**Proiect Baze de Date 2023**

**Gestionarea unui lant de aeroporturi**

**Cerinte Suplimentare**

[14) Creati o vizualizare complexa care sa contina CNP-ul, codul abonamentului si numarul de puncte al pasagerilor care au mai mult de 0 puncte pe abonament. 1](#_Toc137386607)

[15) Interogari 4](#_Toc137386608)

[i) Selectati codul, numele si numarul de calatorii al primelor 5 aeroporturi care au efectuat cele mai multe calatorii. 4](#_Toc137386609)

[ii)Sa se obtina CNP-ul tuturor pasagerilor care au participat la toate calatoriile cu avionul care are id-ul 17 5](#_Toc137386610)

[iii) Selectati codurile si modelul avioanelor care au avut cei mai multi pasageri intr-o calatorie, chiar daca nu se cunoaste modelul avioanelor. 6](#_Toc137386611)

[16) Selectati codul si numele angajatilor al caror salariu minim al jobului pe care il au este mai mare strict decat 6000, iar salariul lor actual este mai mare sau egal cu 10000. 8](#_Toc137386612)

[17) Normalizare in BCNF, FN4 si FN5 si denormalizare 11](#_Toc137386613)

[a) Normalizare 11](#_Toc137386614)

[b)Denormalizare 12](#_Toc137386615)

[18) Tranzactii: reprezentarea Consistency Levels in Oracle SQL 13](#_Toc137386616)

[19) Optimizarea a doua cereri folosind Indexing 21](#_Toc137386617)

# 14) Creati o vizualizare complexa care sa contina CNP-ul, codul abonamentului si numarul de puncte al pasagerilor care au mai mult de 0 puncte pe abonament.

--crearea unei vizualizari+operatii

create view v\_abonat\_puncte as

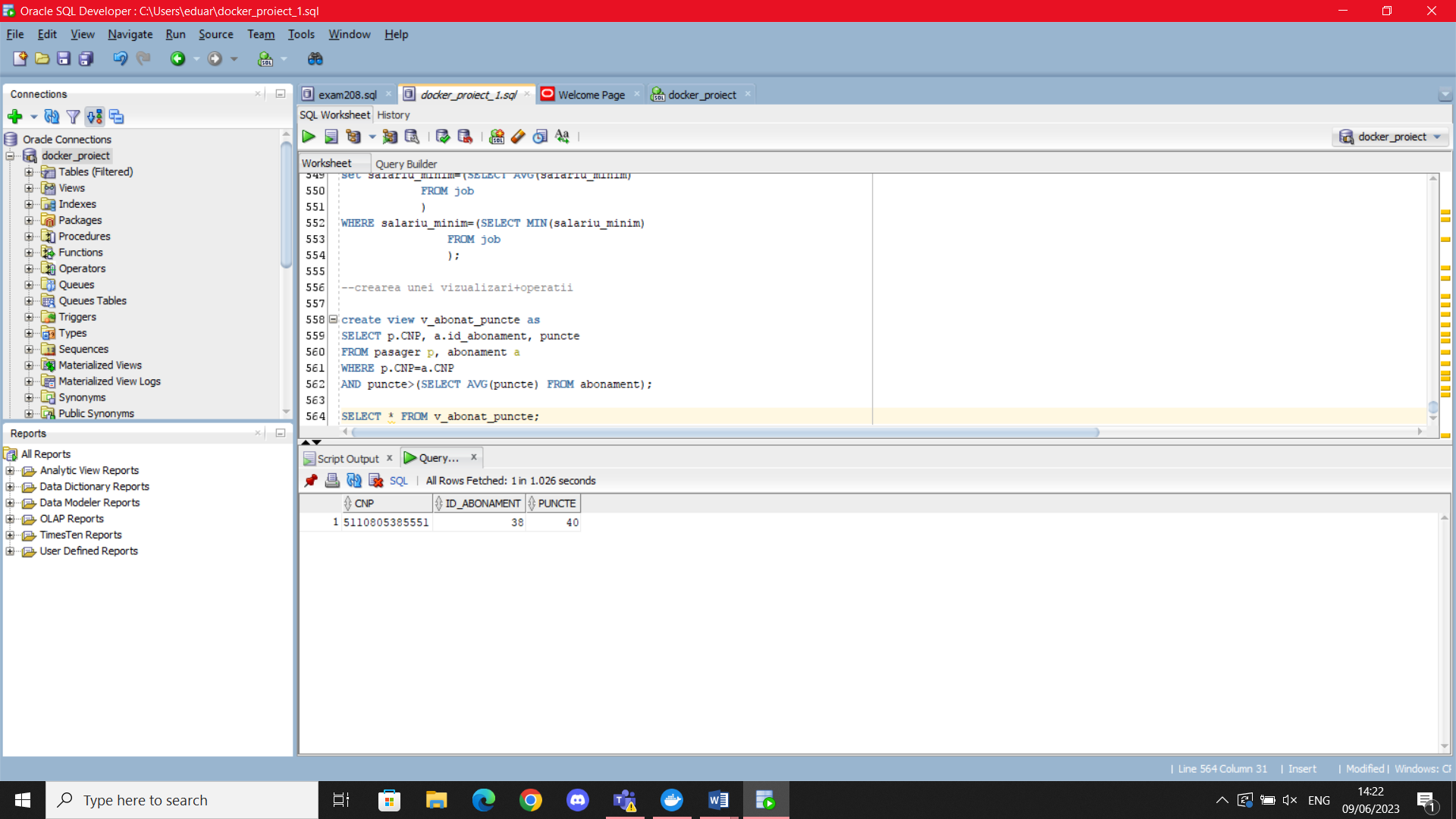
SELECT p.CNP, a.id\_abonament, puncte

FROM pasager p, abonament a

WHERE p.CNP=a.CNP

AND puncte>(SELECT AVG(puncte) FROM abonament);

SELECT \* FROM v\_abonat\_puncte;



--operatie permisa: adunati la numarul de puncte al abonamentelor maximul de puncte al unui abonament.

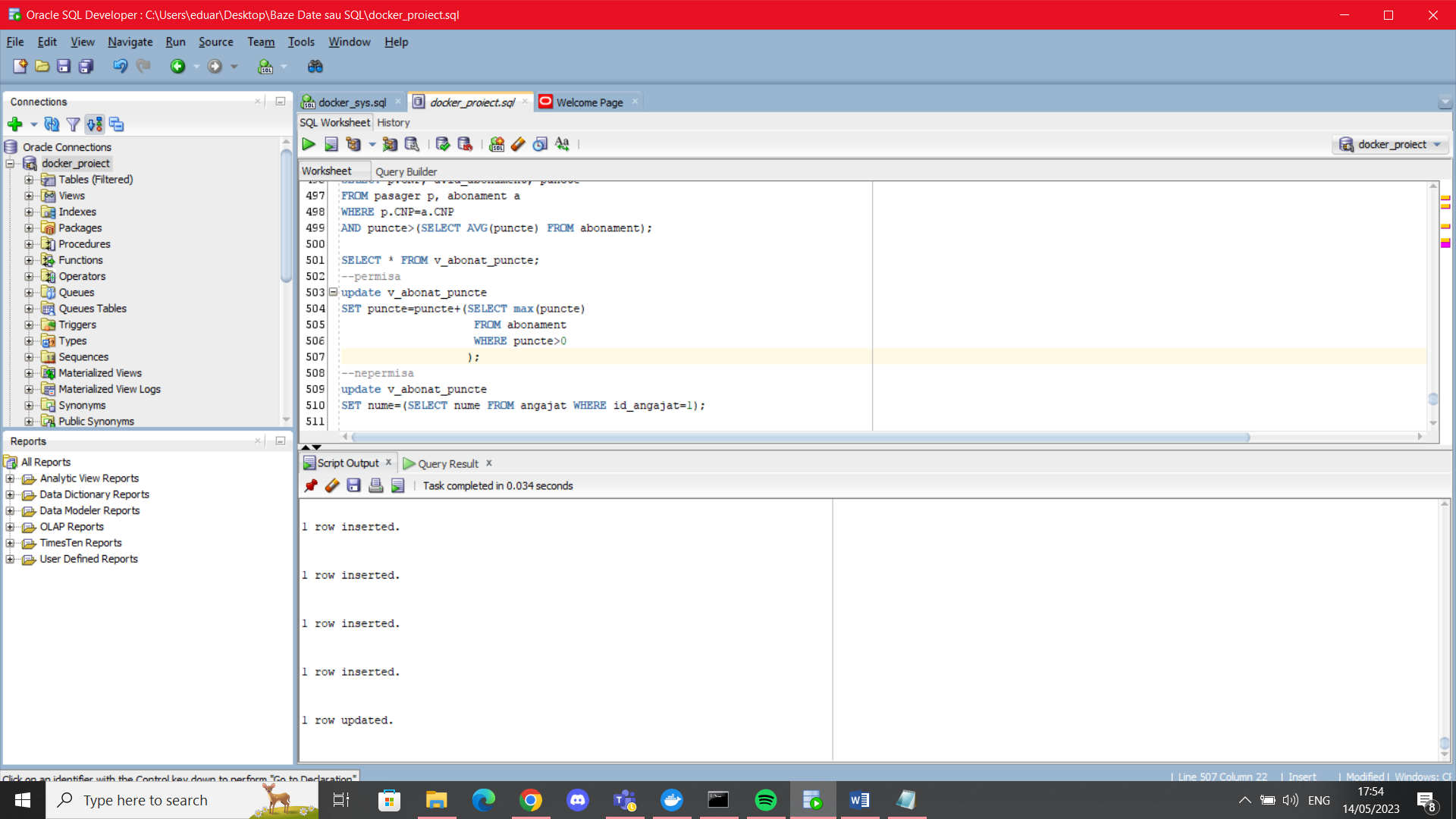
update v\_abonat\_puncte

SET puncte=puncte+(SELECT max(puncte)

