

# IMDB System

## Obligatorisk opgave

---



<b>Filer:</b>	<b>3</b>
<b>Forord:</b>	<b>3</b>
<b>DB Diagram:</b>	<b>3</b>
<b>DB Design Beskrivelse:</b>	<b>4</b>
Table overblik og relationer:	5
MovieBases:	5
TitleTypes:	5
Genres:	5
MovieGenres:	5
Persons:	6
Professions:	6
PersonalCareers:	6
KnownForTitles:	6
Writers:	7
Directors:	7
'Non-Standard' data Columns:	7
Normalizations-form, mål og hvorfor:	7
Normalizations afvigelser, hvorfor:	8
Table Indexes:	8
Views (hvis nogen og formål):	9
Funktioner (hvis nogen og formål):	9
Stored Procedures(hvis nogen og formål):	9
Custom Roles (hvis nogen og formål):	13
<b>SQL-Statements og/eller funktioner til importering af .tsv:</b>	<b>13</b>
CsvLoader:	14
<tsvFileName>Processor:	14
<tsvFileName>Records:	16
Program:	16
<b>DDL/"Schema" i SQL-statements:</b>	<b>18</b>
<b>SQL-Injection beskyttelse? Hvordan?</b>	<b>19</b>
<b>Refleksion:</b>	<b>20</b>

## Filer:

[IMDbLib](#) // Library med DB oprettelse og Services  
[IMDbCons/converter](#) // Console, for convertering af .tsv filer  
[IMDbREST](#) // RESTful API, <entity>Controller setup

## Forord

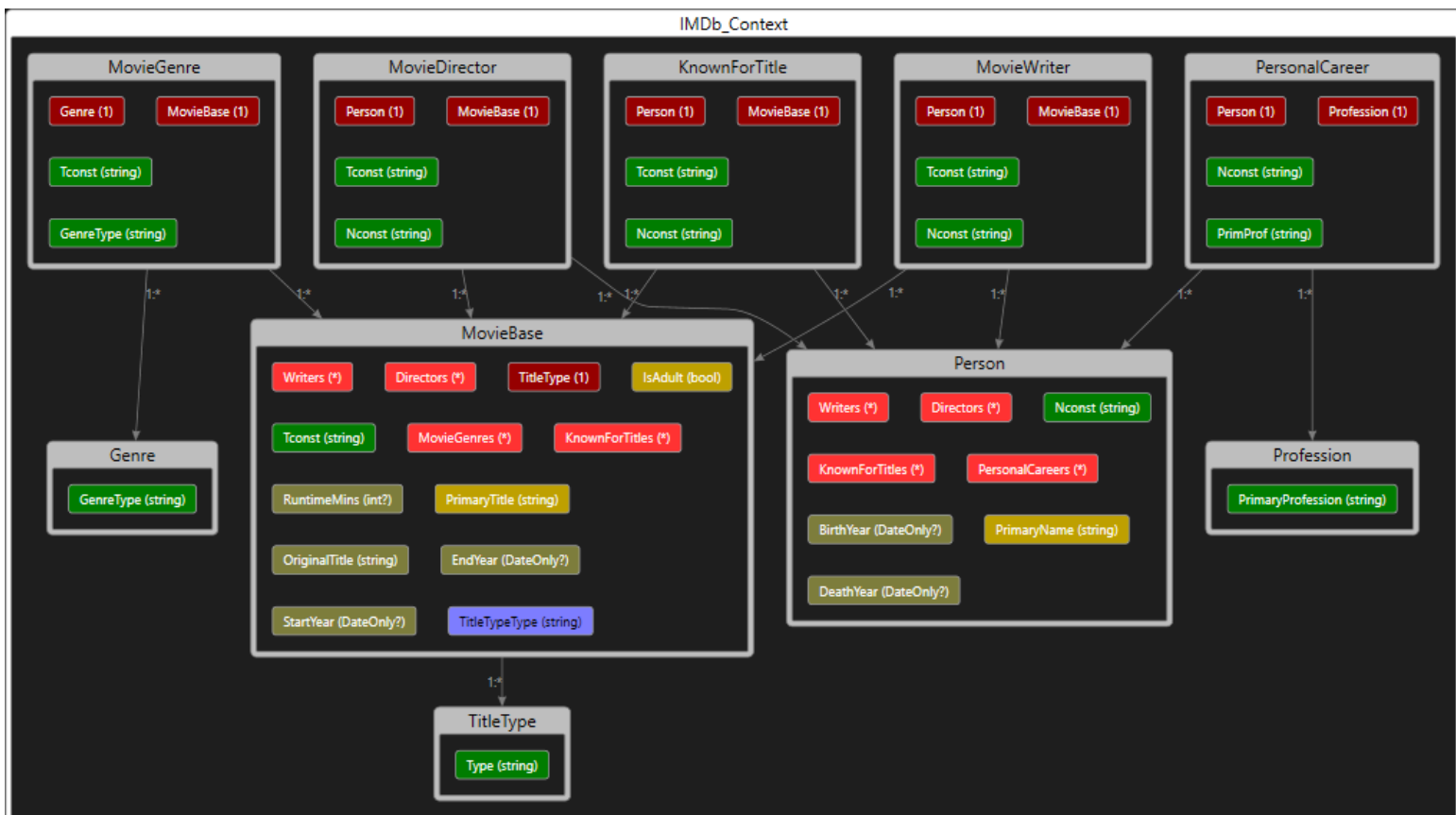
Tilgangen til denne opgave har været at blive klogere på DB opsætning og normalisering af data ved brug af Entity Framework(EF) som udgangspunkt.

