**Cuprins**

[1. Prezentarea modelului si a regulilor sale 1](#_Toc2605)

[1.1. Constrângeri pentru implementare 2](#_Toc1777)

[1.2. Diagrama Entitate Relație 3](#_Toc18367)

[1.3. Diagrama Conceptuală 4](#_Toc21111)

[1.4. Regulile de securitate 5](#_Toc13208)

[2. Procesele aplicației 6](#_Toc4682)

[2.1. Matricea Proces-Utilizator 7](#_Toc7713)

[2.2. Matricea Entitate-Proces 8](#_Toc14401)

[2.3. Matricea Entitate-Utilizator 9](#_Toc20646)

[3. Gestiunea Utilizatorilor și a Resurselor Computaționale 10](#_Toc13341)

[3.1. Configurarea Utilizatorilor și a Schemei 10](#_Toc8440)

[3.2. Memorie alocată pentru cateogriile de utilizatori 11](#_Toc32024)

[3.3. Profile 11](#_Toc1874)

[3.3.1. Plan de consum 11](#_Toc14533)

[3.3.2. Crearea Efectivă 12](#_Toc28728)

[3.4. Permisii Admin 13](#_Toc29162)

[4. Creare Bazei 14](#_Toc4706)

[4.1. Crearea Schemei Admin 14](#_Toc10051)

[4.1.1. Criptare în Schema Admin 15](#_Toc4341)

[4.2. Crearea Schemei Antrenor 16](#_Toc31056)

[4.2.1. Criptare în Schema Antrenor 17](#_Toc13465)

[5. Obiect dependent 19](#_Toc14235)

[6. Audit 21](#_Toc6704)

[6.1. Audit Standard 21](#_Toc20756)

[6.2. Triggeri de Auditare 23](#_Toc27695)

[6.3. FGA 24](#_Toc2171)

[7. Contextul aplicației 26](#_Toc9201)

[7.1. VPD 27](#_Toc5361)

[8. SQL injection 29](#_Toc31358)

[8.1. Procedura Vulnerabilă 29](#_Toc22558)

[8.2. Procedura repartă 31](#_Toc11414)

[9. Mascarea datelor 32](#_Toc24217)

[9.1. Export 33](#_Toc5118)

[9.2. Import 34](#_Toc17520)

[10. Codul SQL al aplicatiei 37](#_Toc26730)

[10.1. Admin 37](#_Toc9981)

[10.1.2. bro\_admin\_audit.sql 49](#_Toc19571)

[10.1.3. bro\_admin\_create\_tables.sql 54](#_Toc22131)

[10.1.4. bro\_admin\_criptare.sql 136](#_Toc12153)

[10.1.5. bro\_admin\_mask.sql 141](#_Toc19656)

[10.1.6. bro\_admin\_programs\_view.sql 151](#_Toc4965)

[10.1.7. bro\_admin\_update\_echipament\_fals.sql 153](#_Toc23777)

[10.2 Antrenor 154](#_Toc32163)

[10.2.1. bro\_antrenor\_insert.sql 154](#_Toc25644)

[10.2.2. bro\_antrenor1\_cript\_show.sql 165](#_Toc21713)

[10.3. Client 174](#_Toc23059)

[10.3.1. bro\_client1\_select\_cript.sql 174](#_Toc14576)

[10.3.2 bro\_client1\_sql\_injection.sql 176](#_Toc13282)

[10.4.1. bro\_import.sql 177](#_Toc4638)

[10.5 Manager 178](#_Toc2337)

[10.5.1. bro\_manager\_filiala1\_context.sql 178](#_Toc12835)

[10.7. SYS 180](#_Toc876)

[10.7.1. sys\_admin\_antrenor\_privilege.sql 180](#_Toc15947)

[10.7.2. sys\_audit\_1.sql 181](#_Toc5010)

[10.7.3. sys\_audit\_2.sql 189](#_Toc7838)

[10.7.4 sys\_context.sql 193](#_Toc12422)

[10.7.5. sys\_mask.sql 199](#_Toc9879)

[10.7.6. sys\_users\_1.sql 200](#_Toc12852)

[11. Codul CMD 238](#_Toc12892)

[11.1. import\_mask\_person.cmd 238](#_Toc17304)

[11.2. mask\_person.cmd 238](#_Toc10940)

[11.3. seed\_antrenor.cmd 239](#_Toc29948)

[12. Link repository 239](#_Toc26277)

**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 ).

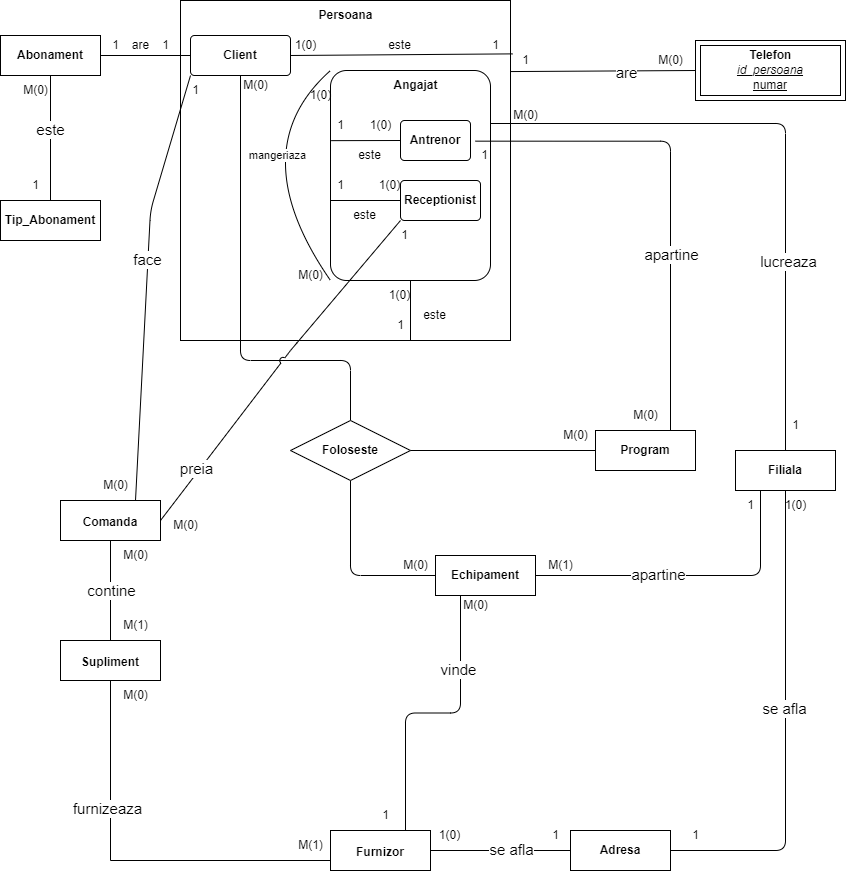
O aprovizionare a firmei se va face cu un singur supliment o dată.

Nu se înregistrează stocuri de suplimente.

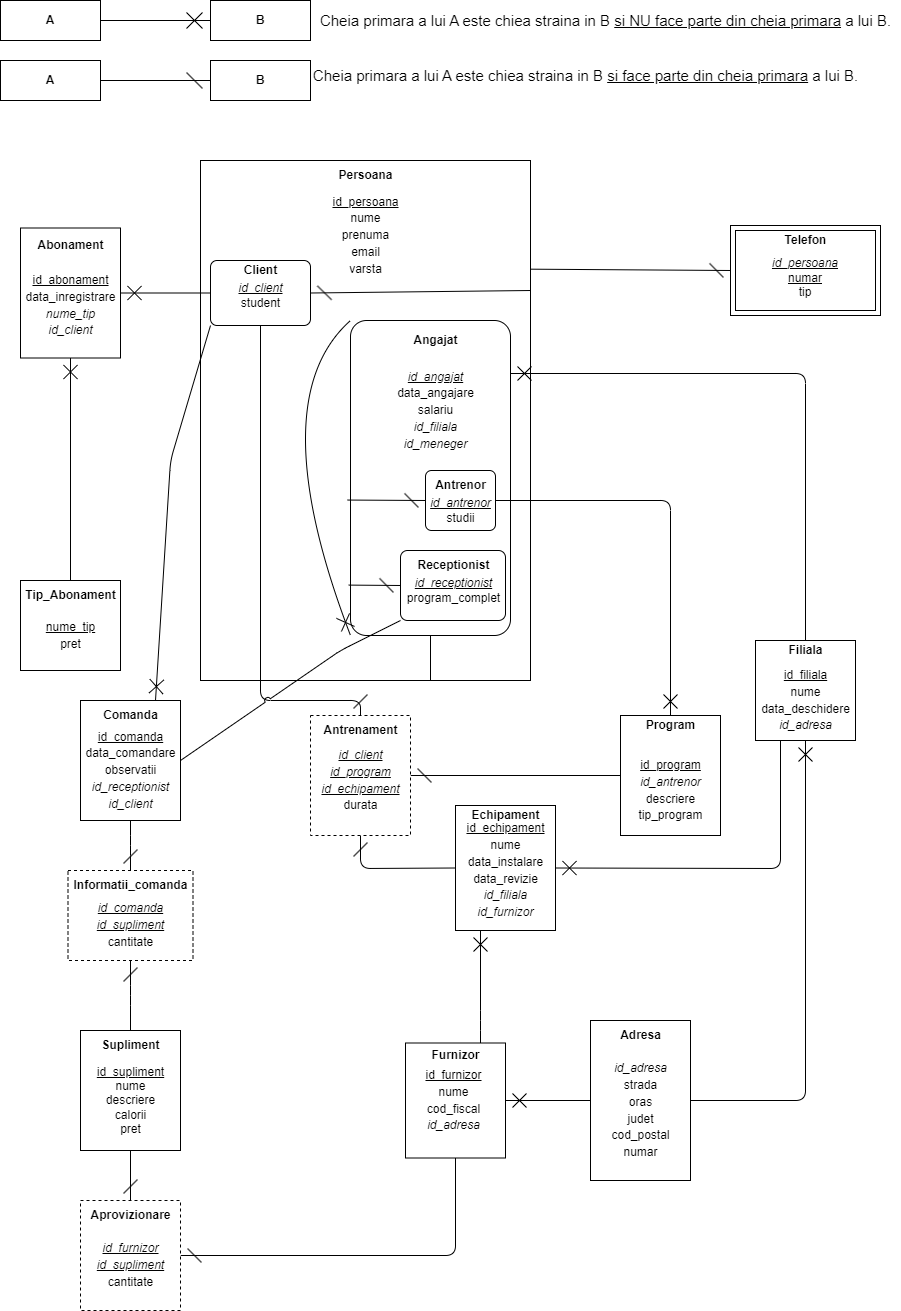
Filialele trebuie să aibă adresele diferite între ele.

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**

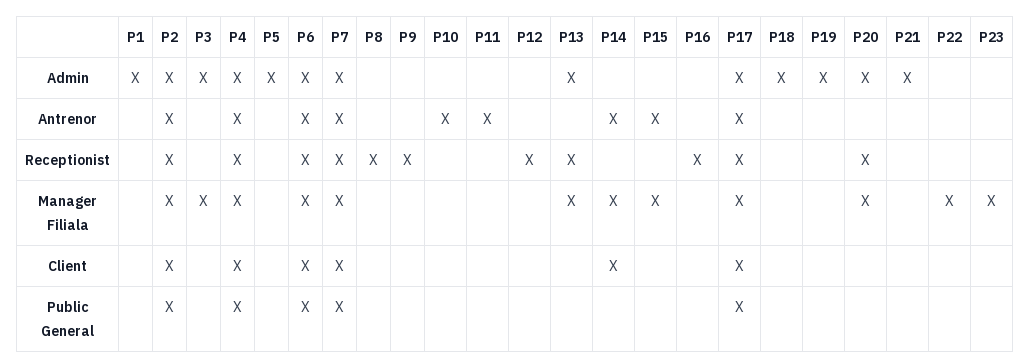
În cadrul proiectului se vor cripta datele antrenamentelor fiecărui client, astfel încât doar el și antrenorul său să poată vedea informațiile. De asemenea, se vor realiza două VPD-uri (Virtual Private Database) pentru a asigura că managerii de filială pot face operații DML doar pe echipamentele din 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**

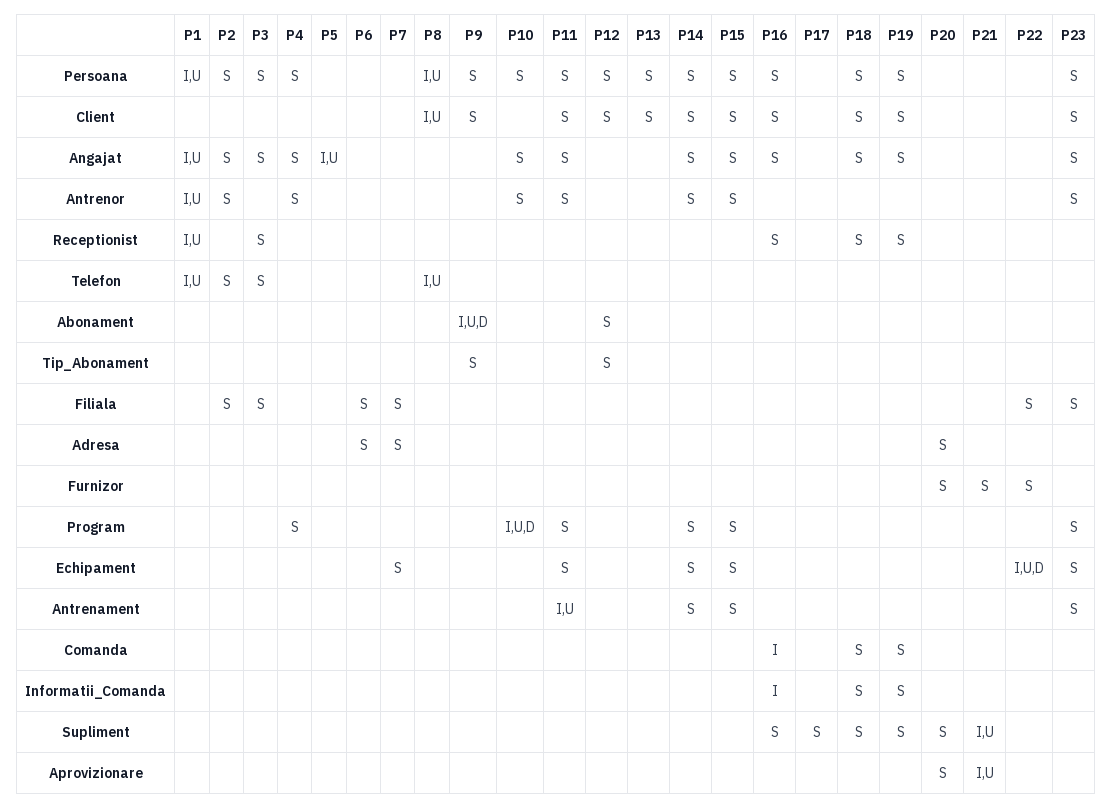
Procesele care pot fi inițiate în cadrul bazei sunt:

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**

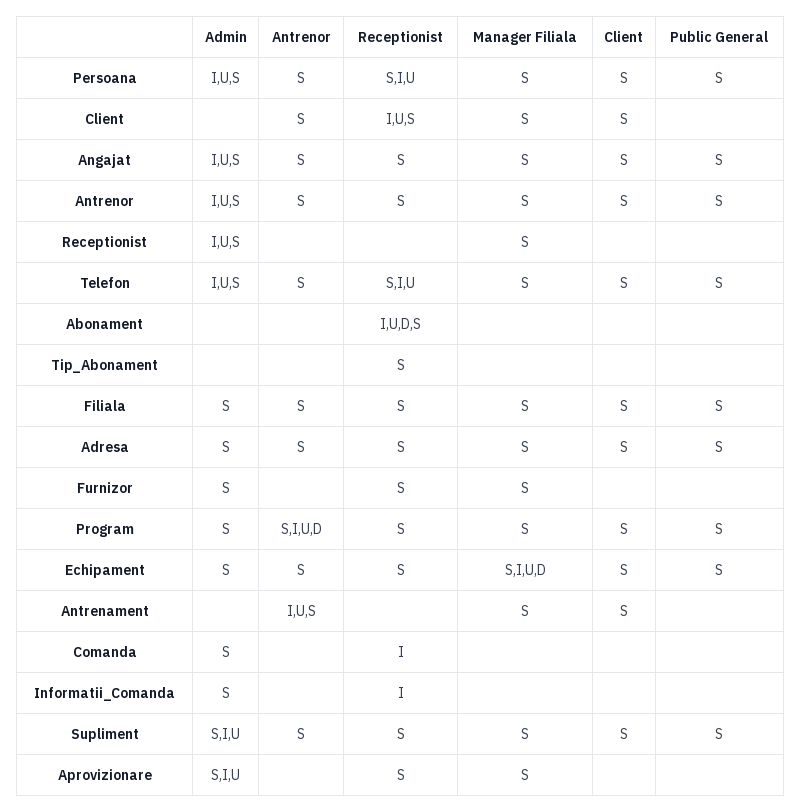


**2.2. Matricea Entitate-Proces**



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

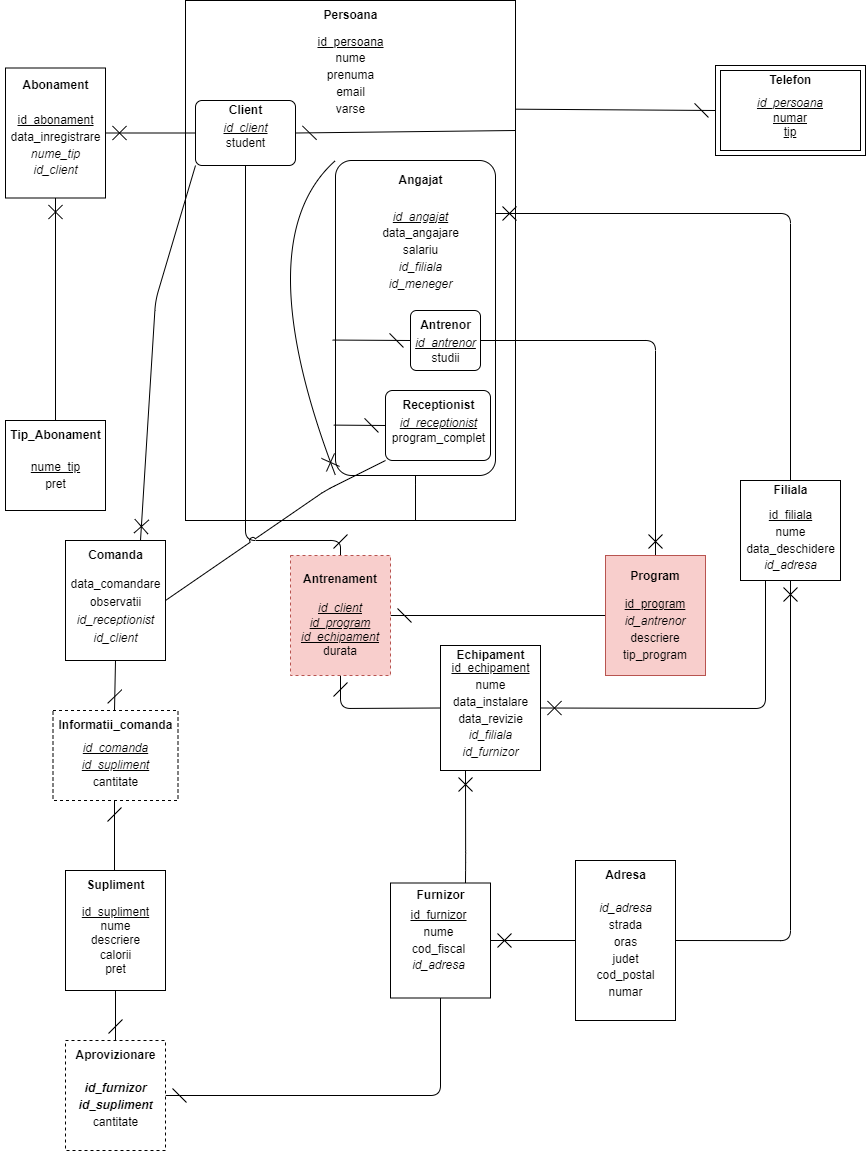
**2.3. Matricea Entitate-Utilizator**



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%

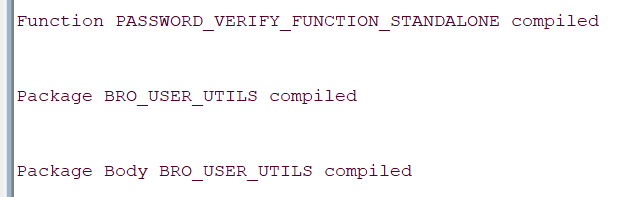
Recepționiști: 15%

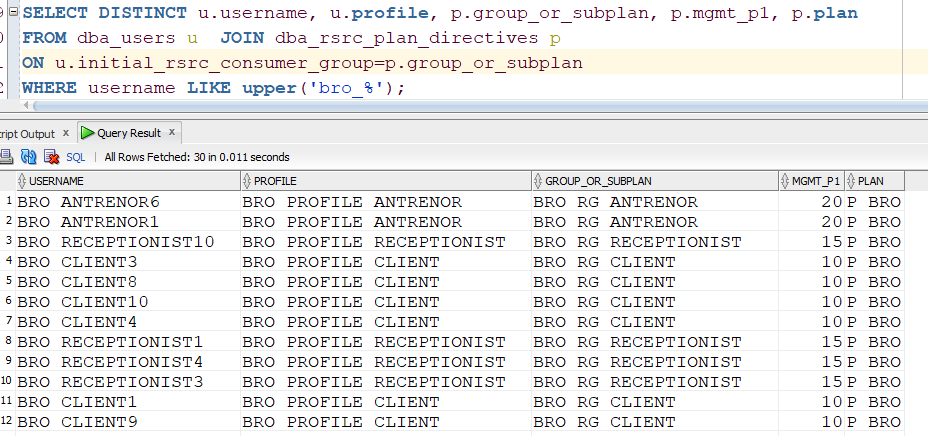
Antrenori: 20%

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:

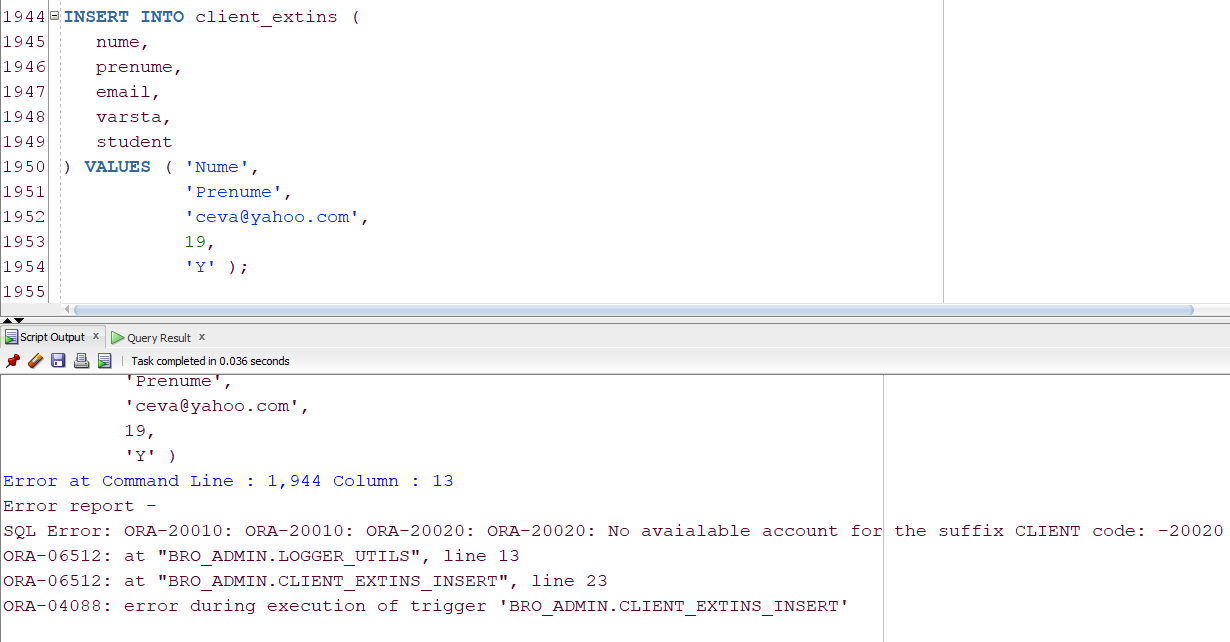
**CREATE TABLE account\_mapping(**

**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:

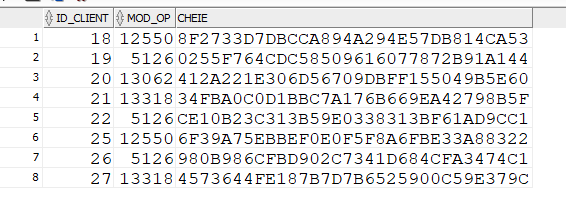


**4.1.1. Criptare în Schema Admin**

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:

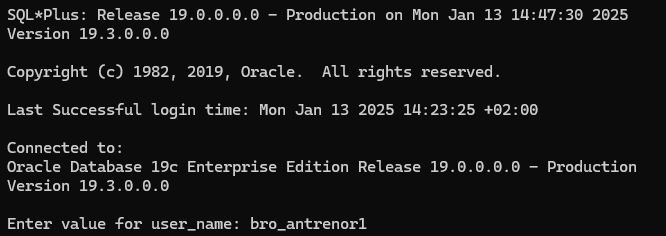


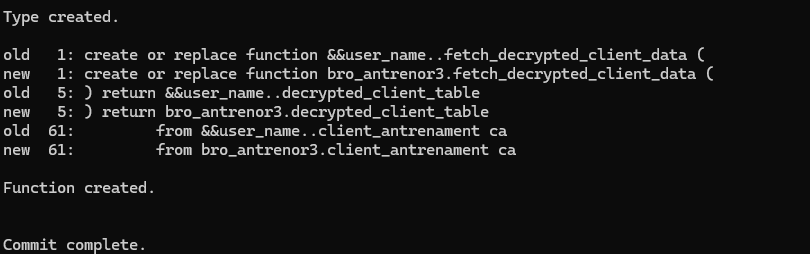
Pentru ca un client să-și ia cheia unică, tot în cadrul scriptului este creată o funcție *get\_client\_key*, care, pentru username-ul luat din contextul *userenv*, va întoarce cheia de criptare asociată.

