**Cuprins**

[1. Prezentarea modelului si a regulilor sale 1](#_Toc775)

[1.1. Constrângeri pentru implementare 2](#_Toc21124)

[1.2. Diagrama Entitate Relație 3](#_Toc17116)

[1.3. Diagrama Conceptuală 4](#_Toc18141)

[1.4. Regulile de securitate 5](#_Toc10100)

[2. Procesele aplicatiei 6](#_Toc15566)

[2.1. Matricea Proces-Utilizator 7](#_Toc8333)

[2.2. Matricea Entitate-Proces 8](#_Toc28246)

[2.3. Matricea Entitate-Utilizator 9](#_Toc11421)

[3. Gestiunea Utilizatorilor si a Resurselor Computationale 10](#_Toc723)

[3.1. Configurarea Utilizatorilor si a Schemei 10](#_Toc30012)

[3.2. Memorie alocata pentru cateogriile de utilizatori 11](#_Toc1617)

[3.3. Profile 11](#_Toc982)

[3.4. Permisii Admin 13](#_Toc31723)

[4. Creare Bazei 14](#_Toc30959)

[4.1. Crearea Schemei Admin 14](#_Toc27784)

[4.1.1. Criptare in Schema Admin 15](#_Toc17023)

[4.2. Crearea Schemei Antrenor 16](#_Toc7236)

[4.2.1. Criptare in Schema Antrenor 17](#_Toc29230)

[5. Obiect dependent 19](#_Toc9270)

[6. Audit 21](#_Toc6779)

[6.1. Audit Standard 21](#_Toc23427)

[6.2. Triggeri de Auditare 23](#_Toc4628)

[6.3. FGA 23](#_Toc4566)

[7. Contextul aplicatiei 26](#_Toc14052)

[7.1. VPD 26](#_Toc28334)

[8. SQL injection 29](#_Toc22181)

[8.1. Procedura Vulnerabila 29](#_Toc9261)

[8.2. Procedura reparta 31](#_Toc20732)

[9. Mascarea datelor 32](#_Toc22857)

[9.1. Export 33](#_Toc26466)

[9.2. Import 34](#_Toc3722)

**1. Prezentarea modelului si a regulilor sale**

Proiectul implementează gestiunea mai multor filiale dintr-un lanț de săli de fitness. Abonamentul unui client este valabil în toate filialele, iar, de asemenea, clienții pot cumpăra suplimente nutritive de la recepția tuturor filialelor (nu se iau în calcul alte tranzacții pe care clientul le face la recepție (e.g. cumpără apă)) contorizăm doar comenzile efective de suplimente nutritive ale clientului). Modalitatea de plată a serviciilor și comenzilor nu este reținută în baza de date. Fiecare filială va avea angajați, aceștia putând fi antrenori sau recepționiști. Totodată, se rețin echipamentele pentru fiecare sediu, iar fiecare angajat lucrează doar într-un singur sediu. Echipamentele și suplimentele nutritive vor avea neapărat cel puțin un furnizor.

Clienții pot avea abonamente lunare, trimestriale, bianuale, anuale sau extinse. Despre clienți se vor înregistra numele, prenumele, vârsta, un email, dacă este student sau nu și posibil unul sau mai multe numere de telefon. De asemenea, un client va urma numai un unic program de antrenament de un anumit tip.

Fiecare filială a sălii are un anumit număr de angajați (antrenori sau recepționiști). Un antrenor poate avea unul sau mai multe programe de antrenament și este obligat să-și ateste studiile. Recepționiștii se vor ocupa de comenzi și vor avea fie program complet fie cu normă redusă. Pentru fiecare angajat se va ține minte numele, prenumele, vârsta, posibil unul sau mai multe numere de telefon, un email, data angajării și salariul (în lei). În fiecare filială va fi un unic manager.

Echipamentele sportive aparțin unei singure filiale, iar numele efectiv al acestora nu depinde de furnizor (e.g. o presă furnizată de X se va numi tot presă dacă este furnizată și de Y). Se vor ține minte data instalării echipamentelor, ultima revizie (data primei revizii va coincide cu data instalării) și numele.

Suplimentele se pot comanda doar de la recepție și numele lor nu depinde de furnizor (e.g. proteina furnizată de X se va numi tot proteină dacă este furnizată și de Y). Se vor ține minte numele, o descriere, caloriile (pe 100g), prețul și furnizorul.

Pentru fiecare filiala se păstrează angajații și echipamentele. De asemenea, se vor reține data de înființare, adresa și numele. Pentru furnizori se vor salva adresa, codul fiscal și numele.

**1.1. Constrângeri pentru implementare**

Fiecare client poate avea un singur abonament

Un client nu poate fi și angajat.

Abonamentul unui client este unic pentru toate filialele.

Tipurile de abonamente sunt: lunar, trimestrial, bianual, anual, extins.

Suplimentele se țin pe toată firma, nu pe filiale.

Un client trebuie să urmeze numai un program de antrenament de un anumit tip.

Tipurile de programe sunt: Mass, Body, Recovery.

Emailul este unic pentru fiecare persoană.

Un angajat poate lucra doar la o singură filială.

Suplimentele nutritive/Echipamentele pot avea același id pentru furnizori diferiți.

Nu se ține minte metoda de plată pentru abonamente și comenzi

Se rețin doar comenzile efective de suplimente de la recepție ( de exemplu nu se rețin plățile pentru apa ).

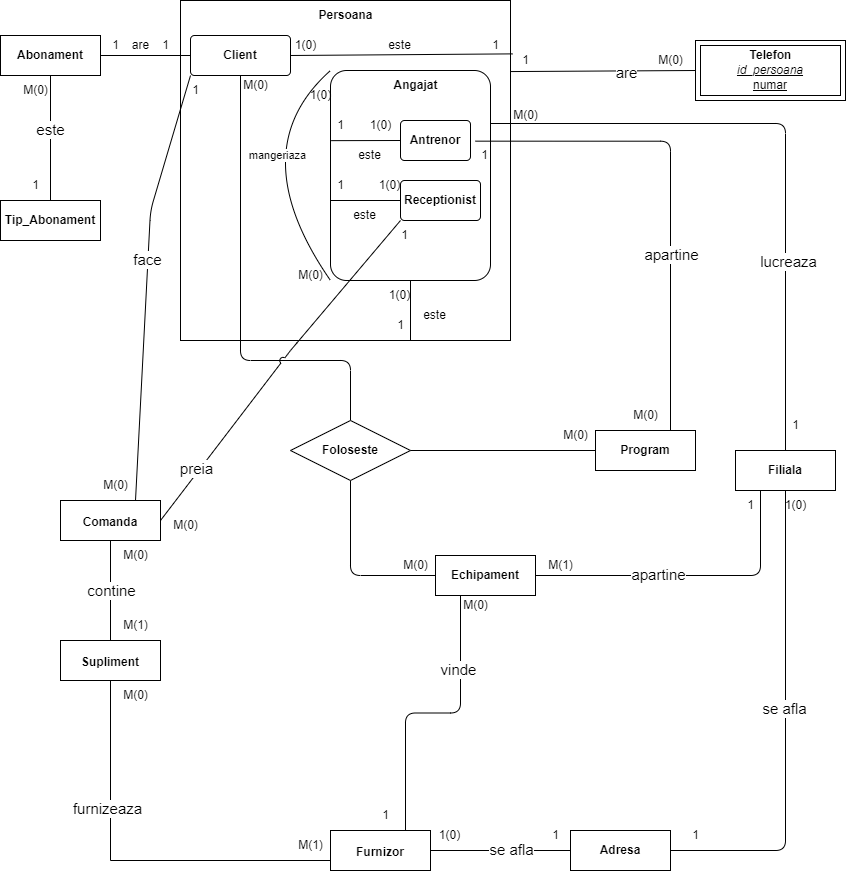
O aprovizionare a firmei se va face cu un singur supliment o dată.

Nu se înregistrează stocuri de suplimente.

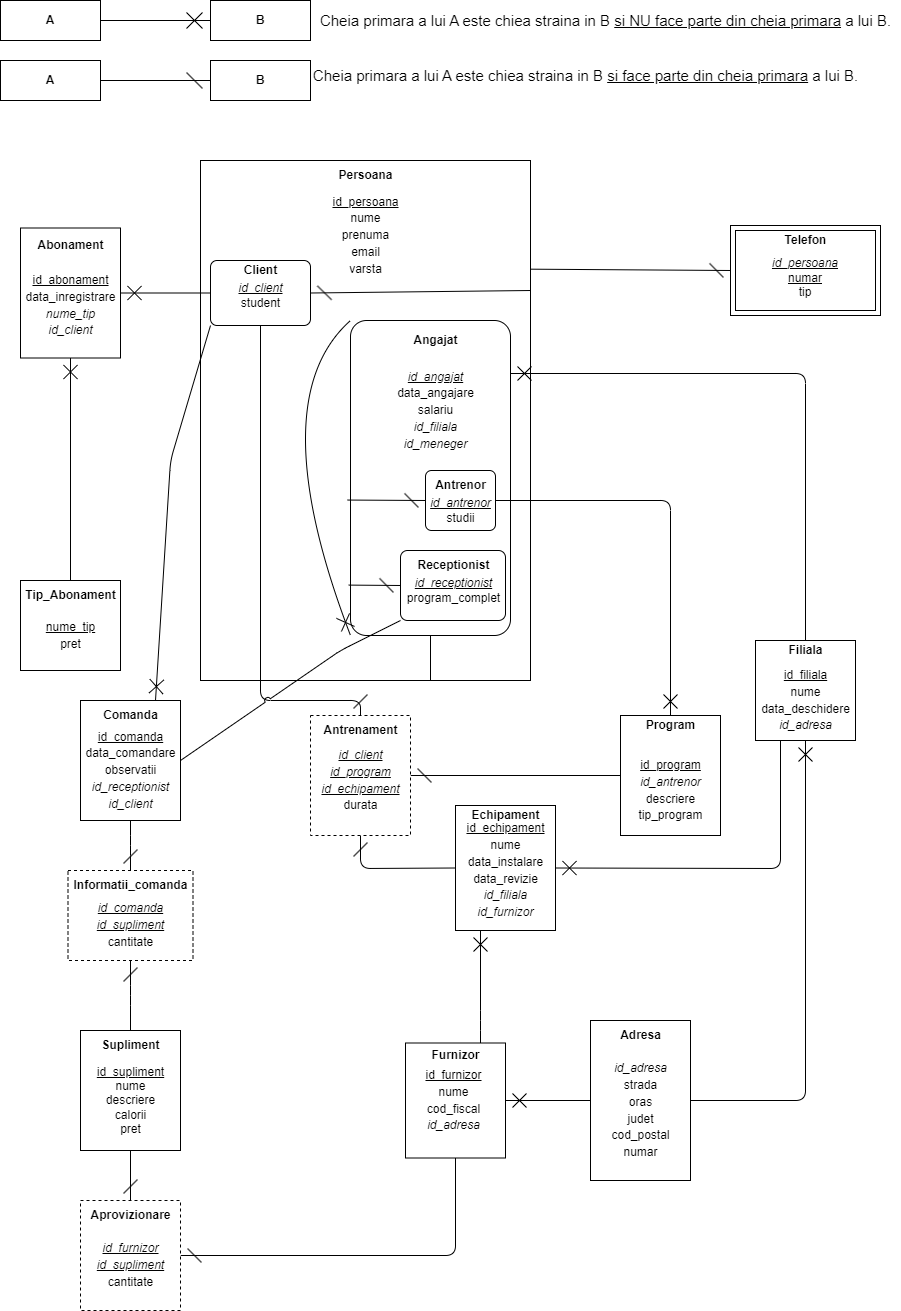
Filialele trebuie să aibă adresele diferite între ele.