Samtidigt har hensigten for mig også været at blive klogere på T-SQL, og brugen deraf. Jeg har ville udforske synergien af brugen af disse to tilgange til at løse de givne User Stories.

Hvad angår UI, har vi ifølge opgavebeskrivelsen haft frit slag. Jeg har valgt at gå med en RESTful API, med tilhørende web-app, da jeg finder det mest relevant mht, til hvad jeg gerne vil beskæftige mig med efter endt uddannelse.

## DB Diagram

Til generering af DB diagram er der brugt [EFCore Power Tools](#) extension.



## DB Design Beskrivelse

Jeg har delt mine DB op i 10 tables via 'Code-first' tilgangen. Tablesne relatere sig fra de 3 .tsv filer på følgende vis:

```
//----- title.basics.tsv -----
0 references | Mikkel Nørfeldt Friborg, 6 days ago | 1 author, 2 changes
public DbSet<MovieBase> MovieBases { get; set; }
0 references | Mikkel Nørfeldt Friborg, 6 days ago | 1 author, 1 change
public DbSet<TitleType> TitleTypes { get; set; }
0 references | Mikkel Nørfeldt Friborg, 6 days ago | 1 author, 1 change
public DbSet<Genre> Genres { get; set; }
0 references | Mikkel Nørfeldt Friborg, 6 days ago | 1 author, 1 change
public DbSet<MovieGenre> MovieGenres { get; set; }

//----- name.basics.tsv -----
0 references | Mikkel Nørfeldt Friborg, 6 days ago | 1 author, 2 changes
public DbSet<Person> Persons { get; set; }
0 references | Mikkel Nørfeldt Friborg, 7 days ago | 1 author, 1 change
public DbSet<Profession> Professions { get; set; }
0 references | Mikkel Nørfeldt Friborg, 7 days ago | 1 author, 1 change
public DbSet<PersonalCareer> PersonalCareers { get; set; }
0 references | Mikkel Nørfeldt Friborg, 7 days ago | 1 author, 1 change
public DbSet<BlockBuster> PersonalBlockbusters { get; set; }

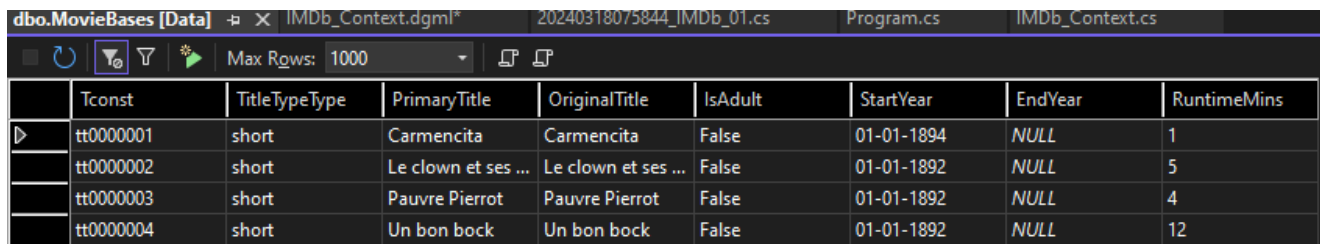
//----- title.crew.tsv -----
0 references | Mikkel Nørfeldt Friborg, 4 days ago | 1 author, 2 changes
public DbSet<MovieWriter> Writers { get; set; }
0 references | Mikkel Nørfeldt Friborg, 4 days ago | 1 author, 2 changes
public DbSet<MovieDirector> Directors { get; set; }
```

