JEGYZŐKÖNYV

Modern adatbázis rendszerek MSc

Oracle PL/SQL APEX, OOP és Neo4J

Készítette: **Garamszegi**

Márton Balázs

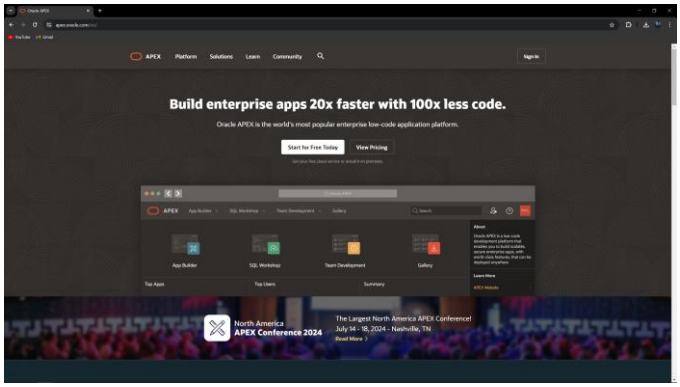
Neptunkód: **AJYKQ3**

Dátum: 2024. május. 13.

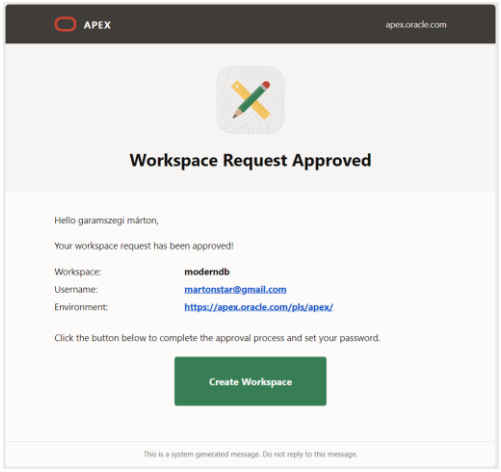
Tartalom

1 Oracle APEX környezet beállítása

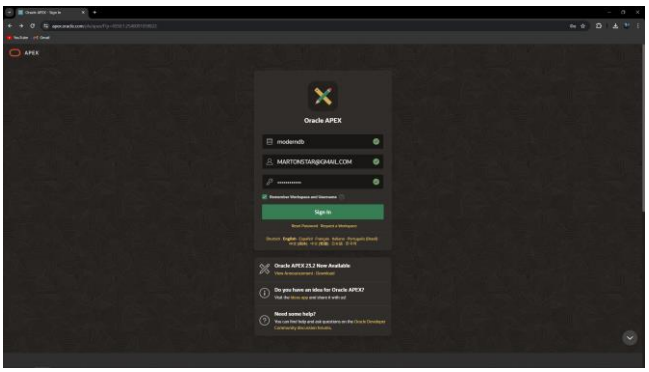
A feladat végrehajtásához kérelmeztem és létrehoztam a „moderndb” adatbázist az Oracle APEX weboldalán. Itt fogom létrehozni az objektumokat, és hajtom végre a feladatokat PL/SQL nyelven.
Először is el kell látogatni az Oracle weboldalára: https://apex.oracle.com



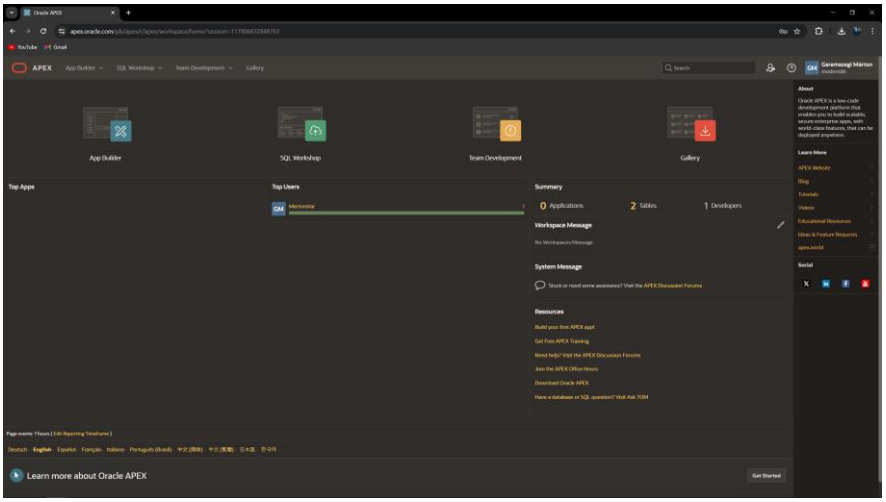
És itt kell kérelmezni egy adatbázist. Ehhez emailes regisztráció, és adatbázis név megadása szükséges. A kérelmezést követően amennyiben elfogadták, elkészülhet az adatbázis és megkezdthető rajta a munka.

