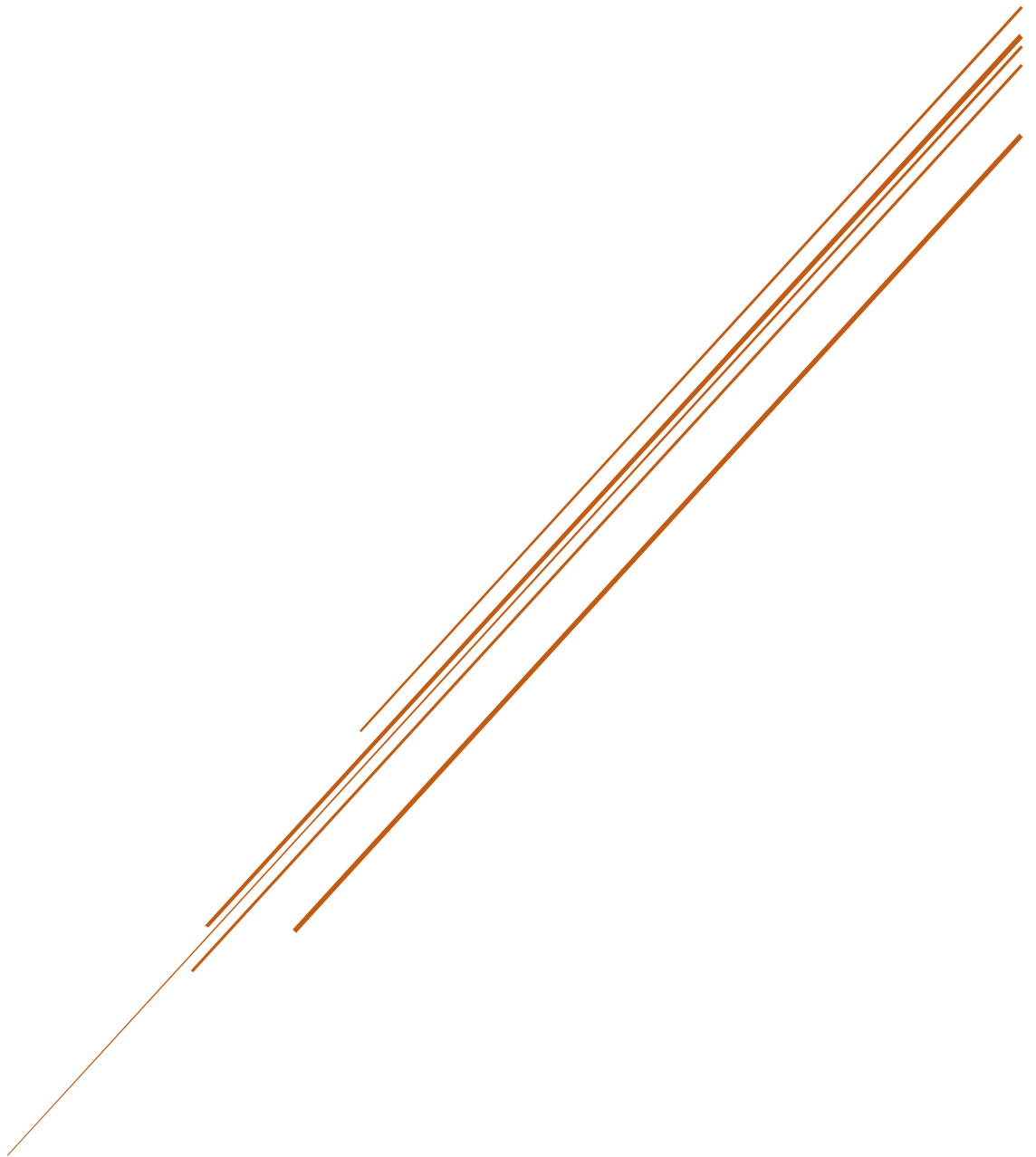


Projektmunka I.

Féléves feladat dokumentáció



Naszály Noémi

Tartalomjegyzék

Bevezetés	3
Téma	3
Az adatok forrása	3
Tervezés.....	3
EK diagram	3
Táblák és adatok	4
COUNTRIES.....	4
LANGUAGES	4
GENRES.....	4
IMDB_MOVIES	4
MOIVE_COUNTRIES	4
MOVIE_LANGUAGES	5
MOVIE_GENRES	5
IMDB_NAMES	5
JOB_CATEGORIES	5
IMDB_TITLE_PRINCIPALS	5
IMDB_RATINGS	6
A végleges táblák és kapcsolataik.....	7
Megvalósítás.....	8
A táblákba kerülő adatok előállítás az adatforrás alapján	8
A táblák feltöltése adatokkal	15
COUNTRIES.....	15
LANGUAGES	16
GENRES.....	16
IMDB_MOVIES	16
MOIVE_COUNTRIES	16
MOVIE_LANGUAGES	16
MOVIE_GENRES	16
IMDB_NAMES	16
JOB_CATEGORIES	16
IMDB_TITLE_PRINCIPALS	16
IMDB_RATINGS	16

Egyéb funkciók létrehozása	16
Lekérdezések elemzése, optimalizáció	17
Személyek és filmjeik.....	17
Elemzés	17
Optimalizáció	18
Személyek és magasságuk	19
Elemzés	19
Optimalizáció	20
NoSQL adatbázis kezelés	22
A táblák átalakítása.....	22
Az adatok átvitele MongoDB-be.....	24
Lekérdezések	25
1. Hány 1985 után készült film szerepel az adatbázisban?.....	25
2. Hány olyan film van, amiben az érintett személyek közül legalább egy az 1800-as években (vagy még annál is előbb) született? Figyeljünk a null értékekre!	25
3. Melyik film kapta a legtöbb 10-es szavazatot? Jelenítsük meg az eredeti címét, a megjelenésének évét és azt, hogy pontosan hány 10-es értékelést kapott!	25
4. Melyik 3 évben érte el a legjobb átlagot a dráma műfajú filmek értékelése az évtizedben?	26
5. Melyek a leghosszabb filmek műfajonként csoportosítva? Ezek közül is csak az első ötöt jelenítsük meg!	26
6. Ki vett részt a legtöbb, pontosan hány film elkészítésében az évtizedben?	26
7. Melyik a 3 leggyakoribb születési év és hányan születtek ezekben az években?	27
8. Milyen átlagot értek el az egyes kategóriák a top 1000 szavazó szavazatainak alapján? Rendezzük sorba a kapott értékelések alapján, majd jelenítsük meg a 3 legkedveltebb kategóriát és az elért átlagaikat!	27
9. Mely forgalmazók készítettek 50-nél több filmet az évtizedben?	27
10. Kikkel készített leggyakrabban filmet Harrison Ford az évtizedben? Csak az első három személyt jelenítsük meg!	28
Mellékletek.....	29
Táblákat létrehozó script	29
Sequence	32
Trigger	32

Bevezetés

Téma

Annyit már tudtam, hogy egy filmadatbázist szeretnék megvalósítani, ahogyan az Adatbázisok tárgyhoz készült féléves feladatomban is. Viszont az egy nagyon egyszerű adatbázis volt, kevés adattal, nekem pedig sokra volt szükségem, bonyolultabb kapcsolatokkal, felépítéssel.

Az adatok forrása

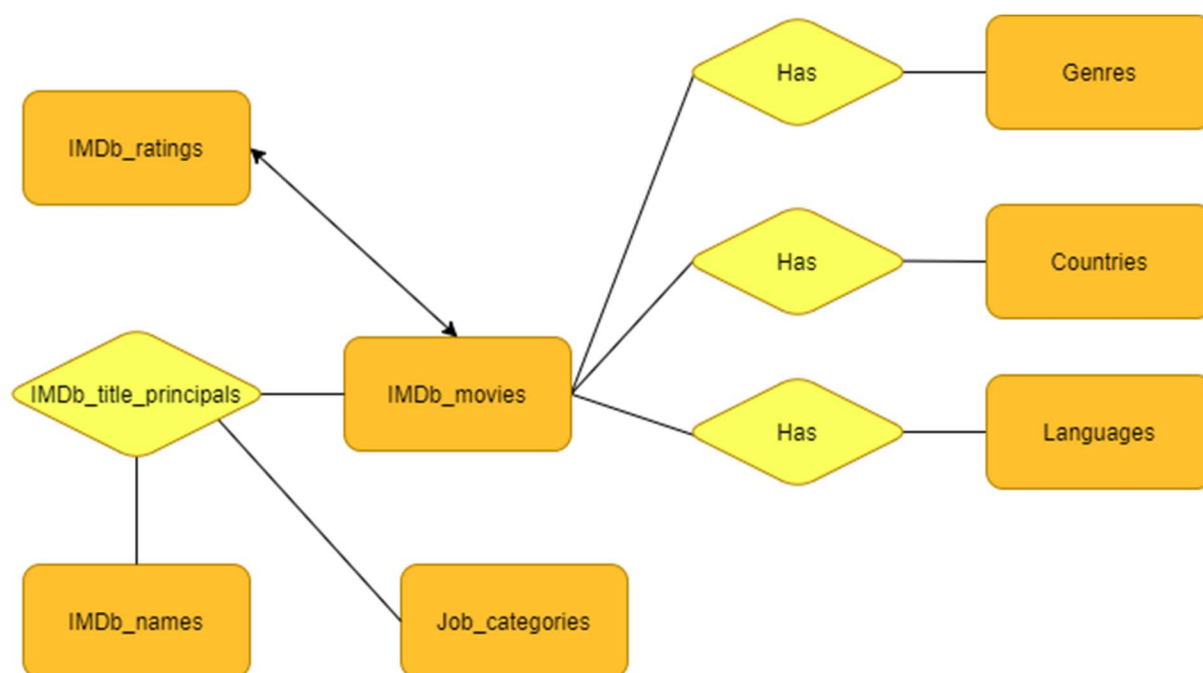
Filmekről, színészekről szóló, összetett adathalmazra volt szükségem, az interneten nagyon sokfélét találtam. A tartalom, az adatok mennyisége és az átalakításuk megvalósíthatósága alapján az alábbi adatokat használtam fel:

<https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset>

Tervezés

EK diagram

A diagram elkészítéséhez a draw.io (<https://app.diagrams.net/>) oldalt használtam.



