Baza de date pentru Disney +

Iștoc Simona Maria

Seria 23

Grupa 231

Anul Universitar 2024-2025

Contents

[**Intoroducere** 3](#_Toc187415111)

[**Exercițiul 1** - Prezentați pe scurt baza de date (utilitatea ei). 4](#_Toc187415112)

[**Exercițiul 2** - Realizați diagrama entitate-relație (ERD): entitățile, relațiile și atributele trebuie definite în limba română 7](#_Toc187415113)

[**Exercițiul 3** - Pornind de la diagrama entitate-relație realizați diagrama conceptuală a modelului propus, integrând toate atributele necesare: entitățile, relațiile și atributele trebuie definite în limba română. 8](#_Toc187415114)

[**Exercițiul 4** - Implementați în Oracle diagrama conceptuală realizată: definiți toate tabelele, adăugând toate constrângerile de integritate necesare (chei primare, cheile externe etc). 9](#_Toc187415115)

[**Exercițiul 5** - Adăugați informații coerente în tabelele create (minim 5 înregistrări pentru fiecare entitate independentă; minim 10 înregistrări pentru fiecare tabelă asociativă). 12](#_Toc187415116)

[**Exercițiul 6** - Formulați în limbaj natural o problemă pe care să o rezolvați folosind un subprogram stocat independent care să utilizeze toate cele 3 tipuri de colecții studiate. Apelați subprogramul. 20](#_Toc187415117)

[**Exercițiul 7** - Formulați în limbaj natural o problemă pe care să o rezolvați folosind un subprogram stocat independent care să utilizeze 2 tipuri diferite de cursoare studiate, unul dintre acestea fiind cursor parametrizat, dependent de celălalt cursor. Apelați subprogramul. 23](#_Toc187415118)

[**Exercițiul 8** - Formulați în limbaj natural o problemă pe care să o rezolvați folosind un subprogram stocat independent de tip funcție care să utilizeze într-o singură comandă SQL 3 dintre tabelele create. Tratați toate excepțiile care pot apărea, incluzând excepțiile predefinite NO\_DATA\_FOUND și TOO\_MANY\_ROWS. Apelați subprogramul astfel încât să evidențiați toate cazurile tratate 25](#_Toc187415119)

[**Exercițiul 9** - Formulați în limbaj natural o problemă pe care să o rezolvați folosind un subprogram stocat independent de tip procedură care să aibă minim 2 parametri și să utilizeze într-o singură comandă SQL 5 dintre tabelele create. Definiți minim 2 excepții proprii, altele decât cele predefinite la nivel de sistem. Apelați subprogramul astfel încât să evidențiați toate cazurile definite și tratate. 28](#_Toc187415120)

[**Exercițiul 10** - Definiți un trigger de tip LMD la nivel de comandă. Declanșați trigger-ul 32](#_Toc187415121)

[**Exercițiul 11** - Definiți un trigger de tip LMD la nivel de linie. Declanșați trigger-ul. 34](#_Toc187415122)

[**Exercițiul 12** - Definiți un trigger de tip LDD. Declanșați trigger-ul. 36](#_Toc187415123)

[**Exercițiul 13** - Formulați în limbaj natural o problemă pe care să o rezolvați folosind un pachet care să includă tipuri de date complexe și obiecte necesare unui flux de acțiuni integrate, specifice bazei de date definite (minim 2 tipuri de date, minim 2 funcții, minim 2 proceduri). 37](#_Toc187415124)

# **Intoroducere**

Tema aleasă pentru proiect este gestionarea bazei de date pentru platforma de streaming Disney Plus. Versiunea de SGBD folosită este Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production. Configurația hardware-ului est: System Global Area – 2416 MB și Program Global Area – 801 MB . Configurația software-ului este: sistemul de operare este Microsoft Windows x86 64-bit și nu s-a folosit mașina virtuală.

# **Exercițiul 1** - Prezentați pe scurt baza de date (utilitatea ei).

Disney + este un serviciu de streaming deținut de Walt Disney Company care conține numeroase filme și seriale produse de această companie precum și alte branduri cum ar fi Pixar, Marvel, Star Wars, National Geographic.

Baza de date este utilă deoarece ajută la gestionarea platformei: gestionarea conținului media (filme și seriale) dar și a directorilor care le-au produs cât și actorii care au jucat cât și rolurile acestora, gestionarea utilizatorului cu toate profilele acestuia, management-ul plăților și abonamentelor.

Un utilizator își poate face un singur abonament și trebuie să aleagă una din metodele de plată (pe an, pe luna). Un utilizator poate urmări orice film sau serial de pe platformă atât timp cât are un abonament și acesta este cu plata la zi.

Fiecare film/serial are un director și unul sau mai mulți actori. Fiecare actor joacă în unul sau mai multe filme/seriale. Fiecare actor are un rol într-un film/serial, iar fiecare film/serial are unul sau mai multe roluri.

Descrierea entităților, împreună cu cheile primare:

UTILIZATOR = persoană fizică care își poate crea un cont pe baza unui email. Entitatea

depinde de plată și abonament. Cheie primara este utilizator\_id.

PLATA = entitate în care ținem informațiile cardului cu care plătește utilizatorului. Cheia

primară este plată\_id.

ABONAMENT = pachet care conține filmele și serialele disponibile pe Disney+. Cheia

primară este abonament\_id.

PROFIL = metodă de partajare a unui abonament între mai multe persoane. Entitatea

depinde de abonament\_id. Cheia primară este profil\_id.

PRODUCȚIE = colecție de conținut video creat pentru divertisment conținând filme și

seriale. Cheia primară este producție\_id.

FILM = producție cinematografică care spun o poveste completă într-o singură lucrare

separate pe mai multe genuri. Cheia primară este productie\_id.

SERIAL = producții episodice care spun o poveste pe parcursul mai multor episoade și

sezoane cu durata și structură variabilă, de asemenea separate pe mai multe genuri. Cheia

primară este productie\_id.

EPISOD = parte individuală dintr-un serial. Cheia primară este episod\_id

DIRECTOR = persoană fizică responsabilă pentru supravegherea și coordonarea tuturor

aspectelor creative ale unei producții. Cheia primară este director\_id

ACTOR = persoană fizică care interpretează roluri în producții. Cheia primară este

actor\_id

ROL = un personaj pe care un actor îl interpretează într-o producție. Cheia primară este

rol\_id

Descrierea relațiilor (incluzând precizarea cardinalitații):

UTILIZATOR are ABONAMENT

- Un utilizator poate să aleagă un singur abonament

- Un abonament poate sa fie ales de zero sau mai multi utilizatori

- Cardinalitate maximă: M:1

- Cardinalitate minimă: 0:1

UTILIZATOR adaugă o PLATĂ

- Un utilizator poate să adauge o singură plată

- O plată poate fi adaugată de 1 sau mai mulți utlizatori

- Cardinalitate maximă: M:1

- Cardinalitate minima: 1:1

ABONAMENT are un PROFIL

- Un utilizator poate sa aiba 1 sau mai multe profile (maxim 7 profile)

- Un profil este asociat unui singur abonament

- Cardinalitate maxima: 1:1

- Cardinalitate minimă: M:1

PRODUCȚIE are ACTORI

- O producție poate avea zero sau mai multi actori

- Un actor poate juca in zero sau mai multe producții

- Cardinalitate maxima: M:0

- Cardinalitate minimă: M:0

PRODUCȚIE are ROLURI

- O producție poate avea zero sau mai multe roluri

- Un rol poate apărea în zero sau mai multe producții

- Cardinalitate maximă: M:M

- Cardinlitate minimă: 0:0

ACTORI are ROLURI

- Un actor poate avea zero sau mai multe roluri

- Un rol poate fi jucat de zero sau mai mulți actori

- Cardinaitate maximă: M:M

- Cardinalitate minimă: 0:0

PROFIL vizionează PRODUCȚIE

- Un profil poate viziona zero sau mai multe producții

- O producție poate fi vizionată de una sau mai multe producții

- Cardinalitate maximă: M:M

- Cardinalitate minimă: 0:0

DIRECTOR regizează PRODUCȚIE

- Un director regizeazi una sau mai multe producții

- O producție este regizată de unul sau mai mulți directori

- Cardinalitate maximă: M:M

- Cardinalitate minimă: 1:1

SERIAL are EPISOD

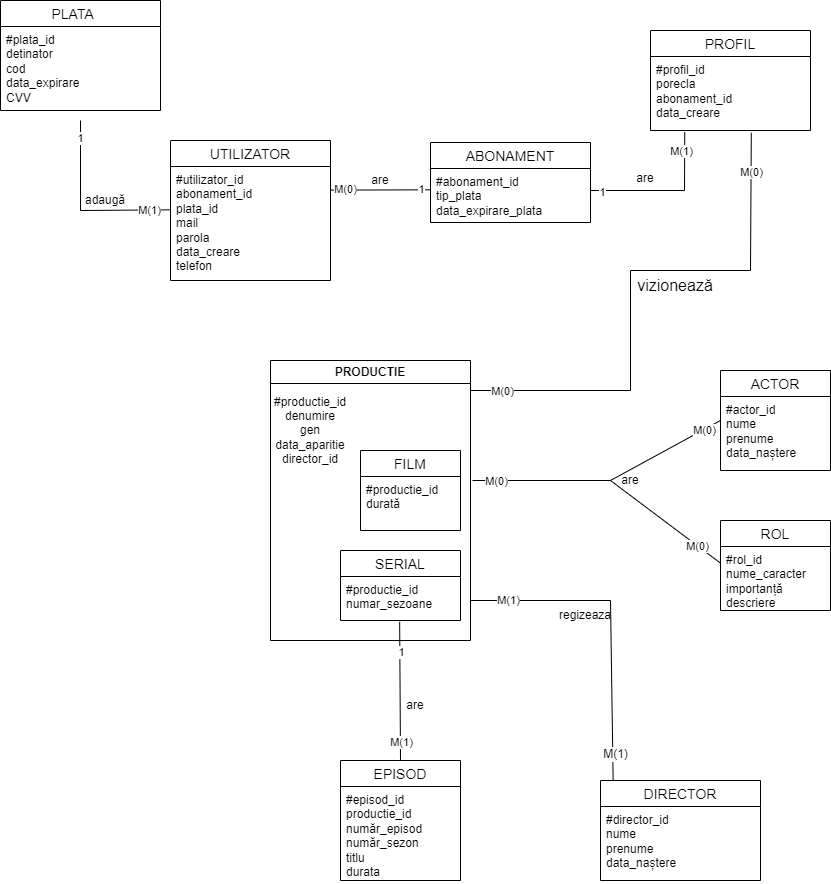
- Un serial are unul sau mai multe episoade