**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:



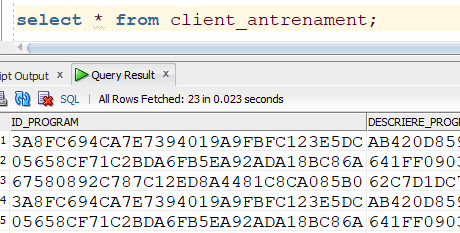


**4.2.1. Criptare în Schema Antrenor**

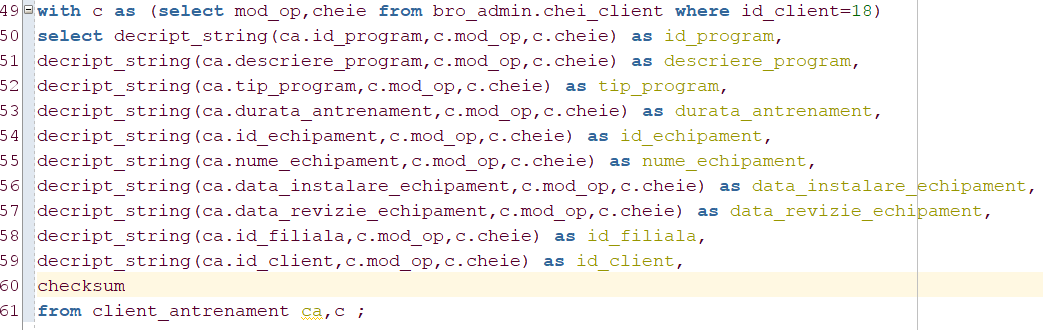
Întrucât antrenorii trebuie să aibă acces la cheile de criptare ale clienților, li se va da acces la tabelul de chei, iar, totodată, scriptul creează și un view care va arăta datele antrenamentelor pentru clienți într-un mod criptat cu cheile individuale. Pentru a regăsi datele și a le valida, sunt create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

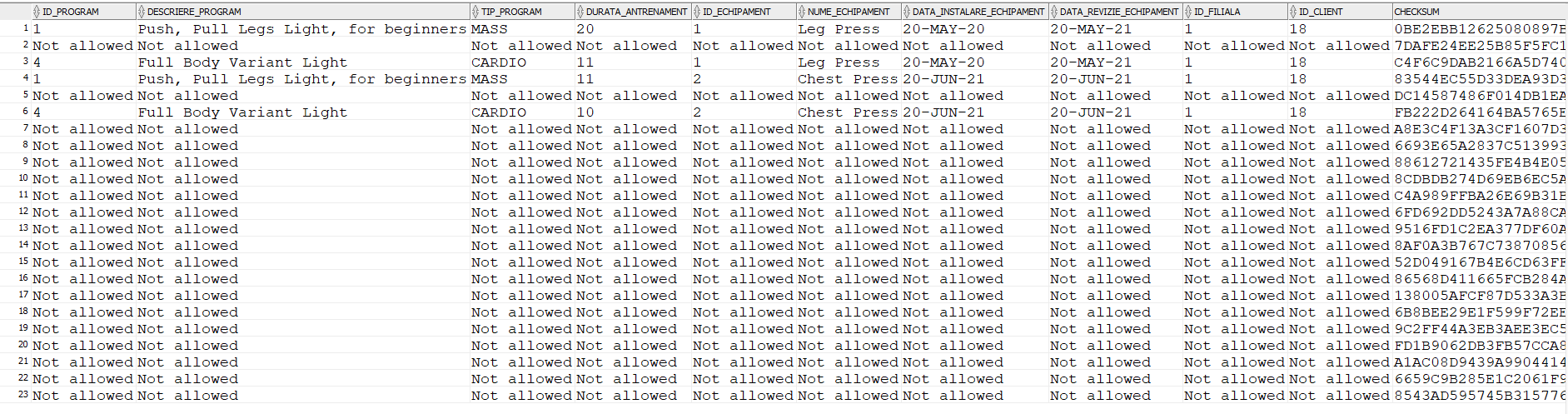
Pentru a putea insera date vom rula din antrenor1 scriptul *bro\_antrenor\_insert.sql*.

Pentru a vedea datele criptate vom apela:

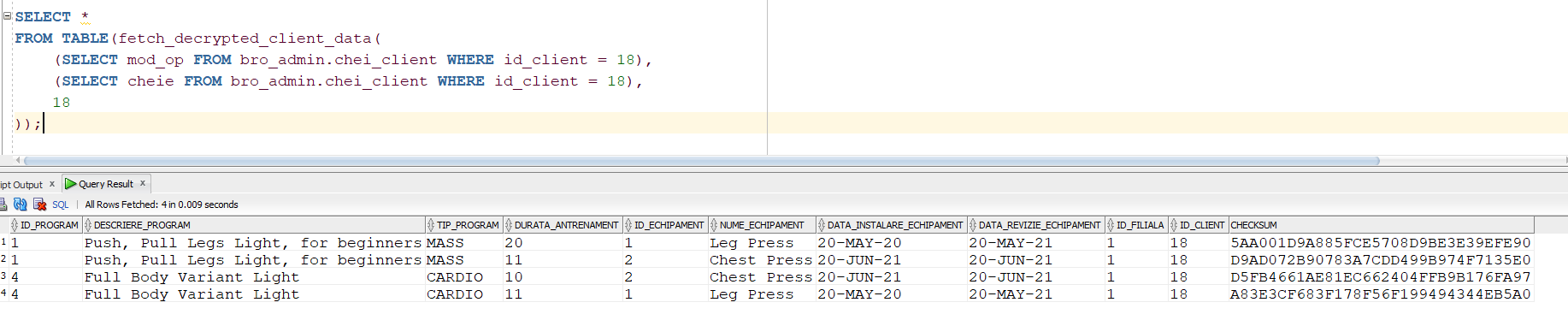


De exemplu, dacă dorim să decriptăm din tabel totul, dar doar pentru clientul cu id-ul 18, outputul va arăta:





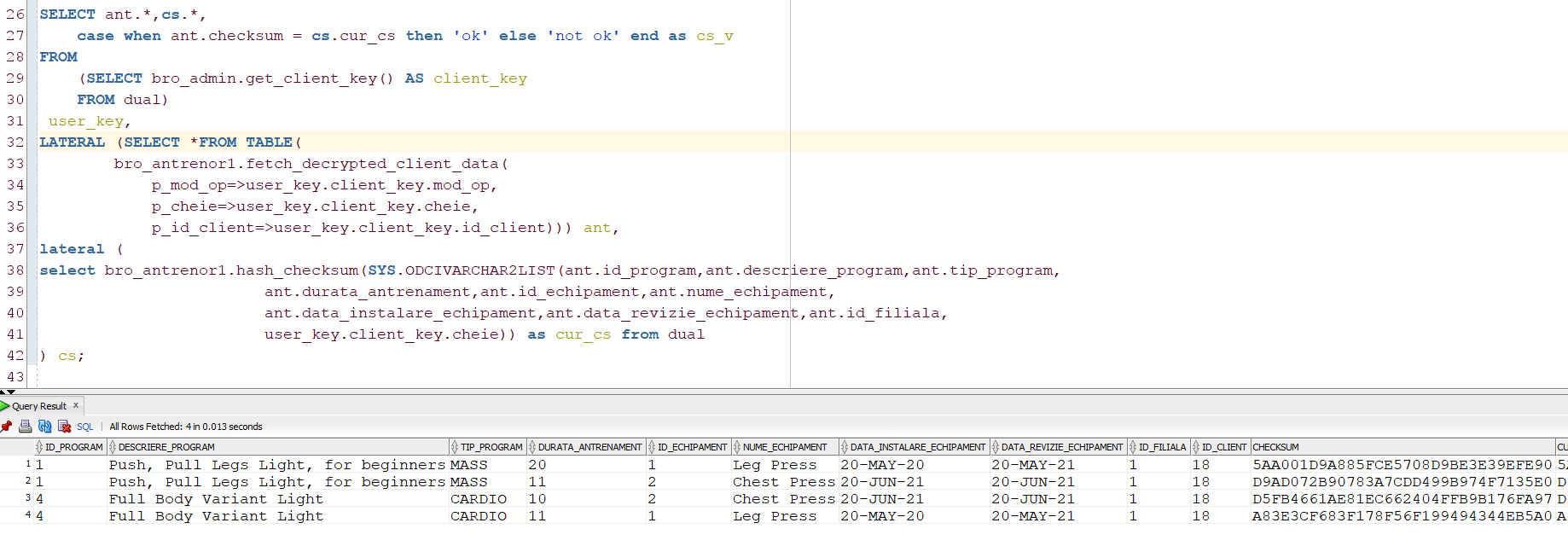
“Not allowed” este pus manual în caz de eroare de decriptare. Însă, dacă Oracle nu emite o eroare, el decide că a decriptat, deși, după cum se poate observa, doar pentru clientul cu id-ul 18 este descifrabil outputul. De asemenea, în scriptul de seed este prezentă și o funcție care returnează automat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

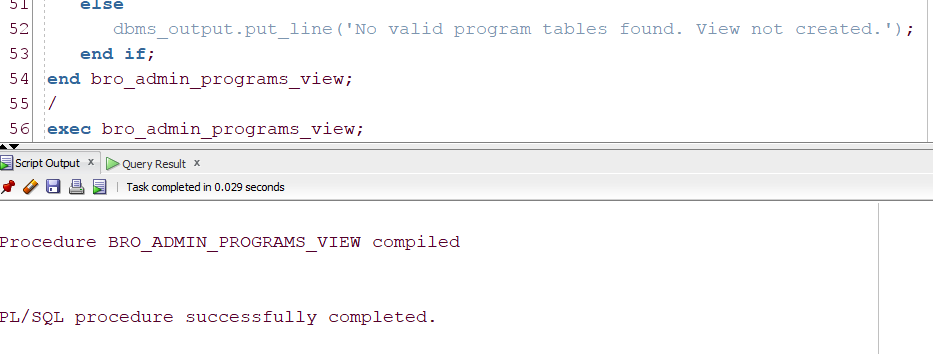
În acest moment, avem 1 antrenor cu o schemă creată și cu date, și adminul cu schema creată și date în ea. În continuare, vom da și restul permisiunilor necesare utilizatorilor, conform matricii entitate-utilizator. Scriptul asociat este în fișierul *sys\_users\_2.sql*. Acest script conține un rol de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

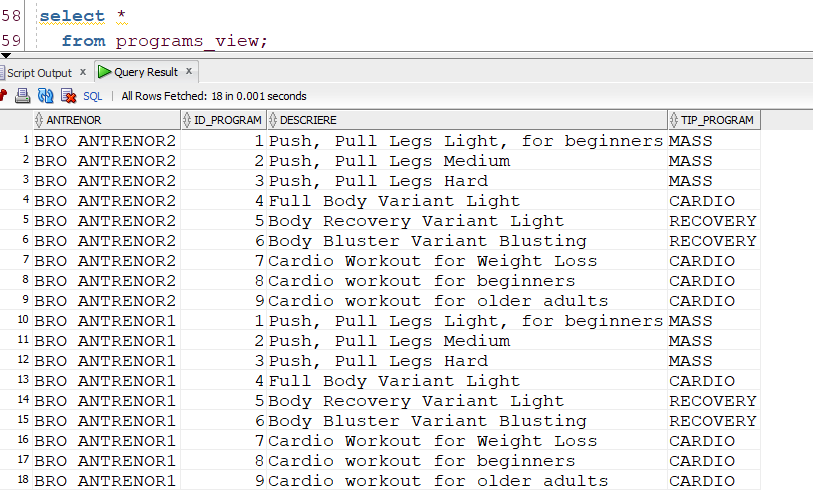
Pentru a ilustra criptarea în conexiunea lui client1 vom rula ultima comandă din fișierul *bro\_client1\_select\_cript.sql*



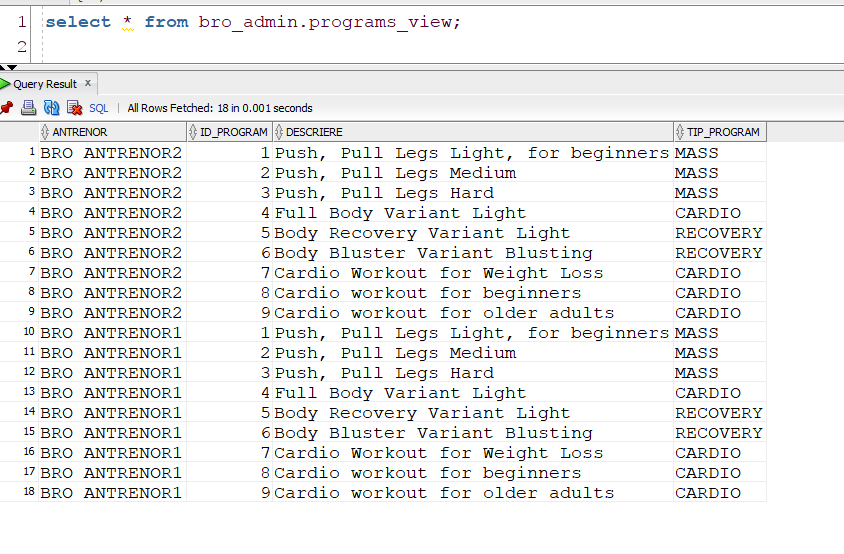
1. **Obiect dependent**

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* :





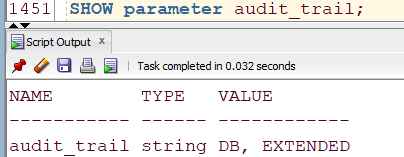
Deoarece, vom dori ca oricine să poată accesa acest view, tot în cadrul acestui script vom oferi grant select din admin pentru rolul de bază. (*Obs*: deși adminul nu vede dba roles, el poate da grant pe un rol existent, deoarece are implicit with grant option obiectele din schema sa). Dacă ne conectăm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



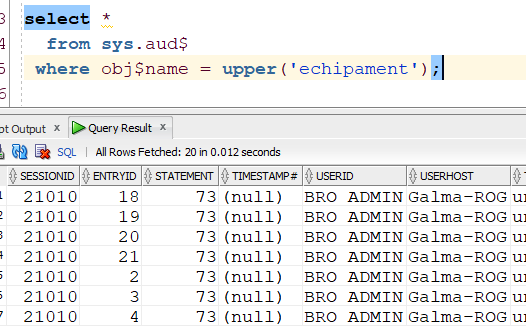
Vom audita astfel:

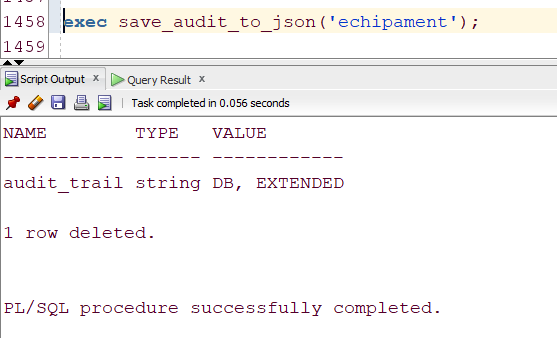
1. *bro\_admin.client\_extins*: inserarile și actualizările nereușite
2. *bro\_admin.echipament*: inserarile, acutalizările și ștergerile
3. *bro\_admin.account\_mapping*: inserarile, acutalizările și ștergerile
4. *bro\_admin.supliment* : inserările și actualizarile

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:

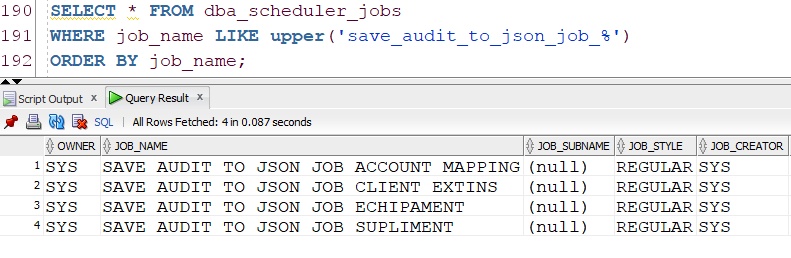




Iar în File Explorer avem fișierul:



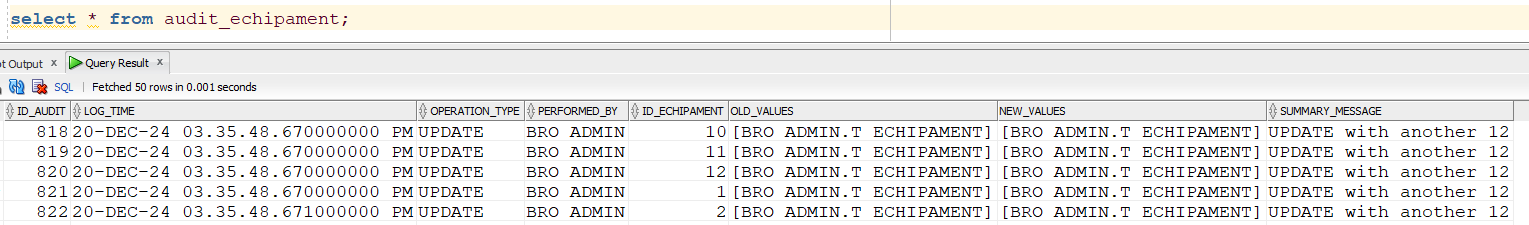
Pentru a vedea joburile se va rula:

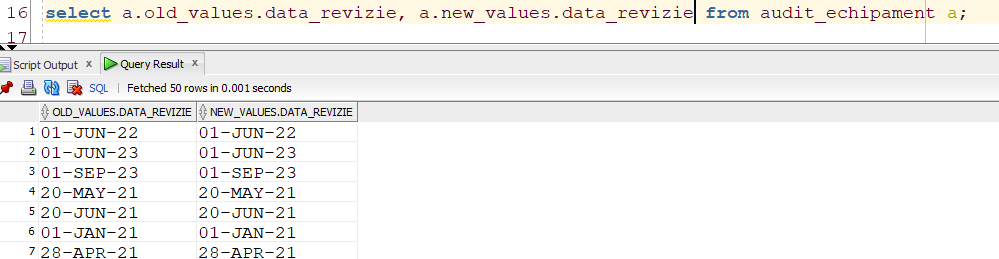


**6.2. Triggeri de Auditare**

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*.

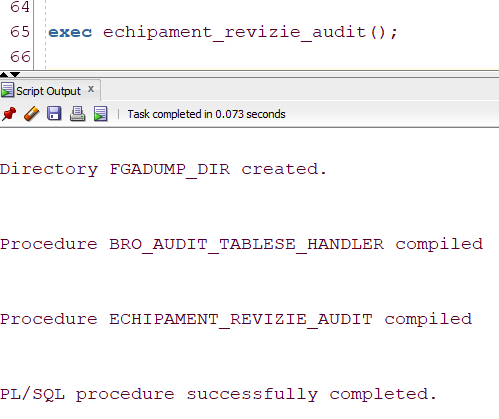
Pentru a ilustra triggerul de audit, vom rula din admin scriptul de update fals, apoi vom selecta din tabelul de audit:



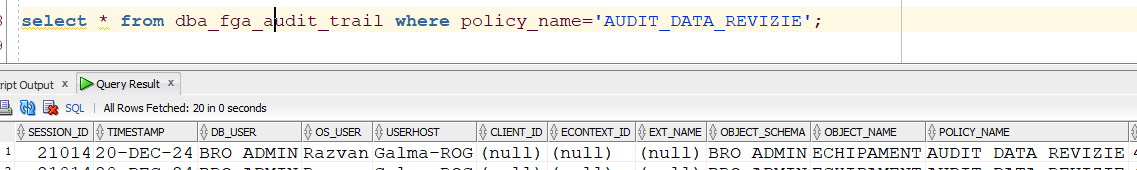


**6.3. FGA**

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*.