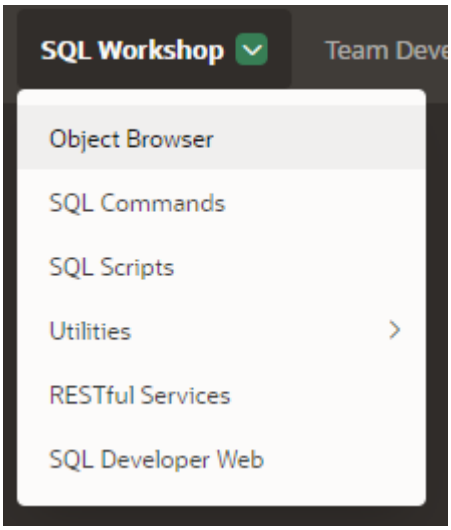


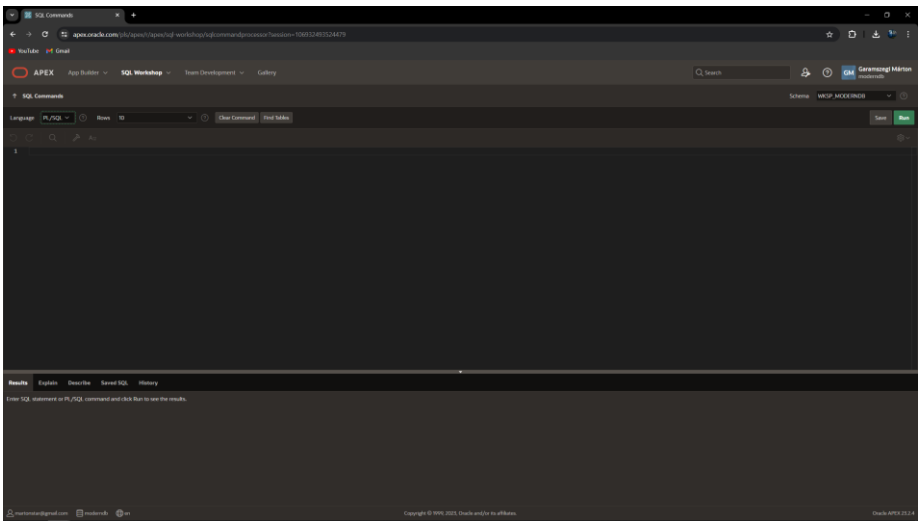
Az elkészült workspace létrehozása után be kell jelentkeznünk a felületre. A bejelentkezéshez szintén szükséges megadni az email címet, az adatbázis nevét, illetve a jelszót



Sikeres bejelentkezést követően az alábbi felület fogad minket.
Az SQL Workshop gombon belül található SQL Commands menüponttal tudjuk elkezdeni a munkát.

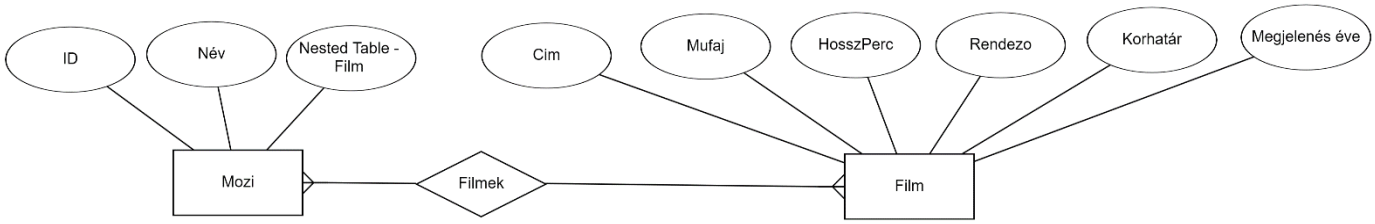






2.1 APEX feladat: Objektum és tábla létrehozása, adatok feltöltése, lekérdezések, módosítások

A feladatot az alábbi ER modell segítségével készítettem, ami egy előző beadandó egy részét képzi. A feladat megoldása során a szintaktikai helyesség és óvatosság végett angolul vannak kezelve az adatok.