Táblák és adatok

COUNTRIES

Azok az országok, amelyekhez kapcsolhatók a filmek. Egy filmhez több is tartozhat.

- COUNTRY_ID: Az ország azonosítója, elsődleges kulcs.
- COUNTRY_NAME: Az ország neve.

LANGUAGES

A filmek nyelve. Egy filmhez több is tartozhat.

- LANGUAGE_ID: A nyelv azonosítója, elsődleges kulcs.
- LANGUAGE_NAME: A nyelv szövegesen.

GENRES

A filmek műfaja. Egy filmhez több is tartozhat.

- GENRE_ID: A műfaj azonosítója, elsődleges kulcs.
- GENRE_NAME: A műfaj neve.

IMDB_MOVIES

Az adatbázisban tárolt filmek.

- IMDB_TITLE_ID: A film azonosítója, elsődleges kulcs.
- TITLE: A film címe.
- ORIGINAL_TITLE: A film eredeti címe.
- YEAR: A film megjelenésének éve.
- DATE_PUBLISHED: A film megjelenési dátuma.
- DURATION: A film hossza percekben.
- PRODUCTION_COMPANY: A film forgalmazója.

MOIVE_COUNTRIES

A filmeket és országokat összekapcsoló tábla. A több-több kapcsolat miatt szükséges.

- IMDB_TITLE_ID: A film azonosítója, itt idegen kulcs.
- COUNTRY_ID: Az ország azonosítója, itt idegen kulcs.

MOVIE_LANGUAGES

A filmeket és a nyelveket összekapcsoló tábla. A több-több kapcsolat miatt szükséges.

- IMDB_TITLE_ID: A film azonosítója, itt idegen kulcs.
- LANGUAGE_ID: A nyelv azonosítója, itt idegen kulcs.

MOVIE_GENRES

A filmeket és a műfajokat összekapcsoló tábla. A több-több kapcsolat miatt szükséges.

- IMDB_TITLE_ID: A film azonosítója, itt idegen kulcs.
- GENRE_ID: A műfaj azonosítója, itt idegen kulcs.

IMDB_NAMES

A filmekben szereplő, elkészítésükben résztvevő személyeket tartalmazó tábla.

- IMDB_NAME_ID: A személy azonosítója, elsődleges kulcs.
- NAME: A személy neve.
- BIRTH_NAME: A személy születési neve.
- HEIGHT: A személy magassága (centiméterben).
- DATE_OF_BIRTH: A személy születési dátuma.
- DATE_OF_DEATH: A személy elhalálzásának dátuma.

JOB_CATEGORIES

A filmek készítésében résztvevő személyek által végzett munkák kategóriáit tartalmazó tábla.

- JOB_ID: A kategória azonosítója, elsődleges kulcs.
- CATEGORY: A kategória neve.
- JOB: A kategórián belüli konkrét feladat neve, amennyiben meghatározható.

IMDB_TITLE_PRINCIPALS

A filmeket és a személyeket összekötő tábla. Tartalmazza, hogy mennyire fontos és milyen munkával vette ki a részét a készítésben.

- IMDB_TITLE_ID: A film azonosítója, itt idegen kulcs.
- ORDERING: A személy fontossága a film készítése során.
- IMDB_NAME_ID: A személy azonosítója, itt idegen kulcs.
- JOB_ID: A munka kategória azonosítója, itt idegen kulcs.

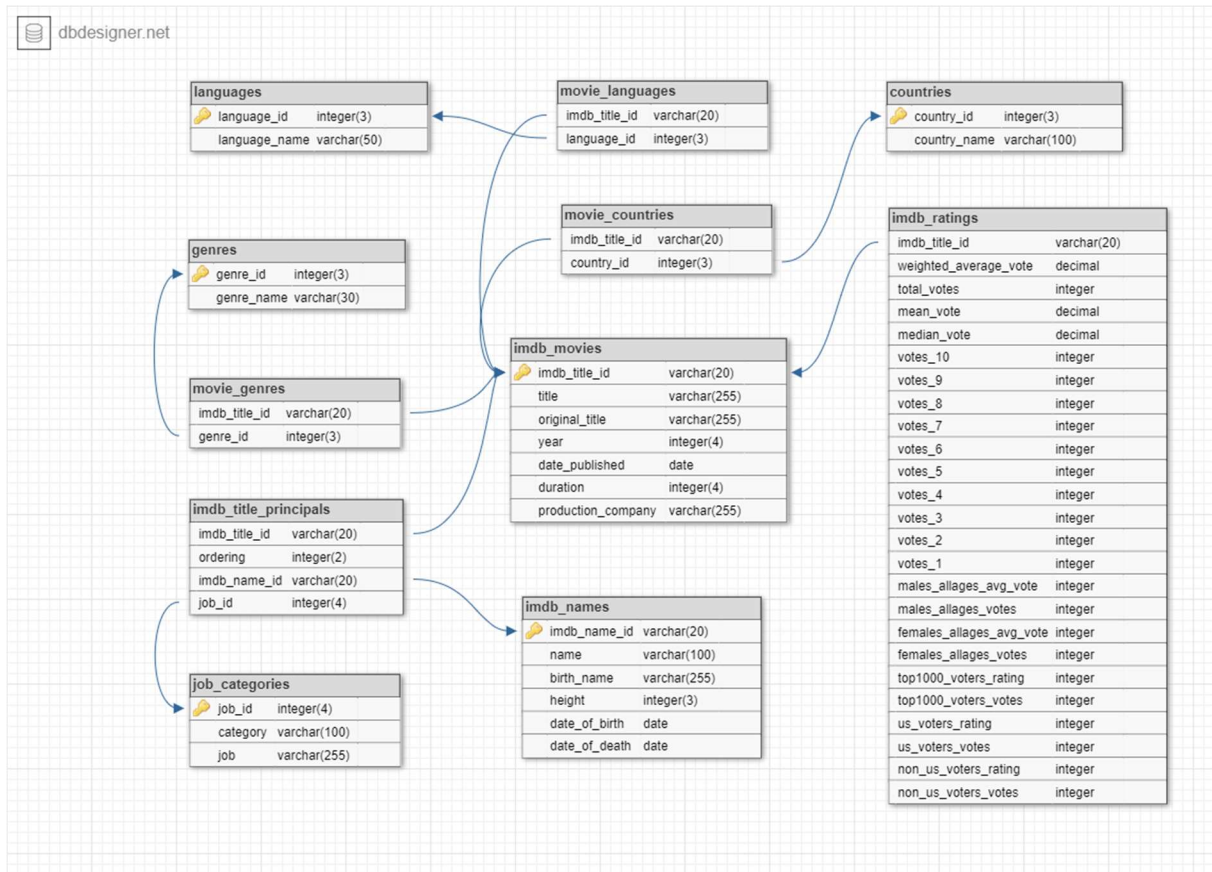
IMDB_RATINGS

A filmekhez tartozó értékeléseket tartalmazó tábla.