- Un episod face parte dintr-un serial

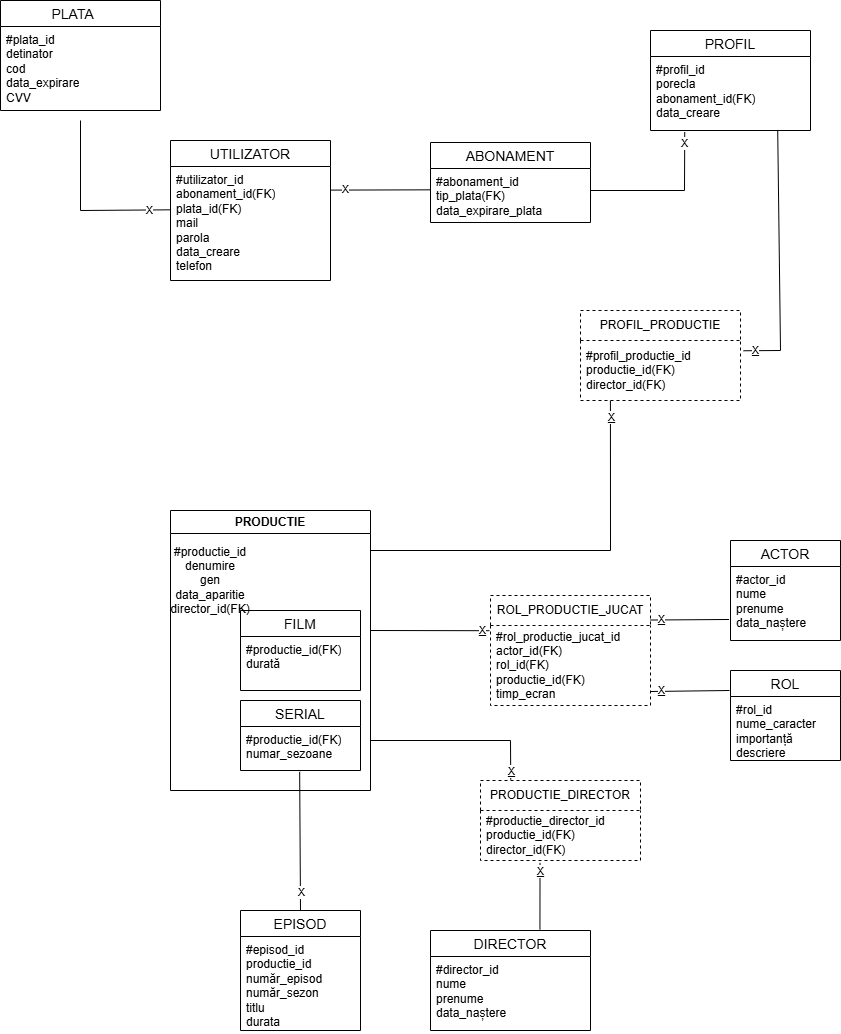
- Cardinalitate maxima: M:1

- Cardinalitate minimă: 1:1

# **Exercițiul 2** - Realizați diagrama entitate-relație (ERD): entitățile, relațiile și atributele trebuie definite în limba română



# **Exercițiul 3** - Pornind de la diagrama entitate-relație realizați diagrama conceptuală a modelului propus, integrând toate atributele necesare: entitățile, relațiile și atributele trebuie definite în limba română.



# **Exercițiul 4** - Implementați în Oracle diagrama conceptuală realizată: definiți toate tabelele, adăugând toate constrângerile de integritate necesare (chei primare, cheile externe etc).

**PLATA**

CREATE TABLE PLATA(

plata\_id NUMBER(6) PRIMARY KEY,

detinator VARCHAR2(50) NOT NULL,

cod NUMBER(16) NOT NULL UNIQUE,

data\_expirare DATE NOT NULL,

CVV NUMBER(3) NOT NULL

);

**ABONAMENT**

CREATE TABLE ABONAMENT(

abonament\_id NUMBER(6) PRIMARY KEY,

tip\_plata VARCHAR(10) NOT NULL,

data\_expirare\_plata DATE NOT NULL

);

**PROFIL**

CREATE TABLE PROFIL(

profil\_id NUMBER(6) PRIMARY KEY,

abonament\_id NUMBER(6) NOT NULL,

porecla VARCHAR2(20) NOT NULL,

data\_creare DATE NOT NULL,

FOREIGN KEY (abonament\_id) REFERENCES ABONAMENT(abonament\_id)

);

**UTILIZATOR**

CREATE TABLE UTILIZATOR(

utilizator\_id NUMBER(6) PRIMARY KEY,

abonament\_id NUMBER(6) NOT NULL,

plata\_id NUMBER(6) NOT NULL,

mail VARCHAR2(50) NOT NULL UNIQUE,

parola VARCHAR(50) NOT NULL,

data\_creare DATE NOT NULL,

telefon VARCHAR2(16),

FOREIGN KEY (abonament\_id) REFERENCES ABONAMENT(abonament\_id),

FOREIGN KEY (plata\_id) REFERENCES PLATA(plata\_id)

);

**DIRECTOR**

CREATE TABLE DIRECTOR(

director\_id NUMBER(5) PRIMARY KEY,

nume VARCHAR2(50),

prenume VARCHAR2(50),

data\_nastere DATE

);

**ACTOR**

CREATE TABLE ACTOR(

actor\_id NUMBER(5) PRIMARY KEY,

nume VARCHAR2(50) NOT NULL,

prenume VARCHAR2(50) NOT NULL,

data\_natere DATE

);

**ROL**

CREATE TABLE ROL(

rol\_id NUMBER(5) PRIMARY KEY,

nume\_caracter VARCHAR2(50) NOT NULL,

importanta VARCHAR2(50) NOT NULL,

descriere VARCHAR2(200)

);

**PRODUCTIE**

CREATE TABLE PRODUCTIE(

productie\_id NUMBER(6) PRIMARY KEY,

denumire VARCHAR2(50) NOT NULL,

gen VARCHAR2(50) NOT NULL,

data\_aparitie DATE NOT NULL,

director\_id NUMBER(5) NOT NULL,

FOREIGN KEY (director\_id) REFERENCES DIRECTOR(director\_id)

);

**FILM**

CREATE TABLE FILM(

productie\_id NUMBER(6) PRIMARY KEY,

durata NUMBER(3) NOT NULL,

FOREIGN KEY(productie\_id) REFERENCES PRODUCTIE(productie\_id)

);

**SERIAL**

CREATE TABLE SERIAL(

productie\_id NUMBER(6) PRIMARY KEY,

numar\_sezoane NUMBER(2) NOT NULL,

FOREIGN KEY(productie\_id) REFERENCES PRODUCTIE(productie\_id)

);

**EPISOD**

CREATE TABLE EPISOD(

episod\_id NUMBER(6) PRIMARY KEY,

productie\_id NUMBER(6) NOT NULL,

numar\_episod NUMBER(2) NOT NULL,

numar\_sezon NUMBER(2) NOT NULL,

titlu VARCHAR2(50) NOT NULL,

durata NUMBER(3) NOT NULL,

FOREIGN KEY(productie\_id) REFERENCES PRODUCTIE(productie\_id)

);

**PROFIL\_PRODUCTIE**

CREATE TABLE PROFIL\_PRODUCTIE(

profil\_productie\_id NUMBER(6) PRIMARY KEY,

profil\_id NUMBER(6) NOT NULL,

productie\_id NUMBER(6) NOT NULL,

FOREIGN KEY(profil\_id) REFERENCES PROFIL(profil\_id),

FOREIGN KEY(productie\_id) REFERENCES PRODUCTIE(productie\_id)

);

**PRODUCTIE\_DIRECTOR**

CREATE TABLE PRODUCTIE\_DIRECTOR(

productie\_director\_id NUMBER(6) PRIMARY KEY,

productie\_id NUMBER(6) NOT NULL,

director\_id NUMBER(5) NOT NULL,

FOREIGN KEY(productie\_id) REFERENCES PRODUCTIE(productie\_id),

FOREIGN KEY(director\_id) REFERENCES DIRECTOR(director\_id)

);

**ROL\_PRODUCTIE\_JUCAT**

CREATE TABLE ROL\_PRODUCTIE\_JUCAT(

rol\_productie\_jucat NUMBER(7) PRIMARY KEY,

actor\_id NUMBER(5) NOT NULL,

rol\_id NUMBER(5) NOT NULL,

productie\_id NUMBER(6) NOT NULL,

timp\_ecran NUMBER(3),

FOREIGN KEY(actor\_id) REFERENCES ACTOR(actor\_id),

FOREIGN KEY(rol\_id) REFERENCES ROL(rol\_id),

FOREIGN KEY(productie\_id) REFERENCES PRODUCTIE(productie\_id)

);

# **Exercițiul 5** - Adăugați informații coerente în tabelele create (minim 5 înregistrări pentru fiecare entitate independentă; minim 10 înregistrări pentru fiecare tabelă asociativă).

**PLATA**

INSERT INTO PLATA VALUES(1, 'JOHN SMITH', 1234567890123456, to\_date('15.1.2024','dd.mm.yyyy'), 123);

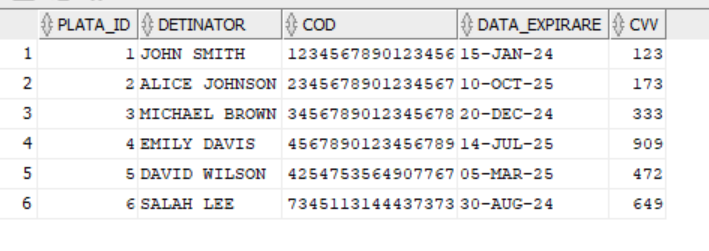
INSERT INTO PLATA VALUES(2, 'ALICE JOHNSON', 2345678901234567, to\_date('10.10.2025','dd.mm.yyyy'), 173);

INSERT INTO PLATA VALUES(3, 'MICHAEL BROWN', 3456789012345678, to\_date('20.12.2024','dd.mm.yyyy'), 333);

INSERT INTO PLATA VALUES(4, 'EMILY DAVIS', 4567890123456789, to\_date('14.7.2025','dd.mm.yyyy'), 909);

INSERT INTO PLATA VALUES(5, 'DAVID WILSON', 4254753564907767, to\_date('5.3.2025','dd.mm.yyyy'), 472);

INSERT INTO PLATA VALUES(6, 'SALAH LEE', 7345113144437373, to\_date('30.8.2024','dd.mm.yyyy'), 649);



**ABONAMENT**

INSERT INTO ABONAMENT VALUES(10, 'lunar', to\_date('13.2.2024', 'dd.mm.yyyy'));

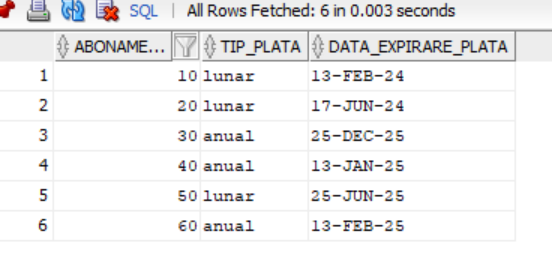
INSERT INTO ABONAMENT VALUES(20, 'lunar', to\_date('17.6.2024', 'dd.mm.yyyy'));

INSERT INTO ABONAMENT VALUES(30, 'anual', to\_date('25.12.2025', 'dd.mm.yyyy'));

INSERT INTO ABONAMENT VALUES(40, 'anual', to\_date('13.1.2025', 'dd.mm.yyyy'));

INSERT INTO ABONAMENT VALUES(50, 'lunar', to\_date('25.6.2025', 'dd.mm.yyyy'));

INSERT INTO ABONAMENT VALUES(60, 'anual', to\_date('13.2.2025', 'dd.mm.yyyy'));



**PROFIL**

INSERT INTO PROFIL VALUES(11, 10, 'Spark', to\_date('13.1.2023', 'dd.mm.yyyy'));

INSERT INTO PROFIL VALUES(22, 10, 'Luna', to\_date('14.2.2024', 'dd.mm.yyyy'));

INSERT INTO PROFIL VALUES(33, 20, 'Rocket', to\_date('16.1.2022', 'dd.mm.yyyy'));