Az első feladat a Film objektum típus létrehozása volt name, production_year, director, price attribútumokkal.
Ezek a UDT-k (User Defined Type) hasznosak az összetett entitások egyetlen objektumként való megjelenítésében, amely megosztható az alkalmazások között, és segít a kód újrafelhasználására is.

```
1  CREATE TYPE film_type AS OBJECT (
2  name VARCHAR2(255),
3  production_year NUMBER,
4  director VARCHAR2(255),
5  price NUMBER
6  );
7
```

Results | Explain | Describe | Saved SQL | History

Type created.

0.03 seconds

Ezután létrehoztam egy film_showroom táblát, amely tartalmazza a filmeklet. Ezek tartalmaznak egy automatikusan generált film_id-t is.

```
1  CREATE TABLE film_showroom (
2    film_id NUMBER GENERATED ALWAYS AS IDENTITY,
3    film film_type
4  );
```

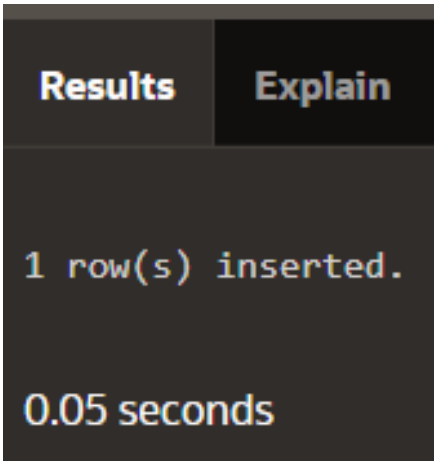
Results | Explain | Describe | Saved SQL | History

Table created.

0.04 seconds

Ezek után feltöltöttem adatokat a táblába, hogy tudjuk kezelni a filmeket:

```
BEGIN

INSERT INTO film_showroom (film) VALUES (
  film_type('The Shawshank Redemption', 1994, 'Frank Darabont', 1100)
);

INSERT INTO film_showroom (film) VALUES (
  film_type('The Godfather', 1972, 'Francis Ford Coppola', 900)
);

INSERT INTO film_showroom (film) VALUES (
  film_type('The Dark Knight', 2008, 'Christopher Nolan', 1300)
);

INSERT INTO film_showroom (film) VALUES (
  film_type('Pulp Fiction', 1994, 'Quentin Tarantino', 1000)
);

INSERT INTO film_showroom (film) VALUES (
  film_type('Forrest Gump', 1994, 'Robert Zemeckis', 950)
);

END;
```



Lekérdezések

Minden film objektum nevének, majd árának lekérdezése a film_showroom táblából. Ezt az f aliasnév használatával tudjuk elérni. Erre szükség van, mivel azok egy táblában vannak, csak így érhetőek el.

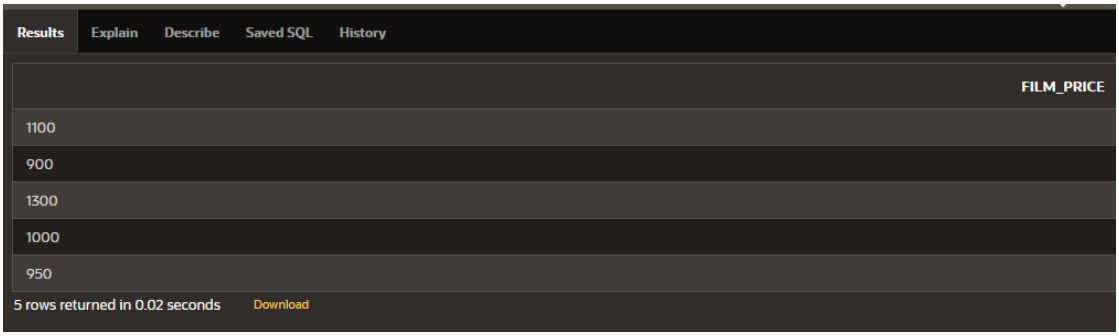
```
SELECT f.film.name AS film_name FROM film_showroom f;
```

