**Cuprins**

[1. Prezentarea modelului si a regulilor sale 1](#_Toc23495)

[1.1. Constrângeri pentru implementare 2](#_Toc10568)

[1.2. Diagrama Entitate Relație 3](#_Toc436)

[1.3. Diagrama Conceptuală 4](#_Toc20052)

[1.4. Regulile de securitate 5](#_Toc23735)

[2. Procesele aplicației 6](#_Toc13049)

[2.1. Matricea Proces-Utilizator 7](#_Toc20834)

[2.2. Matricea Entitate-Proces 8](#_Toc25158)

[2.3. Matricea Entitate-Utilizator 9](#_Toc15404)

[3. Gestiunea Utilizatorilor și a Resurselor Computaționale 10](#_Toc22007)

[3.1. Configurarea Utilizatorilor și a Schemei 10](#_Toc27284)

[3.2. Memorie alocată pentru cateogriile de utilizatori 11](#_Toc15729)

[3.3. Profile 11](#_Toc11945)

[3.3.1. Plan de consum 11](#_Toc2506)

[3.3.2. Crearea Efectivă 12](#_Toc26493)

[3.4. Permisii Admin 13](#_Toc20174)

[4. Creare Bazei 14](#_Toc28304)

[4.1. Crearea Schemei Admin 14](#_Toc18778)

[4.1.1. Criptare în Schema Admin 15](#_Toc6457)

[4.2. Crearea Schemei Antrenor 16](#_Toc7572)

[4.2.1. Criptare în Schema Antrenor 17](#_Toc13963)

[5. Obiect dependent 19](#_Toc13245)

[6. Audit 21](#_Toc30038)

[6.1. Audit Standard 21](#_Toc4271)

[6.2. Triggeri de Auditare 23](#_Toc15147)

[6.3. FGA 24](#_Toc12110)

[7. Contextul aplicației 26](#_Toc31890)

[7.1. VPD 27](#_Toc19054)

[8. SQL injection 29](#_Toc26575)

[8.1. Procedura Vulnerabilă 29](#_Toc17151)

[8.2. Procedura repartă 31](#_Toc32486)

[9. Mascarea datelor 32](#_Toc27772)

[9.1. Export 33](#_Toc25288)

[9.2. Import 34](#_Toc6640)

**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 ).

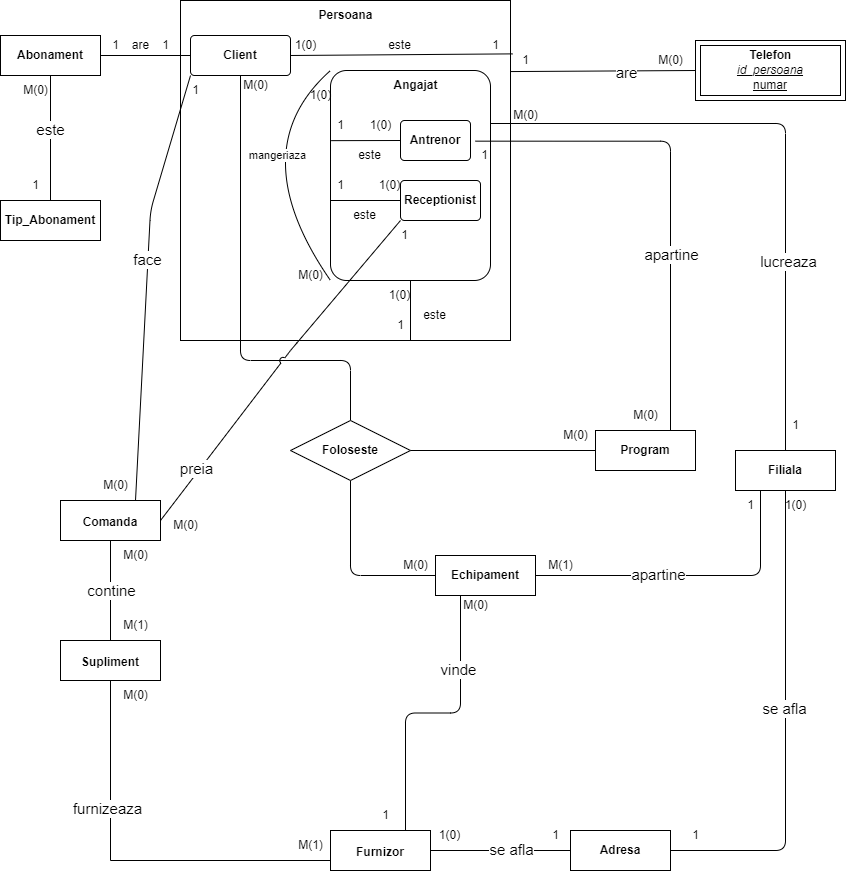
O aprovizionare a firmei se va face cu un singur supliment o dată.

Nu se înregistrează stocuri de suplimente.

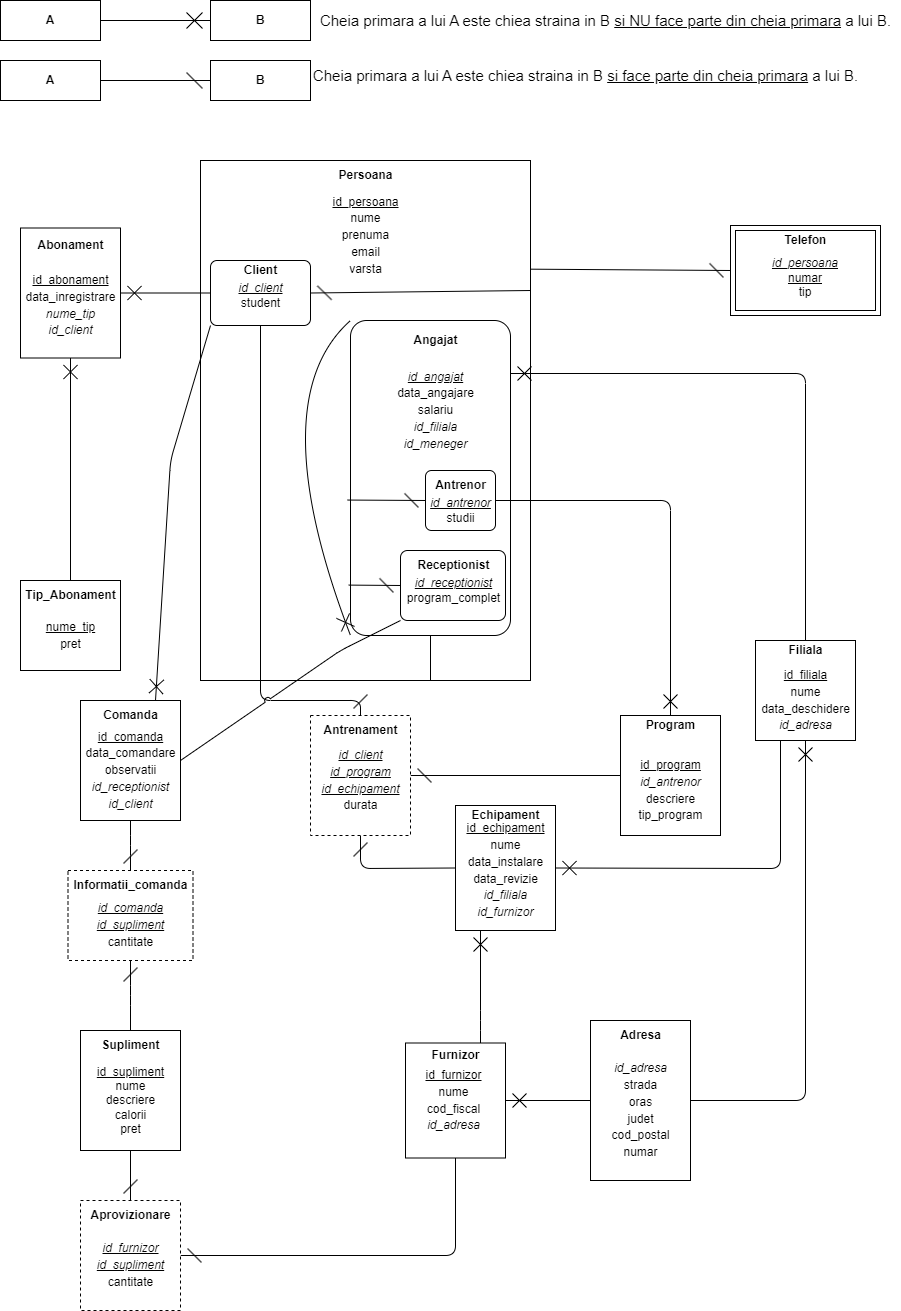
Filialele trebuie să aibă adresele diferite între ele.

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**

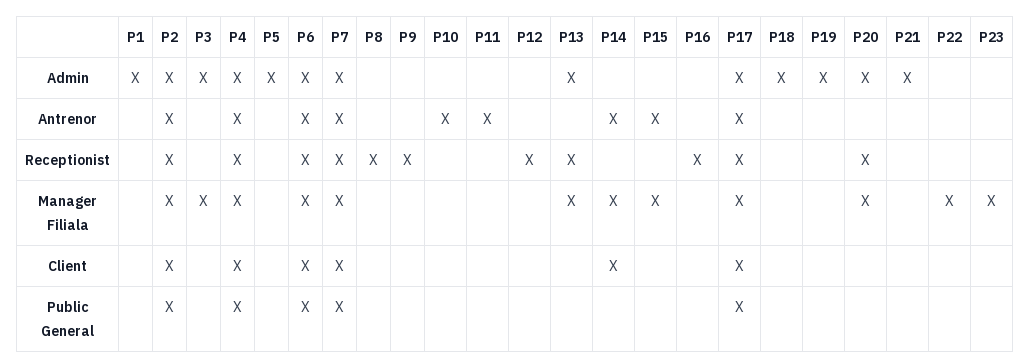
În cadrul proiectului se vor cripta datele antrenamentelor fiecărui client, astfel încât doar el și antrenorul său să poată vedea informațiile. De asemenea, se vor realiza două VPD-uri (Virtual Private Database) pentru a asigura că managerii de filială pot face operații DML doar pe echipamentele din filiala lor și că pot vedea doar istoricul echipamentelor care sunt sau au fost în acea filială. De asemenea, se vor respecta permisiunile din matricea entitate-utilizator și se vor da cote de memorie conform necesităților de stocare.

