then dbms\_output.put\_line('Exista mai multe facultati care au ranking-ul egal cu ' || ranking\_facultate);

when others then dbms\_output.put\_line('Alt tip de eroare!');

end exercitiul\_9;

/

**-- Rezultate/Explicatii:**

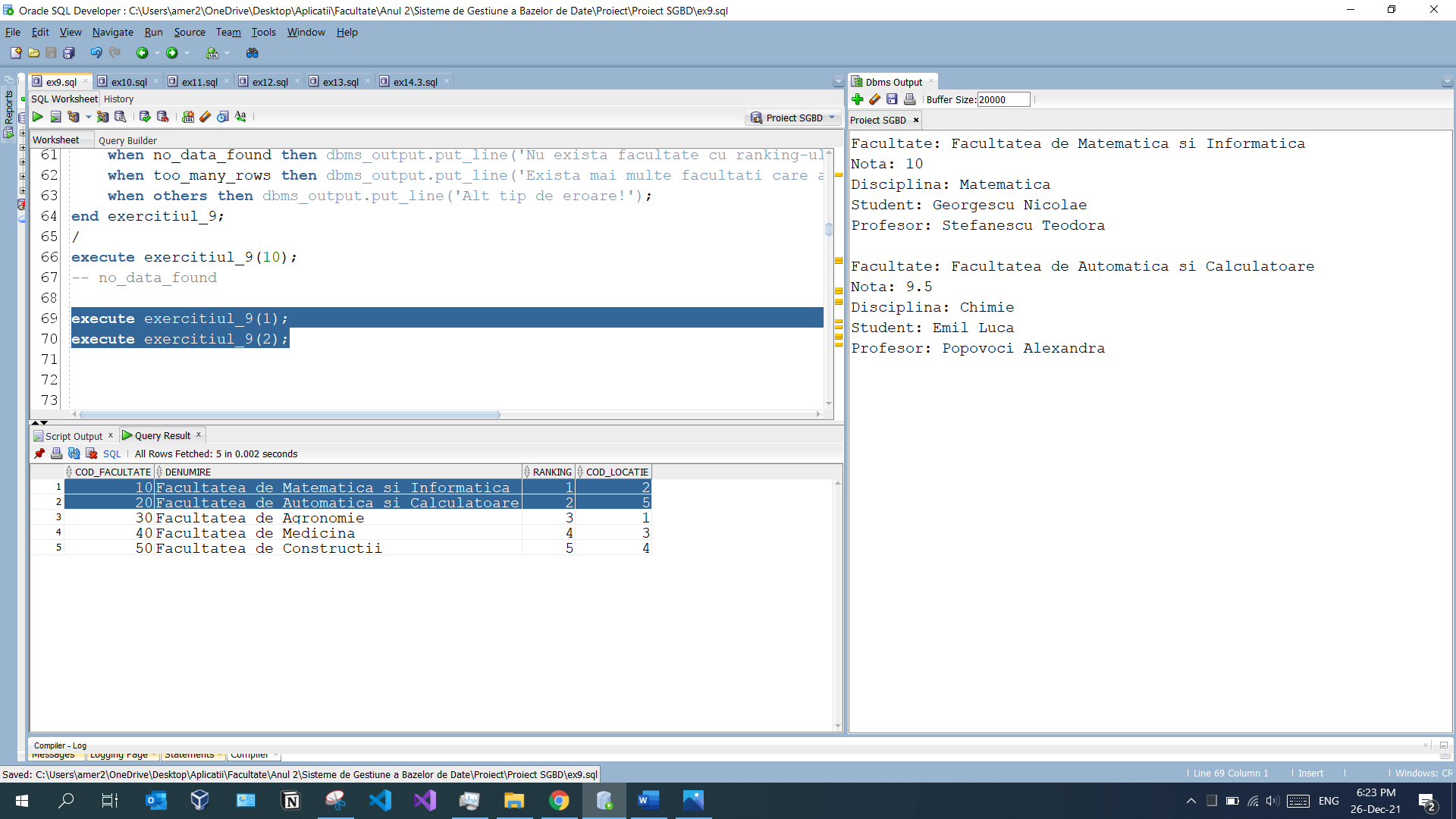
select \* from facultate;



execute exercitiul\_9(1);

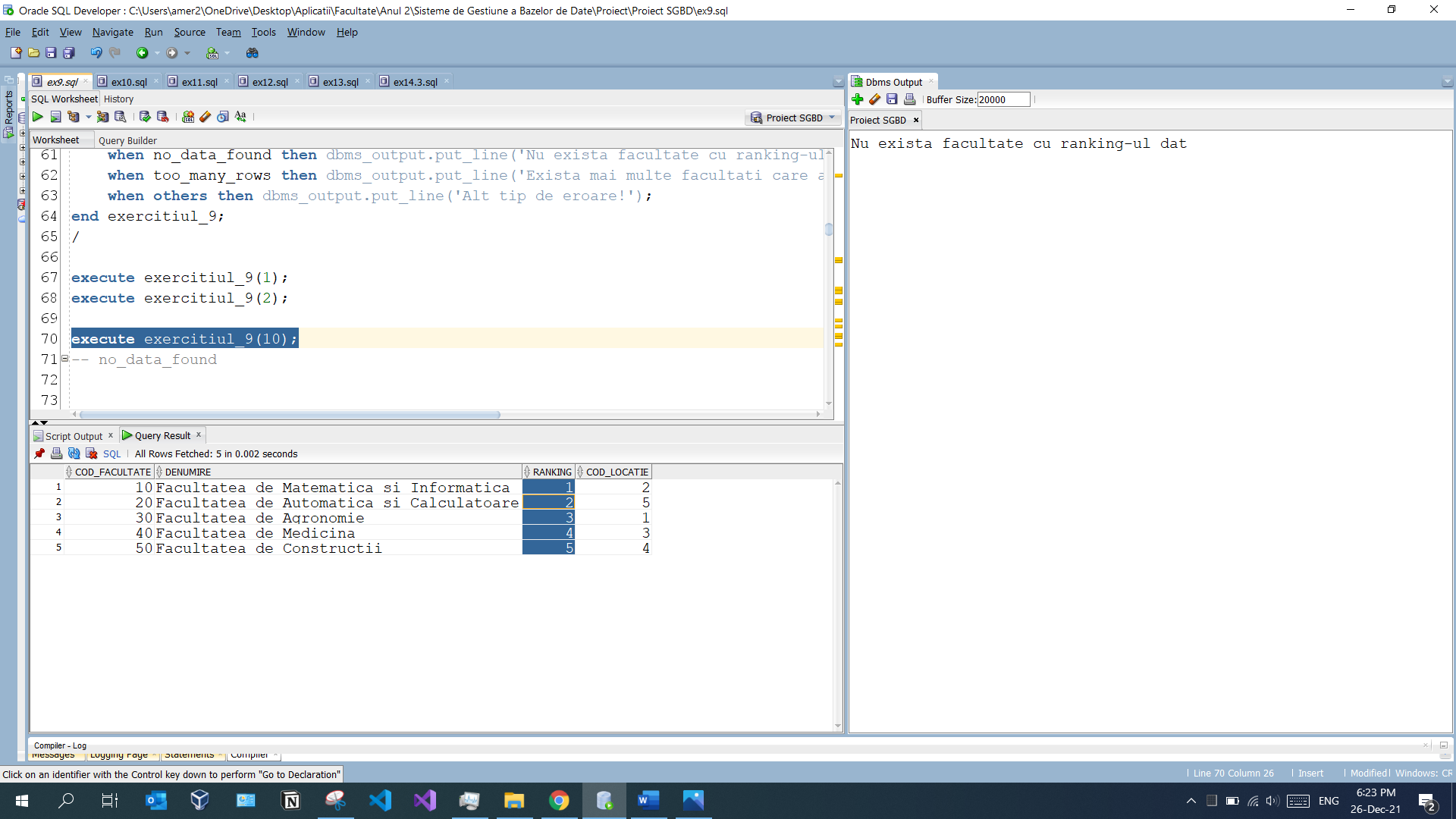
execute exercitiul\_9(2);

-- Rezultate Corecte



execute exercitiul\_9(10);

-- no\_data\_found



/\*

Nu exista facultate al carei ranking sa fie 10 si deci se intra pe exceptia no\_data\_found.

\*/

**10.**

/\*

Datorita deficitului bugetar, la inserarea unui nou angajat in tabel, sa se micsoreze salariul tuturor angajatilor cu un procent de 20% pentru a putea acomoda salariul noului angajat.

Din moment ce dorim sa ne folosim de acelasi tabel care va declansa trigger-ul vom intra in cazul de tabel mutating, caz pe care il rezolvam folosind trigger compound.

\*/

**-- Cod Exercitiul 10:**

create or replace trigger exercitiul\_10 -- Cream(sau inlocuim in cazul existentei) triggerul

for insert on angajat -- Acesta va actiona asupra inserarilor asupra tabelei angajat

compound trigger -- Trigger de tip Compound

before statement is -- La declansarea trigger-ului, inainte de inserare se va afisa un mesaj sugestiv

begin

dbms\_output.put\_line('A fost apelat trigger-ul compound');

end before statement;

after statement is -- Dupa ce s-a inserat angajatul, se va micsora salariul fiecarui angajat

begin

update angajat

set salariu = salariu - 0.2\*salariu; -- Nu exista clausa where, intrucat dorim sa modificam salariul tuturor angajatilor

end after statement;

end exercitiul\_10;

/

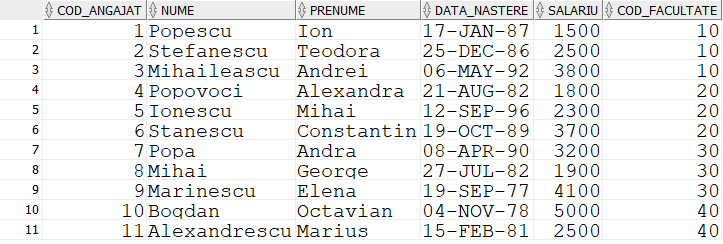
**-- Rezultate/Explicatii:**

/\*

Tabelul angajatilor inainte de orice inserare

\*/

select \* from angajat;



-- In cazul in care inca era activat trigger-ul de la exercitiul 11, il dezactivam

alter trigger nivel\_linie disable;

select avg(salariu) from angajat;

-- salariul mediu este aproximativ 2936

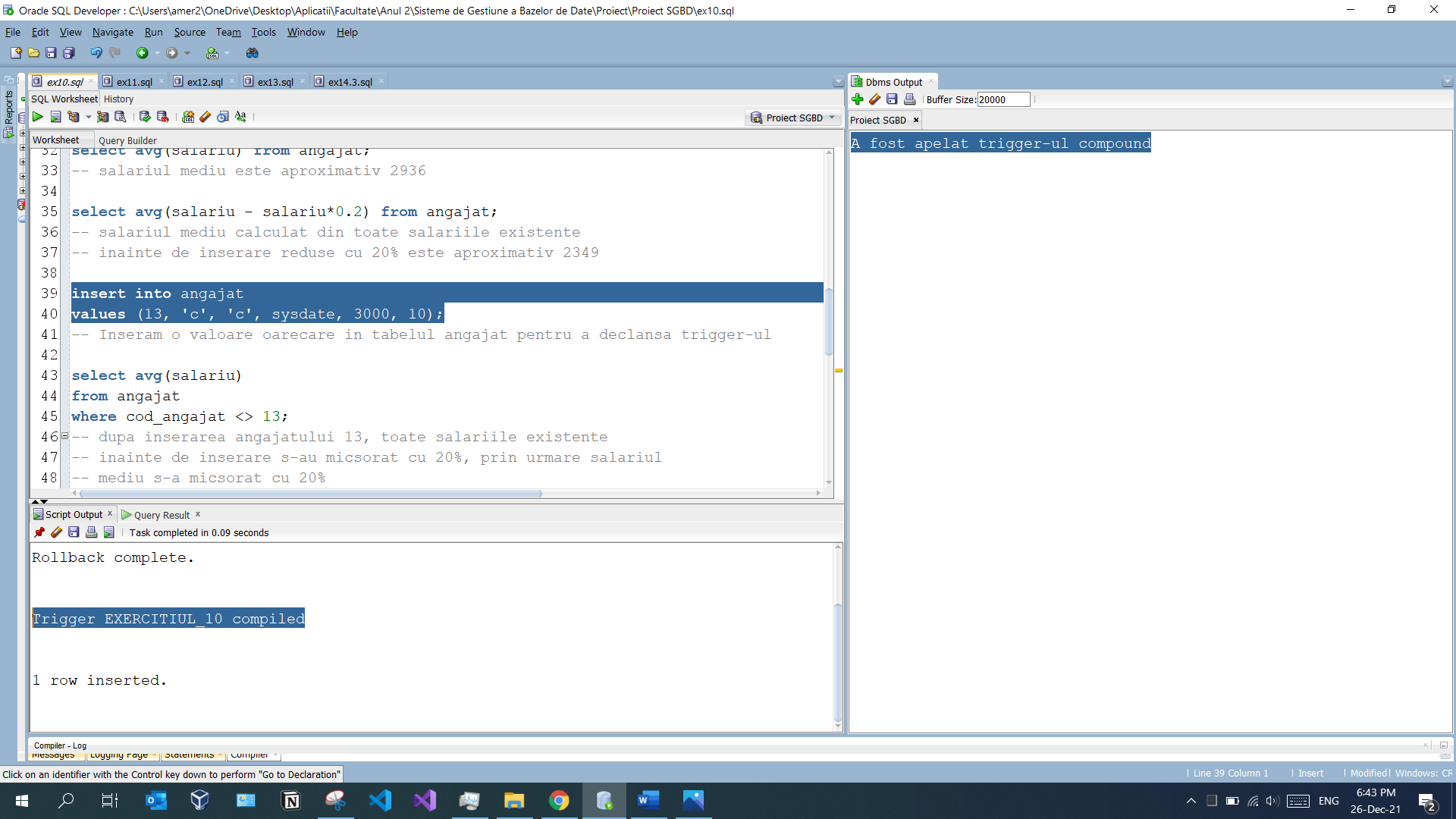
select avg(salariu - salariu\*0.2) from angajat;

-- salariul mediu calculat din toate salariile existente

-- inainte de inserare reduse cu 20% este aproximativ 2349

insert into angajat

values (13, 'c', 'c', sysdate, 3000, 10);