Results | Explain | Describe | Saved SQL | History

FILM_NAME
The Shawshank Redemption
The Godfather
The Dark Knight
Pulp Fiction
Forrest Gump

5 rows returned in 0.01 seconds Download

```
SELECT f.film.price AS film_price FROM film_showroom f;
```



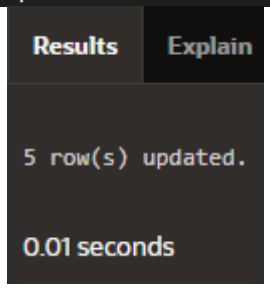
Results | Explain | Describe | Saved SQL | History

FILM_PRICE
1100
900
1300
1000
950

5 rows returned in 0.02 seconds Download

Feladat: Minden film ára legyen 900, ami 5 évnél idősebb. Itt is aliasneveket kellett használni, az adott objektumon belül elérni kívánt adattagot a pont segítségével érhetjük el. Az adatok módosításához a „price” és a „production_year” attribútumokat kellett elérni.

```
UPDATE film_showroom f SET f.film.price = 900 WHERE f.film.production_year < 2019;
```



Results | Explain

5 row(s) updated.

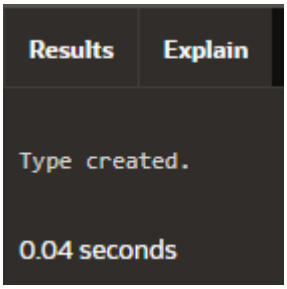
0.01 seconds

Ezután a lekérdezésben már az új adatokkal fognak megjelenni az objektumok:

2.2 APEX feladat: Beágyazott tábla létrehozása, kezelése. Adatok felvétele, frissítése adott kritériumnak megfelelően. Adatok, illetve tábla adatok lekérdezése.

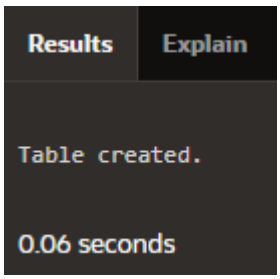
Beágyazott tábla létrehozása, amely a film típusokat fogja tudni tartalmazni:

```
CREATE TYPE nested_film_table IS TABLE OF film_type;
```



A showroom_network tábla létrehozása az id, cinema, és a film_table adattagokkal. A szintaktikának megfelelően, a beágyazott tábla mezője csak adott táblát tartalmazhat. Ezt a kód utolsó sora adja utasításba a fordítónak.

```
CREATE TABLE showroom_network (
  id NUMBER,
  cinema VARCHAR2(255),
  film_table nested_film_table)
  NESTED TABLE film_table STORE AS nested_film_tables
```



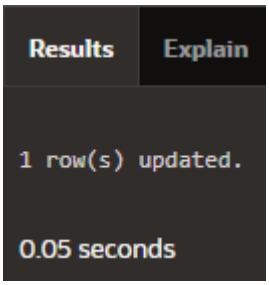
Adatok felvétele

Először is létrehozzuk a táblákat a beágyazott táblában, amik a filmeket fogja tartalmazni. Ezeket tudjuk később kritériumként megszabni a feltöltésnél.

```
INSERT INTO showroom_network (cinema) VALUES ('Cinema City Miskolc');
INSERT INTO showroom_network (cinema) VALUES ('Cinema City Debrecen');
```

Ezután 2 filmet felvesuünk a Miskolci Cinema Citybe

```
UPDATE showroom_network
SET film_table = nested_film_table(
  film_type('Schindlers List', 1993, 'Steven Spielberg', 1800),
  film_type('Inglourious Basterds', 2009, 'Quentin Tarantino', 2200)
)
WHERE cinema = 'Cinema City Miskolc';
```