**2. Procesele aplicației**

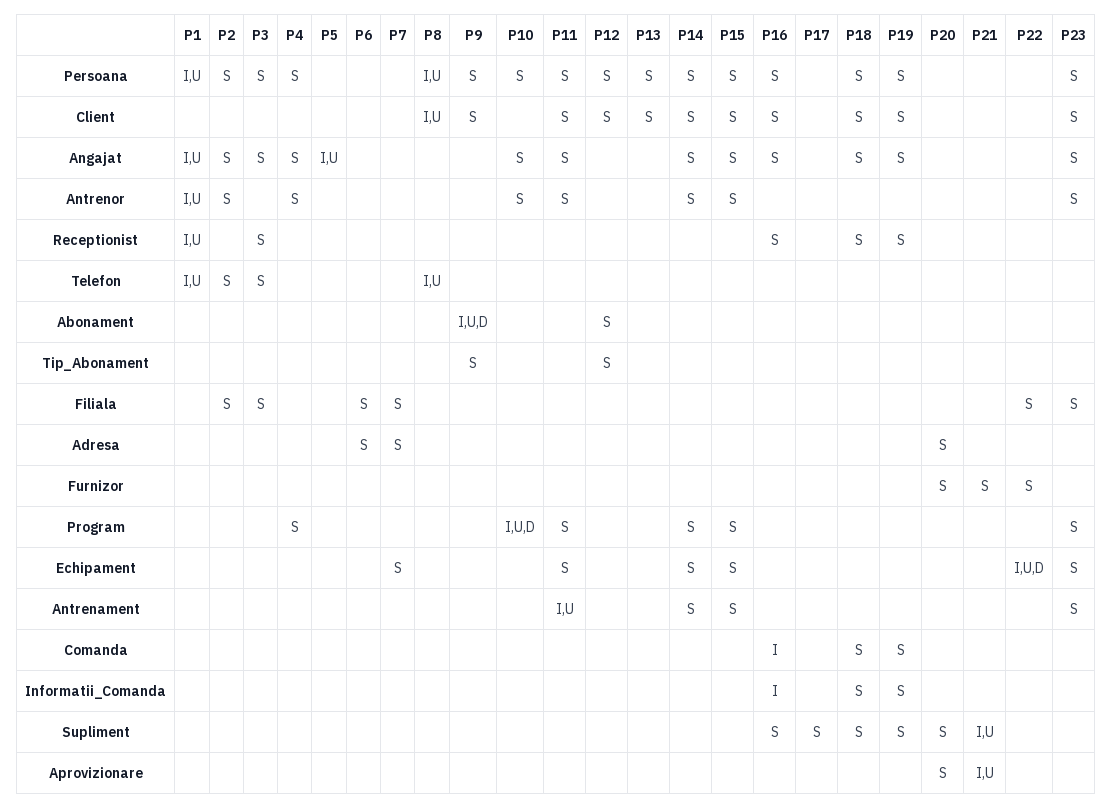
Procesele care pot fi inițiate în cadrul bazei sunt:

1. Configurarea angajați
2. Vizualizarea antrenorilor dintr-o filială
3. Vizualizarea recepționiștilor dintr-o filială
4. Vizualizarea programelor de antrenament pentru un anumit antrenor
5. Delegarea/Revocarea de manager pentru o filială
6. Vizualizarea caracteristicilor unei filiale
7. Vizualizarea echipamentelor dintr-o filială
8. Managementul clienților
9. Configurarea abonamentului pentru un client
10. Configurarea programelor pentru un antrenor
11. Configurarea antrenamentelor pentru un antrenor
12. Verificarea validității abonamentului pentru un client
13. Vizualizarea clienților
14. Vizualizarea antrenamentelor unui client
15. Vizualizarea antrenamentelor unui antrenor
16. Crearea unei comenzi
17. Vizualizarea suplimentelor puse la vânzare
18. Vizualizarea tuturor comenzilor
19. Vizualizarea comenzilor pentru un client
20. Vizualizarea aprovizionărilor suplimente
21. Gestionarea aprovizionărilor suplimente
22. Gestionarea echipamentelor dintr-o filială
23. Vizualizarea tuturor antrenamentelor dintr-o filială
24. Matricea Proces-Utilizator
25. În cadrul proiectului se disting șase categorii distincte de utilizatori:
26. Admin – cel care gestionează aplicația
27. Antrenor – angajatul de tip antrenor din cadrul unei săli
28. Recepționist – angajatul de tip recepționist din cadrul unei săli
29. Manager Filială – managerul unei singure filiale
30. Client – client pe întregul lanț de săli
31. Public General

**2.1. Matricea Proces-Utilizator**

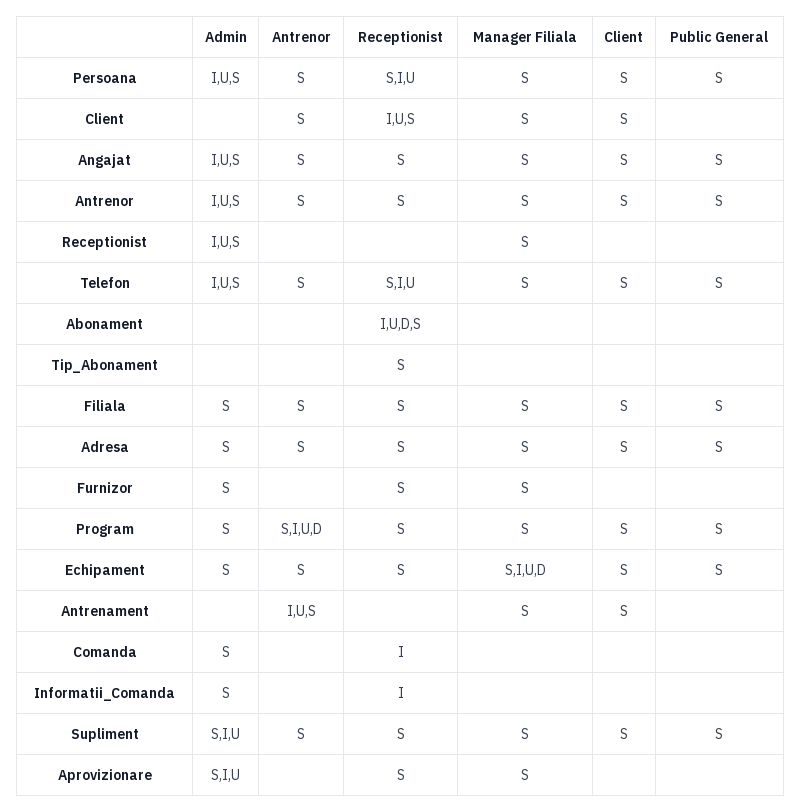


**2.2. Matricea Entitate-Proces**



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

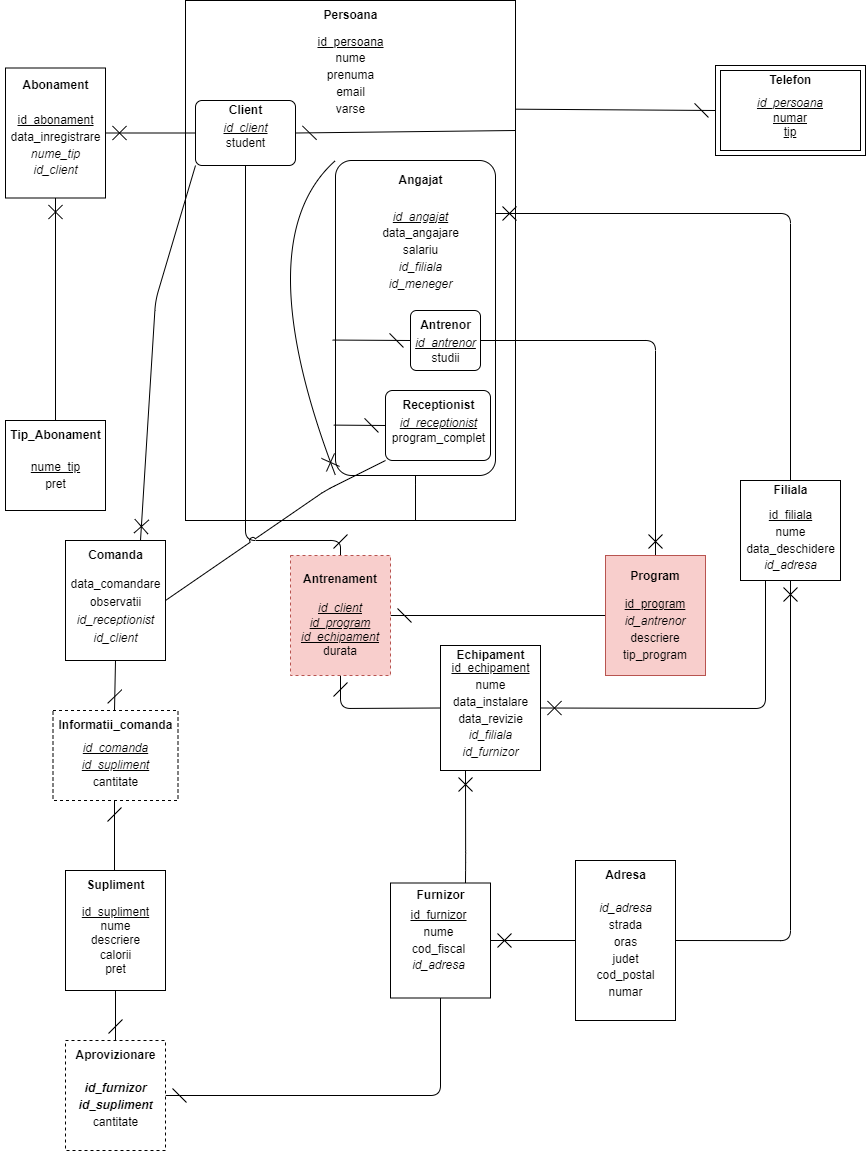
**2.3. Matricea Entitate-Utilizator**



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

1. **Gestiunea Utilizatorilor și a Resurselor Computaționale**

**3.1. Configurarea Utilizatorilor și a Schemei**



Din cadrul diagramei conceptuale, toate tabelele vor fi create în schema adminului, mai puțin Program și Antrenament care vor fi create separat de fiecare antrenor în schema lui proprie.

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

· Deoarece majoritatea bazei de date va fi creată în schema adminului, acesta va avea la dispoziție 500MB.