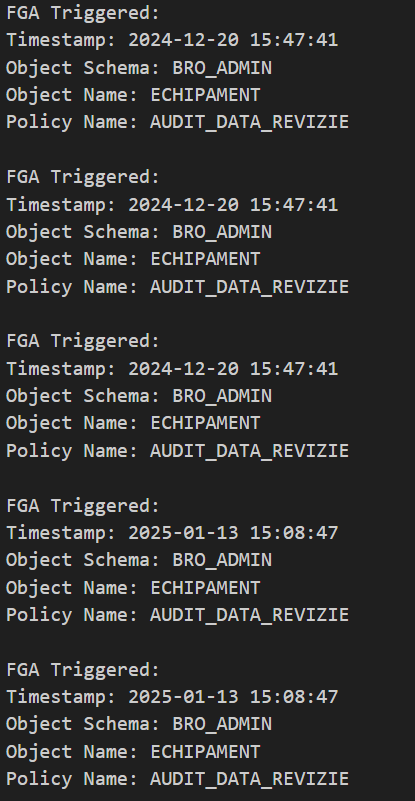


Pentru a ilustra FGA vom rula din nou update-ul fals din admin:



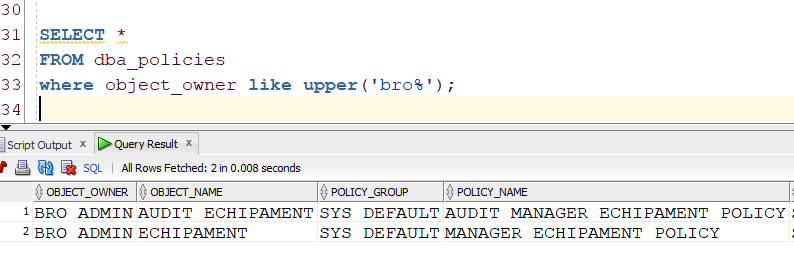
Fișierul txt de pe disk va salva, într-un mod append, momentele când auditul a fost activat:



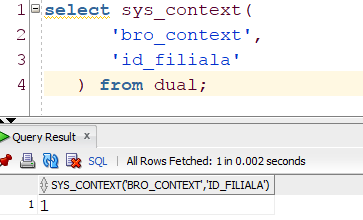


**7. Contextul aplicației**

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*:

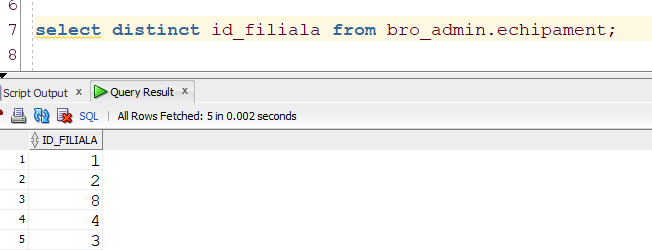


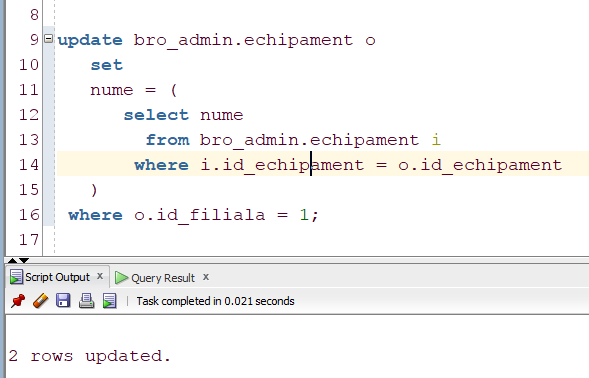
Conectandu-ne în manager1 putem vedea contextul:

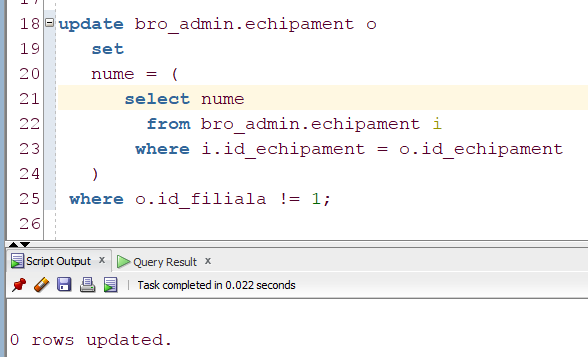


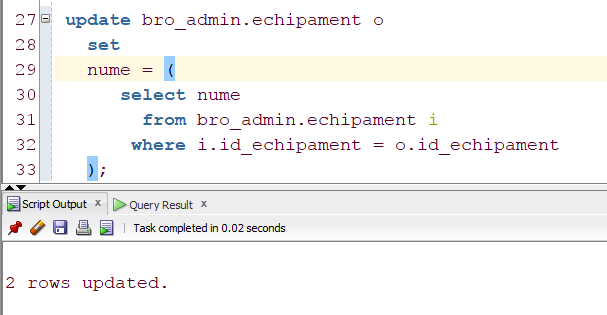
**7.1. VPD**

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.*

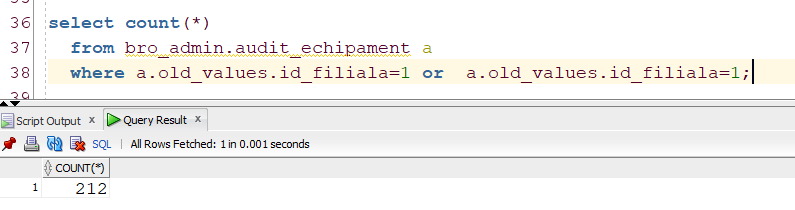


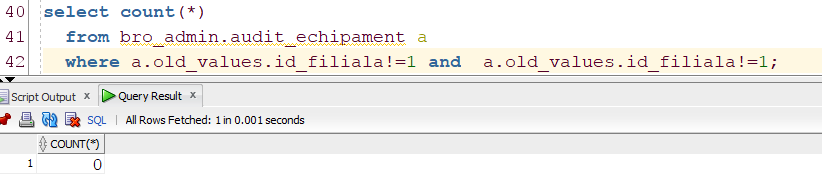


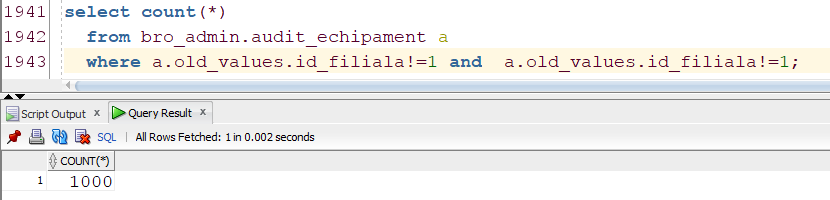




Politica de select pe tabelul de audit are rolul de a lăsa managerul unei filiale să vadă doar auditul pe echipamentele care fie au fost în filiala sa (i.e. old\_values.id\_filiala=1 aici), fie sunt (i.e. new\_values.id\_filiala=1):





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 ''%'**

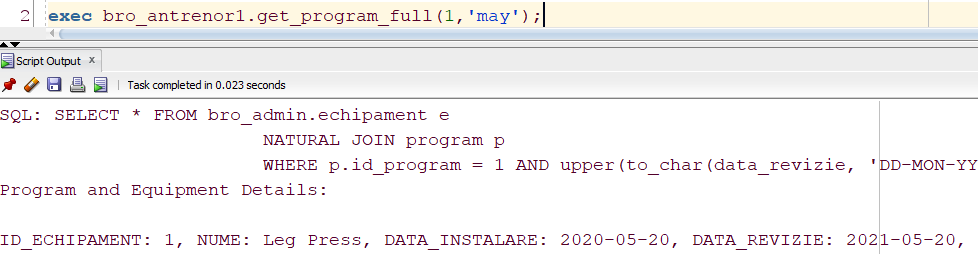
**|| upper(data\_inst)||**

**'%'''**

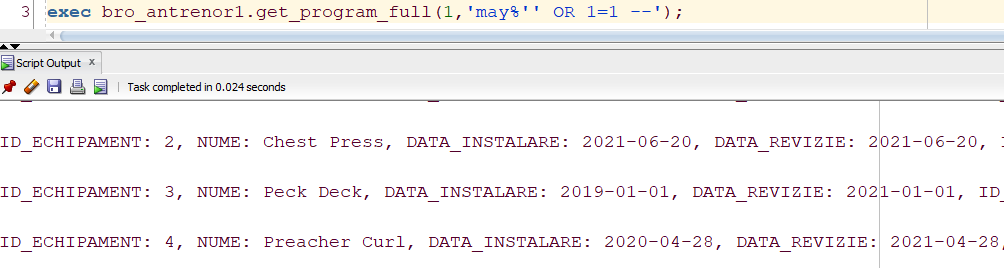
După cum se poate observa, inputul este direct concatenat în selectul care va fi transmis motorului bazei de date, fără a fi sanitizat.

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:

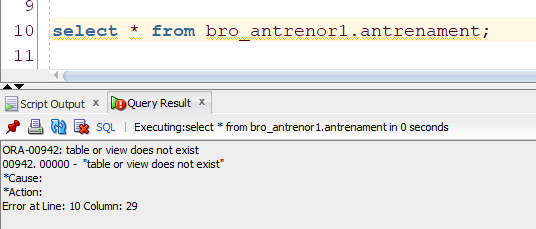
Apel onest:

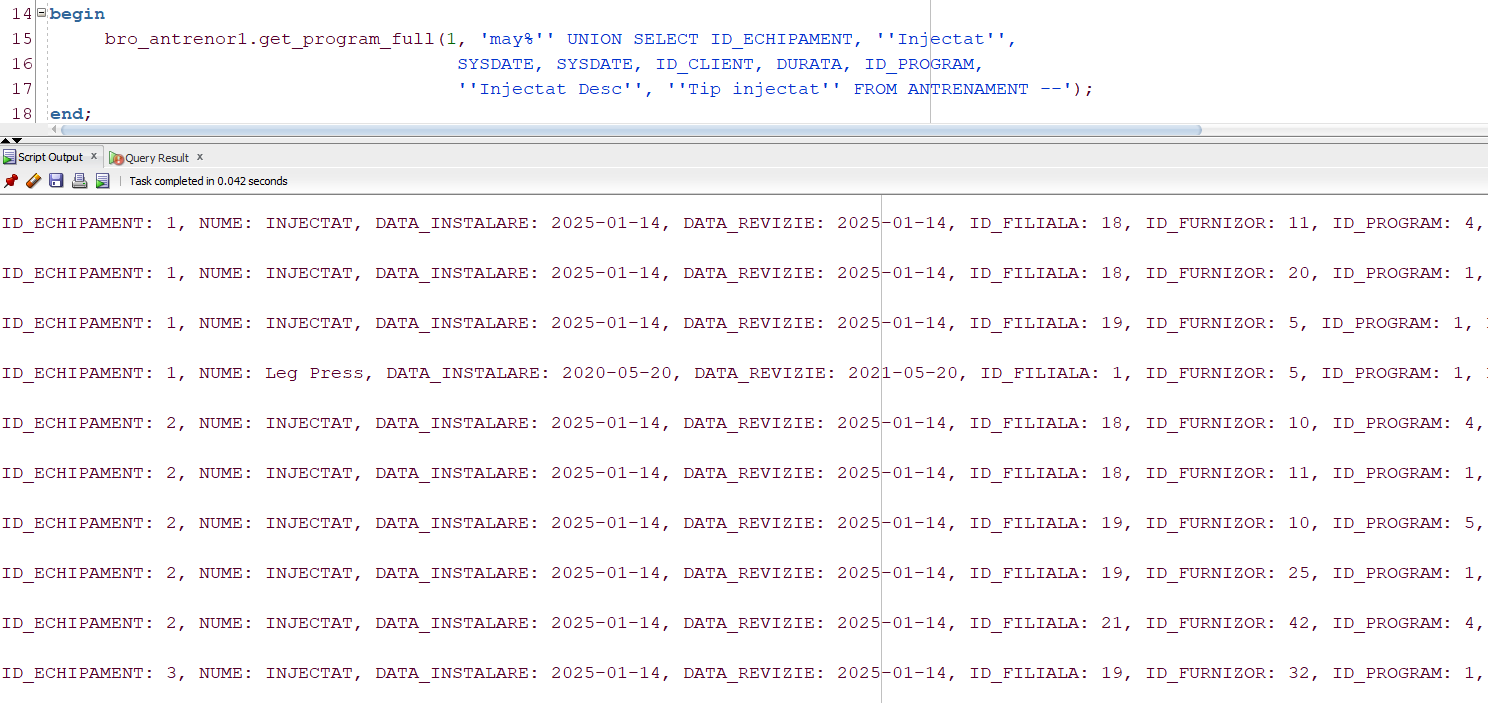


Apel care întoarce toate programele cu echipamentele asociate, subminând filtrarea:



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'**

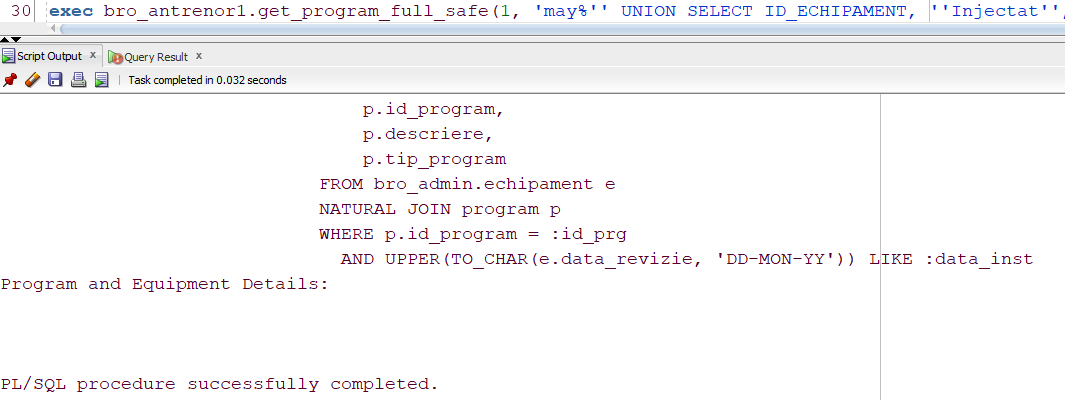
Iar, apelarea sa va fi urmatoarea:

**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:

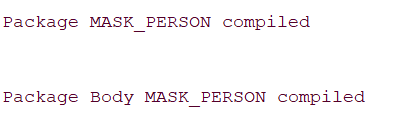
1. Valorile coloanelor numerice care nu sunt chei se vor schimba în valori care încep cu aceeași cifră și au aceeași lungime, restul de cifre vor fi random.
2. Valorile coloanelor de tip string se vor schimba astfel: prima dată se alege random dacă se va dubla lungimea stringului, după care se păstrează 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**

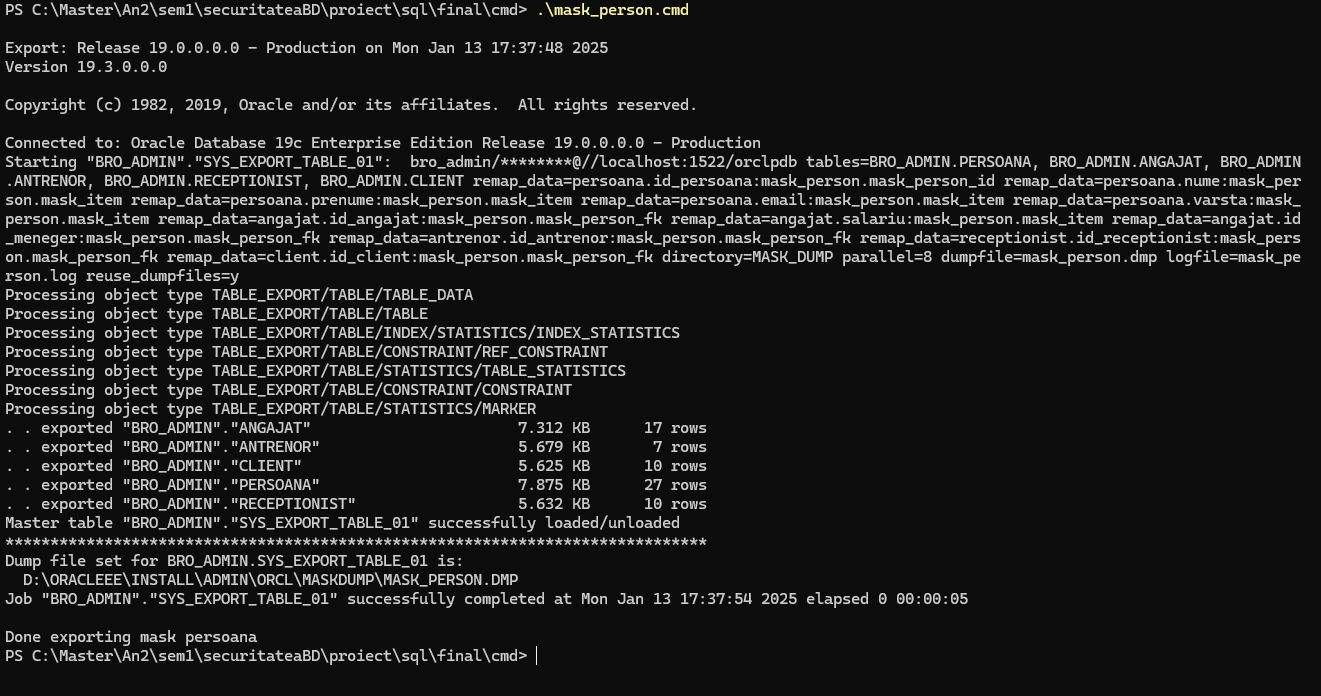
Pentru export se va crea în sys un nou director, iar adminul va primi permisiuni pe acesta. Pentru import se va crea un nou utilizator cu drepturi de import, ie *datapump\_imp\_full\_database*, pentru a putea remapa schema lui bro\_admin la schema noului utilizator. S-a ales acest model pentru a demonstra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

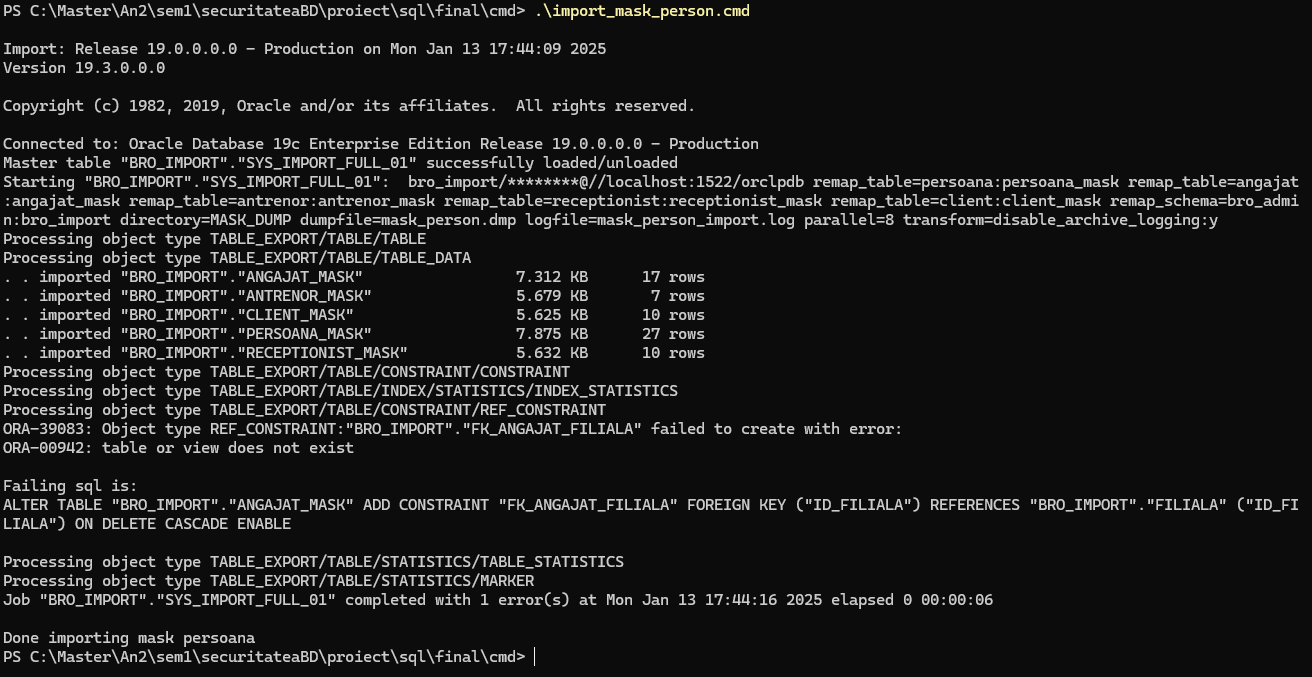


*Obs:* Deși am pus tables într-o anumită ordine, Oracle ia tables alfabetic, în minunata lor documentație nu am găsit nimic. Așa că a trebuit să preinitializez la mask\_person\_id și mask\_person\_fk cheile din persoană, dacă stateul de chei este gol. Nu cred că este un comportament normal, pe internet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

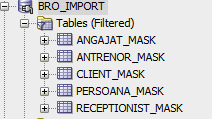
**9.2. Import**

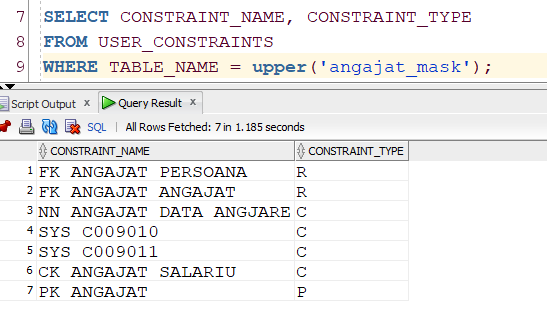
Pentru import în schema bro\_import, sa va rula scriptul *import\_mask\_person.cmd*:

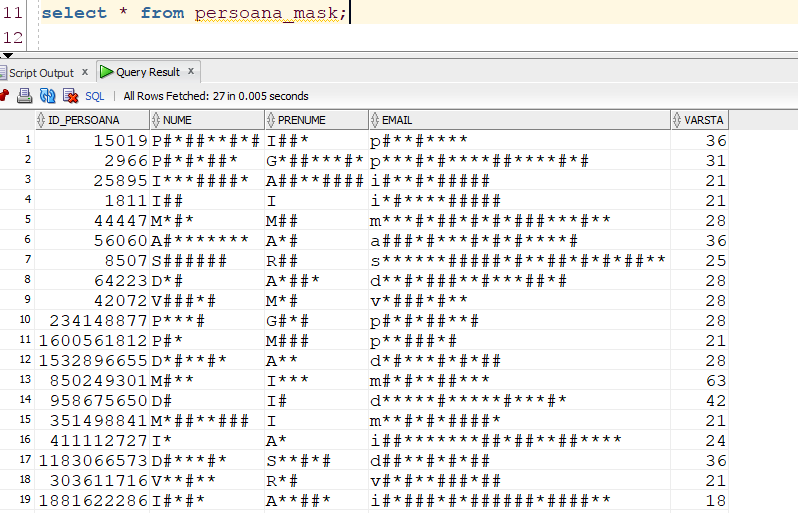


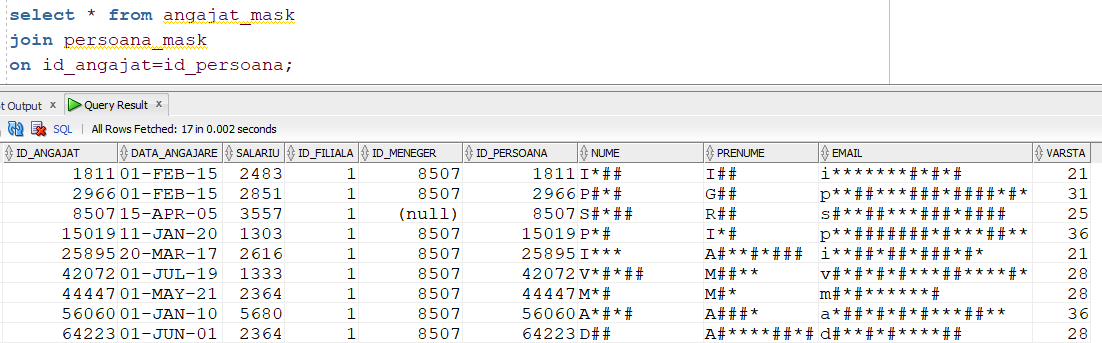
Întrucât am exportat doar tabelele **persoana**, **angajat**, **antrenor**, **receptionist** și **client**, nu și tabela **filiala**, și am păstrat la export constrângerile, la import vom avea o eroare care spune că nu se poate rezolva constrângerea de FK pentru tabela **filiala**. Pentru că suntem conștienți 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*.