Furnizorii trebuie să aibă adresele diferite între ei.

**1.2. Diagrama Entitate Relație**



**1.3. Diagrama Conceptuală**



Persoana(id\_persoana, nume, prenume, email, varsta)

Client(*id\_client*, student) {id\_client este și cheie străina către tabelul Persoana}

Angajat(*id\_angajat*, data\_angajare, salariu, *id\_filiala*, *id\_meneger*) {id\_angajat este și cheie străina către tabelul Persoana, id\_meneger este cheie străina tot către tabelul Angajat}

Antrenor(*id\_antrenor*, studii) {id\_antrenor este și cheie străina către tabelul Angajat}

Receptionist(*id\_receptionist*, program\_complet) {id\_receptionist este și cheie străina către tabelul Angajat}

Telefon(*id\_persoana*, numar, tip)

Abonament(id\_abonament, data\_inregistrare, *nume\_tip*, *id\_client*)

Tip\_Abonament(nume\_tip, pret)

Filiala(id\_filiala, nume, data\_deschidere, *id\_adresa*)

Adresa(id\_adresa, strada, oras, judet, cod\_postal, numar)

Furnizor(id\_furnizor, nume, cod\_fiscal, *id\_adresa*)

Program(id\_program, *id\_antrenor*, descriere, tip\_program)

Echipament(id\_echipament, nume, data\_instalare, data\_revizie, *id\_filiala*, *id\_furnizor*)

Antrenament(*id\_client*, *id\_program*, *id\_echipament*, durata)

Comanda(id\_comanda, data\_comandare, observatii, *id\_receptionist*, *id\_client*)

Informatii\_Comanda(*id\_comanda*, *id\_supliment*, cantitate)

Supliment(id\_supliment, nume, descriere, calorii, pret)

Aprovizionare(*id\_furnizor*, *id\_supliment*, cantitate)

**1.4. Regulile de securitate**

In cadrul proiectului se vor cripta datele antrenamentelor fiecarui client, astfel incat doar el si antrenorul sau sa poata vedea informatiile. De asemenea, se vor realiza doua VPD-uri (Virtual Private Database) pentru a asigura ca managerii de filiala pot face operatii DML doar pe echipamentele din filala lor si ca pot vedea doar istoricul echipamentelor care sunt sau au fost in acea filiala. De asemenea, sa vor respecta permisiile din matricea entitate-utilizator si se vor da cote de memorie conform necesitatilor de stocare.

**2. Procesele aplicatiei**

Procesele care pot fi initiate in cadrul bazei sunt:

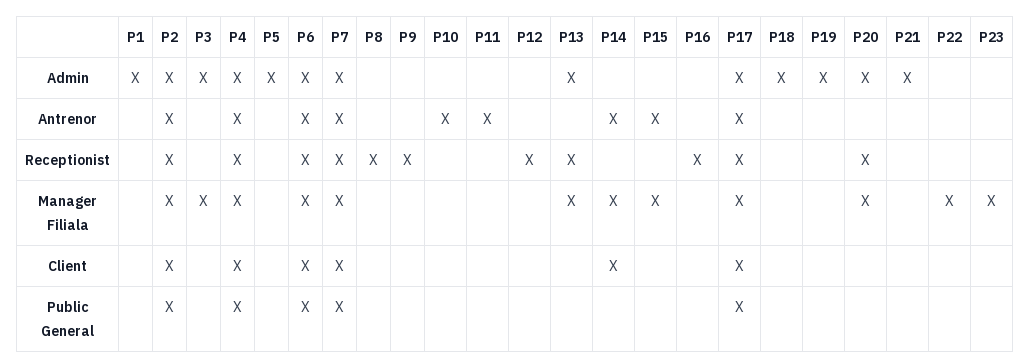
1. Configurarea angajati
2. Vizualizarea antrenorilor dintr-o filiala
3. Vizualizarea receptionistilor dintr-o filiala
4. Vizualizarea programelor de antrenament pentru un anumit antrenor
5. Delegarea/Revocarea de manager pentru o filiala
6. Vizualizarea caracteristicilor unei filiale
7. Vizualizarea echipamentelor dintr-o filiala
8. Manageiriere clienti client
9. Configurare abonament pentru un client
10. Configurarea programelor pentru un antrenor
11. Configurarea antrenamentelor pentru un antrenor
12. Verficare validitate abonament pentru un client
13. Vizualizarea clientilor
14. Vizualizarea antrenamentelor unui client
15. Vizualizarea antrenamentelor unui antrenor
16. Creare comanda
17. Vizualizarea suplimentelor puse la vanzare
18. Vizualizarea tuturor comenzilor
19. Vizualizarea comenzilor pentru un client
20. Vizualizarea aprovizionarilor suplimente
21. Gestiune aprovizionari suplimente
22. Gestiune echipamente dintr-o filiala
23. Vizualizarea tutoror antrenamentelor dintr-o filiala

Matricea Proces Utilizator

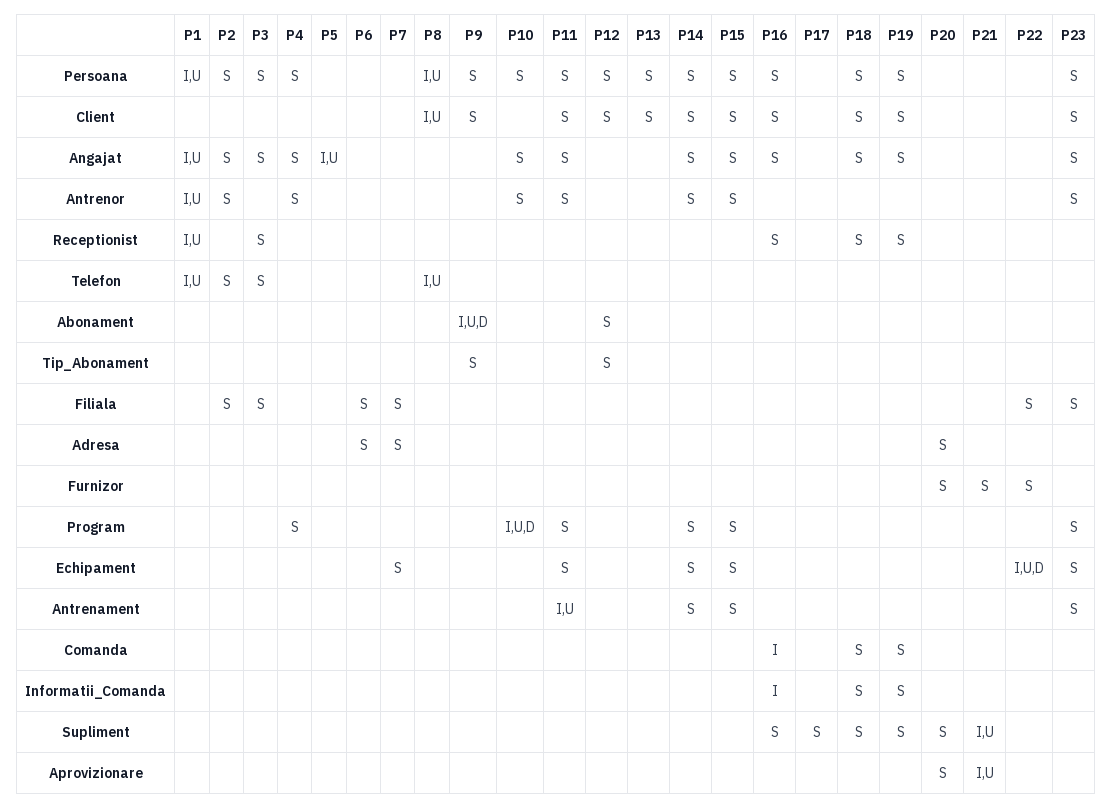
In cadrul proiectului se disting patru categorii distincte de utilizatori:

1. Admin - cel care gestioneaza aplicatia
2. Antrenor - angajatul de tip antrenor din cadrul unei sali
3. Receptionist - angajatul de tip receptionist din cadrul unei sali
4. Manger Filila - managerul unei singure filiale
5. Client - client pe intregul lant de sali
6. Public General