· Fiecare antrenor va avea la dispoziție 10MB pentru a crea obiecte.

· Întrucât restul utilizatorilor nu vor crea obiecte, aceștia nu vor avea memorie alocată.

**3.3. Profile**

În cadrul aplicației, parolele trebuie să conțină minim un „\_”. Această condiție va fi verificată în cadrul fiecărui profil utilizând opțiunea *password\_verify\_function*.  
Pentru admin se permit mai multe sesiuni, sunt permise 15 minute de idle per sesiune, parola trebuie schimbată o dată la 90 de zile, sunt permise 5 greșeli consecutive ale credentialelor și are dreptul la 6 minute maxime de CPU time per comandă.  
Pentru publicul general se permit 6 sesiuni, sunt permise 5 minute de idle per sesiune, un maxim de 20 de minute per sesiune și dreptul la 1 minut maxim de CPU time per comandă.  
Pentru restul tipurilor de utilizatori se permite doar o sesiune, sunt permise 15 minute de idle, parola trebuie schimbată o dată la 90 de zile, sunt permise 5 greșeli consecutive ale credentialelor și are dreptul la 2 minute maxime de CPU time per comandă.

**3.3.1. Plan de consum**

Pentru planul de consum, când CPU-ul ajunge la 100%, în cadrul aplicației există următoarea regulă:

Admin: 30%

Public General: 5%

Clienți: 10%

Manageri: 15%

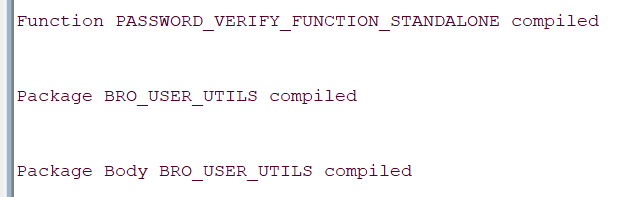
Recepționiști: 15%

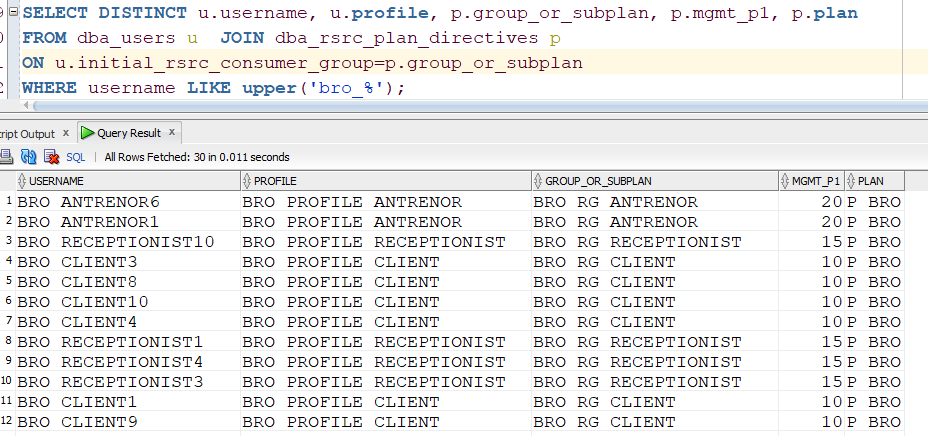
Antrenori: 20%

Other Groups: 5%

**3.3.2. Crearea Efectivă**

Crearea utilizatorilor, mai puțin a adminului, a fost realizată utilizând un pachet custom utilitar. Codul pentru acest pachet și configurarea inițială a utilizatorilor cu profile și resource group este prezentă în fișierul *sys\_users\_1.sql*. Mai jos sunt câteva outputuri din rularea acestui cod:





**3.4. Permisii Admin**

Tot în acest fișier inițial de configurare sunt prezente și 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 utilizând scriptul din fișierul *bro\_admin\_antrenor\_seed.sql*, adminul are nevoie de permisiuni speciale pentru a putea rula scriptul de creare cu succes (create/select any).

Permisia Select Any Sequence este necesară deoarece tabelele din schema antrenorilor vor avea cheile primare generate folosind autoincrementul din Oracle. Astfel, adminul are nevoie de permisia de selectare a acestei secvențe generate la crearea 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**

În cadrul adminului, adiacent tabelelor din schema conceptuală, se vor implementa la început încă două tabele, anume unul este de logging al erorilor care apar în operațiile DML pe tabelele sau view-urile de persoane ale schemei sale, iar celălalt este unul de mapare a unui utilizator creat cu un cont al bazei de date, astfel asigurând că există un cont înainte de a insera într-un tabel al bazei. Pentru popularea celui de-al doilea tabel este nevoie de legătura dintre admin și useri și, de aceea, există permisia de a executa funcția *get\_users\_by\_suffix* din sys către admin. Scriptul inițial al bazei admin, alături de inserarea unor date, este prezent în fișierul *bro\_admin\_create\_tables.sql*.

Structura tableului de mapări de conturi este urmatoarea:

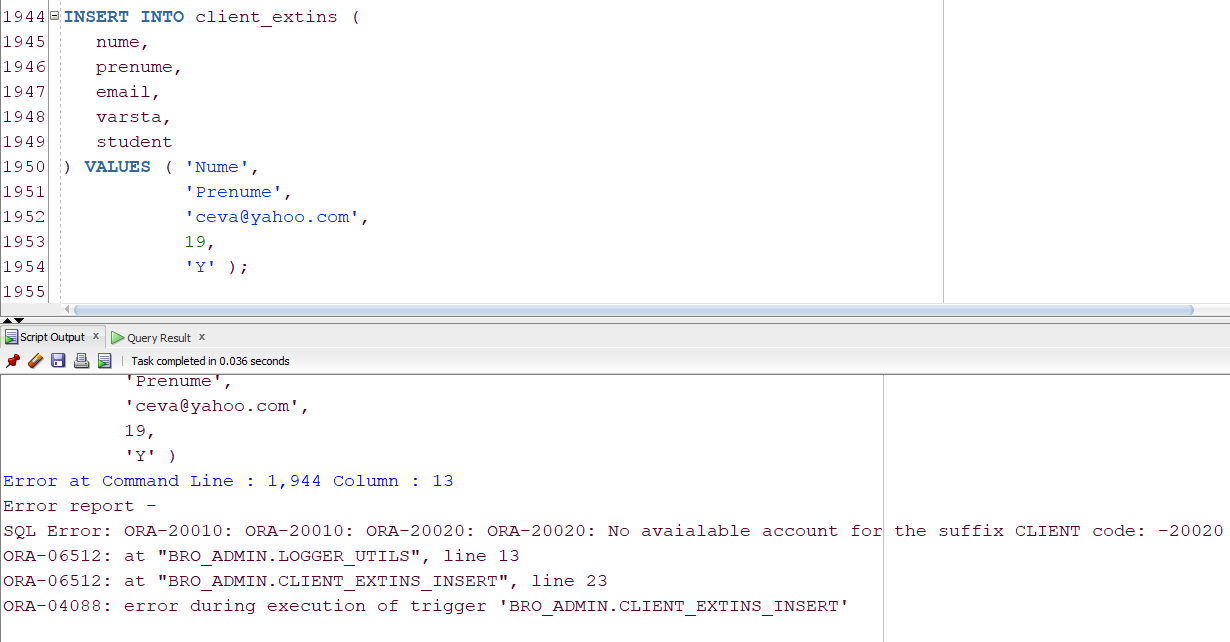
**CREATE TABLE account\_mapping(**

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

**username VARCHAR2(128) UNIQUE**

**);**

Acesta are cheia primară cea din tabela persoană, la momentul respectiv, care are contul și va ține minte și username-ul asociat acesteia. Inserările persoanelor în baza de date se vor face folosind view-urile specifice fiecărui tip, iar în trigger-ii “instead of” se va apela funcția *insert\_into\_account\_mapping*, care asigură că există conturi ale bazei de date pentru tipul de persoană dorit a fi inserat. De exemplu, pentru clienți, dacă am utilizat toate conturile și dorim să mai adăugăm un client, vom primi eroarea:

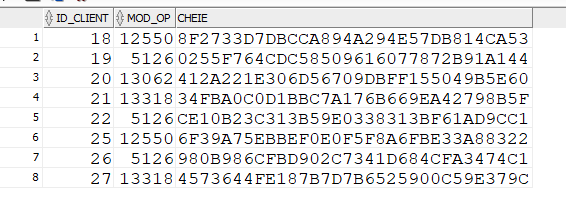


**4.1.1. Criptare în Schema Admin**

Întrucât scriptul de seed pentru antrenori include și partea de criptare corespunzătoare lor, înainte de a rula acel script voi prezenta criptarea.

Așadar, în cadrul aplicației, criptarea se va face pentru clienți la nivelul antrenamentelor unui antrenor, astfel încât un client să-și poată vedea doar propriile antrenamente. Pentru aceasta, în schema admin vom crea mai multe obiecte, scriptul asociat acestora se găsește în fișierul *bro\_admin\_criptare.sql*.

În acest script este prezent un tabel numit *chei\_client* care, pentru fiecare client, păstrează modul de criptare și cheia asociată. La inserarea în tabelul de chei se va folosi algoritmul AES128, iar paddingul va fi ales random pentru fiecare client, dintre PKCS5 și PADZERO, iar, tot random pentru fiecare client, se va alege și chainingul. După ce adminul a inserat pentru fiecare *id\_client* în *chei\_client*, vom avea:

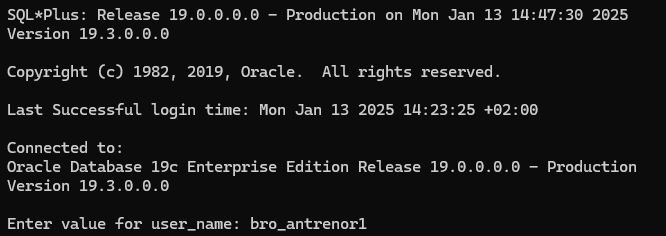


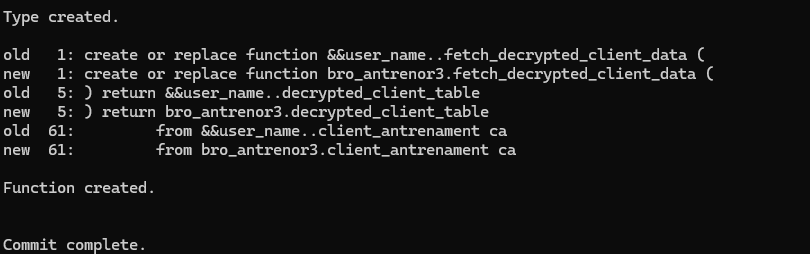
Pentru ca un client să-și ia cheia unică, tot în cadrul scriptului este creată o funcție *get\_client\_key*, care, pentru username-ul luat din contextul *userenv*, va întoarce cheia de criptare asociată.