-- Inseram o valoare oarecare in tabelul angajat pentru a declansa trigger-ul

select avg(salariu)

from angajat

where cod\_angajat <> 13;

-- dupa inserarea angajatului 13, toate salariile existente

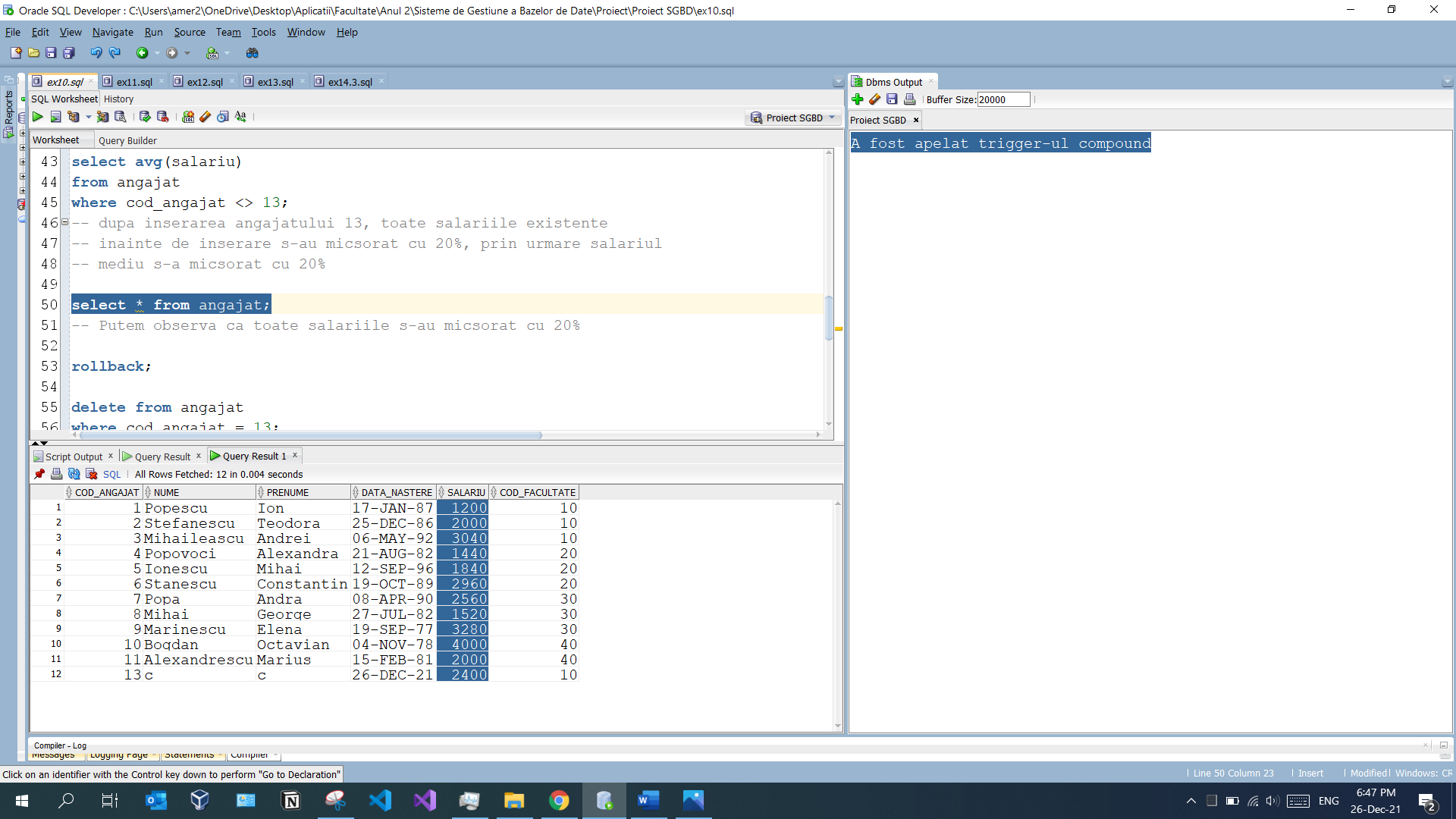
-- inainte de inserare s-au micsorat cu 20%, prin urmare salariul

-- mediu s-a micsorat cu 20%



select \* from angajat;

-- Putem observa ca toate salariile s-au micsorat cu 20%



-- Odata ce am rulat codul, putem realiza un rollback pentru

-- A modifica la loc salariile si a sterge angajatul inserat

rollback;

-- Putem face disable la trigger

alter trigger exercitiul\_10 disable;

**11.**

/\*

Daca se doreste inserarea unui nou angajat, acesta va putea fi inserat doar daca salariul lui adaugat cu salariul tuturor angajatilor nu depaseste valoarea de 37000.

Daca conditia nu este indeplinita atunci se vor genera erori diferite in felul urmator:

- Daca salariul este inserat(insert) si este mai mare decat salariul minim, se va afisa un mesaj, daca este mai mic, se va afisa alt mesaj

- Daca salariul este modificat(update) si este mai mare decat salariul mediu, se va afisa un mesaj, daca este mai mic, se va afisa alt mesaj

Salariul minim si cel mediu vor fi calculate din salariile tuturor angajatilor deja existenti in baza de date, cu exceptia salariului angajatului care declanseaza triggerul.

In orice caz, daca triggerul va fi declansat acesta va apela o eroare.

Din moment ce se va lucra pe tabel mutating se va crea un pachet cu toate informatiile necesare din tabelul respectiv. Pachetul va fi populat prin intermediul unui trigger la nivel de comanda folosind precedenta triggerilor before la nivel de comanda vs la nivel de linie

\*/

**-- Cod Exercitiul 11:**

create or replace package informatii\_angajat -- Cream pachetul ce va contine toate informatiile pe care le vrem din tabela mutating

is

salariu\_minim angajat.salariu%type; -- Salariul minim din tabelul angajat

salariu\_mediu angajat.salariu%type; -- Salariul mediu din tabelul angajat

suma\_salarii angajat.salariu%type; -- Suma salariilor din tabelul angajat

end;

/

-- Triggerul la nivel de comanda before are o precedenta mai mare fata de triggerul la nivel de linie before

-- Prin urmare, inainte de a declansa trigger-ul la nivel de linie

-- ne putem folosi de cel de la nivel de comanda pentru a extrage informatiile de care avem nevoie

-- din tabela mutating angajat

create or replace trigger nivel\_comanda -- Cream triggerul la nivel de comanda

before insert or update on angajat -- Acesta va actiona inainte de insert-urile sau update-urile pe tabela angajat

begin

select min(salariu), avg(salariu), sum(salariu) -- Selectam salariul minim, mediu si suma salariilor

into informatii\_angajat.salariu\_minim, -- In cele 3 atribute ale pachetului informatii\_angajat

informatii\_angajat.salariu\_mediu,

informatii\_angajat.suma\_salarii

from angajat;

-- where cod\_angajat <> :old.cod\_angajat;

end;

/

create or replace trigger nivel\_linie -- Cream triggerul la nivel de linie

before insert or update on angajat -- Acesta va actiona inainte de insert-urile sau update-urile pe tabela angajat

referencing new as nou -- numim variabila externa new ca fiind "nou"

for each row -- Trigger la nivel de linie

begin

if :nou.salariu + informatii\_angajat.suma\_salarii > 37000 then -- Daca salariul nou adaugat la suma salariilor este mai mare decat 37000

if inserting then -- In cazul in care salariul nou este inserat

if :nou.salariu >= informatii\_angajat.salariu\_minim then -- Daca acesta este mai mare sau egal decat salariul minim afisam un mesaj de eroare

raise\_application\_error(-20001,'Salariul inserat este mai mare decat salariul minim');

elsif :nou.salariu < informatii\_angajat.salariu\_minim then -- Altfel, afisam alt mesaj de eroare

raise\_application\_error(-20001, 'Salariul inserat este mai mic decat salariul minim');

end if;

elsif updating then -- In cazul in care salariul nou este updatat

if :nou.salariu >= informatii\_angajat.salariu\_mediu then -- Daca acesta este mai mare sau egal cu salariul mediu afisam un mesaj de eroare

raise\_application\_error(-20001, 'Salariul updatat este mai mare decat salariul mediu');

elsif :nou.salariu < informatii\_angajat.salariu\_mediu then -- Altfel, afisam alt mesaj de eroare

raise\_application\_error(-20001, 'Salariul updatat este mai mic decat salariul mediu');

end if;

end if;

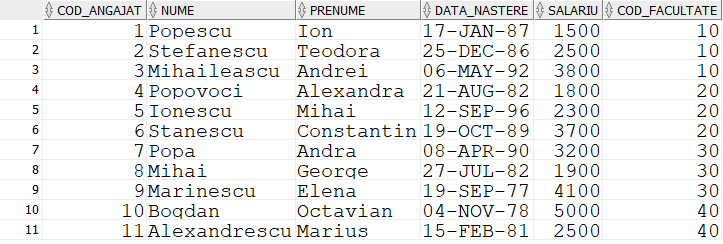
end if;

end;

/

**-- Rezultate/Explicatii:**

select \* from angajat;



select min(salariu), avg(salariu), sum(salariu)

from angajat;

-- Salariul minim: 1500

-- Salariul mediu: 2936(aproximativ)

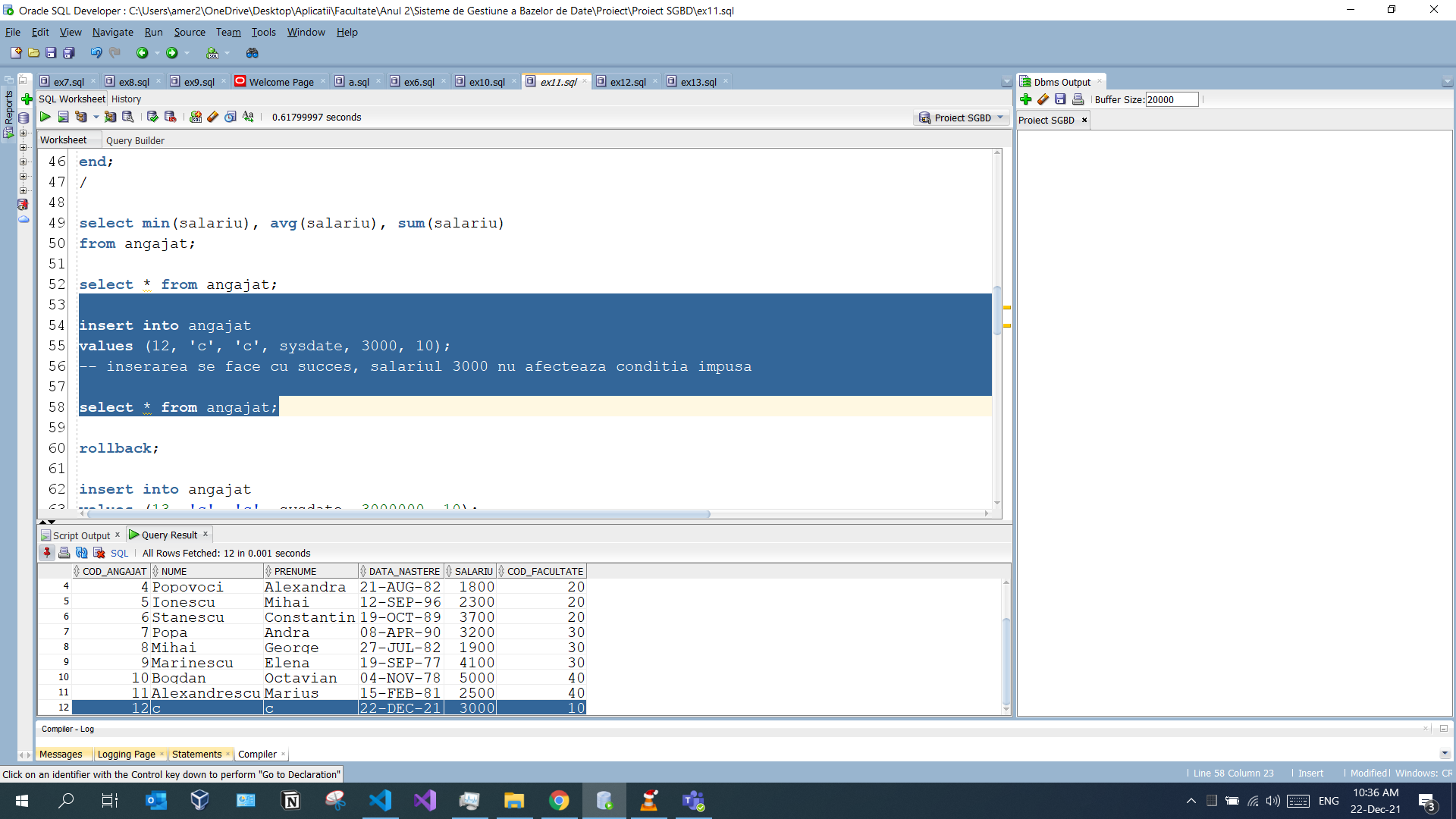
-- Suma salariilor: 32300



insert into angajat

values (12, 'c', 'c', sysdate, 3000, 10);