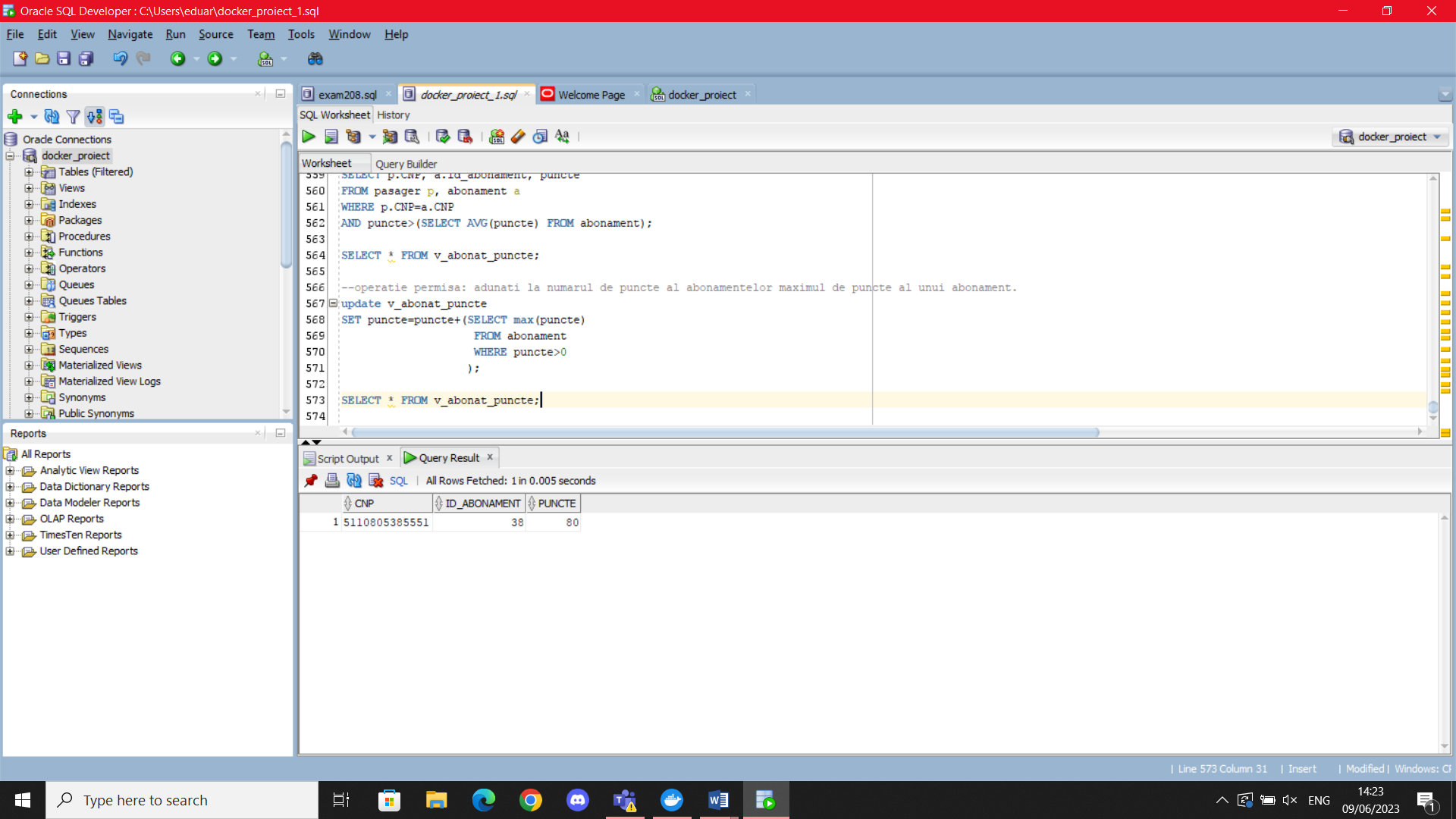
FROM abonament

WHERE puncte>0

);



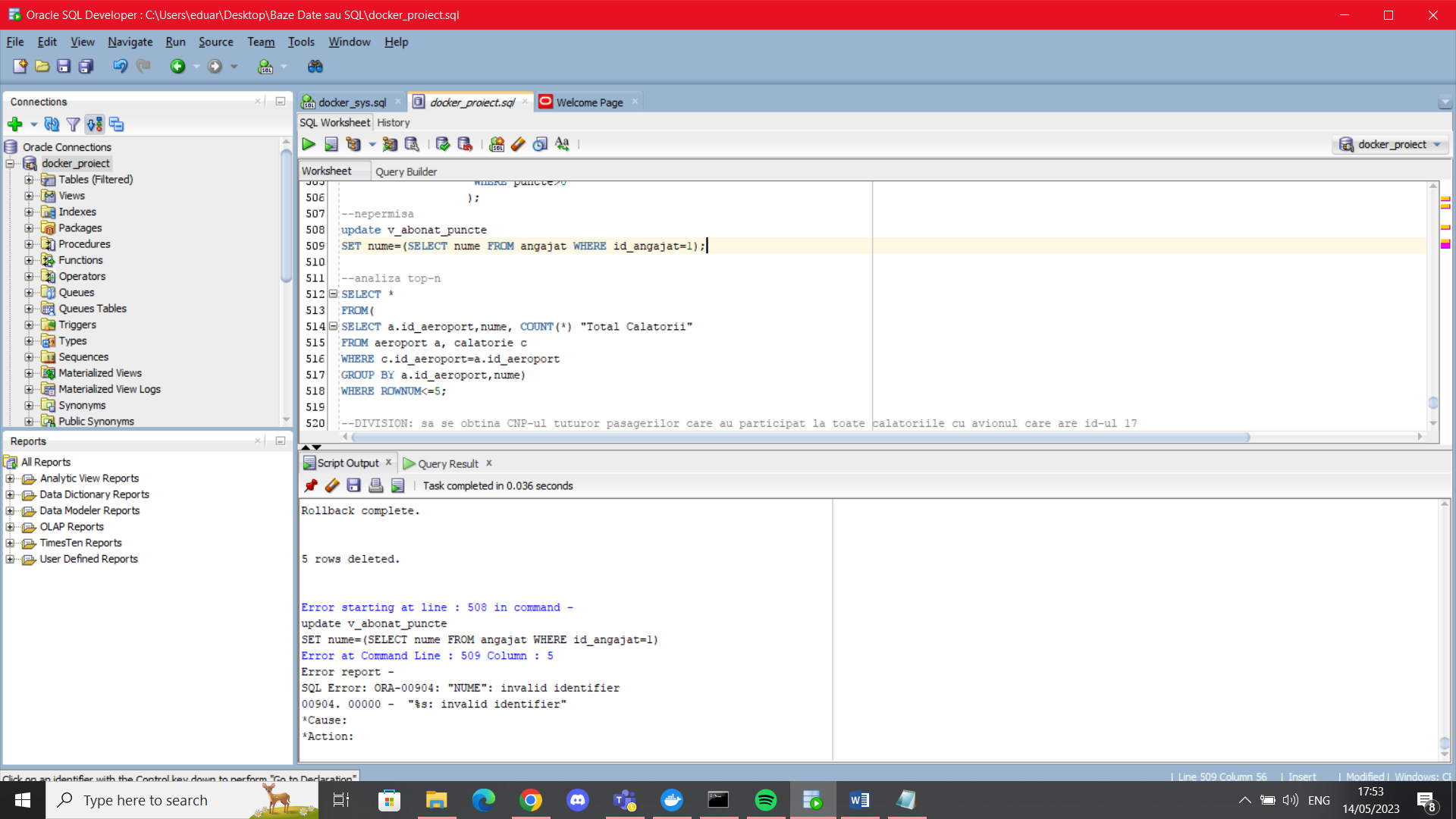




--operatie nepermisa: schimbati numele pasagerilor cu numele angajatului care are id-ul 1.

update v\_abonat\_puncte

SET nume=(SELECT nume FROM angajat WHERE id\_angajat=1);



# 15) Interogari

## i) Selectati codul, numele si numarul de calatorii al primelor 5 aeroporturi care au efectuat cele mai multe calatorii.

--analiza top-n

SELECT \*

FROM(

SELECT a.id\_aeroport,nume, COUNT(\*) "Total Calatorii"

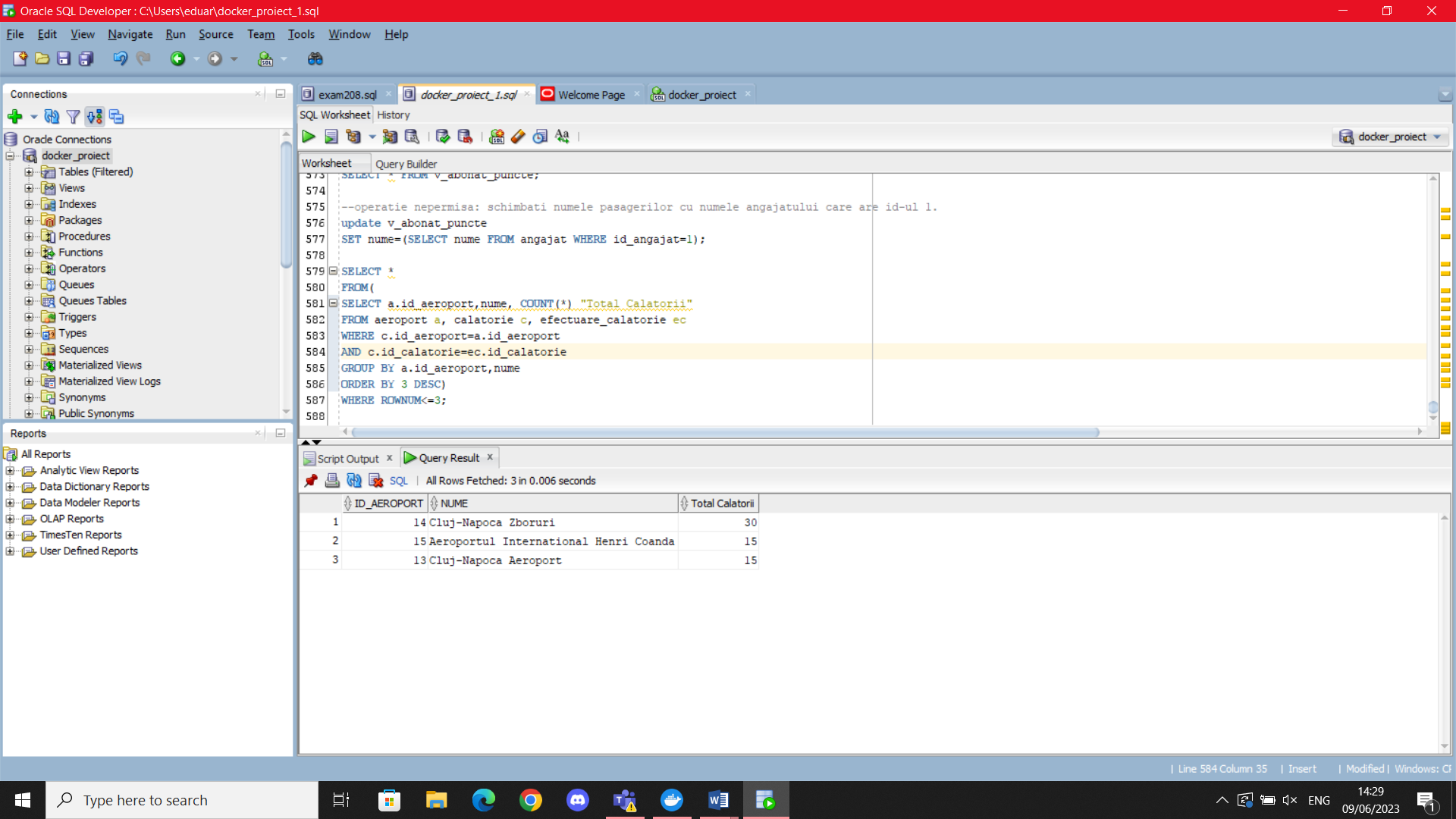
FROM aeroport a, calatorie c, efectuare\_calatorie ec