Majd 1 film felvétele a Debreceni Cinema Citybe

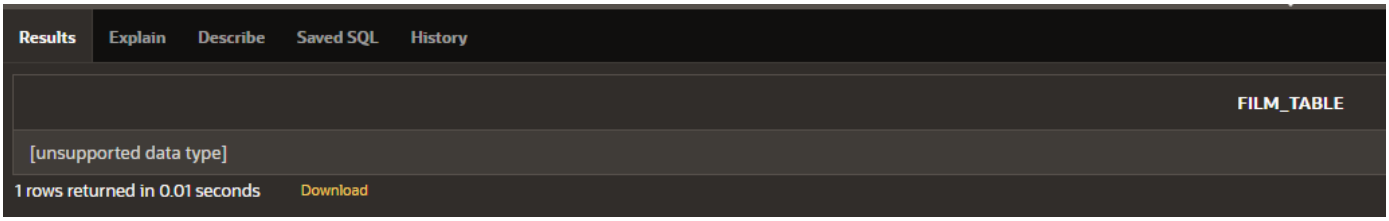
```
UPDATE showroom_network
SET film_table = nested_film_table(
  film_type('The Lord of the Rings: The Return of the King', 2003, 'Peter Jackson', 1700)
)
WHERE cinema = 'Cinema City Debrecen';
```

Lekérdezések

A Miskolci Cinema Cityben található filmek lekérdezése

```
SELECT film_table FROM showroom_network WHERE cinema = 'Cinema City Miskolc';
```

Érdekesség, hogy a visszakapott érték nem alapértelmezett típus, így a ResultSet nem tudja megfelelően megjeleníteni. Erről később még lesz szó. Az alábbi üzenetet kapjuk:



Minden adat lekérdezése

```
SELECT id, cinema, film_table FROM showroom_network;
```

ID	CINEMA	FILM_TABLE
-	Cinema City Debrecen	[unsupported data type]
-	Cinema City Miskolc	[unsupported data type]

Egyszerűbben lekérdezve, ha **minden** adatot le szeretnénk kérdezni, a „*” operátort is használhatjuk

```
SELECT * FROM showroom_network;
```

ID	CINEMA	FILM_TABLE
-	Cinema City Debrecen	[unsupported data type]
-	Cinema City Miskolc	[unsupported data type]

2 rows returned in 0.01 seconds Download

Beágyazott tábla adatainak lekérése

```
SELECT id, cinema, f.* FROM showroom_network s, TABLE(s.film_table) f;
```

ID	CINEMA	NAME	PRODUCTION_YEAR	DIRECTOR	PRICE
-	Cinema City Debrecen	The Lord of the Rings: The Return of the King	2003	Peter Jackson	1700
-	Cinema City Miskolc	Schindlers List	1993	Steven Spielberg	1800
-	Cinema City Miskolc	Inglourious Basterds	2009	Quentin Tarantino	2200

3 rows returned in 0.01 seconds Download

3.1 OOP feladat: Származtatható objektum interface-ének létrehozása.

Létrehozzuk az objektumot (interface rész) ha már létezik, kicseréljük a meglévőt, erről a REPLACE gondoskodik.

```
CREATE OR REPLACE TYPE Filmek_o AS OBJECT (
    cim VARCHAR2(100),
    mufaj VARCHAR2(20),
    hosszPerc NUMBER,
    rendezo VARCHAR2(100),
    korhatar NUMBER,
    megjelenes_ev NUMBER(4),
    MEMBER FUNCTION get_hosszOra (H NUMBER) RETURN NUMBER,
    MEMBER FUNCTION get_megjelenes (P NUMBER) RETURN NUMBER,
    MEMBER FUNCTION is_retro (P NUMBER) RETURN VARCHAR2,
    MEMBER PROCEDURE get_korhatar,
    MEMBER PROCEDURE set_korhatar (G VARCHAR2)
) NOT FINAL;
```

Results | Explain

Type created.

A deklarációs rész tartalmazza az objektum attribútumait, metódusait.
A NOT FINAL rész lényeges, hiszen e nélkül nem származtatható a létrehozott objektum.

3.2 OOP feladat: Objektum interface implementálása, body létrehozása, függvények és procedúrák elkészítése, programozása.

A body rész az adott már definiált típus implementációját tartalmazza. Ez a blokk felel az üzleti logikáért, illetve itt kerülnek kifejtésre a metódusok és procedúrák is.

```
CREATE OR REPLACE TYPE BODY Filmek_o IS
```

Függvények
get_hosszOra: A megkapott percet (időt) számban kapja meg, visszaadja azt órában.

```
MEMBER FUNCTION get_hosszOra (H NUMBER) RETURN NUMBER IS

    BEGIN
        RETURN (H/60);
    END;
```

get_megjelenes: Visszaadja a megjelenési évet.

```
MEMBER FUNCTION get_megjelenes (P NUMBER) RETURN NUMBER IS

    BEGIN
        RETURN P;
    END;
```

is_retro: Megadott megjelenési év alapján eldönti, hogy egy film retronak számít-e vagy sem.

```
MEMBER FUNCTION is_retro (P NUMBER) RETURN VARCHAR2 IS

    BEGIN
        IF (P < 1980) THEN
            RETURN 'Igen';
        ELSE
            RETURN 'Nem';
        END IF;
    END;
```