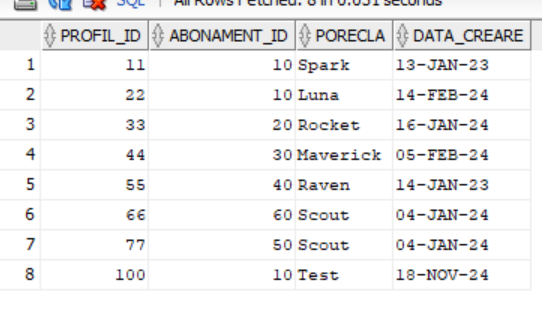
INSERT INTO PROFIL VALUES(44, 30, 'Maverick', to\_date('5.2.2024', 'dd.mm.yyyy'));

INSERT INTO PROFIL VALUES(55, 40, 'Raven', to\_date('14.1.2023', 'dd.mm.yyyy'));

INSERT INTO PROFIL VALUES(66, 60, 'Scout', to\_date('4.1.2024', 'dd.mm.yyyy'));

INSERT INTO PROFIL VALUES(77, 50, 'Scout', to\_date('4.1.2024', 'dd.mm.yyyy'));

INSERT INTO PROFIL VALUES(100, 10, 'Test', to\_date('18.11.2024', 'dd.mm.yyyy'));



**UTILIZATOR**

INSERT INTO UTILIZATOR VALUES(100, 10, 1, 'johnsmith@yahoo.com', 'josmith123', to\_date('13.01.2023', 'dd.mm.yyyy'), '0762599250');

INSERT INTO UTILIZATOR VALUES(101, 20, 2, 'aljohnson@gmail.com', 'johnsonAls', to\_date('16.01.2024', 'dd.mm.yyyy'), '0742911211');

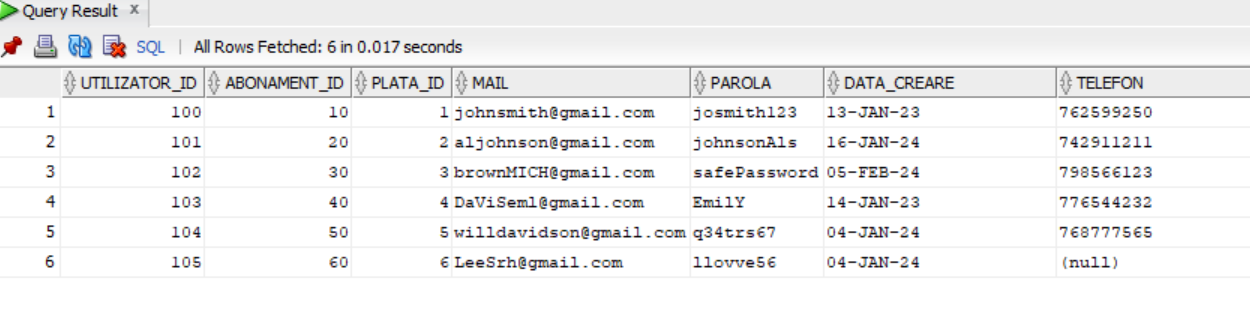
INSERT INTO UTILIZATOR VALUES(102, 30, 3, 'brownMICH@gmail.com', 'safePassword', to\_date('5.02.2024', 'dd.mm.yyyy'), '0798566123');

INSERT INTO UTILIZATOR VALUES(103, 40, 4, 'DaViSeml@yahoo.com', 'EmilY', to\_date('14.01.2023', 'dd.mm.yyyy'), '0776544232');

INSERT INTO UTILIZATOR VALUES(104, 50, 5, 'willdavidson@gmail.com', 'q34trs67', to\_date('4.01.2024', 'dd.mm.yyyy'), '0768777565');

INSERT INTO UTILIZATOR(utilizator\_id, abonament\_id, plata\_id, mail, parola, data\_creare)

VALUES(105, 60, 6, 'LeeSrh@gmail.com', 'llovve56', to\_date('4.01.2024', 'dd.mm.yyyy'));



**DIRECTOR**

INSERT INTO DIRECTOR VALUES(101, 'Hart', 'Hanson', to\_date('26.7.1957', 'dd.mm.yyyy'));

INSERT INTO DIRECTOR VALUES(202, 'Steven', 'Knight', to\_date('1.4.1959', 'dd.mm.yyyy'));

INSERT INTO DIRECTOR (director\_id, nume, prenume) VALUES(303, 'Adam', 'Horowitz');

INSERT INTO DIRECTOR (director\_id, nume, prenume) VALUES(404, 'Paul', 'Scheuring');

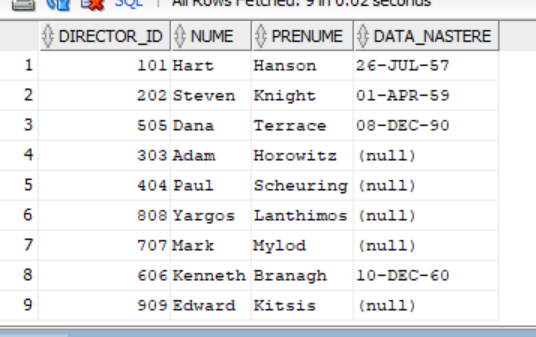
INSERT INTO DIRECTOR VALUES(505, 'Dana', 'Terrace', to\_date('8.12.1990', 'dd.mm.yyyy'));

INSERT INTO DIRECTOR VALUES(606, 'Kenneth', 'Branagh', to\_date('10.12.1960', 'dd.mm.yyyy'));

INSERT INTO DIRECTOR (director\_id, nume, prenume) VALUES(707, 'Mark', 'Mylod');

INSERT INTO DIRECTOR (director\_id, nume, prenume) VALUES(808, 'Yargos ', 'Lanthimos');

INSERT INTO DIRECTOR (director\_id, nume, prenume) VALUES(909, 'Edward', 'Kitsis');



**ACTOR**

INSERT INTO ACTOR VALUES(increment\_actor.nextval, 'Cillian', 'Murphy', to\_date('25.5.1976', 'dd.mm.yyyy'));

INSERT INTO ACTOR VALUES(increment\_actor.nextval, 'David', 'Boreanaz', to\_date('16.5.1969', 'dd.mm.yyyy'));

INSERT INTO ACTOR VALUES(increment\_actor.nextval, 'Lana', 'Parrilla', to\_date('15.7.1977', 'dd.mm.yyyy'));

INSERT INTO ACTOR VALUES(increment\_actor.nextval, 'Josh', 'Dallas', to\_date('18.12.1978', 'dd.mm.yyyy'));

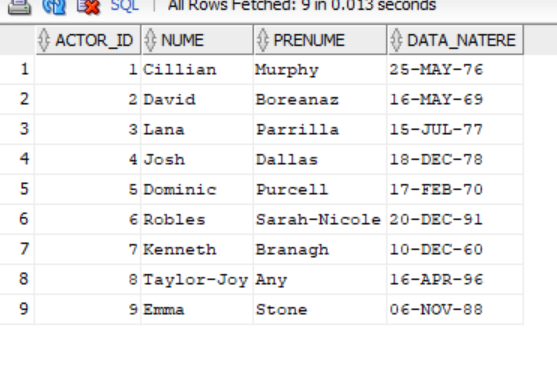
INSERT INTO ACTOR VALUES(increment\_actor.nextval, 'Dominic', 'Purcell', to\_date('17.2.1970', 'dd.mm.yyyy'));

INSERT INTO ACTOR VALUES(increment\_actor.nextval, 'Robles', 'Sarah-Nicole', to\_date('20.12.1991', 'dd.mm.yyyy'));

INSERT INTO ACTOR VALUES(increment\_actor.nextval, 'Kenneth', 'Branagh', to\_date('10.12.1960', 'dd.mm.yyyy'));

INSERT INTO ACTOR VALUES(increment\_actor.nextval, 'Taylor-Joy', 'Any', to\_date('16.4.1996', 'dd.mm.yyyy'));

INSERT INTO ACTOR VALUES(increment\_actor.nextval, 'Emma', 'Stone', to\_date('6.11.1988', 'dd.mm.yyyy'));



**ROL**

INSERT INTO ROL(rol\_id, nume\_caracter, importanta) VALUES(1, 'Thomas Shelby', 'principal');

INSERT INTO ROL(rol\_id, nume\_caracter, importanta) VALUES(2, 'Seeley Booth', 'principal');

INSERT INTO ROL(rol\_id, nume\_caracter, importanta) VALUES(3, 'Evil Queen', 'principal');

INSERT INTO ROL(rol\_id, nume\_caracter, importanta) VALUES(4, 'Prince Charming', 'principal');

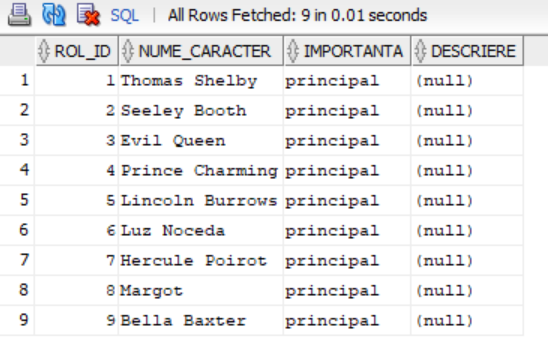
INSERT INTO ROL(rol\_id, nume\_caracter, importanta) VALUES(5, 'Lincoln Burrows', 'principal');

INSERT INTO ROL(rol\_id, nume\_caracter, importanta) VALUES(6, 'Luz Noceda', 'principal');

INSERT INTO ROL(rol\_id, nume\_caracter, importanta) VALUES(7, 'Hercule Poirot', 'principal');

INSERT INTO ROL(rol\_id, nume\_caracter, importanta) VALUES(8, 'Margot', 'principal');

INSERT INTO ROL(rol\_id, nume\_caracter, importanta) VALUES(9, 'Bella Baxter', 'principal');



**PRODUCTIE**

INSERT INTO PRODUCTIE VALUES(increment\_productie.nextval, 'Peaky Blinders', 'drama', to\_date('12.12.2013', 'dd.mm.yyyy'), 202);

INSERT INTO PRODUCTIE VALUES(increment\_productie.nextval, 'Death On The Nile', 'politist', to\_date('11.2.2022', 'dd.mm.yyyy'), 606);

INSERT INTO PRODUCTIE VALUES(increment\_productie.nextval, 'Bones', 'drama', to\_date('13.12.2005', 'dd.mm.yyyy'), 101);

INSERT INTO PRODUCTIE VALUES(increment\_productie.nextval, 'The Menu', 'comedie', to\_date('18.11.2022', 'dd.mm.yyyy'), 707);

INSERT INTO PRODUCTIE VALUES(increment\_productie.nextval, 'Prison Break', 'drama', to\_date('29.8.2005', 'dd.mm.yyyy'), 404);

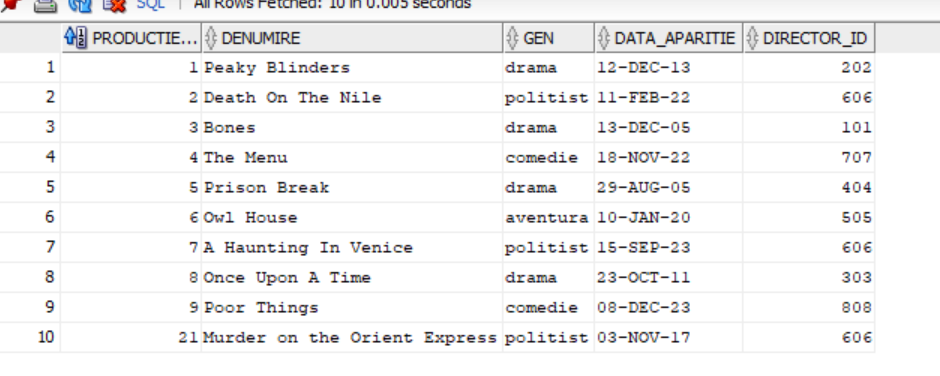
INSERT INTO PRODUCTIE VALUES(increment\_productie.nextval, 'Owl House', 'aventura', to\_date('10.1.2020', 'dd.mm.yyyy'), 505);