WHERE c.id\_aeroport=a.id\_aeroport

AND c.id\_calatorie=ec.id\_calatorie

GROUP BY a.id\_aeroport,nume

ORDER BY 3 DESC)

WHERE ROWNUM<=3; 

## ii)Sa se obtina CNP-ul tuturor pasagerilor care au participat la toate calatoriile cu avionul care are id-ul 17

--DIVISION: sa se obtina CNP-ul tuturor pasagerilor care au participat la toate calatoriile cu avionul care are id-ul 17

INSERT INTO efectuare\_calatorie VALUES('5030802385555',17,10);

select DISTINCT CNP

FROM efectuare\_calatorie ec

WHERE NOT EXISTS(SELECT 1

FROM avion a

WHERE id\_avion=3

AND NOT EXISTS(SELECT 2

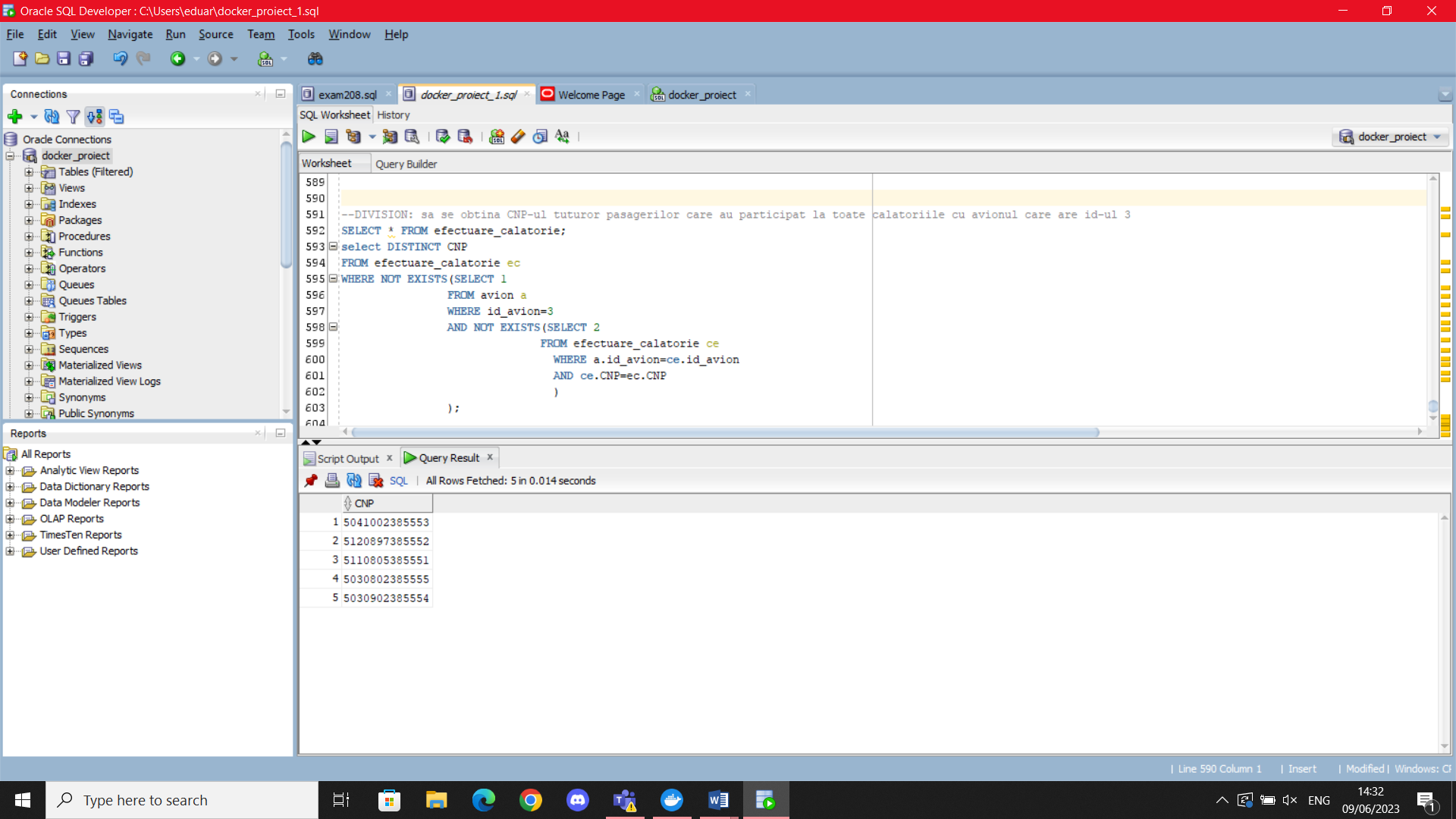
FROM efectuare\_calatorie ce

WHERE a.id\_avion=ce.id\_avion

AND ce.CNP=ec.CNP

)

);



## iii) Selectati codurile si modelul avioanelor care au avut cei mai multi pasageri intr-o calatorie, chiar daca nu se cunoaste modelul avioanelor.

Pentru a demonstra efectul outer join-ului, voi introduce in tabelul AVION un avion care are valoarea NULL in cheia straina model\_avion, care va efectua o calatorie cu 2 pasageri.

INSERT INTO avion(id\_avion) VALUES(30);

INSERT INTO efectuare\_calatorie(cnp,id\_avion,id\_calatorie)VALUES('5030802385555', 30, 3);

INSERT INTO efectuare\_calatorie(cnp,id\_avion,id\_calatorie)VALUES('5030902385554', 30, 3);

WITH pasageri\_per\_calatorie AS(

SELECT a.id\_avion,a.tip\_avion,ec.id\_calatorie, count(distinct p.CNP) nr\_pasageri