Procedúrák

get_korhatar: Kiírja a konzolba a film korhatárát.

```
MEMBER PROCEDURE get_korhatar IS

    BEGIN
        DBMS_OUTPUT.PUT_LINE('A film korhatára: ' || SELF.korhatar);
    END;
```

set_korhatar: Beállítja az adott műfajhoz tartozó korhatárt. Amennyiben nem lehet meghatározni, ezt jelzi a konzolon, és automatikusan 12-es korhatár kerül beállításra.

```
MEMBER PROCEDURE set_korhatar (G VARCHAR2) IS

    BEGIN
        IF (G = 'Horror' OR G = 'Thriller') THEN
            SELF.korhatar := 18;
        ELSIF (G = 'Akció' OR G = 'Sci-fi') THEN
            SELF.korhatar := 16;
        ELSE
            DBMS_OUTPUT.PUT_LINE('Ismeretlen műfaj, nincs megadot korhatár! Az alapértelmezett 12-es korhatár kerül beállításra.');
            SELF.korhatar := 12;
        END IF;
END;
```

3.3. Alobjektum létrehozása, konstruktorral.

Az alobjektum örökli az eredeti objektum adatstruktúráját, attribútumait, függvényeit és procedúráit. Csak egyszeres öröklődés lehetséges. Az öröklődésért az UNDER szó felel. Ez után kell megadnunk, hogy melyik már létező objektumnak lesz az „örököse”.

```
CREATE OR REPLACE TYPE Film_o UNDER Filmek_o (

title VARCHAR2(100),
genre VARCHAR2(20),
movieLength NUMBER,
director VARCHAR2(100),
publication NUMBER(4),

CONSTRUCTOR FUNCTION Film_o (
    title VARCHAR2,
    genre VARCHAR2,
    movieLength NUMBER,
    director VARCHAR2,
    publication NUMBER
)
    RETURN SELF AS RESULT,
        STATIC PROCEDURE film_adatok(title VARCHAR2, genre VARCHAR2, director VARCHAR2, retro VARCHAR2),
        MEMBER FUNCTION get_kor RETURN NUMBER
);
```

A konstruktor olyan speciális függvény, amely a felhasználó által definiált típusból egy új példánnyal tér vissza, illetve beállítja attribútumainak értékeit.

3.4 Alobjektum body meghatározása, konstruktor kifejtése. Statikus procedúrák és függvények létrehozása.

Konstruktor kifejtése, példány adattagok beállítása a hívott értékekre

```
CREATE OR REPLACE TYPE BODY Film_o AS
    CONSTRUCTOR FUNCTION Film_o(
        title VARCHAR2,
        genre VARCHAR2,
        movieLength NUMBER,
        director VARCHAR2,
        publication NUMBER)
    RETURN SELF AS RESULT AS
        BEGIN
            SELF.title := title;
            SELF.genre := genre;
            SELF.movieLength := movieLength;
            SELF.director := director;
            SELF.publication := publication;
                RETURN;
        END;
```

Statikus procedúra:
film_adatok

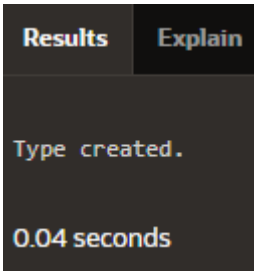
Ez a statikus procedúra a megadott film adatait írja ki. Egy if szerkezet dönt arról, hogy a film retronak számít-e, aszerint kerülnek az adatok kiírásra a konzolban.

```
STATIC PROCEDURE film_adatok(title VARCHAR2, genre VARCHAR2, director VARCHAR2, retro VARCHAR2) AS
    BEGIN
        DBMS_OUTPUT.PUT_LINE('------ A film adatai ------');
        DBMS_OUTPUT.NEW_LINE;
        DBMS_OUTPUT.NEW_LINE;
        DBMS_OUTPUT.PUT_LINE('A film címe: ' || title);
        DBMS_OUTPUT.PUT_LINE('Műfaja: ' || genre);
        DBMS_OUTPUT.PUT_LINE('Rendezte: ' || director);
        IF (retro = 'Igen') THEN
            DBMS_OUTPUT.NEW_LINE;
            DBMS_OUTPUT.PUT_LINE('A film az 1980-es évek előtt készült, így retrónak számít.');
        END IF;
    END;
```

Alosztály hozzáadott függvénye:
get_kor

Ezt a függvényt csak az alosztály tartalmazza. A függvény a jelenlegi dátumból és a jelenlegi példány (SELF) megjelenési idejéből kiszámolja a film korát, majd visszatér vele.