**2.1. Matricea Proces-Utilizator**

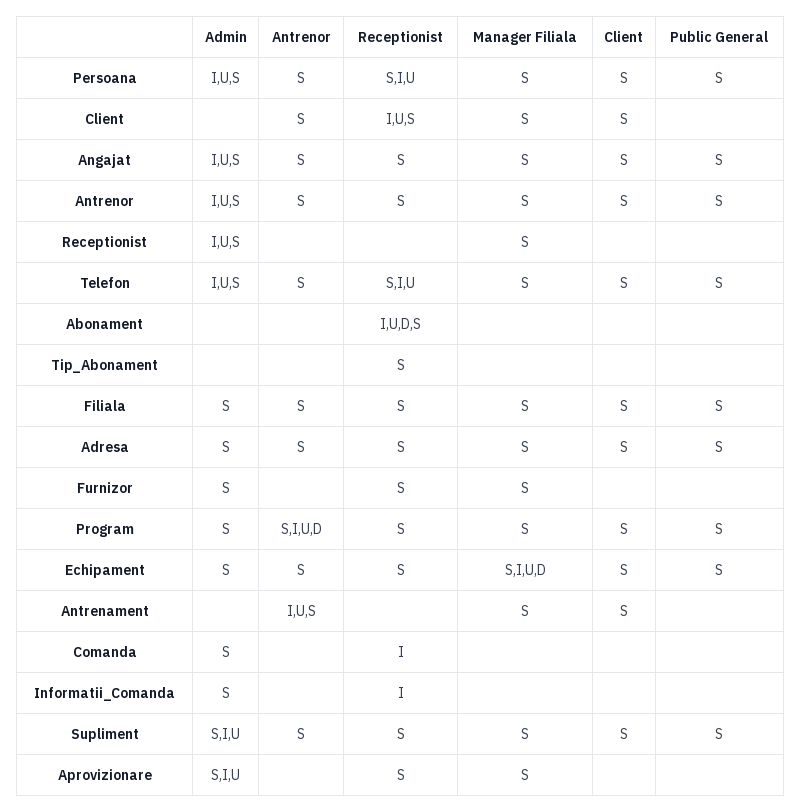


**2.2. Matricea Entitate-Proces**



Legenda: S = Select; I = Insert; U = Update; D = Delete

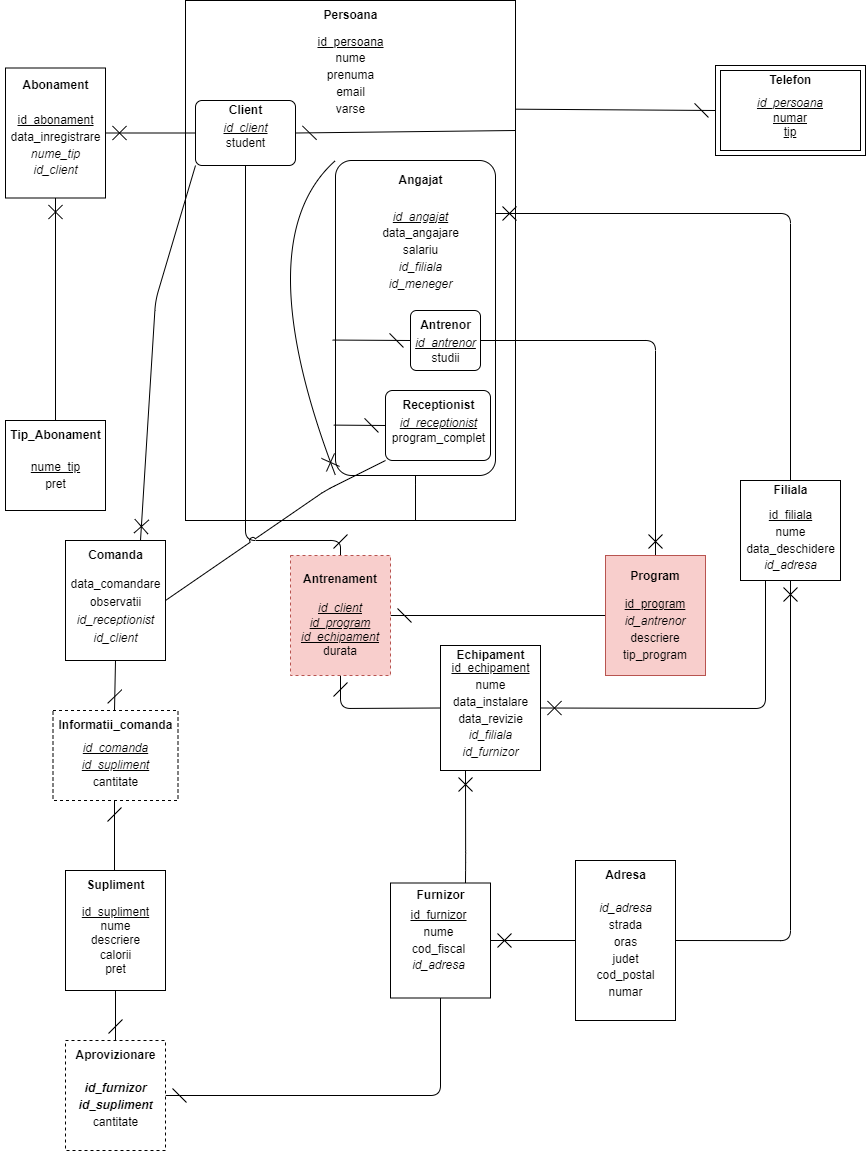
**2.3. Matricea Entitate-Utilizator**



Legenda: S = Select; I = Insert; U = Update; D = Delete

1. **Gestiunea Utilizatorilor si a Resurselor Computationale**

**3.1. Configurarea Utilizatorilor si a Schemei**



Din cadrul diagramei conceptuale, toate tabelele vor fi crete in schema adminului, mai putin Program si Antrenament care vor fi create separat de fiecare antrenor in schema lui proprie.

**3.2. Memorie alocata pentru cateogriile de utilizatori**

- Deoarece majoritatea bazei de date va fi creata in schema adminului, acesta va avea la dispozite 500MB

- Fiecare antrenor va avea la dispozitie 10MB pentru a crea obiecte

- Intrucat restul utilizatorilor nu vor crea obiecte acestia nu vor avea memorie alocata

**3.3. Profile**

In cadrul aplicatiei, parolele trebuie sa contine minim un “\_” . Aceasta conditie va fi verficiata in cadrul fieacrui profil utilizand optiunea *password\_verify\_function*.

Pentru admin se permit mai multe sesiuni, sunt permise 15 minute de idle per sesiune, parola trebuie schimbata o data la 90 de zile, sunt permise 5 greseli consecutive ale credentialelor si are o are dreptul la 6 minute maxime de cpu time per comanda.

Pentru publicul general se permit 6 sesiuni, sunt permise 5 minute de idle per sesiune, un maxim de 20 de minute per sesiune si dreptul la 1 minut maxim de cpu time per comanda.

Pentru restul tipurilor de unitilizatori se permite doar o sesiune, sunt permise 15 minute de ide, parola trebuie schimbata o data la 90 de zile, , sunt permise 5 greseli consecutive ale credentialelor si are dreptul la 2 minute maxime de cpu time per comanda.

Plan de consum

Pentru planul de consum, cand CPU-ul ajunge la 100%, in cadrul aplicatiei exista urmatoarea regula:

Admin: 30%

Public General: 5%

Clienti: 10%

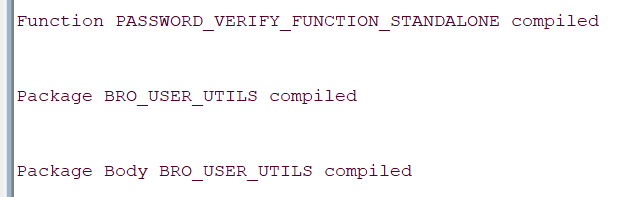
Manageri: 15%

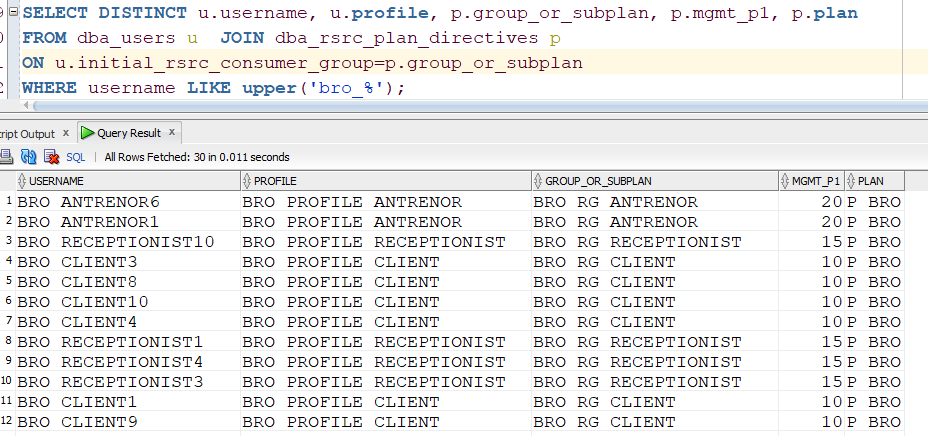
Receptionisti: 15%

Antrenori: 20%

Other Groups: 5%

Crearea utilizatorilor, mai putin a adminului, a fost realizata utilizand un pachet custom utilitar. Codul pentru acest pachet si configurarea initiala a utilizatorilor cu profile si resource group este prezenta in fisierul *sys\_users\_1.sql* . Mai jos sunt cateva outputuri din rularea acestui cod:





**3.4. Permisii Admin**

Tot in acest fisier initial de configurare sunt prezente si drepturile adminului.

**GRANT CREATE SESSION**

**TO bro\_admin;**

**GRANT CREATE ANY TABLE**

**TO bro\_admin;**

**GRANT CREATE ANY VIEW**

**TO bro\_admin;**

**GRANT CREATE ANY TRIGGER**

**TO bro\_admin;**

**GRANT CREATE ANY PROCEDURE**

**TO bro\_admin;**

**GRANT CREATE ANY SEQUENCE**

**TO bro\_admin;**

**GRANT CREATE ANY INDEX**

**TO bro\_admin;**

**GRANT CREATE ANY TYPE**

**TO bro\_admin;**

**GRANT CREATE TYPE**

**TO bro\_admin;**

**-- Pt proceduri**

**GRANT EXECUTE**

**ON dbms\_crypto**

**TO bro\_admin**

**WITH GRANT OPTION;**

**--Pt generated by default on null as identity la create in antrenor**

**GRANT SELECT ANY SEQUENCE**

**TO bro\_admin;**

**GRANT EXECUTE**

**ON get\_users\_by\_suffix**

**TO bro\_admin;**

Deoarece schemele antrenorilor se vor crea utilizand scriptul din fisierul *bro\_admin\_antrenor\_seed.sql,* adminul are nevoie de permisii speciale pentru a putea rula scriptul de creare cu succes (create/select any).

Permisia Select Any Sequence este necesara deoarce tabelele din schema antrenorilor vor avea cheile primare generate folosind autoincrementul din Oracle. Astfel, adminul are nevoie de permisia de selectare a acestei secvente generate la creare tabelului:

**id\_program number(\*, 0) generated BY DEFAULT ON NULL AS identity CONSTRAINT pk\_program PRIMARY KEY**

1. **Creare Bazei**

**4.1. Crearea Schemei Admin**

In cadrul adminului, adiacent tabelelor din schema conceptuala se vor implementa la inceput inca doua tabele, anume unul este de Logging al erorilor care apar in operatiile DML pe tablele sau viewurile de persoane ale schemei sale, iar celalalt este unul de mapare a unui utilizator creat cu un cont al bazei de date, astfel asigurand ca exista un cont inainte de a insera intr-un tabel al bazei. Pentru popularea celui de-al doilea tabel este nevoie de legatura dintre admin si useri si de aceea exista permisia de a executa functia *get\_users\_by\_suffix* din sys catre admin. Scriptul initial al bazei admin alaturi de inserarea unor date este prezent in fisierul *bro\_admin\_create\_tables.sql*.

Structura tableului de mapari de conturi este urmatoarea:

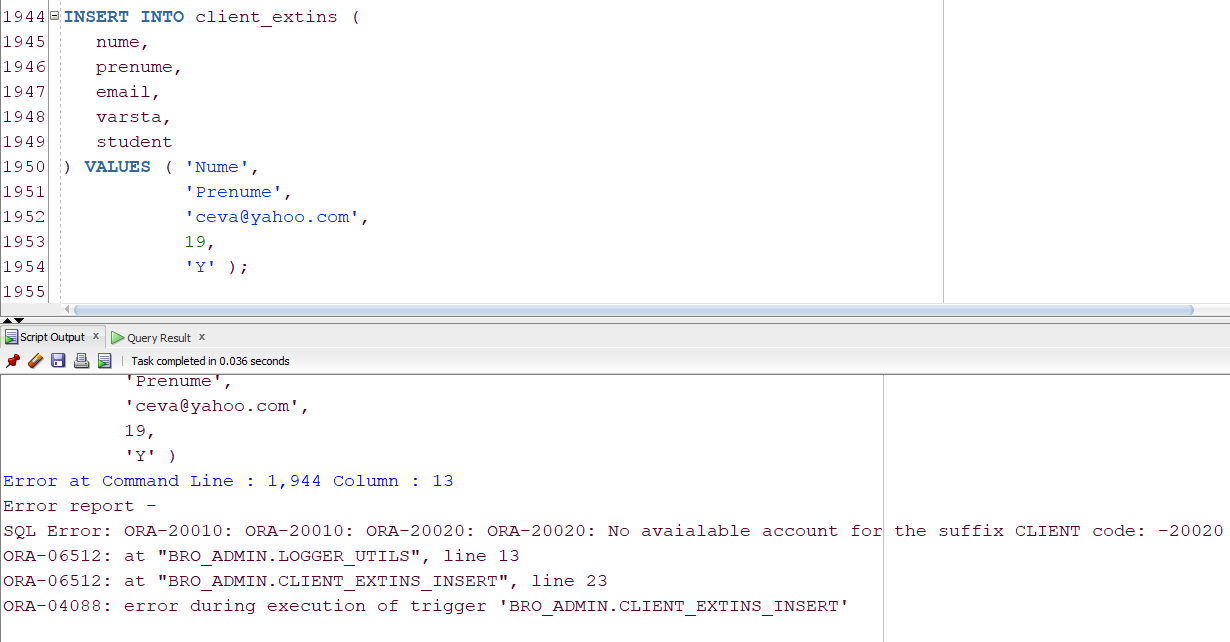
**CREATE TABLE account\_mapping(**

**id\_persoana NUMBER(\*,0) CONSTRAINT pk\_account\_mapping PRIMARY KEY,**

**username VARCHAR2(128) UNIQUE**

**);**

Acesta are cheia primara cea din tabela persoana, la momentul respectiv, care are contul si va tine minte si usernameul asociat acesteia. Inserarile persoanelor in baza de date se vor face folosind viewurile specifice fiecarui tip, iar in trigerii “instead of” se va apela functia *insert\_into\_account\_mapping* care asigura ca exista conturi ale bazei de date pentru tipul de persoana dorit a fi inserat. De exemplu, pentru clienti, daca am utilizat toate contrurile si dorim sa mai adaugam un client vom primi eroarea:

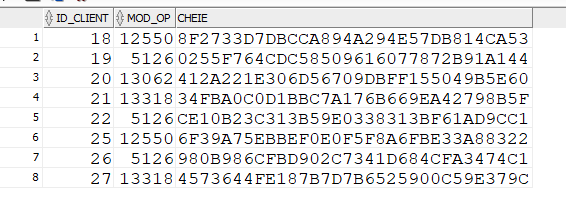


**4.1.1. Criptare in Schema Admin**

Intrucat sciptul de seed pentru antrenori include si partea de criptare corespunzatoare lor, inainte de a rula acel script voi prezenta criptarea.

Asadar, in cadrul aplicatiei, criptarea sa va face pentru clienti la nivelul antrenamentelor unui antrenor, astfel incat un client sa-si poate vedea doar propriile antrenamentele. Pentru aceasta, in schema admin vom crea mai multe obiecte, scriptul asociat acestora se gaseste in fisierul *bro\_admin\_critptare.sq*.

In acest script este prezent un tabel numit *chei\_client* care pentru fiecare client pastreaza modul de cripatare si cheia asociata. La inserarea in tabelu de chei sa va folosi algoritmul AES128, iar paddingul va fi ales random pentru fiecare client, dintre PKCS5 si PADZERO, iar tot random pentru fiecare client si va alege si chainingul. Dupa ce adminul a inserat pentru fiecare *id\_client* in *chei\_client,* vom avea:

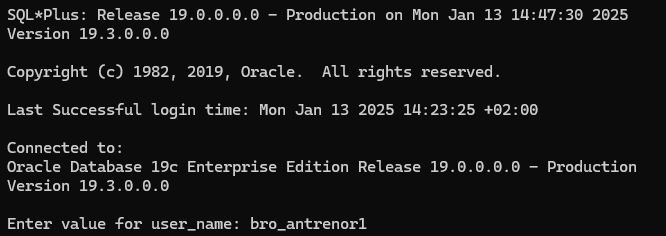


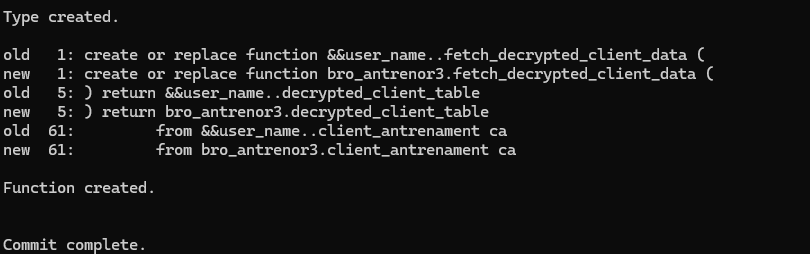
Pentru ca un client sa-si ia cheia unica, tot in cadrul scriptului este creata o functie *get\_client\_key* care pentru usernameul luat din contextul **userenv** va intoarce cheia de criptare asociata.

**4.2. Crearea Schemei Antrenor**

Acum, pentru a ilustra criptarea vom rula scriptul din fisierul *bro\_antrenor\_seed.sql.* Pentru a usura inserarea, in cadrul acestui fisier schema este data ca un argument, de exemplu: **create table &&user\_name..program**

Tot in cadrul seedului se vor da antrenorilor permisiile necesare crearii obiectelor. De asemenea, pentru a rula scriptul sa va folosi fisierul *seed\_antrenor.cmd*. La rularea acestui fisier sa va introduce numele antrenorului:



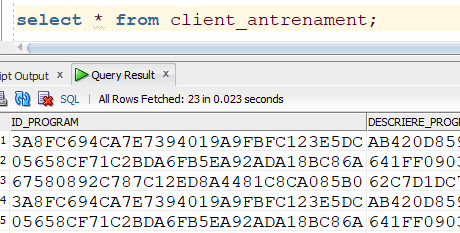


**4.2.1. Criptare in Schema Antrenor**

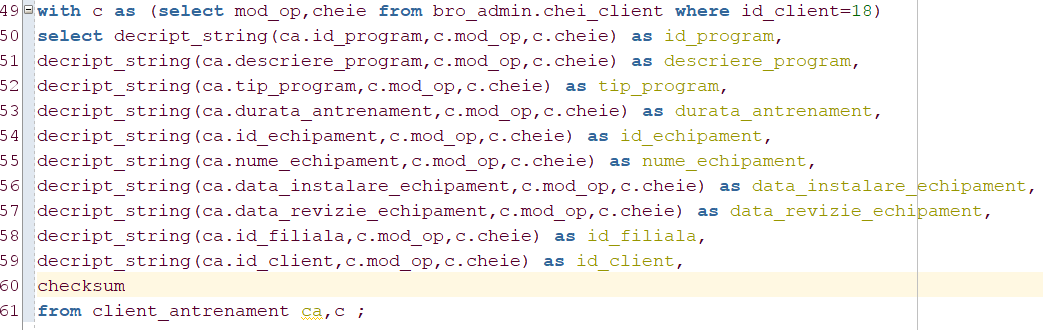
Intrucat antrenorii trebuie sa aiba acces la cheile de cripatre ale clientilor, li se va da acces la tabelul de chei, iar totodata scriptul creeaza si un view care va arata datele antrenamentelor pentru clienti intr-un mod cripatat cu cheile individuale. Pentru a regasi datele si a le valida sunt create mai multe functii si obiecte utilitare pe care un client le poate folosi.

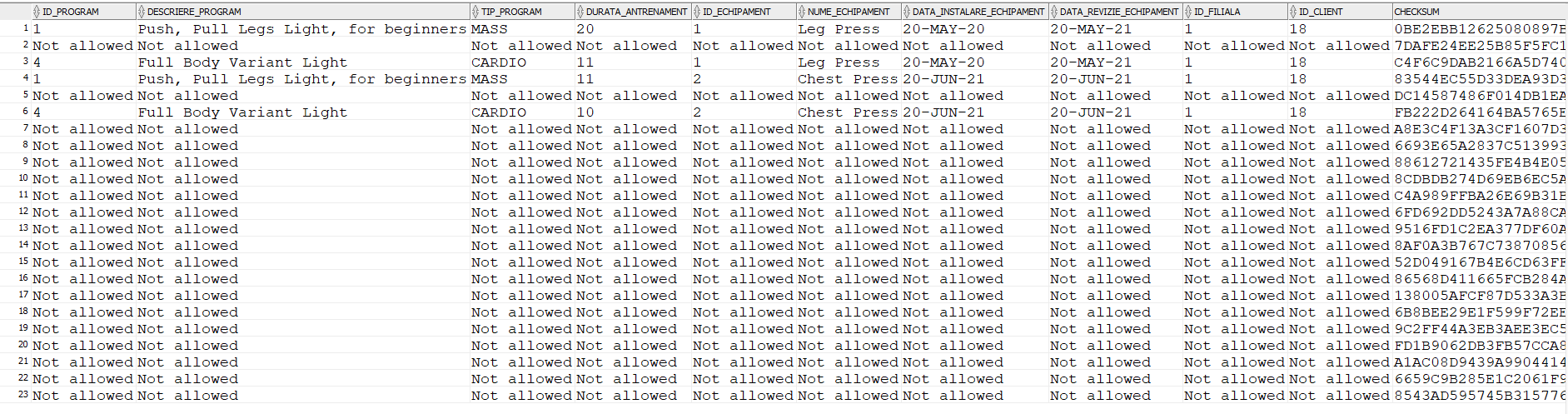
Pentru a putea insera date vom rula din antrenor1 scriptul *bro\_antrenor\_insert.sql*.

Pentru a vedea datele cripate vom apela:

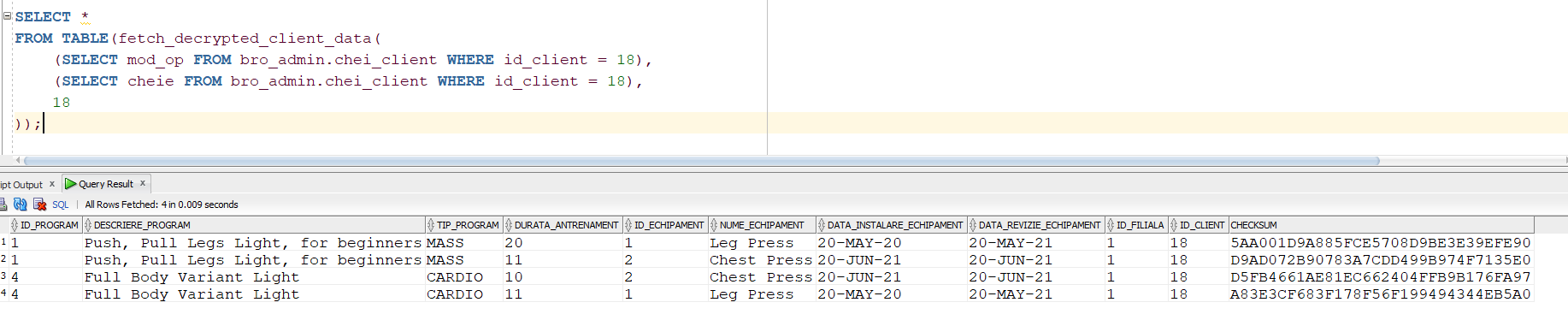


De exemplu, daca dorim sa decriptam din tabel totul, dar doar pentru clientul cu idul 18, outputul va arata:





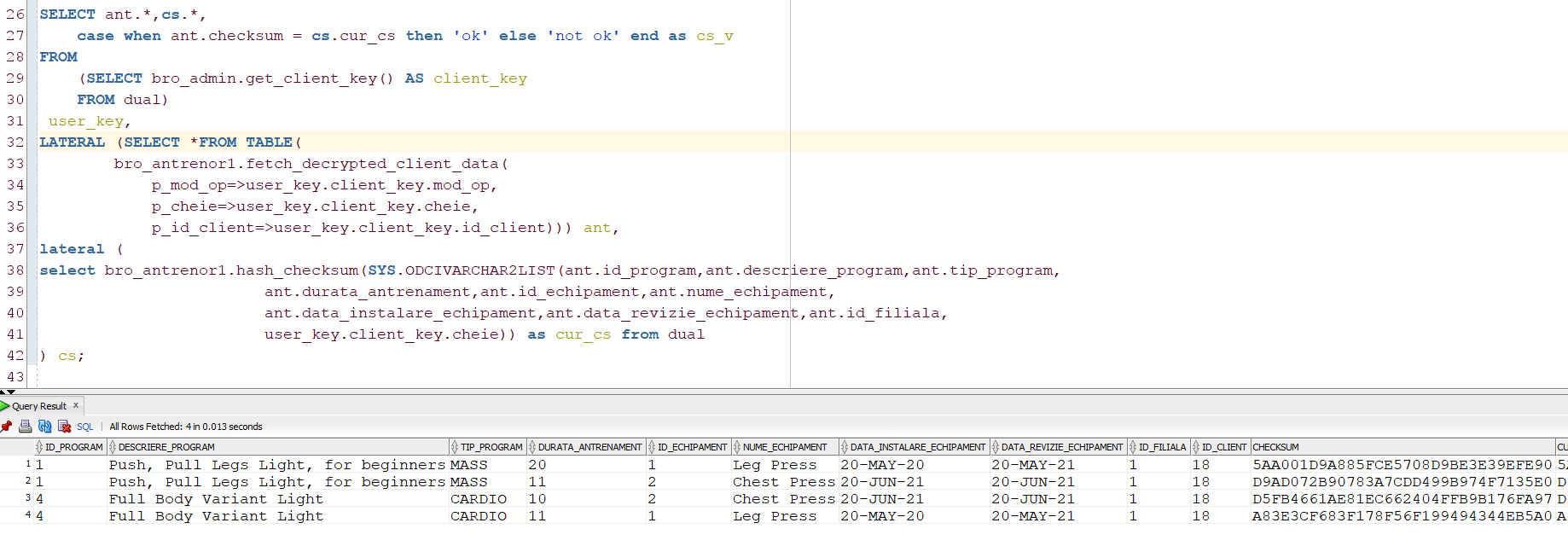
Not allowed este pus manual in caz de eroare de decriptare, insa daca oracle nu emite eroare, el decide ca a decriptat, insa dupa cum se poate observa, doar pentru clientul cu idul 18 este descifrabil outputul. De asemenea, in scriptul de seed este prezenta si o functie care returneaza automat dupa id numai liniile decriptate corect:



Functia intoarce si un checksum pentru a valida datele in sine ulterior, daca se doreste. Aceste doua selecturi sunt prezente in fisierul *bro\_antrenor1\_cript\_show.sql*.

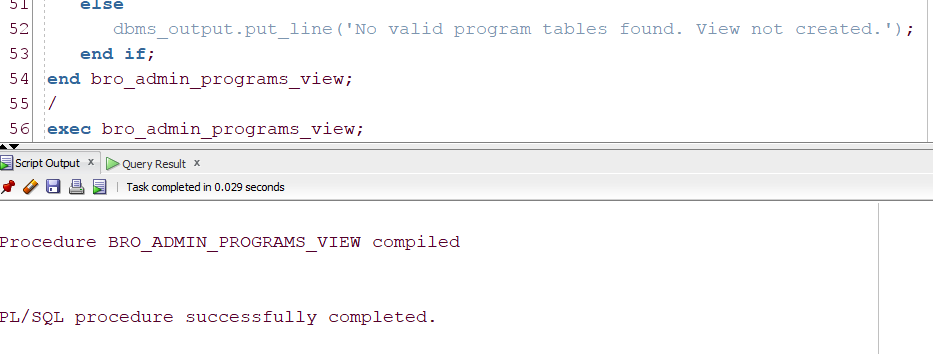
In acest moment, avem 1 antrenor cu o schema creata si cu date, si adminul cu schema creata si date in ea. In continuare vom da si restul permisiilor neceasare utilizatorilor conform matricii entitate-utlizator. Scriptul asociat este in fisierul *sys\_users\_2.sql.* Acest script contine un rol de baza, adica cel pentru publicul general si celelalte roluri care deriva din acesta, plus alte permisii individuale necesare.

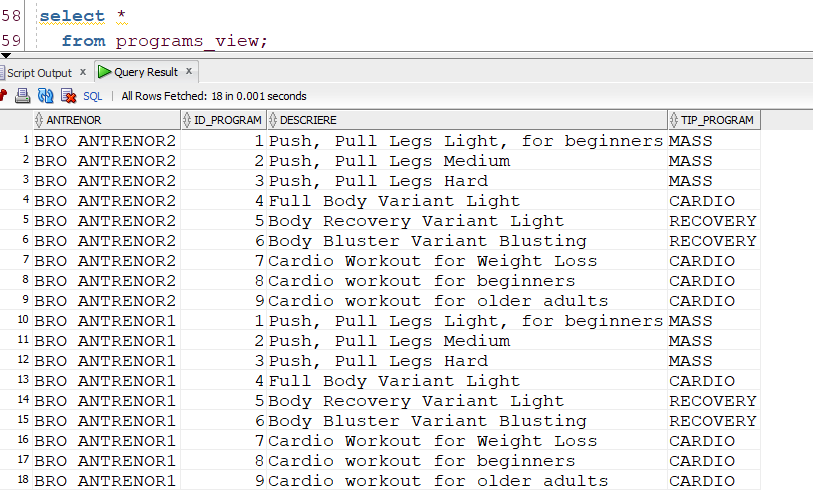
Pentru a ilustra criptarea in conexiunea lui client1 vom rula ultima comanda din fisierul *bro\_client1\_select\_cript.sql*



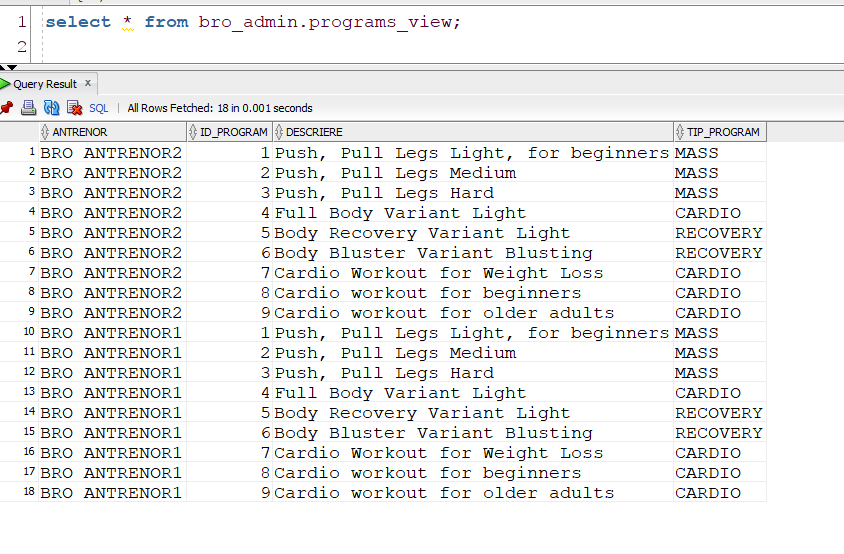
1. **Obiect dependent**

Dupa ce am rulat scriptul *bro\_antrenor\_insert.sql* in schema mai multor antrenori, in cadrul sys vom rula *sys\_admin\_antrenor\_privilege.sql*. Acest script contine o procedura care ofera adminului permisiunea de select with grant option pe tabela **program** doar pentru schemele antrenorilor care au aceasta tabela. Avem nevoie de acest grant option, intrucat pentru a face mai usoara selectarea tuturor antrenamenelor in admin vom crea un view care uneste toate aceste programe. Scriptul pentru acest view se gaseste in fisierul *bro\_admin\_programs\_view.sql* :





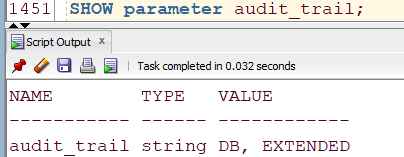
Deoarece, vom dori ca oricine sa poate accesa acest view, tot in cadrul acestui script vom oferi grant select din admin pentru rolul de baza. (*Obs:* desi adminul nu vede dba roles, el poate da grant pe un rol existent, deoarece are implicit with grant option obiectele din schema sa). Daca ne conectam in contul de public general vom vedea ca se poate selecta viewul:



**6. Audit**

**6.1. Audit Standard**

In cadrul proiectului vom avea auditul standard de forma **db,extended**:



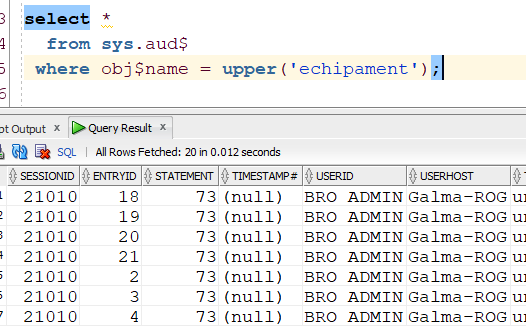
Vom audita astfel:

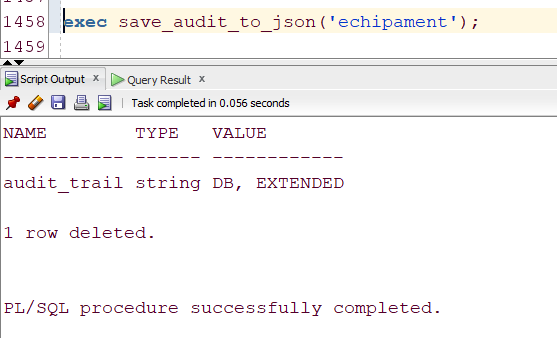
1. *bro\_admin.client\_extins*: inserarile si actualizarile nereusite
2. *bro\_admin.echipament*: inserarile, acutalizarile si stergerile
3. *bro\_admin.account\_mapping*: inserarile, acutalizarile si stergerile
4. *bro\_admin.supliment* : inserarile si actualizarile

Intrucat tinem auditarile in baza de date, tabelul poate creste foarte mult, iar pentru a nu pierde datele vom crea in sys o procedura care va salva datele in format json intr-un director, iar apoi pentru a nu fi nevoie sa se puna alarma cand se doreste salvarea, vom asocia unor joburi acesata procedura. Scriptul asociat se afla in fisierul *sys\_audit\_1.sql.*

Pentru a distinge fisierele usor, la salavare acestea vor avea numele: audit\_json\_bro\_<owner\_object>\_<object\_name>\_to\_char(sysdate,'YYYYMMDD\_HH24MISS').json

De exemplu, daca in admin updatam ‘fals’ echipamentele de mai multe ori (ie facem where data\_revizie=data\_revizie) si apoi rulam procedura de export, vom avea:

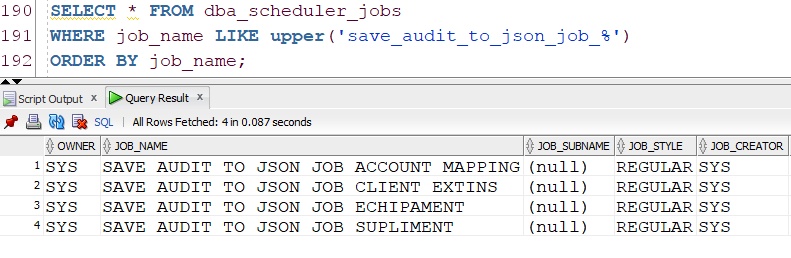




Iar in File Explorer avem fisierul:



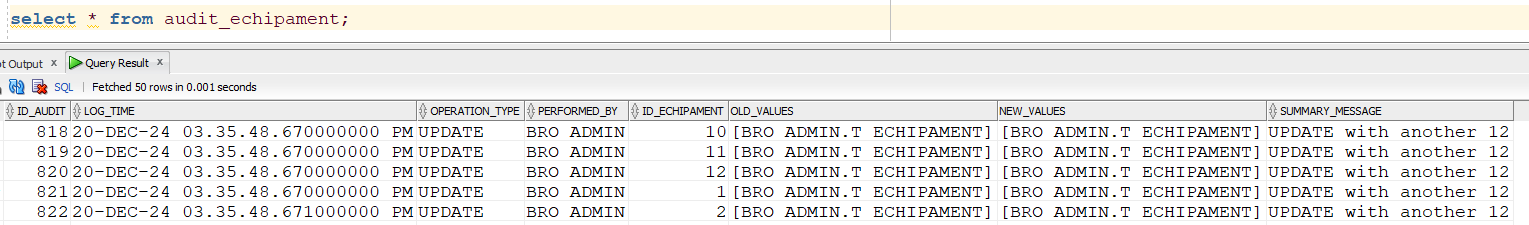
Pentru a vedea joburile se va rula:

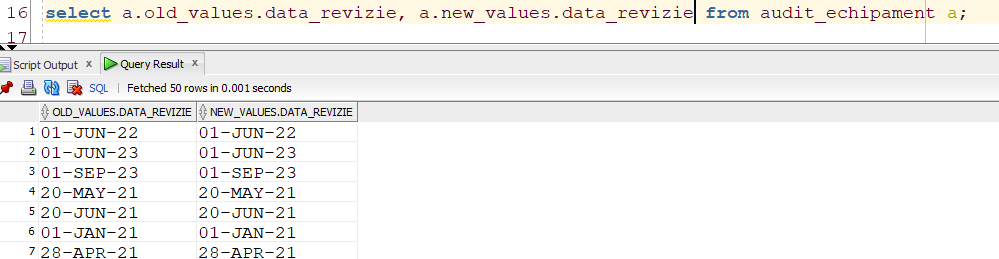


**6.2. Triggeri de Auditare**

Pentru triggerii de audit, in schema admin vom crea un tabel care va pastra opearatiile DML pentru tabelul **echipament**. In acest audit se va memora atat valoarea veche cat si cea noua pentru datele inserate/modificate/sterse in cadrul unei comenzi. Scriptul asociat poate fi gasit in fisierul *bro\_admin\_audit.sql*.

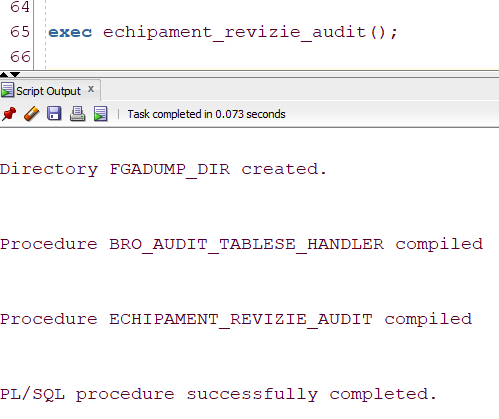
Pentru a ilustra triggerul de audit vom rula din admin scirptul de update fals, apoi vom selecta din tabelul de audit:



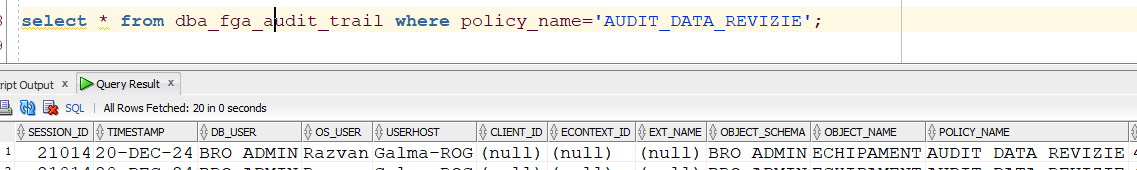


**6.3. FGA**

Pentru FGA vom auidta din nou tabela **echipament** pentru a vedea schimbarile din coloana **data\_revizie**. De asemena, vom salva pe disk logurile asocitate. Aceasta salvare va fi facuta in cadrul handlerului. Scriptul asociat poate fi gasit in *sys\_audit\_2.sql*.

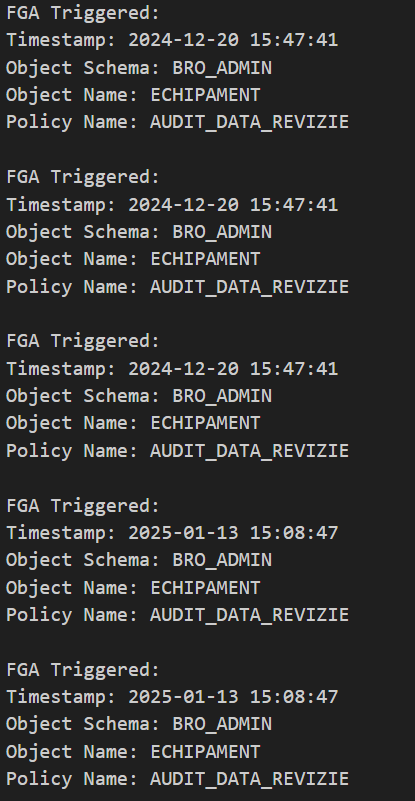


Pentru a ilustra FGA vom rula din nou updateul fals din admin:



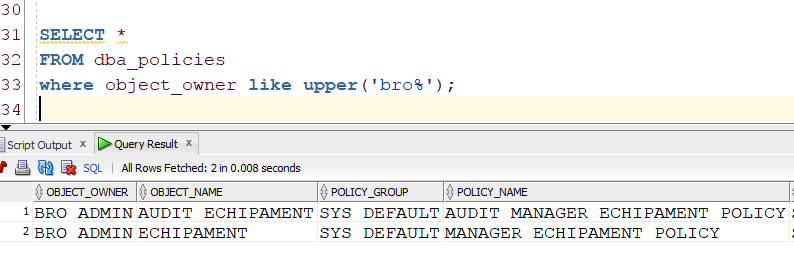
Fisierul txt de pe disk va salva intr-un mod append momentele cand auditul a fost activat:



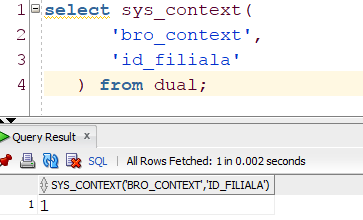


**7. Contextul aplicatiei**

Vom crea un context care pentru fiecare manager de filiala va extrage din username filiala asociata. Numele managerilor in baza de date este astfel: bro\_manger\_filiala<id\_filiala>, de exemplu pentru filaiala 1 avem userul bro\_manager\_filiala1. De asemena, contextul va fi folosit in doua VPD-uri pentru a asigura ca un manager de filala face opearii DML doar pe filiala sa si poate accesa din tabelul **audit\_echipament** doar campurile care au legatura cu filiala sa. Scriptul asociat este in fisierul *sys\_context.sql*:

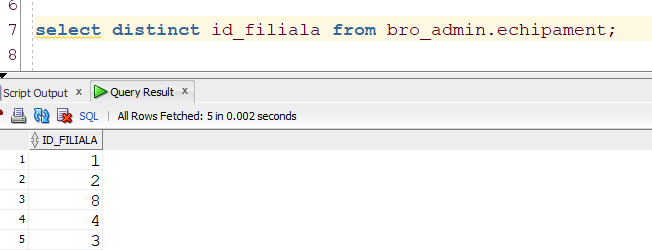


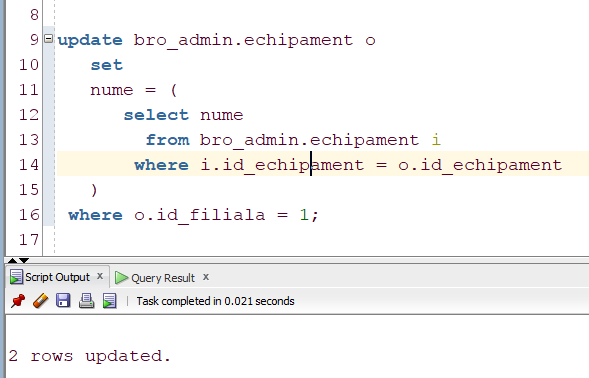
Conectandu-ne in manager1 putem vedea contextul:

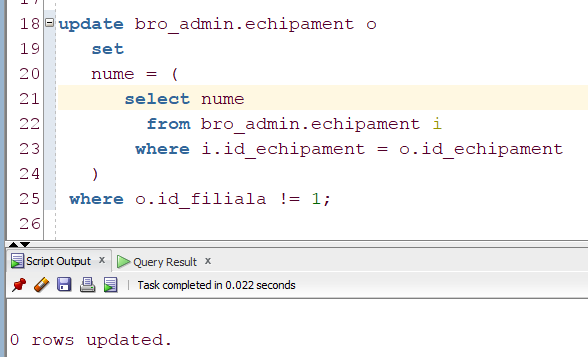


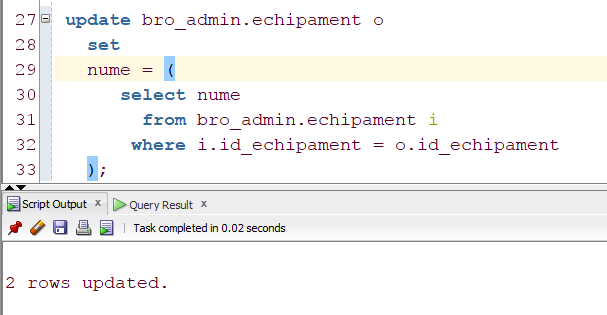
**7.1. VPD**

De exemplu, pentru VPD de update, daca se va incerca update pentru echipementele din filiala 1 se vor arata un numar de linii updatate, in schimb daca se incearca updateul pe echipamente din alte filiale se vor afisa 0 linii updatate fara nicio eroare, desi avem echipamente si in alte filiale. SQLul asociat managerului este in fisierul *bro\_manager\_filiala1\_context.sql .*

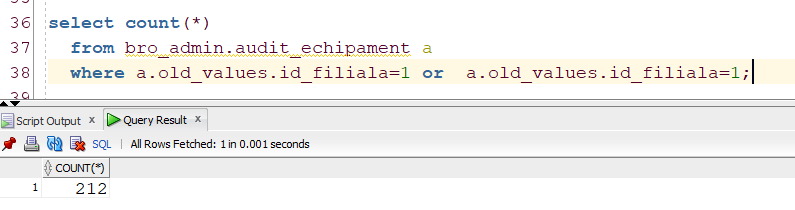


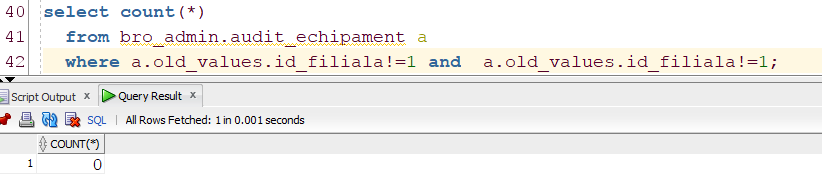


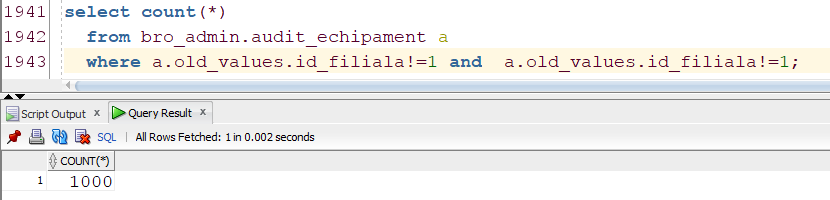




Politica de select pe tabelul de audit are rolul de lasa managerul unei filiale sa vada doar auditul pe echipamentele care fie au fost in filiala sa (ie old\_values.id\_filiala=1 aici) sau sunt (ie new\_values.id\_filial=1):