**4.2. Crearea Schemei Antrenor**

Acum, pentru a ilustra criptarea, vom rula scriptul din fișierul *bro\_antrenor\_seed.sql*. Pentru a ușura inserarea, în cadrul acestui fișier schema este dată ca un argument, de exemplu: **create table &&user\_name..program**

Tot în cadrul seed-ului se vor da antrenorilor permisiunile necesare creării obiectelor. De asemenea, pentru a rula scriptul se va folosi fișierul *seed\_antrenor.cmd*. La rularea acestui fișier se va introduce numele antrenorului:



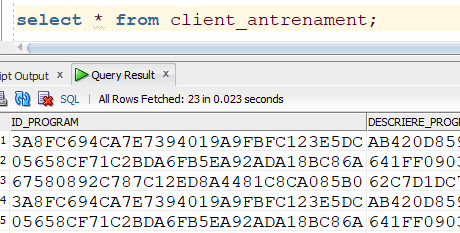


**4.2.1. Criptare în Schema Antrenor**

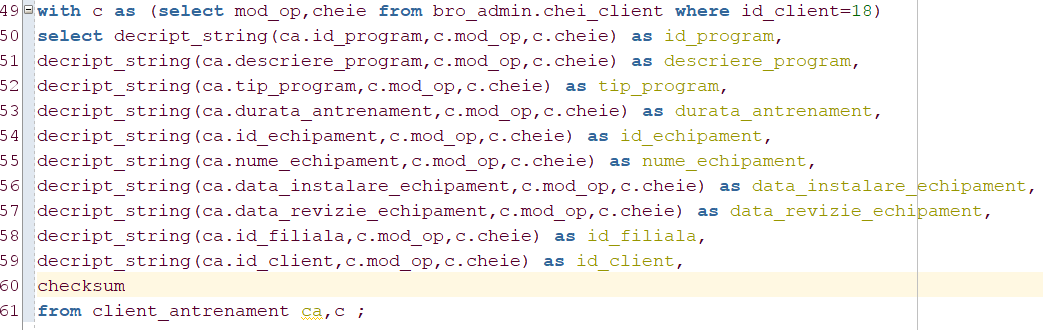
Întrucât antrenorii trebuie să aibă acces la cheile de criptare ale clienților, li se va da acces la tabelul de chei, iar, totodată, scriptul creează și un view care va arăta datele antrenamentelor pentru clienți într-un mod criptat cu cheile individuale. Pentru a regăsi datele și a le valida, sunt create mai multe funcții și obiecte utilitare pe care un client le poate folosi.

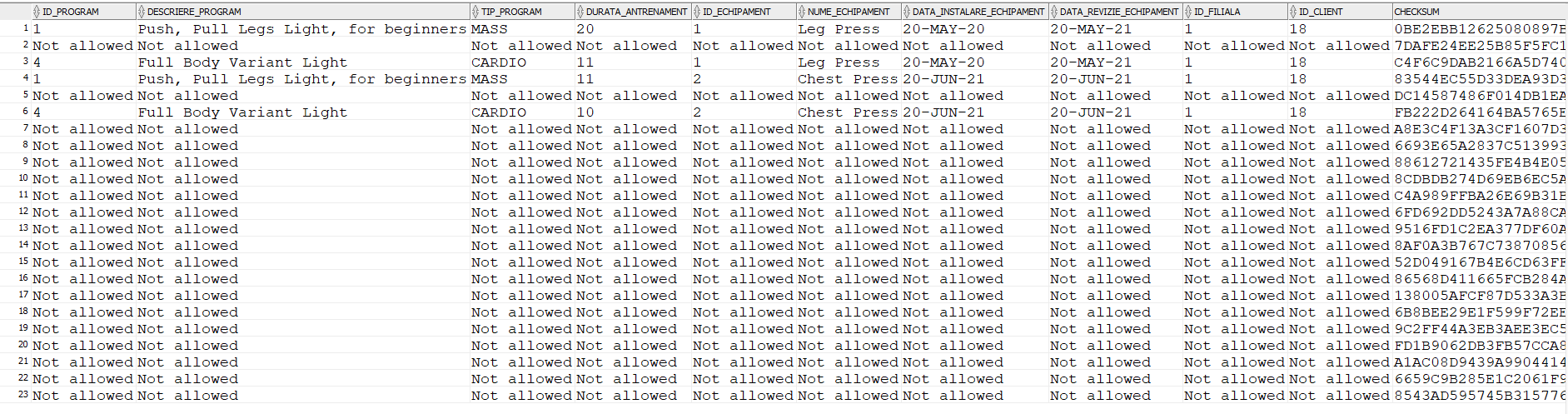
Pentru a putea insera date vom rula din antrenor1 scriptul *bro\_antrenor\_insert.sql*.

Pentru a vedea datele criptate vom apela:

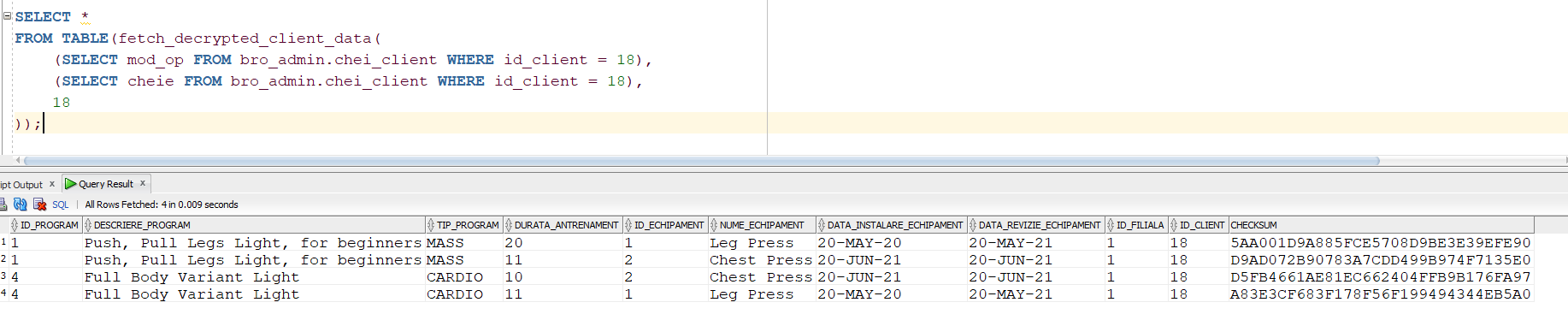


De exemplu, dacă dorim să decriptăm din tabel totul, dar doar pentru clientul cu id-ul 18, outputul va arăta:





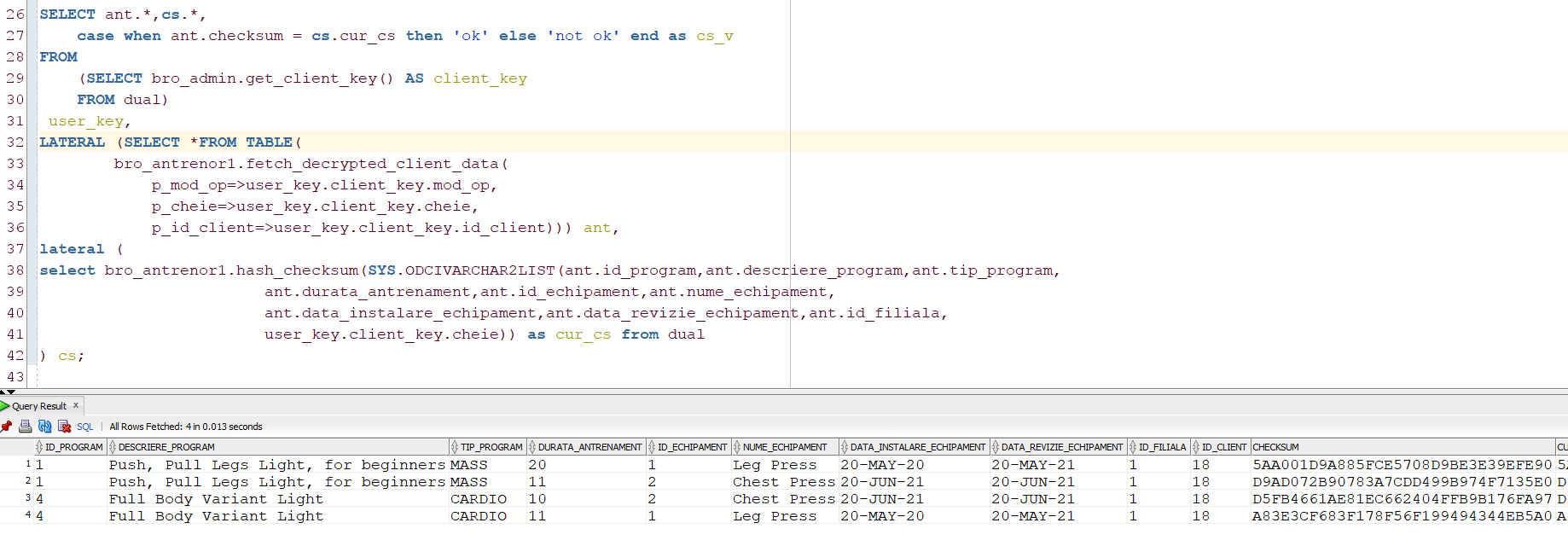
“Not allowed” este pus manual în caz de eroare de decriptare. Însă, dacă Oracle nu emite o eroare, el decide că a decriptat, deși, după cum se poate observa, doar pentru clientul cu id-ul 18 este descifrabil outputul. De asemenea, în scriptul de seed este prezentă și o funcție care returnează automat după id numai liniile decriptate corect:



Funcția întoarce și un checksum pentru a valida datele în sine ulterior, dacă se dorește. Aceste două selecturi sunt prezente în fișierul *bro\_antrenor1\_cript\_show.sql*.