1. **Codul SQL al aplicatiei** 
   1. **Admin**

**10.1.1 bro\_admin\_antrenor\_seed.sql**

-- Seed pentru schema bro\_antrenor rulat de catre bro\_admin

grant select,references on bro\_admin.antrenor to &&user\_name;

grant references,select on bro\_admin.echipament to &&user\_name;

grant references on bro\_admin.client to &&user\_name;

grant references,select on bro\_admin.chei\_client to &&user\_name;

grant execute on dbms\_crypto to &&user\_name;

create table &&user\_name..program (

id\_program number(\*,0)

generated by default on null as identity

constraint pk\_program primary key,

descriere varchar2(255),

tip\_program varchar2(20)

constraint ck\_program\_tip\_program not null

check ( tip\_program in ( 'CARDIO',

'MASS',

'RECOVERY' ) )

);

create table &&user\_name..antrenament (

id\_client number(\*,0),

id\_program number(\*,0),

id\_echipament number(\*,0),

durata number(3)

constraint nn\_antrenament\_durata not null,

constraint pk\_antrenament primary key ( id\_client,

id\_program,

id\_echipament ),

constraint fk\_antrenament\_client foreign key ( id\_client )

references bro\_admin.client ( id\_client ),

constraint fk\_antrenament\_program foreign key ( id\_program )

references &&user\_name..program ( id\_program ),

constraint fk\_antrenament\_echipament foreign key ( id\_echipament )

references bro\_admin.echipament ( id\_echipament )

);

create or replace function &&user\_name..cript\_string (

org varchar2,

mod\_op pls\_integer,

cheie raw

) return raw is

begin

return dbms\_crypto.encrypt(

utl\_i18n.string\_to\_raw(

org,

'AL32UTF8'

),

mod\_op,

cheie

);

end;

/

create or replace function &&user\_name..decript\_string (

cript raw,

mod\_op pls\_integer,

cheie raw

) return varchar2 is

result varchar2(4000);

begin

result := utl\_i18n.raw\_to\_char(

dbms\_crypto.decrypt(

cript,

mod\_op,

cheie

),

'AL32UTF8'

);

if result is null

or length(result) = 0 then

return 'Not allowed';

end if;

return result;

exception

when others then

return 'Not allowed';

end;

/

create or replace function &&user\_name..hash\_checksum (

input\_array sys.odcivarchar2list

) return raw is

concatenated\_string varchar2(4000);

begin

concatenated\_string := '';

for i in 1..input\_array.count loop

concatenated\_string := concatenated\_string || input\_array(i);

end loop;

return dbms\_crypto.hash(

utl\_i18n.string\_to\_raw(

concatenated\_string,

'AL32UTF8'

),

dbms\_crypto.hash\_md5

);

end;

/

create or replace view &&user\_name..client\_antrenament as

select cript\_string(

p.id\_program,

c.mod\_op,

c.cheie

) as id\_program,

cript\_string(

p.descriere,

c.mod\_op,

c.cheie

) as descriere\_program,

cript\_string(

p.tip\_program,

c.mod\_op,

c.cheie

) as tip\_program,

cript\_string(

a.durata,

c.mod\_op,

c.cheie

) as durata\_antrenament,

cript\_string(

e.id\_echipament,

c.mod\_op,

c.cheie

) as id\_echipament,

cript\_string(

e.nume,

c.mod\_op,

c.cheie

) as nume\_echipament,

cript\_string(

e.data\_instalare,

c.mod\_op,

c.cheie

) as data\_instalare\_echipament,

cript\_string(

e.data\_revizie,

c.mod\_op,

c.cheie

) as data\_revizie\_echipament,

cript\_string(

e.id\_filiala,

c.mod\_op,

c.cheie

) as id\_filiala,

cript\_string(

a.id\_client,

c.mod\_op,

c.cheie

) as id\_client,

hash\_checksum(sys.odcivarchar2list(

p.id\_program,

p.descriere,

p.tip\_program,

a.durata,

e.id\_echipament,

e.nume,

e.data\_instalare,

e.data\_revizie,

e.id\_filiala,

c.cheie

)) as checksum

from &&user\_name..program p

join &&user\_name..antrenament a

on p.id\_program = a.id\_program

join bro\_admin.echipament e

on a.id\_echipament = e.id\_echipament

join bro\_admin.chei\_client c

on c.id\_client = a.id\_client;

create or replace function &&user\_name..number\_to\_raw (

n number

) return raw is

begin

return hextoraw(to\_char(

n,

'FM0X'

));

end;

/

create or replace type &&user\_name..decrypted\_client\_record as object (

id\_program varchar2(100),

descriere\_program varchar2(2500),

tip\_program varchar2(100),

durata\_antrenament varchar2(50),

id\_echipament varchar2(100),

nume\_echipament varchar2(255),

data\_instalare\_echipament varchar2(50),

data\_revizie\_echipament varchar2(50),

id\_filiala varchar2(100),

id\_client varchar2(100),

checksum raw(16)

);

/

create or replace type &&user\_name..decrypted\_client\_table as

table of &&user\_name..decrypted\_client\_record;

/

create or replace function &&user\_name..fetch\_decrypted\_client\_data (

p\_mod\_op number,

p\_cheie raw,

p\_id\_client number

) return &&user\_name..decrypted\_client\_table

pipelined

is

begin

for r in (

select decript\_string(

ca.id\_program,

p\_mod\_op,

p\_cheie

) as id\_program,

decript\_string(

ca.descriere\_program,

p\_mod\_op,

p\_cheie

) as descriere\_program,

decript\_string(

ca.tip\_program,

p\_mod\_op,

p\_cheie

) as tip\_program,

decript\_string(

ca.durata\_antrenament,

p\_mod\_op,

p\_cheie

) as durata\_antrenament,

decript\_string(

ca.id\_echipament,

p\_mod\_op,

p\_cheie

) as id\_echipament,

decript\_string(

ca.nume\_echipament,

p\_mod\_op,

p\_cheie

) as nume\_echipament,

decript\_string(

ca.data\_instalare\_echipament,

p\_mod\_op,

p\_cheie

) as data\_instalare\_echipament,

decript\_string(

ca.data\_revizie\_echipament,

p\_mod\_op,

p\_cheie

) as data\_revizie\_echipament,

decript\_string(

ca.id\_filiala,

p\_mod\_op,

p\_cheie

) as id\_filiala,

decript\_string(

ca.id\_client,

p\_mod\_op,

p\_cheie

) as id\_client,

checksum

from &&user\_name..client\_antrenament ca

where regexp\_like ( decript\_string(

ca.id\_client,

p\_mod\_op,

p\_cheie

),

'^\d+$' )

) loop

pipe row ( decrypted\_client\_record(

r.id\_program,

r.descriere\_program,

r.tip\_program,

r.durata\_antrenament,

r.id\_echipament,

r.nume\_echipament,

r.data\_instalare\_echipament,

r.data\_revizie\_echipament,

r.id\_filiala,

r.id\_client,

r.checksum

) );

end loop;

return;

end;

/

commit;

exit;

**10.1.2. bro\_admin\_audit.sql**

create or replace type t\_echipament as object (

id\_echipament int,

nume varchar2(40),

data\_instalare date,

data\_revizie date,

id\_filiala int,

id\_furnizor int

);

/

create table audit\_echipament (

id\_audit int

generated by default as identity

primary key,

log\_time timestamp default systimestamp,

operation\_type varchar2(15),

performed\_by varchar2(128),

id\_echipament int,

old\_values t\_echipament,

new\_values t\_echipament,

summary\_message varchar2(2500)

);

create or replace trigger audit\_echipament\_trg for

insert or update or delete on echipament

compound trigger

type t\_row\_change is record (

operation\_type varchar2(15),

id\_echipament int,

old\_values t\_echipament,

new\_values t\_echipament

);

type t\_change\_table is

table of t\_row\_change index by pls\_integer;

g\_changes t\_change\_table;

g\_count\_op int := 0;

before each row is begin

if inserting then

g\_count\_op := g\_count\_op + 1;

g\_changes(g\_changes.count + 1) := t\_row\_change(

'INSERT',

:new.id\_echipament,

null,

t\_echipament(

:new.id\_echipament,

:new.nume,

:new.data\_instalare,

:new.data\_revizie,

:new.id\_filiala,

:new.id\_furnizor

)

);

elsif updating then

g\_count\_op := g\_count\_op + 1;

g\_changes(g\_changes.count + 1) := t\_row\_change(

'UPDATE',

:new.id\_echipament,

t\_echipament(

:old.id\_echipament,

:old.nume,

:old.data\_instalare,

:old.data\_revizie,

:old.id\_filiala,

:old.id\_furnizor

),

t\_echipament(

:new.id\_echipament,

:new.nume,

:new.data\_instalare,

:new.data\_revizie,

:new.id\_filiala,

:new.id\_furnizor

)

);

elsif deleting then

g\_count\_op := g\_count\_op + 1;

g\_changes(g\_changes.count + 1) := t\_row\_change(

'DELETE',

:old.id\_echipament,

t\_echipament(

:old.id\_echipament,

:old.nume,

:old.data\_instalare,

:old.data\_revizie,

:old.id\_filiala,

:old.id\_furnizor

),

null

);

end if;

end before each row;

after statement is begin

for i in 1..g\_changes.count loop

insert into audit\_echipament (

operation\_type,

performed\_by,

id\_echipament,

old\_values,

new\_values,

summary\_message

) values ( g\_changes(i).operation\_type,

user,

g\_changes(i).id\_echipament,

g\_changes(i).old\_values,

g\_changes(i).new\_values,

upper(g\_changes(i).operation\_type)

|| ' with another '

|| g\_count\_op );

end loop;

end after statement;

end;

/

select \* from audit\_echipament;

select a.old\_values.data\_revizie, a.new\_values.data\_revizie from audit\_echipament a;

grant select,update on audit\_echipament to bro\_manager\_filiala1;

select count(\*)

from audit\_echipament a

where a.old\_values.id\_filiala!=1 and a.new\_values.id\_filiala!=1;

**10.1.3. bro\_admin\_create\_tables.sql**

SET SERVEROUTPUT ON;

-- Crearea tabelelor si inserarea datelor initiale in schema bro\_admin

CREATE TABLE persoana(

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

nume VARCHAR2(20) CONSTRAINT nn\_persoana\_nume NOT NULL,

prenume VARCHAR2(30) CONSTRAINT nn\_persoana\_prenume NOT NULL,

email VARCHAR2(30) CONSTRAINT nn\_u\_persoana\_email NOT NULL UNIQUE,

varsta NUMBER(3,0) CONSTRAINT nn\_persoana\_varsta NOT NULL

);

CREATE TABLE telefon(

tip VARCHAR2(20) CONSTRAINT nn\_telefon\_tip NOT NULL ,

numar VARCHAR2(20) CONSTRAINT nn\_telefon\_numar NOT NULL,

id\_persoana NUMBER(\*,0) CONSTRAINT fk\_telefon\_persoana REFERENCES persoana(id\_persoana) ON DELETE CASCADE,

CONSTRAINT pk\_telefon PRIMARY KEY (id\_persoana, numar)

);

CREATE TABLE client(

id\_client NUMBER(\*,0) CONSTRAINT pk\_client PRIMARY KEY ,

student VARCHAR2(1) DEFAULT 'N' CONSTRAINT ck\_client\_student CHECK (student IN('Y','N')) ,

CONSTRAINT fk\_client\_persoana FOREIGN KEY (id\_client) REFERENCES persoana(id\_persoana) ON DELETE CASCADE

);

CREATE TABLE adresa(

id\_adresa NUMBER(\*,0) CONSTRAINT pk\_adresa PRIMARY KEY,

strada VARCHAR2(40) CONSTRAINT nn\_adresa\_strada NOT NULL,

oras VARCHAR2(20) CONSTRAINT nn\_adresa\_oras NOT NULL,

judet VARCHAR2(20) CONSTRAINT nn\_adresa\_judet NOT NULL,

cod\_postal NUMBER(10,0) CONSTRAINT nn\_adresa\_cod\_postal NOT NULL,

numar NUMBER(4,0) CONSTRAINT nn\_adresa\_numar NOT NULL

);

CREATE TABLE filiala (

id\_filiala NUMBER(\*,0) CONSTRAINT pk\_filiala PRIMARY KEY,

nume VARCHAR2(40) CONSTRAINT nn\_filiala\_nume NOT NULL,

data\_deschidere DATE CONSTRAINT nn\_filiala\_data\_deschidere NOT NULL,

id\_adresa NUMBER(\*,0) CONSTRAINT fk\_filiala\_adresa REFERENCES adresa(id\_adresa) NOT NULL UNIQUE

);

CREATE TABLE angajat(

id\_angajat NUMBER(\*,0) CONSTRAINT pk\_angajat PRIMARY KEY,

data\_angajare DATE CONSTRAINT nn\_angajat\_data\_angjare NOT NULL,

salariu NUMBER(20,2) CONSTRAINT ck\_angajat\_salariu CHECK (salariu > 0) NOT NULL,

id\_filiala NUMBER(\*,0) CONSTRAINT fk\_angajat\_filiala REFERENCES filiala(id\_filiala) ON DELETE CASCADE NOT NULL ,

id\_meneger NUMBER(\*,0) CONSTRAINT fk\_angajat\_angajat REFERENCES angajat(id\_angajat),

CONSTRAINT fk\_angajat\_persoana FOREIGN KEY (id\_angajat) REFERENCES persoana(id\_persoana) ON DELETE CASCADE

);

CREATE TABLE receptionist(

id\_receptionist NUMBER(\*,0) CONSTRAINT pk\_receptionist PRIMARY KEY,

program\_complet VARCHAR2(1) CONSTRAINT ck\_receptionist\_program\_complet CHECK(program\_complet IN ('Y','N')),

CONSTRAINT fk\_receptionist\_angajat FOREIGN KEY (id\_receptionist) REFERENCES angajat(id\_angajat) ON DELETE CASCADE

);

CREATE TABLE antrenor(

id\_antrenor NUMBER(\*,0) CONSTRAINT pk\_antrenor PRIMARY KEY,

studii VARCHAR2(40) CONSTRAINT nn\_antrenor\_studii NOT NULL,

CONSTRAINT fk\_antrenor\_angajat FOREIGN KEY (id\_antrenor) REFERENCES angajat(id\_angajat) ON DELETE CASCADE

);

CREATE TABLE furnizor(

id\_furnizor NUMBER(\*,0) CONSTRAINT pk\_furnizor PRIMARY KEY,

nume VARCHAR2(40) CONSTRAINT nn\_furnizor\_nume NOT NULL,

cod\_fiscal NUMBER(10,0) CONSTRAINT ck\_furnizor\_cod\_fiscal NOT NULL UNIQUE,

id\_adresa NUMBER(\*,0) CONSTRAINT fk\_furnizor\_adresa REFERENCES adresa(id\_adresa) NOT NULL UNIQUE

);

CREATE TABLE echipament(

id\_echipament NUMBER(\*,0) CONSTRAINT pk\_echipament PRIMARY KEY,

nume VARCHAR2(40) CONSTRAINT nn\_echipament\_nume NOT NULL,

data\_instalare DATE CONSTRAINT nn\_echipament\_data\_instalare NOT NULL,

data\_revizie DATE CONSTRAINT nn\_echipament\_data\_revizie NOT NULL ,

id\_filiala NUMBER(\*,0) CONSTRAINT fk\_echipament\_filiala REFERENCES filiala(id\_filiala) NOT NULL,

id\_furnizor NUMBER(\*,0) CONSTRAINT fk\_echipament\_furnizor REFERENCES furnizor(id\_furnizor) NOT NULL,

CONSTRAINT ck\_echipament\_instalare\_revizie CHECK(data\_instalare <= data\_revizie)

);

CREATE TABLE tip\_abonament (

nume\_tip VARCHAR2(40) CONSTRAINT pk\_tip\_abonament PRIMARY KEY CHECK (nume\_tip IN ( 'lunar', 'trimestrial', 'bianual' ,'anual','extins')),

pret NUMBER(8,2) CONSTRAINT ck\_tip\_abonament\_pret NOT NULL UNIQUE

);

CREATE TABLE abonament(

id\_abonament NUMBER(\*,0) CONSTRAINT pk\_abonament PRIMARY KEY,

nume\_tip VARCHAR2(40) CONSTRAINT fk\_abonament\_tip\_abonament REFERENCES tip\_abonament(nume\_tip) NOT NULL,

id\_client NUMBER(\*,0) CONSTRAINT fk\_abonament\_client REFERENCES client(id\_client) NOT NULL UNIQUE,

data\_inregistrare DATE CONSTRAINT nn\_abonament\_data\_intregistrare NOT NULL

);

CREATE TABLE comanda(

id\_comanda NUMBER(\*,0) CONSTRAINT pk\_comanda PRIMARY KEY,

id\_receptionist NUMBER(\*,0) CONSTRAINT fk\_comanda\_receptionist REFERENCES receptionist(id\_receptionist) NOT NULL,

id\_client NUMBER(\*,0) CONSTRAINT fk\_comanda\_client REFERENCES client(id\_client) NOT NULL,

data\_comandare DATE CONSTRAINT nn\_comanda\_data\_comandare NOT NULL,

observatii VARCHAR2(255)

);

CREATE TABLE supliment (

id\_supliment NUMBER(\*,0) CONSTRAINT pk\_supliment PRIMARY KEY,

nume VARCHAR2(50) CONSTRAINT nn\_supliment\_nume NOT NULL,

descriere VARCHAR2(255),

calorii NUMBER(10,4)CONSTRAINT nn\_suplimen\_calorii NOT NULL,

pret NUMBER(10,4) CONSTRAINT nn\_supliment\_pret NOT NULL

);

CREATE TABLE aprovizionare (

id\_furnizor NUMBER(\*,0),

id\_supliment NUMBER(\*,0),

cantitate NUMBER(4) CONSTRAINT ck\_aprovizionare\_cantitate CHECK(cantitate > 0) NOT NULL,

CONSTRAINT pk\_aprovizionare PRIMARY KEY (id\_furnizor,id\_supliment),

CONSTRAINT fk\_aprovizionare\_furnizor FOREIGN KEY (id\_furnizor) REFERENCES furnizor(id\_furnizor),

CONSTRAINT fk\_aprovizionare\_supliment FOREIGN KEY (id\_supliment) REFERENCES supliment(id\_supliment)

);

CREATE TABLE informatii\_comanda (

id\_comanda NUMBER(\*,0),

id\_supliment NUMBER(\*,0),

cantitate NUMBER(4) CONSTRAINT ck\_ic\_cantitate CHECK(cantitate > 0) NOT NULL,

CONSTRAINT pk\_informatii\_comanda PRIMARY KEY (id\_comanda,id\_supliment),

CONSTRAINT fk\_ic\_comanda FOREIGN KEY (id\_comanda) REFERENCES comanda(id\_comanda),

CONSTRAINT fk\_ic\_supliment FOREIGN KEY (id\_supliment) REFERENCES supliment(id\_supliment)

);

CREATE TABLE account\_mapping(

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

username VARCHAR2(128) UNIQUE

);

COMMIT;

CREATE TABLE logger(

id\_logger NUMBER(\*,0) CONSTRAINT pk\_logger PRIMARY KEY,

message VARCHAR2(255),

message\_type VARCHAR2(1)CONSTRAINT ck\_logger\_message\_type CHECK (message\_type IN('E','W','I')),

created\_by VARCHAR2(40)CONSTRAINT nn\_logger\_created\_by NOT NULL,

created\_at TIMESTAMP CONSTRAINT nn\_logger\_created\_at NOT NULL

);

CREATE OR REPLACE PACKAGE logger\_utils IS

PROCEDURE logger\_entry(mesaj VARCHAR2,tip\_mesaj VARCHAR2, cod NUMBER);

PROCEDURE logger\_entry(mesaj VARCHAR2,tip\_mesaj VARCHAR2);

END logger\_utils;

/

-- PRAGMA AUTONOMOUS\_TRANSACTION este necesara, deoarece functia

-- RAISE\_APPLICATION\_ERROR opreste tranzactia originala, ceea ce impiedica

-- inserarea in Logger. In acest caz folosirea acesteia nu conduce

-- la probleme pentru ca nu folosim date

-- din noua tranzactie in cea originala.

CREATE OR REPLACE PACKAGE BODY logger\_utils IS

PROCEDURE logger\_entry(mesaj VARCHAR2,tip\_mesaj VARCHAR2, cod NUMBER) IS

PRAGMA autonomous\_transaction;

BEGIN

INSERT INTO logger(message, message\_type,created\_by, created\_at)

VALUES (substr(mesaj,1,255), tip\_mesaj,user, TO\_DATE(to\_char(sysdate, 'DD-MON-YYYY HH24:MI:SS'), 'DD-MON-YYYY HH24:MI:SS'));

COMMIT;

raise\_application\_error(cod,mesaj);

dbms\_output.put\_line(cod || ' : '||mesaj);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

raise\_application\_error(sqlcode,sqlerrm);

END logger\_entry;

PROCEDURE logger\_entry(mesaj VARCHAR2,tip\_mesaj VARCHAR2) IS

PRAGMA autonomous\_transaction;

BEGIN

dbms\_output.put\_line(tip\_mesaj || ' : '||mesaj);

INSERT INTO logger(message, message\_type,created\_by, created\_at) VALUES

(substr(mesaj,1,255), tip\_mesaj,user, TO\_DATE(to\_char(sysdate, 'DD-MON-YYYY HH24:MI:SS'), 'DD-MON-YYYY HH24:MI:SS'));

COMMIT;

END;

END logger\_utils;

/

CREATE OR REPLACE PACKAGE sequence\_utils IS

PROCEDURE create\_sequence(p\_seq\_name IN VARCHAR2);

PROCEDURE create\_sequence\_trigger (p\_tbl\_name IN VARCHAR2);

END sequence\_utils;

/

CREATE OR REPLACE PACKAGE BODY sequence\_utils IS

PROCEDURE create\_sequence(p\_seq\_name IN VARCHAR2) IS

seq\_count INT;

seq\_name VARCHAR2(128);

BEGIN

-- dbms\_output.put\_line(p\_seq\_name);

seq\_name:=dbms\_assert.simple\_sql\_name(p\_seq\_name);

-- dbms\_output.put\_line(seq\_name);

SELECT COUNT(\*) INTO seq\_count FROM user\_sequences WHERE sequence\_name = upper(seq\_name);

IF seq\_count > 0 THEN

EXECUTE IMMEDIATE 'DROP SEQUENCE '|| seq\_name;

END IF;

EXECUTE IMMEDIATE 'CREATE SEQUENCE ' || seq\_name || ' START WITH 1 INCREMENT BY 1';

EXCEPTION

WHEN OTHERS THEN

logger\_utils.logger\_entry(sqlerrm,'E',sqlcode);

END create\_sequence;

PROCEDURE create\_sequence\_trigger (p\_tbl\_name IN VARCHAR2) IS

count\_tables NUMBER;

v\_id\_count INT;

no\_id EXCEPTION;

table\_not\_found EXCEPTION;

tbl\_name VARCHAR2(128);

BEGIN

dbms\_output.put\_line(p\_tbl\_name);

tbl\_name:=dbms\_assert.simple\_sql\_name(p\_tbl\_name);

dbms\_output.put\_line(tbl\_name);

SELECT COUNT(\*)

INTO count\_tables

FROM all\_tables

WHERE table\_name = upper(tbl\_name);

IF count\_tables = 0 THEN

RAISE table\_not\_found;

END IF;

EXECUTE IMMEDIATE

'SELECT COUNT(\*) FROM all\_tab\_columns WHERE upper(table\_name) = upper(''' || tbl\_name ||

''') AND upper(column\_name) = upper(''id\_' || tbl\_name || ''')' INTO v\_id\_count;

IF v\_id\_count=0 THEN

RAISE no\_id;

END IF;

create\_sequence(tbl\_name ||'\_seq');

EXECUTE IMMEDIATE 'CREATE OR REPLACE TRIGGER ' || tbl\_name ||

'\_trigger BEFORE INSERT ON ' || tbl\_name ||

' FOR EACH ROW BEGIN SELECT ' ||tbl\_name||'\_seq.NEXTVAL INTO :NEW.id\_'||

lower(tbl\_name)||' FROM dual; END;';

EXCEPTION

WHEN no\_id THEN

logger\_utils.logger\_entry('Column named id\_'|| tbl\_name || ' does not exist in '||tbl\_name,'E',-20006);

WHEN table\_not\_found THEN

logger\_utils.logger\_entry('Table '|| tbl\_name || ' does not exist.','E',-20007);

WHEN OTHERS THEN

logger\_utils.logger\_entry( sqlerrm || ' code: ' || sqlcode,'E',-20010);

END create\_sequence\_trigger;

END sequence\_utils;

/

TRUNCATE TABLE logger;

EXEC sequence\_utils.create\_sequence\_trigger('Logger');

CREATE OR REPLACE PROCEDURE insert\_into\_account\_mapping(

id\_persoana NUMBER,acc\_suff VARCHAR2

) IS

v\_user VARCHAR2(128);

BEGIN

SELECT column\_value INTO v\_user FROM (

SELECT column\_value,TO\_NUMBER(regexp\_substr(column\_value, '[0-9]+$')) AS numeric\_part

FROM TABLE(sys.get\_users\_by\_suffix(acc\_suff))

WHERE NOT EXISTS(SELECT 1 FROM account\_mapping WHERE username=upper(column\_value))

ORDER BY numeric\_part ASC)

WHERE ROWNUM =1 ;

dbms\_output.put\_line('user '||v\_user);

INSERT INTO account\_mapping VALUES (id\_persoana,upper(v\_user));

EXCEPTION

WHEN OTHERS THEN

logger\_utils.logger\_entry( 'No avaialable account for the suffix '|| acc\_suff,'E',-20020);

END;

/

CREATE OR REPLACE PACKAGE global\_constants IS

persoana\_seq CONSTANT VARCHAR2(20) := 'PERSOANA\_SEQ\_GLOBAL';

END global\_constants;

/

COMMIT;

-- multiple vizualizari si triggere de tipul instead of pentru a usura inserarea

CREATE OR REPLACE VIEW client\_extins AS(

SELECT c.id\_client, p.nume, p.prenume,p.email,p.varsta, c.student

FROM persoana p JOIN client c ON c.id\_client = p.id\_persoana

);

CREATE OR REPLACE TRIGGER client\_extins\_insert INSTEAD OF INSERT ON client\_extins

FOR EACH ROW

DECLARE

seq\_count NUMBER;

seq\_not\_found EXCEPTION;

id\_nr persoana.id\_persoana%TYPE;

BEGIN

SELECT COUNT(\*)

INTO seq\_count

FROM user\_sequences

WHERE sequence\_name = global\_constants.persoana\_seq;

IF seq\_count = 0 THEN

RAISE seq\_not\_found;

END IF;

EXECUTE IMMEDIATE 'SELECT ' || global\_constants.persoana\_seq || '.NEXTVAL FROM dual' INTO id\_nr;

INSERT INTO persoana(id\_persoana,nume,prenume,email,varsta) VALUES

(id\_nr, :new.nume, :new.prenume, :new.email, :new.varsta);

INSERT INTO client VALUES

(id\_nr,:new.student);

insert\_into\_account\_mapping(id\_nr,'CLIENT');

EXCEPTION

WHEN seq\_not\_found THEN

logger\_utils.logger\_entry('Secventa pentru persoana nu exista.','E',-20005);

WHEN OTHERS THEN

logger\_utils.logger\_entry( sqlerrm || ' code: ' || sqlcode,'E',-20010);

END;

/

CREATE OR REPLACE VIEW angajat\_extins AS (

SELECT a.id\_angajat,p.nume, p.prenume,p.email,p.varsta,

a.data\_angajare, a.salariu, a.id\_filiala, a.id\_meneger

FROM persoana p JOIN angajat a ON p.id\_persoana = a.id\_angajat

);

CREATE OR REPLACE TRIGGER angajat\_extins\_insert INSTEAD OF INSERT ON angajat\_extins