INSERT INTO PRODUCTIE VALUES(increment\_productie.nextval, 'A Haunting In Venice', 'politist', to\_date('15.9.2023', 'dd.mm.yyyy'), 606);

INSERT INTO PRODUCTIE VALUES(increment\_productie.nextval, 'Once Upon A Time', 'drama', to\_date('23.10.2011', 'dd.mm.yyyy'), 303);

INSERT INTO PRODUCTIE VALUES(increment\_productie.nextval, 'Poor Things', 'comedie', to\_date('8.12.2023', 'dd.mm.yyyy'), 808);

INSERT INTO PRODUCTIE VALUES(increment\_productie.nextval, 'Murder on the Orient Express', 'politist', to\_date('3.11.2017', 'dd.mm.yyyy'), 606);



**FILM**

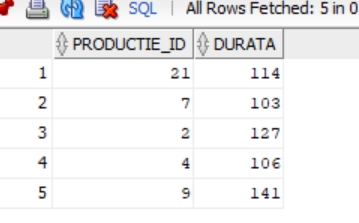
INSERT INTO FILM VALUES(7, 103);

INSERT INTO FILM VALUES(2, 127);

INSERT INTO FILM VALUES(4, 106);

INSERT INTO FILM VALUES(9, 141);

INSERT INTO FILM VALUES(21, 114);



**SERIAL**

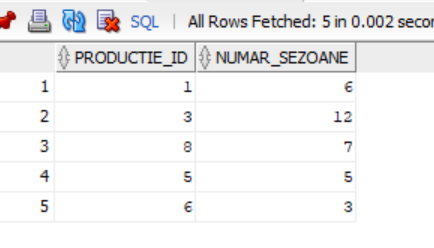
INSERT INTO SERIAL VALUES(1, 6);

INSERT INTO SERIAL VALUES(3, 12);

INSERT INTO SERIAL VALUES(8, 7);

INSERT INTO SERIAL VALUES(5, 5);

INSERT INTO SERIAL VALUES(6, 3);



**EPISOD**

INSERT INTO EPISOD VALUES(1, 1, 1, 1, 'Episod 1', 58);

INSERT INTO EPISOD VALUES(2, 1, 2, 1, 'Episod 2', 58);

INSERT INTO EPISOD VALUES(3, 3, 5, 7, 'The Twist int the Twister', 42);

INSERT INTO EPISOD VALUES(4, 3, 6, 7, 'The Crack In The Code', 45);

INSERT INTO EPISOD VALUES(5, 3, 11, 12, 'The Day in the Life', 46);

INSERT INTO EPISOD VALUES(6, 8, 1, 1, 'Pilot', 42);

INSERT INTO EPISOD VALUES(7, 8, 2, 1, 'The Thing You Love Most', 41);

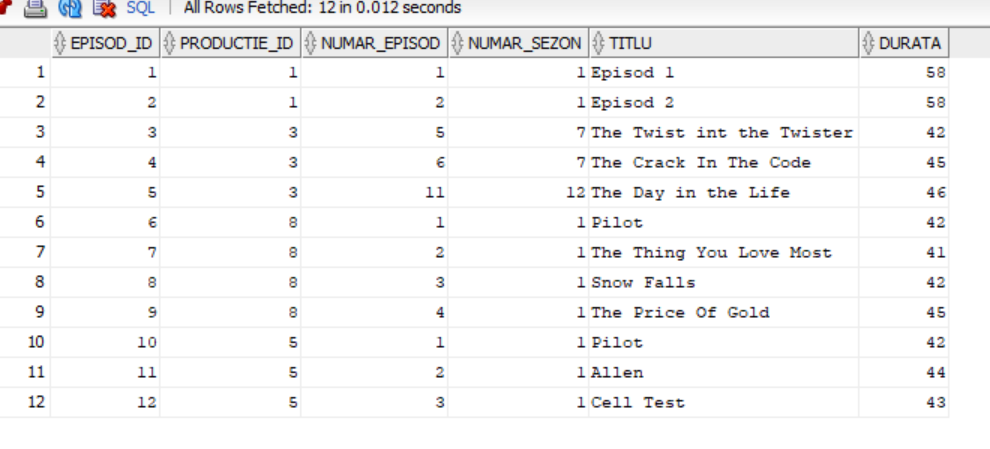
INSERT INTO EPISOD VALUES(8, 8, 3, 1, 'Snow Falls', 42);

INSERT INTO EPISOD VALUES(9, 8, 4, 1, 'The Price Of Gold', 45);

INSERT INTO EPISOD VALUES(10, 5, 1, 1, 'Pilot', 42);

INSERT INTO EPISOD VALUES(11, 5, 2, 1, 'Allen', 44);

INSERT INTO EPISOD VALUES(12, 5, 3, 1, 'Cell Test', 43);



**PROFIL-PRODUCTIE**

INSERT INTO PROFIL\_PRODUCTIE VALUES(111, 11, 1);

INSERT INTO PROFIL\_PRODUCTIE VALUES(112, 11, 2);

INSERT INTO PROFIL\_PRODUCTIE VALUES(113, 11, 7);

INSERT INTO PROFIL\_PRODUCTIE VALUES(114, 22, 7);

INSERT INTO PROFIL\_PRODUCTIE VALUES(115, 33, 7);

INSERT INTO PROFIL\_PRODUCTIE VALUES(116, 44, 9);

INSERT INTO PROFIL\_PRODUCTIE VALUES(117, 44, 3);

INSERT INTO PROFIL\_PRODUCTIE VALUES(118, 55, 9);

INSERT INTO PROFIL\_PRODUCTIE VALUES(119, 66, 6);

INSERT INTO PROFIL\_PRODUCTIE VALUES(120, 11, 6);

INSERT INTO PROFIL\_PRODUCTIE VALUES(121, 55, 7);

INSERT INTO PROFIL\_PRODUCTIE VALUES(122, 11, 4);

INSERT INTO PROFIL\_PRODUCTIE VALUES(123, 11, 9);

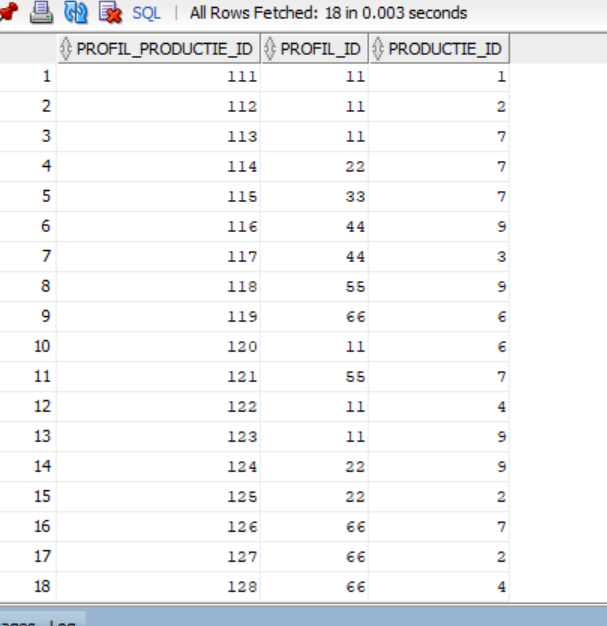
INSERT INTO PROFIL\_PRODUCTIE VALUES(124, 22, 9);

INSERT INTO PROFIL\_PRODUCTIE VALUES(125, 22, 2);

INSERT INTO PROFIL\_PRODUCTIE VALUES(126, 66, 7);

INSERT INTO PROFIL\_PRODUCTIE VALUES(127, 66, 2);

INSERT INTO PROFIL\_PRODUCTIE VALUES(128, 66, 4);



**PRODUCTIE-DIRECTOR**

INSERT INTO PRODUCTIE\_DIRECTOR VALUES(1, 1, 202);

INSERT INTO PRODUCTIE\_DIRECTOR VALUES(2, 2, 606);

INSERT INTO PRODUCTIE\_DIRECTOR VALUES(3, 3, 101);

INSERT INTO PRODUCTIE\_DIRECTOR VALUES(4, 4, 707);

INSERT INTO PRODUCTIE\_DIRECTOR VALUES(5, 5, 404);

INSERT INTO PRODUCTIE\_DIRECTOR VALUES(6, 6, 505);

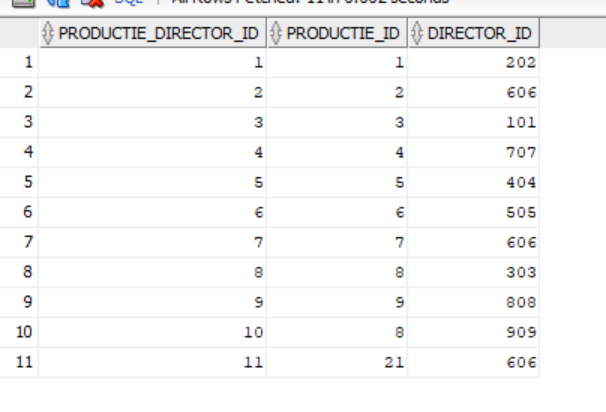
INSERT INTO PRODUCTIE\_DIRECTOR VALUES(7, 7, 606);

INSERT INTO PRODUCTIE\_DIRECTOR VALUES(8, 8, 303);

INSERT INTO PRODUCTIE\_DIRECTOR VALUES(9, 9, 808);

INSERT INTO PRODUCTIE\_DIRECTOR VALUES(10, 8, 909);

INSERT INTO PRODUCTIE\_DIRECTOR VALUES(11, 21, 606);



INSERT INTO ROL\_PRODUCTIE\_JUCAT VALUES(1, 1, 1, 1, 5);

INSERT INTO ROL\_PRODUCTIE\_JUCAT VALUES(2, 2, 2, 3, 5);

INSERT INTO ROL\_PRODUCTIE\_JUCAT VALUES(3, 3, 3, 8, 5);

INSERT INTO ROL\_PRODUCTIE\_JUCAT VALUES(4, 4, 4, 8, 5);

INSERT INTO ROL\_PRODUCTIE\_JUCAT VALUES(5, 5, 5, 5, 5);

INSERT INTO ROL\_PRODUCTIE\_JUCAT VALUES(6, 6, 6, 6, 359);

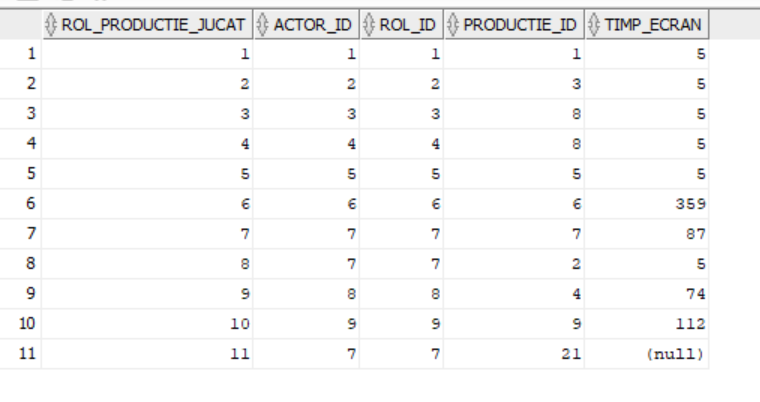
INSERT INTO ROL\_PRODUCTIE\_JUCAT VALUES(7, 7, 7, 7, 87);

INSERT INTO ROL\_PRODUCTIE\_JUCAT VALUES(8, 7, 7, 2, 5);

INSERT INTO ROL\_PRODUCTIE\_JUCAT VALUES(9, 8, 8, 4, 74);

INSERT INTO ROL\_PRODUCTIE\_JUCAT VALUES(10, 9, 9, 9, 112);

INSERT INTO ROL\_PRODUCTIE\_JUCAT(rol\_productie\_jucat, actor\_id, rol\_id, productie\_id) VALUES(11, 7, 7, 21);



# **Exercițiul 6** - Formulați în limbaj natural o problemă pe care să o rezolvați folosind un subprogram stocat independent care să utilizeze toate cele 3 tipuri de colecții studiate. Apelați subprogramul.