## Table overblik og relationer

Denne sektion giver et overblik over hvert table sammensætning i form af deres PKs, FKs.

title.basics.tsv.:

MovieBases:

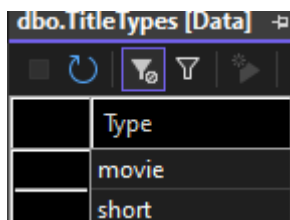


	Tconst	TitleType	PrimaryTitle	OriginalTitle	IsAdult	StartYear	EndYear	RuntimeMins
▶	tt0000001	short	Carmencita	Carmencita	False	01-01-1894	NULL	1
	tt0000002	short	Le clown et ses ...	Le clown et ses ...	False	01-01-1892	NULL	5
	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	False	01-01-1892	NULL	4
	tt0000004	short	Un bon bock	Un bon bock	False	01-01-1892	NULL	12

Dette table har sin PK sat som Tconst.

Tablet har hele 5 conjunction tables.

TitleTypes:

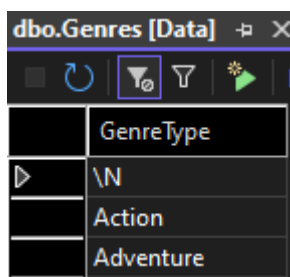


	Type
▶	movie
	short

Dette table består kun af én kolonne som også er PK (Type)

TitleTypes type er normaliseret således der kun er 1 pr. defineret type af TitleType (short, tv-series, movie osv)

Genres:

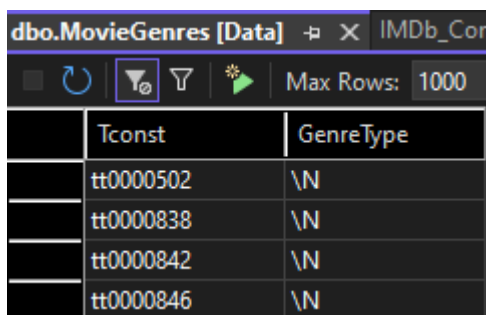


	GenreType
▶	\N
	Action
	Adventure

Dette table består også kun af en kolonne som er PK (GenreType).

Tablet er blevet normaliseret ud fra samme princip som TitleTypes. Således repeteres de samme GenreTypes ikke igen og igen.

MovieGenres:



	Tconst	GenreType
	tt0000502	\N
	tt0000838	\N
	tt0000842	\N
	tt0000846	\N

Tablet her er et conjunction table af MovieBases og Genres.

PK er en composite key bestående af 2 FKs fra de to relaterede tables. Den ene værende Tconst og den anden værende GenreType

name.basics.tsv

#### Persons:

	Nconst	PrimaryName	BirthYear	DeathYear
▶	nm0000001	Fred Astaire	01-01-1899	01-01-1987
	nm0000002	Lauren Bacall	01-01-1924	01-01-2014
	nm0000003	Brigitte Bardot	01-01-1934	NULL

Tablet har Nconst som PK.

Tablet relatere sig til flere andre.

#### Professions:

	PrimaryProfes...
	actor
	actress
	animation_dep...

Dette table har også kun én kolonne med en PK. Hvorfor give tablet et nyt unikt Id med en tilhørende unik profession? Derfor kun en kolonne.

#### PersonalCareers:

	Nconst	PrimProf
	nm0000001	actor
	nm0000001	miscellaneous
	nm0000001	soundtrack
	nm0000002	actress
	nm0000002	soundtrack

Dette table er et conjunction table af Persons og Professions.

Tablet har en PK i form af en composite-key bestående af to FKs fra hvert sit table (nævnt ovenfor, værende: Persons og Professions)

#### KnownForTitles:

	Nconst	Tconst
	nm0000001	tt0027125
	nm0000001	tt0050419
	nm0000001	tt0053137

Tablet her er også et conjunction table. Tablet relatere sig imellem Persons og MovieBases tabellerne.