- IMDB_TITLE_ID: A film azonosítója, itt idegen kulcs.
- WEIGHTED_AVERAGE_VOTE: Az összes szavazat súlyozott átlaga.
- TOTAL_VOTES: Az összes szavazat darabszáma.
- MEAN_VOTE: A szavazatok átlaga.
- MEDIAN_VOTE: A szavazatok mediánja.
- VOTES_10: A 10-es értékelések darabszáma.
- VOTES_9: A 9-es értékelések darabszáma.
- VOTES_8: A 8-as értékelések darabszáma.
- VOTES_7: A 7-es értékelések darabszáma.
- VOTES_6: A 6-os értékelések darabszáma.
- VOTES_5: Az 5-ös értékelések darabszáma.
- VOTES_4: A 4-es értékelések darabszáma.
- VOTES_3: A 3-as értékelések darabszáma.
- VOTES_2: A 2-es értékelések darabszáma.
- VOTES_1: Az 1-es értékelések darabszáma.
- MALES_ALLAGES_AVG_VOTE: A férfiak által leadott szavazatok átlaga.
- MALES_ALLAGES_VOTES: A férfiak által leadott szavazatok darabszáma.
- FEMALES_ALLAGES_AVG_VOTE: A nők által leadott szavazatok átlaga.
- FEMALES_ALLAGES_VOTES: A nők által leadott szavazatok darabszáma.
- TOP1000_VOTERS_RATING: Az 1000 legaktívabb szavazó által leadott szavazatok átlaga.
- TOP1000_VOTERS_VOTES: Az 1000 legaktívabb szavazó által leadott szavazatok darabszáma.
- US_VOTERS_RATING: Az USA-ban élő szavazók által leadott szavazatok átlaga.
- US_VOTERS_VOTES: Az USA-ban élő szavazók által leadott szavazatok darabszáma.
- NON_US_VOTERS_RATING: A nem USA-ban élő szavazók által leadott szavazatok átlaga.

- **NON_US_VOTERS_VOTES:** A nem USA-ban élő szavazók által leadott szavazatok darabszáma.

A végleges táblák és kapcsolataik



Az ábra a dbdesigner.net segítségével készült.

Megvalósítás

Az megfelelő adatforrás megtalálása, a táblák és kapcsolataik megtervezése után következhetett az adatok egységesítése, átalakítása, bizonyos oszlopoknak külön táblába való kiszervezése.

A táblákba kerülő adatok előállítás az adatforrás alapján

A forrásban szereplő IMDB movies nevű fájl tartalmazza a filmek adatait. Ebben szerepel 3 olyan oszlop is, amelyet át kellett alakítanom, mert sértette az 1NF normálformát. Ráadásul ezekre az adatokra szükségem is volt, mivel az országok, a nyelvek és a műfajok szerepeltek ilyen módon az eredeti fájlban.

Először Excel segítségével eltávolítottam a felesleges oszlopokat, csak az azonosítót, az országokat, nyelveket és műfajokat tartalmazó oszlopokat tartottam meg.

	A	B	C	D
1	imdb_title_id	genre	country	language
2	tt0000009	Romance	USA	None
3	tt0000574	Biography, Crime, Drama	Australia	None
4	tt0001892	Drama	Germany, Denmark	
5	tt0002101	Drama, History	USA	English
6	tt0002130	Adventure, Drama, Fantasy	Italy	Italian
7	tt0002199	Biography, Drama	USA	English
8	tt0002423	Biography, Drama, Romance	Germany	German
9	tt0002445	Drama, History	Italy	Italian
10	tt0002452	History, War	Romania	
11	tt0002461	Drama	France, USA	English
12	tt0002646	Drama	Denmark	Danish
13	tt0002844	Crime, Drama	France	French
14	tt0003014	Drama	Sweden	
15	tt0003037	Crime, Drama	France	French
16	tt0003102	Drama	Italy	Italian
17	tt0003131	Drama, War	Belgium	French
18	tt0003165	Crime, Drama, Mystery	France	French
19	tt0003167	Drama	USA	English
20	tt0003419	Drama, Fantasy, Horror	Germany	German, English
21	tt0003471	Crime, Drama	USA	English
22	tt0003489	Adventure, Drama	Italy	Italian
23	tt0003637	Drama	Italy	
24	tt0003643	Crime, Drama, Horror	USA	English
25	tt0003657	Western	USA	English
26	tt0003740	Adventure, Drama, History	Italy	Italian
27	tt0003772	Fantasy, Drama	USA	English
28	tt0003883	Crime, Drama	France	French

Ezt csv-ként elmentve saját programmal oldottam meg a normalizálást.

Beolvastam a csv fájlt:

```
string file = "csvproghoz.csv";
StreamReader sr = new StreamReader(file);
List<string> list = new List<string>();
while (!sr.EndOfStream)
{
    list.Add(sr.ReadLine());
}
sr.Close();
```

Ezután létrehoztam egy listát, amiben már oszloponként szerepeltek az adatok. Egy listaelem egy tömb, aminek az egyes indexein lévő adatok mindig ugyanabba a típusba tartoznak:

```
List<string[]> splittedList = new List<string[]>();
foreach (var item in list)
{
    splittedList.Add(item.Split(";"));
}
```

Így az oszlopok kaptak egy indexet, ami alapján fel tudtam osztani és külön listákba gyűjteni minden 1NF-et sértő oszlopot külön-külön. Ez a lista tömbként tárolja az egyes sorokhoz (filmekhez) tartozó jellemzőket:

```
List<string[]> genres = new List<string[]>();
List<string[]> countries = new List<string[]>();
List<string[]> languages = new List<string[]>();

foreach (var item in splittedList)
{
    genres.Add(item[1].Split(","));
    countries.Add(item[2].Split(","));
    languages.Add(item[3].Split(","));
}
```

A vesszők után általában szóköz is van (pl. Biography, Crime, Drama), ami nehezíti az egyeztetést, az ismétlődések kiválogatását. Emiatt az összes listán elvégeztem az alábbi műveletet, a vesszők elhagyásának érdekében:

```
for (int i = 0; i < genres.Count; i++)
{
    for (int j = 0; j < genres[i].Length; j++)
    {
        genres[i][j] = genres[i][j].Trim();
    }
}
```

A külön táblába való kiszervezés miatt szükség volt a műfajok/országok/nyelvek ismétlődésmentes listájára. Mivel az előzőleg készített listák tömbök listái, ezért azokból nem lehetett eszerint a szempont szerint kinyerni az adatokat. Ezt a következő módon oldottam meg:

```

List<string> allGenres = new List<string>();
for (int i = 0; i < genres.Count; i++)
{
    for (int j = 0; j < genres[i].Length; j++)
    {
        allGenres.Add(genres[i][j]);
    }
}

```

A tömb (az aktuális listaelem) minden elemét hozzáadtam egy új listához. Így egy nagyon hosszú listát kaptam, amiben szerepelt az összes műfaj/nyelv/ország, sok ismétlődéssel, de felesleges szóközők nélkül. Már csak egy szűrést kellett elvégezni, és elő is állt az új tábla alapja (a genre a fejléc miatt maradt meg. Ezt a listát használtam következő lépésben a tábla előállításához, így ideje volt eltávolítani a nem szükséges elemet):

```

var filteredGenres = allGenres.Distinct().OrderBy(x=>x).ToList();
filteredGenres.Remove("genre");

```

Ezután az ismétlődésmentes listából készítettem egy olyan listát, amiből már egyszerűen létrehozható a műfajok/nyelvek/országok táblája.

```

List<string[]> tableOfGenres = new List<string[]>();
for (int i = 0; i < filteredGenres.Count; i++)
{
    string[] row = new string[2];
    row[0] = (i + 1).ToString();
    row[1] = filteredGenres[i];
    tableOfGenres.Add(row);
}

```

```

StreamWriter swGenreTable = new StreamWriter("genretable.sql");
for (int i = 0; i < tableOfGenres.Count; i++)
{
    string row =
        $"INSERT INTO genres VALUES({tableOfGenres[i][0]}, '{tableOfGenres[i][1]}')";
    swGenreTable.WriteLine(row);
}
swGenreTable.Close();

```

A három tábla elkészítése után még szükség volt a kapcsolótáblák létrehozására is. Ehhez felhasználtam azt, hogy a „splittedList”-ben található tömbök elemeinek tudtam a sorrendjét, így meg tudtam állapítani, hogy az ismétlődésmentes listából melyik elemeket tartalmazza. Az indexet ugyanúgy állítottam elő, mint a táblánál. A kapott lista alapján, amely immár

tartalmazza a film azonosítóját és a hozzájuk tartozó műfajokat/országokat/nyelveket könnyen létre tudtam hozni a szükséges kapcsolótáblákat.

```
List<string[]> id_genre_table = new List<string[]>();

for (int i = 0; i < splittedList.Count; i++)
{
    for (int j = 0; j < filteredGenres.Count; j++)
    {
        if (splittedList[i][1].Contains(filteredGenres[j]))
        {
            var id = splittedList[i][0];
            var genreid = (j + 1).ToString();
            string[] row = new string[2];
            row[0] = id;
            row[1] = genreid;
            id_genre_table.Add(row);
        }
    }
}

StreamWriter sw_movie_genre_table = new StreamWriter("movie_genres_table.sql");
for (int i = 0; i < id_genre_table.Count; i++)
{
    string row =
        $"INSERT INTO movie_genres VALUES('{id_genre_table[i][0]}', {id_genre_table[i][1]});";
    sw_movie_genre_table.WriteLine(row);
}
sw_movie_genre_table.Close();
```

Az imdb title_principals nevű fájl legfontosabb oszlopainak megtartása után egy csv fájlt készítettem belőle, hogy az adatokat tudjam használni a programomban.