-- inserarea se face cu succes, salariul 3000 nu afecteaza conditia impusa

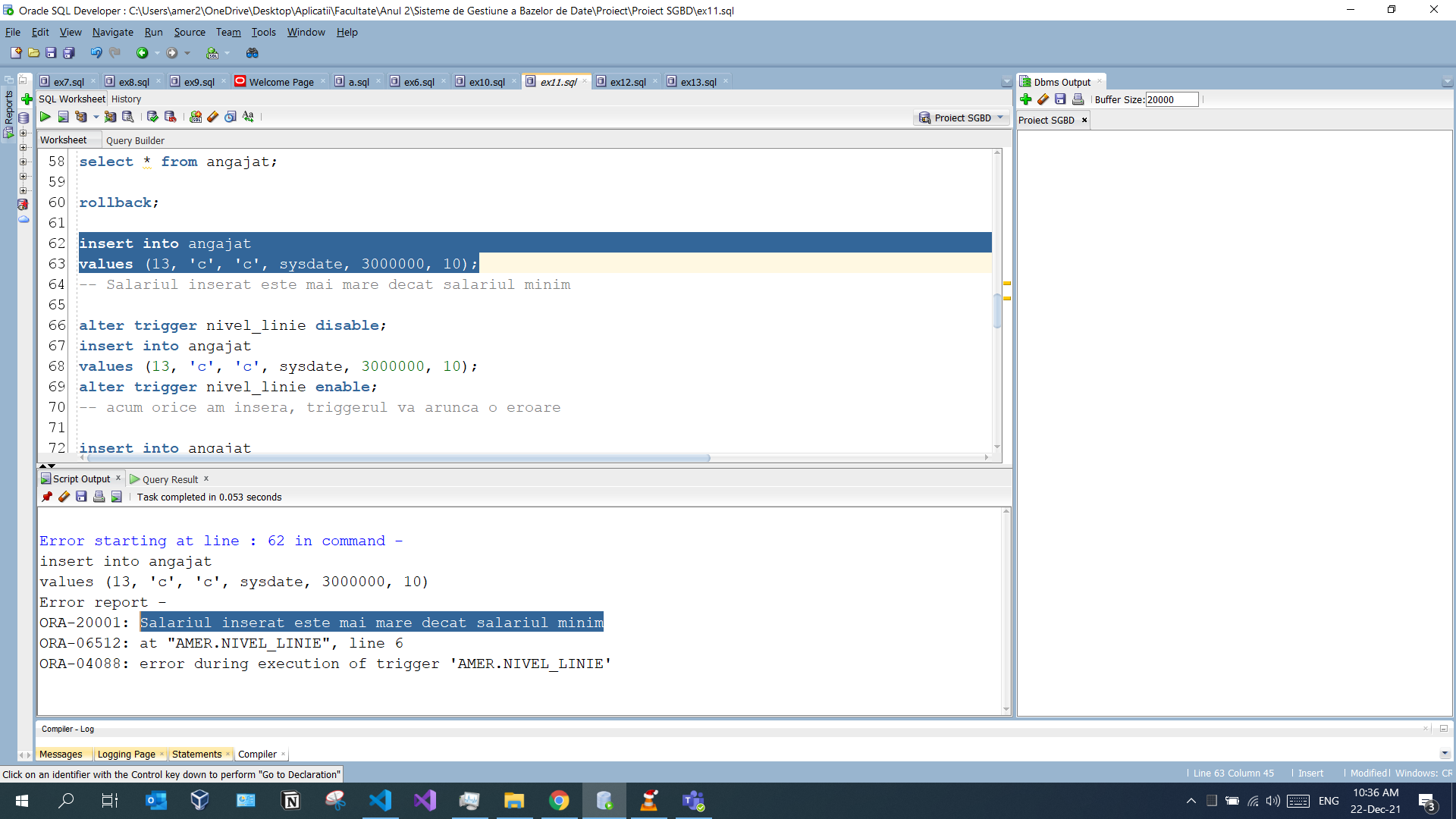


rollback;

insert into angajat

values (13, 'c', 'c', sysdate, 3000000, 10);

-- Salariul inserat este mai mare decat salariul minim



-- Nu este nevoie sa folosim rollback, intrucat ultima inserare a declansat trigger-ul

alter trigger nivel\_linie disable;

insert into angajat

values (13, 'c', 'c', sysdate, 3000000, 10);

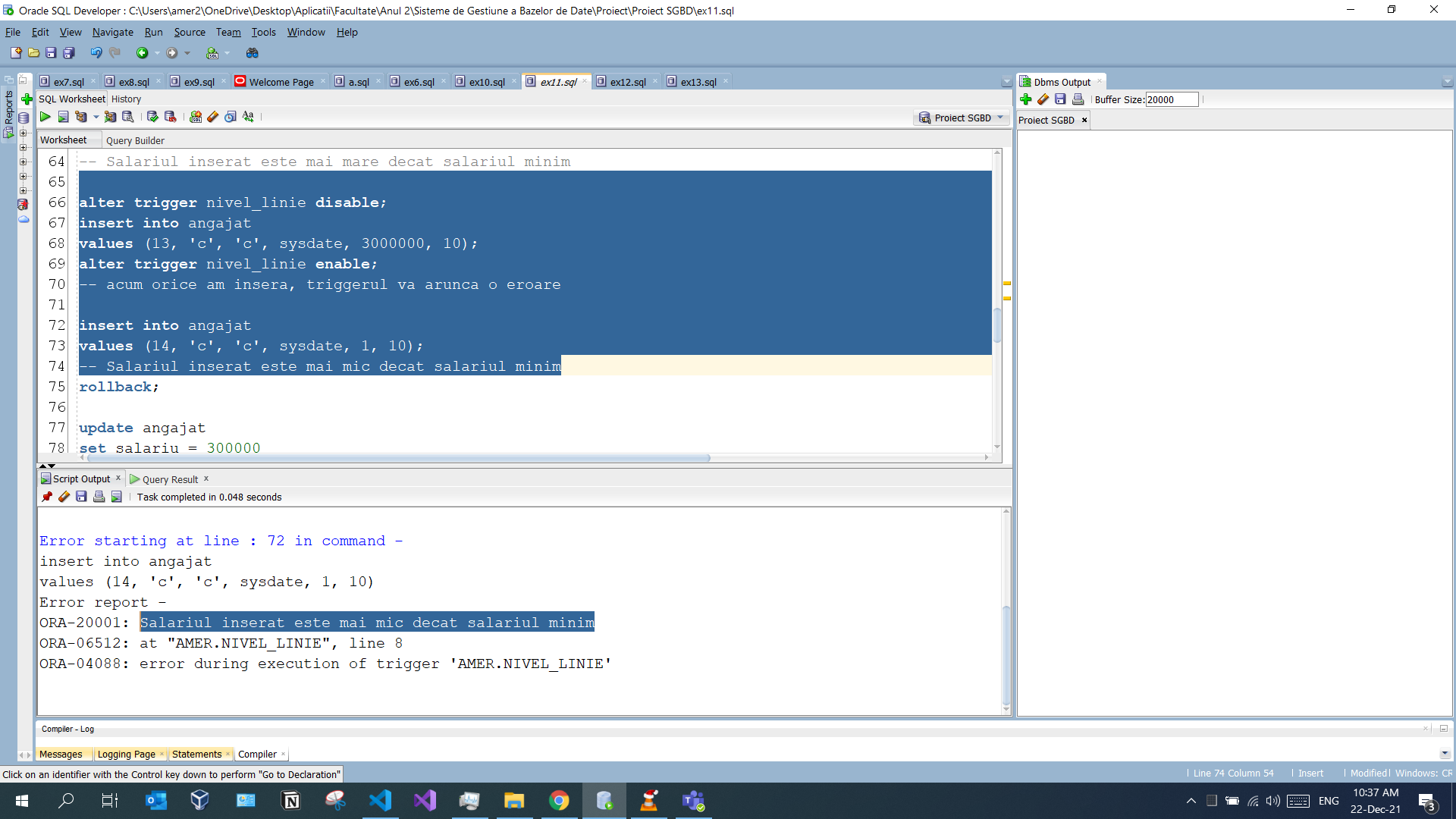
alter trigger nivel\_linie enable;

-- acum orice am insera, triggerul va arunca o eroare

insert into angajat

values (14, 'c', 'c', sysdate, 1, 10);

-- Salariul inserat este mai mic decat salariul minim



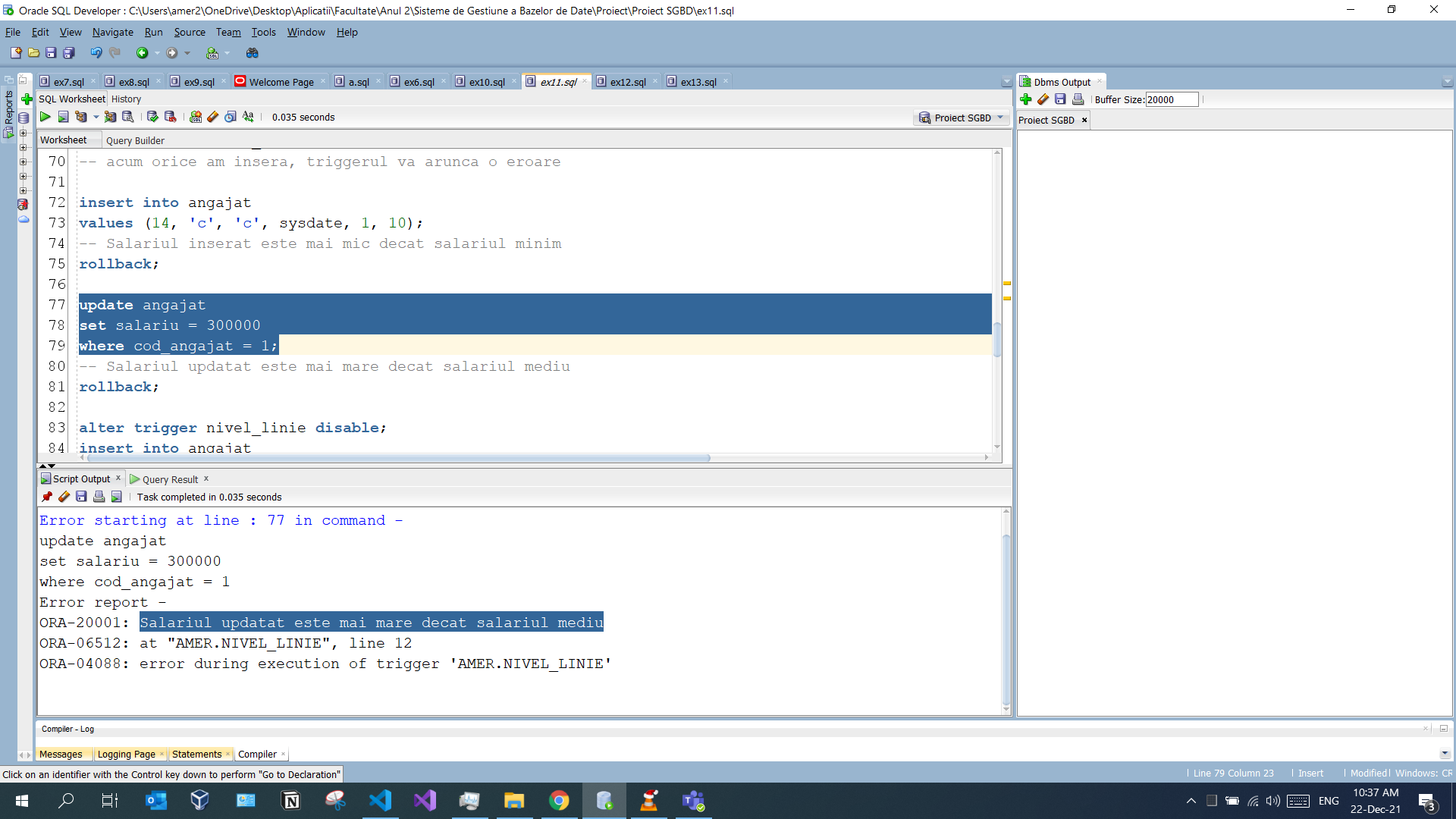
rollback;

update angajat

set salariu = 300000

where cod\_angajat = 1;

-- Salariul updatat este mai mare decat salariul mediu



rollback;

alter trigger nivel\_linie disable;

insert into angajat

values (13, 'c', 'c', sysdate, 3000000, 10);

alter trigger nivel\_linie enable;

-- acum orice am modifica, triggerul va arunca o eroare

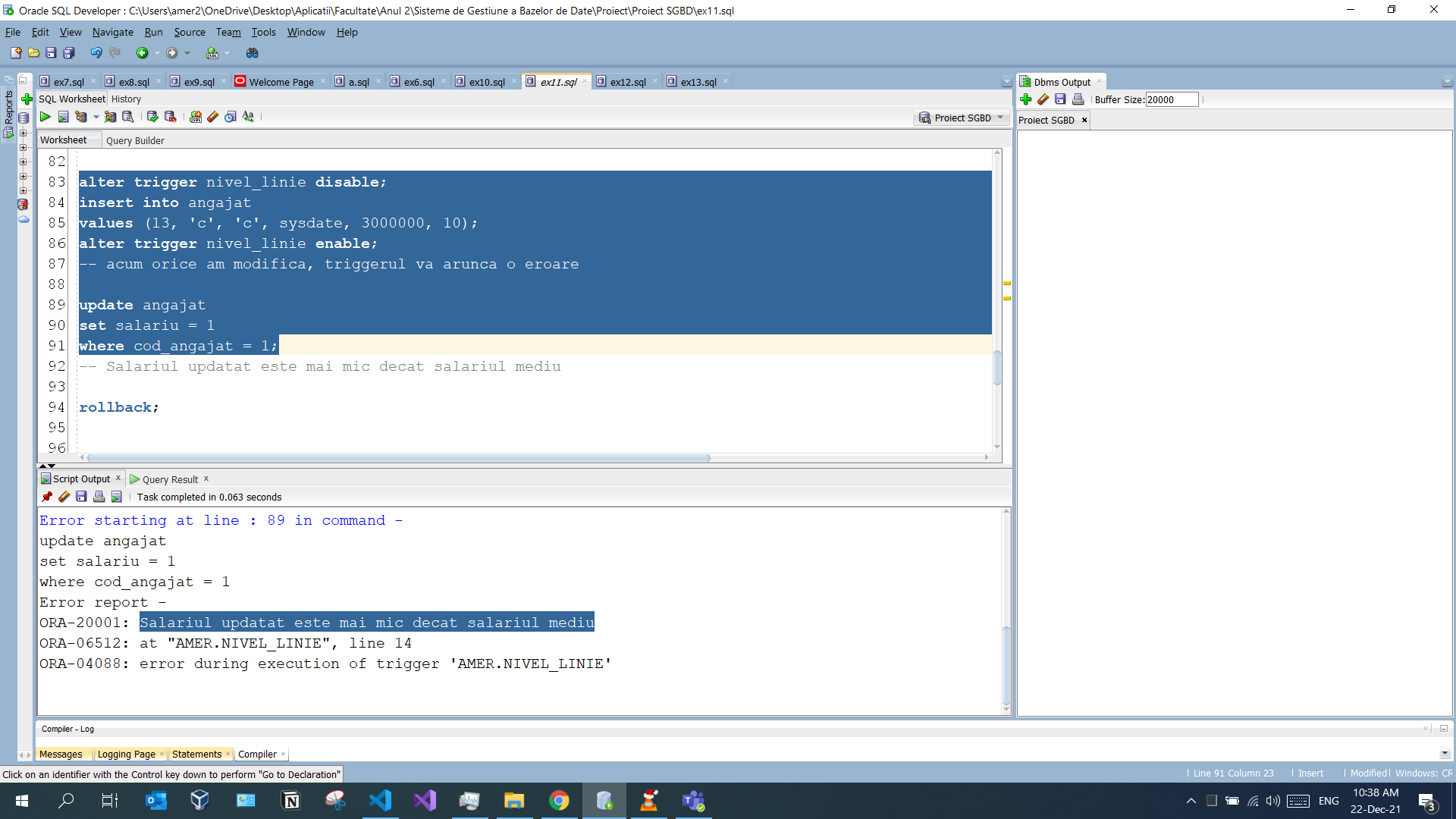
update angajat

set salariu = 1

where cod\_angajat = 1;