Ligesom ovenstående table har det en composite-key bestående af sammensætningen af en FK fra hvert af hver table: Persons og MovieBases.

title.crew.tsv

### Writers:

	Tconst	Nconst
	tt0000369	\N
	tt0000370	nm0002042
	tt0000370	nm1304340
	tt0000371	\N

Tablet Writers er et conjunction table imellem MovieBases og Persons. Tablet har to FKs. Et fra hvert af disse tables. Sammen danner de en PK i form af en composite-key.

### Directors:

	Tconst	Nconst
	tt0000504	nm0185426
	tt0000505	nm0381874
	tt0000505	nm1563072
	tt0000506	nm0353576
	tt0000507	\N

Tablet her virker på samme måde som ovenstående. Et conjunction table, der består af to FKs, fra hver sit table, Persons og MovieBases. Sammen danner de en PK i form af en composite-key.

### 'Non-Standard' data Columns

For denne opgave er der ikke brugt nogle 'non-standard data' - men hvad er det overhovedet? Her er et uddrag af nogle som går inden for denne betegnelse:  
[array], GEOMETRY, json og enum..

Jeg var dog tæt på at benytte enums for de data som gik igen da det er en god måde sikrer sig imod typos.. Det viser sig dog ikke at være så let for mig at få implementeret, da jeg løb i forskellige problemstillinger.

### **Normalizations-form, mål og hvorfor**

Mit overordnede mål for normaliseringen har været at undgå repetering (redundancy) af data, som vi har kunne se, er i tsv filerne. Heraf fx: 'titleType' og 'genre' som kunne ses i title.basics.tsv filen.

Dette mål vil også opfylde den 3,5'de normaliserings form, Boyce-Codd), som oftest bruges i mange-til-mange relationer og - igen - benyttes for overskuelighed og undgå data redundancy.

Jeg har ikke tidligere prøvet at arbejde med så stor datamængde og kompleksitet, men vil gerne udfordre mig selv og se om jeg vil kunne løse denne problematik.

## Normalizations afvigelser, hvorfor

*primaryTitle*, *originalTitle* fra *title.basic.tsv*

*primaryName* fra *name.basic.tsv*

Fælles for disse er at de ikke er blevet normaliseret. Begrundelsen har været, at de alle har bestået af længere titler som ikke har givet mening at normalisere mht til film og navne hvad angår personerne.

Der vil forekomme unødigt kompleksitet ved at bryde disse kolonner op i flere, ala: *FirstName*, *MiddleName* og *LastName*. Dataen har differentieret sig for meget til at det vil give mening da nogle personer har haft helt op til 6 navne. Derfor har jeg vurderet den bedste løsning har været at lade disse kolonner være for at undgå null spaces eller unødvendigt mange tables i DB.

## Table Indexes

Følgende tables benytter sig af indexes:

```
migrationBuilder.CreateIndex(  
    name: "IX_Directors_Nconst",  
    table: "Directors",  
    column: "Nconst");  
  
migrationBuilder.CreateIndex(  
    name: "IX_KnownForTitles_Tconst",  
    table: "KnownForTitles",  
    column: "Tconst");  
  
migrationBuilder.CreateIndex(  
    name: "IX_MovieBases_TitleTypeType",  
    table: "MovieBases",  
    column: "TitleTypeType");  
  
migrationBuilder.CreateIndex(  
    name: "IX_MovieGenres_GenreType",  
    table: "MovieGenres",  
    column: "GenreType");  
  
migrationBuilder.CreateIndex(  
    name: "IX_PersonalCareers_PrimProf",  
    table: "PersonalCareers",  
    column: "PrimProf");  
  
migrationBuilder.CreateIndex(  
    name: "IX_Writers_Nconst",  
    table: "Writers",  
    column: "Nconst");  
}
```



## Views (hvis nogen og formål)

Ingen

## Funktioner (hvis nogen og formål)

Ingen

## Stored Procedures(hvis nogen og formål)

Jeg har for min DB opsat følgende STORED PROCEDURES:

Filtreringer:

SearchMovies:

```
IMDb_DB
1 CREATE PROCEDURE SearchMovies @title NVARCHAR(255)
2 AS
3 BEGIN
4     SELECT * FROM MovieBases
5     WHERE PrimaryTitle LIKE '%' + @title + '%'
6     ORDER BY PrimaryTitle ASC;
7 END
```

SearchPersons:

```
1 CREATE PROCEDURE SearchPersons @name NVARCHAR(255)
2 AS
3 BEGIN
4     SELECT * FROM Persons
5     WHERE PrimaryName LIKE '%' + @name + '%'
6     ORDER BY PrimaryName ASC;
7 END
```