Mivel a job értéke függ a category-tól, ezért sérti a 3NF normálformát.

	A	B	C	D	E
1	imdb_title_id	ordering	imdb_name_id	category	job
2	tt0000009	1	nm0063086	actress	
3	tt0000009	2	nm0183823	actor	
4	tt0000009	3	nm1309758	actor	
5	tt0000009	4	nm0085156	director	
6	tt0000574	1	nm0846887	actress	
7	tt0000574	2	nm0846894	actor	
8	tt0000574	3	nm3002376	actor	
9	tt0000574	4	nm0170118	actress	
10	tt0000574	5	nm0846879	director	
11	tt0000574	6	nm0317210	producer	producer
12	tt0000574	7	nm0425854	producer	producer
13	tt0000574	8	nm0846911	producer	producer
14	tt0000574	9	nm2421834	composer	
15	tt0000574	10	nm0675239	cinematographer	
16	tt0001892	1	nm0003425	actress	
17	tt0001892	2	nm0699637	actor	
18	tt0001892	3	nm0375839	actor	
19	tt0001892	4	nm0016799	actor	
20	tt0001892	5	nm0300487	director	
21	tt0001892	6	nm2131092	writer	screenplay
22	tt0001892	7	nm0423762	cinematographer	
23	tt0001892	8	nm0005869	cinematographer	
24	tt0001892	9	nm0282348	actor	
25	tt0001892	10	nm2325688	actress	
26	tt0002101	1	nm0306947	actress	
27	tt0002101	2	nm0801774	actress	

A beolvasás után szétválasztottam a sorokat, így megkaptam az eredeti oszlopokat. Mivel az utolsó 2 oszlop összevonásából szerettem volna az új táblát létrehozni, ezért készítettem egy segédlistát az ismétlődések kiszűrésére:

```
StreamReader reader = new StreamReader("proghoz2.csv");
List<string> lista = new List<string>();
while (!reader.EndOfStream)
{
    lista.Add(reader.ReadLine());
}
reader.Close();

List<string[]> splittedList = new List<string[]>();
for (int i = 0; i < lista.Count; i++)
{
    splittedList.Add(lista[i].Split(";"));
}

List<string> categoryPlusJob = new List<string>();
for (int i = 0; i < splittedList.Count; i++)
{
    categoryPlusJob.Add(splittedList[i][3] + ";" + splittedList[i][4]);
}
//ismétlődések szűrése
var filtered = categoryPlusJob.Distinct().OrderBy(x => x).ToList();
filtered.Remove("category;job");
```


A tábla előállításakor figyelembe kellett vennem, hogy problémát okozhatnak a különböző jelek, ezért kicseréltem őket már a létrehozása során:

```
List<string[]> jobsTable = new List<string[]>();
for (int i = 0; i < filtered.Count; i++)
{
    string[] row = new string[3];
    row[0] = (i + 1).ToString();
    row[1] = filtered[i].Split(";")[0];
    row[2] = filtered[i].Split(";")[1];
    if (row[2].Contains(@"'"))
    {
        row[2] = row[2].Replace(@"'", @"' '");
    }
    jobsTable.Add(row);
}
```

Létrehoztam a jobs nevű táblából egy olyan változatot is, ami a kategóriákat bár szűrve, de pontosvesszővel elválasztva tartalmazza. Így egyszerűbb volt hasonlítani annak a listának az elemeihez, amit még az elején hoztam létre és aminek alapján a sorokhoz (film+személy+munkája) hozzá tudtam kapcsolni az új (jobs) táblámhoz rendelt azonosítókat:

```
List<string[]> jobs=new List<string[]>() ;
for (int i = 0; i < filtered.Count; i++)
{
    string[] row = new string[2];
    row[0] = (i + 1).ToString();
    row[1] = filtered[i];
    jobs.Add(row);
}
```

A másik új tábla, amiben már szerepelnek az új (job) azonosítók alapja:

```

List<string[]> newPrincipalsTable = new List<string[]>();
for (int i = 0; i < splittedList.Count; i++)
{
    string[] a = new string[4];
    a[0] = splittedList[i][0];
    a[1] = splittedList[i][1];
    a[2] = splittedList[i][2];
    newPrincipalsTable.Add(a);
}

```

Az új title principals tábla az új jobs-ra mutató idegen kulcs elemének a beillesztése:

```

for (int i = 1; i < lista.Count; i++)
{
    var q = from x in jobs
            where categoryPlusJob[i].Equals(x[1])
            select x[0];
    var idx = q.SingleOrDefault();

    newPrincipalsTable[i][3] = idx;
}

```

Ezek után már csak a fájlokat kellett előállítanom a listák alapján:

```

StreamWriter swPrincipals = new StreamWriter("imdb_title_principals.sql");
for (int i = 1; i < newPrincipalsTable.Count; i++)
{
    string row =
        $"INSERT INTO imdb_title_principals VALUES('{newPrincipalsTable[i][0]}',
        {newPrincipalsTable[i][1]}, '{newPrincipalsTable[i][2]}', {newPrincipalsTable[i][3]});";
    swPrincipals.WriteLine(row);
}
swPrincipals.Close();

```

Itt használtam egy sequence-t, mivel az indexek kiosztását is 1-essel kezdtem. A korábban előállított jobsTable sorainak mindig a 0. indexen lévő eleme volt az azonosító, amit felhasználtam idegen kulcsként az imdb_title_principals táblában. A job_categories nevű táblában pedig biztosan így kapták meg az indexeket, így beírhattam sequence használatával is az értékeket.

```

StreamWriter swJobs = new StreamWriter("job_categories.sql");
for (int i = 0; i < jobsTable.Count; i++)
{
    string row;
    if (jobsTable[i][2] == "")
    {
        row =
            $"INSERT INTO job_categories VALUES(jobSequence.NEXTVAL, '{jobsTable[i][1]}', null);";
    }
    else
    {
        row =
            $"INSERT INTO job_categories VALUES(jobSequence.NEXTVAL, '{jobsTable[i][1]}', '{jobsTable[i][2]}');";
    }
    swJobs.WriteLine(row);
}
swJobs.Close();

```

Az IMDB_RATINGS, IMDB_MOVIES és IMDB_NAMES táblák adatainak tisztítására is saját programot írtam. Ezeknél szintén kiválasztottam az Excel segítségével a szükséges oszlopokat, csv-ként elmentettem, aztán beolvastam őket a programomban.

A beolvasás után ratings táblában javítanom kellett az üres értékeket, átírtam őket null-ra, hogy a beszúrásakor, az utolsó lépésnél már ne kelljen ezzel törődnöm.

Néhány értéknél, ami biztosan csak egész szám lehetett, (például ilyen a szavazók száma) módosítást végeztem, hogy ne legyen feleslegesen például 100.0 az érték.

Ezután már csak simán fel kellett sorolnom az összes elemet a script létrehozásához.

A names és a movies átírásakor már több akadályba ütköztem. Ügyelnem kellett a lehetséges üres vagy hibás értékekre. Hibás volt például pár helyen a dátumok mezője (nem a formátumuk, csak például szöveg szerepelt bennük), a különböző jeleket le kellett cserélnem (pl. ' -> " , így nem azt érzékelte, mintha vége lenne a VARCHAR2 mezőnek), null-ra cseréltem a dátumoknál az üres értékeket, hogy a script létrehozásánál már ez alapján tudjak következtetni arra, hogy szükség van-e a TO_DATE() függvényre.

Ezekre a hibákra nagyrészt akkor derült fény, amikor megpróbáltam futtatni a scripteket, tehát a sorokat beilleszteni. A hibás sorokról szerencsére könnyen, ránézésre meg tudtam állapítani a hiba okát, és a programomban ki tudtam küszöbölni a továbbiakban való előfordulásukat is.

A táblák feltöltése adatokkal

Az adatok előállítása és a táblákat létrehozó script futtatása után elkezdhettem feltölteni a táblákat a kész adatokkal. A saját programmal készített scripteknek hála elég volt bemásolni a létrehozott .sql-ekből a sorokat. Mivel mindegyikhez külön-külön scriptet hoztam létre, a sorrendre is figyelnem kellett, többek közt az idegen kulcsok miatt.

COUNTRIES

A movies nevű forrásfájlból kinyert országok, összesen **194** id-ország páros.

LANGUAGES

A movies nevű forrásfájlból kinyert nyelvek, összesen **267** id-nyelv páros.

GENRES

A movies nevű forrásfájlból kinyert műfajok, összesen **25** id-műfaj páros.

IMDB_MOVIES

A movies nevű forrásfájl egy része, ami eredetileg is **85855** sort tartalmazott.

MOIVE_COUNTRIES

Az egyik általam létrehozott kapcsolótábla, amibe végül **111399** sor került.

MOVIE_LANGUAGES

Az egyik általam létrehozott kapcsolótábla, amibe végül **110500** sor került.

MOVIE_GENRES

Az egyik általam létrehozott kapcsolótábla, amibe végül **177860** sor került.

IMDB_NAMES

A names nevű forrásfájl egy része, ami eredetileg is **297705** sort tartalmazott.