-- Salariul updatat este mai mic decat salariul mediu

rollback;



alter trigger nivel\_linie disable;

**12.**

/\*

Definiti un declansator care sa introduca date intr-un tabel creat

dupa ce utilizatorul curent a folosit o comanda LDD

(declansator sistem - la nivel de schema)

\*/

**-- Cod Exercitiul 12:**

create table informatii\_tabel( -- Cream tabelul in care o sa retinem informatiile despre utilizator

baza\_de\_date varchar2(50), -- Numele bazei de date

user\_logat varchar2(30), -- Numele utilizatorului

eveniment\_sistem varchar2(20), -- Tipul comenzii(CREATE, ALTER, DROP)

tip\_obiect varchar2(30), -- Tipul obiectului asupra caruia a fost aplicata instructiunea(TABLE, INDEX)

nume\_obiect varchar2(30), -- Numele obiectului

data timestamp(3) -- Data la care a fost executata instructiunea/comanda

);

create or replace trigger exercitiul\_12 -- Cream trigger-ul LDD

after create or drop or alter on schema -- Acesta va actiona dupa comenzile create, alter sau drop

begin

insert into informatii\_tabel -- Inseram in tabelul definit anterior

values (sys.database\_name, -- Toate valorile generate de sistem

sys.login\_user,

sys.sysevent,

sys.dictionary\_obj\_type,

sys.dictionary\_obj\_name,

systimestamp(3)

);

end;

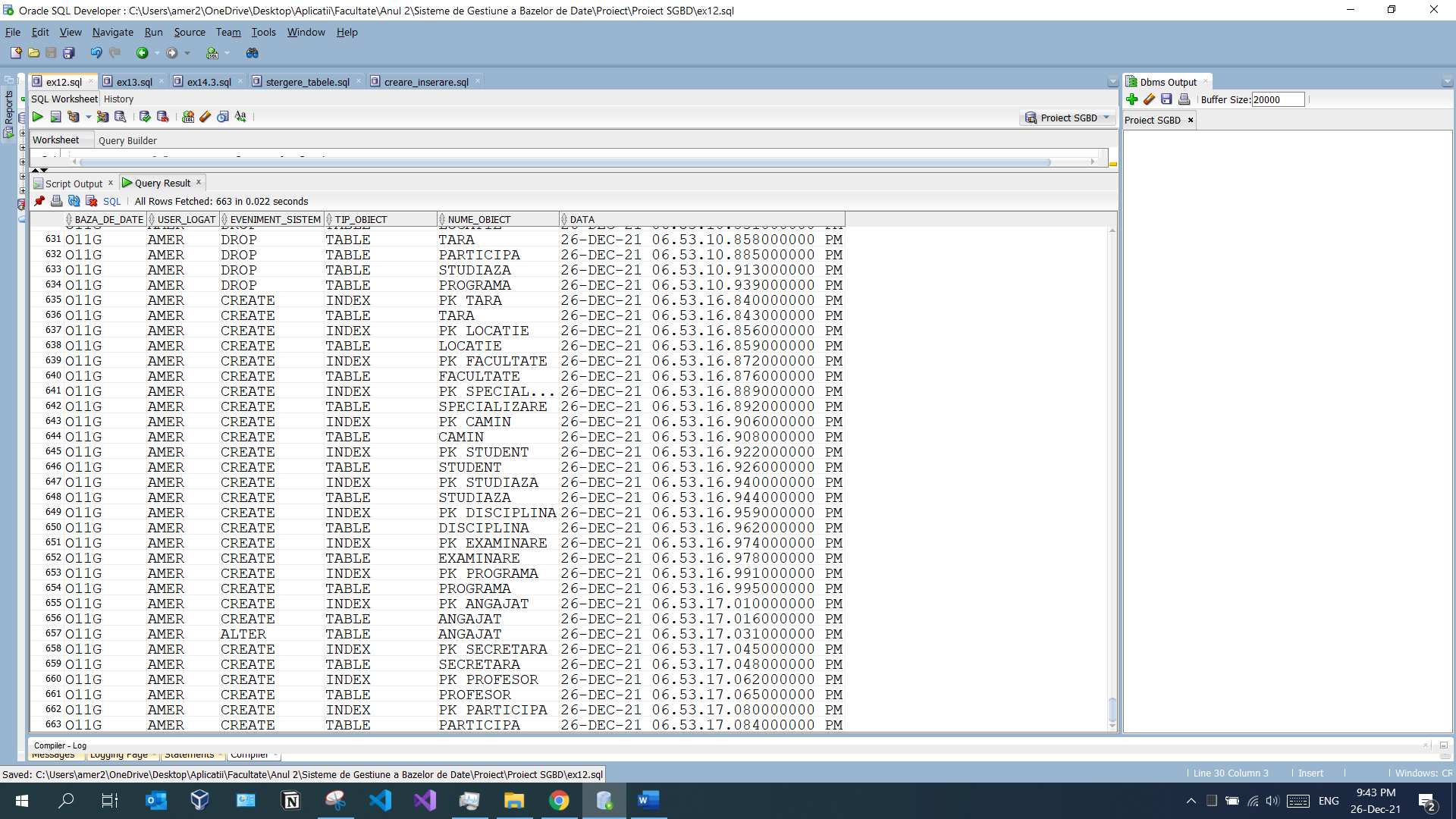
/

**-- Rezultate/Explicatii:**

/\*

Poza atasata mai jos prezinta doar cateva dintre comenzile pe care le-am rulat dupa ce am trecut de exercitiul 12, comenzi rulate in scopul realizarii/imbunatatirii celorlalte exercitii din cadrul proiectului. In total, sunt peste 600 de astfel de comenz

\*/



/\*

Exemple de comenzi rulate imediat dupa crearea tabelului informatii\_tabel:

\*/

create table exemplu\_tabel (

col\_1 number(2)

);

alter table exemplu\_tabel

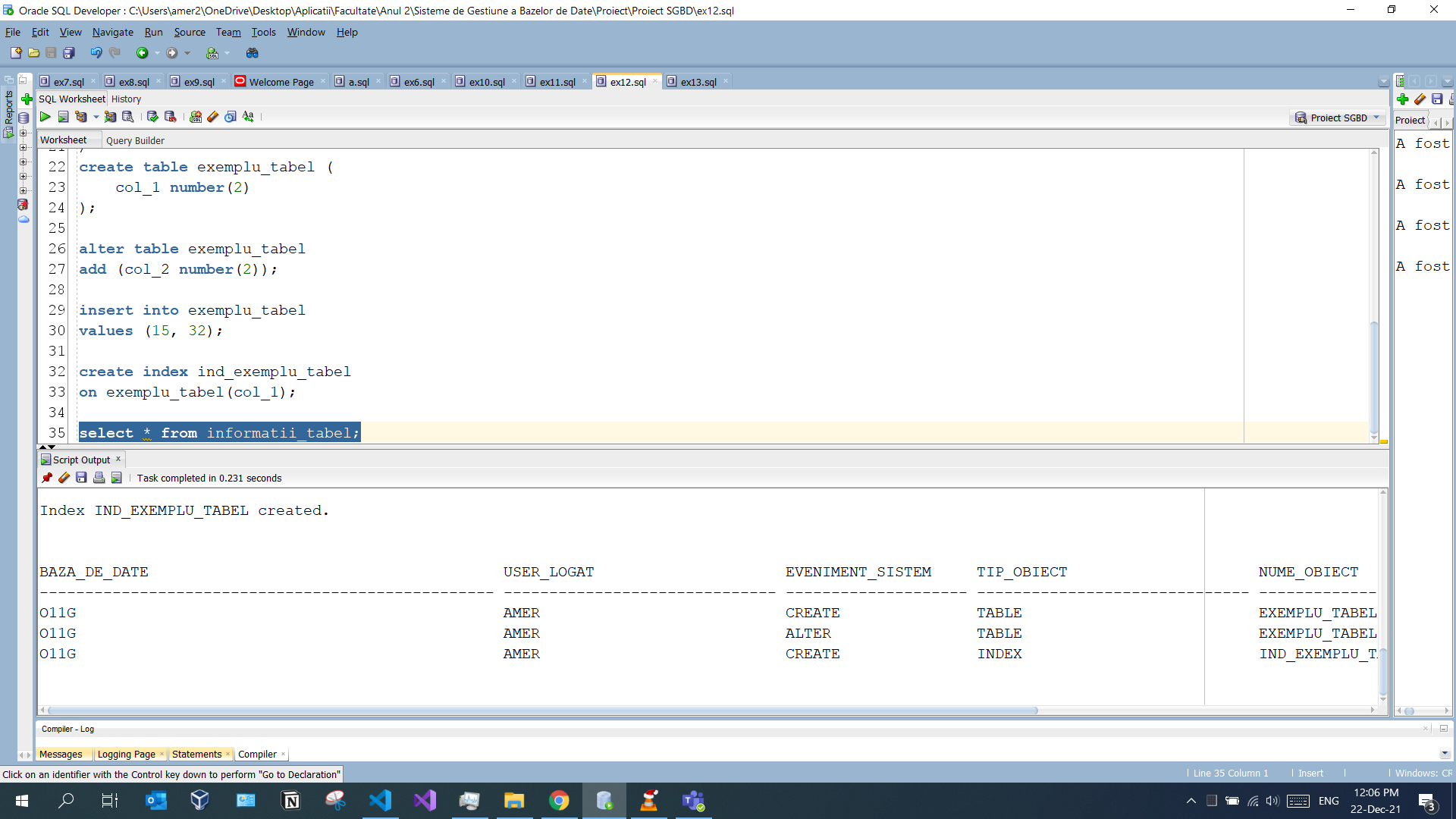
add (col\_2 number(2));

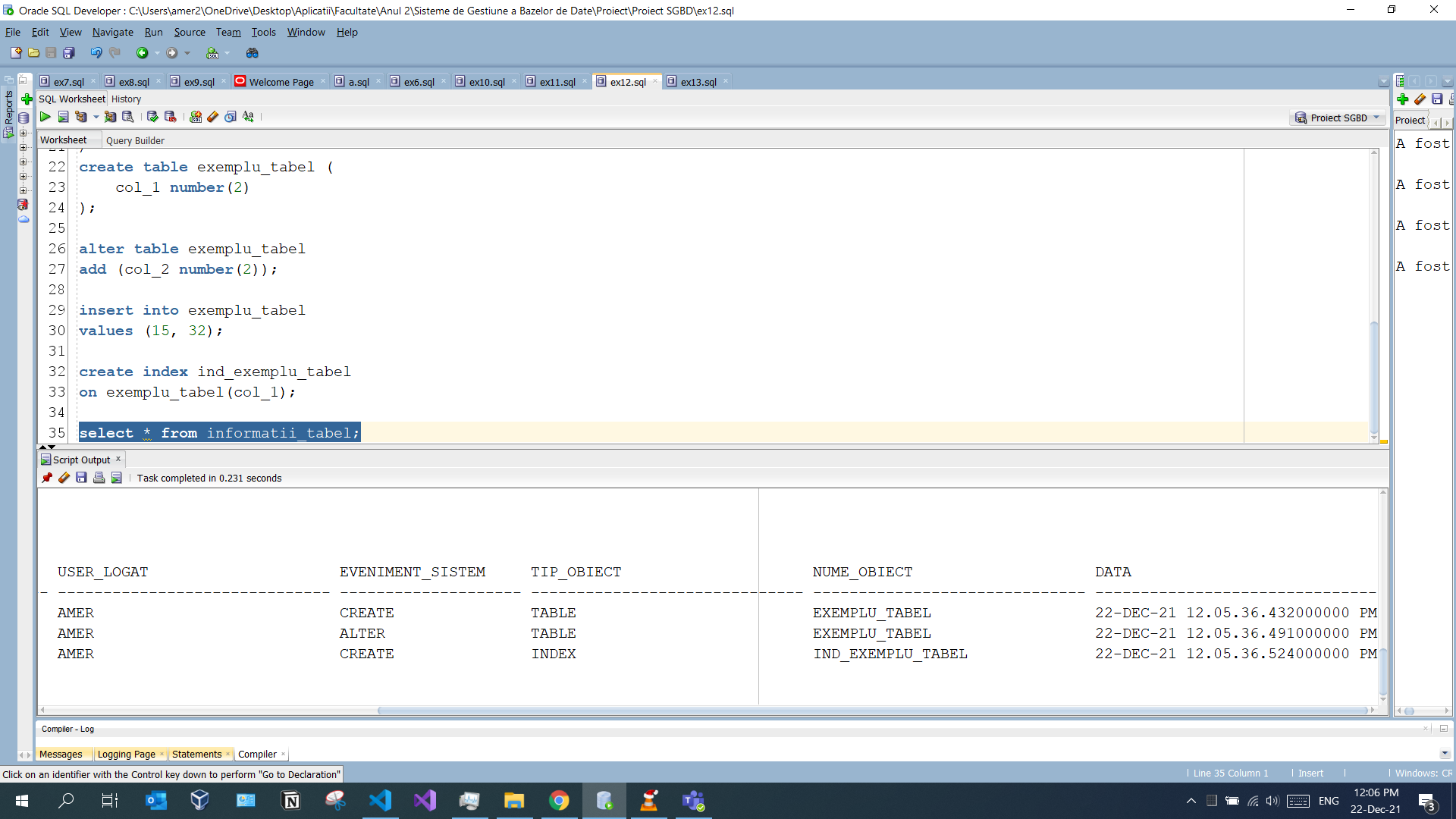
insert into exemplu\_tabel

values (15, 32);

create index ind\_exemplu\_tabel

on exemplu\_tabel(col\_1);





/\*

Observam ca doar comenzile de create si alter au fost inserate in tabel, nu si comanda de insert.