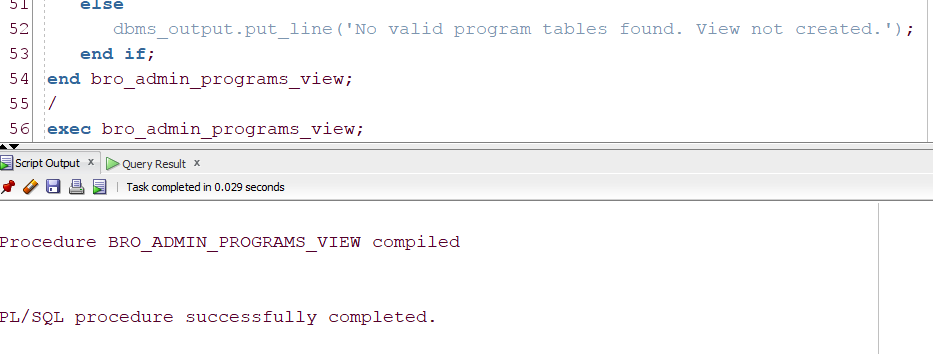
În acest moment, avem 1 antrenor cu o schemă creată și cu date, și adminul cu schema creată și date în ea. În continuare, vom da și restul permisiunilor necesare utilizatorilor, conform matricii entitate-utilizator. Scriptul asociat este în fișierul *sys\_users\_2.sql*. Acest script conține un rol de bază, adică cel pentru publicul general, și celelalte roluri care derivă din acesta, plus alte permisiuni individuale necesare.

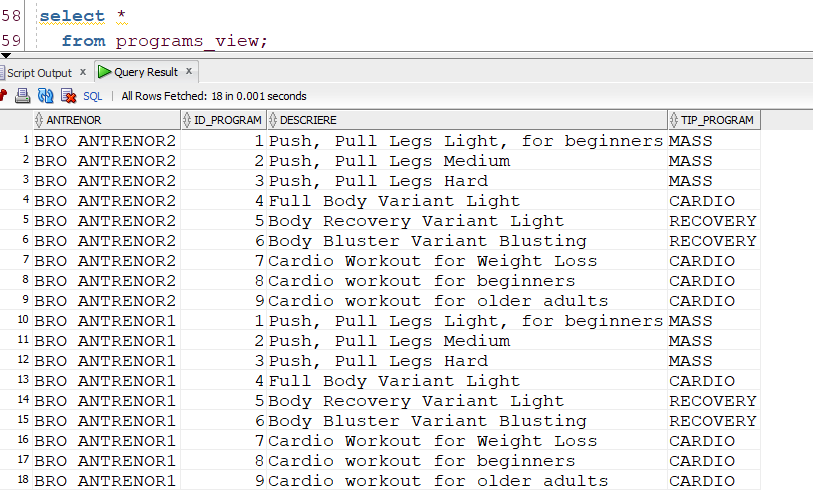
Pentru a ilustra criptarea în conexiunea lui client1 vom rula ultima comandă din fișierul *bro\_client1\_select\_cript.sql*



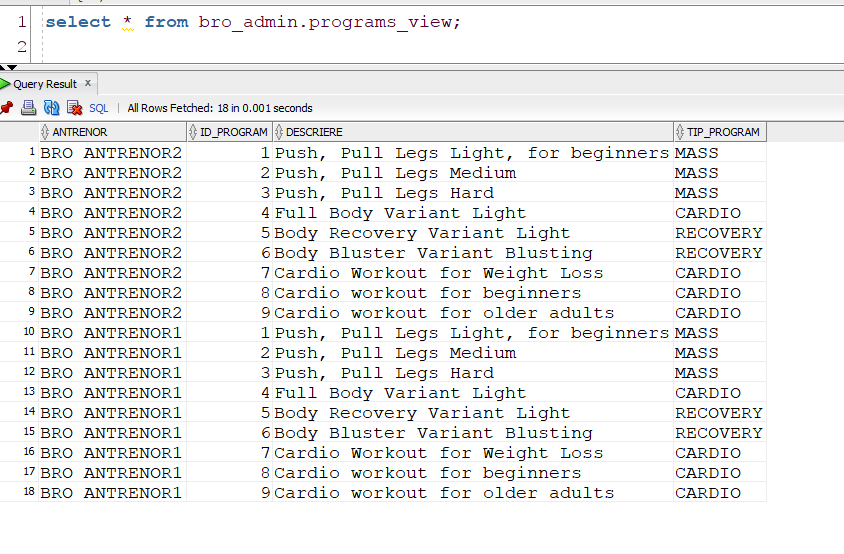
1. **Obiect dependent**

După ce am rulat scriptul *bro\_antrenor\_insert.sql* în schema mai multor antrenori, în cadrul sys vom rula *sys\_admin\_antrenor\_privilege.sql*. Acest script conține o procedură care oferă adminului permisiunea de select with grant option pe tabela **program** doar pentru schemele antrenorilor care au această tabelă. Avem nevoie de acest grant option, întrucât pentru a face mai ușoară selectarea tuturor antrenamentelor în admin vom crea un view care unește toate aceste programe. Scriptul pentru acest view se găsește în fișierul *bro\_admin\_programs\_view.sql* :





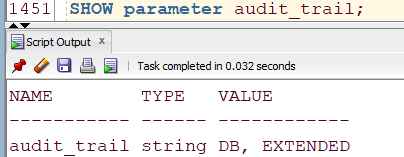
Deoarece, vom dori ca oricine să poată accesa acest view, tot în cadrul acestui script vom oferi grant select din admin pentru rolul de bază. (*Obs*: deși adminul nu vede dba roles, el poate da grant pe un rol existent, deoarece are implicit with grant option obiectele din schema sa). Dacă ne conectăm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



**6. Audit**

**6.1. Audit Standard**

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



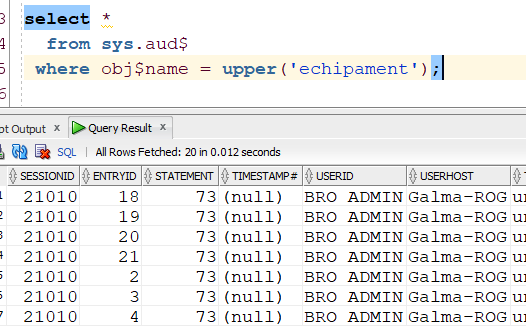
Vom audita astfel:

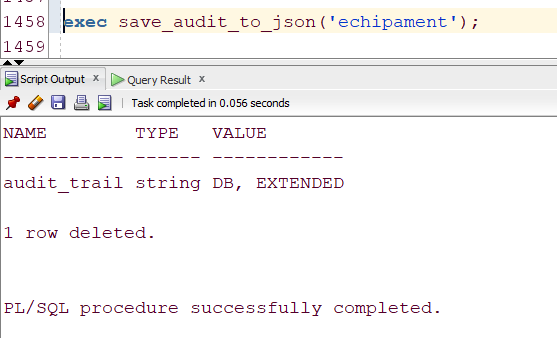
1. *bro\_admin.client\_extins*: inserarile și actualizările nereușite
2. *bro\_admin.echipament*: inserarile, acutalizările și ștergerile
3. *bro\_admin.account\_mapping*: inserarile, acutalizările și ștergerile
4. *bro\_admin.supliment* : inserările și actualizarile

Întrucât ținem auditările în baza de date, tabelul poate crește foarte mult, iar pentru a nu pierde datele vom crea în sys o procedură care va salva datele în format json într-un director, iar apoi, pentru a nu fi nevoie să se pună alarmă când se dorește salvarea, vom asocia unor joburi această procedură. Scriptul asociat se află în fișierul *sys\_audit\_1.sql.*

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

De exemplu, dacă în admin updatăm ‘fals’ echipamentele de mai multe ori (i.e. facem where data\_revizie=data\_revizie) și apoi rulăm procedura de export, vom avea:

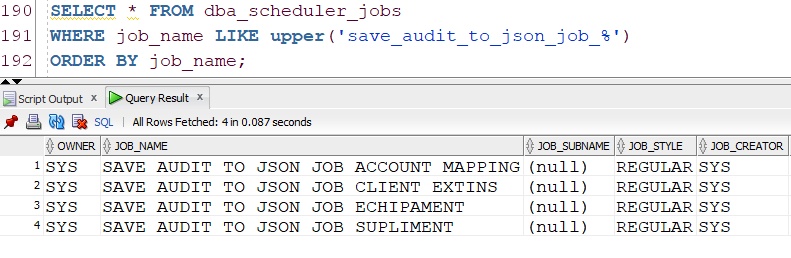




Iar în File Explorer avem fișierul:



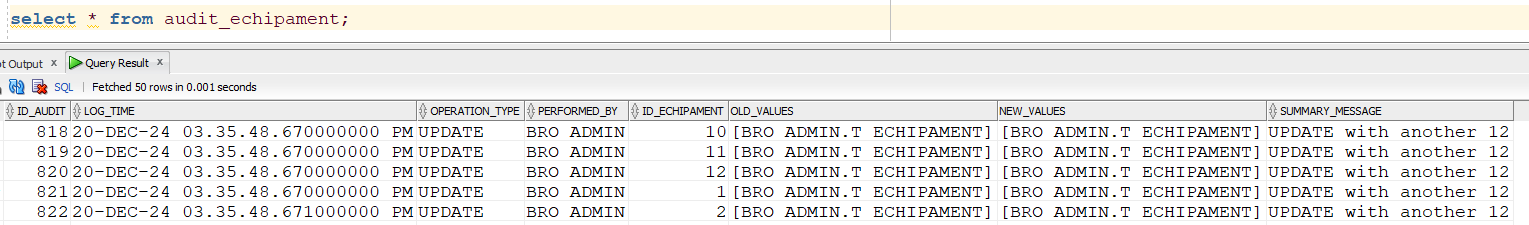
Pentru a vedea joburile se va rula:

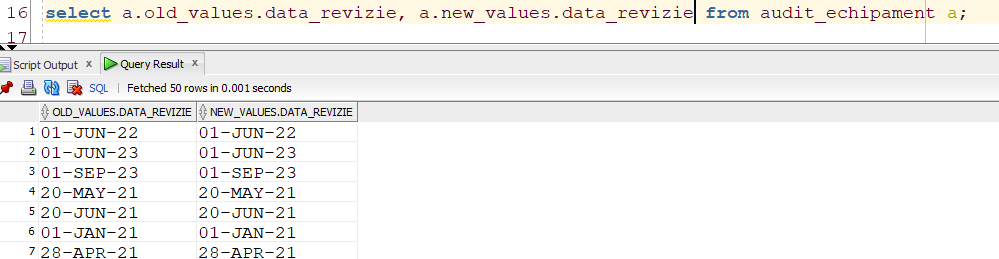


**6.2. Triggeri de Auditare**

Pentru triggerii de audit, în schema admin vom crea un tabel care va păstra operațiile DML pentru tabelul **echipament**. În acest audit se va memora atât valoarea veche, cât și cea nouă pentru datele inserate/modificate/șterse în cadrul unei comenzi. Scriptul asociat poate fi găsit în fișierul *bro\_admin\_audit.sql*.