*Cerință:*

Creează un subprogram stocat (procedură sau funcție în SQL) care să realizeze următoarele operații:

1. Determinarea numărului maxim de producții vizionate de un profil pentru un aboanement dat:
   * Voi utiliza un **vector** pentru a stoca poreclele fiecarul profil și numărul total de episoade vizionate de fiecare
2. Afișarea tuturor rolurilor jucate de actori pentru o producție dată:
   * Voi utiliza un **tablou indexat** pentru a stoca numele actorilor și rolurile lor în producție.
3. Găsirea tuturor episoadelor care aparțin unui serial specific și identificarea directorilor implicați:
   * Voi utiliza un **tablou îmbricat** pentru a organiza informațiile despre episoade și directorii corespunzători.

*Explicație:*

Vom declara o procedură unde parametrii de intrare sunt (p\_abonament\_id, p\_producție\_id și p\_serial\_id). Dupa aceea declarăm vectorul și cele două tablouri.

Etapele procedurii:

1. vom face o interogare SQL pentru a găsi numărul de producții vizionate (COUNT) pentru fiecare profil al abonamentului specificat. Rezultatele sunt stocate într-ul varray numit v\_vizionari. După aceea voi parcurge vectorul pentru a determina valoarea maximă și se afișează.
2. Vom face o interogare SQL pentru a extrage actorii și rolurile lor din tabelele actor, rol\_producție\_jucat și rol pentru o producție specificată. Înformațiile sunt stocate în două tablouri indexate v\_actori și v\_roluri. Se parcurge fiecare tablou pentru a afișa perechi Actor-Rol.
3. Vom face o interogare SQL pentru a obține episoadele și directorii asociați unui serial specificat. Rezultatele sunt stocate într-un tabel imbricat v\_tabel\_imbricat episod\_director. După aceea parcurgem tabloul și afișam perechi Episod-Director.

1. CREATE OR REPLACE PROCEDURE PROCESARE\_DATE(

2. P\_ABONAMENT\_ID IN NUMBER,

3. P\_PRODUCTIE\_ID IN NUMBER,

4. P\_SERIAL\_ID IN NUMBER

5. )

6. IS

7. -- 1. VECTOR pentru stocarea numărului total de producții vizionate per profil

8. TYPE VECTOR\_VIZIONARI IS VARRAY(20) OF NUMBER;

9. V\_VIZIONARI VECTOR\_VIZIONARI := VECTOR\_VIZIONARI(); -- Inițializare VARRAY

10.

11. AUXILIAR NUMBER := 0;

12.

13. -- 2. TABLOU INDEXAT pentru stocarea actorilor și a rolurilor lor

14. TYPE TABLOU\_ROLURI IS TABLE OF VARCHAR2(200) INDEX BY PLS\_INTEGER;

15. V\_ACTORI TABLOU\_ROLURI;

16. V\_ROLURI TABLOU\_ROLURI;

17.

18. -- 3. TABLOU IMBRICAT pentru stocarea episoadelor și a directorilor

19. TYPE EPISOD\_DIRECTOR IS RECORD (

20. EPISOD VARCHAR2(100),

21. DIRECTOR VARCHAR2(100)

22. );

23. TYPE TABEL\_IMBRICAT IS TABLE OF EPISOD\_DIRECTOR;

24. V\_TABEL\_IMBRICAT TABEL\_IMBRICAT := TABEL\_IMBRICAT(); -- Inițializare TABEL IMBRICAT

25.

26. BEGIN

27. -- 1. Determinarea numărului maxim de producții vizionate de un profil al abonamentului

28. FOR R IN (

29. SELECT P.PROFIL\_ID, P.PORECLA, COUNT(PP.PRODUCTIE\_ID) AS NUMAR\_PRODUCTII

30. FROM PROFIL P

31. LEFT JOIN PROFIL\_PRODUCTIE PP

32. ON P.PROFIL\_ID = PP.PROFIL\_ID

33. JOIN ABONAMENT A

34. ON A.ABONAMENT\_ID = P.ABONAMENT\_ID

35. WHERE A.ABONAMENT\_ID = P\_ABONAMENT\_ID

36. GROUP BY P.PROFIL\_ID, P.PORECLA

37. ) LOOP

38. V\_VIZIONARI.EXTEND; -- Extindere VARRAY

39. V\_VIZIONARI(V\_VIZIONARI.COUNT) := R.NUMAR\_PRODUCTII; -- Adăugare valoare

40. END LOOP;

41.

42. -- Calcularea valorii maxime

43. FOR I IN 1 .. V\_VIZIONARI.COUNT LOOP

44. IF V\_VIZIONARI(I) > AUXILIAR THEN

45. AUXILIAR := V\_VIZIONARI(I);

46. END IF;

47. END LOOP;

48. DBMS\_OUTPUT.PUT\_LINE('Numărul maxim de producții vizionate de un profil al abonamentului ' || P\_ABONAMENT\_ID || ' este ' || AUXILIAR || '.');

49.

50. -- 2. Afișarea rolurilor jucate de actori pentru o producție dată

51. FOR T IN (

52. SELECT A.NUME AS ACTOR, R.NUME\_CARACTER AS ROL

53. FROM ACTOR A

54. JOIN ROL\_PRODUCTIE\_JUCAT RPJ

55. ON A.ACTOR\_ID = RPJ.ACTOR\_ID

56. JOIN ROL R

57. ON RPJ.ROL\_ID = R.ROL\_ID

58. WHERE RPJ.PRODUCTIE\_ID = P\_PRODUCTIE\_ID

59. ) LOOP

60. V\_ACTORI(V\_ACTORI.COUNT + 1) := T.ACTOR; -- Adăugare actor

61. V\_ROLURI(V\_ROLURI.COUNT + 1) := T.ROL; -- Adăugare rol

62. END LOOP;

63.

64. DBMS\_OUTPUT.PUT\_LINE('Roluri jucate pentru producția ' || P\_PRODUCTIE\_ID || ':');

65. FOR I IN 1 .. V\_ACTORI.COUNT LOOP

66. DBMS\_OUTPUT.PUT\_LINE('Actor: ' || V\_ACTORI(I) || ', Rol: ' || V\_ROLURI(I));

67. END LOOP;

68.

69. -- 3. Găsirea episoadelor și a directorilor unui serial

70. FOR S IN (

71. SELECT E.TITLU AS EPISOD, D.NUME AS DIRECTOR

72. FROM EPISOD E

73. JOIN PRODUCTIE\_DIRECTOR PD

74. ON E.PRODUCTIE\_ID = PD.PRODUCTIE\_ID

75. JOIN DIRECTOR D

76. ON PD.DIRECTOR\_ID = D.DIRECTOR\_ID

77. WHERE E.PRODUCTIE\_ID = P\_SERIAL\_ID

78. ) LOOP

79. V\_TABEL\_IMBRICAT.EXTEND; -- Extindere tabel îmbricat

80. V\_TABEL\_IMBRICAT(V\_TABEL\_IMBRICAT.LAST) := EPISOD\_DIRECTOR(S.EPISOD, S.DIRECTOR); -- Adăugare în tabel

81. END LOOP;

82.

83. DBMS\_OUTPUT.PUT\_LINE('Episoade și directori pentru serialul ' || P\_SERIAL\_ID || ':');

84. FOR I IN 1 .. V\_TABEL\_IMBRICAT.COUNT LOOP

85. DBMS\_OUTPUT.PUT\_LINE('Episod: ' || V\_TABEL\_IMBRICAT(I).EPISOD || ', Director: ' || V\_TABEL\_IMBRICAT(I).DIRECTOR);

86. END LOOP;

87.

88. END;

89. /

90.

91. -- Testare procedură

92. BEGIN

93. PROCESARE\_DATE(

94. P\_ABONAMENT\_ID => 10,

95. P\_PRODUCTIE\_ID => 2,

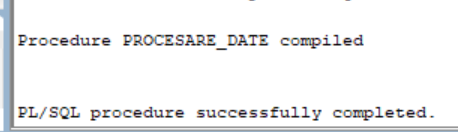
96. P\_SERIAL\_ID => 3

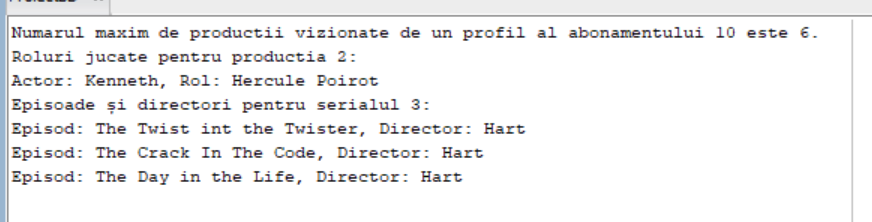
97. );

98. END;

99. /

100.





# **Exercițiul 7** - Formulați în limbaj natural o problemă pe care să o rezolvați folosind un subprogram stocat independent care să utilizeze 2 tipuri diferite de cursoare studiate, unul dintre acestea fiind cursor parametrizat, dependent de celălalt cursor. Apelați subprogramul.

*Cerință:*

Creează un subprogram stocat (procedură sau funcție în SQL) care să realizeze următoarele operații:

1. Să afișeze titlurile producțiilor (filme sau seriale) vizionate de utilizatori pentru un anumit tip de abonament.
2. Pentru fiecare producție găsită, să afișeze detalii despre actorii principali care au jucat în producție (actorii și rolurile lor).

Pentru a rezolva această problemă:

* Vom utiliza două **cursoare**:
  + Un **ciclu cursor parametrizat** care va extrage informațiile despre producții pentru un tip de abonament specificat.
  + Un al doilea **ciclu cursor cu subcereri simplu** dependent de primul, care va extrage actorii principali pentru fiecare producție găsită.

*Explicație:*

Vom declara o procedură unde parametrul de intrare este p\_abonament\_id.

Declar un ciclu cursor simplu cu care voi extrage producțiile vizionate de profilele asociate cu abonamentul cerut. Parcurc cursorul de producții și afișez titlul. Folosind ciclu cursor cu subcerere voi parcurge toti actori care joaca în producția din primul cursor si ii voi afisa împreună cu rolul jucat.

1. CREATE OR REPLACE PROCEDURE analiza\_vizionari(

2. p\_abonament\_id IN NUMBER -- ID-ul abonamentului pentru care facem analiza

3. )

4. IS

5. -- Ciclu cursor parametrizat pentru extragerea producțiilor vizionate de utilizatorii abonamentului

6. CURSOR c\_productions(p\_abonament\_id NUMBER) IS

7. SELECT DISTINCT p.productie\_id, pr.denumire

8. FROM profil\_productie p

9. JOIN profil prf ON p.profil\_id = prf.profil\_id

10. JOIN productie pr ON p.productie\_id = pr.productie\_id

11. WHERE prf.abonament\_id = p\_abonament\_id;

12.