JOB_CATEGORIES

A title_principals nevű fájlból kiemelt új tábla, **8891** sort tartalmaz.

IMDB_TITLE_PRINCIPALS

A title_principals nevű fájl egy része, ami eredetileg 835513 sort tartalmazott. Azonban a script futtatása után csak **835493** rekordnak került sor a beszúrására, mivel nem talált idegen kulcsokat. Egy esetben a filmek közt, 19 esetben pedig a személyek közt nem találta a hivatkozott értéket. Ezek a módosításaimtól mentes, eredeti állományokban sem találhatók meg.

IMDB_RATINGS

A ratings nevű fájl egy része, ami eredetileg is **85855** sort tartalmazott.

Egyéb funkciók létrehozása

Létrehoztam egy szekvenciát és egy triggert is, amelyek a mellékletben találhatók.

jobSequence

NewMovie

Lekérdezések elemzése, optimalizáció

Személyek és filmjeik

Azokat a személyeket keressük, akiknek a neve 'A'-l kezdődik. Listázzuk ki a személyek id-ját, nevét és a filmek címét, amikben szerepeltek!

```
select imdb_name_id, name, title
from imdb_names
  inner join imdb_title_principals using(imdb_name_id)
  inner join imdb_movies using(imdb_title_id)
where name like ('A%');
```

Elemzés

A lekérdezés végrehajtási terve:

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		20	1598
NESTED LOOPS		20	1598
NESTED LOOPS		20	1598
HASH JOIN		20	1578
Access Predicates			
IMDB_NAMES.IMDB_NAME_ID=IMDB_TITLE_PRINCIPALS.IMDB_NAME_ID			
TABLE ACCESS (FULL)	IMDB_NAMES	7	582
Filter Predicates			
IMDB_NAMES.NAME LIKE 'A%'			
TABLE ACCESS (FULL)	IMDB_TITLE_PRINCIPALS	835493	994
INDEX (UNIQUE SCAN)	TITLE_PK	1	0
Access Predicates			
IMDB_TITLE_PRINCIPALS.IMDB_TITLE_ID=IMDB_MOVIES.IMDB_TITLE_ID			
TABLE ACCESS (BY INDEX ROWID)	IMDB_MOVIES	1	1

- TABLE ACCESS (FULL) - IMDB_NAMES, FILTER: Az *imdb_names* táblát full table scan-nel beolvassa, majd végrehajtja a feltételben található szűrést a *name* oszlopra.
- TABLE ACCESS (FULL) - IMDB_TITLE_PRINCIPALS: Az *imdb_title_principals* egy kapcsolótábla, ami az *imdb_names*-t és az *imdb_movies* táblát köti össze, több-több kapcsolat miatt szükséges. Full table scan-nel kiolvassa a sorokat, majd:
- HASH JOIN – NAME_ID alapján: Az *imdb_title_principals* és az *imdb_names* táblát hash join-nal összeköti.
- INDEX (UNIQUE SCAN) – TITLE_PK, ACCESS PREDICATES: Az *imdb_movies* és *imdb_title_principals* tábla összekapcsolása, az *imdb_movies* elsődleges kulcsának segítségével.
- TABLE ACCESS (BY INDEX ROWID) – IMDB_MOVIES: Az *imdb_movies* táblából szükség lesz a *title* mezőre. Mivel az előző lépésben kikérte az elsődleges kulcs (és az *imdb_title_principals* tábla *imdb_movies*-ra vonatkozó idegen kulcsa) segítségével a szükséges sorokat, ezért az *imdb_movies* tábla ezen sorait row id-k alapján el tudjuk érni.
- NESTED LOOPS: Mivel a két eredményhalmaz (*imdb_names* és *imdb_title_principals*, valamint *imdb_movies*) kicsi, ezért nested loops segítségével is gyorsan ki tudja keresni a megfelelő párokat, kapcsolásokat.

Optimalizáció

Létrehoztam 2 indexet:

- name_title_idx: összetett index az **IMDB_TITLE_PRINCIPALS** tábla *imdb_name_id* és *imdb_title_id* mezőjére
- name_idx: az **IMDB_NAMES** tábla *name* mezőjére

Ezek hatására a végrehajtási terv:

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		20	45
HASH JOIN		20	45
Access Predicates			
IMDB_TITLE_PRINCIPALS.IMDB_TITLE_ID=IMDB_MOVIES.IMDB_TITLE_ID			
NESTED LOOPS		20	45
NESTED LOOPS		20	45
STATISTICS COLLECTOR			
HASH JOIN		20	25
Access Predicates			
IMDB_NAMES.IMDB_NAME_ID=IMDB_TITLE_PRINCIPALS.IMDB_NAME_ID			
NESTED LOOPS		20	25
STATISTICS COLLECTOR			
TABLE ACCESS (BY INDEX ROWID BATCHED)	IMDB_NAMES	7	11
INDEX (RANGE SCAN)	NAME_IDX	7	3
Access Predicates			
IMDB_NAMES.NAME LIKE 'A%'			
Filter Predicates			
IMDB_NAMES.NAME LIKE 'A%'			
INDEX (RANGE SCAN)	NAME_TITLE_IDX	3	2
Access Predicates			
IMDB_NAMES.IMDB_NAME_ID=IMDB_TITLE_PRINCIPALS.IMDB_NAME_ID			
INDEX (FAST FULL SCAN)	NAME_TITLE_IDX	3	2
Access Predicates			
IMDB_TITLE_PRINCIPALS.IMDB_TITLE_ID=IMDB_MOVIES.IMDB_TITLE_ID			
INDEX (UNIQUE SCAN)	TITLE_PK	1	0
Access Predicates			
IMDB_TITLE_PRINCIPALS.IMDB_TITLE_ID=IMDB_MOVIES.IMDB_TITLE_ID			
TABLE ACCESS (BY INDEX ROWID)	IMDB_MOVIES	1	1
TABLE ACCESS (FULL)	IMDB_MOVIES	1	1

- INDEX (RANGE SCAN) - NAME_IDX: A létrehozott name_idx segítségével csak azokat a sorokat keresi ki, amelyekre teljesül a feltétel.
- TABLE ACCESS (BY INDEX ROWID BATCHED) – IMDB_NAMES: Az *imdb_names* táblából csak azokat a sorokat olvassa ki, amelyek megfelelnek az index (range scan) után visszakapott sorokkal.
- INDEX (RANGE SCAN) – NAME_TITLE_IDX: Az *imdb_title_principals* táblából a létrehozott index segítségével kikeresi a megfelelő sorokat, összekapcsolja a szűrt *imdb_names* táblát az *imdb_title_principals* táblával. Mivel ez egy összetett index, ezért nincs is szükség az *imdb_title_principals* táblára, hiszen ez az index tartalmazza a title_id-t is, aminek a segítségével megtalálhatjuk a filmeket az *imdb_movies* táblában.
- NESTED LOOPS: Mivel a kapott adathalmazok ismét kicsik, ezért nested loops-szal oldja meg a join-olást.
- INDEX (FAST FULL SCAN) – NAME_TITLE_IDX: Az *imdb_title_principals* táblából a létrehozott index segítségével kikeresi a megfelelő sorokat.
- HASH JOIN: imdb_name_id alapján összekapcsolja az *imdb_names* és az *imdb_title_principals* táblát.
- INDEX (UNIQUE SCAN) – TITLE_PK: Az *imdb_movies* tábla megfelelő sorait most is az elsődleges kulcs segítségével keresi ki, majd:
- TABLE ACCESS (BY INDEX ROWID) – IMDB_MOVIES: Az *imdb_movies* táblát az előbb kapott (index) row id alapján éri el.

- NESTED LOOPS: A join-t nested loops-szal végzi el.
- TABLE ACCESS (FULL) – IMDB_MOVIES: Az *imdb_movies* táblából szükség lesz a film címére is.
- HASH JOIN: *imdb_title_id* alapján összekapcsolja az *imdb_movies* és az *imdb_title_principals* táblát.