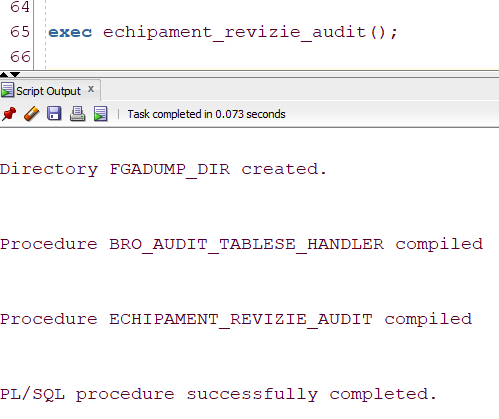
Pentru a ilustra triggerul de audit, vom rula din admin scriptul de update fals, apoi vom selecta din tabelul de audit:



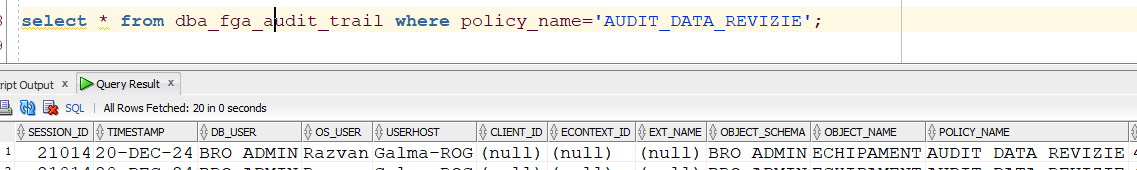


**6.3. FGA**

Pentru FGA vom audita din nou tabela **echipament** pentru a vedea schimbările din coloana **data\_revizie**. De asemenea, vom salva pe disk logurile asociate. Această salvare va fi făcută în cadrul handlerului. Scriptul asociat poate fi găsit în *sys\_audit\_2.sql*.

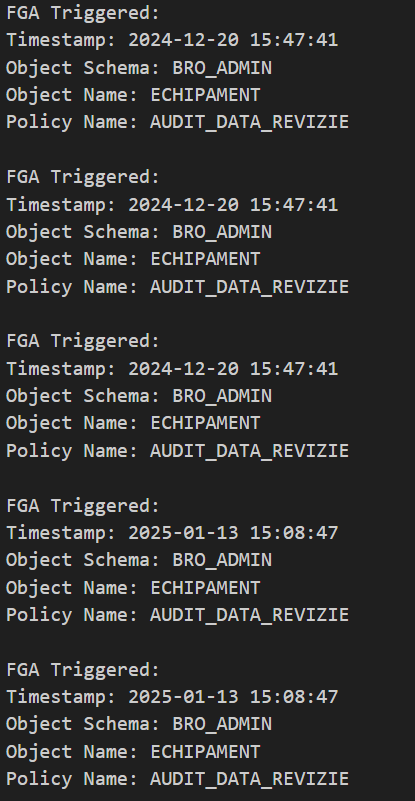


Pentru a ilustra FGA vom rula din nou update-ul fals din admin:



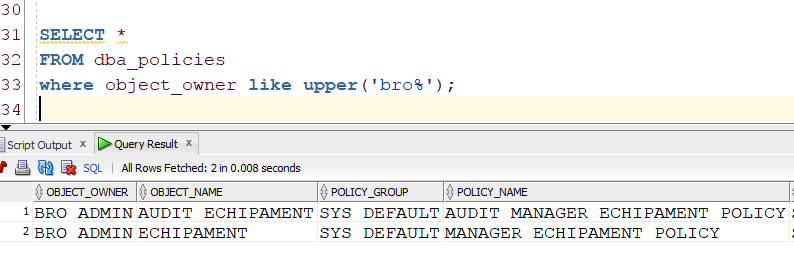
Fișierul txt de pe disk va salva, într-un mod append, momentele când auditul a fost activat:



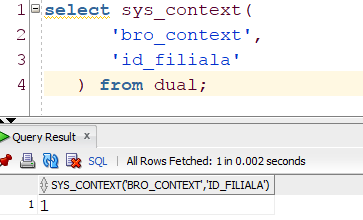


**7. Contextul aplicației**

Vom crea un context care, pentru fiecare manager de filială, va extrage din username filiala asociată. Numele managerilor în baza de date este astfel: bro\_manager\_filiala<id\_filiala>, de exemplu, pentru filiala 1 avem userul bro\_manager\_filiala1. De asemenea, contextul va fi folosit în două VPD-uri pentru a asigura că un manager de filială face operații DML doar pe filiala sa și poate accesa din tabelul **audit\_echipament** doar câmpurile care au legătură cu filiala sa. Scriptul asociat este în fișierul *sys\_context.sql*:

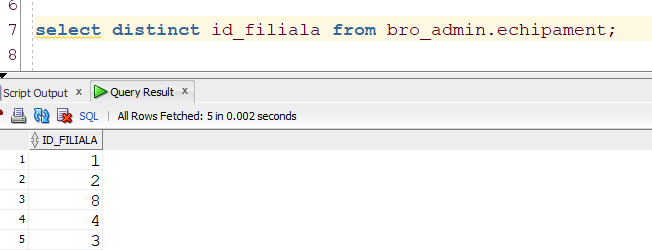


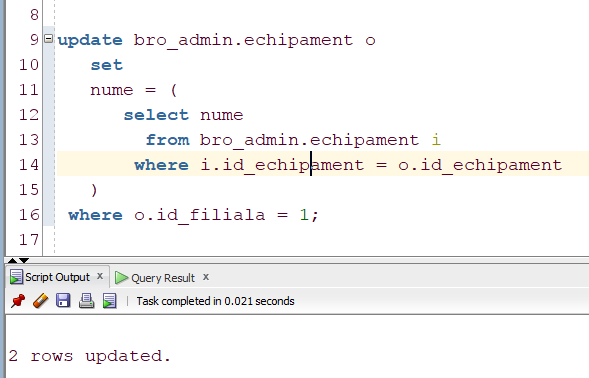
Conectandu-ne în manager1 putem vedea contextul:

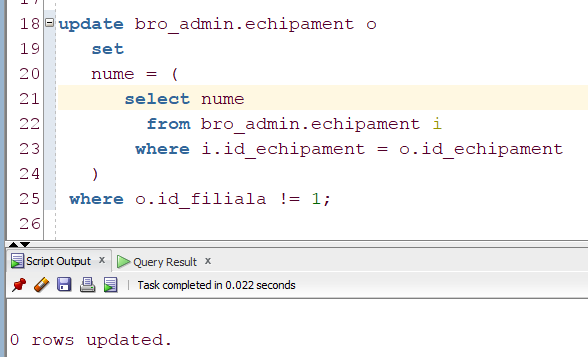


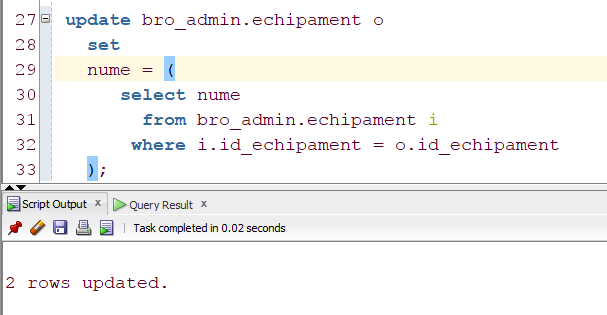
**7.1. VPD**

De exemplu, pentru VPD de update, dacă se va încerca update pentru echipamentele din filiala 1, se va afișa un număr de linii updatate. În schimb, dacă se încearcă update-ul pe echipamente din alte filiale, se vor afișa 0 linii updatate, fără nicio eroare, deși avem echipamente și în alte filiale. SQL-ul asociat managerului este în fișierul *bro\_manager\_filiala1\_context.sql.*

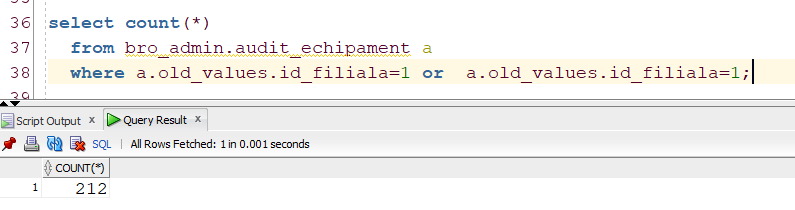


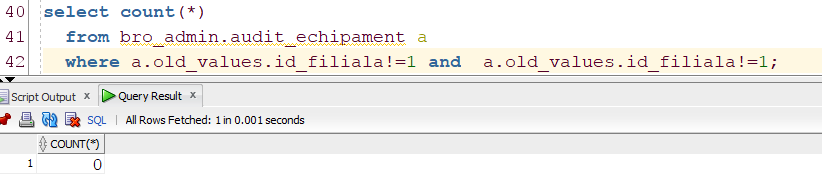


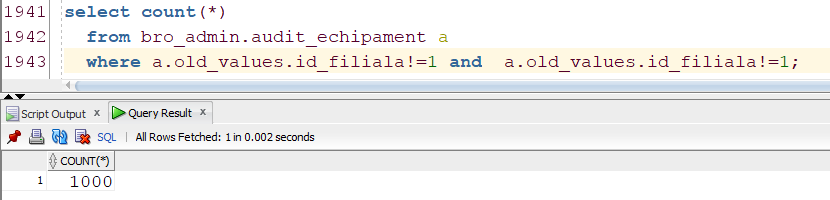




Politica de select pe tabelul de audit are rolul de a lăsa managerul unei filiale să vadă doar auditul pe echipamentele care fie au fost în filiala sa (i.e. old\_values.id\_filiala=1 aici), fie sunt (i.e. new\_values.id\_filiala=1):