\*/

**13.**

-- Pachet ce contine subprogramele exercitiilor 6-9.

**-- Cod Exercitiul 13:**

create or replace package exercitiul\_13

is

procedure exercitiul\_6(id student.cod\_student%type);

procedure exercitiul\_7(a angajat.salariu%type, b angajat.salariu%type);

function exercitiul\_8(

nume\_student in student.nume%type,

prenume\_student in student.prenume%type

)

return tara.nume%type;

procedure exercitiul\_9(ranking\_facultate facultate.ranking%type);

end exercitiul\_13;

/

create or replace package body exercitiul\_13

is

procedure exercitiul\_6(

id student.cod\_student%type -- codul studentului dat ca parametru

)

is

cursor c is select rownum, cod\_student, nume, prenume from student; -- cursor pentru a parcurge liniile din tabelul student

nr\_linie\_id number; -- numarul liniei parcurse de cursor

id\_student student.cod\_student%type; -- id-ul studentului parcurs de cursor

nume student.nume%type; -- numele studentului parcurs de cursor

prenume student.prenume%type; -- prenumele studentului parcurs de cursor

id\_maxim student.cod\_student%type; -- id-ul de valoare maxima a studentilor aflati pe liniile urmatoare id-ului de student dat ca parametru

type tab\_ind\_fibonacci is table of number index by pls\_integer; -- tablou indexat ce memoreaza numerele fibonacci mai mici +-1 decat id\_maxim

ind tab\_ind\_fibonacci; -- tabloul indexat in care o sa memoram numerele fibonacci calculate

contor number; -- contor folosit pentru a parcurge tabloul indexat anterior declarat

nr\_studenti number; -- numarul total de studenti

sem boolean; -- semafor folosit pentru eficientizarea timpului de executie

no\_fibonacci\_found exception; -- exceptie specifica exercitiului

too\_many\_fibonacci exception; -- exceptie specifica exercitiului

type detalii\_profesor is record( -- tip inregistrare(record) pentru memorarea detaliilor despre un profesor

nume angajat.nume%type,

prenume angajat.prenume%type,

salariu angajat.salariu%type

);

type v\_detalii\_profesor is varray(25) of detalii\_profesor; -- varray de 25 de detalii\_profesor

type tab\_imb\_nume\_profesori is table of v\_detalii\_profesor; -- tablou imbricat al carui elemente sunt varray-uri de detalii\_profesor(matrice)

imb tab\_imb\_nume\_profesori := tab\_imb\_nume\_profesori(); -- imb este practic o matrice de detalii\_profesor

begin

-- verificam daca id-ul dat ca parametru exista de fapt in baza de date

select cod\_student into id\_student from student where cod\_student = id;

-- generam NO\_DATA\_FOUND daca nu gasim id-ul dat ca parametru

-- Daca nu s-a generat eroarea NO\_DATA\_FOUND inseamna ca id-ul exista si il avem memorat in variabila auxiliara id\_student

open c; -- deschidem cursorul pentru a incepe parcurgerea tabelului student

loop -- cat timp nu am parcurs tot tabelul

fetch c into nr\_linie\_id, id\_student, nume, prenume; -- plasam valorile liniei curente in variabilele asociate

exit when c%notfound; -- oprim bucla atunci cand am trecut de ultima linie a tabelului

if id\_student = id then -- daca id-ul curent este egal cu id-ul dat ca parametru

exit; -- inseamna ca putem sa iesim din bucla parcurgerii cursorului

end if;

end loop;

-- am gasit linia unde se afla student-ul cu id-ul dat

-- trebuie sa calculam id-ul maxim dintre cei ramasi

id\_maxim := id\_student; -- initial, id-ul maxim va fi fix id-ul dat ca parametru

loop -- incepand cu linia imediat urmatoare liniei in care se afla id-ul dat ca parametru

exit when c%notfound; -- verificam intai daca id-ul dat ca parametru se afla deja pe ultima linie a tabelului student

fetch c into nr\_linie\_id, id\_student, nume, prenume; -- Daca nu era pe ultima linie, atunci incepem cautarea id-ului maxim

if id\_student > id\_maxim then -- Daca id-ul curent este mai mare decat id-ul maxim calculat pana in acest moment

id\_maxim := id\_student; -- Atunci noul id\_maxim devine id-ul curent

end if;

end loop;

dbms\_output.put\_line('ID-ul dat: ' || id); -- Printam id-ul dat ca parametru

dbms\_output.put\_line('ID-ul maxim: ' || id\_maxim); -- Printam id-ul maxim incepand cu linia urmatoare id-ului curent

close c;

dbms\_output.new\_line; -- Printam o noua linie

-- Acum ca avem id-ul maxim calculat, putem incepe calcularea numerelor fibonacci mai mici +-1 decat id\_maxim

if 1 <= id\_maxim + 1 then -- Daca primul termen fibonacci(1) este mai mic sau egal decat id\_maxim + 1 atunci putem incepe calcularea

ind(1) := 1; -- Primul termen fibonacci este 1

ind(2) := 1; -- Al doilea termen fibonacci este 2

contor := 3; -- Calculul incepe cu al treilea termen fibonacci, deci contor este 3

else -- Daca in schimb id\_maxim + 1 este mai mic decat primul termen fibonacci, atunci nu avem ce termeni fibonacci sa calculam(numerele fibonacci sunt strict pozitive)

raise no\_fibonacci\_found; -- Prin urmare, aruncam exceptia no\_fibonacci\_found, intrucat nu avem numere fibonacci

end if;

-- Daca nu s-a aruncat exceptia, atunci putem continua calcularea numerelor fibonacci

while ind(contor-1) + 1 <= id\_maxim loop -- Cat timp termenul anterior + 1 este mai mic sau egal decat id\_maxim calculam urmatorul termen fibonacci

if ind(contor - 1) + ind(contor - 2) > id\_maxim then -- Daca urmatorul termen fibonacci va fi mai mare strict decat id\_maxim, nu il mai calculam

exit; -- Ci in schimb iesim din bucla

end if;

ind(contor) := ind(contor-1) + ind(contor-2); -- fib(i) = fib(i-1) + fib(i-2);

contor := contor + 1; -- i++;

end loop;

dbms\_output.put\_line('Numerele fibonacci mai mici ca ' || id\_maxim || ':'); -- Incepem afisarea termenilor fibonacci calculati

for i in ind.first..ind.last loop -- Parcurgem tabloul indexat folosind o bucla for

dbms\_output.put(ind(i) || ' '); -- Printam toti termenii fibonacci intr-o singura linie separand prin spatiu

end loop;

dbms\_output.new\_line; -- Afisam 3 linii noi pentru a separa continutul afisat

dbms\_output.new\_line;

dbms\_output.new\_line;

-- Determinam numarul de studenti care se afla dupa id-ul dat, precum si numarul de studenti al caror id este fibonacci

nr\_studenti := 0; -- Initial consideram ca numarul de studenti aflati sub id-ul dat este 0

contor := 0; -- In contor vom memora numarul de id-uri fibonacci, contor este initializat cu 0

open c; -- Deschidem cursorul din nou, acesta se afla inainte de prima linie din tabel

loop -- Cautam sa ajungem la linia unde se afla id-ul dat ca parametru

fetch c into nr\_linie\_id, id\_student, nume, prenume;

exit when id\_student = id; -- Odata ce am ajuns la linia cu id-ul dat ca parametru oprim bucla

end loop;

loop -- Incepem sa numaram studentii aflati sub id-ul dat

nr\_studenti := nr\_studenti + 1; -- Luam in considerare si id-ul dat ca fiind unul elegibil

-- Testam daca id-ul curent este fibonacci

sem := true; -- Consideram sem = true atunci cand id-ul nu este fibonacci

for i in ind.first..ind.last loop -- Parcurgem tabloul de numere fibonacci

if id\_student >= ind(i) - 1 and id\_student <= ind(i) + 1 then -- Daca id-ul curent este egal +-1 fata de termenul fibonacci curent

contor := contor + 1; -- Incrementam contor-ul pentru a creste numarul de id-uri fibonacci

sem := false; -- sem devine false, intrucat stim ca id-ul este fibonacci

end if;

if sem = false then -- Daca am determinat ca id-ul este fibonacci, nu mai este nevoie sa comparam cu restul numerelor fibonacci din tablou

exit; -- Prin urmare, putem iesi din bucla for

end if;

end loop;

fetch c into nr\_linie\_id, id\_student, nume, prenume; -- Trecem la urmatorul id

exit when c%notfound; -- Repetam procesul pentru fiecare id in parte

end loop;

close c; -- Inchidem cursorul odata ce am terminat

/\*

Pana acum avem urmatoarele:

contor = numarul de studenti cu id fibonacci

nr\_studenti = numarul tuturor studentilor

\*/

-- dbms\_output.put\_line(contor);

-- dbms\_output.put\_line(nr\_studenti);

if contor = 0 then -- daca contor = 0 inseaman ca nu avem id-uri fibonacci si aruncam exceptia no\_fibonacci\_found

raise no\_fibonacci\_found;

elsif contor > 0.5 \* nr\_studenti then -- daca avem mai mult de jumatate + 1 dintre toti studentii eligibil cu id-uri fibonacci, intram pe exceptia TOO\_MANY\_FIBONACCI

raise too\_many\_fibonacci;

end if;

-- daca nu am intrat pe exceptia TOO\_MANY\_FIBONACCI, inseamna putem sa

-- afisam toti studentii care se incadreaza in conditia id-ului:

-- Practic nu mai avem nevoie de numarul de id-uri fibonacci, deci putem sa reutilizam variabila contor

open c; -- Deschidem cursorul pentru parcurgerea tabelului student

loop

fetch c into nr\_linie\_id, id\_student, nume, prenume;

exit when id\_student = id; -- Parcurgem toate liniile pana la linia cu id-ul dat ca parametru

end loop;

-- Incepem sa construim matricea de detalii\_profesor din imb

contor := 1; -- Contor ia primul indice al tabloului imb

loop

-- Pentru fiecare id de dupa id-ul parametru, verificam daca este fibonacci si afisam datele

for i in ind.first..ind.last loop

if id\_student >= ind(i) - 1 and id\_student <= ind(i) + 1 then -- Daca id-ul este id fibonacci afisam informatiile

dbms\_output.put\_line('Nr linie: ' || nr\_linie\_id);

dbms\_output.put\_line('ID: ' || id\_student);

dbms\_output.put\_line('Nume Student: ' || nume || ' ' || prenume);

dbms\_output.put('Profesori: ');

imb.extend; -- necesar tablourilor imbricate si varray-urilor

select a.nume, a.prenume, a.salariu -- Selectam toti profesorii studentului cu id-ul curent

bulk collect into imb(contor) -- Stocam toate campurile selectate in imb(contor), care este de tipul varray(25) de detalii\_profesor

from participa p join student s on (s.cod\_student = p.cod\_student)

join profesor pf on (p.cod\_angajat = pf.cod\_angajat)

join angajat a on (p.cod\_angajat = a.cod\_angajat)

where s.cod\_student = id\_student

order by s.nume;

for i in imb(contor).first..imb(contor).last loop -- Parcurgem lista tuturor profesorilor unui student

dbms\_output.put(imb(contor)(i).nume || ' ' || imb(contor)(i).prenume || ' ' || imb(contor)(i).salariu); -- Afisam numele, prenumele si salariul profesorului curent

if i < imb(contor).last then -- Daca nu am ajuns la ultimul profesor

dbms\_output.put(', '); -- Afisam virgula ca separator

else

dbms\_output.put('.'); -- Daca am ajuns la ultimul profesor, punem simbolul punct.

end if;

end loop;

contor := contor + 1; -- Incrementam contor-ul pentru a trece la urmatorul element al lui imb

dbms\_output.new\_line; -- Punem new\_line pentru a se afisa toate dbms\_output.put()-urile executate in for

dbms\_output.new\_line; -- Punem inca un new\_line pentru a pastra afisarea mai spatiata

end if;

end loop;

fetch c into nr\_linie\_id, id\_student, nume, prenume; -- Trecem la urmatorul student

exit when c%notfound; -- Incheiem bucla atunci cand am terminat studentii

end loop;

close c;

ind.delete; -- stergem tabloul indexat

for i in imb.first..imb.last loop

imb(i).delete; -- Stergem fiecare varray existent in imb

end loop;

imb.delete; -- Stergem tabloul imbricat

exception

when no\_data\_found then dbms\_output.put\_line('ID-ul dat nu exista in baza de date'); -- Atunci cand avem no\_data\_found inseamna ca nu exista student cu id-ul dat ca parametru

when no\_fibonacci\_found then dbms\_output.put\_line('Niciun student nu are id fibonacci'); -- Atunci cand nu exista niciun id fibonacci sau daca id\_maxim este mai mic decat primul termen fibonacci

when too\_many\_fibonacci then dbms\_output.put\_line('Mai mult de jumatate + 1 dintre studentii eligibili au id-uri fibonacci'); -- Atunci cand mai mult de jumatate + 1 dintre id-uri sunt fibonacci primim aceasta eroare

-- nu intalnim erori de tipul too\_many\_rows, intrucat toate select-urile pe care le

-- efectuam se fac impreuna cu clauza where cod\_student = id;

-- din moment ce prin definitie cod\_student este cheie primara in tabela student

-- o sa avem cel mult un rezultat in urma select-ului.