FROM model\_avion ma, avion a, efectuare\_calatorie ec, pasager p

WHERE ma.tip\_avion(+)=a.tip\_avion

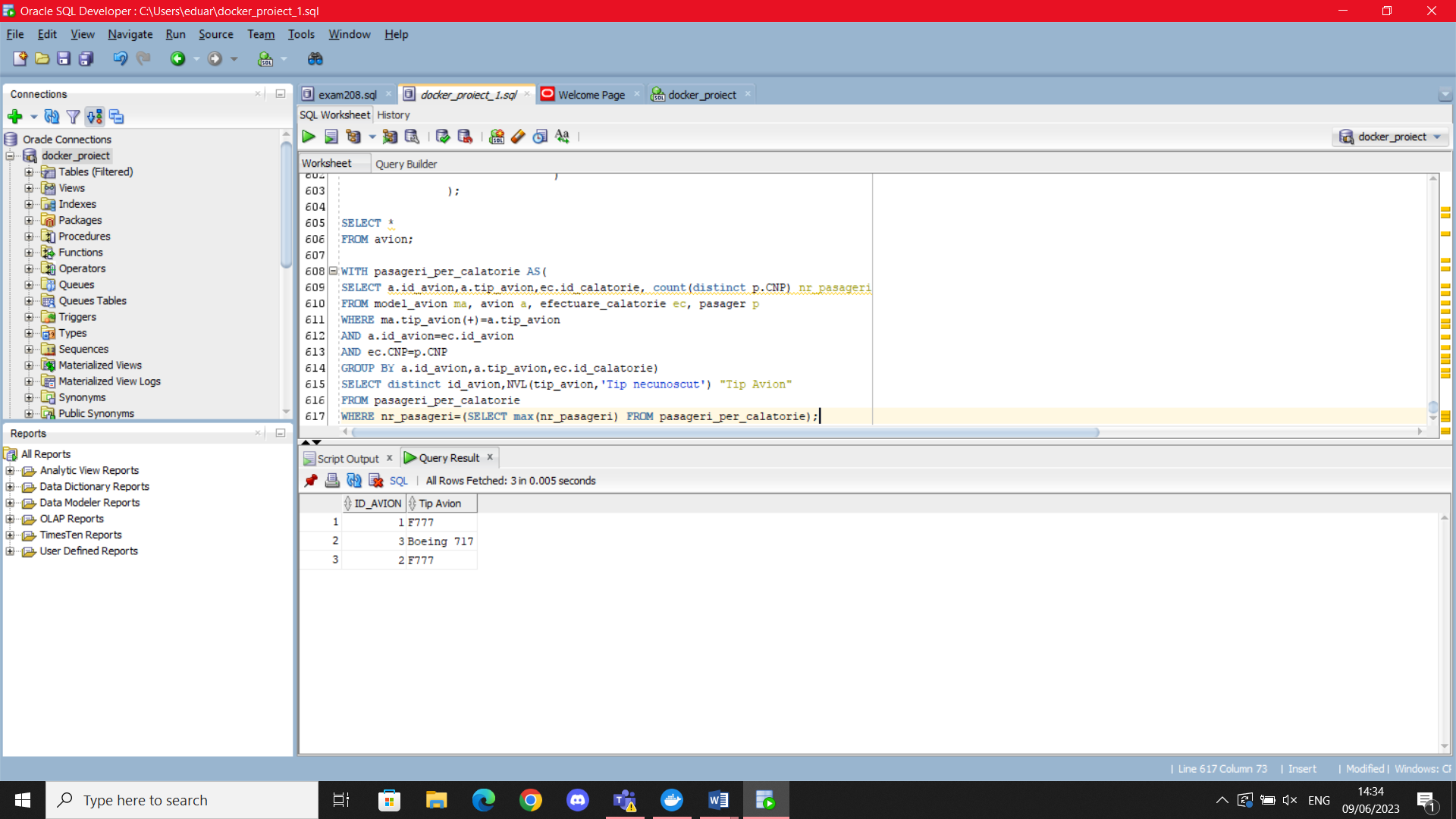
AND a.id\_avion=ec.id\_avion

AND ec.CNP=p.CNP

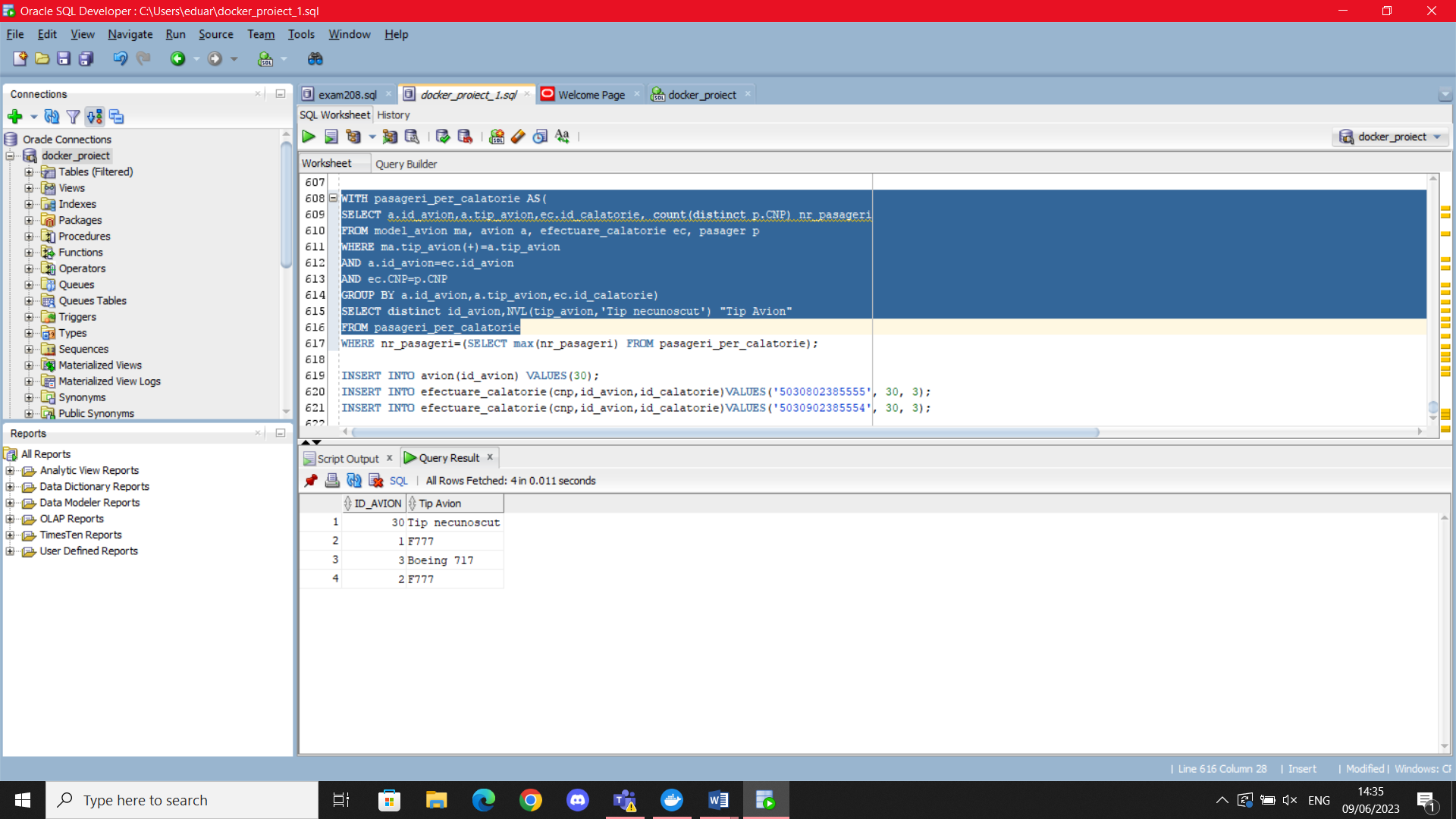
GROUP BY a.id\_avion,a.tip\_avion,ec.id\_calatorie)

SELECT distinct id\_avion,NVL(tip\_avion,'Tip necunoscut') "Tip Avion"

FROM pasageri\_per\_calatorie

WHERE nr\_pasageri=(SELECT max(nr\_pasageri) FROM pasageri\_per\_calatorie); 

In blocul WITH se implementeaza operatia de outer join. Daca nu am fi avut ultima conditie, rezultatul ar fi fost urmatorul:



# 16) Selectati codul si numele angajatilor al caror salariu minim al jobului pe care il au este mai mare strict decat 6000, iar salariul lor actual este mai mare sau egal cu 10000.

Formulez urmatoarele doua cereri:

i)

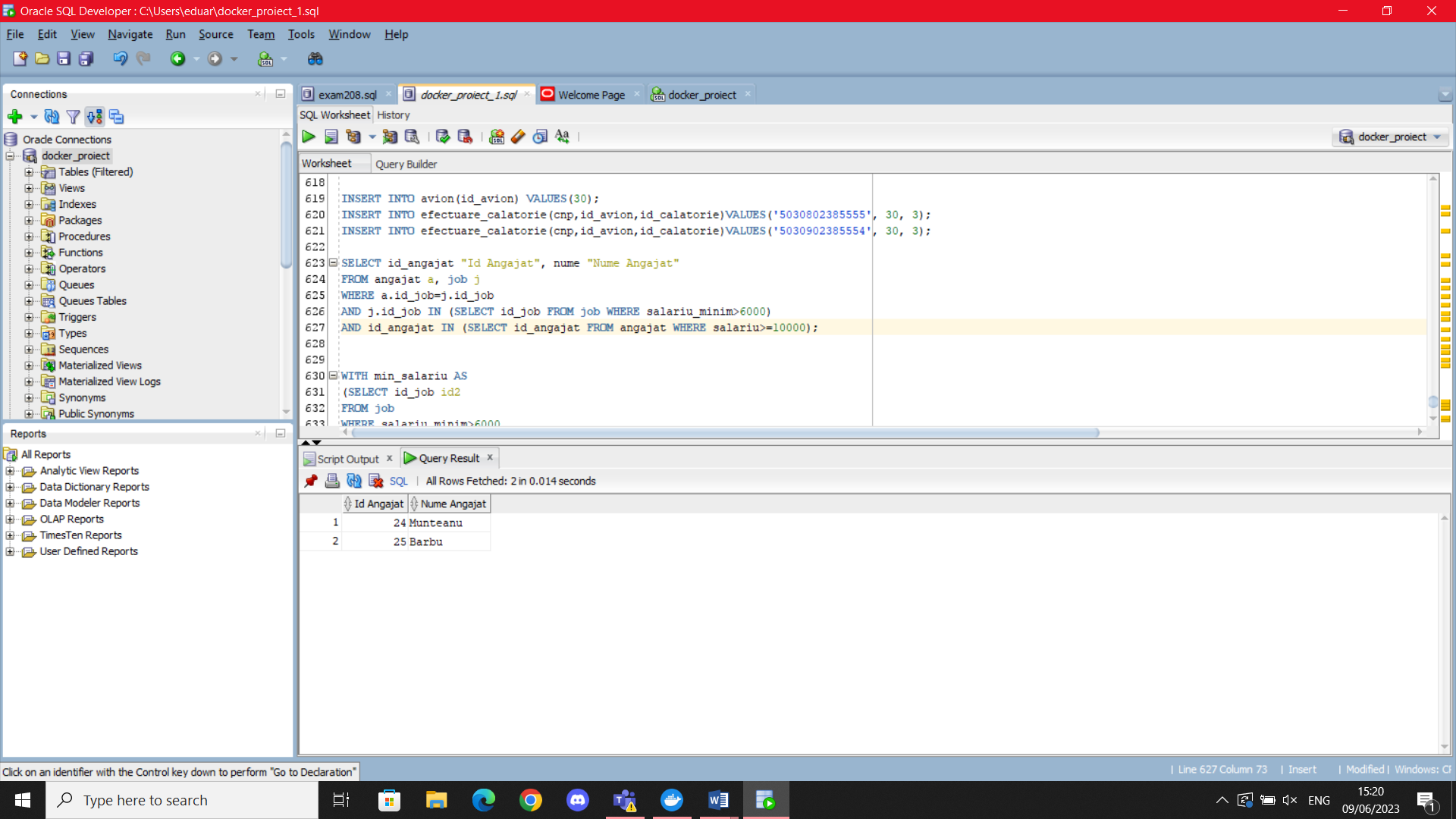
SELECT id\_angajat "Id Angajat", nume "Nume Angajat"

FROM angajat a, job j

WHERE a.id\_job=j.id\_job

AND j.id\_job IN (SELECT id\_job FROM job WHERE salariu\_minim>6000)

AND id\_angajat IN (SELECT id\_angajat FROM angajat WHERE salariu>=10000);



Aceasta prima cerere realizeaza intai subcererile nesincronizate, pe tabelul ANGAJAT, cu 11 elemente si pe tabelul JOB cu 7 elemente. Apoi, face un join intre aceste doua tabele, din cererea primara, pastrand doar id-urile care se afla in multimea de date din subcereri. Astfel se realizeaza un produs cartezian intre 7 si 11 elemente, in total incercand 7\*11 combinatii.

ii)