Dacă rulăm în admin ultimul select, vom vedea că valoarea este diferită de 0: 

**8. SQL injection**

**8.1. Procedura Vulnerabilă**

Pentru SQL injection, să presupunem că antrenor1 vrea să creeze o procedură care permite utilizatorilor să vadă un program cu echipamentele care vor fi folosite în cadrul acestuia, filtrând echipamentele după data reviziei. Scriptul este în fișierul *bro\_antrenor1\_sql\_injection.sql*. Procedura va primi doi parametri: primul, id-ul programului, iar cel de-al doilea, data reviziei echipamentelor. Partea relevantă 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 ''%'**

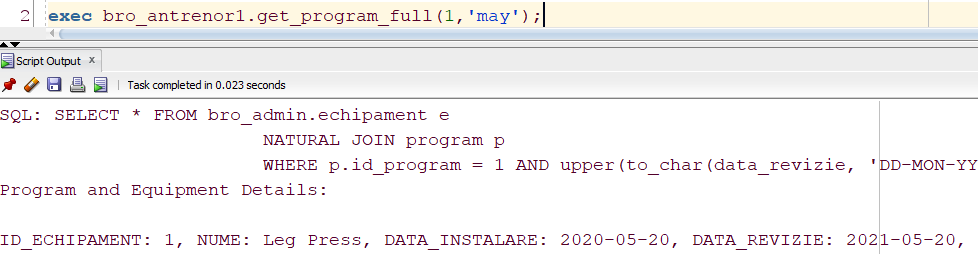
**|| upper(data\_inst)||**

**'%'''**

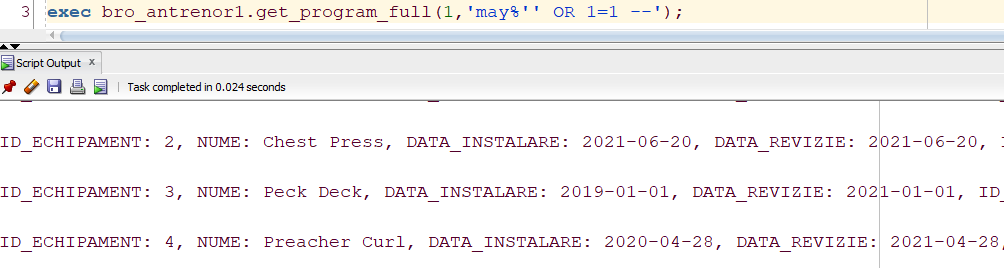
După cum se poate observa, inputul este direct concatenat în selectul care va fi transmis motorului bazei de date, fără a fi sanitizat.

Pentru a putea rula procedura din antrenor, în cadrul fișierului menționat vom da drepturi de execuție lui client1. Vom rula scriptul *bro\_client1\_sql\_injection.sql* pentru a demonstra un apel onest și două apeluri menite să arate vulnerabilitățile procedurii:

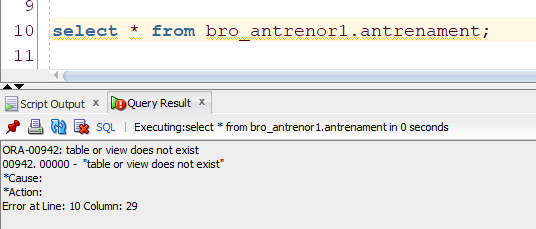
Apel onest:

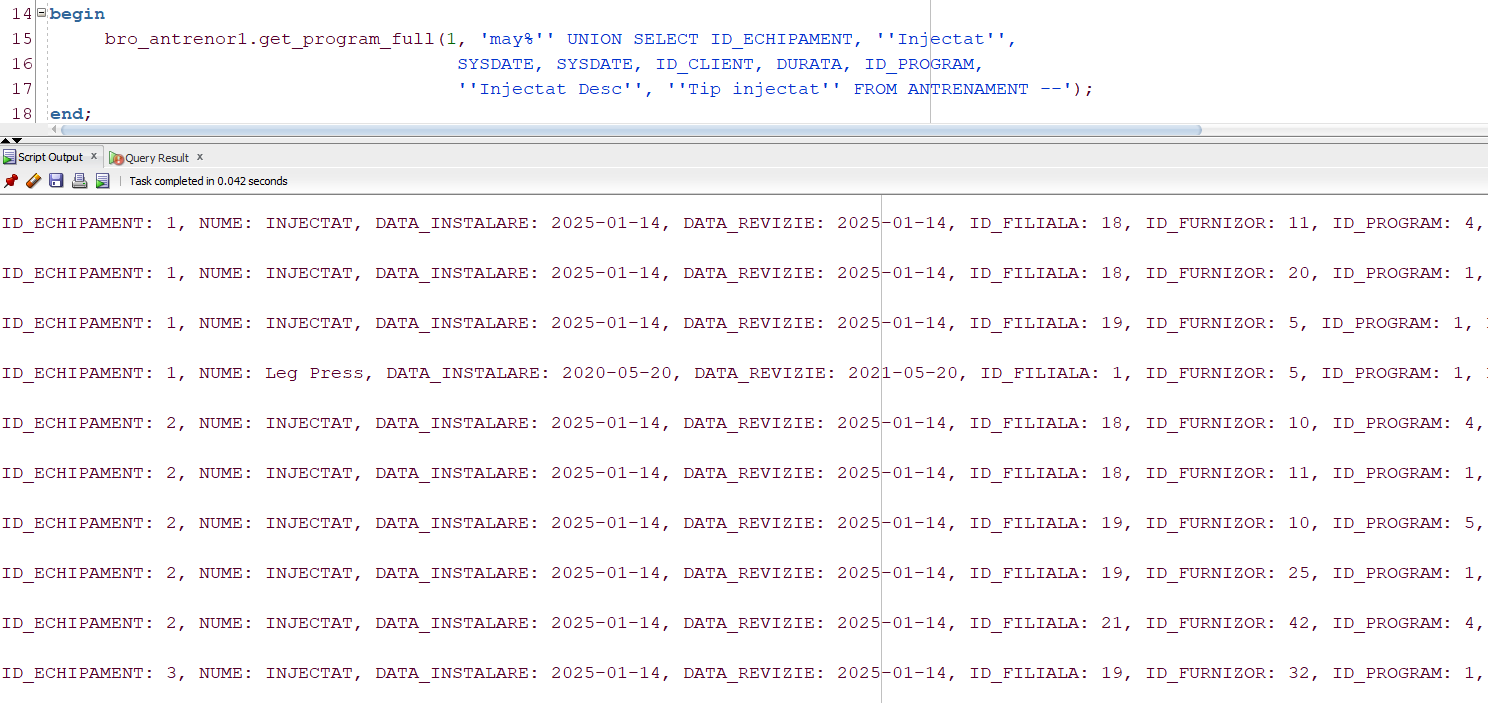


Apel care întoarce toate programele cu echipamentele asociate, subminând filtrarea:



Apel care întoarce toate antrenamentele, deși clientul nu are drept de select pe tabela antrenament:





**8.2. Procedura repartă**

Pentru a face procedura mai sigură la atacuri de tip injection, se va schimba crearea query-ului care folosește parametrii de intrare: se va înlocui 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'**

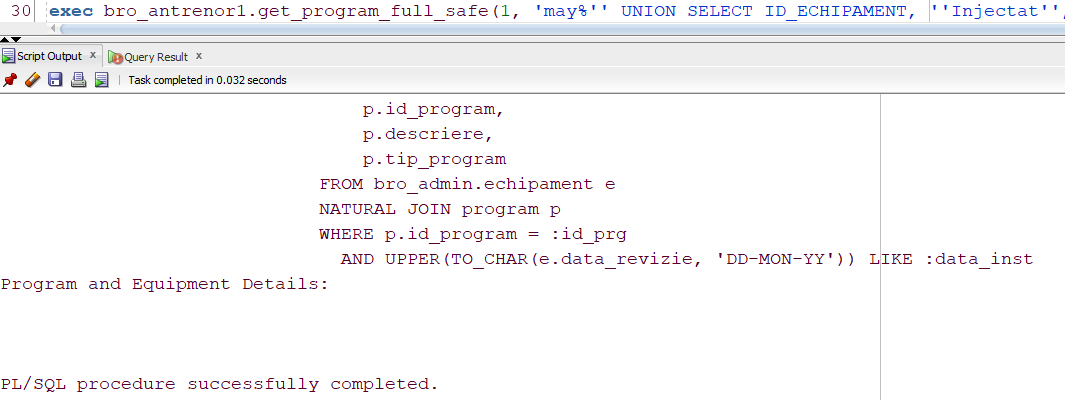
Iar, apelarea sa va fi urmatoarea:

**EXECUTE IMMEDIATE v\_sql BULK COLLECT**

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

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

Dacă se va încerca oricare dintre apelurile rău intenționate, procedura nu va întoarce nicio linie, întrucât în acest moment apelantul nu mai are posibilitatea de a altera structura efectivă a stringului de interogare:



1. **Mascarea datelor**

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

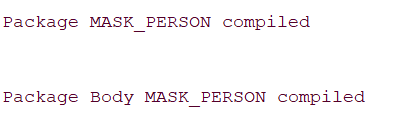
1. Valorile coloanelor numerice care nu sunt chei se vor schimba în valori care încep cu aceeași cifră și au aceeași lungime, restul de cifre vor fi random.
2. Valorile coloanelor de tip string se vor schimba astfel: prima dată se alege random dacă se va dubla lungimea stringului, după care se păstrează primul caracter, apoi se adaugă random până la noua lungime câte un caracter ‘\*’ sau ‘#’.
3. Cheile își vor păstra unicitatea dar vor putea avea dimensiunea până de 5 ori mai mare.