Személyek és magasságuk

Listázzuk a személy id-ját, a magasságát, a fontosságát az adott produkcióban, továbbá azt, hogy hány olyan személy található még az adatbázisban, aki ugyanolyan magas, mint ő!

```
select imdb_name_id, height, pcs, ordering
from imdb_names inner join (
  select height, count(imdb_name_id) pcs from imdb_names where height is not null group by height ) using(height)
inner join imdb_title_principals using(imdb_name_id);
```

Elemzés

A lekérdezés végrehajtási terve:

OPERATION	OBJECT_NAME	CARDINALITY	COST	
SELECT STATEMENT			18875	2162
HASH JOIN			18875	2162
Access Predicates				
IMDB_NAMES.IMDB_NAME_ID=IMDB_TITLE_PRINCIPALS.IMDB_NAME_ID				
HASH JOIN			6786	1165
Access Predicates				
IMDB_NAMES.HEIGHT=FROM_QUERY_002.HEIGHT				
VIEW			115	584
HASH (GROUP BY)			115	584
TABLE ACCESS (FULL)	IMDB_NAMES	44681	44681	582
Filter Predicates				
HEIGHT IS NOT NULL				
TABLE ACCESS (FULL)	IMDB_NAMES	44681	44681	582
Filter Predicates				
IMDB_NAMES.HEIGHT IS NOT NULL				
TABLE ACCESS (FULL)	IMDB_TITLE_PRINCIPALS	835493	835493	554

- TABLE ACCESS (FULL) – IMDB_NAMES, FILTER – HEIGHT IS NOT NULL: Full table scan-eléri az *imdb_names* táblát, de csak azokat a sorokat, ahol a height nem null értékkel szerepel.
- HASH GROUP BY: Megcsinálja a csoportosítást.
- VIEW: Elkészíti a nézetet, ami az *imdb_names* táblát használja (group by miatt, ebben a nézetben magasságok szerepelnek és darabszámok).
- TABLE ACCESS (FULL) – IMDB_NAMES, FILTER – HEIGHT IS NOT NULL: Ismét szükség van az *imdb_names* táblára, mert ebből szeretnénk az adatok egy részét kinyerni, de csak azokat, ahol nem null a height értéke.
- HASH JOIN: Az előbb előállított view és a (szűrt) *imdb_names* tábla összekapcsolása a height alapján. Ezt hash join-nal teszi, mivel a view viszonylag kicsi a táblához képest.
- TABLE ACCESS (FULL) – IMDB_TITLE_PRINCIPALS: Szükség van az *imdb_title_principals* táblára is, mivel ennek az egyik mezőjét is lekérdeztük, ami nem az *imdb_name_id* idegen kulcs, ami megtalálható az *imdb_names* táblában is).
- HASH JOIN: name_id alapján *imdb_title_principals* és *imdb_names* összekapcsolása.

Optimalizáció

Létrehoztam 2 indexet:

- height_idx: az IMDB_NAMES tábla **height** mezőjére
- prin_name_fk_idx: az IMDB_TITLE_PRINCIPALS tábla **imdb_name_id** mezőjére

Ezek hatására a végrehajtási terv:

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT			1607
HASH JOIN		16875	1607
Access Predicates			
IMDB_NAMES.IMDB_NAME_ID=IMDB_TITLE_PRINCIPALS.IMDB_NAME_ID			
NESTED LOOPS		16875	1607
NESTED LOOPS			
STATISTICS COLLECTOR			
HASH JOIN		6786	611
Access Predicates			
IMDB_NAMES.HEIGHT=from\$_subquery\$_602.HEIGHT			
NESTED LOOPS		6786	611
STATISTICS COLLECTOR			
VIEW		115	29
HASH (GROUP BY)		115	29
INDEX (FAST FULL SCAN)	HEIGHT_IDX	44681	27
Filter Predicates			
HEIGHT IS NOT NULL			
TABLE ACCESS (BY INDEX ROWID BATCHED)	IMDB_NAMES	58	582
INDEX (RANGE SCAN)	HEIGHT_IDX		
Access Predicates			
IMDB_NAMES.HEIGHT=from\$_subquery\$_082.HEIGHT			
Filter Predicates			
IMDB_NAMES.HEIGHT IS NOT NULL			
TABLE ACCESS (FULL)	IMDB_NAMES	44681	582
Filter Predicates			
IMDB_NAMES.HEIGHT IS NOT NULL			
INDEX (RANGE SCAN)	PRIN_NAME_FK_IDX		
Access Predicates			
IMDB_NAMES.IMDB_NAME_ID=IMDB_TITLE_PRINCIPALS.IMDB_NAME_ID			
TABLE ACCESS (BY INDEX ROWID)	IMDB_TITLE_PRINCIPALS	3	994
TABLE ACCESS (FULL)	IMDB_TITLE_PRINCIPALS	835493	994

- INDEX (FAST FULL SCAN) – HEIGHT_IDX, FILTER – HEIGHT IS NOT NULL: A létrehozott indexet használva csak azokat a sorokat választja ki, ahol az index nem null. Ennek a segítségével nem szükséges az egész *imdb_names* táblát elérni, így is meg tudja határozni a darabszámokat magasságonként.
- HASH GROUP BY: Megcsinálja a csoportosítást.
- VIEW: Elkészül a nézet.
- INDEX RANGE SCAN – HEIGHT_IDX: Az index alapján kikeresi a megfelelő sorokat az *imdb_names* táblából.
- TABLE ACCESS (BY INDEX ROWID BATCHED) – IMDB_NAMES: Az előzőleg kikeresett indexek alapján kiválasztja az *imdb_names* táblából a megfelelő sorokat.
- NESTED LOOPS: Nested loops-t használ a tábla és a nézet összekapcsolásához.
- TABLE ACCESS (FULL) – IMDB_NAMES: Az *imdb_names* táblából szükségünk lesz adatokra, ezért full table scan-t használ. Csak azokat választja ki, ahol a height nem null, mivel majd csak azokhoz fog tudni kapcsolni darabszámokat később, a nézet alapján.
- HASH JOIN: *imdb_names* és a nézet összekapcsolása height mező alapján. Hash join-nal történik, mivel a nézet sorainak száma csekély a tábla sorainak számához képest.
- INDEX (RANGE SCAN) – PRIN_NAME_FK_IDX, ACCESS PREDICATES: *imdb_title_principals* táblából kikeresi azokat a sorokat az index segítségével, amikre szükség lesz (ami kapcsolható az *imdb_names* tábla valamely sorához).

- TABLE ACCESS (BY INDEX ROW ID) – IMDB_TITLE_PRINCIPALS: az *imdb_title_principals* tábla megfelelő sorait az előbb kapott indexek alapján kiolvassa, majd:
- NESTED LOOPS: Nested loops-t használ a join-oláshoz, az *imdb_names* tábla és az *imdb_title_principals* tábla összekötéséhez.
- TABLE ACCESS FULL – IMDB_TITLE_PRINCIPALS: az *imdb_title_principals* tábla sorait kiolvassa.
- HASH JOIN: Az *imdb_name_id* alapján összeköti az *imdb_names* és az *imdb_title_principals* táblát.

NoSQL adatbázis kezelés

Az adatbázisom dokumentum jellegű felépítése és a JSON fájlok egyszerű importálása miatt a MongoDB-t választottam.

A táblák átalakítása

<div><div>(10) ObjectId("607ee3fbda896a6efab58b24")</div><div><div><div>_id</div><div>imdb_title_id</div><div>title</div><div>original_title</div><div>year</div><div>date_published</div><div>duration</div><div>production_company</div><div>names</div><div>imdb_name_id</div><div>name</div><div>birth_name</div><div>height</div><div>date_of_birth</div><div>date_of_death</div></div><div>[1]</div><div>[2]</div><div>[3]</div><div>[4]</div><div>[5]</div><div>[6]</div><div>[7]</div><div>[8]</div><div>[9]</div><div>genres</div><div>[0]</div><div>[1]</div><div>imdb_rating</div><div>weighted_average_vote</div><div>total_votes</div><div>mean_vote</div><div>median_vote</div><div>votes_10</div><div>votes_9</div><div>votes_8</div><div>votes_7</div><div>votes_6</div><div>votes_5</div><div>votes_4</div><div>votes_3</div><div>votes_2</div><div>votes_1</div><div>males_allages_avg_vote</div><div>males_allages_votes</div><div>females_allages_avg_vote</div><div>females_allages_votes</div><div>top_1000_voters_rating</div><div>top_1000_voters_votes</div><div>us_voters_rating</div><div>us_voters_votes</div><div>non_us_voters_rating</div><div>non_us_voters_votes</div></div></div>	<div>{ 11 fields }</div> <div>ObjectId("607ee3fbda896a6efab58b24")</div> <div>tt0097576</div> <div>Indiana Jones e l'ultima crociata</div> <div>Indiana Jones and the Last Crusade</div> <div>1989</div> <div>1989-10-06 00:00:00.000Z</div> <div>127</div> <div>Paramount Pictures</div> <div>[10 elements]</div> <div>{ 6 fields }</div> <div>nm0000148</div> <div>Harrison Ford</div> <div>Harrison Ford</div> <div>183</div> <div>1942-07-13 00:00:00.000Z</div> <div>null</div> <div>{ 6 fields }</div> <div>{ 6 fields }</div> <div>{ 6 fields }</div> <div>{ 6 fields }</div> <div>{ 6 fields }</div> <div>{ 6 fields }</div> <div>{ 6 fields }</div> <div>{ 6 fields }</div> <div>{ 6 fields }</div> <div>{ 6 fields }</div> <div>[2 elements]</div> <div>Action</div> <div>Adventure</div> <div>{ 24 fields }</div> <div>8.2</div> <div>676976</div> <div>8.2</div> <div>8</div> <div>123667</div> <div>167019</div> <div>218816</div> <div>109567</div> <div>33269</div> <div>11776</div> <div>4820</div> <div>2513</div> <div>1710</div> <div>3819</div> <div>8.3</div> <div>436676</div> <div>8.1</div> <div>67565</div> <div>8.2</div> <div>871</div> <div>8.4</div> <div>132224</div> <div>8.2</div> <div>274062</div>	<div>Object</div> <div>ObjectId</div> <div>String</div> <div>String</div> <div>String</div> <div>Int32</div> <div>Date</div> <div>Int32</div> <div>String</div> <div>Array</div> <div>Object</div> <div>String</div> <div>String</div> <div>Int32</div> <div>Date</div> <div>Null</div> <div>Object</div> <div>Object</div> <div>Object</div> <div>Object</div> <div>Object</div> <div>Object</div> <div>Object</div> <div>Object</div> <div>Object</div> <div>Object</div> <div>Array</div> <div>String</div> <div>String</div> <div>Object</div> <div>Double</div> <div>Int32</div> <div>Double</div> <div>Int32</div> <div>Int32</div> <div>Int32</div> <div>Int32</div> <div>Int32</div> <div>Int32</div> <div>Int32</div> <div>Int32</div> <div>Int32</div> <div>Int32</div> <div>Double</div> <div>Int32</div> <div>Double</div> <div>Int32</div> <div>Double</div> <div>Int32</div> <div>Double</div> <div>Int32</div> <div>Double</div>
---	--	---