Daca rulam in admin ultimul select, vom vedea ca valoarea este diferita de 0: 

**8. SQL injection**

**8.1. Procedura Vulnerabila**

Pentru sql injection sa presupunem ca antrenor1 vrea sa creeze o procedura care permite utilizatorilor sa vada un program cu echipamentele care vor fi folosite in cadrul acestuia, filtrand echipamentele dupa data reviziei. Scriptul este in fisierul *bro\_antrenor1\_sql\_injection.sql*. Procedura va primi doi parametri: primul id-ul programului, iar ce-l de-l doilea data reviziei echipmentelor. Partea relevanta a procedurii este:

**'SELECT \* FROM bro\_admin.echipament e**

**NATURAL JOIN program p**

**WHERE p.id\_program = '**

**|| id\_prg ||**

**' AND upper(to\_char(data\_revizie, ''DD-MON-YY'')) LIKE ''%'**

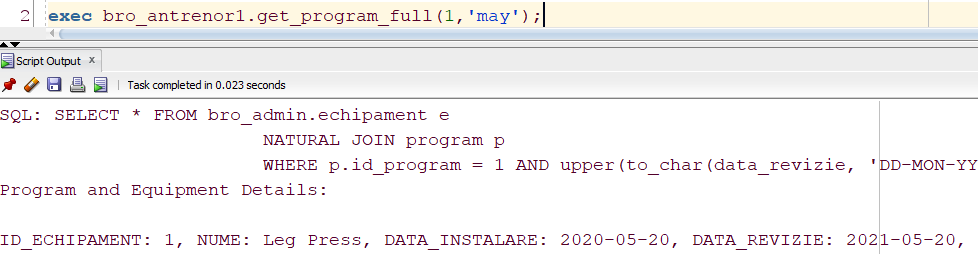
**|| upper(data\_inst)||**

**'%'''**

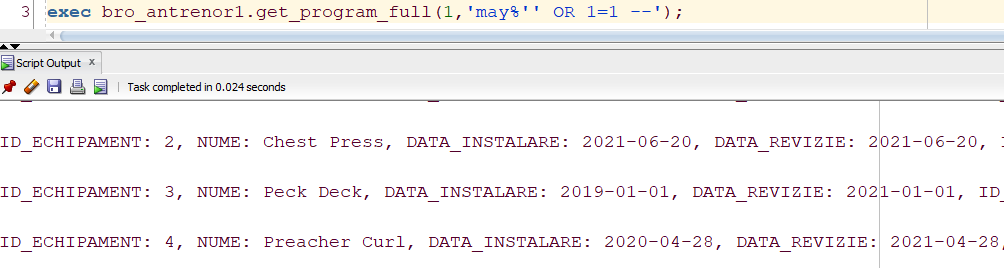
Dupa cum se poate observa, inputul este direct concatenat in selectul care va fi transmis motorului bazei de date, fara a fi sanitizat.

Pentru a putea rula procedura din antrenor, in cadrul fisierului mentionat vom da drepturi de execute lui client1. Vom rula scriptul *bro\_client1\_sql\_injection.sql* pentru a demonstra un apel onest si doua apeluri menite sa arate vulnerabiliatile procedurii:

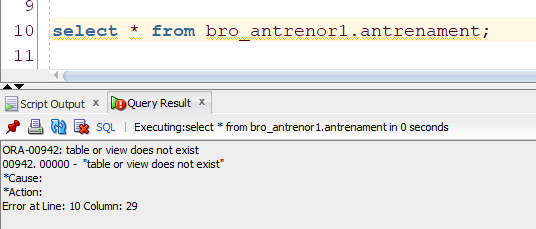
Apel onest:

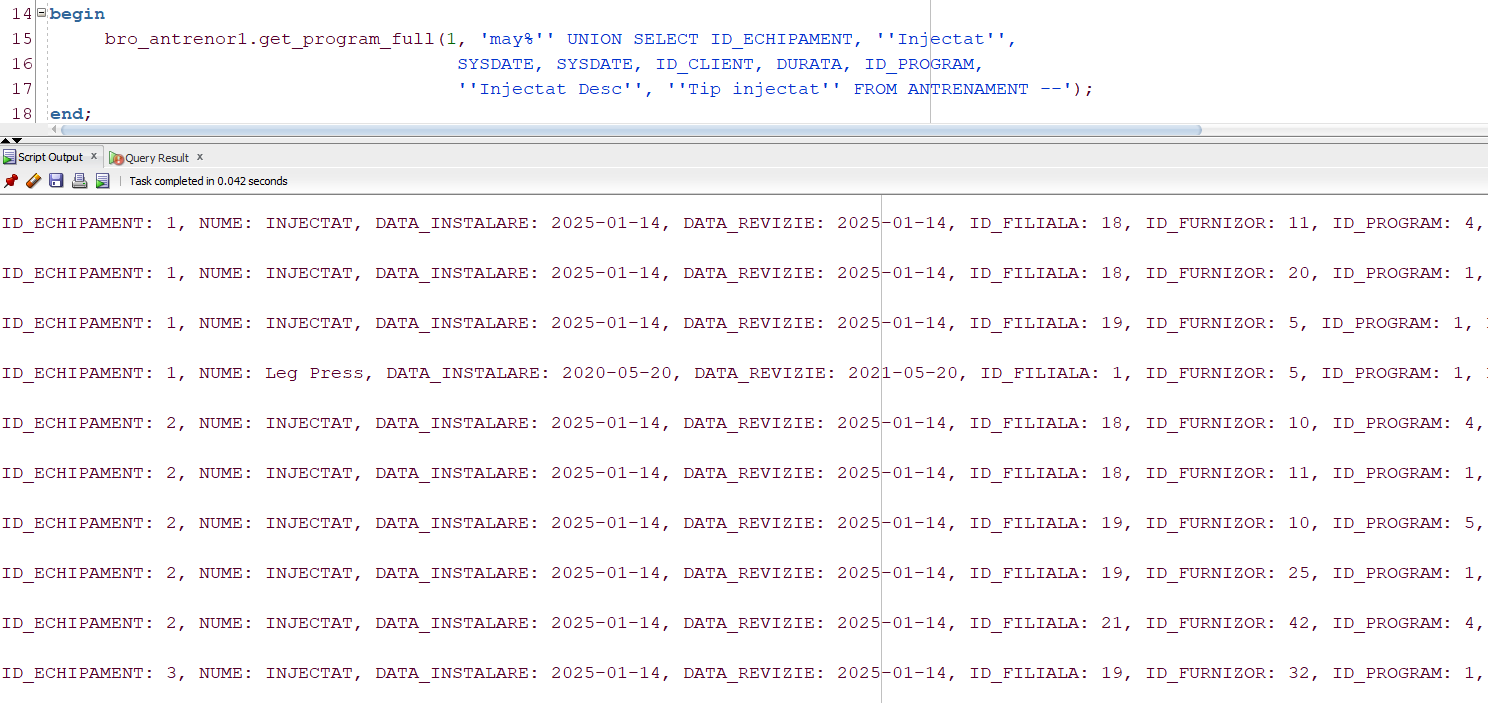


Apel care intoarce toate programele cu echipamentele asociate, subminand filtrarea:



Apel care intorace toate antrenamentele, desi clientul nu are drept de select pe tabela antrenament:





**8.2. Procedura reparta**

Pentru a face procedura mai sigura la atacuri de tip injection se va schimba crearea queryului care folosete parametrii de intrare: se va inlocui simpla concatenare cu binding astfel:

**'SELECT**

**e.id\_echipament,**

**e.nume,**

**e.data\_instalare,**

**e.data\_revizie,**

**e.id\_filiala,**

**e.id\_furnizor,**

**p.id\_program,**

**p.descriere,**

**p.tip\_program**

**FROM bro\_admin.echipament e**

**NATURAL JOIN program p**

**WHERE p.id\_program = :id\_prg**

**AND**

**UPPER(TO\_CHAR(e.data\_revizie, ''DD-MON-YY'')) LIKE :data\_inst'**

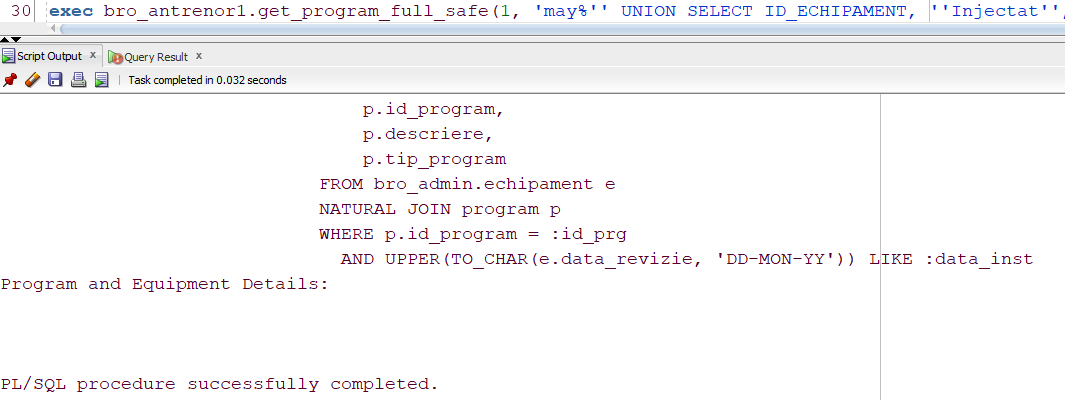
Iar, apelarea sa va fi urmatoarea:

**EXECUTE IMMEDIATE v\_sql BULK COLLECT**

**INTO v\_program\_echipament USING id\_prg**

**,'%' || upper(data\_inst) || '%';**

Daca se va incerca oricare dintre apelurile rau intentionate procedura nu va intoarce nicio linie, intrucat in acest moment apelantul nu mai are posibiliatatea de a altera strucutra efectiva a stringului de interogare:



1. **Mascarea datelor**

Pentru mascarea datelor se vor exporta persoanele din baza de date modifcand astfel coloanele:

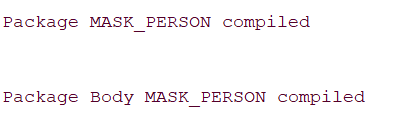
1. Valorile coloanelor numerice care nu sunt chei sa vor schimba in valori care incep cu aceeasi cifra si au aceeasi lungime, restul de cifre vor fi random.
2. Valorile coloanelor de tip string se vor schimba astfel: prima data se alege random daca se va dubla lungimea stringului, dupa care se pastreaza primul caracter, apoi se adauga random pana la noua lungime cate un caracater ‘\*’ sau ‘#’.
3. Cheila isi vor pastra unicitatea dar vor putea avea dimensiunea pana de 5 ori mai mare.