-- when too\_many\_rows then dbms\_output.put\_line('Too Many Rows');

when others then dbms\_output.put\_line('Alt tip de eroare!'); -- Tratam orice alt tip de eroare

end exercitiul\_6;

procedure exercitiul\_7(

a angajat.salariu%type, -- Limita inferioara a intervalului salarial

b angajat.salariu%type -- Limita superioara a intervalului salarial

)

is

-- cursor explicit parametrizat

-- selecteaza codul, numele, prenumele, salariul profesorilor ca se incadreaza in intervalul salarial

-- precum si denumirea facultatii la care acesti profesori predau

cursor c(x angajat.salariu%type, y angajat.salariu%type) is

select cod\_angajat, nume, prenume, salariu, denumire

from angajat join facultate using(cod\_facultate)

where salariu between x and y;

a\_cod angajat.cod\_angajat%type; -- variabila pentru codul profesorului

a\_nume angajat.nume%type; -- variabila pentru numele profesorului

a\_prenume angajat.prenume%type; -- variabila pentru prenumele profesorului

a\_salariu angajat.salariu%type; -- variabila pentru salariul profesorului

f\_denumire facultate.denumire%type; -- variabila pentru denumirea facultatii la care preda profesorul

nr\_linii number := 0; -- numarul de iteratii ale cursorului c

salariu\_inexistent exception; -- exceptie pentru cazul in care nu exista salariu din intervalul [a, b]

begin

open c(a, b); -- Deschidem cursorul c pentru parametri a si b

loop

fetch c into a\_cod, a\_nume, a\_prenume, a\_salariu, f\_denumire; -- Asignam valorile linie curente variabilelor corespunzatoare

exit when c%notfound; -- Folosim atributul notfound pentru a verifica daca am parcurs toate liniile tabelului. In caz afirmativ iesim din bucla

nr\_linii := nr\_linii + 1; -- La fiecare iteratie a cursorului, incrementam variabila nr\_linii

end loop;

close c; -- Dupa ce am iterat prin toate liniile tabelului inchidem cursorul

if nr\_linii = 0 then -- Daca cursorul c nu a gasit nicio linie din tabel, inseamna ca nu exista un salariu din intervalul [a, b]

raise salariu\_inexistent; -- Prin urmare, se arunca exceptia "salariu\_inexistent"

end if;

-- Daca nu s-a aruncat exceptia "salariu\_inexistent"

open c(a, b); -- Putem incepe afisarea tuturor angajatilor cu salariul din intervalul [a, b]

loop

fetch c into a\_cod, a\_nume, a\_prenume, a\_salariu, f\_denumire; -- Extragem informatiile angajatului curent

exit when c%notfound; -- atribut notfound

dbms\_output.put\_line('Nume Angajat: ' || a\_nume || ' ' || a\_prenume); -- Afisam informatiile angajatului

dbms\_output.put\_line('Salariu: ' || a\_salariu);

dbms\_output.put\_line('Facultate: ' || f\_denumire);

dbms\_output.put('Colegi: ');

-- Obtinem lista colegilor angajatului curent folosindu-ne de un ciclu cursor in bucla for

for i in (

select nume, prenume

from angajat join facultate using(cod\_facultate) -- ciclu cursor

where denumire = f\_denumire and cod\_angajat <> a\_cod

) loop

dbms\_output.put(i.nume || ' ' || i.prenume || ', '); -- Afisam lista tuturor colegilor angajatului curent

end loop;

dbms\_output.new\_line; -- Afisam new\_line pentru a se afisa lista anterior mentionata

dbms\_output.new\_line; -- Afisam inca 2 new\_line-uri pentru spatiere

dbms\_output.new\_line;

end loop;

close c; -- Inchidem cursorul c

exception

-- nu intalnim cazul in care sa avem eroarea no\_data\_found

-- intrucat in cazul cursoarelor, chiar daca nu se gaseste nicio linie in urma

-- select-urilor, cursoarele raman valide, dar goale

-- in cazul care cursoarelor goale, avem exceptia "salariu\_inexistent"

-- when no\_data\_found then dbms\_output.put\_line('Nu au fost gasite date in baza de date');

-- nu intalnim cazul in care sa avem eroarea too\_many\_rows

-- intrucat toate select-urile facute sunt pentru cursoare

-- when too\_many\_rows then dbms\_output.put\_line('Too Many Rows');

when salariu\_inexistent then dbms\_output.put\_line('Nu exista niciun salariu din intervalul [' || a || ', ' || b || ']');

when others then dbms\_output.put\_line('Alt tip de eroare!');

end exercitiul\_7;

function exercitiul\_8(

nume\_student in student.nume%type, -- Numele studentului dat ca parametru de intrare de tip IN

prenume\_student in student.prenume%type -- Prenumele studentului dat ca parametru de intrare de tip IN

)

return tara.nume%type -- Tipul de data returnat este numele tarii in care se afla facultatea de ranking maxim a studentului dat ca parametru

is

cod student.cod\_student%type; -- Codul studentului dat ca parametru

nume\_tara tara.nume%type; -- Numele tarii in care afla facultatea studentului

nume\_specializare specializare.denumire%type; -- Numele specializarii la care este inscris studentul

nume\_camin camin.denumire%type; -- Numele caminului studentului

begin

-- putem avea erorile no\_data\_found sau too\_many\_rows

-- no\_data\_found atunci cand nu exista un student cu numele si prenumele dat

-- too\_many\_rows atunci cand avem cel putin doi studenti cu nume si prenume identice

select cod\_student -- selectam codul studentului cu numele si prenumele date

into cod

from student

where initcap(nume) = initcap(nume\_student) and initcap(prenume) = initcap(prenume\_student);

-- Selectarea se face fara case sensitivity, numele studentului putand fii scris cu numere mici sau mari.

-- Daca nu s-a aruncat nicio exceptie, atunci putem cauta numele caminului, numele specializarii si numele tarii

-- Din moment ce selectam informatii din minim 3 tabele diferite,

-- avem de facut join-uri care sa conecteze informatiile astfel incat

-- sa se pastreze corectitudinea datelor.

select c.denumire "Camin", sp.denumire "Specializare", t.nume "Tara"

into nume\_camin, nume\_specializare, nume\_tara

from student s

join camin c on (s.cod\_camin = c.cod\_camin)

join specializare sp on (s.cod\_specializare = sp.cod\_specializare)

join studiaza st on (s.cod\_student = st.cod\_student)

join facultate f on (f.cod\_facultate = st.cod\_facultate)

join locatie l on (l.cod\_locatie = f.cod\_locatie)

join tara t on (l.cod\_tara = t.cod\_tara)

where f.ranking = (

select min(ranking) -- cu cat rankingul este mai mic cu atat facultatea este mai de top

from student s2

join studiaza st2 on (s2.cod\_student = st2.cod\_student)

join facultate f2 on (f2.cod\_facultate = st2.cod\_facultate)

where s2.cod\_student = cod -- Selectam ranking-ul maxim dintre toate facultatiile urmate de studentul dat ca parametru

) and s.cod\_student = cod; -- Selectam detaliile despre facultatea al carei ranking este egal cu ranking-ul maxim dintre toate facultatile urmate de studentul dat ca parametru

-- Afisam detaliile obtinute

dbms\_output.put('Studentul ' || nume\_student || ' ' || prenume\_student);

dbms\_output.put(' urmeaza specializarea ' || nume\_specializare);

dbms\_output.put(' dintr-o facultate din ' || nume\_tara || ',');

if nume\_camin = null then -- Daca studentul nu este cazat la vreun camin, afisam un mesaj sugestiv

dbms\_output.put(' studentul nefiind cazat la vreun camin.');

else

dbms\_output.put(' fiind cazat in caminul ' || nume\_camin || '.');

end if;

dbms\_output.new\_line; -- Pentru dbms\_output.put();

return(nume\_tara); -- Functia returneaza numele tarii in care se afla facultatea de ranking maxim urmata de student

exception

when no\_data\_found then -- Atunci cand nu exista studentul dat ca parametru in baza de date

dbms\_output.put\_line('Nu exista niciun student cu numele de "' || nume\_student || ' ' || prenume\_student || '"');

return(''); -- Returnam sirul vid, intrucat nu putem afla tara in care se afla facultatea

when too\_many\_rows then -- Exista mai multi studenti cu acelasi nume si prenume

dbms\_output.put\_line('Exista mai multi studenti cu numele de "' || nume\_student || ' ' || prenume\_student || '"');

return('');

when others then -- Orice alt tip de eroare

dbms\_output.put\_line('Alt tip de eroare!');

return('');

end exercitiul\_8;

procedure exercitiul\_9(

ranking\_facultate facultate.ranking%type -- ranking-ul facultatii dat ca parametru

)

is

cod facultate.cod\_facultate%type; -- Codul facultatii

nume\_facultate facultate.denumire%type; -- Numele facultatii

nota\_maxima examinare.nota%type; -- Nota maxima obtinuta in cadrul facultatii

nume\_disciplina disciplina.denumire%type; -- Numele disciplinei asupra carei s-a obtinut nota maxima

nume\_student student.nume%type; -- Numele studentului care a obtinut nota maxima la disciplina din cadrul facultatii

nume\_profesor angajat.nume%type; -- Numele profesorului coordonator disciplinei cu nota maxima

begin

-- verificam daca codul exista

select cod\_facultate

into cod

from facultate

where ranking = ranking\_facultate;

-- Daca codul nu ar fi exista, s-ar fi generat eroare no\_data\_found si s-ar fi oprit executia procedurii

select \* -- Selectam informatiile necesare din tabele multiple folosind join-uri

into nume\_facultate, nota\_maxima, nume\_disciplina, nume\_student, nume\_profesor

from(

select f.denumire "Facultate",

e.nota "Nota",

d.denumire "Disciplina",

s.nume || ' ' || s.prenume "Student", -- Numele studentului si profesorului sunt complete

a.nume || ' ' || a.prenume "Profesor"

from student s

join participa p on (p.cod\_student = s.cod\_student)

join examinare e on (e.cod\_disciplina = p.cod\_disciplina)

join angajat a on (a.cod\_angajat = p.cod\_angajat)

join profesor pf on (a.cod\_angajat = pf.cod\_angajat)

join disciplina d on (p.cod\_disciplina = d.cod\_disciplina)

join programa pr on (d.cod\_disciplina = pr.cod\_disciplina)

join specializare sp on (pr.cod\_specializare = sp.cod\_specializare)

join facultate f on (f.cod\_facultate = sp.cod\_facultate)

where nota = ( -- Selectam toate detaliile pentru disciplina cu nota maxima

select max(nota) -- Subcerere in care electam nota maxima din cadrul facultatii date

from examinare e

join disciplina d on (e.cod\_disciplina = d.cod\_disciplina)

join programa p on (d.cod\_disciplina = p.cod\_disciplina)

join specializare sp on (sp.cod\_specializare = p.cod\_specializare)

join facultate f on (sp.cod\_facultate = f.cod\_facultate)

where f.cod\_facultate = cod

) and f.cod\_facultate = cod

order by s.nume, s.prenume -- Ordonam selectia dupa nume si prenume

)

where rownum = 1; -- In cazul in care exista mai multi studenti, selectam doar primul ordonat dupa nume si prenume

-- Afisam informatiile obtinute

dbms\_output.put\_line('Facultate: ' || nume\_facultate);

dbms\_output.put\_line('Nota: ' || nota\_maxima);

dbms\_output.put\_line('Disciplina: ' || nume\_disciplina);

dbms\_output.put\_line('Student: ' || nume\_student);

dbms\_output.put\_line('Profesor: ' || nume\_profesor);

exception

when no\_data\_found then dbms\_output.put\_line('Nu exista facultate cu ranking-ul dat');

when too\_many\_rows then dbms\_output.put\_line('Exista mai multe facultati care au ranking-ul egal cu ' || ranking\_facultate);

when others then dbms\_output.put\_line('Alt tip de eroare!');

end exercitiul\_9;

end exercitiul\_13;

/

**-- Rezultate/Explicatii:**

declare

a tara.nume%type;

begin

exercitiul\_13.exercitiul\_6(11);

exercitiul\_13.exercitiul\_7(3800, 5000);

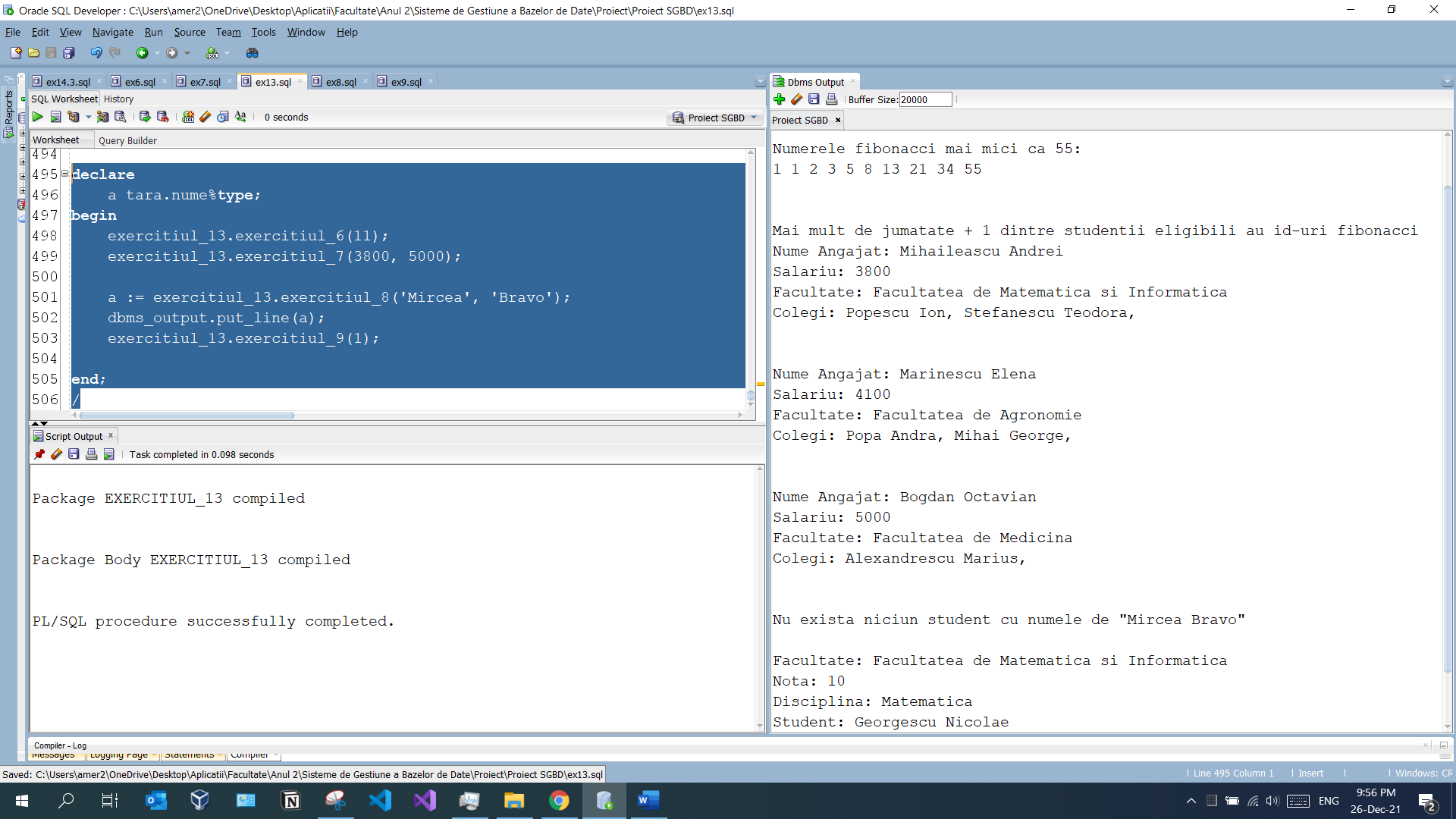
a := exercitiul\_13.exercitiul\_8('Mircea', 'Bravo');

dbms\_output.put\_line(a);

exercitiul\_13.exercitiul\_9(1);

end;

/



/\*

Exemplele rulate in cadrul blocului anonim PL/SQL sunt exemplele executate in cadrul exercitiilor 6-9.