WITH min\_salariu AS

(SELECT id\_job id2

FROM job

WHERE salariu\_minim>6000

),

salariu\_min\_angajat AS

(SELECT id\_angajat ida, nume n, id\_job id1

FROM angajat

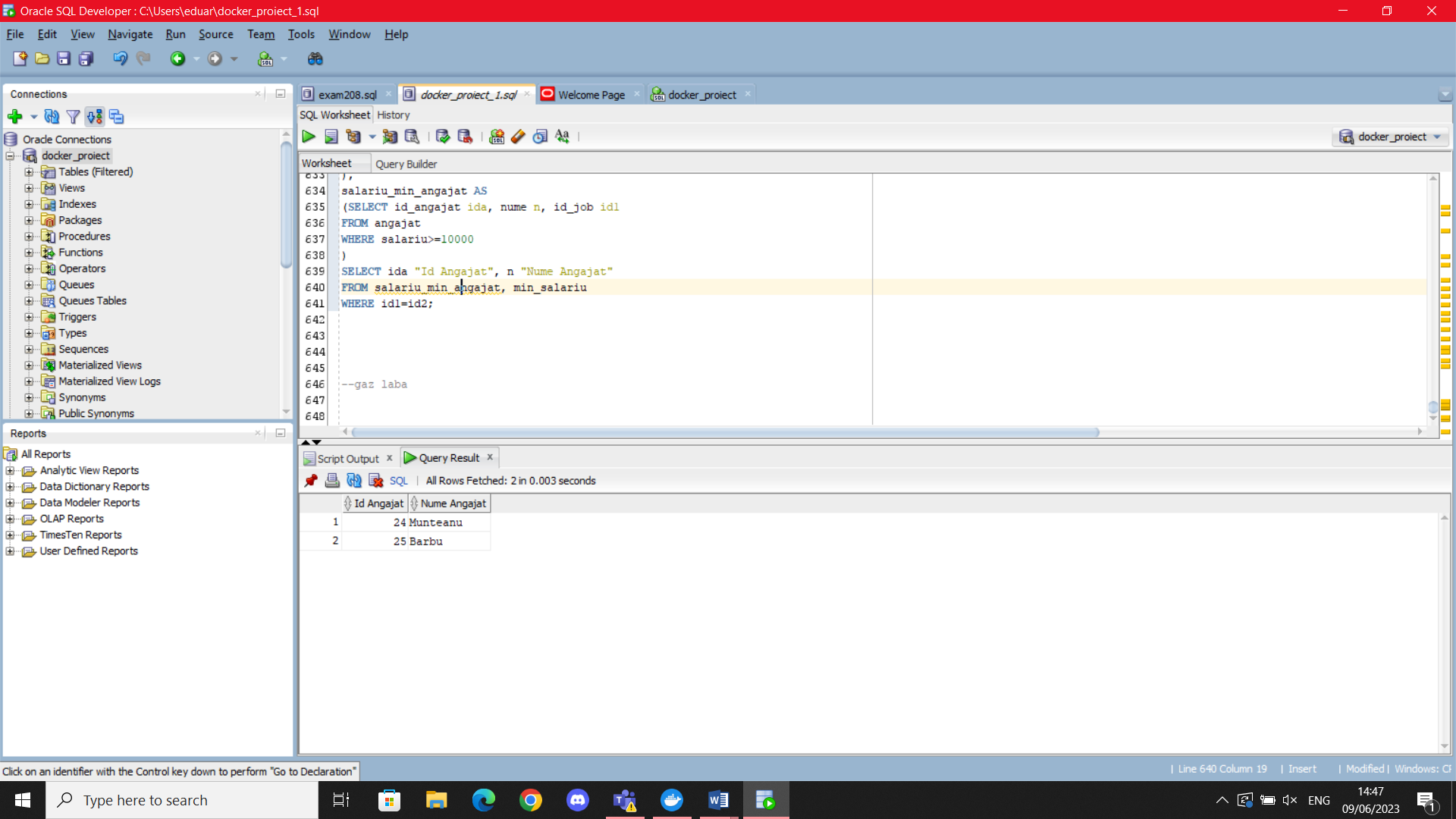
WHERE salariu>=10000

)

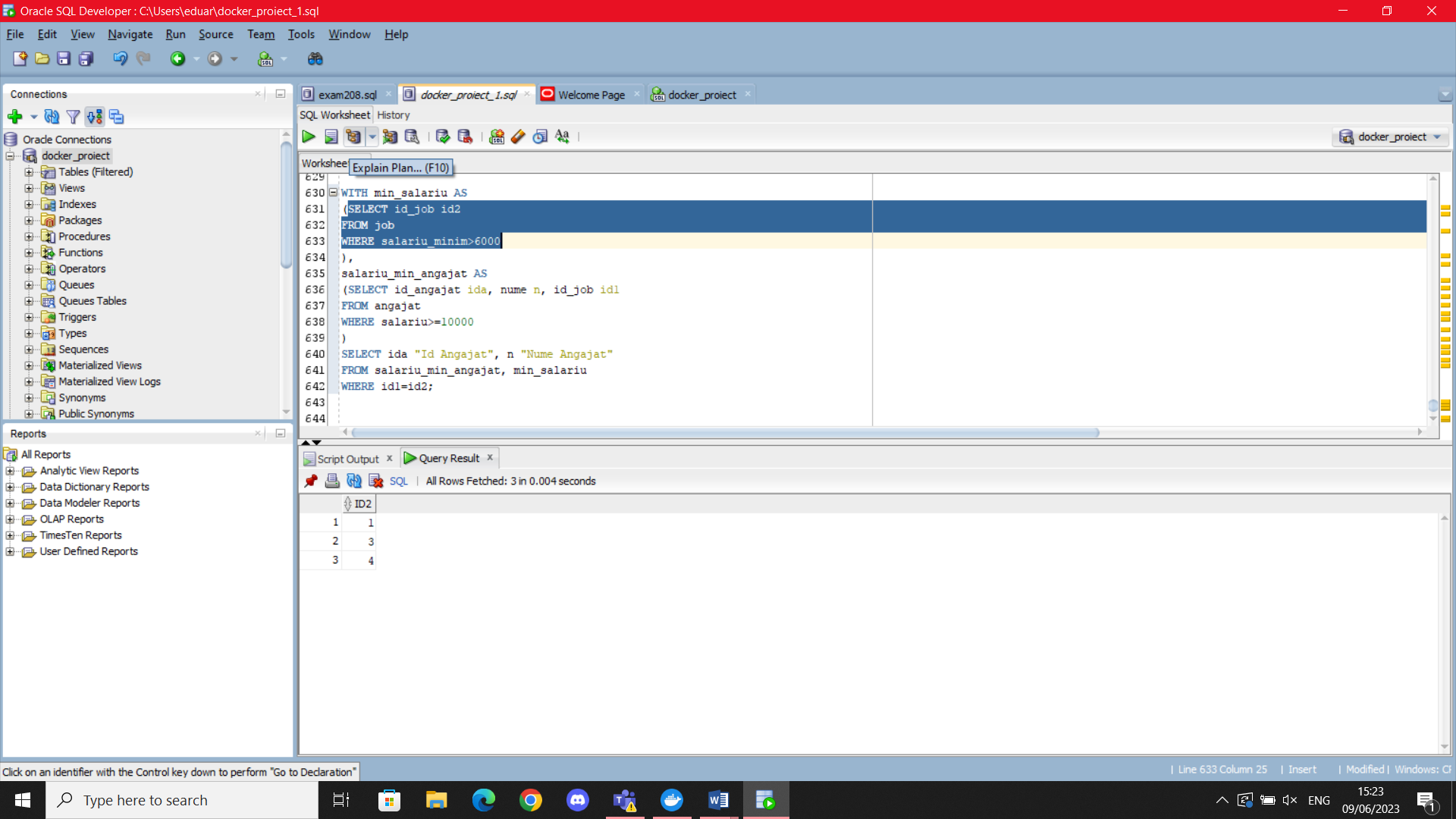
SELECT ida "Id Angajat", n "Nume Angajat"

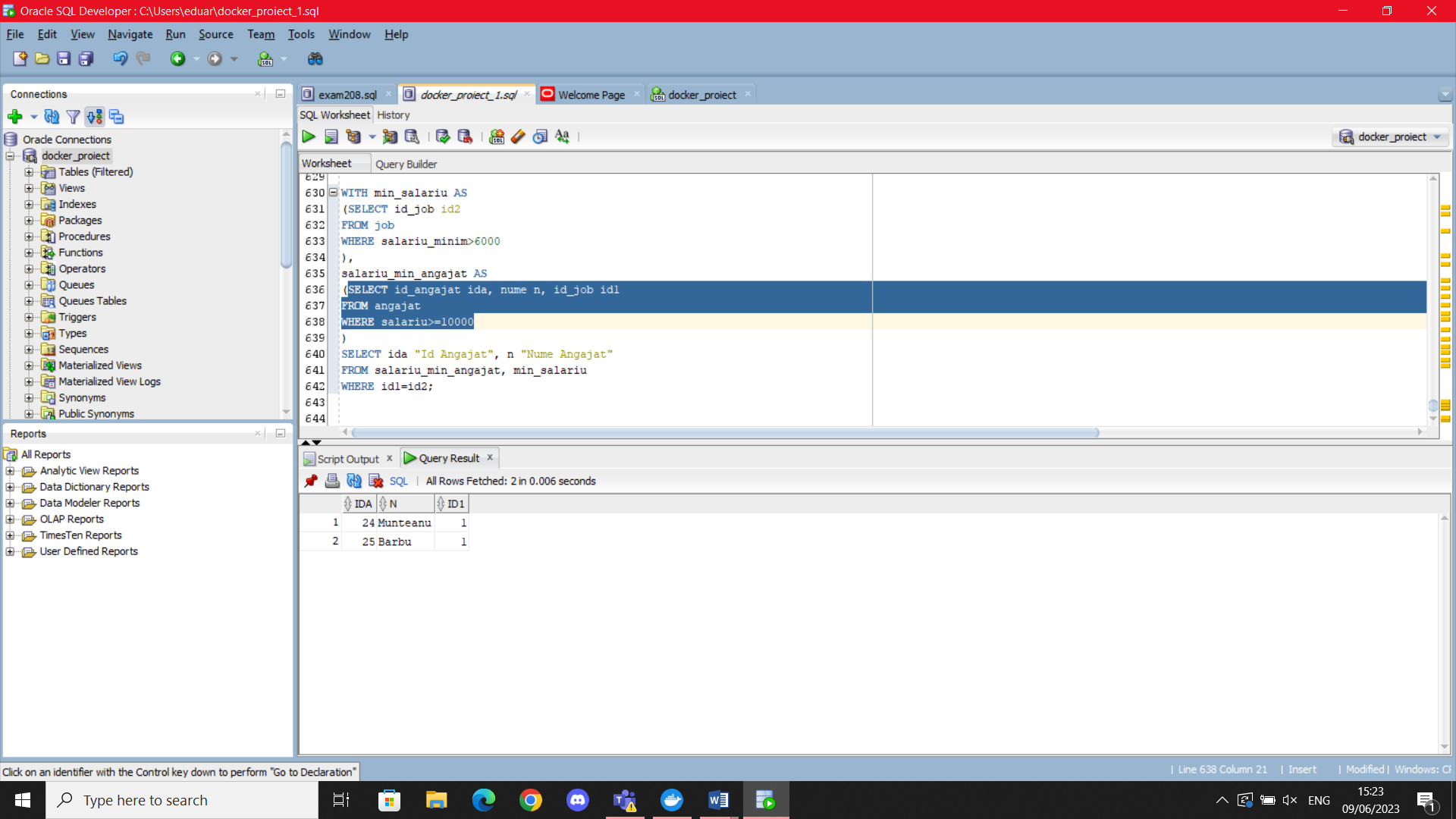
FROM salariu\_min\_angajat, min\_salariu

WHERE id1=id2;



Se observa ca rezultatele sunt aceleasi, deci sunt echivalente din punct de vedere sintactic. Insa, in a doua cerinta, mai intai se fac cererile din blocul WITH. Acestea au urmatoarele rezultate:





Initial, se realizeaza aceste doua tabele prin verificarea si pastrarea datelor care indeplinesc cerintele. In total, se aplica operatii asupra 11 elemente din tabelul ANGAJAT si 7 elemente din tabelul JOB.

Dupa aceea, folosind rezultatele din cele doua tabele, se realizeaza join-ul pe cele doua tabele pentru a afisa rezultatul final. Aceasta operatie realizeaza produsul cartezian dintre o multime cu 3 elemente si o a doua multime cu doua elemente, in total realizand 6 operatii.