FOR EACH ROW

BEGIN

INSERT INTO persoana(id\_persoana,nume,prenume,email,varsta) VALUES

(:new.id\_angajat, :new.nume, :new.prenume, :new.email, :new.varsta);

INSERT INTO angajat(id\_angajat, data\_angajare, salariu, id\_filiala, id\_meneger) VALUES

(:new.id\_angajat,:new.data\_angajare, :new.salariu, :new.id\_filiala, :new.id\_meneger);

EXCEPTION

WHEN OTHERS THEN

logger\_utils.logger\_entry( sqlerrm || ' code: ' || sqlcode,'E',-20010);

END;

/

CREATE OR REPLACE VIEW antrenor\_extins AS (

SELECT ant.id\_antrenor,a.nume, a.prenume,a.email,a.varsta,

a.data\_angajare, a.salariu, a.id\_filiala, a.id\_meneger,ant.studii

FROM antrenor ant JOIN angajat\_extins a ON ant.id\_antrenor = a.id\_angajat

);

CREATE OR REPLACE TRIGGER antrenor\_extins\_insert INSTEAD OF INSERT ON antrenor\_extins

FOR EACH ROW

DECLARE

seq\_count NUMBER;

seq\_not\_found EXCEPTION;

id\_nr persoana.id\_persoana%TYPE;

id\_men persoana.id\_persoana%TYPE := NULL;

BEGIN

SELECT COUNT(\*)

INTO seq\_count

FROM user\_sequences

WHERE sequence\_name = global\_constants.persoana\_seq;

IF seq\_count = 0 THEN

RAISE seq\_not\_found;

END IF;

EXECUTE IMMEDIATE 'SELECT ' || global\_constants.persoana\_seq || '.NEXTVAL FROM dual' INTO id\_nr;

IF :new.id\_meneger IS NOT NULL THEN

id\_men:=:new.id\_meneger;

END IF;

INSERT INTO angajat\_extins(id\_angajat,nume, prenume,email,varsta,

data\_angajare, salariu, id\_filiala, id\_meneger) VALUES

(id\_nr, :new.nume, :new.prenume,:new.email,:new.varsta,

:new.data\_angajare, :new.salariu,:new.id\_filiala, id\_men);

INSERT INTO antrenor VALUES

(id\_nr,:new.studii);

insert\_into\_account\_mapping(id\_nr,'ANTRENOR');

EXCEPTION

WHEN seq\_not\_found THEN

logger\_utils.logger\_entry('Secventa pentru persoana nu exista.','E',-20005);

WHEN OTHERS THEN

dbms\_output.put\_line(sqlerrm);

logger\_utils.logger\_entry( sqlerrm || ' code: ' || sqlcode,'E',-20010);

END;

/

CREATE OR REPLACE VIEW receptionist\_extins AS (

SELECT r.id\_receptionist,a.nume, a.prenume,a.email,a.varsta,

a.data\_angajare, a.salariu, a.id\_filiala, a.id\_meneger,r.program\_complet

FROM receptionist r JOIN angajat\_extins a ON r.id\_receptionist = a.id\_angajat

);

/

CREATE OR REPLACE TRIGGER receptionist\_extins\_insert INSTEAD OF INSERT ON receptionist\_extins

FOR EACH ROW

DECLARE

seq\_count NUMBER;

seq\_not\_found EXCEPTION;

id\_nr persoana.id\_persoana%TYPE;

id\_men persoana.id\_persoana%TYPE := NULL;

BEGIN

SELECT COUNT(\*)

INTO seq\_count

FROM user\_sequences

WHERE sequence\_name = global\_constants.persoana\_seq;

IF seq\_count = 0 THEN

RAISE seq\_not\_found;

END IF;

EXECUTE IMMEDIATE 'SELECT ' || global\_constants.persoana\_seq || '.NEXTVAL FROM dual' INTO id\_nr;

IF :new.id\_meneger IS NOT NULL THEN

id\_men:=:new.id\_meneger;

END IF;

INSERT INTO angajat\_extins(id\_angajat,nume, prenume,email,varsta,

data\_angajare, salariu, id\_filiala, id\_meneger) VALUES

(id\_nr, :new.nume, :new.prenume,:new.email,:new.varsta,

:new.data\_angajare, :new.salariu,:new.id\_filiala, id\_men);

INSERT INTO receptionist(id\_receptionist,program\_complet) VALUES

(id\_nr,:new.program\_complet);

insert\_into\_account\_mapping(id\_nr,'RECEPTIONIST');

EXCEPTION

WHEN seq\_not\_found THEN

logger\_utils.logger\_entry('Secventa pentru persoana nu exista.','E',-20005);

WHEN OTHERS THEN

logger\_utils.logger\_entry( sqlerrm || ' code: ' || sqlcode,'E',-20010);

END;

/

EXEC sequence\_utils.create\_sequence\_trigger('Adresa');

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Bd. Lujerului',

33,

'Bucuresti',

'Bucuresti',

'405985' );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Bd. Tineretului',

21,

'Bucuresti',

'Bucuresti',

'582155' );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Bd. Bucuresti',

11,

'Brasov',

'Brasov',

'123456' );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Bd. Republicii',

3,

'Ploiesti',

'Prahova',

55231 );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Str Parangului',

100,

'Craiova',

'Dolj',

7742101 );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Matei Basarab',

18,

'Bucuresti',

'Bucuresti',

665842 );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Unirii',

33,

'Bucuresti',

'Bucuresti',

868605 );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Mihai Bravu',

22,

'Bucuresti',

'Bucuresti',

78592 );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Frigului',

77,

'Brasov',

'Brasov',

888801 );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Calea Traian',

99,

'Craiova',

'Dolj',

224402 );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Calea Serban Voda',

232,

'Bucuresti',

'Bucuresti',

40578 );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Viilor',

12,

'Bucuresti',

'Bucuresti',

232454 );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Alea Tomis',

36,

'Arad',

'Arad',

111454 );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Anastasie Panu',

56,

'Iasi',

'Iasi',

999454 );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Aleea Tomis',

1,

'Dej',

'Cluj',

123454 );

INSERT INTO adresa (

strada,

numar,

oras,

judet,

cod\_postal

) VALUES ( 'Tiberiu Popoviciu ',

22,

'Cluj',

'Cluj',

538454 );

EXEC sequence\_utils.create\_sequence\_trigger('Filiala');

INSERT INTO filiala (

nume,

data\_deschidere,

id\_adresa

) VALUES ( 'Lujerului',

TO\_DATE('21-JAN-2014','DD-MON-YYYY'),

1 );

INSERT INTO filiala (

nume,

data\_deschidere,

id\_adresa

) VALUES ( 'Tineretului',

TO\_DATE('21-FEB-2000','DD-MON-YYYY'),

2 );

INSERT INTO filiala (

nume,

data\_deschidere,

id\_adresa

) VALUES ( 'Brasov',

TO\_DATE('14-FEB-2010','DD-MON-YYYY'),

3 );

INSERT INTO filiala (

nume,

data\_deschidere,

id\_adresa

) VALUES ( 'Ploiesti',

TO\_DATE('11-DEC-1999','DD-MON-YYYY'),

4 );

INSERT INTO filiala (

nume,

data\_deschidere,

id\_adresa

) VALUES ( 'Craiova',

TO\_DATE('01-NOV-1995','DD-MON-YYYY'),

5 );

INSERT INTO filiala (

nume,

data\_deschidere,

id\_adresa

) VALUES ( 'Filiala Sector 4',

TO\_DATE('01-FEB-1999','DD-MON-YYYY'),

11 );

INSERT INTO filiala (

nume,

data\_deschidere,

id\_adresa

) VALUES ( 'Filiala Sector 3',

TO\_DATE('15-MAR-2005','DD-MON-YYYY'),

6 );

INSERT INTO filiala (

nume,

data\_deschidere,

id\_adresa

) VALUES ( 'Sediul Unirii',

TO\_DATE('01-MAY-2000','DD-MON-YYYY'),

7 );

INSERT INTO filiala (

nume,

data\_deschidere,

id\_adresa

) VALUES ( 'Filiala Viilor',

TO\_DATE('01-APR-2012','DD-MON-YYYY'),

12 );

EXEC sequence\_utils.create\_sequence(global\_constants.persoana\_seq);

SELECT \*

FROM account\_mapping;

INSERT INTO antrenor\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

studii,

id\_meneger

) VALUES ( 'Popescu',

'Ion',

'popescuI@yahoo.com',

30,

TO\_DATE('11-JAN-2020','DD-MON-YYYY'),

1500,

1,

'Liceul Sportiv 1 Bucuresti',

NULL );

INSERT INTO antrenor\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

studii,

id\_meneger

) VALUES ( 'Popescu',

'George',

'popescuG@yahoo.com',

31,

TO\_DATE('01-FEB-2015','DD-MON-YYYY'),

2100,

1,

'Liceul Sportiv Breaza',

NULL );

INSERT INTO antrenor\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

studii,

id\_meneger

) VALUES ( 'Ionescu',

'Andrei',

'ionescuA@yahoo.com',

21,

TO\_DATE('20-MAR-2017','DD-MON-YYYY'),

2200,

1,

'Facultate Kinetoterapie',

NULL );

INSERT INTO antrenor\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

studii,

id\_meneger

) VALUES ( 'Ionescu',

'Ion',

'ionescuI@yahoo.com',

21,

TO\_DATE('01-FEB-2015','DD-MON-YYYY'),

2000,

1,

'IEFS',

NULL );

INSERT INTO antrenor\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

studii,

id\_meneger

) VALUES ( 'Mihai',

'Marcel',

'mihaimarcel@yahoo.com',

22,

TO\_DATE('01-MAY-2021','DD-MON-YYYY'),

2600,

1,

'Facultate Kinetoterapie',

NULL );

INSERT INTO antrenor\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

studii,

id\_meneger

) VALUES ( 'Aioanei',

'Andrei',

'aioaneiandrei@yahoo.com',

30,

TO\_DATE('01-JAN-2010','DD-MON-YYYY'),

5000,

1,

'IEFS',

NULL );

INSERT INTO antrenor\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

studii,

id\_meneger

) VALUES ( 'Stancioiu',

'Razvan',

'stancioiurazvan@yahoo.com',

28,

TO\_DATE('15-APR-2005','DD-MON-YYYY'),

3500,

1,

'Curs FRCF',

NULL );

SELECT \*

FROM antrenor\_extins;

INSERT INTO receptionist\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

program\_complet,

id\_meneger

) VALUES ( 'Dinca',

'Antoaneta',

'dincaa@yahoo.com',

22,

TO\_DATE('01-JUN-2001','DD-MON-YYYY'),

2600,

1,

'Y',

NULL );

INSERT INTO receptionist\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

program\_complet,

id\_meneger

) VALUES ( 'Vasilescu',

'Marcel',

'vasilescum@yahoo.com',

22,

TO\_DATE('01-JUL-2019','DD-MON-YYYY'),

1300,

1,

'N',

NULL );

INSERT INTO receptionist\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

program\_complet,

id\_meneger

) VALUES ( 'Popescu',

'George',

'popescug@yahoo.com',

22,

TO\_DATE('01-JAN-2018','DD-MON-YYYY'),

2300,

1,

'Y',

NULL );

INSERT INTO receptionist\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

program\_complet,

id\_meneger

) VALUES ( 'Preda',

'Marina',

'predam@yahoo.com',

27,

TO\_DATE('01-FEB-2017','DD-MON-YYYY'),

2500,

1,

'Y',

NULL );

INSERT INTO receptionist\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

program\_complet,

id\_meneger

) VALUES ( 'Dumitrescu',

'Anca',

'dumitrescua@yahoo.com',

22,

TO\_DATE('01-MAR-2015','DD-MON-YYYY'),

2750,

1,

'Y',

NULL );

INSERT INTO receptionist\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

program\_complet,

id\_meneger

) VALUES ( 'Marinica',

'Ion',

'marinicaion@yahoo.com',

60,

TO\_DATE('01-JUN-2016','DD-MON-YYYY'),

1700,

1,

'N',

NULL );

INSERT INTO receptionist\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

program\_complet,

id\_meneger

) VALUES ( 'Dinca',

'Ion',

'dincaion@yahoo.com',

45,

TO\_DATE('01-APR-2021','DD-MON-YYYY'),

1700,

9,

'N',

NULL );

INSERT INTO receptionist\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

program\_complet,

id\_meneger

) VALUES ( 'Marinescu',

'Ion',

'marinescuion@yahoo.com',

23,

TO\_DATE('01-JUN-2022','DD-MON-YYYY'),

2900,

9,

'Y',

NULL );

INSERT INTO receptionist\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

program\_complet,

id\_meneger

) VALUES ( 'Ignat',

'Ana',

'ignatana@yahoo.com',

20,

TO\_DATE('01-FEB-2023','DD-MON-YYYY'),

2600,

5,

'Y',

NULL );

INSERT INTO receptionist\_extins (

nume,

prenume,

email,

varsta,

data\_angajare,

salariu,

id\_filiala,

program\_complet,

id\_meneger

) VALUES ( 'Dancescu',

'Sorin',

'dancescusorin@yahoo.com',

35,

TO\_DATE('01-FEB-2020','DD-MON-YYYY'),

3000,

4,

'Y',

NULL );

SELECT \*

FROM account\_mapping;

UPDATE angajat

SET

id\_meneger = NULL

WHERE id\_angajat = 7;

UPDATE angajat

SET

id\_meneger = 7

WHERE id\_angajat != 7

AND id\_filiala = 1;

UPDATE angajat

SET

id\_meneger = 14

WHERE id\_angajat = 15;

SELECT \*

FROM angajat;

INSERT INTO client\_extins (

nume,

prenume,

email,

varsta,

student

) VALUES ( 'Vasilescu',

'Razvan',

'vasilescurazvan@yahoo.com',

21,

'N' );

INSERT INTO client\_extins (

nume,

prenume,

email,

varsta,

student

) VALUES ( 'Ionescu',

'Andrei',

'ionescua@yahoo.com',

19,

'Y' );

INSERT INTO client\_extins (

nume,

prenume,

email,

varsta,

student

) VALUES ( 'Tanasescu',

'Ion',

'tanasescui@yahoo.com',

19,

'Y' );

INSERT INTO client\_extins (

nume,

prenume,

email,

varsta,

student

) VALUES ( 'Ionescu',

'Vasile',

'ionescuv@yahoo.com',

32,

'N' );

INSERT INTO client\_extins (

nume,

prenume,

email,

varsta,

student

) VALUES ( 'Tanasescu',

'Anca',

'tanasescua@yahoo.com',

50,

'N' );

INSERT INTO client\_extins (

nume,

prenume,

email,

varsta,

student

) VALUES ( 'Marinecu',

'Vlad',

'vladutz@yahoo.com',

27,

'N' );

INSERT INTO client\_extins (

nume,

prenume,

email,

varsta,

student

) VALUES ( 'Dobrescu',

'Marcel',

'dorescu\_mar@yahoo.com',

37,

'N' );

INSERT INTO client\_extins (

nume,

prenume,

email,

varsta,

student

) VALUES ( 'Marinica',

'Stefan',

'marinicastefan@yahoo.com',

35,

'N' );

INSERT INTO client\_extins (

nume,

prenume,

email,

varsta,

student

) VALUES ( 'Marinica',

'Bogdan',

'marinicabogdan@yahoo.com',

22,

'Y' );

INSERT INTO client\_extins (

nume,

prenume,

email,

varsta,

student

) VALUES ( 'Stefanescu',

'Ana',

'stefanescuana@yahoo.com',

19,

'Y' );

SELECT \*

FROM account\_mapping;

EXEC sequence\_utils.create\_sequence\_trigger('Furnizor');

INSERT INTO furnizor (

nume,

cod\_fiscal,

id\_adresa

) VALUES ( 'MyProtein',

'8859692',

1 );

INSERT INTO furnizor (

nume,

cod\_fiscal,

id\_adresa

) VALUES ( 'Gym Beam',

'9859692',

2 );

INSERT INTO furnizor (

nume,

cod\_fiscal,

id\_adresa

) VALUES ( 'Redis',

'7859692',

3 );

INSERT INTO furnizor (

nume,

cod\_fiscal,

id\_adresa

) VALUES ( 'Decathlon',

'1859692',

4 );

INSERT INTO furnizor (

nume,

cod\_fiscal,

id\_adresa

) VALUES ( 'Vexio',

'9959692',

5 );

INSERT INTO furnizor (

nume,

cod\_fiscal,

id\_adresa

) VALUES ( 'BEWIT',

'9059692',

13 );

INSERT INTO furnizor (

nume,

cod\_fiscal,

id\_adresa

) VALUES ( 'BODY NEWLINE CONCEPT',

'48393052',

14 );

INSERT INTO furnizor (

nume,

cod\_fiscal,

id\_adresa

) VALUES ( 'Pro Nutrition',

'12420890',

15 );

INSERT INTO furnizor (

nume,

cod\_fiscal,

id\_adresa

) VALUES ( 'Arena Systems',

'32120890',

16 );

EXEC sequence\_utils.create\_sequence\_trigger('Supliment');

INSERT INTO supliment (

nume,

descriere,

calorii,

pret

) VALUES ( 'Whey Protein',

'Zer premium cu 21 g de proteine per portie.',

'430',

'100' );

INSERT INTO supliment (

nume,

descriere,

calorii,

pret

) VALUES ( 'Izolat proteic din soia',

'O alegere excelenta pentru vegetarieni si vegani.',

300,

150 );

INSERT INTO supliment (

nume,

descriere,

calorii,

pret

) VALUES ( 'Vitafiber',

'Derivat din amidon de porumb nemodificat genetic.',

150,

210 );

INSERT INTO supliment (

nume,

descriere,

calorii,

pret

) VALUES ( 'Unt de arahide',

'Amestec pudra cu 70% mai putine grasimi.',

300,

90 );

INSERT INTO supliment (

nume,

descriere,

calorii,

pret

) VALUES ( 'Impact Diet Lean',

'Amestec fibre sub forma de fructo-oligozaharide.',

250,

200 );

INSERT INTO supliment (

nume,

descriere,

calorii,

pret

) VALUES ( 'Muscle Mass - pachet premium',

'Pachet complet: gainer de top + preworkout Complete Workout + formula pe baza de creatina.',

1000,

334 );

INSERT INTO supliment (

nume,

descriere,

calorii,

pret

) VALUES ( 'X-plode plicuri',

'Imbunatateste performanta fizica, regenerarea si volumizarea celulelor musculare.',

80,

56 );

INSERT INTO supliment (

nume,

descriere,

calorii,

pret

) VALUES ( 'Essential Amino Acids',

'Con?ine un mix de 8 aminoacizi esen?iali.',

30,

54 );

INSERT INTO supliment (

nume,

descriere,

calorii,

pret

) VALUES ( 'Jeleuri cu arom? de otet de cidru de mere',

'Ajuta la protejarea celulelor impotriva stresului oxidativ.',

10,

79 );

INSERT INTO supliment (

nume,

descriere,

calorii,

pret

) VALUES ( 'Jeleuri pre-antrenament',

'Un mod simplu de a va pregati mintal si fizic pentru fiecare antrenament.',

15,

129 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 1,

1,

10 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 3,

2,

10 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 1,

3,

20 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 3,

4,

50 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 1,

5,

15 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 2,

1,

10 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 2,

5,

20 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 2,

3,

90 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 3,

3,

70 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 8,

6,

5 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 8,

7,

15 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 8,

8,

20 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 8,

9,

15 );

INSERT INTO aprovizionare (

id\_furnizor,

id\_supliment,

cantitate

) VALUES ( 8,

10,

20 );

EXEC sequence\_utils.create\_sequence\_trigger('Echipament');

INSERT INTO echipament (

nume,

data\_instalare,

data\_revizie,

id\_filiala,

id\_furnizor

) VALUES ( 'Leg Press',

TO\_DATE('20-MAY-2020','DD-MON-YYYY'),

TO\_DATE('20-MAY-2021','DD-MON-YYYY'),

1,

5 );

INSERT INTO echipament (

nume,

data\_instalare,

data\_revizie,

id\_filiala,

id\_furnizor

) VALUES ( 'Chest Press',

TO\_DATE('20-JUN-2021','DD-MON-YYYY'),

TO\_DATE('20-JUN-2021','DD-MON-YYYY'),

1,

5 );

INSERT INTO echipament (

nume,

data\_instalare,

data\_revizie,

id\_filiala,

id\_furnizor

) VALUES ( 'Peck Deck',

TO\_DATE('01-JAN-2019','DD-MON-YYYY'),

TO\_DATE('01-JAN-2021','DD-MON-YYYY'),

2,

4 );

INSERT INTO echipament (

nume,

data\_instalare,

data\_revizie,

id\_filiala,

id\_furnizor

) VALUES ( 'Preacher Curl',

TO\_DATE('28-APR-2020','DD-MON-YYYY'),

TO\_DATE('28-APR-2021','DD-MON-YYYY'),

3,

3 );

INSERT INTO echipament (

nume,

data\_instalare,

data\_revizie,

id\_filiala,

id\_furnizor

) VALUES ( 'Calves Raises',

TO\_DATE('20-APR-2021','DD-MON-YYYY'),

TO\_DATE('20-APR-2022','DD-MON-YYYY'),

4,

3 );

INSERT INTO echipament (

nume,

data\_instalare,

data\_revizie,

id\_filiala,

id\_furnizor

) VALUES ( 'Lateral Raises',

TO\_DATE('10-APR-2021','DD-MON-YYYY'),

TO\_DATE('10-APR-2022','DD-MON-YYYY'),

4,

5 );

INSERT INTO echipament (

nume,

data\_instalare,

data\_revizie,

id\_filiala,

id\_furnizor

) VALUES ( 'Frontal Raises',

TO\_DATE('20-MAR-2020','DD-MON-YYYY'),

TO\_DATE('20-MAR-2022','DD-MON-YYYY'),

4,

5 );

INSERT INTO echipament (

nume,

data\_instalare,

data\_revizie,

id\_filiala,

id\_furnizor

) VALUES ( 'Sistem de catarare cu prindere pe perete',

TO\_DATE('30-SEP-2022','DD-MON-YYYY'),

TO\_DATE('30-SEP-2023','DD-MON-YYYY'),

8,

9 );

INSERT INTO echipament (

nume,

data\_instalare,

data\_revizie,

id\_filiala,

id\_furnizor

) VALUES ( 'Semisfera de echilibru cu manere',

TO\_DATE('30-JUN-2022','DD-MON-YYYY'),

TO\_DATE('30-JUN-2023','DD-MON-YYYY'),

8,

9 );

INSERT INTO echipament (

nume,

data\_instalare,

data\_revizie,

id\_filiala,

id\_furnizor

) VALUES ( 'Banca de gimnastica tip Pivetta',

TO\_DATE('01-JUN-2021','DD-MON-YYYY'),

TO\_DATE('01-JUN-2022','DD-MON-YYYY'),

3,

9 );

INSERT INTO echipament (

nume,

data\_instalare,

data\_revizie,

id\_filiala,

id\_furnizor

) VALUES ( 'Coarda sarituri cu maner din lemn',

TO\_DATE('01-JAN-2022','DD-MON-YYYY'),

TO\_DATE('01-JUN-2023','DD-MON-YYYY'),

3,

9 );

INSERT INTO echipament (

nume,

data\_instalare,

data\_revizie,

id\_filiala,

id\_furnizor

) VALUES ( 'Plan propioceptiv rotativ',

TO\_DATE('01-JUN-2023','DD-MON-YYYY'),

TO\_DATE('01-SEP-2023','DD-MON-YYYY'),

3,

9 );

EXEC sequence\_utils.create\_sequence\_trigger('Comanda');

INSERT INTO comanda (

id\_client,

id\_receptionist,

observatii,

data\_comandare

) VALUES ( 18,

8,

'Urgenta',

TO\_DATE('22-FEB-2022','DD-MON-YYYY') );

INSERT INTO comanda (

id\_client,

id\_receptionist,

observatii,

data\_comandare

) VALUES ( 18,

8,

'Preluare dupa ora 17',

TO\_DATE('11-MAR-2022','DD-MON-YYYY') );

INSERT INTO comanda (

id\_client,

id\_receptionist,

observatii,

data\_comandare

) VALUES ( 20,

9,

NULL,

TO\_DATE('01-APR-2022','DD-MON-YYYY') );

INSERT INTO comanda (

id\_client,

id\_receptionist,

observatii,

data\_comandare

) VALUES ( 20,

9,

NULL,

TO\_DATE('02-APR-2022','DD-MON-YYYY') );

INSERT INTO comanda (

id\_client,

id\_receptionist,

observatii,

data\_comandare

) VALUES ( 22,

9,

NULL,

TO\_DATE('22-APR-2022','DD-MON-YYYY') );