13.

14. -- Variabile pentru gestionarea datelor din primul cursor

15. v\_productie\_id NUMBER;

16. v\_titlu VARCHAR2(200);

17. BEGIN

18. -- Parcurgem cursorul pentru producții

19. OPEN c\_productions(p\_abonament\_id);

20.

21. LOOP

22. FETCH c\_productions INTO v\_productie\_id, v\_titlu;

23. EXIT WHEN c\_productions%NOTFOUND;

24.

25. -- Afișăm titlul producției

26. DBMS\_OUTPUT.PUT\_LINE('Producție vizionată: ' || v\_titlu);

27.

28. -- Cilcu cursor cu cubcereri simplu pentru extragerea actorilor principali pentru o producție

29. for i in(SELECT a.nume AS actor, r.nume\_caracter AS rol

30. FROM actor a

31. JOIN rol\_productie\_jucat rpj ON a.actor\_id = rpj.actor\_id

32. JOIN rol r ON rpj.rol\_id = r.rol\_id

33. WHERE rpj.productie\_id = v\_productie\_id)

34. loop

35. DBMS\_OUTPUT.PUT\_LINE(' Actor: ' || i.actor || ', Rol: ' || i.rol);

36. END LOOP;

37.

38. END LOOP;

39.

40. CLOSE c\_productions;

41.

42. END;

43. /BEGIN

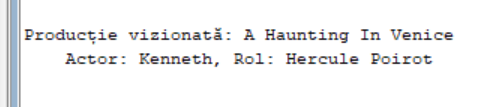
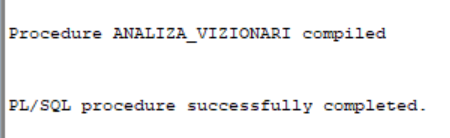
44. analiza\_vizionari(20); -- ID-ul abonamentului pentru care dorim să analizăm vizionările

45. END;

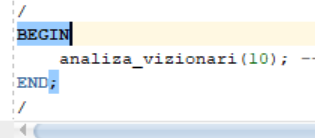
46. /

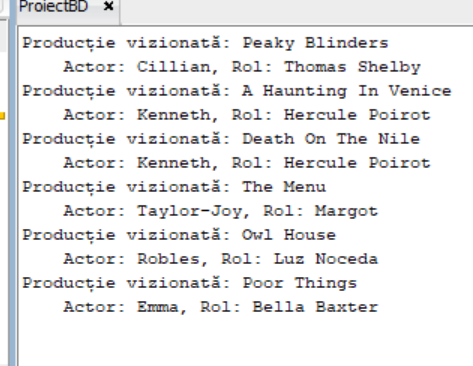
47.

**Exemplu din cod pentru abonamentul 20 avem:**



**Alt exemplu pentru abonamentul 10 avem:**





# **Exercițiul 8** - Formulați în limbaj natural o problemă pe care să o rezolvați folosind un subprogram stocat independent de tip funcție care să utilizeze într-o singură comandă SQL 3 dintre tabelele create. Tratați toate excepțiile care pot apărea, incluzând excepțiile predefinite NO\_DATA\_FOUND și TOO\_MANY\_ROWS. Apelați subprogramul astfel încât să evidențiați toate cazurile tratate

*Cerință:*

Se dorește identificarea numărului total de producții vizionate de către un profil, împreună cu detalii despre cea mai vizionată producție a utilizatorului (titlul producției și numărul de vizionări)

*Explicație:*

Creez procedura cu parametrul p\_porecla.

Declar variabile pentru stocarea rezultatului. O porecla este asociată unui profil\_id si vom cauta acest id. Calculez numărul de producții vizionate pentru afisarea finală.

Caut cea mai vizionată producție și numă de câte ori a fost vizionată și afișez textul final cu numărul de producții vizionate și care a fost vizionată cea mai des.

1. CREATE OR REPLACE FUNCTION analiza\_utilizator(

2. p\_porecla IN VARCHAR2 -- Porecla profilului pentru care dorim analiza

3. ) RETURN VARCHAR2

4. IS

5. -- Variabile pentru stocarea rezultatului

6. v\_profil\_id NUMBER;

7. v\_total\_vizionari NUMBER := 0;

8. v\_titlu\_cel\_mai\_vizionat VARCHAR2(200);

9. v\_nr\_vizionari\_max NUMBER := 0;

10. v\_result VARCHAR2(500);

11. BEGIN

12. -- Găsim ID-ul profilului asociat cu porecla. Aici se va declanșa TOO\_MANY\_ROWS dacă există mai multe profiluri.

13. SELECT

14. profil\_id

15. INTO

16. v\_profil\_id

17. FROM

18. profil

19. WHERE

20. porecla = p\_porecla;

21.

22. -- Calculăm numărul total de producții vizionate

23. SELECT

24. COUNT(pp.productie\_id) AS total\_vizionari

25. INTO

26. v\_total\_vizionari

27. FROM

28. profil\_productie pp

29. WHERE

30. pp.profil\_id = v\_profil\_id;

31.

32. -- Calculăm titlul celei mai vizionate producții și numărul de vizionări, utilizând ROW\_NUMBER

33. SELECT

34. titlu, nr\_vizionari

35. INTO

36. v\_titlu\_cel\_mai\_vizionat, v\_nr\_vizionari\_max

37. FROM (

38. SELECT

39. pr.denumire AS titlu,

40. COUNT(pp.productie\_id) AS nr\_vizionari,

41. ROW\_NUMBER() OVER (ORDER BY COUNT(pp.productie\_id) DESC, pr.denumire) AS rn

42. FROM

43. profil\_productie pp

44. JOIN productie pr ON pp.productie\_id = pr.productie\_id

45. WHERE

46. pp.profil\_id = v\_profil\_id

47. GROUP BY pr.denumire

48. )

49. WHERE rn = 1; -- Selectăm doar producția cu cel mai mare număr de vizionări

50.

51. -- Construim rezultatul final

52. v\_result := 'Utilizatorul cu porecla ' || p\_porecla ||

53. ' a vizionat în total ' || v\_total\_vizionari || ' producții. ' ||CHR(10)||

54. 'Cea mai vizionată producție este "' || v\_titlu\_cel\_mai\_vizionat ||

55. '" cu ' || v\_nr\_vizionari\_max || ' vizionări.';

56.

57. RETURN v\_result;

58.

59. -- Tratarea excepțiilor

60. EXCEPTION

61. WHEN NO\_DATA\_FOUND THEN

62. RETURN 'Nu există date pentru utilizatorul cu porecla "' || p\_porecla || '".';

63. WHEN TOO\_MANY\_ROWS THEN

64. RETURN 'Eroare: există mai mulți utilizatori cu porecla "' || p\_porecla ||

65. '". Verificați unicitatea datelor în baza de date.';

66. WHEN OTHERS THEN

67. RETURN 'Eroare neașteptată: ' || SQLERRM;

68. END analiza\_utilizator;

69. /

70.

71.

72.

73. BEGIN

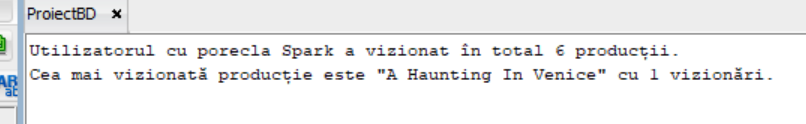
74. DBMS\_OUTPUT.PUT\_LINE(analiza\_utilizator('Spark'));

75. END;

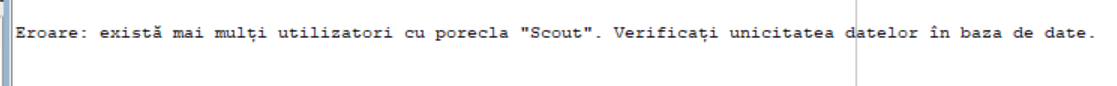
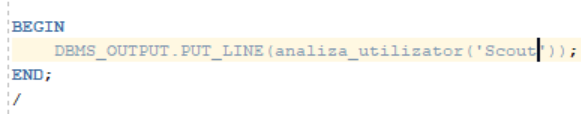
76. /

77.

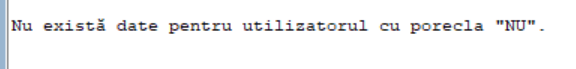
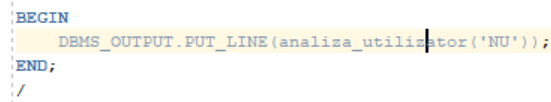
**Exemplu din cod pentru porecla Sparck:**



**Exemplu pentru TOO\_MANY\_ROWS – exista 2 conturi cu porecla Scout**



**Exemplu pentru NO\_DATA\_FOUND – nu exista porecla NU**



# **Exercițiul 9** - Formulați în limbaj natural o problemă pe care să o rezolvați folosind un subprogram stocat independent de tip procedură care să aibă minim 2 parametri și să utilizeze într-o singură comandă SQL 5 dintre tabelele create. Definiți minim 2 excepții proprii, altele decât cele predefinite la nivel de sistem. Apelați subprogramul astfel încât să evidențiați toate cazurile definite și tratate.

*Cerință:*

Scrieți o procedură stocată care, dat fiind o porecla și un gen de producție, să returneze lista de producții (filme sau seriale) vizionate de utilizator în genul respectiv. Procedura trebuie să trateze următoarele excepții definite:

1. NO\_DATA\_FOUND – dacă utilizatorul cu porecla cerută nu exista
2. TOO\_MANY\_ROWS – dacă există mai mulți utilizatori cu porecla cerută
3. NoProductionsForGenreException – genul specificat există în baza de date, dar profilul nu a vizionat nimic de acest gen
4. NoSuchGenreInDatabaseException – genul specificat nu există deloc în baza de date

*Explicație:*

Creez procedura care primeste ca parametru o poreclă și un gen.

După aceea definesc excepțiile proprii și variabilele intermediare.

Încep prin a verifica daca profilul cu porecla specificată există. Daca user\_exist returnează 0 înseamnă ca nu a fost gasit nici un profil cu aceasă poreclă - no data found). Dacă returnează orice mai mare decât 1 înseamnă că există mai multe profile cu această porecla - too many rows).

Dacă există un singur profil cu porecla cerută caut daca genul specificat exista în baza de date selectând toate producțiile cu genul specificat. Dacă nu exista nici una - NoSuchGenreInDatabaseException.

Dacă există genul specificat caut daca utilizatorul a vizionat vreo producție de acest gen. Dacă nu a văzul – NoProductionForGenreException.

Dacă am trecut de toate excepțile parcurc toate producțiile de un anumit gen vizionate de un anumit profil si le afisez.

1. CREATE OR REPLACE PROCEDURE get\_productions\_by\_user\_and\_genre(

2. p\_porecla IN Varchar2,

3. production\_genre IN VARCHAR2

4. )

5. IS

6. -- Excepții proprii

7. NoProductionsForGenreException EXCEPTION;

8. NoSuchGenreInDatabaseException EXCEPTION;

9.

10. -- Variabile intermediare

11. user\_exists NUMBER;

12. production\_count NUMBER;

13. genre\_exists number;

14.

15. BEGIN

16. -- Verifică dacă utilizatorul există

17. SELECT COUNT(\*)

18. INTO user\_exists

19. FROM profil

20. WHERE porecla = p\_porecla;

21.