# 17) Normalizare in BCNF, FN4 si FN5 si denormalizare

## a) Normalizare

**Un exemplu NON-BCNF este urmatorul:**

Voi porni de la schema relationala:

ANGAJAT(id\_angajat, nume, prenume, email, salariu, data\_angajarii, id\_job, id\_aeroport).

In acest caz, id\_angajat, fiind cheie primara, este si o cheie candidat, intrucat, impreuna cu adresa de mail, care are constrangerea de a fi unica, identifica in mod unic fiecare rand al tabelului. Fiindca id\_job si id\_aeroport sunt chei straine, nu vor fi luate in considerare la determinarea cheii candidat.

Daca as modifica schema astfel:

ANGAJAT(id\_angajat, nume, prenume, email, salariu, data\_angajarii, job, aeroport)

, unde job si aeroport nu mai sunt chei straine, ci atribute varchar2 care acum depind de atributul salariu. Atunci, job si aeroport depind de un atribut non-prim, iar tabelul nu ar fi BCNF. Pentru a rezolva problema, am putea imparti tabelul in doua, separand atributele job si aeroport de ANGAJAT astfel:

ANGAJAT(id\_angajat, nume, prenume, email, salariu, data\_angajarii)

LOC\_MUNCA(salariu, job, aeroport).

**Un exemplu NON-FN4 este urmatorul:**

Pornind de la schema relationala CERTIFICAT(id\_certificat, data\_primirii, id\_angajat), voi adauga atributul calificare\_angajat si voi presupune ca id\_angajat nu mai este o cheie straina, ci o cheie primara. Astfel, id\_certificat si id\_angajat formeaza cheia candidat a tabelului. data\_primirii depinde de id\_certificat, dar calificare\_angajat depinde de id\_angajat. Tabelul este in BCNF, dar apare fenomenul Multi-Valued Dependancy.

Astfel, nu exista depdenta functionala intre Id\_certificat si calificare\_angajat, dar nici intre id\_angajat si data\_primirii, care pot cauza probleme. Deci, pentru a transforma tabelul in FN4, certificate poate fi impartit astfel:

CERTIFICAT(id\_certificat, data\_primirii)

PRIMIRE\_CERTIFICAT(id\_certificat, id\_angajat)

ANGAJAT\_ACREDITAT(id\_angajat, calificare\_angajat).

**Aducerea la FN5:**

Singurul tabel care poate avea probleme ar fi EFECTUARE\_CALATORIE(CNP, id\_avion, id\_calatorie). Cum CNP, id\_avion si id\_calatorie formeaza o cheie primara compusa, respective o cheie candidat, fiecare intrare este unica. Daca as descompune tabelul in 3 tabele diferite, as pierde informatie. Spre exemplu, daca pasagerul A calatoreste cu avionul B in calatoria C si daca am imparti tabelul astfel:

IMBARCARE(CNP, id\_avion)

AVION\_CALATORIE(id\_avion, id\_calatorie)

EFECTUARE\_CALATORIE(CNP, id\_calatorie)

Atunci am sti ca pasagerul A se imbarca in avionul B, avionul B efectueaza calatoria C, iar pasagerul A efectueaza calatoria C. Totusi, cand unim tabelele, nu putem relata cu exactitate informatia din tabelul initial, deoarece poate mai exista un alt avion care il ducea pe calatorul A in calatoria C.

Daca, insa, s-ar fi adaugat o calatorie noua , lucrurile se schimba. Daca nu stim nimic despre avionul utilizat si pasagerii care s-ar imbarca, ar trebui sa avem NULL la id\_avion si CNP, dar trebuie sa avem date, fiind cheie primara. Asadar, pentru a normaliza in FN5, tabelul trebuie impartit in 3, folosindu-ne de schema relationala de mai sus.

## b)Denormalizare

Incepem cu denormalizarea din FN5 in FN4, unind cele 3 tabele din schema relationala de mai sus in tabelul original, EFECTUARE\_CALATORIE. Desi am avea join dependency, modelul FN5 este utilizat extrem de rar intr-o baza de date, dar ajuta si la o performanta ridicata, reducand numarul de join-uri In query-uri si prin simplificarea acestora daca am vrea sa selectam date intre AVION, CALATORIE si PASAGER.

Denormalizarea din FN4 in BCNF este benefica pentru a retine mai multe date in mai putine tabele, realizandu-se accesarea acestora intr-un mod mai facil. Cum tabelele mele, mai putin cele asociative nu au chei primare compuse, iar tabelele asociative nu au alte date, pot considera ca am denormalizat la BCNF.

Denormalizarea din BCNF in FN3 relaxeaza conditiile din schema relationala, putand exista mai multe atribute care nu depind de cheia candidat, ci de un atribut non-prim. Nu voi face o normalizare specifica din BCNF in FN3.

Denormalizand catre FN2, putem avea depdente tranzitive, care ar duce la reducerea numarului de entitati din tabel. Putem uni AVION si MODEL\_AVION in AVION(Id\_avion, tip\_avion, intretinere, nr\_locuri, data\_reviziei). Putem uni PASAGER si ABONAMENT in PASAGER(CNP, nume, prenume, email, nume\_abonament, puncte), unde nume\_abonament ar depinde de ID, dar punctele depind de abonament. Putem uni AEROPORT si LOCATIE in tabelul AEROPORT(id\_aeroport, nume\_aeroport, id\_locatie, nume\_locatie), unde id\_locatie nu este cheie primara, dar tot este unic.

Tabelele ANGAJAT, JOB, CERTIFICAT, CAMPANIE, AEROPORT ar putea deveni un tabel foarte mare, care imbina schemele relationale ale tuturor acestor tabele. Se elimina tabelul PARTICIPARE. Nota: id-urile job-urilor, certificatului, campaniei, aeroportului nu vor face parte din cheia primara, intrucat ar fi considerat in forma BCNF.

Pentru accesarea informatiei, ne putem folosi de indexing si de query-uri simplificate pentru eficienta de timp.

Denormalizand catre FN1, putem sa introducem acum date multiple in cadrul unui atribut. Asta ar ajuta in cazuri specifice, in care putem cere mai multe date specifice unei singure entitati. Spre exemplu, in tabelul denormalizat ANGAJAT(id\_angajat, nume, prenume, email, salariu, data\_angajarii, id\_job, id\_aeroport, campanii, certificate, id\_job, nume\_job, salariu\_minim), as adauga in cadrul atributului campanii toate informatiile legate de toate campaniile le care a participat angajatul si voi scrie in cadrul atributului certificate toate certificatele pe care le detine angajatul.

# 18) Tranzactii: reprezentarea Consistency Levels in Oracle SQL

O sa am doua conexiuni separate care ruleaza in acelasi timp pe proiect.

In prima, vor fi urmatoarele linii de cod:

drop table ang;

create table ang as select \* from angajat;

--dirty write

--1

--6000,9000,10000,9000

UPDATE ang SET salariu = salariu + 1000 WHERE lower(nume) = 'popescu';

--7000,10000,11000,10000

--3

rollback;

--5

SELECT nume, prenume, salariu

FROM ang

WHERE lower(nume)='popescu';

--8000,11000,12000,11000

--lost update

--1

SELECT nume, prenume, salariu

FROM ang

WHERE lower(nume)='popescu';

UPDATE ang SET salariu=salariu+2000 WHERE lower(nume)='popescu';

--3

commit;

--5

SELECT nume, prenume, salariu

FROM ang

WHERE lower(nume)='popescu';

--11000,14000,15000,14000

--dirty read

--1

SELECT salariu, id\_job, id\_aeroport FROM ang WHERE id\_angajat=300;

UPDATE ang

SET id\_job=4

WHERE id\_angajat=300;

commit; --intra in deadlock fara commit

--3

rollback;

SELECT salariu, id\_job, id\_aeroport FROM ang WHERE id\_angajat=300;

--non-repeatable reads: posibil

--1

SELECT salariu, id\_job, id\_aeroport FROM ang WHERE id\_angajat=300;

--30000, null, 17

UPDATE ang

SET id\_job=6

WHERE id\_angajat=300;

--3

commit;

--non-repeatable reads: imposibil

--1

SELECT salariu, id\_job, id\_aeroport FROM ang WHERE id\_angajat=300;

--30000,6,17

UPDATE ang

SET id\_job=5

WHERE id\_angajat=300;

--3

commit;

--phantom possible

--2

INSERT INTO ang(id\_angajat, nume, prenume, email, salariu) VALUES(400,'Marculescu', 'Alina', 'ama@yahoo.com',70000);

commit;

--phantom possible

--2

INSERT INTO ang(id\_angajat, nume, prenume, email, salariu) VALUES(401,'Marculescu1', 'Alina1', 'ama1@yahoo.com',80000);

SELECT avg(salariu) FROM ang;

--26800

commit;

In a doua, urmatoarele:

--dirty write

--2

UPDATE ang

SET salariu=2000+salariu

WHERE lower(nume)='popescu';

--4

SELECT nume, prenume, salariu

FROM ang

WHERE lower(nume)='popescu';

commit;

--lost update

--2

UPDATE ang SET salariu=salariu+1000 WHERE lower(nume)='popescu';

--4

SELECT nume, prenume, salariu

FROM ang

WHERE lower(nume)='popescu';

commit;

--dirty read

--2

UPDATE ang

SET salariu=30000

WHERE id\_angajat=300;

SELECT salariu, id\_job, id\_aeroport FROM ang WHERE id\_angajat=300;

--4

SELECT salariu, id\_job, id\_aeroport FROM ang WHERE id\_angajat=300;

commit;

--non-repeatable reads: posibil

--2

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SELECT salariu, id\_job, id\_aeroport FROM ang WHERE id\_angajat=300;

--4

UPDATE ang

SET salariu=40000

WHERE id\_angajat=300;

SELECT salariu, id\_job, id\_aeroport FROM ang WHERE id\_angajat=300;

--40000,6,17

rollback;

--non-repeatable reads: imposibil

--2

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT salariu, id\_job, id\_aeroport FROM ang WHERE id\_angajat=300;

--4

--nu se poate serializa:eroare

UPDATE ang

SET salariu=40000

WHERE id\_angajat=300;

SELECT salariu, id\_job, id\_aeroport FROM ang WHERE id\_angajat=300;

commit;

--30000,6,17

--phantom possible

--1

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

select avg(salariu) from ang;

--14750

--3

SELECT avg(salariu) FROM ang;

--20888.88

commit;

--phantom possible

--1

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT avg(salariu) FROM ang;