**9.1. Export**

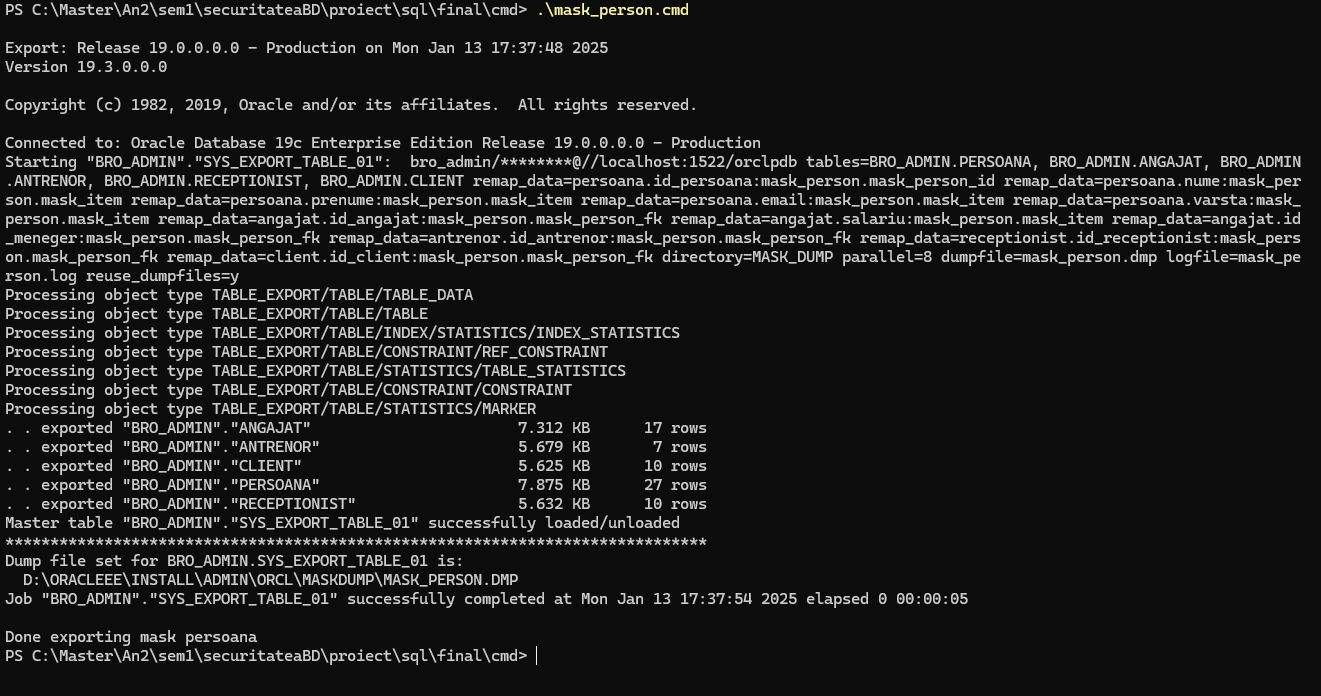
Pentru export se va crea în sys un nou director, iar adminul va primi permisiuni pe acesta. Pentru import se va crea un nou utilizator cu drepturi de import, ie *datapump\_imp\_full\_database*, pentru a putea remapa schema lui bro\_admin la schema noului utilizator. S-a ales acest model pentru a demonstra exportul și importul păstrând constrângerile inițiale ale tabelelor.

Pentru sys, fișierul asociat este *sys\_mask.sql*, în care se creează noul user, directorul și se dau drepturile asociate.

În fișierul *bro\_admin\_mask.sql* se găsește definirea pachetului care realizează maparea.



Comnada de realizare a mapării se află în *mask\_person.cmd*:

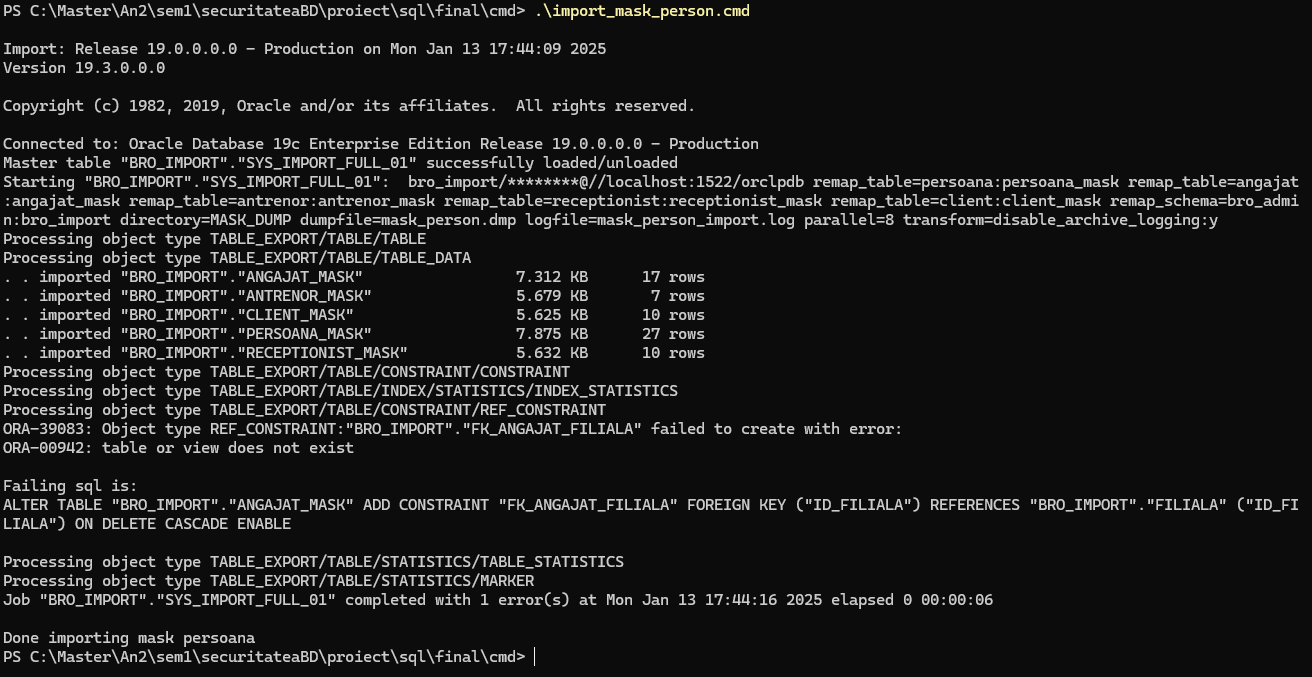


*Obs:* Deși am pus tables într-o anumită ordine, Oracle ia tables alfabetic, în minunata lor documentație nu am găsit nimic. Așa că a trebuit să preinitializez la mask\_person\_id și mask\_person\_fk cheile din persoană, dacă stateul de chei este gol. Nu cred că este un comportament normal, pe internet nu am găsit pe cineva să se plângă de acest aspect.

(Nu am folosit package body init, ie un begin, pentru a asigura că datele din persoană sunt cele din momentul exportului.)

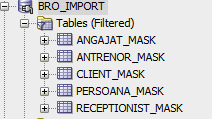
**9.2. Import**

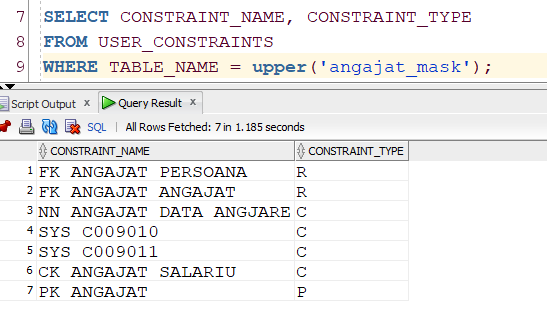
Pentru import în schema bro\_import, sa va rula scriptul *import\_mask\_person.cmd*:

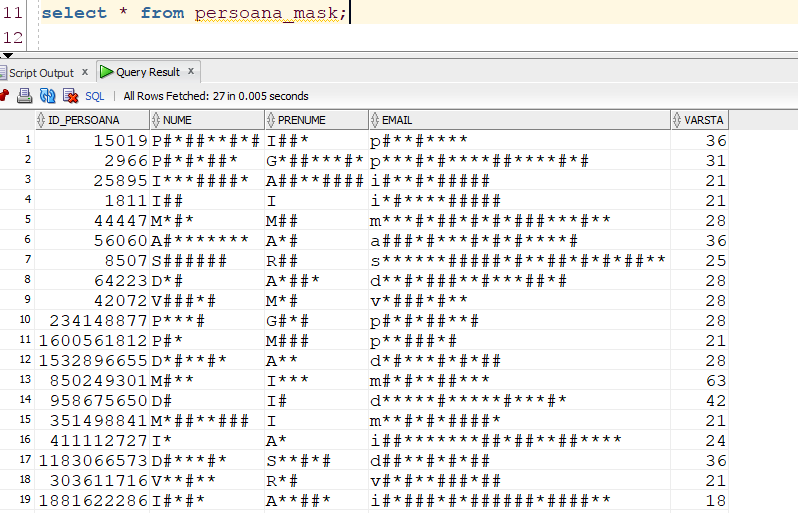


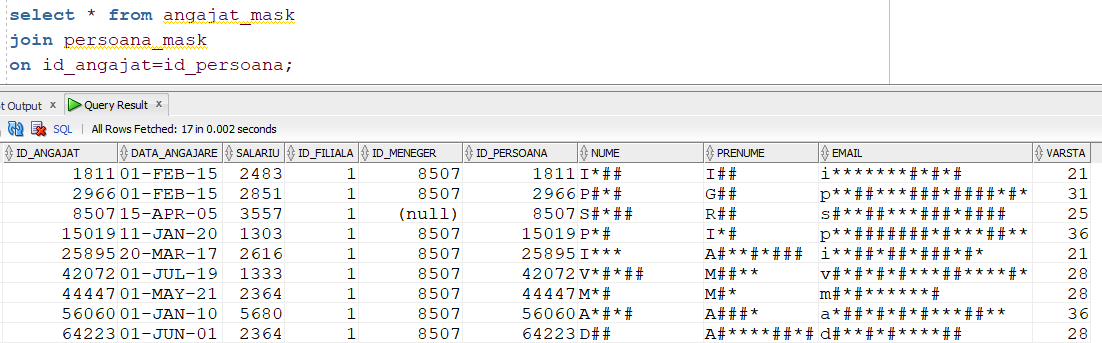
Întrucât am exportat doar tabelele **persoana**, **angajat**, **antrenor**, **receptionist** și **client**, nu și tabela **filiala**, și am păstrat la export constrângerile, la import vom avea o eroare care spune că nu se poate rezolva constrângerea de FK pentru tabela **filiala**. Pentru că suntem conștienți că nu am exportat acea tabelă, putem ignora această eroare, întrucât singurul lucru care se va întâmpla este că în tabela **angajat\_mask**, cea importată, nu vom mai avea acea constrângere de FK.

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









SQL-ul asociat userului de import se găsește în fișierul *bro\_import.sql*.