INSERT INTO comanda (

id\_client,

id\_receptionist,

observatii,

data\_comandare

) VALUES ( 26,

16,

'In curs de achitare',

TO\_DATE('01-SEP-2023','DD-MON-YYYY') );

INSERT INTO comanda (

id\_client,

id\_receptionist,

observatii,

data\_comandare

) VALUES ( 27,

17,

'Platita',

TO\_DATE('01-OCT-2023','DD-MON-YYYY') );

INSERT INTO comanda (

id\_client,

id\_receptionist,

observatii,

data\_comandare

) VALUES ( 27,

10,

'Platita',

TO\_DATE('11-OCT-2023','DD-MON-YYYY') );

INSERT INTO comanda (

id\_client,

id\_receptionist,

observatii,

data\_comandare

) VALUES ( 27,

11,

'Platita',

TO\_DATE('21-OCT-2023','DD-MON-YYYY') );

INSERT INTO comanda (

id\_client,

id\_receptionist,

observatii,

data\_comandare

) VALUES ( 27,

12,

'Platita',

TO\_DATE('22-OCT-2023','DD-MON-YYYY') );

INSERT INTO comanda (

id\_client,

id\_receptionist,

observatii,

data\_comandare

) VALUES ( 27,

13,

'Platita',

TO\_DATE('22-SEP-2022','DD-MON-YYYY') );

INSERT INTO tip\_abonament (

nume\_tip,

pret

) VALUES ( 'lunar',

100 );

INSERT INTO tip\_abonament (

nume\_tip,

pret

) VALUES ( 'trimestrial',

280 );

INSERT INTO tip\_abonament (

nume\_tip,

pret

) VALUES ( 'bianual',

550 );

INSERT INTO tip\_abonament (

nume\_tip,

pret

) VALUES ( 'anual',

800 );

INSERT INTO tip\_abonament (

nume\_tip,

pret

) VALUES ( 'extins',

1500 );

EXEC sequence\_utils.create\_sequence\_trigger('Abonament');

INSERT INTO abonament (

nume\_tip,

id\_client,

data\_inregistrare

) VALUES ( 'lunar',

18,

'01-APR-22' );

INSERT INTO abonament (

nume\_tip,

id\_client,

data\_inregistrare

) VALUES ( 'trimestrial',

19,

'01-APR-21' );

INSERT INTO abonament (

nume\_tip,

id\_client,

data\_inregistrare

) VALUES ( 'bianual',

20,

'01-FEB-22' );

INSERT INTO abonament (

nume\_tip,

id\_client,

data\_inregistrare

) VALUES ( 'extins',

21,

'01-SEP-21' );

INSERT INTO abonament (

nume\_tip,

id\_client,

data\_inregistrare

) VALUES ( 'anual',

22,

'01-NOV-20' );

INSERT INTO abonament (

nume\_tip,

id\_client,

data\_inregistrare

) VALUES ( 'anual',

23,

'01-NOV-22' );

INSERT INTO abonament (

nume\_tip,

id\_client,

data\_inregistrare

) VALUES ( 'anual',

25,

'01-DEC-22' );

INSERT INTO abonament (

nume\_tip,

id\_client,

data\_inregistrare

) VALUES ( 'bianual',

26,

'15-JUL-23' );

INSERT INTO abonament (

nume\_tip,

id\_client,

data\_inregistrare

) VALUES ( 'extins',

27,

'01-JAN-22' );

DECLARE

nr NUMBER;

BEGIN

FOR i IN 1..22 LOOP

SELECT round(dbms\_random.value(

1000000000,

9999999999

))

INTO nr

FROM dual;

IF i <= 10 THEN

INSERT INTO telefon (

tip,

numar,

id\_persoana

) VALUES ( 'serviciu',

nr,

i );

ELSE

INSERT INTO telefon (

tip,

numar,

id\_persoana

) VALUES ( 'personal',

nr,

i );

END IF;

END LOOP;

END;

/

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 1,

1,

2 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 1,

2,

1 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 1,

3,

4 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 1,

4,

3 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 1,

5,

7 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 2,

1,

2 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 3,

1,

2 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 3,

2,

1 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 3,

4,

1 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 3,

5,

5 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 6,

4,

3 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 6,

3,

1 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 6,

7,

1 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 6,

8,

2 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 6,

2,

1 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 7,

1,

4 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 7,

3,

2 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 7,

7,

1 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 7,

8,

5 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 7,

9,

2 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 7,

10,

1 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 8,

7,

1 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 9,

8,

5 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 10,

9,

2 );

INSERT INTO informatii\_comanda (

id\_comanda,

id\_supliment,

cantitate

) VALUES ( 11,

10,

1 );

commit;

**10.1.4. bro\_admin\_criptare.sql**

--- Cripatre in schema lui bro\_admin

create or replace function select\_random\_from\_nr\_list (

input\_list sys.odcinumberlist

) return number is

selected\_value number;

begin

select input\_list(trunc(dbms\_random.value(

1,

input\_list.count + 1

)))

into selected\_value

from dual;

return selected\_value;

end;

/

create table chei\_client (

id\_client number

constraint fk\_client\_chei

references client ( id\_client )

primary key,

mod\_op int not null,

cheie raw(16) not null

);

create or replace procedure insert\_into\_chei\_client (

p\_client number

) is

mod\_op\_value number;

v\_nr smallint;

v\_cheie raw(16) := dbms\_crypto.randombytes(16);

begin

select count(\*)

into v\_nr

from chei\_client c

where c.id\_client = p\_client;

if v\_nr > 0 then

raise\_application\_error(

-20022,

'The client is already in the table keys. Client ' || p\_client

);

end if;

mod\_op\_value := dbms\_crypto.encrypt\_aes128 + bro\_admin.select\_random\_from\_nr\_list(sys.odcinumberlist(

dbms\_crypto.pad\_pkcs5,

dbms\_crypto.pad\_zero

)) + bro\_admin.select\_random\_from\_nr\_list(sys.odcinumberlist(

dbms\_crypto.chain\_cbc,

dbms\_crypto.chain\_cfb,

dbms\_crypto.chain\_ecb,

dbms\_crypto.chain\_ofb

));

insert into chei\_client (

id\_client,

mod\_op,

cheie

) values ( p\_client,

mod\_op\_value,

v\_cheie );

end;

/

exec insert\_into\_chei\_client(18);

exec insert\_into\_chei\_client(18);

exec insert\_into\_chei\_client(19);

exec insert\_into\_chei\_client(20);

exec insert\_into\_chei\_client(21);

exec insert\_into\_chei\_client(22);

exec insert\_into\_chei\_client(25);

exec insert\_into\_chei\_client(26);

exec insert\_into\_chei\_client(27);

select \*

from chei\_client;

commit;

create or replace type chei\_client\_object as object (

id\_client number,

mod\_op int,

cheie raw(16)

);

/

create or replace function get\_client\_key return chei\_client\_object is

res chei\_client\_object;

begin

select chei\_client\_object(

c.id\_client,

c.mod\_op,

c.cheie

)

into res

from account\_mapping a

join chei\_client c

on a.id\_persoana = c.id\_client

where username = sys\_context(

'userenv',

'session\_user'

);

return res;

exception

when no\_data\_found then

raise\_application\_error(

-20023,

'Client with username '

|| sys\_context(

'userenv',

'session\_user'

)

|| ' does not have a cript key'

);

end;

/

**10.1.5. bro\_admin\_mask.sql**

create or replace package mask\_person is

function mask\_item (

item varchar2

) return varchar2;

function mask\_item (

item number

) return number;

function mask\_person\_id (

item number

) return number;

function mask\_person\_fk (

item number

) return number;

procedure empty\_person\_ids;

end;

/

create or replace package body mask\_person is

type id\_rec is record (

old\_value int,

new\_value int

);

type new\_key\_rec is record (

new\_val int,

new\_max int,

new\_min int

);

type ids\_tbl is

table of id\_rec index by pls\_integer;

person\_ids ids\_tbl;

function find\_person\_by\_value (

val int,

raise\_empty boolean := false,

use\_old\_val boolean := true

) return int is

begin

dbms\_output.put\_line(person\_ids.count);

for i in 1..person\_ids.count loop

if use\_old\_val then

if person\_ids(i).old\_value = val then

return person\_ids(i).new\_value;

end if;

else

if person\_ids(i).new\_value = val then

return person\_ids(i).old\_value;

end if;

end if;

end loop;

if raise\_empty then

raise\_application\_error(

-20020,

'Id-ul '

|| val

|| ' nu este prezent in baza originala'

);

else

return null;

end if;

end;

function generate\_new\_key (

val int,

max\_len\_factor int := 1

) return new\_key\_rec is

len int := length(to\_char(val));

new\_max\_len int := len \* max\_len\_factor;

new\_key new\_key\_rec;

begin

if new\_max\_len > 38 then

new\_max\_len := 38; -- mx nr id

end if;

new\_key.new\_min := to\_number ( rpad(

substr(

to\_char(val),

1,

1

),

len,

'0'

) );

new\_key.new\_max := to\_number ( rpad(

substr(

to\_char(val),

1,

1

),

new\_max\_len,

'9'

) ); -- to not have colission as often

dbms\_random.seed(val => val);

new\_key.new\_val := round(

dbms\_random.value(

low => new\_key.new\_min,

high => new\_key.new\_max

),

0

);

return new\_key;

end;

function append\_to\_person\_list (

val int

) return int is

rec int := find\_person\_by\_value(val);

pers\_cnt int := person\_ids.count + 1;

new\_key int;

begin

if rec is not null then

return rec;

end if;

new\_key := generate\_new\_key(

val,

5

).new\_val;

while ( find\_person\_by\_value(

val => new\_key,

use\_old\_val => false

) is not null

or find\_person\_by\_value(

val => new\_key,

use\_old\_val => true

) is not null ) loop

new\_key := generate\_new\_key(

val,

5

).new\_val; -- no colisions

end loop;

person\_ids(pers\_cnt).old\_value := val;

person\_ids(pers\_cnt).new\_value := new\_key;

return new\_key;

end;

-- ne trebuie tampenie asta crunta pt ca oracle exporta alfabetic

-- nu stiu dc face asta, dar asa face

procedure load\_person\_ids is

TYPE id\_list\_type IS TABLE OF persoana.id\_persoana%TYPE;

id\_list id\_list\_type;

dummy\_val int;

begin

if person\_ids.count = 0 then

SELECT id\_persoana

BULK COLLECT INTO id\_list

FROM persoana;

FOR i IN 1 .. id\_list.COUNT LOOP

dummy\_val := append\_to\_person\_list(id\_list(i));

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('IDs have been initialized in the table type.');

end if;

end;

--public

function mask\_person\_id (

item number

) return number is

begin

load\_person\_ids();

-- return append\_to\_person\_list(item);

return find\_person\_by\_value(

val => item,

raise\_empty => true

);

end;

function mask\_item (

item varchar2

) return varchar2 is

masked\_item varchar2(30);

new\_length number;

random\_char char(1);

begin

if dbms\_random.value(

0,

1

) > 0.5 then

new\_length := length(item) \* 2;

else

new\_length := length(item);

end if;

if new\_length > 30 then

new\_length := 30; --max string length

end if;

masked\_item := substr(

item,

1,

1

);

for i in 2..new\_length loop

if dbms\_random.value(

0,

1

) > 0.5 then

random\_char :=

case

when dbms\_random.value(

0,

1

) > 0.5 then

'\*'

else '#'

end;

masked\_item := masked\_item || random\_char;

end if;

end loop;

return masked\_item;

end;

function mask\_item (

item number

) return number is

begin

return generate\_new\_key(item).new\_val;

end;

function mask\_person\_fk (

item number

) return number is

begin

if item is null then

return null;

end if;

load\_person\_ids();

return find\_person\_by\_value(

val => item,

raise\_empty => true

);

end;

procedure empty\_person\_ids is

begin

person\_ids.delete;

end;

end;

/

**10.1.6. bro\_admin\_programs\_view.sql**

set serveroutput on;

declare

users sys.global\_user\_table;

begin

users := sys.get\_users\_by\_suffix('ANTRENOR');

for i in 1..users.count loop

dbms\_output.put\_line(users(i));

end loop;

end;

/

create or replace procedure bro\_admin\_programs\_view is

v\_user sys.global\_user\_table := sys.get\_users\_by\_suffix('ANTRENOR');

v\_sql clob := 'CREATE OR REPLACE VIEW programs\_view AS ';

v\_first boolean := true;

begin

for i in 1..v\_user.count loop

begin

-- check existance of program table for antrenor

execute immediate 'SELECT 1 FROM '

|| v\_user(i)

|| '.program WHERE ROWNUM = 1';

if v\_first then

v\_sql := v\_sql

|| 'SELECT '''

|| v\_user(i)

|| ''' AS antrenor, id\_program, descriere, tip\_program FROM '

|| v\_user(i)

|| '.program';

v\_first := false;

else

v\_sql := v\_sql

|| ' UNION ALL SELECT '''

|| v\_user(i)

|| ''' AS antrenor, id\_program, descriere, tip\_program FROM '

|| v\_user(i)

|| '.program';

end if;

exception

when others then

-- skip if table its not in antrenor

dbms\_output.put\_line('Skipping '

|| v\_user(i)

|| '.program as it does not exist.');

end;

end loop;

dbms\_output.put\_line(v\_sql);

if not v\_first then

execute immediate v\_sql;

dbms\_output.put\_line('View programs\_view created successfully.');

else

dbms\_output.put\_line('No valid program tables found. View not created.');

end if;

end bro\_admin\_programs\_view;

/

exec bro\_admin\_programs\_view;

select \*

from programs\_view;

-- poate da grant la role, desi nu vede rolurile

--SELECT role FROM dba\_roles WHERE role = 'R\_BRO\_PUBLIC\_GENERAL';

grant select on bro\_admin.programs\_view to r\_bro\_public\_general;

**10.1.7. bro\_admin\_update\_echipament\_fals.sql**

begin

for i in 1..20 loop

update echipament o

set

data\_revizie = (

select data\_revizie

from echipament i

where i.id\_echipament = o.id\_echipament

);

commit;

end loop;

end;

/

**10.2 Antrenor**

**10.2.1. bro\_antrenor\_insert.sql**

-- Inserare date in tabelele din baza de date pentru antrenor

insert into program (

descriere,

tip\_program

) values ( 'Push, Pull Legs Light, for beginners',

'MASS' );

insert into program (

descriere,

tip\_program

) values ( 'Push, Pull Legs Medium',

'MASS' );

insert into program (

descriere,

tip\_program

) values ( 'Push, Pull Legs Hard',

'MASS' );

insert into program (

descriere,

tip\_program

) values ( 'Full Body Variant Light',

'CARDIO' );

insert into program (

descriere,

tip\_program

) values ( 'Body Recovery Variant Light',

'RECOVERY' );

insert into program (

descriere,

tip\_program

) values ( 'Body Bluster Variant Blusting',

'RECOVERY' );

insert into program (

descriere,

tip\_program

) values ( 'Cardio Workout for Weight Loss',

'CARDIO' );

insert into program (

descriere,

tip\_program

) values ( 'Cardio workout for beginners',

'CARDIO' );

insert into program (

descriere,

tip\_program

) values ( 'Cardio workout for older adults',

'CARDIO' );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 20,

1,

1,

18 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 11,

1,

2,

18 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 11,

4,

1,

18 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 10,

4,

2,

18 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 5,

1,

1,

19 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 25,

1,

2,

19 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 32,

1,

3,

19 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 10,

5,

2,

19 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 5,

4,

3,

20 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 25,

4,

4,

20 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 12,

4,

5,

21 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 42,

4,

2,

21 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 20,

4,

5,

22 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 10,

4,

4,

22 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 10,

7,

8,

25 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 20,

7,

9,

25 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 10,

7,

10,

25 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 10,

8,

3,

26 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 20,

8,

4,

19 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 10,

8,

5,

19 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 10,

8,

12,

27 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 20,

8,

9,

27 );

insert into antrenament (

durata,

id\_program,

id\_echipament,

id\_client

) values ( 10,

8,

11,

27 );

commit;

**10.2.2. bro\_antrenor1\_cript\_show.sql**

with c as (

select mod\_op,

cheie

from bro\_admin.chei\_client

where id\_client = 18

)

select decript\_string(

ca.id\_program,

c.mod\_op,

c.cheie

) as id\_program,

decript\_string(

ca.descriere\_program,

c.mod\_op,

c.cheie

) as descriere\_program,

decript\_string(

ca.tip\_program,

c.mod\_op,

c.cheie

) as tip\_program,

decript\_string(

ca.durata\_antrenament,

c.mod\_op,

c.cheie

) as durata\_antrenament,

decript\_string(

ca.id\_echipament,

c.mod\_op,

c.cheie

) as id\_echipament,

decript\_string(

ca.nume\_echipament,

c.mod\_op,

c.cheie

) as nume\_echipament,

decript\_string(

ca.data\_instalare\_echipament,

c.mod\_op,

c.cheie

) as data\_instalare\_echipament,

decript\_string(

ca.data\_revizie\_echipament,

c.mod\_op,

c.cheie

) as data\_revizie\_echipament,

decript\_string(

ca.id\_filiala,

c.mod\_op,

c.cheie

) as id\_filiala,

decript\_string(

ca.id\_client,

c.mod\_op,

c.cheie

) as id\_client,

checksum

from client\_antrenament ca,

c;

select \*

from table ( fetch\_decrypted\_client\_data(

(

select mod\_op

from bro\_admin.chei\_client

where id\_client = 18

),

(

select cheie

from bro\_admin.chei\_client

where id\_client = 18

),

18

) );

**10.2.3. bro\_antrenor1\_sql\_injection.sql**

CREATE OR REPLACE PROCEDURE get\_program\_full (

id\_prg NUMBER,

data\_inst VARCHAR2

) IS

TYPE program\_echipament\_rec IS RECORD (

id\_echipament bro\_admin.echipament.id\_echipament%TYPE,

nume bro\_admin.echipament.nume%TYPE,

data\_instalare bro\_admin.echipament.data\_instalare%TYPE,

data\_revizie bro\_admin.echipament.data\_revizie%TYPE,

id\_filiala bro\_admin.echipament.id\_filiala%TYPE,

id\_furnizor bro\_admin.echipament.id\_furnizor%TYPE,

id\_program program.id\_program%TYPE,

descriere program.descriere%TYPE,

tip\_program program.tip\_program%TYPE

);

TYPE program\_echipament\_tab IS

TABLE OF program\_echipament\_rec;

v\_program\_echipament program\_echipament\_tab;

v\_sql VARCHAR2(2500) := '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)

|| '%''';

BEGIN

dbms\_output.put\_line('SQL: ' || v\_sql);

EXECUTE IMMEDIATE v\_sql

BULK COLLECT

INTO v\_program\_echipament;

dbms\_output.put\_line('Program and Equipment Details:');

dbms\_output.new\_line();

FOR i IN 1..v\_program\_echipament.count LOOP

dbms\_output.put\_line('ID\_ECHIPAMENT: '

|| v\_program\_echipament(i).id\_echipament

|| ', NUME: '

|| v\_program\_echipament(i).nume

|| ', DATA\_INSTALARE: '

|| to\_char(

v\_program\_echipament(i).data\_instalare,

'YYYY-MM-DD'

)

|| ', DATA\_REVIZIE: '

|| to\_char(

v\_program\_echipament(i).data\_revizie,

'YYYY-MM-DD'

)

|| ', ID\_FILIALA: '

|| v\_program\_echipament(i).id\_filiala

|| ', ID\_FURNIZOR: '

|| v\_program\_echipament(i).id\_furnizor

|| ', ID\_PROGRAM: '

|| v\_program\_echipament(i).id\_program

|| ', DESCRIERE: '

|| v\_program\_echipament(i).descriere

|| ', TIP\_PROGRAM: '

|| v\_program\_echipament(i).tip\_program);

dbms\_output.new\_line();

END LOOP;

EXCEPTION

WHEN no\_data\_found THEN

dbms\_output.put\_line('No data found for the specified program ID: ' || id\_prg);

WHEN OTHERS THEN

dbms\_output.put\_line('An error occurred running get\_program\_full: ' || sqlerrm);

END;

/

GRANT EXECUTE ON get\_program\_full TO bro\_client1;

/

-- sql injection fix

CREATE OR REPLACE PROCEDURE get\_program\_full\_safe (

id\_prg NUMBER,

data\_inst VARCHAR2

) IS

TYPE program\_echipament\_rec IS RECORD (

id\_echipament bro\_admin.echipament.id\_echipament%TYPE,

nume bro\_admin.echipament.nume%TYPE,

data\_instalare bro\_admin.echipament.data\_instalare%TYPE,

data\_revizie bro\_admin.echipament.data\_revizie%TYPE,

id\_filiala bro\_admin.echipament.id\_filiala%TYPE,

id\_furnizor bro\_admin.echipament.id\_furnizor%TYPE,

id\_program program.id\_program%TYPE,

descriere program.descriere%TYPE,

tip\_program program.tip\_program%TYPE

);

TYPE program\_echipament\_tab IS TABLE OF program\_echipament\_rec;

v\_program\_echipament program\_echipament\_tab;

v\_sql VARCHAR2(2500) := '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';

BEGIN

dbms\_output.put\_line('Executing SQL: ' || v\_sql);

EXECUTE IMMEDIATE v\_sql BULK COLLECT

INTO v\_program\_echipament

USING id\_prg, '%' || upper(data\_inst) || '%';

dbms\_output.put\_line('Program and Equipment Details:');

dbms\_output.new\_line();

FOR i IN 1 .. v\_program\_echipament.count LOOP

dbms\_output.put\_line('ID\_ECHIPAMENT: ' || v\_program\_echipament(i).id\_echipament ||

', NUME: ' || v\_program\_echipament(i).nume ||

', DATA\_INSTALARE: ' || to\_char(v\_program\_echipament(i).data\_instalare, 'YYYY-MM-DD') ||

', DATA\_REVIZIE: ' || to\_char(v\_program\_echipament(i).data\_revizie, 'YYYY-MM-DD') ||

', ID\_FILIALA: ' || v\_program\_echipament(i).id\_filiala ||

', ID\_FURNIZOR: ' || v\_program\_echipament(i).id\_furnizor ||

', ID\_PROGRAM: ' || v\_program\_echipament(i).id\_program ||

', DESCRIERE: ' || v\_program\_echipament(i).descriere ||

', TIP\_PROGRAM: ' || v\_program\_echipament(i).tip\_program);

dbms\_output.new\_line();

END LOOP;

EXCEPTION

WHEN no\_data\_found THEN

dbms\_output.put\_line('No data found for the specified program ID: ' || id\_prg);

WHEN OTHERS THEN

dbms\_output.put\_line('An error occurred running get\_program\_full: ' || sqlerrm);

END;

/

GRANT EXECUTE ON get\_program\_full\_safe TO bro\_client1;

**10.3. Client**

**10.3.1. bro\_client1\_select\_cript.sql**

select bro\_admin.get\_client\_key

from dual;

select ( bro\_admin.get\_client\_key() ).id\_client as id\_client,

( bro\_admin.get\_client\_key() ).mod\_op as mod\_op,

( bro\_admin.get\_client\_key() ).cheie as cheie

from dual;

with user\_key as (

select bro\_admin.get\_client\_key() as client\_key

from dual

)

select ( user\_key.client\_key ).id\_client as id\_client,

( user\_key.client\_key ).mod\_op as mod\_op,

( user\_key.client\_key ).cheie as cheie

from user\_key;

with user\_key as (

select bro\_admin.get\_client\_key() as client\_key

from dual

)

SELECT ant.\*,cs.\*,

case when ant.checksum = cs.cur\_cs then 'ok' else 'not ok' end as cs\_v

FROM

(SELECT bro\_admin.get\_client\_key() AS client\_key

FROM dual)

user\_key,

LATERAL (SELECT \*FROM TABLE(

bro\_antrenor1.fetch\_decrypted\_client\_data(

p\_mod\_op=>user\_key.client\_key.mod\_op,

p\_cheie=>user\_key.client\_key.cheie,

p\_id\_client=>user\_key.client\_key.id\_client))) ant,

lateral (

select bro\_antrenor1.hash\_checksum(SYS.ODCIVARCHAR2LIST(ant.id\_program,ant.descriere\_program,ant.tip\_program,

ant.durata\_antrenament,ant.id\_echipament,ant.nume\_echipament,

ant.data\_instalare\_echipament,ant.data\_revizie\_echipament,ant.id\_filiala,

user\_key.client\_key.cheie)) as cur\_cs from dual

) cs;

select \* from bro\_admin.programs\_view;

**10.3.2 bro\_client1\_sql\_injection.sql**

set serveroutput on;

-- apel onest

exec bro\_antrenor1.get\_program\_full(1,'may');

--apel care intoarce toate programele cu echipamente, subminand filtrarea

exec bro\_antrenor1.get\_program\_full(1,'may%'' OR 1=1 --');