Az alábbi táblák adataira volt szükségem az átalakításhoz:

- IMDB_MOVIES
- IMDB_NAMES
- IMDB_TITLE_PRINCIPALS
- IMDB_RATINGS
- MOVIE_GENRES
- GENRES

A táblák átalakításához saját programot írtam. Az adatok adatforrás alapján való előállítás után keletkezett listákat itt is fel tudtam használni.

A filmekkel kezdtem, mivel abba szerettem volna beágyazni a többi adatot. Készítettem egy osztályt:

```

public class Movie
{
    4 references
    public string imdb_title_id { get; set; }
    1 reference
    public string title { get; set; }
    1 reference
    public string original_title { get; set; }
    2 references
    public int? year { get; set; }
    2 references
    public DateTime? date_published { get; set; }
    1 reference
    public int duration { get; set; }
    1 reference
    public string production_company { get; set; }
    2 references
    public List<Person> names { get; set; }
    2 references
    public List<string> genres { get; set; }
    2 references
    public Rating imdb_rating { get; set; }

    1 reference
    public Movie()
    {
        names = new List<Person>();
        genres = new List<string>();
    }
}

```

Person és Rating: tulajdonságaik megegyeznek az eredeti adatbázisban található oszlopnevekkel.

Ahogy a képen látható, a Movie nevű osztályban is megtalálható minden tulajdonság, ami a relációs adatbázisban volt, kiegészítve néhány újjal:

- A **names** tartalmazza az **imdb_names**-ből kinyert adatokat, hogy mely személyek érintettek a film elkészítésében. Ennek előállításához használnom kellett az **imdb_title_principals** táblát is, ami egy kapcsolótábla (több-több kapcsolat). Így ez a mező *beágyazott dokumentumok tömbje* lett a MongoDB-ben.
- A **genres** a műfajokat tárolja, a **genres** tábla alapján. Mivel itt is több-több kapcsolat áll fent, ezért szükség volt a **movie_genres** nevű kapcsolótáblára, ami alapján hozzá tudtam adni a film műfajaihoz a megfelelő műfajokat. A genres mező egy *tömb* a MongoDB-ben.
- Az **imdb_rating** az értékeléseket tartalmazza. Ezt egyszerűbben, egy-egy kapcsolat alapján nyertem ki az **imdb_movies** imdb_title_id-ja és az **imdb_ratings** szintén imdb_title_id nevű idegen kulcsa alapján. Így egy *beágyazott dokumentum* keletkezett.

Ezek után egyszerű metódusok segítségével (amelyek már korábban előállították a megfelelő adathalmazt a forrásból, és amelyek hozzáadták a Movie osztály megfelelő példányához az ahhoz tartozó adatokat) létrehoztam egy Movie-kból álló listát.

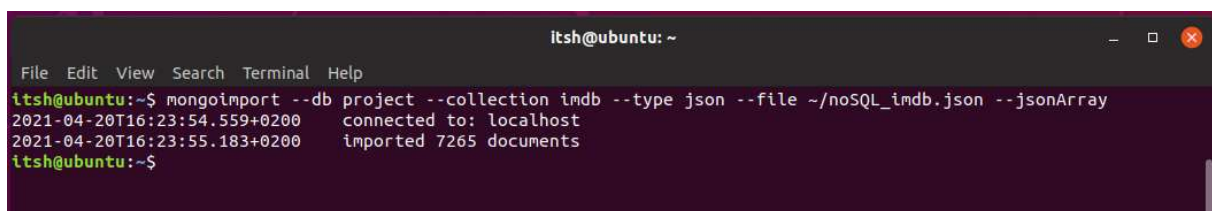
A lista csak az 1980-89 közötti filmeket tartalmazza. Ez összesen 7265 db.

Mivel a MongoDB a JSON formátumú fájlok importálásával könnyen elboldogul, valamint a programom által előállított lista alapján szintén könnyen előállítható a JSON fájl, így ezt a megoldást választottam:

```
static void Json(List<Movie> movies)
{
    string json = JsonSerializer.Serialize(movies);
    File.WriteAllText("noSQL_imdb.json", json);
}
```

Az adatok átvitele MongoDB-be

A kapott JSON fájlt a mongoimport parancs segítségével vittem át a noSQL adatbázisba:



```
itsh@ubuntu: ~
File Edit View Search Terminal Help
itsh@ubuntu:~$ mongoimport --db project --collection imdb --type json --file ~/noSQL_imdb.json --jsonArray
2021-04-20T16:23:54.559+0200 connected to: localhost
2021-04-20T16:23:55.183+0200 imported 7265 documents
itsh@ubuntu:~$
```

A dátum formátumot string-ként tárolta el, ezért át kellett alakítanom, hogy felismerje és dátumként kezelje.

Az egyes dokumentumokban (film) a *date_published* mezőt, valamint minden személynél (beágyazott dokumentumok tömbje a filmen belül) a születésének és a halálának dátumát (*date_of_birth*, *date_of_death*) tartalmazó mezőket kellett átalakítanom. Ezt az alábbi módon vittem véghez:

```
db.getCollection('imdb').find({})
  .forEach(
    function(item)
    {
      if(typeof(item.date_published)=="string")
      {
        item.date_published = new ISODate(item.date_published);
        db.getCollection('imdb').save(item);
      }
    }
  );
```

```

db.getCollection('imdb').find({}).forEach(function(item){
    item.names.forEach(function(person){
        if(person!=null){
            if(typeof(person.date_of_birth)=="string"){
                person.date_of_birth=new ISODate(person.date_of_birth);
            }
        }
    })
    db.getCollection('imdb').save(item)
})

```

```

db.getCollection('imdb').find({}).forEach(function(item){
    item.names.forEach(function(person){
        if(person!=null){
            if(typeof(person.date_of_death)=="string"){
                person.date_of_death=new ISODate(person.date_of_death);
            }
        }
    })
    db.getCollection('imdb').save(item)
})

```

Lekérdezések

1. Hány 1985 után készült film szerepel az adatbázisban?

```
db.getCollection('imdb').find({"year": {$gt:1985}}).count();
```

3380

2. Hány olyan film van, amiben az érintett személyek közül legalább egy az 1800-as években (vagy még annál is előbb) született? Figyeljünk a null értékekre!

```
db.getCollection('imdb').find({"names.date_of_birth": {$lt: new Date('1900-01-01'), $ne: null}}).count();
```

50

3. Melyik film kapta a legtöbb 10-es szavazatot? Jelenítsük meg az eredeti címét, a megjelenésének évét és azt, hogy pontosan hány 10-es értékelést kapott!

```

db.getCollection('imdb').find({}, {_id:0, "original_title":1, "year":1, "imdb_rating.votes_10":1})
.sort({"imdb_rating.votes_10":-1})
.limit(1);

```

Key	Value	Type
original_title	{ 3 fields }	Object
year	Star Wars: Episode V - The Empire Strikes Back	String
imdb_rating	1980	Int32
votes_10	{ 1 field }	Object
	391978	Int32

4. Melyik 3 évben érte el a legjobb átlagot a dráma műfajú filmek értékelése az évtizedben?

```
db.getCollection('imdb').aggregate([
  {$match: {"genres":"Drama"}},
  {$group: {_id:"$year", avgRating:{$avg:"$imdb_rating.weighted_average_vote"}}},
  {$sort: {avgRating:-1}},
  {$limit:3}
]);
```

Key	Value	Type
▼ (1) 1982	{ 2 fields }	Object
_id	1982	Int32
avgRating	6.55108359133127	Double
▼ (2) 1981	{ 2 fields }	Object
_id	1981	Int32
avgRating	6.51490066225166	Double
▼ (3) 1983	{ 2 fields }	Object
_id	1983	Int32
avgRating	6.43825301204819	Double