\*/

**14.**

/\*

Sa se selecteze tara in care se afla facultatea studentului al carui salariu profesoral cumulativ este maximal.

Definim salariul profesoral cumulativ al unui student ca fiind suma tututor salariilor profesorilor la care studentul invata.

Salariu profesoral cumulativ maximal reprezinta multimea profesorilor al caror salarii combinate este cea mai mare.

Daca exista mai multi studenti ai caror salarii cumulative sunt egale se va lua in cosiderare primul in ordinea alfabetica dupa nume si prenume.

Dupa ce se afiseaza tara, sa se afiseze nota maxima obtinuta in cadrul facultatii precum si numele materiei, numele profesorului coordonator, numele studentului care a obtinut nota respectiva si, daca exista, caminul la care acesta este cazat.

Daca sunt mai multe facultati la care este inscris studentul, se va lua in considerare doar prima facultate la care s-a inscris.

pasi:

Pentru fiecare student, printam lista profesorilor si salariilor acestora

Pentru fiecare student, calculam salariul profesoral cumulativ

Selectam studentul cu salariul profesoral cumulativ maximal

\*/

**-- Cod Exercitiul 14:**

create or replace package exercitiul\_14

is

cursor studenti is select \* from student; -- Cursor pentru parcurgerea tabelului student

-- Cursor parametrizat pentru parcurgerea listei tuturor profesorilor unui student cu codul dat ca parametru

-- Acest cursor va fi folosit in afiseaza\_profesori()

cursor detalii\_profesor(c\_student student.cod\_student%type)is

select a.nume, a.prenume, a.salariu

from angajat a join profesor prof on (a.cod\_angajat = prof.cod\_angajat)

join participa p on (a.cod\_angajat = p.cod\_angajat)

where p.cod\_student = c\_student

order by a.nume, a.prenume, a.salariu desc;

-- Tip inregistrare(record) folosit pentru extragerea informatiilor(fetch) din cursorul detalii\_profesor

-- Acesta va fi folosit in afiseaza\_profesori()

type rec\_detalii\_profesor is record(

nume angajat.nume%type,

prenume angajat.prenume%type,

salariu angajat.salariu%type

);

-- Tablou indexat de inregistrari de tip rec\_detalii\_profesor

-- Acesta va fi folosit inafiseaza\_profesori()

type ind\_profesori is table of rec\_detalii\_profesor index by pls\_integer;

ind ind\_profesori; -- Declaram un tablou indexat de tipul anterior mentionat, folosit in afiseaza\_profesori()

-- Tablou indexat de salarii cumulative, folosit in afiseaza\_profesori()

type salarii\_cumulative is table of angajat.salariu%type index by pls\_integer;

ind\_salarii salarii\_cumulative; -- Declaram un tablou indexat de tipul anterior mentionat, folosit in afiseaza\_profesori()

type salariu\_maxim is record( -- Tip de data complex ce memoreaza detaliile angajatului ce are salariul maxim

cod student.cod\_student%type,

nume student.nume%type,

prenume student.prenume%type,

salariu angajat.salariu%type

);

maxim salariu\_maxim; -- Declaram un obiect de tipul record anterior mentionat

procedure afiseaza\_detalii\_studenti; -- Procedura ce afiseaza detaliile despre studenti si apeleaza afiseaza\_profesori()

function afiseaza\_profesori(c\_student in student.cod\_student%type) return boolean; -- Functie afiseaza detaliile despre toti profesorii la care invata studentul cu codul dat ca parametru, returneaza true daca studentul are cel putin un profesor, false in caz contrar

function salariu\_cumulativ\_maxim return student.cod\_student%type; -- Functie ce afiseaza detaliile despre salariul cumulativ maximal determinat in prin intermediul apelarii succesive a functiei afiseaza\_profesori() in cadrul procedurii afiseaza\_detalii\_studenti()

function tara\_facultate(c\_student in student.cod\_student%type) return facultate.cod\_facultate%type; -- Functie ce afiseaza tara primei facultati la care este inscris studentul al carui cod este dat ca parametru si returneaza codul respectivei facultati

procedure nota\_maxima(c\_facultate facultate.cod\_facultate%type); -- Procedura ce afiseaza nota maxima obtinuta in cadrul facultatii al carei cod este data ca parametru, impreuna cu restul detaliilor legate de student si profesorul coordonator

function main return boolean; -- Functia principala din care se vor apela toate subprogramele in cascada

end exercitiul\_14;

/

create or replace package body exercitiul\_14

is

procedure afiseaza\_detalii\_studenti

is

contor number := 1; -- Contor folosit strict pentru a tine evidenta indicelui studentului afisat(utilizat strict pentru o afisare mai stilistica)

exista\_profesori boolean; -- Variabila ce primeste ca valoare rezultatul apelarii functiei afiseaza\_profesori() pe un cod de student din cursorul studenti declarat in antetul pachetului

begin

dbms\_output.put\_line('Studenti:');

dbms\_output.new\_line;

dbms\_output.put\_line('Forma afisarii: <<nume>> <<<prenume>> (<<cod>>)');

dbms\_output.put\_line('Profesori: <<nume>> <<prenume>> <<salariu>>');

dbms\_output.new\_line;

for i in studenti loop -- Parcurgem liniile din cursorul studenti

dbms\_output.put\_line(contor || '. ' || i.nume || ' ' || i.prenume || ' (' || i.cod\_student || ')'); -- Afisam indicele, numele, prenumele si codul studentului curent

dbms\_output.put('Profesori: '); -- Pregatim afisarea tuturor profesorilor studentului curent

exista\_profesori := afiseaza\_profesori(i.cod\_student); -- apelam functia afiseaza\_profesori() pentru codul studentului curent

-- Functia afiseaza\_profesori() se va ocupa de afisare si va returna true daca studentul are profesori sau false daca nu are profesori

if exista\_profesori = false then -- Daca studentul nu are niciun profesor atunci afisam un mesaj sugestiv

dbms\_output.put\_line('Studentul nu are niciun profesor');

end if;

dbms\_output.new\_line;

dbms\_output.new\_line;

contor := contor + 1; -- Incrementam contor-ul pentru a trece la urmatorul indice la afisare

end loop;

end afiseaza\_detalii\_studenti;

function afiseaza\_profesori -- Functie ce afiseaza toti profesorii unui student al carui cod este dat ca parametru

(c\_student in student.cod\_student%type)

return boolean -- Functia returneaza true daca studentul are cel putin un profesor, false in caz contrar

is

aux rec\_detalii\_profesor; -- Variabila de tip record rec\_detalii\_profesor ce memoreaza detaliile unui profesor

cnt number := 1; -- Variabila contor/count care incepe de la 1

suma angajat.salariu%type := 0; -- Variabila ce memoreaza suma salariilor profesorilor studentului, suma este initializata cu 0

s\_nume student.nume%type; -- Variabila ce memoreaza numele studentului al cauri cod este dat ca parametru

s\_prenume student.prenume%type; -- Variabila ce memoreaza prenumele studentului al carui cod este dat ca parametru

begin

-- imb.extend;

select nume, prenume -- Selectam numele si prenumele studentului al carui cod este dat ca parametru

into s\_nume, s\_prenume

from student

where cod\_student = c\_student;

-- Acest select nu va genera exceptia no\_data\_found,

-- intrucat aceasta va fi apelata prin intermediul procedurii afiseaza\_detalii\_studenti

-- procedura care facilitateaza apeluri numai pentru coduri de student existente in baza de date

-- De asemenea, nu se va genera exceptia too\_many\_rows, intrucat cheia primara nu se poate repeta in baza de date

open detalii\_profesor(c\_student); -- Deschidem cursorul parametrizat declarat in antetul pachetului si il apelam pentru codul dat ca parametru in functie

loop -- Acest cursor ne parcurge lista tuturor profesorilor studentului cu codul dat ca parametru

fetch detalii\_profesor into aux; -- Obtinem informatiile fiecarui profesor la care studentul invata si le transmitem variabilei record aux

exit when detalii\_profesor%notfound; -- Oprim bucla atunci cand am parcurs toata lista profesorilor studentului al carui cod este dat ca parametru

ind(cnt) := aux; -- Tabloul indexat ind va retine pe pozitia cnt detaliile profesorului curent

-- imb(c\_student)(cnt) := ind(cnt);

cnt := cnt + 1; -- Incrementam cnt pentru urmatoarea iteratie

end loop;

close detalii\_profesor; -- Inchidem cursorul

if ind.count = 0 then -- Daca tabloul indexat ind nu are niciun element, inseamna ca studentul nu are niciun profesor

ind.delete; -- Dezalocam memoria alocata in tabloul indexat(pas inutil din moment ce oricum ind.count = 0)

return false; -- Returnam false, intrucat studentul nu are niciun profesor

else -- Altfel, inseamna ca studentul are cel putin un profesor

for i in ind.first..ind.last loop -- Parcurgem tabloul indexat ind pentru a obtine informatiile pentru fiecare profesor in parte

dbms\_output.put(ind(i).nume || ' ' || ind(i).prenume || ' ' || ind(i).salariu); -- Afisam numele, prenumele si salariul profesorului curent la care studentul invata

suma := suma + ind(i).salariu; -- Calculam salariul profesoral cumulativ pentru studentul dat ca parametru

-- Acest lucru ne va folosi pentru determinarea salariului profesoral cumulativ maximal

-- Salariu cumulativ = suma tuturor salariilor profesorilor din tabloului ind

if i = ind.last then -- Daca am ajuns la ultimul profesor

dbms\_output.put\_line('.'); -- Printam punct

else

dbms\_output.put(', '); -- Altfel separam profesorii prin virgula

end if; -- Observam ca nu mai este nevoie sa folosim dbms\_output.new\_line pentru a se printa toate virgulele din dbms\_output.put()

end loop; -- intrucat oricum la sfarsit se va printa punctul prin dbms\_output.put\_line()

dbms\_output.put\_line('Salariu Cumulativ: ' || suma); -- Afisam salariul cumulativ calculat insumand salariul tuturor profesorilor din ind

if suma > maxim.salariu or -- Daca salariul cumulativ este mai mare decat salariul cumulativ maximal

(suma = maxim.salariu and (s\_nume < maxim.nume)) or -- sau daca salariile sunt egale, dar numele studentului dat ca parametru este mai mic alfabetic decat numele studentului cu salariu cumulativ maximal

(suma = maxim.salariu and s\_nume = maxim.nume and s\_prenume < maxim.prenume)then -- sau daca atat salariile cumulative, cat si numele sunt egale iar prenumele studentului cu codul dat este mai mic alfabetic decat prenumele studentului cu salariu cumulativ maximal

maxim.salariu := suma; -- atunci updatam salariul cumulativ maximal

maxim.nume := s\_nume; -- precum si numele

maxim.prenume := s\_prenume; -- prenumele

maxim.cod := c\_student; -- si codul studentului posesor al acestui salariu cumulativ maximal

end if;

-- ind\_salarii(c\_student) := suma;

suma := 0; -- la sfarsit reinitializam suma cu 0, intrucat aceasta va fi folosita de urmatorul student dat ca parametru

ind.delete; -- dezalocam tabloul de profesori, intrucat si acesta va fi folosit de urmatorul student dat ca parametru

return true; -- returnam true, intrucat am gasit cel putin un profesor pentru studentul dat ca parametru

end if;

end afiseaza\_profesori;

function salariu\_cumulativ\_maxim return student.cod\_student%type -- Functie ce afiseaza detaliile studentului posesor al salariului cumulativ maximal

is -- si returneaza codul acestui student

begin

-- Afisam numele, prenumele si codul studentului posesor al salariului profesoral cumulativ maximal

-- Aceste detalii se afla in variabila record "maxim", declarata in antetul pachetului

-- Iar detaliile din cadrul acestei variabile au fost calculate in functia afiseaza\_profesori() prin intermediul procedurii afiseaza\_detalii\_studenti()

dbms\_output.put\_line('Prin urmare, studentul al carui salariu cumulativ este maxim este:');

dbms\_output.put\_line(maxim.nume || ' ' || maxim.prenume || ' (' || maxim.cod || ') ');

dbms\_output.put\_line('Cu salariul cumulativ de: ' || maxim.salariu);

dbms\_output.new\_line;

dbms\_output.new\_line;

return maxim.cod; -- Returnam codul studentului posesor al salariului profesoral cumulativ maximal

end salariu\_cumulativ\_maxim;

function tara\_facultate -- Functie ce afiseaza tara primei facultati la care este inscris