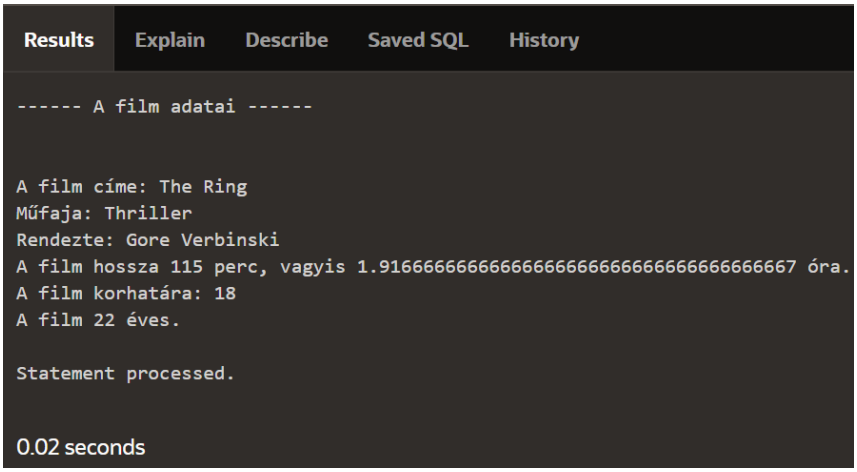
```
MEMBER FUNCTION get_kor RETURN NUMBER AS
    BEGIN
        RETURN TO_CHAR(SYSDATE,'YYYY') - SELF.publication;
    END;
```



3.5 Futtatás, függvények és procedúrák tesztelése

```
DECLARE
film Film_o;
BEGIN
    FILM := Film_o('The Ring', 'Thriller', 115, 'Gore Verbinski', 2002);
    Film_o.film_adatok(film.title, film.genre, film.director, film.is_retro(film.publication));
    DBMS_OUTPUT.PUT_LINE('A film hossza ' || film.movieLength || ' perc, vagyis ' || Film.get_hosszOra(film.movieLength) || ' óra.');
    Film.set_korhatar(film.genre);
    Film.get_korhatar();
    DBMS_OUTPUT.PUT_LINE('A film ' || Film.get_kor() || ' éves.');
END;
```

A futás eredménye:



4. Neo4J Java JDBC API létrehozása

A Java API létrehozásához szükségünk lesz a neo4j Java driverre. Ezt a https://github.com/neo4j/neo4j-jdbc Github oldalon lehet elérni és letölteni.
Amennyiben projektet hozunk létre, a `POM.XML`-be az alábbi dependencyt tudjuk beállítani:

```
<dependency>
    <groupId>org.neo4j</groupId>
    <artifactId>neo4j-jdbc-full-bundle</artifactId>
    <version>6.0.0-M03</version>
</dependency>
```

Ez biztosítja a megfelelő kapcsolat és kommunikáció kialakítását a Neo4J és a Java között.

Először is létre kell hoznunk a táblát, ami a filmeket fogja tartalmazni:

```
package neo4j;

public class filmT {
     String id;
     String name;
     int releaseDate;
     String director;
     int price;


     public filmT(String id, String name, int releaseDate, String
director, int price) {
          super();
          this.id = id;
          this.name = name;
          this.releaseDate = releaseDate;
          this.director = director;
          this.price = price;
     }
}
```

Ez egy sémát tartalmaz, amit majd a fő alkalmazás fog használni a Queryk és parancsok végrehajtásakor.

Ezután a fő alkalmazást írjuk meg.