--apel care intorace toate antrenamentele, desi clientul nu are drept de select pe tabela antrenament

select \* from bro\_antrenor1.antrenament;

exec bro\_antrenor1.get\_program\_full(1, 'may%'' UNION SELECT ID\_ECHIPAMENT, ''Injectat'', SYSDATE, SYSDATE, ID\_CLIENT, DURATA, ID\_PROGRAM, ''Injectat Desc'', ''Tip injectat'' FROM ANTRENAMENT --');

begin

bro\_antrenor1.get\_program\_full(1, 'may%'' UNION SELECT ID\_ECHIPAMENT, ''Injectat'',

SYSDATE, SYSDATE, ID\_CLIENT, DURATA, ID\_PROGRAM,

''Injectat Desc'', ''Tip injectat'' FROM ANTRENAMENT --');

end;

/

-- repetam cu safe

-- apel onest

exec bro\_antrenor1.get\_program\_full\_safe(1,'may');

--apel care intoarce toate programele cu echipamente, subminand filtrarea

exec bro\_antrenor1.get\_program\_full\_safe(1,'may%'' OR 1=1 --');

exec bro\_antrenor1.get\_program\_full\_safe(1, 'may%'' UNION SELECT ID\_ECHIPAMENT, ''Injectat'', SYSDATE, SYSDATE, ID\_CLIENT, DURATA, ID\_PROGRAM, ''Injectat Desc'', ''Tip injectat'' FROM ANTRENAMENT --');

begin

bro\_antrenor1.get\_program\_full\_safe(1, 'may%'' UNION SELECT ID\_ECHIPAMENT, ''Injectat'',

SYSDATE, SYSDATE, ID\_CLIENT, DURATA, ID\_PROGRAM,

''Injectat Desc'', ''Tip injectat'' FROM ANTRENAMENT --');

end;

/

**10.4. Import**

**10.4.1. bro\_import.sql**

SELECT CONSTRAINT\_NAME, CONSTRAINT\_TYPE

FROM USER\_CONSTRAINTS

WHERE TABLE\_NAME = upper('angajat\_mask');

select \* from persoana\_mask;

select \* from angajat\_mask

join persoana\_mask

on id\_angajat=id\_persoana;

**10.5 Manager**

**10.5.1. bro\_manager\_filiala1\_context.sql**

select sys\_context(

'bro\_context',

'id\_filiala'

) from dual;

select distinct id\_filiala from bro\_admin.echipament;

update bro\_admin.echipament o

set

nume = (

select nume

from bro\_admin.echipament i

where i.id\_echipament = o.id\_echipament

)

where o.id\_filiala = 1;

update bro\_admin.echipament o

set

nume = (

select nume

from bro\_admin.echipament i

where i.id\_echipament = o.id\_echipament

)

where o.id\_filiala != 1;

update bro\_admin.echipament o

set

nume = (

select nume

from bro\_admin.echipament i

where i.id\_echipament = o.id\_echipament

);

select count(\*)

from bro\_admin.audit\_echipament a

where a.old\_values.id\_filiala=1 or a.old\_values.id\_filiala=1;

select count(\*)

from bro\_admin.audit\_echipament a

where a.old\_values.id\_filiala!=1 and a.old\_values.id\_filiala!=1;

**10.7. SYS**

**10.7.1. sys\_admin\_antrenor\_privilege.sql**

-- dupa ce admin a create tabelel pt antrenori

alter session set container = orclpdb;

create or replace procedure bro\_admin\_programs\_privilegies is

v\_user global\_user\_table := get\_users\_by\_suffix('ANTRENOR');

v\_first boolean := true;

begin

for i in 1..v\_user.count loop

dbms\_output.put\_line(v\_user(i));

begin

execute immediate 'SELECT 1 FROM '

|| v\_user(i)

|| '.program WHERE ROWNUM = 1';

execute immediate 'grant select on '

|| v\_user(i)

|| '.program to bro\_admin with grant option';

dbms\_output.put\_line('Giving select privileges on '

|| v\_user(i)

|| '.program to bro\_admin.');

if v\_first then

v\_first := false;

end if;

exception

when others then

dbms\_output.put\_line('Skipping '

|| v\_user(i)

|| '.program as it does not exist.');

end;

end loop;

if not v\_first then

dbms\_output.put\_line('Giving select privileges on program tables to bro\_admin.');

else

dbms\_output.put\_line('No valid program tables found');

end if;

end bro\_admin\_programs\_privilegies;

/

set SERVEROUTPUT ON;

exec bro\_admin\_programs\_privilegies;

**10.7.2. sys\_audit\_1.sql**

alter session set container = orclpdb;

SHOW parameter audit\_trail;

audit insert,update on bro\_admin.client\_extins by access whenever not successful;

audit insert,update,delete on bro\_admin.echipament;

audit insert,update,delete on bro\_admin.account\_mapping;

audit insert,update on bro\_admin.supliment;

create or replace directory json\_dir as 'D:\ORACLEEE\INSTALL\ADMIN\ORCL\MYDUMP';

-- Multumiri deosebite primului raspuns de aici si nu documentatiei oracle care este oribila

-- https://stackoverflow.com/questions/50417586/write-a-clob-to-file-in-oracle

create or replace procedure convert\_clob\_2\_file (

p\_filename in varchar2,

p\_dir in varchar2,

p\_clob in clob

) as

v\_lob\_image\_id number;

v\_clob clob := p\_clob;

v\_buffer raw(32767);

c\_buffer varchar2(32767);

v\_buffer\_size binary\_integer;

v\_amount binary\_integer;

v\_pos number(38) := 1;

v\_clob\_size integer;

v\_out\_file utl\_file.file\_type;

begin

v\_pos := 1;

v\_clob\_size := dbms\_lob.getlength(v\_clob);

v\_buffer\_size := 32767;

v\_amount := v\_buffer\_size;

if ( dbms\_lob.isopen(v\_clob) = 0 ) then

dbms\_lob.open(

v\_clob,

dbms\_lob.lob\_readonly

);

end if;

v\_out\_file := utl\_file.fopen(

p\_dir,

p\_filename,

'WB',

max\_linesize => 32767

);

while v\_amount >= v\_buffer\_size loop

dbms\_lob.read(

v\_clob,

v\_amount,

v\_pos,

c\_buffer

);

v\_buffer := utl\_raw.cast\_to\_raw(c\_buffer);

v\_pos := v\_pos + v\_amount;

utl\_file.put\_raw(

v\_out\_file,

v\_buffer,

true

);

utl\_file.fflush(v\_out\_file);

end loop;

utl\_file.fflush(v\_out\_file);

utl\_file.fclose(v\_out\_file);

if ( dbms\_lob.isopen(v\_clob) = 1 ) then

dbms\_lob.close(v\_clob);

end if;

exception

when others then

if ( dbms\_lob.isopen(v\_clob) = 1 ) then

dbms\_lob.close(v\_clob);

end if;

raise;

end;

/

create or replace procedure save\_audit\_to\_json (

p\_obj\_name varchar2

) is

obj\_name varchar2(128) := dbms\_assert.simple\_sql\_name(p\_obj\_name);

a\_file utl\_file.file\_type;

a\_json clob;

a\_filename varchar2(1000);

begin

for owner\_rec in (

select distinct obj$creator

from sys.aud$

where obj$name = upper(obj\_name)

) loop

a\_filename := 'audit\_json\_bro\_'

|| owner\_rec.obj$creator

|| '\_'

|| obj\_name

|| '\_'

|| to\_char(

sysdate,

'YYYYMMDD\_HH24MISS'

)

|| '.json';

select to\_clob(json\_arrayagg(

json\_object(

key 'sessionId' value sessionid,

key 'userId' value userid,

key 'entryId' value entryid,

key 'userhost' value userhost,

key 'returncode' value returncode,

key 'timestamp' value to\_char(

ntimestamp#,

'yyyy-mm-dd hh24:mi:ss'

),

key 'sqltext' value sqltext,

key 'objectName' value obj\_name,

key 'objectOwner' value owner\_rec.obj$creator

returning clob)

returning clob))

into a\_json

from sys.aud$

where obj$name = upper(obj\_name)

and obj$creator = upper(owner\_rec.obj$creator);

convert\_clob\_2\_file(

p\_clob => a\_json,

p\_dir => 'JSON\_DIR',

p\_filename => a\_filename

);

execute immediate 'delete from SYS.AUD$ where obj$name = upper(:1) and obj$creator = upper(:2)'

using obj\_name,

owner\_rec.obj$creator;

end loop;

commit;

exception

when others then

dbms\_output.put\_line('Error occurred: ' || sqlerrm);

raise;

end;

/

exec save\_audit\_to\_json('client\_extins');

exec save\_audit\_to\_json('echipament');

exec save\_audit\_to\_json('account\_mapping');

-- create audit jobs

declare

job\_names odcivarchar2list := odcivarchar2list(

'client\_extins',

'echipament',

'account\_mapping',

'supliment'

);

begin

for i in 1..job\_names.count loop

dbms\_scheduler.create\_job(

job\_name => 'SAVE\_AUDIT\_TO\_JSON\_JOB\_' || upper(job\_names(i)),

job\_type => 'PLSQL\_BLOCK',

job\_action => 'BEGIN save\_audit\_to\_json('''

|| upper(job\_names(i))

|| '''); END;',

start\_date => systimestamp,

repeat\_interval => 'FREQ=DAILY; BYHOUR=17; BYMINUTE=22; BYSECOND=0',

enabled => true

);

end loop;

end;

/

-- drop audit jobs

-- declare

-- job\_names odcivarchar2list;

-- begin

-- select job\_name

-- bulk collect

-- into job\_names

-- from user\_scheduler\_jobs

-- where job\_name like 'SAVE\_AUDIT\_TO\_JSON\_JOB\_%';

-- for i in 1..job\_names.count loop

-- dbms\_scheduler.drop\_job(job\_name => upper(job\_names(i)));

-- end loop;

-- end;

-- /

-- noaudit all on bro\_admin.client\_extins;

-- noaudit all on bro\_admin.echipament;

-- noaudit all on bro\_admin.account\_mapping;

-- noaudit all on bro\_admin.supliment;

select \* from dba\_audit\_trail

where username like upper('bro%');

SELECT \* FROM dba\_scheduler\_jobs

WHERE job\_name LIKE upper('save\_audit\_to\_json\_job\_%')

ORDER BY job\_name;

**10.7.3. sys\_audit\_2.sql**

alter session set container = orclpdb;

create or replace directory fgadump\_dir as 'D:\OracleEE\install\admin\orcl\fgadump';

-- salvare intr-un fisier txt a logurilor

create or replace procedure bro\_audit\_tablese\_handler (

object\_schema varchar2,

object\_name varchar2,

policy\_name varchar2

) is

fga\_file utl\_file.file\_type;

log\_message varchar2(5000);

begin

log\_message := 'FGA Triggered:'

|| chr(10)

|| 'Timestamp: '

|| to\_char(

sysdate,

'YYYY-MM-DD HH24:MI:SS'

)

|| chr(10)

|| 'Object Schema: '

|| object\_schema

|| chr(10)

|| 'Object Name: '

|| object\_name

|| chr(10)

|| 'Policy Name: '

|| policy\_name

|| chr(10);

fga\_file := utl\_file.fopen(

'FGADUMP\_DIR',

policy\_name || '.txt',

'a'

);

utl\_file.put\_line(

fga\_file,

log\_message

);

utl\_file.fflush(fga\_file);

utl\_file.fclose(fga\_file);

exception

when others then

if utl\_file.is\_open(fga\_file) then

utl\_file.fclose(fga\_file);

end if;

raise;

end;

/

create or replace procedure echipament\_revizie\_audit is

begin

dbms\_fga.add\_policy(

object\_schema => 'BRO\_ADMIN',

object\_name => 'ECHIPAMENT',

policy\_name => 'AUDIT\_DATA\_REVIZIE',

audit\_condition => 'data\_revizie IS NOT NULL',

audit\_column => 'data\_revizie',

handler\_schema => 'SYS',

handler\_module => 'bro\_audit\_tablese\_handler',

enable => true,

statement\_types => 'insert,update'

);

end;

/

exec echipament\_revizie\_audit();

select \*

from dba\_fga\_audit\_trail

where policy\_name = 'AUDIT\_DATA\_REVIZIE';

begin

dbms\_fga.enable\_policy(

object\_schema => 'BRO\_ADMIN',

object\_name => 'ECHIPAMENT',

policy\_name => 'AUDIT\_DATA\_REVIZIE'

);

end;

/

-- begin

-- dbms\_fga.disable\_policy(

-- object\_schema => 'BRO\_ADMIN',

-- object\_name => 'ECHIPAMENT',

-- policy\_name => 'AUDIT\_DATA\_REVIZIE'

-- );

-- end;

-- /

-- begin

-- dbms\_fga.drop\_policy(

-- object\_schema => 'BRO\_ADMIN',

-- object\_name => 'ECHIPAMENT',

-- policy\_name => 'AUDIT\_DATA\_REVIZIE'

-- );

-- end;

-- /

**10.7.4 sys\_context.sql**

alter session set container = orclpdb;

--creare contextului

create or replace procedure bro\_context\_proc is

begin

if regexp\_like(

upper(user),

'^BRO\_ADMIN'

) then

dbms\_session.set\_context(

'bro\_context',

'id\_filiala',

-1

); -- pentru a lasa adminul in pace

elsif regexp\_like(

upper(user),

'^BRO\_MANAGER\_FILIALA[0-9]+$'

) then

dbms\_session.set\_context(

'bro\_context',

'id\_filiala',

regexp\_substr(

user,

'BRO\_MANAGER\_FILIALA([0-9]+)',

1,

1,

null,

1

)

);

end if;

end;

/

drop context bro\_context;

create context bro\_context using bro\_context\_proc;

--trigger pt a popula contextul

create or replace trigger manager\_id\_filiala\_trg

after logon on database begin

bro\_context\_proc();

end;

/

-- fiecare manager face dml pe filiala lui

create or replace function manager\_echipament\_management\_policy (

schema\_name in varchar2,

table\_name in varchar2

) return varchar2 is

v\_id\_filiala int := sys\_context(

'bro\_context',

'id\_filiala'

);

begin

if v\_id\_filiala = -1 then

return ''; -- pentru a lasa adminul in pace

else

return 'id\_filiala = ' || v\_id\_filiala;

end if;

end;

/

begin

dbms\_rls.add\_policy(

object\_schema => 'BRO\_ADMIN',

object\_name => 'ECHIPAMENT',

policy\_name => 'MANAGER\_ECHIPAMENT\_POLICY',

function\_schema => 'SYS',

policy\_function => 'manager\_echipament\_management\_policy',

statement\_types => 'INSERT, UPDATE, DELETE',

update\_check => true,

enable => true

);

end;

/

-- begin

-- dbms\_rls.drop\_policy(

-- object\_schema => 'BRO\_ADMIN',

-- object\_name => 'ECHIPAMENT',

-- policy\_name => 'MANAGER\_ECHIPAMENT\_POLICY'

-- );

-- end;

-- /

-- fiecare manager vede istoric cu leagatura la filiala lui

create or replace function audit\_echipament\_policy (

schema\_name in varchar2,

table\_name in varchar2

) return varchar2 as

v\_id\_filiala int := sys\_context(

'bro\_context',

'id\_filiala'

);

begin

if upper(user) = upper(schema\_name) then

return ''; -- pentru a lasa adminul in pace

else

return '

(TREAT(old\_values AS bro\_admin.t\_echipament).id\_filiala = '

|| v\_id\_filiala

|| ' OR TREAT(new\_values AS bro\_admin.t\_echipament).id\_filiala = '

|| v\_id\_filiala

|| ')';

end if;

end audit\_echipament\_policy;

/

begin

dbms\_rls.add\_policy(

object\_schema => 'bro\_admin',

object\_name => 'audit\_echipament',

policy\_name => 'AUDIT\_MANAGER\_ECHIPAMENT\_POLICY',

function\_schema => 'SYS',

policy\_function => 'audit\_echipament\_policy',

statement\_types => 'SELECT',

update\_check => true,

enable => true

);

end;

/

-- begin

-- dbms\_rls.drop\_policy(

-- object\_schema => 'BRO\_ADMIN',

-- object\_name => 'audit\_echipament',

-- policy\_name => 'AUDIT\_MANAGER\_ECHIPAMENT\_POLICY'

-- );

-- end;

-- /

SELECT \*

FROM dba\_policies

where object\_owner like upper('bro%');

**10.7.5. sys\_mask.sql**

create or replace directory mask\_dump as 'D:\OracleEE\install\admin\orcl\maskdump';

grant read,write on directory mask\_dump to bro\_admin;

--pt import ca sa nu avem coliziuni la import

create user bro\_import identified by bro\_import;

grant

create session

to bro\_import;

grant

create table,

create sequence

to bro\_import;

alter user bro\_import

quota 20M on users;

grant datapump\_imp\_full\_database to bro\_import;

**10.7.6. sys\_users\_1.sql**

-- Conexiune sys pentru crearea utilizatorilor

ALTER SESSION SET container = "orclpdb";

alter database orclpdb open;

-- Functie ce asigura ca parolele contin cel putin un \_

-- Trebuie sa fie standalone pt ca oracle nu vrea din pachet

CREATE OR REPLACE FUNCTION password\_verify\_function\_standalone (

username VARCHAR2,

new\_password VARCHAR2,

old\_password VARCHAR2

) RETURN BOOLEAN IS

BEGIN

IF instr(

new\_password,

'\_'

) = 0 THEN

raise\_application\_error(

-20003,

'Password must contain at least one underscore (\_) character.'

);

ELSE

RETURN FALSE;

END IF;

END;

/

--Pachet utiliztar pentru gestiunea utilizatorilor

CREATE OR REPLACE PACKAGE bro\_user\_utils IS

antrenor\_suffix CONSTANT VARCHAR2(40) := 'ANTRENOR';

receptionist\_suffix CONSTANT VARCHAR2(40) := 'RECEPTIONIST';

manager\_suffix CONSTANT VARCHAR2(40) := 'MANAGER\_FILIALA';

client\_suffix CONSTANT VARCHAR2(40) := 'CLIENT';

public\_suffix CONSTANT VARCHAR2(40) := 'PUBLIC\_GENERAL';

admin\_suffix CONSTANT VARCHAR2(40) := 'ADMIN';

bro\_tablespace CONSTANT VARCHAR2(40) := 'USERS';

antrenor\_quota CONSTANT VARCHAR2(40) := '10M';

bro\_profile\_public\_general CONSTANT VARCHAR2(40) := 'BRO\_PROFILE\_PUBLIC\_GENERAL';

bro\_profile\_antrenor CONSTANT VARCHAR2(40) := 'BRO\_PROFILE\_ANTRENOR';

bro\_profile\_receptionist CONSTANT VARCHAR2(40) := 'BRO\_PROFILE\_RECEPTIONIST';

bro\_profile\_manager CONSTANT VARCHAR2(40) := 'BRO\_PROFILE\_MANAGER';

bro\_profile\_client CONSTANT VARCHAR2(40) := 'BRO\_PROFILE\_CLIENT';

bro\_plan CONSTANT VARCHAR2(10) := 'P\_BRO';

bro\_role CONSTANT VARCHAR2(10) := 'R\_BRO\_';

TYPE user\_names IS

TABLE OF VARCHAR2(128) INDEX BY PLS\_INTEGER;

TYPE suffix\_names IS

TABLE OF VARCHAR2(40) INDEX BY PLS\_INTEGER;

FUNCTION get\_suffixes RETURN suffix\_names;

FUNCTION get\_users (

suffix VARCHAR2

) RETURN user\_names;

PROCEDURE create\_user\_by\_suffix (

suffix VARCHAR2,

password\_expire BOOLEAN := FALSE

);

PROCEDURE create\_user (

user\_name VARCHAR2,

password\_expire BOOLEAN := FALSE

);

PROCEDURE alter\_quota (

user\_name VARCHAR2

);

PROCEDURE alter\_quota\_all (

suffix VARCHAR2

);

FUNCTION password\_verify\_function (

username VARCHAR2,

new\_password VARCHAR2,

old\_password VARCHAR2

) RETURN BOOLEAN;

PROCEDURE create\_profile (

suffix VARCHAR2

);

PROCEDURE assign\_profile (

user\_name VARCHAR2

);

PROCEDURE assign\_profile\_all (

suffix VARCHAR2

);

PROCEDURE configure\_user\_by\_suffix (

suffix VARCHAR2,

create\_user BOOLEAN := TRUE

);

PROCEDURE create\_bro\_plan\_rg;

PROCEDURE clear\_bro\_plan\_rg;

PROCEDURE create\_role (

suffix VARCHAR2,

ovveride BOOLEAN := FALSE

);

PROCEDURE assign\_role (

suffix VARCHAR2

);

END bro\_user\_utils;

/

CREATE OR REPLACE PACKAGE BODY bro\_user\_utils IS

PROCEDURE is\_valid\_suffix (

suffix VARCHAR2

) IS

BEGIN

IF suffix NOT IN ( antrenor\_suffix,

receptionist\_suffix,

manager\_suffix,

client\_suffix,

public\_suffix ) THEN

raise\_application\_error(

-20002,

'Invalid SUFFIX provided. Must be one of: ANTRENOR, RECEPTIONIST, MANAGER\_FILIALA, CLIENT, PUBLIC\_GENERAL.'

);

END IF;

END;

FUNCTION is\_valid\_user\_name (

user\_name VARCHAR2

) RETURN VARCHAR2 IS

v\_suffix VARCHAR2(40);

BEGIN

IF NOT regexp\_like(

lower(user\_name),

'^bro\_[a-z0-9\_]+$',

'i'

) THEN

raise\_application\_error(

-20005,

'Invalid user name provided. Must start with "bro\_" and contain only letters, numbers, and underscores (case-insensitive).'