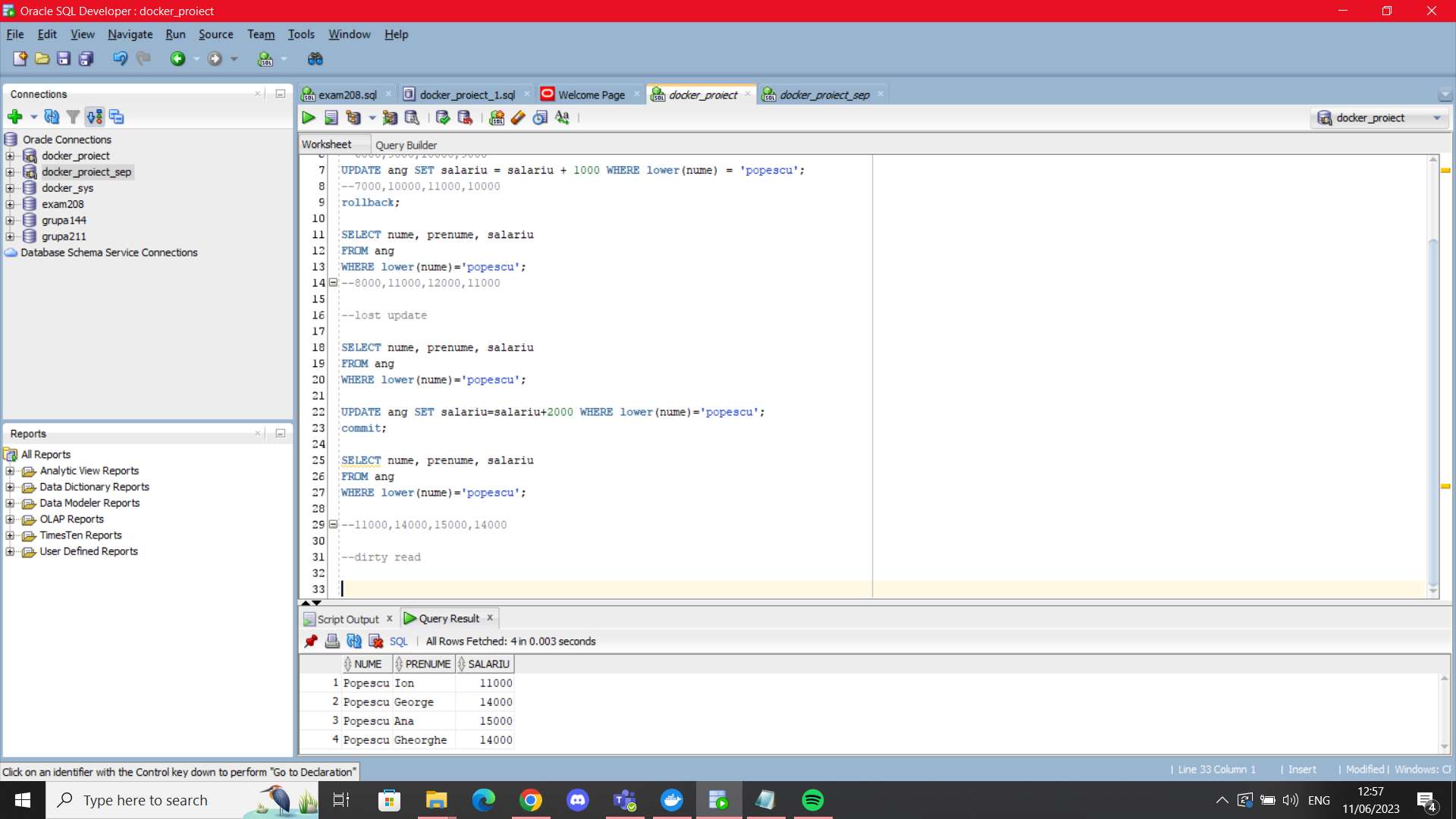
--20888.88

--3

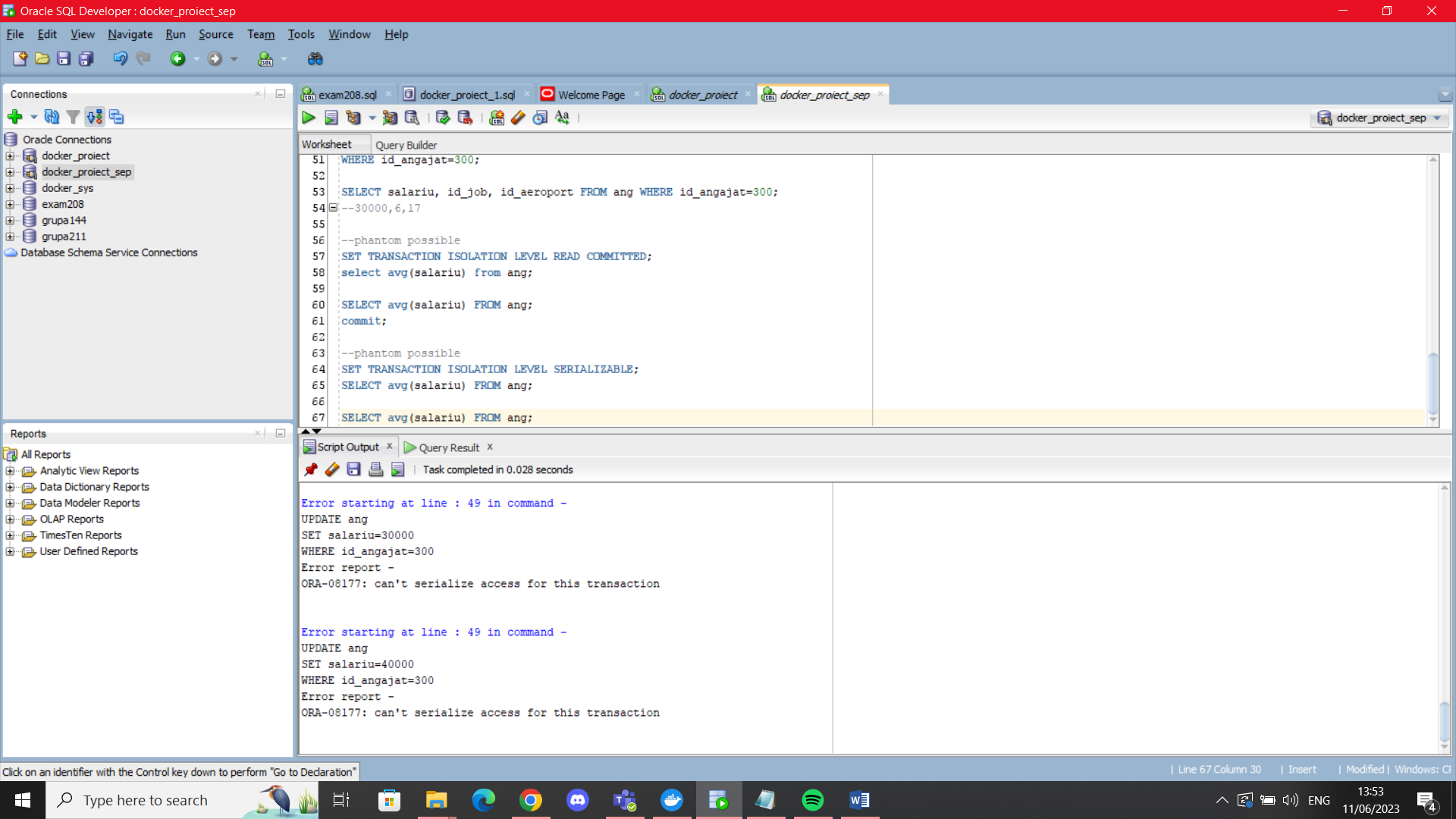
SELECT avg(salariu) FROM ang;

--20888.88

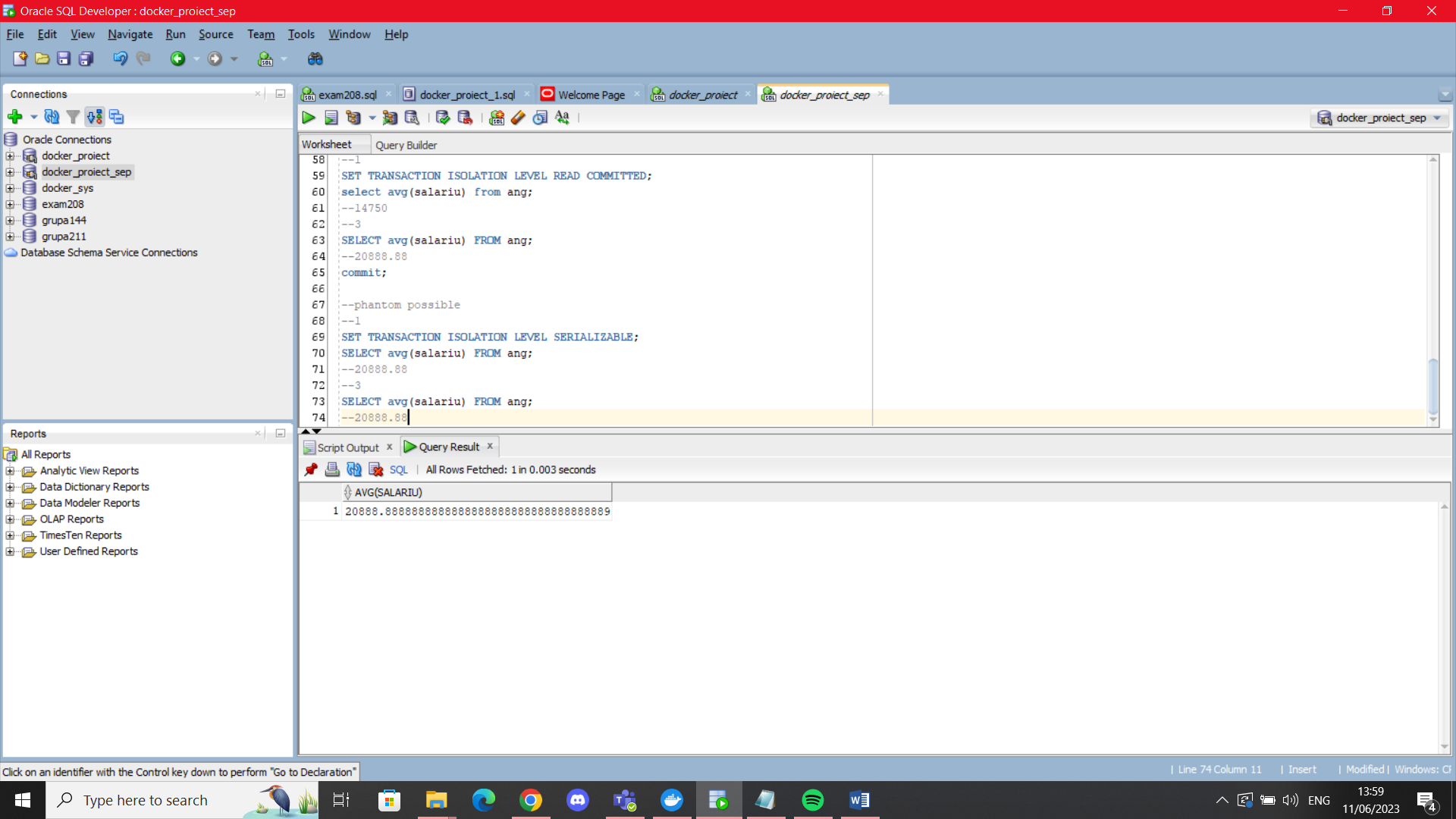
*Nota: comentariile cu cifre reprezinta ordinea de executie a tranzactiilor pentru ilustrarea fiecarui concept.*











# 19) Optimizarea a doua cereri folosind Indexing

Se dau doua cereri:

i) Selectati id-ul si e-mailul angajatilor care au numele de familie Popescu si au fost angajati dupa ce a fost primit ultimul certificat de catre un angajat care are codul mai mare decat 22.