Bekérjük a megfelelő könyvtárakat, amik szükségesek az adatbázis kezeléséhez, kapcsolódáshoz, driver kezeléséhez, és egyéb JDBC parancsokhoz:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Properties;
```

Majd bekérjük az előbb elkészített sémát:

```
import neo4j.filmT;
```

Az osztályban 2 metódust hozunk létre. Az add_films metódus filmeket ad hozzá az adatbázishoz, a read_films pedig beolvassa azokat az adatbázisból megadott kritérium alapján. A metódusok végrehajtásának végén visszajelzünk a felhasználónak.

```
public class app {

    public static void main(String[] args) {
        add_films();
        read_films(1800);

        System.out.println("A Querynek vége, kilépés.\n");
    }
```

Az add_films metódus létrehozása. Megkeressük a megfelelő drivert, beállítjuk a kapcsolódáshoz szükséges URL-t, felhasználónevet és jelszót, létrehozzunk a kapcsolatot.

```
    public static void add_films() {
        try {
            Class.forName("org.neo4j.jdbc.Driver");
            Properties props = new Properties();
            props.setProperty("user", "neo4j");
            props.setProperty("password", "neo4j");
            Connection con =
DriverManager.getConnection("jdbc:neo4j:http://localhost:8389",
props);
```

Ezután létrehozzuk a queryt, és a filmT sémáját követve megadjuk a felvinni kívánt adatokat az alábbi módon.

```
            String query;

            filmT[] films = new filmT[5];
            films[0] = new filmT('1', 'The Shawshank Redemption',
1994, 'Frank Darabont', 1100);
            films[1] = new filmT('2', 'The Godfather', 1972,
'Francis Ford Coppola', 900);
            films[2] = new filmT('3', 'The Dark Knight', 2008,
'Christopher Nolan', 1300);
            films[3] = new filmT('4', 'Pulp Fiction', 1994,
'Quentin Tarantino', 1000);
            films[4] = new filmT('5', 'Forrest Gump', 1994,
'Robert Zemeckis', 950);
```

A query fogja az adatbázison belül a táblát létrehozó szintaktikát tárolni:

```
            query = "CREATE (a:film {_id:{1}, name:{2},
releaseDate:{3}, director:{4}, price:{5}})";
```

A PreparedStatement előre definiált parancsot (query-t) képes végrehajtani az adatbázison. Ezt létrehozzuk, majd feltöltjük a már létrehozott film adataival:

```
            PreparedStatement stmt = con.prepareStatement(query);

            for (int i=0; i<6; i++) {
                 stmt.setString(1, films[i].id);
                 stmt.setString(2, films[i].name);
                 stmt.setInt(3, films[i].releaseDate);
                 stmt.setString(4, films[i].director);
                 stmt.setString(5, films[i].price);

                 stmt.executeUpdate();
            }
```

Ha végeztünk, és már nem teszünk egyebet, lezárjuk a kapcsolatot, amennyiben hiba jelentkezik azt kiírjuk a felhasználónak:

```
            con.close();

        }   catch(Exception e) {
                 System.out.println("Hiba történt: " +
e.getMessage());
    }
}
```

A read_films metódus megírása. Ezt egy PreparedStatement segítségével ismét elküldjük az adatbázisnak, ami visszaadja majd a találatokat.

```
    public static void read_films(int priceLimit) {

        try
        {
            Class.forName("org.neo4j.jdbc.Driver");
```

```
            Properties props = new Properties();
            props.setProperty("user", "neo4j");
            props.setProperty("password", "neo4j");
            Connection con =
DriverManager.getConnection("jdbc:neo4j:http://localhost:8389",prop
s);
```

Az ellenőrzések és kapcsolódás után ismét egy queryt hozunk létre, ami most nem a létrehozást, hanem keresést (MATCH) tartalmaz.

```
            String query = "MATCH (A:film) WHERE A.price > {1}
RETURN A.name";
            PreparedStatement stmt = con.prepareStatement(query);
            stmt.setInt(1, priceLimit);
```

A queryben lévő keresés eredményét a ResultSet-ben tároljuk. A talált film nevét kiírjuk a felhasználónak.

```
            ResultSet rs = stmt.executeQuery();
            while(rs.next()) {
                System.out.println(rs.getString("A.name"));
            }
```

Ha végeztünk, és már nem teszünk egyebet, lezárjuk a kapcsolatot, amennyiben hiba jelentkezik azt kiírjuk a felhasználónak:

```
            con.close();

        }   catch(Exception e) {
                System.out.println("Hiba történt: " +
e.getMessage());
        }
    }

}
```