);

END IF;

v\_suffix := regexp\_replace(

upper(substr(

user\_name,

5

)),

'[0-9]',

''

);

is\_valid\_suffix(v\_suffix);

RETURN v\_suffix;

END;

FUNCTION get\_profile\_name (

suffix VARCHAR2

) RETURN VARCHAR2 IS

v\_profile\_name VARCHAR2(40);

BEGIN

is\_valid\_suffix(suffix);

CASE

WHEN suffix = antrenor\_suffix THEN

v\_profile\_name := bro\_profile\_antrenor;

WHEN suffix = receptionist\_suffix THEN

v\_profile\_name := bro\_profile\_receptionist;

WHEN suffix = manager\_suffix THEN

v\_profile\_name := bro\_profile\_manager;

WHEN suffix = client\_suffix THEN

v\_profile\_name := bro\_profile\_client;

WHEN suffix = public\_suffix THEN

v\_profile\_name := bro\_profile\_public\_general;

END CASE;

RETURN v\_profile\_name;

END get\_profile\_name;

PROCEDURE drop\_profile (

v\_profile\_name VARCHAR2

) IS

v\_cnt INT;

BEGIN

SELECT COUNT(DISTINCT profile)

INTO v\_cnt

FROM dba\_profiles

WHERE profile = upper(v\_profile\_name);

IF v\_cnt > 1 THEN

raise\_application\_error(

-20004,

'Impossible situation occured'

);

ELSIF v\_cnt = 1 THEN

EXECUTE IMMEDIATE 'DROP PROFILE '

|| v\_profile\_name

|| ' CASCADE';

END IF;

END;

FUNCTION get\_users (

suffix VARCHAR2

) RETURN user\_names IS

v\_names user\_names;

BEGIN

is\_valid\_suffix(suffix);

SELECT username

BULK COLLECT

INTO v\_names

FROM dba\_users

WHERE username LIKE upper('bro\_'

|| suffix

|| '%');

RETURN v\_names;

EXCEPTION

WHEN no\_data\_found THEN

dbms\_output.put\_line('No users found for suffix: ' || suffix);

raise\_application\_error(

-20006,

'No users found for suffix: ' || suffix

);

END;

FUNCTION get\_user\_by\_suffix (

suffix VARCHAR2,

next\_user BOOLEAN := FALSE

) RETURN VARCHAR2 IS

v\_suffix\_cnt INT;

v\_user\_name VARCHAR2(128);

BEGIN

SELECT COUNT(\*)

INTO v\_suffix\_cnt

FROM dba\_users

WHERE username LIKE upper('bro\_'

|| suffix

|| '%');

IF next\_user THEN

v\_suffix\_cnt := v\_suffix\_cnt + 1;

END IF;

dbms\_output.put\_line('V\_SUFFIX\_CNT: ' || v\_suffix\_cnt);

v\_user\_name := upper('bro\_'

|| suffix

|| v\_suffix\_cnt);

RETURN v\_user\_name;

END;

--public

FUNCTION get\_suffixes RETURN suffix\_names IS

v\_suffixes suffix\_names;

BEGIN

v\_suffixes(1) := antrenor\_suffix;

v\_suffixes(2) := receptionist\_suffix;

v\_suffixes(3) := manager\_suffix;

v\_suffixes(4) := client\_suffix;

v\_suffixes(5) := public\_suffix;

RETURN v\_suffixes;

END;

PROCEDURE create\_user (

user\_name VARCHAR2,

password\_expire BOOLEAN := FALSE

) IS

user\_count SMALLINT;

stmt VARCHAR2(200);

sanitized\_user\_name VARCHAR2(128);

BEGIN

sanitized\_user\_name := dbms\_assert.simple\_sql\_name(user\_name);

SELECT COUNT(\*)

INTO user\_count

FROM dba\_users

WHERE username = upper(sanitized\_user\_name);

IF user\_count > 1 THEN

raise\_application\_error(

-20001,

'Impossible situation occured'

);

ELSIF user\_count = 1 THEN

EXECUTE IMMEDIATE 'DROP USER "'

|| upper(sanitized\_user\_name)

|| '" CASCADE';

ELSE

EXECUTE IMMEDIATE 'CREATE USER "'

|| upper(sanitized\_user\_name)

|| '" IDENTIFIED BY "'

|| lower(sanitized\_user\_name)

|| '"';

IF password\_expire THEN

EXECUTE IMMEDIATE 'ALTER USER "'

|| upper(sanitized\_user\_name)

|| '" PASSWORD EXPIRE';

END IF;

EXECUTE IMMEDIATE 'GRANT CREATE SESSION TO "'

|| upper(sanitized\_user\_name)

|| '"';

END IF;

COMMIT;

END;

PROCEDURE create\_user\_by\_suffix (

suffix VARCHAR2,

password\_expire BOOLEAN := FALSE

) IS

v\_user\_name VARCHAR2(128);

BEGIN

v\_user\_name := get\_user\_by\_suffix(

suffix,

TRUE

);

create\_user(

v\_user\_name,

password\_expire

);

END;

PROCEDURE alter\_quota (

user\_name VARCHAR2

) IS

v\_suffix VARCHAR2(40);

v\_quota VARCHAR2(40);

BEGIN

v\_suffix := is\_valid\_user\_name(user\_name);

IF v\_suffix = antrenor\_suffix THEN

v\_quota := antrenor\_quota;

ELSE

v\_quota := '0M';

END IF;

EXECUTE IMMEDIATE 'ALTER USER '

|| user\_name

|| ' QUOTA '

|| v\_quota

|| ' ON '

|| bro\_tablespace;

EXECUTE IMMEDIATE 'ALTER USER '

|| user\_name

|| ' DEFAULT TABLESPACE '

|| bro\_tablespace;

COMMIT;

END;

PROCEDURE alter\_quota\_all (

suffix VARCHAR2

) IS

v\_names user\_names;

v\_quota VARCHAR2(40);

BEGIN

v\_names := get\_users(suffix);

dbms\_output.put\_line('V\_NAMES.COUNT: ' || v\_names.count);

FOR i IN 1..v\_names.count LOOP

alter\_quota(v\_names(i));

END LOOP;

COMMIT;

END;

FUNCTION password\_verify\_function (

username VARCHAR2,

new\_password VARCHAR2,

old\_password VARCHAR2

) RETURN BOOLEAN IS

BEGIN

IF instr(

new\_password,

'\_'

) = 0 THEN

raise\_application\_error(

-20003,

'Password must contain at least one underscore (\_) character.'

);

ELSE

RETURN FALSE;

END IF;

END;

PROCEDURE create\_profile (

suffix VARCHAR2

) IS

v\_cnt SMALLINT;

v\_profile\_name VARCHAR2(40);

v\_password\_verify\_function CONSTANT VARCHAR2(80) := ' PASSWORD\_VERIFY\_FUNCTION PASSWORD\_VERIFY\_FUNCTION\_STANDALONE';

BEGIN

v\_profile\_name := get\_profile\_name(suffix);

drop\_profile(v\_profile\_name);

CASE v\_profile\_name

WHEN bro\_profile\_public\_general THEN

EXECUTE IMMEDIATE 'CREATE PROFILE '

|| v\_profile\_name

|| ' LIMIT SESSIONS\_PER\_USER 6 IDLE\_TIME 5 CONNECT\_TIME 20 CPU\_PER\_CALL 6000 ';

ELSE

EXECUTE IMMEDIATE 'CREATE PROFILE '

|| v\_profile\_name

|| ' LIMIT SESSIONS\_PER\_USER 1 IDLE\_TIME 15 PASSWORD\_LIFE\_TIME 90 FAILED\_LOGIN\_ATTEMPTS 5 '

|| 'CPU\_PER\_CALL 12000 '

|| v\_password\_verify\_function;

END CASE;

END;

PROCEDURE assign\_profile (

user\_name VARCHAR2

) IS

v\_suffix VARCHAR2(40);

v\_profile\_name VARCHAR2(40);

BEGIN

v\_suffix := is\_valid\_user\_name(user\_name);

v\_profile\_name := get\_profile\_name(v\_suffix);

EXECUTE IMMEDIATE 'ALTER USER '

|| user\_name

|| ' PROFILE '

|| v\_profile\_name;

END;

PROCEDURE assign\_profile\_all (

suffix VARCHAR2

) IS

v\_names user\_names;

v\_profile\_name VARCHAR2(40);

BEGIN

v\_names := get\_users(suffix);

FOR i IN 1..v\_names.count LOOP

assign\_profile(v\_names(i));

END LOOP;

END;

PROCEDURE configure\_user\_by\_suffix (

suffix VARCHAR2,

create\_user BOOLEAN := TRUE

) IS

v\_user\_name VARCHAR2(128);

BEGIN

IF create\_user THEN

create\_user\_by\_suffix(suffix);

COMMIT;

END IF;

v\_user\_name := get\_user\_by\_suffix(suffix);

dbms\_output.put\_line('V\_USER\_NAME: ' || v\_user\_name);

alter\_quota(v\_user\_name);

assign\_profile(v\_user\_name);

COMMIT;

END;

PROCEDURE create\_bro\_plan\_rg IS

v\_suffixes suffix\_names := get\_suffixes;

v\_user\_names user\_names;

v\_exists\_default\_group SMALLINT;

v\_rg\_prefix VARCHAR2(40) := 'BRO\_RG\_';

BEGIN

v\_suffixes(v\_suffixes.count + 1) := admin\_suffix;

dbms\_resource\_manager.create\_pending\_area();

dbms\_resource\_manager.create\_plan(

plan => bro\_plan,

comment => 'Consumption plan for BRO users',

active\_sess\_pool\_mth => 'ACTIVE\_SESS\_POOL\_ABSOLUTE',

parallel\_degree\_limit\_mth => 'PARALLEL\_DEGREE\_LIMIT\_ABSOLUTE',

queueing\_mth => 'FIFO\_TIMEOUT',

mgmt\_mth => 'EMPHASIS',

sub\_plan => FALSE

);

FOR i IN 1..v\_suffixes.count LOOP

dbms\_resource\_manager.create\_consumer\_group(

consumer\_group => v\_rg\_prefix || v\_suffixes(i),

comment => 'Consumer group for BRO '

|| v\_suffixes(i)

|| ' users'

);

END LOOP;

SELECT COUNT(\*)

INTO v\_exists\_default\_group

FROM dba\_rsrc\_consumer\_groups

WHERE consumer\_group = 'OTHER\_GROUPS';

IF v\_exists\_default\_group = 0 THEN

dbms\_resource\_manager.create\_consumer\_group(

consumer\_group => 'OTHER\_GROUPS',

comment => 'Default consumer group for all other users'

);

END IF;

FOR i IN 1..v\_suffixes.count LOOP

IF ( v\_suffixes(i) = admin\_suffix ) THEN

dbms\_resource\_manager.set\_consumer\_group\_mapping(

attribute => dbms\_resource\_manager.oracle\_user,

value => 'BRO\_ADMIN',

consumer\_group => v\_rg\_prefix || v\_suffixes(i)

);

ELSE

v\_user\_names := get\_users(v\_suffixes(i));

dbms\_output.put\_line('V\_USER\_NAMES.COUNT: ' || v\_user\_names.count);

FOR j IN 1..v\_user\_names.count LOOP

dbms\_resource\_manager.set\_consumer\_group\_mapping(

attribute => dbms\_resource\_manager.oracle\_user,

value => v\_user\_names(j),

consumer\_group => v\_rg\_prefix || v\_suffixes(i)

);

END LOOP;

END IF;

END LOOP;

-- doar mgmt\_p1 ca nu avem subplanuri

dbms\_resource\_manager.create\_plan\_directive(

plan => bro\_plan,

group\_or\_subplan => 'OTHER\_GROUPS',

comment => 'Default consumer group for all other users',

mgmt\_p1 => 5

);

dbms\_resource\_manager.create\_plan\_directive(

plan => bro\_plan,

group\_or\_subplan => v\_rg\_prefix || antrenor\_suffix,

comment => 'Consumer group for BRO ANTRENOR users',

mgmt\_p1 => 20

);

dbms\_resource\_manager.create\_plan\_directive(

plan => bro\_plan,

group\_or\_subplan => v\_rg\_prefix || receptionist\_suffix,

comment => 'Consumer group for BRO RECEPTIONIST users',

mgmt\_p1 => 15

);

dbms\_resource\_manager.create\_plan\_directive(

plan => bro\_plan,

group\_or\_subplan => v\_rg\_prefix || manager\_suffix,

comment => 'Consumer group for BRO MANAGER users',

mgmt\_p1 => 15

);

dbms\_resource\_manager.create\_plan\_directive(

plan => bro\_plan,

group\_or\_subplan => v\_rg\_prefix || client\_suffix,

comment => 'Consumer group for BRO CLIENT users',

mgmt\_p1 => 10

);

dbms\_resource\_manager.create\_plan\_directive(

plan => bro\_plan,

group\_or\_subplan => v\_rg\_prefix || public\_suffix,

comment => 'Consumer group for BRO PUBLIC users',

mgmt\_p1 => 5

);

dbms\_resource\_manager.create\_plan\_directive(

plan => bro\_plan,

group\_or\_subplan => v\_rg\_prefix || admin\_suffix,

comment => 'Consumer group for BRO ADMIN users',

mgmt\_p1 => 30

);

dbms\_resource\_manager.validate\_pending\_area();

dbms\_resource\_manager.submit\_pending\_area();

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Error: ' || sqlerrm);

dbms\_resource\_manager.clear\_pending\_area();

COMMIT;

END;

PROCEDURE clear\_bro\_plan\_rg IS

v\_suffixes suffix\_names := get\_suffixes;

v\_exists\_default\_group SMALLINT;

v\_rg\_prefix VARCHAR2(40) := 'BRO\_RG\_';

BEGIN

v\_suffixes(v\_suffixes.count + 1) := admin\_suffix;

dbms\_resource\_manager.create\_pending\_area();

BEGIN

dbms\_resource\_manager.delete\_plan(plan => bro\_plan);

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Error deleting plan: ' || sqlerrm);

END;

FOR i IN 1..v\_suffixes.count LOOP

BEGIN

dbms\_resource\_manager.delete\_consumer\_group(consumer\_group => v\_rg\_prefix || v\_suffixes(i));

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Error deleting consumer group: '

|| v\_rg\_prefix

|| v\_suffixes(i)

|| ' - '

|| sqlerrm);

END;

END LOOP;

SELECT COUNT(\*)

INTO v\_exists\_default\_group

FROM dba\_rsrc\_consumer\_groups

WHERE consumer\_group = 'OTHER\_GROUPS';

IF v\_exists\_default\_group > 0 THEN

BEGIN

dbms\_resource\_manager.delete\_consumer\_group(consumer\_group => 'OTHER\_GROUPS');

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Error deleting OTHER\_GROUPS: ' || sqlerrm);

END;

END IF;

dbms\_resource\_manager.validate\_pending\_area();

dbms\_resource\_manager.submit\_pending\_area();

dbms\_output.put\_line('All objects created by CREATE\_BRO\_PLAN\_RG have been cleared.');

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Error: ' || sqlerrm);

dbms\_resource\_manager.clear\_pending\_area();

COMMIT;

END;

PROCEDURE create\_role (

suffix VARCHAR2,

ovveride BOOLEAN := FALSE

) IS

v\_cnt SMALLINT;

v\_role\_name VARCHAR2(40);

BEGIN

is\_valid\_suffix(suffix);

v\_role\_name := dbms\_assert.simple\_sql\_name(upper(bro\_role || suffix));

SELECT COUNT(\*)

INTO v\_cnt

FROM dba\_roles

WHERE role = upper(v\_role\_name);

dbms\_output.put\_line('NOT IMPLEMENTED');

END;

PROCEDURE assign\_role (

suffix VARCHAR2

) IS

v\_role\_name VARCHAR2(40);

v\_users user\_names;

v\_cnt SMALLINT;

BEGIN

v\_role\_name := dbms\_assert.simple\_sql\_name(upper(bro\_role || suffix));

v\_users := get\_users(suffix);

SELECT COUNT(\*)

INTO v\_cnt

FROM dba\_roles

WHERE role = upper(v\_role\_name);

dbms\_output.put\_line('NOT IMPLEMENTED');

END;

END bro\_user\_utils;

/

-- Table si functie care intoarce toti utilizaotrii creati

-- pentru a permite adimnului access doar la cei

-- din cadrul aplicatiei noastre

CREATE OR REPLACE TYPE global\_user\_table AS

TABLE OF VARCHAR2(128);

/

CREATE OR REPLACE FUNCTION get\_users\_by\_suffix (

suffix VARCHAR2

) RETURN global\_user\_table IS

result\_cursor SYS\_REFCURSOR;

user\_indexed\_table bro\_user\_utils.user\_names;

user\_nested\_table global\_user\_table := global\_user\_table();

BEGIN

user\_indexed\_table := bro\_user\_utils.get\_users(suffix);

FOR i IN user\_indexed\_table.first..user\_indexed\_table.last LOOP

user\_nested\_table.extend;

user\_nested\_table(user\_nested\_table.count) := user\_indexed\_table(i);

END LOOP;

RETURN user\_nested\_table;

END;

/

CREATE USER bro\_admin IDENTIFIED BY bro\_admin

PASSWORD EXPIRE;

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;

GRANT EXECUTE ON global\_user\_table 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;

--Profilul adminului

CREATE PROFILE bro\_profile\_admin LIMIT

IDLE\_TIME 15

PASSWORD\_LIFE\_TIME 90

FAILED\_LOGIN\_ATTEMPTS 5

CPU\_PER\_CALL 36000

PASSWORD\_VERIFY\_FUNCTION password\_verify\_function\_standalone;

ALTER USER bro\_admin

PROFILE bro\_profile\_admin;

ALTER USER bro\_admin

QUOTA 500M ON users;

-- Creare profilurilor pentru restul de utilizatori

exec BRO\_USER\_UTILS.CREATE\_PROFILE(BRO\_USER\_UTILS.PUBLIC\_SUFFIX);

exec BRO\_USER\_UTILS.CREATE\_PROFILE(BRO\_USER\_UTILS.ANTRENOR\_SUFFIX);

exec BRO\_USER\_UTILS.CREATE\_PROFILE(BRO\_USER\_UTILS.RECEPTIONIST\_SUFFIX);

exec BRO\_USER\_UTILS.CREATE\_PROFILE(BRO\_USER\_UTILS.MANAGER\_SUFFIX);

exec BRO\_USER\_UTILS.CREATE\_PROFILE(BRO\_USER\_UTILS.CLIENT\_SUFFIX);

-- public general

exec BRO\_USER\_UTILS.CONFIGURE\_USER\_BY\_SUFFIX(BRO\_USER\_UTILS.PUBLIC\_SUFFIX, TRUE);

--manager filiala

exec BRO\_USER\_UTILS.CONFIGURE\_USER\_BY\_SUFFIX(BRO\_USER\_UTILS.MANAGER\_SUFFIX, TRUE);

--antrenor

BEGIN

FOR i IN 1..7 LOOP

bro\_user\_utils.configure\_user\_by\_suffix(

bro\_user\_utils.antrenor\_suffix,

TRUE

);

END LOOP;

END;

/

--receptionist

BEGIN

FOR i IN 1..10 LOOP

bro\_user\_utils.configure\_user\_by\_suffix(

bro\_user\_utils.receptionist\_suffix,

TRUE

);

END LOOP;

END;

/

--client

BEGIN

FOR i IN 1..10 LOOP

bro\_user\_utils.configure\_user\_by\_suffix(

bro\_user\_utils.client\_suffix,

TRUE

);

END LOOP;

END;

/

-- planul de resurse

exec BRO\_USER\_UTILS.CREATE\_BRO\_PLAN\_RG;

SELECT username,

initial\_rsrc\_consumer\_group

FROM dba\_users

WHERE username LIKE upper('bro\_%');

SELECT DISTINCT u.username, u.profile, p.group\_or\_subplan, p.mgmt\_p1, p.plan

FROM dba\_users u JOIN dba\_rsrc\_plan\_directives p

ON u.initial\_rsrc\_consumer\_group=p.group\_or\_subplan

WHERE username LIKE upper('bro\_%');

SELECT DISTINCT

u.username,

u.profile,

p.group\_or\_subplan,

p.mgmt\_p1,

p.plan,

r.granted\_role

FROM

dba\_users u

LEFT JOIN dba\_rsrc\_plan\_directives p

ON u.initial\_rsrc\_consumer\_group = p.group\_or\_subplan

LEFT JOIN dba\_role\_privs r

ON u.username = r.grantee

WHERE

u.username LIKE UPPER('bro\_%')

ORDER BY

u.username, r.granted\_role;

**10.7.7 sys\_users\_2.sql**

---SYS

alter session set container = orclpdb;

-- Dupa ce bro\_admin a create tabele si a si inserat datele

-- Dupa ce bro\_admin a creat tabele si a si inserat datele pt bro\_antrenor1..n

create role r\_bro\_public\_general;

grant

create session

to r\_bro\_public\_general;

grant select on bro\_admin.antrenor\_extins to r\_bro\_public\_general;

grant select on bro\_admin.filiala to r\_bro\_public\_general;

grant select on bro\_admin.adresa to r\_bro\_public\_general;

grant select on bro\_admin.supliment to r\_bro\_public\_general;

grant select on bro\_admin.echipament to r\_bro\_public\_general;

grant select on bro\_antrenor1.program to r\_bro\_public\_general;

grant select on bro\_antrenor2.program to r\_bro\_public\_general;

grant select on bro\_antrenor3.program to r\_bro\_public\_general;

grant r\_bro\_public\_general to bro\_public\_general1;

-- roluri

-- antrenor

create role r\_bro\_antrenor;

grant r\_bro\_public\_general to r\_bro\_antrenor;

grant select on bro\_admin.client\_extins to r\_bro\_antrenor;

grant select on bro\_admin.telefon to r\_bro\_antrenor;

grant select on bro\_admin.antrenor to r\_bro\_antrenor;

grant select on bro\_admin.chei\_client to r\_bro\_antrenor;

grant execute on bro\_admin.select\_random\_from\_nr\_list to r\_bro\_antrenor;

grant

create table,

create view,

create sequence,

create procedure,

create type

to r\_bro\_antrenor;

grant execute on dbms\_crypto to r\_bro\_antrenor;

grant r\_bro\_antrenor to bro\_antrenor1;

grant r\_bro\_antrenor to bro\_antrenor2;

grant r\_bro\_antrenor to bro\_antrenor3;

-- client

create role r\_bro\_client;

grant r\_bro\_public\_general to r\_bro\_client;

grant select on bro\_admin.client\_extins to r\_bro\_client;

grant execute on bro\_admin.get\_client\_key to r\_bro\_client;

-- fiecare antrenor da la clientii sai

grant execute on bro\_antrenor1.fetch\_decrypted\_client\_data to bro\_client1;

grant execute on bro\_antrenor1.hash\_checksum to bro\_client1;

grant execute on bro\_antrenor1.fetch\_decrypted\_client\_data to bro\_client2;

grant execute on bro\_antrenor1.hash\_checksum to bro\_client2;

grant execute on bro\_antrenor1.fetch\_decrypted\_client\_data to bro\_client3;

grant execute on bro\_antrenor1.hash\_checksum to bro\_client3;

--programul e public

grant r\_bro\_client to bro\_client1;

grant r\_bro\_client to bro\_client2;

grant r\_bro\_client to bro\_client3;

--receptionist

create role r\_bro\_receptionist;

grant r\_bro\_public\_general to r\_bro\_receptionist;

grant select,insert,update on bro\_admin.client\_extins to r\_bro\_receptionist;

grant select,insert,update on bro\_admin.telefon to r\_bro\_receptionist;

grant select,insert,update on bro\_admin.abonament to r\_bro\_receptionist;

grant select on bro\_admin.tip\_abonament to r\_bro\_receptionist;

grant select on bro\_admin.furnizor to r\_bro\_receptionist;

grant insert on bro\_admin.comanda to r\_bro\_receptionist;

grant insert on bro\_admin.informatii\_comanda to r\_bro\_receptionist;

grant select on bro\_admin.aprovizionare to r\_bro\_receptionist;

--programul e public

grant r\_bro\_receptionist to bro\_receptionist1;

grant r\_bro\_receptionist to bro\_receptionist2;

grant r\_bro\_receptionist to bro\_receptionist3;

--manager filiala

create role r\_bro\_manager\_filiala;

grant r\_bro\_public\_general to r\_bro\_manager\_filiala;

grant select on bro\_admin.receptionist\_extins to r\_bro\_manager\_filiala;

grant select on bro\_admin.client\_extins to r\_bro\_manager\_filiala;

grant select on bro\_admin.furnizor to r\_bro\_manager\_filiala;

grant select on bro\_antrenor1.program to r\_bro\_manager\_filiala;

--pt fiecare antrenor in filiala sa

grant select on bro\_antrenor1.antrenament to r\_bro\_manager\_filiala;

grant select on bro\_antrenor2.antrenament to r\_bro\_manager\_filiala;

grant select on bro\_antrenor3.antrenament to r\_bro\_manager\_filiala;

grant select,update,insert,delete on bro\_admin.echipament to r\_bro\_manager\_filiala;

grant select on bro\_admin.aprovizionare to r\_bro\_manager\_filiala;

grant r\_bro\_manager\_filiala to bro\_manager\_filiala1;

1. **Codul CMD**

**11.1. import\_mask\_person.cmd**

@echo off

impdp bro\_import/bro\_import@//localhost:1522/orclpdb ^

remap\_table=persoana:persoana\_mask ^

remap\_table=angajat:angajat\_mask ^

remap\_table=antrenor:antrenor\_mask ^

remap\_table=receptionist:receptionist\_mask ^

remap\_table=client:client\_mask ^

remap\_schema=bro\_admin:bro\_import ^

directory=MASK\_DUMP ^

dumpfile=mask\_person.dmp ^

logfile=mask\_person\_import.log ^

parallel=8 ^

transform=disable\_archive\_logging:y

echo Done importing mask persoana

exit /b 0

**11.2. mask\_person.cmd**

@echo off

expdp bro\_admin/bro\_admin@//localhost:1522/orclpdb ^

tables=BRO\_ADMIN.PERSOANA, BRO\_ADMIN.ANGAJAT, BRO\_ADMIN.ANTRENOR, BRO\_ADMIN.RECEPTIONIST, BRO\_ADMIN.CLIENT ^

remap\_data=persoana.id\_persoana:mask\_person.mask\_person\_id ^

remap\_data=persoana.nume:mask\_person.mask\_item ^

remap\_data=persoana.prenume:mask\_person.mask\_item ^

remap\_data=persoana.email:mask\_person.mask\_item ^

remap\_data=persoana.varsta:mask\_person.mask\_item ^

remap\_data=angajat.id\_angajat:mask\_person.mask\_person\_fk ^

remap\_data=angajat.salariu:mask\_person.mask\_item ^

remap\_data=angajat.id\_meneger:mask\_person.mask\_person\_fk ^

remap\_data=antrenor.id\_antrenor:mask\_person.mask\_person\_fk ^

remap\_data=receptionist.id\_receptionist:mask\_person.mask\_person\_fk ^

remap\_data=client.id\_client:mask\_person.mask\_person\_fk ^

directory=MASK\_DUMP parallel=8 dumpfile=mask\_person.dmp logfile=mask\_person.log reuse\_dumpfiles=y

echo Done exporting mask persoana

exit /b 0

**11.3. seed\_antrenor.cmd**

@echo off

set "SQL\_SCRIPT\_PATH=C:\Master\An2\sem1\securitateaBD\proiect\sql\bro\_admin\_antrenor\_seed.sql"

sqlplus bro\_admin/bro\_admin@//localhost:1522/orclpdb @%SQL\_SCRIPT\_PATH%

echo Done seeding antrenor table

exit /b 0

1. **Link repository**

Proiectul de posate găsi pe github: <https://github.com/MocicaRazvan/sbdProiect>