22. IF user\_exists = 0 THEN

23. RAISE no\_data\_found;

24. ELSIF user\_exists > 1 THEN

25. RAISE too\_many\_rows;

26. END IF;

27.

28. -- Check if the genre exists in the database (Productie table)

29. SELECT COUNT(\*)

30. INTO genre\_exists

31. FROM PRODUCTIE

32. WHERE LOWER(gen) = LOWER(production\_genre);

33.

34. IF genre\_exists = 0 THEN

35. RAISE NoSuchGenreInDatabaseException;

36. END IF;

37.

38. -- Verifică dacă există producții vizionate de utilizator pentru genul specificat

39. SELECT COUNT(\*)

40. INTO production\_count

41. FROM UTILIZATOR u

42. JOIN PROFIL p ON u.abonament\_id = p.abonament\_id

43. JOIN PROFIL\_PRODUCTIE pp ON p.profil\_id = pp.profil\_id

44. JOIN PRODUCTIE prod ON pp.productie\_id = prod.productie\_id

45. WHERE p.porecla = p\_porecla AND LOWER(prod.gen) = LOWER(production\_genre);

46.

47. IF production\_count = 0 THEN

48. RAISE NoProductionsForGenreException;

49. END IF;

50.

51. DBMS\_OUTPUT.PUT\_LINE('Utilizatorul cu porecla: '||p\_porecla);

52. -- Afișează lista producțiilor vizionate

53. FOR rec IN (

54. SELECT prod.denumire, prod.gen, prod.data\_aparitie,

55. CASE

56. WHEN f.productie\_id IS NOT NULL THEN 'Film'

57. WHEN s.productie\_id IS NOT NULL THEN 'Serial'

58. END AS productie\_tip

59. FROM UTILIZATOR u

60. JOIN PROFIL p ON u.abonament\_id = p.abonament\_id

61. JOIN PROFIL\_PRODUCTIE pp ON p.profil\_id = pp.profil\_id

62. JOIN PRODUCTIE prod ON pp.productie\_id = prod.productie\_id

63. LEFT JOIN FILM f ON prod.productie\_id = f.productie\_id

64. LEFT JOIN SERIAL s ON prod.productie\_id = s.productie\_id

65. WHERE p.porecla = p\_porecla AND LOWER(prod.gen) = LOWER(production\_genre)

66. ) LOOP

67. DBMS\_OUTPUT.PUT\_LINE('Titlu: ' || rec.denumire || ', Gen: ' || rec.gen ||

68. ', Data Apariție: ' || TO\_CHAR(rec.data\_aparitie, 'dd.mm.yyyy') ||

69. ', Tip Producție: ' || rec.productie\_tip);

70. END LOOP;

71.

72. EXCEPTION

73. WHEN no\_data\_found THEN

74. DBMS\_OUTPUT.PUT\_LINE('Eroare: Utilizatorul cu porecla ' || p\_porecla || ' nu există.');

75. WHEN too\_many\_rows THEN

76. DBMS\_OUTPUT.PUT\_LINE('Eroare: Exista mai multi utilizatori cu porecla: ' || p\_porecla || ' .');

77. WHEN NoProductionsForGenreException THEN

78. DBMS\_OUTPUT.PUT\_LINE('Eroare: Nu există producții vizionate de utilizator pentru genul specificat.');

79. WHEN NoSuchGenreInDatabaseException THEN

80. DBMS\_OUTPUT.PUT\_LINE('Eroare: Nu există producții în baza de date pentru genul: ' || production\_genre);

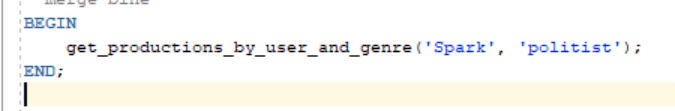
81. WHEN OTHERS THEN

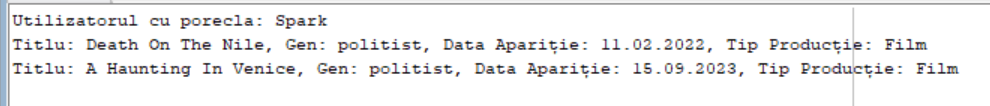
82. DBMS\_OUTPUT.PUT\_LINE('A apărut o eroare necunoscută: ' || SQLERRM);

83. END;

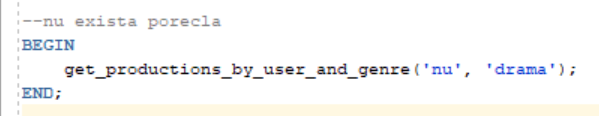
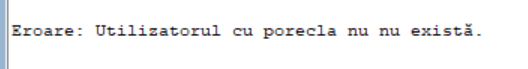
84. /

**Exemplu pentru porecla Spark, genul polițist –** porecla este unică și genul există în baza de date, iar profilul specificat a vizionat cel puțin o producție de acest gen**:**

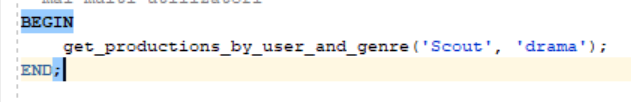




**Exemplu pentru o porecla care nu exista:**

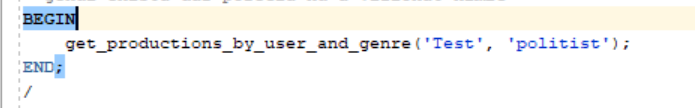
 

**Exemplu pentru o porecla folosită de mai mulți utilizatori:**



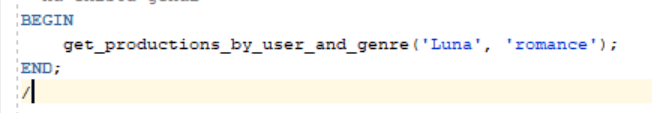


**Exemplu pentru porecla unica si gen existent, dar profilul nu a vazut nici o producție de acest gen:**





**Exemplu de gen care nu există:**





# **Exercițiul 10** - Definiți un trigger de tip LMD la nivel de comandă. Declanșați trigger-ul

*Cerință:*

Creați un trigger pentru tabela abonament care **să blocheze** orice operațiune de insert, update sau delete în zilele de weekend sau in lunile pare => adică operațiunile se pot efectua doar în zilele lucrătoare în lunile impare.

*Explicație:*

Creez trigger-ul. Verific daca ziua este sâmbătă sau duminică sau dacă luna este pară, dacă vreuna din contiții este adevărată atunci triggerul generează un mesaj de eroare.

1. CREATE OR REPLACE TRIGGER trig\_abonament

2. BEFORE INSERT OR UPDATE OR DELETE ON abonament

3. BEGIN

4. IF (TO\_CHAR(SYSDATE,'D') IN ('1', '7'))

5. OR (MOD(TO\_NUMBER(TO\_CHAR(SYSDATE, 'MM')), 2)=0)

6. THEN

7. RAISE\_APPLICATION\_ERROR(-20001,'tabelul nu poate fi actualizat');

8. END IF;

9. END;

10. /

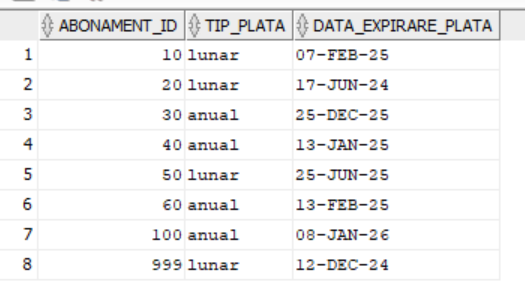
11.

12. insert into abonament values(999, 'lunar', to\_date('12.12.2024', 'dd.mm.yyyy'));

13.

**Exemplu pentru insert** – încercat pe data de 9.01.2024 zi de joi luna impară:





Dacă schimb verificarea lunii pentru a verifica daca luna este pară pentru a verifica mesajul de eroare:

1. CREATE OR REPLACE TRIGGER trig\_abonament

2. BEFORE INSERT OR UPDATE OR DELETE ON abonament

3. BEGIN

4. IF (TO\_CHAR(SYSDATE,'D') IN ('1', '7'))

5. OR (MOD(TO\_NUMBER(TO\_CHAR(SYSDATE, 'MM')), 2)=1)

6. THEN

7. RAISE\_APPLICATION\_ERROR(-20001,'tabelul nu poate fi actualizat');

8. END IF;

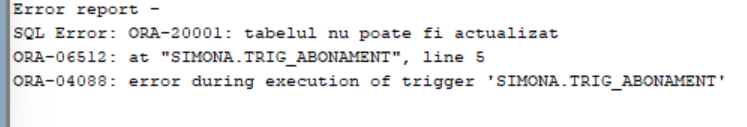
9. END;

10. /

11.

12. insert into abonament values(999, 'lunar', to\_date('12.12.2024', 'dd.mm.yyyy'));

13.



# **Exercițiul 11** - Definiți un trigger de tip LMD la nivel de linie. Declanșați trigger-ul.

*Cerință:*

Scrieți un trigger care să se activeze înainte de orice operațiul de update. Dacă abonamentul este activ nu se pot face schimbari. Dacă abonamentul este lunar adaug 30 de zile la plată, dacă este anual adaug un an.  
*Explicație:*

Verific dacă plata este expirata. Dacă este expirată verific daca plata este lunară – caz în care adaug 30 de zile sau anuală – caz în care adaug un an. Dar dacă data nu este expirată generez mesajul de eroare.

1. CREATE OR REPLACE TRIGGER TRG\_UPDATE\_ABONAMENT

2. BEFORE UPDATE ON ABONAMENT

3. FOR EACH ROW

4. BEGIN

5. -- Verific daca plata a expirat

6. IF :OLD.data\_expirare\_plata < SYSDATE THEN

7. -- Adaug zile sau un an în func?ie de tipul de plat?

8. IF :OLD.tip\_plata = 'lunar' THEN

9. :NEW.data\_expirare\_plata := SYSDATE + 30;

10. ELSIF :OLD.tip\_plata = 'anual' THEN

11. :NEW.data\_expirare\_plata := ADD\_MONTHS(SYSDATE, 12);

12. END IF;

13. ELSE

14. RAISE\_APPLICATION\_ERROR(-20001, 'Nu se poate actualiza un abonament care nu a expirat.');

15. END IF;

16. END;

17. /

18.

19. insert into abonament values(100, 'anual', to\_date('12.12.2024', 'dd.mm.yyyy'));

20. select \* from abonament;

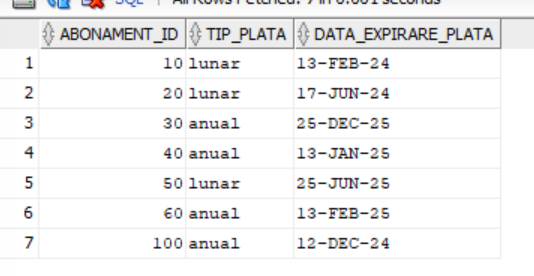
21.

22. UPDATE ABONAMENT

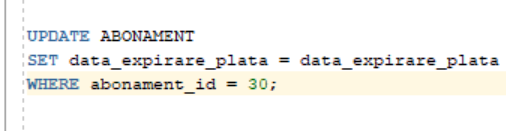
23. SET data\_expirare\_plata = data\_expirare\_plata

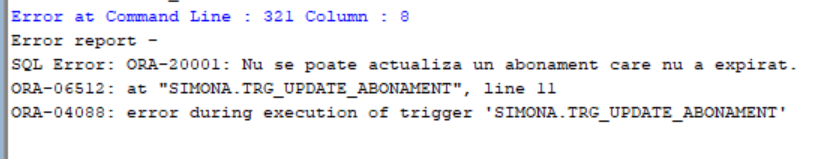
24. where abonament\_id=10;

Tabelul înainte de orice schimbare – abonamentul 100 este adăugat acum pentru verificarea cazului abonament expirat anual.