5. Melyek a leghosszabb filmek műfajonként csoportosítva? Ezek közül is csak az első ötöt jelenítsük meg!

```
db.getCollection('imdb').aggregate([
  {$unwind : "$genres"},
  {$group: {_id:"$genres", maxDuration:{$max:"$duration"}}},
  {$sort: {maxDuration:-1}},
  {$limit: 5}
]);
```

Key	Value	Type
▼ (1) Romance	{ 2 fields }	Object
_id	Romance	String
maxDuration	580	Int32
▼ (2) History	{ 2 fields }	Object
_id	History	String
maxDuration	580	Int32
▼ (3) Drama	{ 2 fields }	Object
_id	Drama	String
maxDuration	580	Int32
▼ (4) Thriller	{ 2 fields }	Object
_id	Thriller	String
maxDuration	335	Int32
▼ (5) War	{ 2 fields }	Object
_id	War	String
maxDuration	323	Int32

6. Ki vett részt a legtöbb, pontosan hány film elkészítésében az évtizedben?

```
db.getCollection('imdb').aggregate([
  {$unwind: "$names"},
  {$group: {_id:"$names.name", moviesCount:{$sum:1}}},
  {$sort: {moviesCount:-1}},
  {$limit: 1}
]);
```

Key	Value	Type
▼ (1) Yoram Globus	{ 2 fields }	Object
_id	Yoram Globus	String
moviesCount	91.0	Double

7. Melyik a 3 leggyakoribb születési év és hányan születtek ezekben az években?

```
db.getCollection('imdb').aggregate([
  {$unwind: "$names"},
  {$match: {"names.date_of_birth":{$ne:null}}},
  {$group: {_id: {$year:"$names.date_of_birth"}, count:{$sum:1}}},
  {$sort: {count:-1}},
  {$limit: 3}
])
```

Key	Value	Type
▼ (1) 1946	{ 2 fields }	Object
_id	1946	Int32
count	1633.0	Double
▼ (2) 1950	{ 2 fields }	Object
_id	1950	Int32
count	1450.0	Double
▼ (3) 1947	{ 2 fields }	Object
_id	1947	Int32
count	1395.0	Double

8. Milyen átlagot értek el az egyes kategóriák a top 1000 szavazó szavazatainak alapján? Rendezzük sorba a kapott értékelések alapján, majd jelenítsük meg a 3 legkedveltebb kategóriát és az elért átlagaikat!

```
db.getCollection('imdb').aggregate([
  {$unwind:"$genres"},
  {$group: {_id:"$genres", top1000sRating:{$avg:"$imdb_rating.top_1000_voters_rating"}}},
  {$sort: {top1000sRating:-1}},
  {$limit: 3}
])
```

Key	Value	Type
▼ (1) Documentary	{ 2 fields }	Object
_id	Documentary	String
top1000sRating	6.3	Double
▼ (2) Biography	{ 2 fields }	Object
_id	Biography	String
top1000sRating	5.85955056179775	Double
▼ (3) History	{ 2 fields }	Object
_id	History	String
top1000sRating	5.57158469945355	Double

9. Mely forgalmazók készítettek 50-nél több filmet az évtizedben?

```
db.getCollection('imdb').aggregate([
  {$match: {"production_company":{$ne:""}}},
  {$group: {_id:"$production_company", count:{$sum:1}}},
  {$sort: {count:-1}},
  {$match: {count:{$gt:50}}}
])
```

Key	Value	Type
▼ (1) Paramount Pictures	{ 2 fields }	Object
_id	Paramount Pictures	String
count	92.0	Double
▼ (2) Columbia Pictures	{ 2 fields }	Object
_id	Columbia Pictures	String
count	85.0	Double
▼ (3) Universal Pictures	{ 2 fields }	Object
_id	Universal Pictures	String
count	74.0	Double
▼ (4) Mosfilm	{ 2 fields }	Object
_id	Mosfilm	String
count	68.0	Double
▼ (5) Warner Bros.	{ 2 fields }	Object
_id	Warner Bros.	String
count	61.0	Double
▼ (6) Shaw Brothers	{ 2 fields }	Object
_id	Shaw Brothers	String
count	55.0	Double

10. Kikkel készített leggyakrabban filmet Harrison Ford az évtizedben? Csak az első három személyt jelenítsük meg!

```
db.getCollection('imdb').aggregate([
  {$match: {"names.name": "Harrison Ford"}},
  {$unwind: "$names"},
  {$group: {_id: "$names.name", count: {$sum: 1}}},
  {$sort: {count: -1}},
  {$match: {_id: {"$ne": "Harrison Ford"}}},
  {$limit: 3}
])
```

▼ (1) George Lucas
_id
count
▼ (2) Lawrence Kasdan
_id
count
▼ (3) John Williams
_id
count

{ 2 fields }
George Lucas
5.0
{ 2 fields }
Lawrence Kasdan
3.0
{ 2 fields }
John Williams
3.0

Object
String
Double
Object
String
Double
Object
String
Double

Mellékletek

Táblákat létrehozó script

```
CREATE TABLE countries(  
    country_id NUMBER(3),  
    country_name varchar2(100),  
    CONSTRAINT c_pk PRIMARY KEY(country_id)  
);
```

```
CREATE TABLE languages(  
    language_id NUMBER(3),  
    language_name VARCHAR2(50),  
    CONSTRAINT l_pk PRIMARY KEY(language_id)  
);
```

```
CREATE TABLE genres(  
    genre_id NUMBER(3),  
    genre_name VARCHAR2(30),  
    CONSTRAINT g_pk PRIMARY KEY(genre_id)  
);
```

```
CREATE TABLE imdb_movies(  
    imdb_title_id VARCHAR2(20),  
    title VARCHAR2(255),  
    original_title VARCHAR2(255),  
    year NUMBER(4),  
    date_published DATE,  
    duration NUMBER(4),  
    production_company VARCHAR2(255),  
    CONSTRAINT title_pk PRIMARY KEY(imdb_title_id)
```

```
);
```

```
CREATE TABLE movie_countries(  
    imdb_title_id VARCHAR2(20) REFERENCES  
imdb_movies(imdb_title_id),  
    country_id NUMBER(3) REFERENCES countries(country_id)  
);
```

```
CREATE TABLE movie_languages(  
    imdb_title_id VARCHAR2(20) REFERENCES  
imdb_movies(imdb_title_id),  
    language_id NUMBER(3) REFERENCES languages(language_id)  
);
```

```
CREATE TABLE movie_genres(  
    imdb_title_id VARCHAR2(20) REFERENCES  
imdb_movies(imdb_title_id),  
    genre_id NUMBER(3) REFERENCES genres(genre_id)  
);
```

```
CREATE TABLE imdb_names(  
    imdb_name_id VARCHAR2(20),  
    name VARCHAR2(100),  
    birth_name VARCHAR2(255),  
    height NUMBER(3),  
    date_of_birth DATE,  
    date_of_death DATE,  
    CONSTRAINT name_pk PRIMARY KEY(imdb_name_id)  
);
```

```

CREATE TABLE job_categories(
    job_id NUMBER(4),
    category VARCHAR2(100),
    job VARCHAR2(255),
    CONSTRAINT job_pk PRIMARY KEY(job_id)
);

```

```

CREATE TABLE imdb_title_principals(
    imdb_title_id VARCHAR2(20) REFERENCES
imdb_movies(imdb_title_id),
    ordering NUMBER(2),
    imdb_name_id VARCHAR2(20) REFERENCES imdb_names(imdb_name_id),
    job_id NUMBER(4) REFERENCES job_categories(job_id)
);

```

```

CREATE TABLE imdb_ratings(
    imdb_title_id VARCHAR2(20) REFERENCES
imdb_movies(imdb_title_id),
    weighted_average_vote NUMBER,
    total_votes NUMBER,
    mean_vote NUMBER,
    median_vote NUMBER,
    votes_10 NUMBER,
    votes_9 NUMBER,
    votes_8 NUMBER,
    votes_7 NUMBER,
    votes_6 NUMBER,
    votes_5 NUMBER,
    votes_4 NUMBER,
    votes_3 NUMBER,
    votes_2 NUMBER,

```



```

        votes_1 NUMBER,
        males_allages_avg_vote NUMBER,
        males_allages_votes NUMBER,
        females_allages_avg_vote NUMBER,
        females_allages_votes NUMBER,
        top1000_voters_rating NUMBER,
        top1000_voters_votes NUMBER,
        us_voters_rating NUMBER,
        us_voters_votes NUMBER,
        non_us_voters_rating NUMBER,
        non_us_voters_votes NUMBER
    );

```

Sequence

```

CREATE SEQUENCE jobSequence
START WITH 1
INCREMENT BY 1
NOCACHE;

```

Trigger

```

SET SERVEROUTPUT ON;
SET VERIFY OFF;
CREATE TRIGGER NewMovie
AFTER INSERT ON imdb_movies
FOR EACH ROW
BEGIN
    dbms_output.put_line('New movie ' || :new.title || ' inserted. ');
END;

```