(c\_student in student.cod\_student%type) -- studentul cu codul dat ca parametru

return facultate.cod\_facultate%type -- si returneaza codul acestei facultati

is

nume\_tara tara.nume%type; -- Variabila in care o sa memoram numele tarii in care se afla facultatea studentului cu codul dat ca parametru

s\_nume student.nume%type; -- Numele studentului cu codul dat ca parametru

s\_prenume student.prenume%type; -- Prenumele studentului cu codul dat ca parametru

f\_nume facultate.denumire%type; -- Numele facultatii studentului cu codul dat ca parametru

f\_cod facultate.cod\_facultate%type; -- Codul facultatii principale al studentului cu codul dat ca parametru

begin

select t.nume, s.nume, s.prenume, f.denumire, f.cod\_facultate -- Selectam numele tarii, numele si prenumele studentului, numele si codul facultatii

into nume\_tara, s\_nume, s\_prenume, f\_nume, f\_cod -- In variabilele anterior declarate in aceasta functie

from tara t join locatie l on (t.cod\_tara = l.cod\_tara)

join facultate f on (l.cod\_locatie = f.cod\_locatie)

join studiaza st on (f.cod\_facultate = st.cod\_facultate)

join student s on(st.cod\_student = s.cod\_student)

where s.cod\_student = c\_student -- Acolo unde codul studentului este egal cu cel dat ca parametru

and rownum = 1; -- Daca exista mai multe facultati, se va selecta prima la care a fost inscris studentul

-- Acest select nu va genera exceptia no\_data\_found, intrucat avem garantia validitatii codului dat ca parametru, datorita apelarii in cascada a subprogramelor din acest pachet

-- Afisam detaliile obtinute

dbms\_output.put\_line('Studentul ' || s\_nume || ' ' || s\_prenume || ' este inscris la ' || f\_nume || ', care se afla in tara: ' || nume\_tara);

dbms\_output.new\_line; -- Afisam o noua linie pentru spatiere

return f\_cod; -- Returnam codul facultatii principale

end tara\_facultate;

procedure nota\_maxima(c\_facultate facultate.cod\_facultate%type) -- Procedura ce afiseaza nota maxima obtinuta in cadrul facultatii cu codul dat ca parametru

is

f\_denumire facultate.denumire%type; -- Variabila pentru numele facultatii cu codul dat ca parametru

s\_nume student.nume%type; -- Variabila pentru numele studentului cu nota maxima din cadrul facultatii cu codul dat ca parametru

s\_prenume student.prenume%type; -- Prenumele studentului cu nota maxima din facultate

a\_nume angajat.nume%type; -- Numele profesorului coordonator al disciplinei in care s-a obtinut nota maxima

a\_prenume angajat.prenume%type; -- Prenumele profesorului anterior mentionat

c\_denumire camin.denumire%type; -- Denumirea caminului in care este cazat studentul ce a obtinut nota maxima

d\_denumire disciplina.denumire%type; -- Denumirea disciplinei la care s-a obtinut nota maxima

e\_nota examinare.nota%type; -- Nota maxima obtinuta in cadrul facultatii cu codul dat ca parametru

begin

/\*

Dupa ce se afiseaza tara, sa se afiseze nota maxima obtinuta in cadrul facultatii

precum si numele materiei, numele profesorului coordonator,

numele studentului care a obtinut nota respectiva si, daca exista,

caminul la care acesta este cazat

\*/

-- Selectam toate detaliile cerute si le memoram in variabilele declarate in cadrul procedurii

select f.denumire, s.nume, s.prenume, a.nume, a.prenume, c.denumire, d.denumire, e.nota

into f\_denumire, s\_nume, s\_prenume, a\_nume, a\_prenume, c\_denumire, d\_denumire, e\_nota

from facultate f join studiaza st on (f.cod\_facultate = st.cod\_facultate)

join student s on (st.cod\_student = s.cod\_student)

join participa p on (s.cod\_student = p.cod\_student)

join disciplina d on (p.cod\_disciplina = d.cod\_disciplina)

join examinare e on (d.cod\_disciplina = e.cod\_disciplina)

join angajat a on (a.cod\_angajat = p.cod\_angajat)

join profesor prof on (a.cod\_angajat = prof.cod\_angajat)

join camin c on (s.cod\_camin = c.cod\_camin)

where f.cod\_facultate = c\_facultate and rownum = 1 -- Selectam toate detaliile pentru facultatea cu codul dat ca parametru, daca exista mai multe materii la care s-a obtinut nota maxima, o selectam doar pe prima

order by nota desc; -- Ordonam descrescator dupa nota

-- Afisam detaliile cerute

dbms\_output.put\_line('Pentru facultatea "' || f\_denumire || '" nota maxima este ' || e\_nota || ' si este la materia ' || d\_denumire || ', ');

dbms\_output.put\_line('nota fiind obtinuta de studentul ' || s\_nume || ' ' || s\_prenume || ' alaturi de profesorul coordonator ');

dbms\_output.put\_line(a\_nume || ' ' || a\_prenume || ', studentul fiind cazat la caminul ' || c\_denumire || '.');

dbms\_output.new\_line;

end nota\_maxima;

function main return boolean -- In functia main vom apela toate subprogramele create in cadrul pachetului

is

cod\_salariu\_maxim student.cod\_student%type; -- Variabila in care vom memora codul studentului al carui salariu profesoral cumulativ este maxim

c\_facultate facultate.cod\_facultate%type;

begin

maxim.salariu := -1; -- Initializam primul salariu maxim ca fiind -1, astfel incat orice salariul profesoral sa fie mai mare decat -1

afiseaza\_detalii\_studenti(); -- Afisam detaliile studentilor, aceasta procedura va apela si functia afiseaza\_profesori()

-- Dupa ce s-au apelat procedura afiseaza\_detalii\_student() si functia afiseaza\_profesori(), salariul cumulativ maximal a fost calculat

cod\_salariu\_maxim := salariu\_cumulativ\_maxim(); -- Prin urmare, apelam functia salariu\_cumulativ\_maxim() pentru a afisa detaliile studentului posesor al salariului cumulativ maximal si returnam codul acestui student, cod pe care il memoram in cod\_salariu\_maxim

c\_facultate := tara\_facultate(cod\_salariu\_maxim); -- Dupa ce am obtinut codul studentului cu salariul cumulativ maximal, apelam functia tara\_facultate() pentru a afisa tara in care se afla prima facultate la care este inscris studentul si returnam codul acestei facultati

nota\_maxima(c\_facultate); -- Cu codul facultatii obtinut, apelam procedura nota\_maxima() pentru a afisa nota maxima obtinuta in cadrul acelei facultati, precum si toate detaliile legate de studentul si profesoorul coordonator

return true; -- Functia main returneaza true atunci cand toate procedurile si functiile s-au executat cu succes

end main;

end exercitiul\_14;

/

**-- Rezultate/Explicatii:**

-- Bloc anonim in care testam pachetul definit

declare

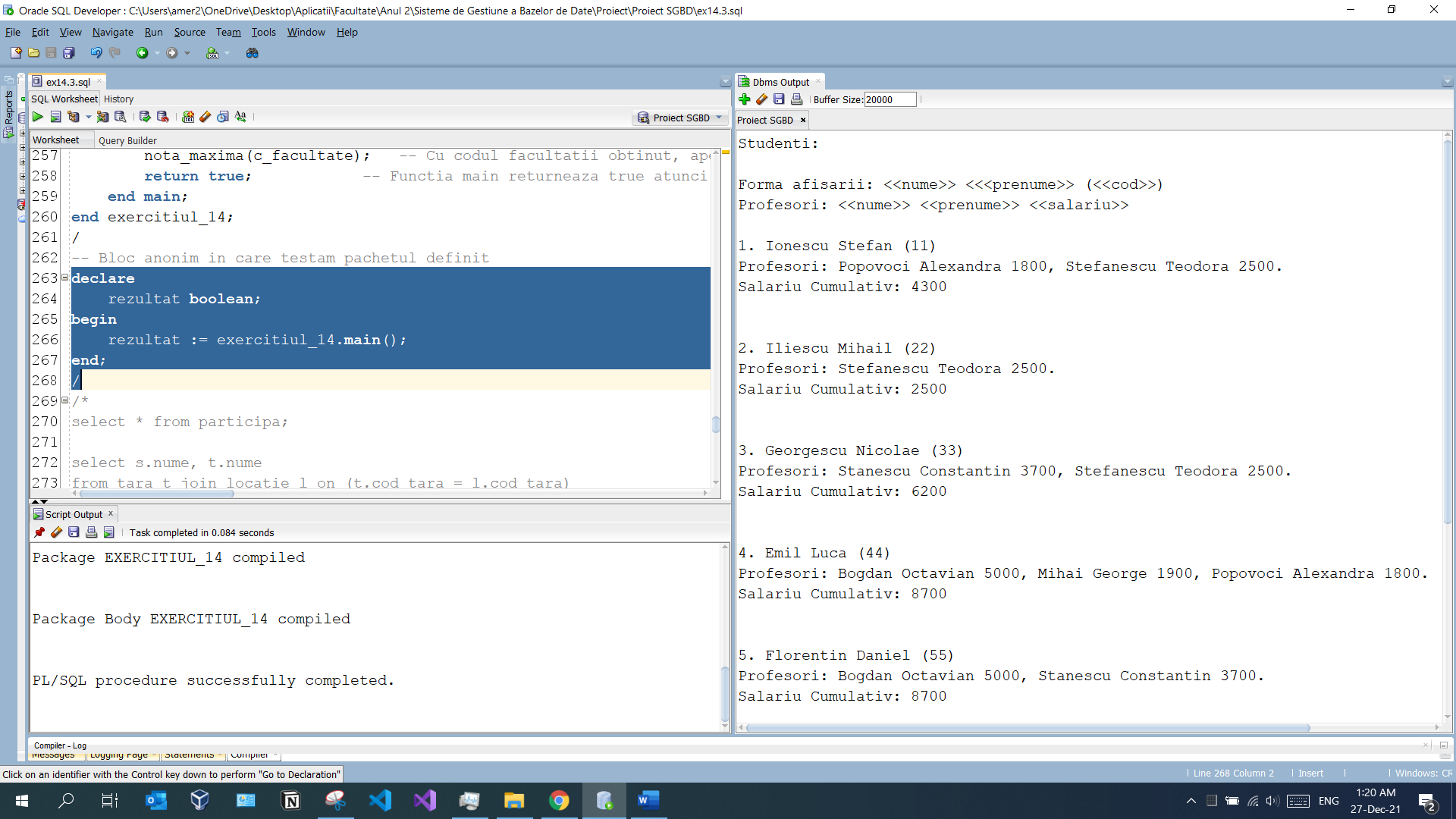
rezultat boolean;

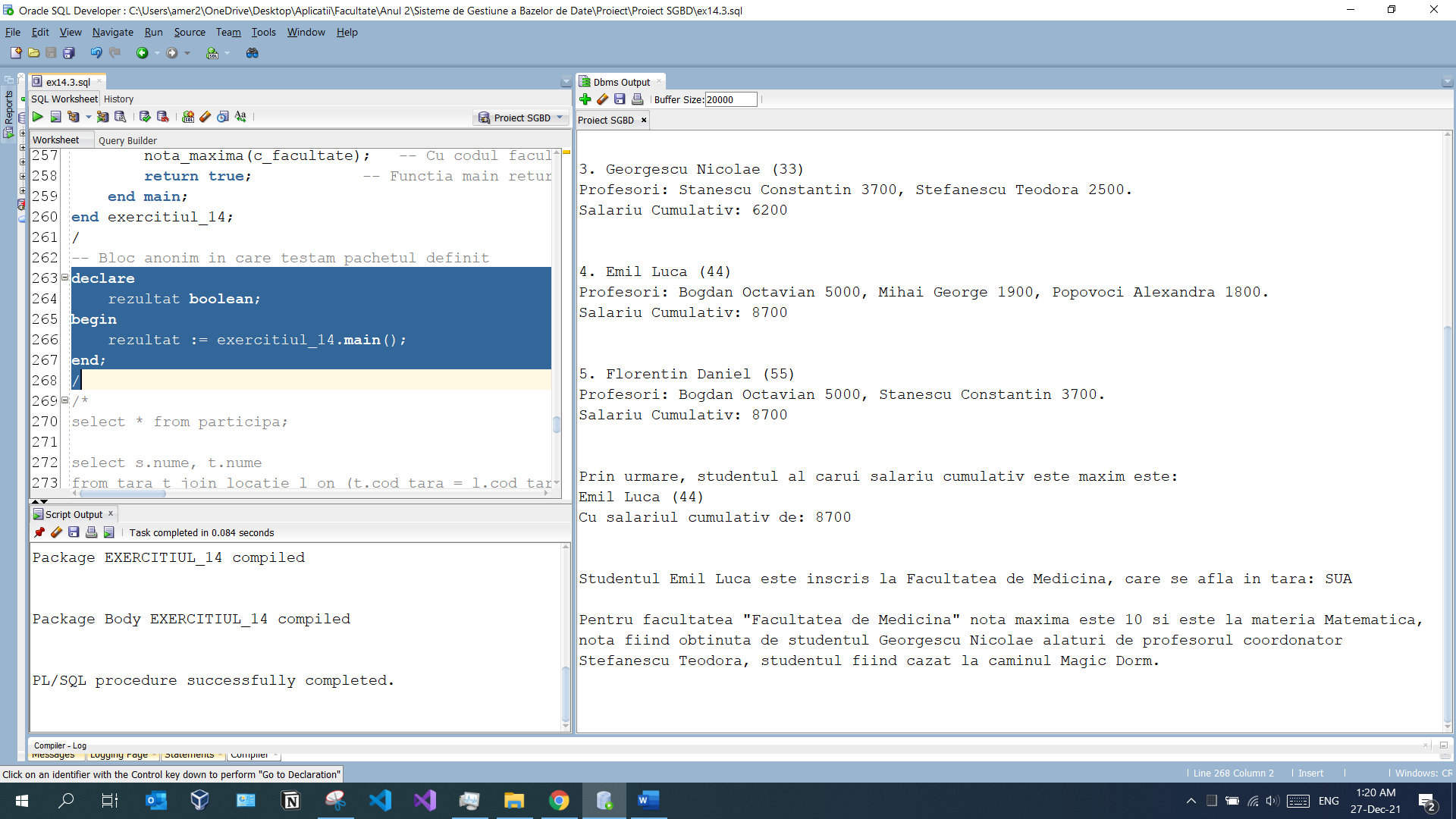
begin

rezultat := exercitiul\_14.main();

end;

/





-- Toate explicatiile necesare se regasesc in cadrul dbms\_output din cele doua poze