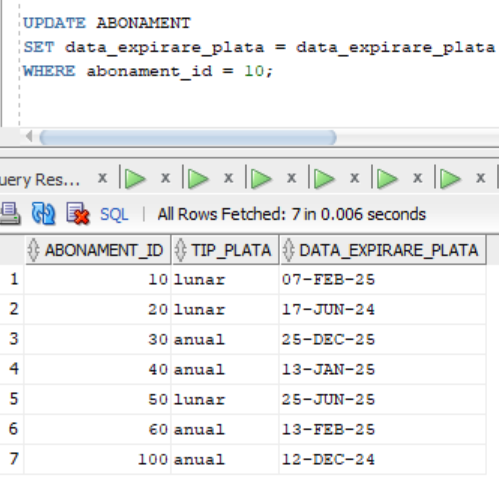


**Exemplu în care abonamentul nu este expirat:**

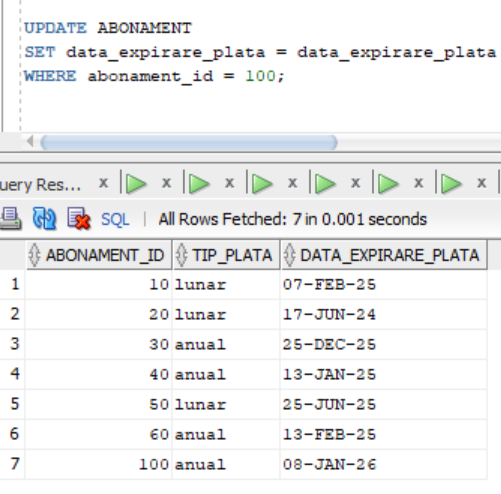




**Exemplu de abonament expirat lunar:**



**Exemplul de abonament expirat annual:**



# **Exercițiul 12** - Definiți un trigger de tip LDD. Declanșați trigger-ul.

*Cerință:*

Scrieți un trigger LDD care monitorizează evenimentele de tip create, alter sau alter.

*Explicație:*

Creez tabelul log\_disney salvand utilizatorul si numele bazei de date, evenimentul, obiectul, tipul de obiect și data.

Creez triggerul care după fiecare create, drop sau alter inserează cele menționate în log\_disney.

1. CREATE TABLE log\_disney

2. (

3. utilizator VARCHAR2(50),

4. nume\_baza\_date VARCHAR2(50),

5. eveniment VARCHAR2(50),

6. obiect VARCHAR2(50),

7. tip\_obj VARCHAR2(50),

8. data\_even DATE

9. );

10.

11.

12. CREATE OR REPLACE TRIGGER log\_disney\_trig

13. AFTER CREATE OR DROP OR ALTER ON SCHEMA

14. BEGIN

15. INSERT INTO log\_disney VALUES(SYS.LOGIN\_USER,

16. SYS.DATABASE\_NAME,

17. SYS.SYSEVENT,

18. SYS.DICTIONARY\_OBJ\_NAME,

19. SYS.DICTIONARY\_OBJ\_TYPE,

20. SYSDATE);

21. END;

22. /

23.

24. create table exemple (

25. nr number);

26.

27. alter table exemple add (nr2 number);

28.

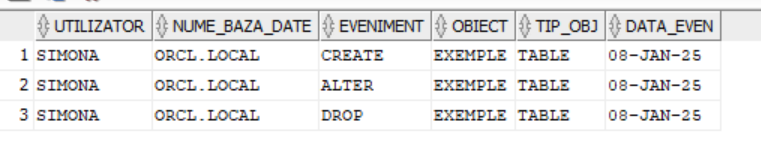
29. drop table exemple;

30.

31. select \* from log\_disney;

32.

Tabela log\_disney dupa create table exemple, alter table example și drop table exemple.



# **Exercițiul 13** - Formulați în limbaj natural o problemă pe care să o rezolvați folosind un pachet care să includă tipuri de date complexe și obiecte necesare unui flux de acțiuni integrate, specifice bazei de date definite (minim 2 tipuri de date, minim 2 funcții, minim 2 proceduri).

*Cerință:*

Scrieți un pachet care gestionează informațiile despre filme, seriale și actorii asociați și conține:

1. 2 tipuri de date complexe
   * Just\_movies – detalii despre filme
   * Just\_series – detalii despre seriale
2. 2 proceduri
   * What\_series – procedură care afisează informații despre toate serialele văzute de un anumit profil
   * Which\_actor – procedură care identifică și afisează actorul cu cele mai multe roluri jucate, împreună cu numărul total de roluri.
3. 2 funcții

* Most\_series\_year – funcție care determină și returnează anul în care au apărut cele mai multe seriale
* How\_many\_minutes – funcție care returnează numărul total de minute de filme vizionate de un anumit profil

Explicație:

Creez pachetul și definesc cele 2 tipuri de tipuri de date record.

În body-ul pachetului avem:

1. what\_series – definesc series\_detail de tip just\_series. Folosind un cursor parcurc serialele vizionate de profilul cerut și folosind cursorul le afisez.
2. How\_many\_minutes – definesc number\_of\_minutes inițializat cu 0. Selectez suma duratelor filmelor fizionate de profilul cerut și returnez numărul de minute.
3. Most\_series\_year – definesc max\_year și extrag anul din data apariției fiecărul serial, grupez serialele după an, sortez rezultatele în funcție de numărul de seriale și selectez primul an.
4. Which\_actor – definesc actor\_detail si role\_count. Folosind cursoul selectez actorul cu cele mai multe roluri

1. CREATE OR REPLACE PACKAGE BODY movie\_package AS

2. PROCEDURE what\_series(p\_profil\_id number) IS

3. -- Local variable of type table to hold movie details

4. series\_detail just\_series;

5.

6. -- Cursor to fetch movie details

7. CURSOR series\_cursor IS

8. SELECT s.productie\_id AS serial\_id,

9. p.denumire,

10. p.gen,

11. p.data\_aparitie,

12. p.director\_id,

13. s.numar\_sezoane

14. FROM profil\_productie pp

15. join serial s on pp.productie\_id=s.productie\_id

16. JOIN productie p ON s.productie\_id = p.productie\_id

17. where pp.profil\_id=p\_profil\_id;

18. BEGIN

19. DBMS\_OUTPUT.PUT\_LINE('Seriale vazute de profilul cu id-ul:'||p\_profil\_id);

20. DBMS\_OUTPUT.PUT\_LINE('----------------------------------------------');

21.

22. -- Open the cursor and fetch rows

23. FOR series\_detail IN series\_cursor LOOP

24. DBMS\_OUTPUT.PUT\_LINE('Serial ID: ' || series\_detail.serial\_id);

25. DBMS\_OUTPUT.PUT\_LINE('Denumire: ' || series\_detail.denumire);

26. DBMS\_OUTPUT.PUT\_LINE('Gen: ' || series\_detail.gen);

27. DBMS\_OUTPUT.PUT\_LINE('Data Aparitie: ' || TO\_CHAR(series\_detail.data\_aparitie, 'DD.MM.YYYY'));

28. DBMS\_OUTPUT.PUT\_LINE('Director ID: ' || series\_detail.director\_id);

29. DBMS\_OUTPUT.PUT\_LINE('Numar\_sezoane: ' || series\_detail.numar\_sezoane );

30. DBMS\_OUTPUT.PUT\_LINE('----------------------------------------------');

31. END LOOP;

32. END what\_series;

33.

34. function how\_many\_minutes

35. (p\_profil\_id number)

36. return number is

37. number\_of\_minutes number:=0;

38. begin

39. select sum(f.durata)

40. into number\_of\_minutes

41. from profil\_productie pp

42. join film f on pp.productie\_id=f.productie\_id

43. where pp.profil\_id=p\_profil\_id;

44.

45. return number\_of\_minutes;

46. end how\_many\_minutes;

47.

48. function most\_series\_year return number is

49. max\_year number; -- Variabilă pentru anul cu cele mai multe seriale

50. begin

51. -- Interogare SQL pentru a găsi anul cu cele mai multe seriale

52. select extract(year from p.data\_aparitie)

53. into max\_year

54. from serial s

55. join productie p on s.productie\_id = p.productie\_id

56. group by extract(year from p.data\_aparitie)

57. order by count(s.productie\_id) desc

58. fetch first row only;

59.

60. return max\_year; -- Returnează anul

61. end most\_series\_year;

62.

63.

64. procedure which\_actor is

65. actor\_detail actor.nume%type;

66. role\_count number;

67. begin

68. for rec in(

69. select a.nume as actor\_name,

70. count(rj.productie\_id) as role\_count

71. from rol\_productie\_jucat rj

72. join actor a on rj.actor\_id=a.actor\_id

73. group by a.nume

74. order by role\_count desc

75. fetch first row only

76. ) loop

77. actor\_detail := rec.actor\_name;

78. role\_count := rec.role\_count;

79. end loop;

80.

81. DBMS\_OUTPUT.PUT\_LINE('Actor with the most roles: ' || actor\_detail);

82. DBMS\_OUTPUT.PUT\_LINE('Number of roles: ' || role\_count);

83. end which\_actor;

84. END movie\_package;

85. /

86.

**Exemplu pentru what\_series:**

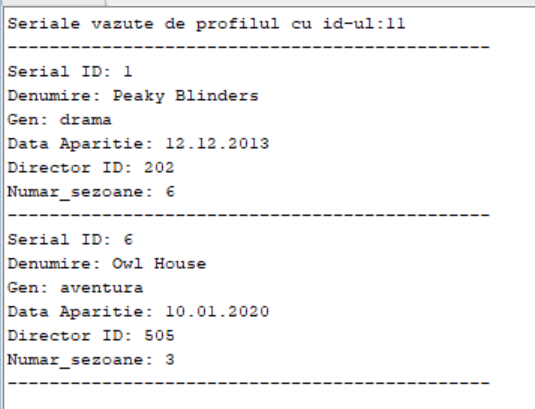
1. begin

2. movie\_package.what\_series(11);

3. end;

4. /

5.



**Exemplu pentru how\_many\_minutes:**

1. DECLARE

2. total\_minutes NUMBER;

3. BEGIN

4. -- Call the function and store the result in a variable

5. total\_minutes := movie\_package.how\_many\_minutes(11);

6.

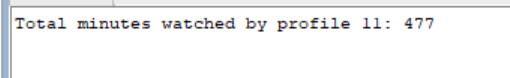
7. -- Output the result

8. DBMS\_OUTPUT.PUT\_LINE('Total minutes watched by profile 11: ' || total\_minutes);

9. END;

10. /

11.



**Exemplu pentru most\_series\_year:**

1. DECLARE

2. year\_with\_most\_series NUMBER;

3. BEGIN

4. -- Call the function to find the year with the most series

5. year\_with\_most\_series := movie\_package.most\_series\_year;

6.

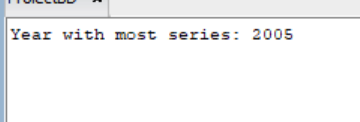
7. -- Output the result

8. DBMS\_OUTPUT.PUT\_LINE('Year with most series: ' || year\_with\_most\_series);

9. END;

10. /

11.



**Exemplu pentru which\_actor:**

1. BEGIN

2. movie\_package.which\_actor;

3. END;

4. /

5.