**9.1. Export**

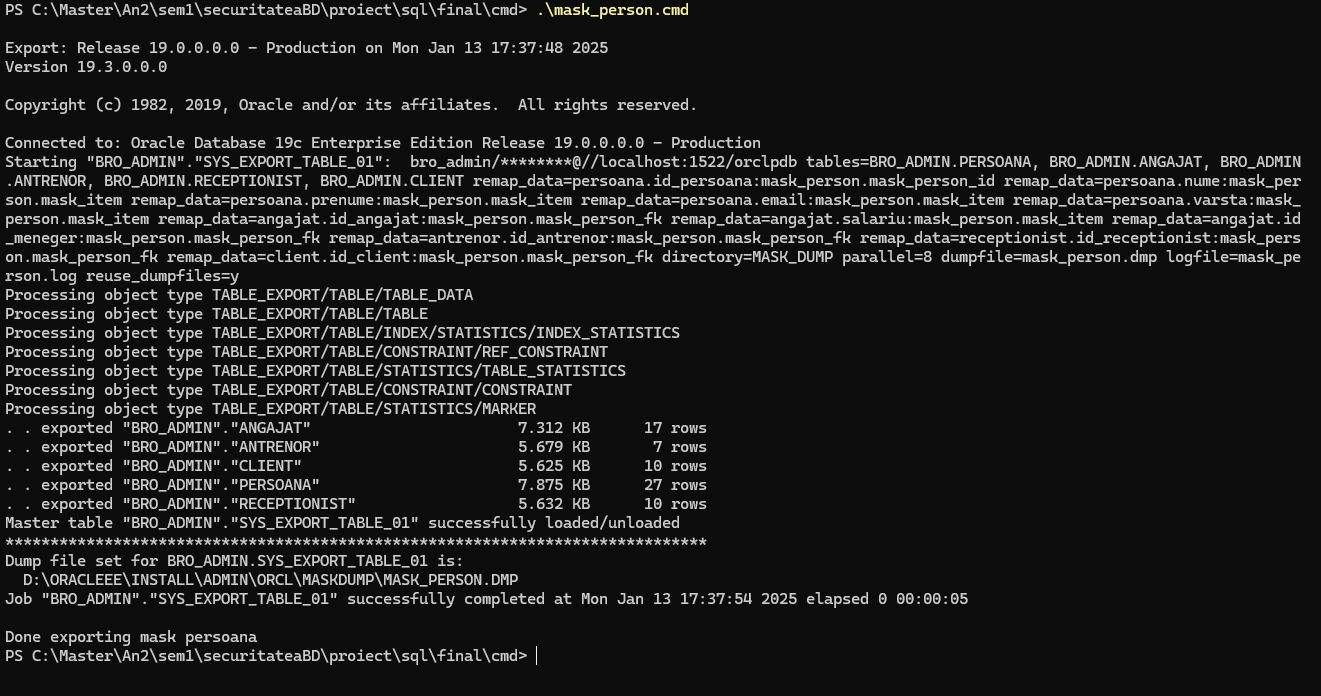
Pentru export se va crea in sys un nou director, iar adminul va primi permisuni pe acesta. Pentru import sa va crea un nou utilizator cu drepturi de import, ie **datapump\_imp\_full\_database**, pentru a putea remapa schema lui bro\_admin la schema noului utilzator. S-a ales acest model pentru a demonstra exportul si importul pastrand constrainturile initiale ale tabelelor.

Pentru sys, fisierul asociat este *sys\_mask.sql,* in care se creeaza noul user, directorul si se dau drepturile asociate.

In fisierul *bro\_admin\_mask.sql* se gaseste definirea pachetului care realizeaza maparea.



Comnada de realizare a maparii se afla in *mask\_person.cmd*:

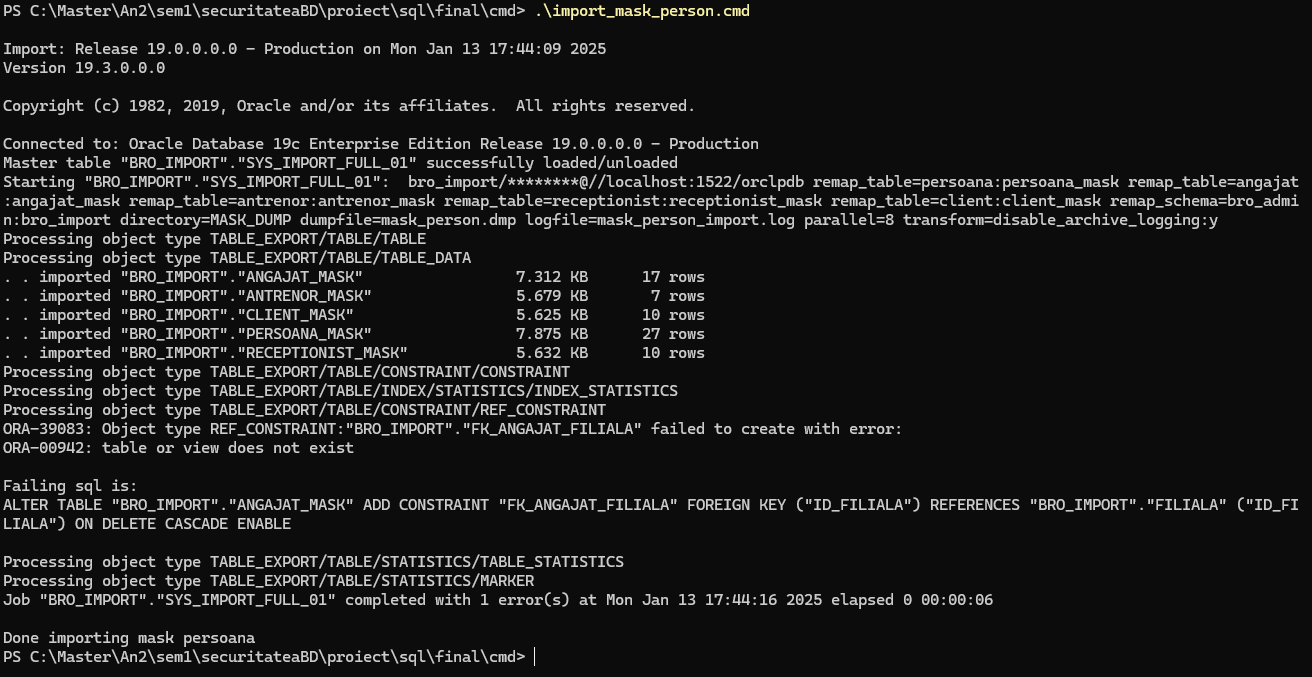


*Obs:* Desi am pus tables intr-o anumita ordine, Oracle ia tables alfabetic, in minunata lor documentatie nu am gasit nimic. Asa ca a trebuit sa preinitializez la mask\_person\_id si mask\_person\_fk cheile din persoana, daca stateul de chei este gol. Nu cred ca este un comportament normal, pe internet nu am gasit pe cineva sa se planga de acest aspect.

(Nu am folosit package body init ie un begin, pentru a asiugra ca datele din perosana sunt cele din momentul exportului)

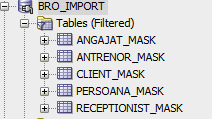
**9.2. Import**

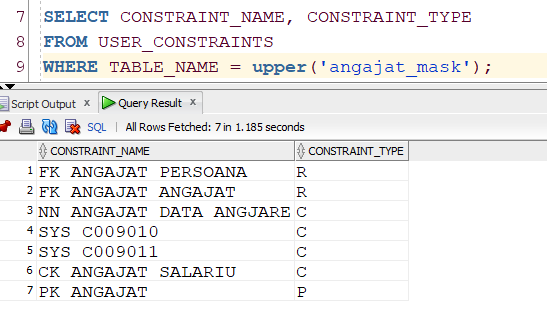
Pentru import in schema bro\_import, sa va rula scriptul *import\_mask\_person.cmd*:

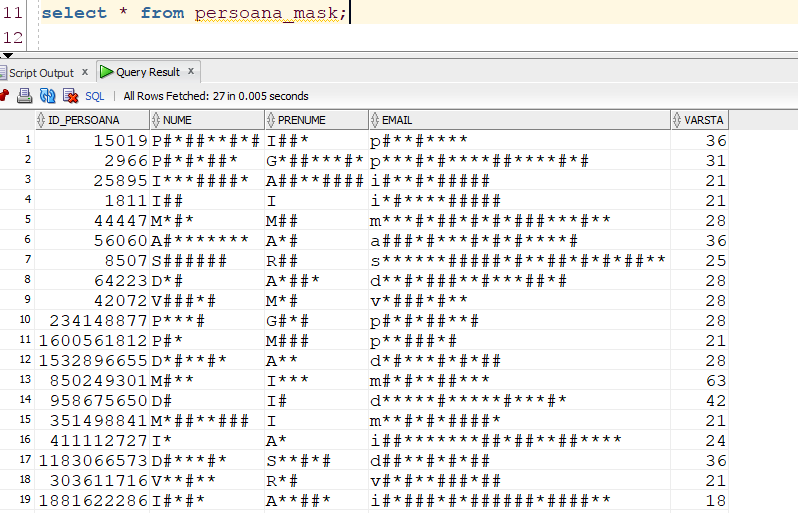


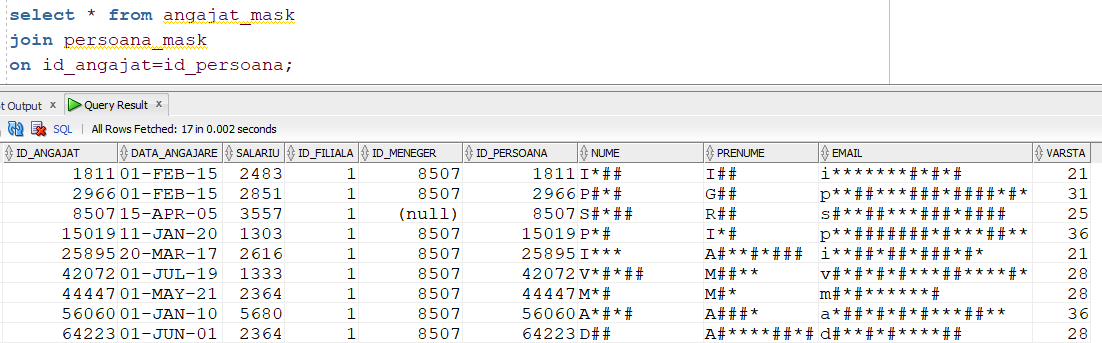
Intrucat am exportat doar tabelele **persoana**, **angajat**, **antrenor**, **receptionist** si **client**, nu si tabela **filiala** si am pastrat la export constrainturile, la import vom avea o eroare care spune ca nu se poate rezolva constraintul de FK pentru tabela **filiala**. Pentru ca suntem constienti ca nu am exportat acea tabela, putem ignora aceasta eroare intrucat singurul lucru care se va intampla este ca in tabela **angajat\_mask**, cea imporatata, nu vom mai avea acel constraint de FK.

Daca deschidem o conexiune cu userul bro\_import, vom putea constata crearea tabelelor, iar pentru angajat mapat nu este prezent constraintul de FK pe **filiala**. De asemenea, in tabele vor fi prezente datele mapate:









SQLul asociat userului de import sa gaseste in fisierul *bro\_import.sql*.