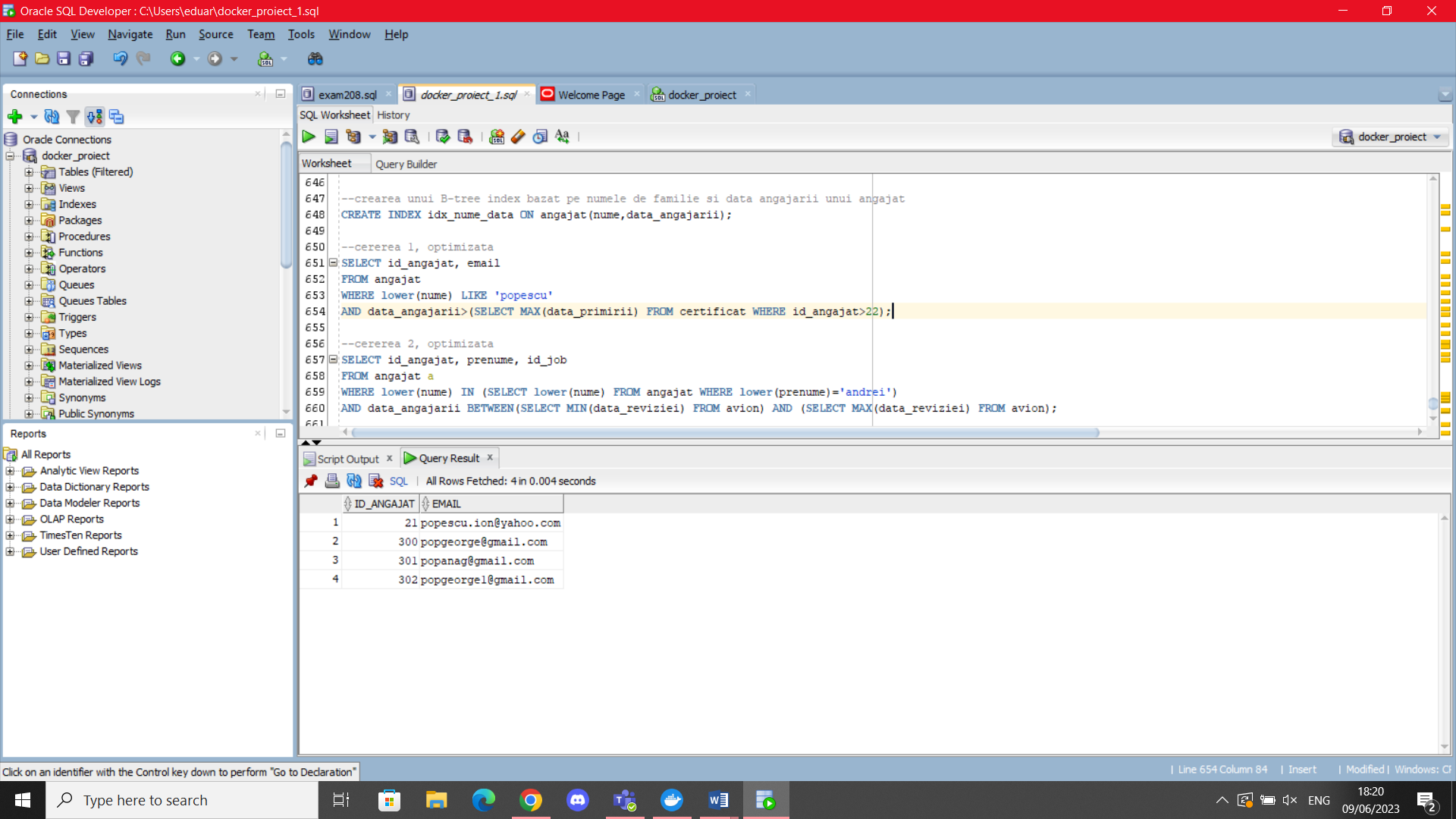
--cererea 1, inainte de indexare

SELECT id\_angajat, email

FROM angajat

WHERE lower(nume) LIKE 'popescu'

AND data\_angajarii>(SELECT MAX(data\_primirii) FROM certificat WHERE id\_angajat>22);



ii) Selectati codul, numele si codul jobului unui angajat care are prenumele andrei si data angajarii intre cea mai indepartata si cea mai apropiata data a reviziei a oricarui avion.

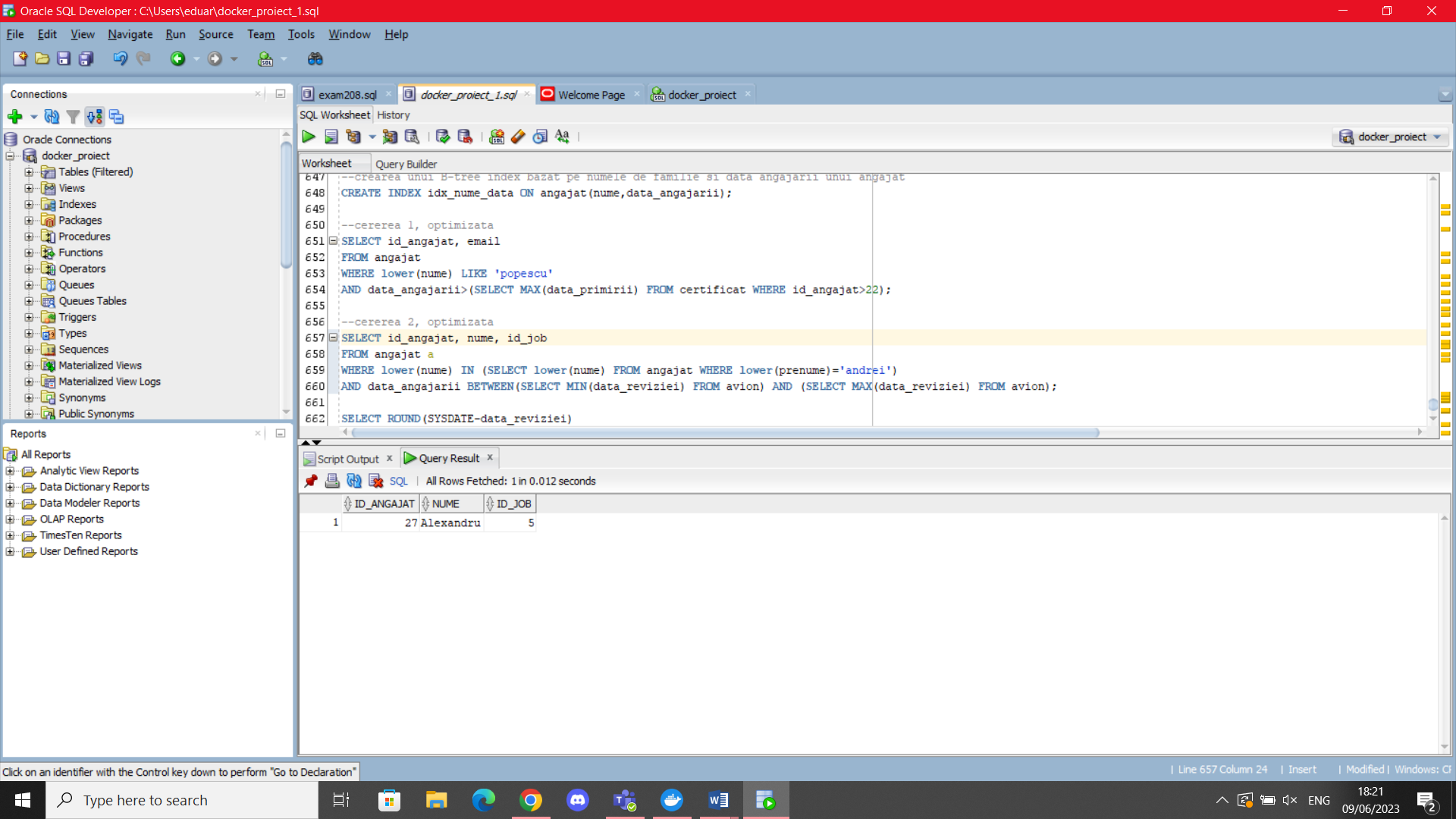
--cererea 2, inainte de indexare

SELECT id\_angajat, nume, id\_job

FROM angajat a

WHERE lower(nume) IN (SELECT lower(nume) FROM angajat WHERE lower(prenume)='andrei')

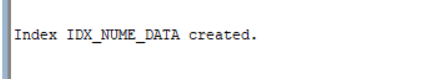
AND data\_angajarii BETWEEN(SELECT MIN(data\_reviziei) FROM avion) AND (SELECT MAX(data\_reviziei) FROM avion);



Aceste doua cereri se folosesc de numele angajatilor si data angajarii acestora. Pentru optimizarea acestora, voi crea in B-tree index pentru tuplul(nume, data\_angajarii) din tabelul ANGAJAT.

--crearea unui B-tree index bazat pe numele de familie si data angajarii unui angajat

CREATE INDEX idx\_nume\_data ON angajat(nume,data\_angajarii);



--cererea 1, optimizata

SELECT id\_angajat, email

FROM angajat

WHERE lower(nume) LIKE 'popescu'

AND data\_angajarii>(SELECT MAX(data\_primirii) FROM certificat WHERE id\_angajat>22);

--cererea 2, optimizata

SELECT id\_angajat, prenume, id\_job

FROM angajat a

WHERE lower(nume) IN (SELECT lower(nume) FROM angajat WHERE lower(prenume)='andrei')

AND data\_angajarii BETWEEN(SELECT MIN(data\_reviziei) FROM avion) AND (SELECT MAX(data\_reviziei) FROM avion);