GetMovieDetails:

```
1 CREATE PROCEDURE GetMovieDetails
2     @tconst NVARCHAR(450)
3 AS
4 BEGIN
5     SELECT * FROM MovieBases WHERE Tconst = @tconst;
6     SELECT p.* FROM Persons p
7     JOIN Directors d ON p.Nconst = d.Nconst
8     WHERE d.Tconst = @tconst;
9     SELECT p.* FROM Persons p
10    JOIN Writers w ON p.Nconst = w.Nconst
11    WHERE w.Tconst = @tconst;
12 END
```

GetPersonDetails:

```
1 CREATE PROCEDURE GetPersonDetails
2     @nconst NVARCHAR(450)
3 AS
4 BEGIN
5     SELECT * FROM Persons WHERE Nconst = @nconst;
6     SELECT m.* FROM MovieBases m
7     JOIN Directors d ON m.Tconst = d.Tconst
8     WHERE d.Nconst = @nconst;
9     SELECT m.* FROM MovieBases m
10    JOIN Writers w ON m.Tconst = w.Tconst
11    WHERE w.Nconst = @nconst;
12 END
```

CRUD Operationer:

AddMovie:

```
1 CREATE PROCEDURE AddMovie
2     @titleType NVARCHAR(450),
3     @primaryTitle NVARCHAR(MAX),
4     @isAdult BIT,
5     @startYear DATE,
6     @endYear DATE,
7     @runtimeMins INT
8 AS
9 BEGIN
10    DECLARE @maxTconst NVARCHAR(450);
11    SELECT @maxTconst = MAX(Tconst) FROM MovieBases;
12
13    DECLARE @maxId INT;
14    SET @maxId = CAST(SUBSTRING(@maxTconst, 3, LEN(@maxTconst) - 2) AS INT);
15
16    DECLARE @newTconst NVARCHAR(450);
17    SET @newTconst = 'tt' + RIGHT('0000000' + CAST((@maxId + 1) AS NVARCHAR(7)), 7);
18
19    INSERT INTO MovieBases (Tconst, TitleType, PrimaryTitle, IsAdult, StartYear, EndYear, RuntimeMins)
20    VALUES (@newTconst, @titleType, @primaryTitle, @isAdult, @startYear, @endYear, @runtimeMins);
21 END
```

Her er ideen jeg igennem en STORED PROCEDURE sikrer mig oprettelsen af en ny PK som følger samme naming i form af en string. To forbogstaver tt, efterfulgt af 7 numre.

UpdateMovie:

```
1 CREATE PROCEDURE UpdateMovie
2     @tconst NVARCHAR(450),
3     @titleType NVARCHAR(450),
4     @primaryTitle NVARCHAR(MAX),
5     @isAdult BIT,
6     @startYear DATE,
7     @endYear DATE,
8     @runtimeMins INT
9 AS
10 BEGIN
11     UPDATE MovieBases
12     SET TitleType = @titleType,
13         PrimaryTitle = @primaryTitle,
14         IsAdult = @isAdult,
15         StartYear = @startYear,
16         EndYear = @endYear,
17         RuntimeMins = @runtimeMins
18     WHERE Tconst = @tconst;
19 END
```

DeleteMovie:

```
1 CREATE PROCEDURE DeleteMovie
2     @tconst NVARCHAR(450)
3 AS
4 BEGIN
5     DELETE FROM MovieBases WHERE Tconst = @tconst;
6 END
```

AddPerson:

```
1 CREATE PROCEDURE AddPerson
2     @primaryName NVARCHAR(MAX),
3     @birthYear DATE,
4     @deathYear DATE
5 AS
6 BEGIN
7     DECLARE @newId INT;
8     SELECT @newId = ISNULL(MAX(CAST(SUBSTRING(Nconst, 3, LEN(Nconst) - 2) AS INT)), 0) + 1 FROM Persons;
9     DECLARE @newNconst NVARCHAR(450) = 'nm' + RIGHT('0000000' + CAST(@newId AS NVARCHAR(7)), 7);
10
11     INSERT INTO Persons (Nconst, PrimaryName, BirthYear, DeathYear)
12     VALUES (@newNconst, @primaryName, @birthYear, @deathYear);
13 END
```

UpdatePerson:

```
1 CREATE PROCEDURE UpdatePerson
2     @nconst NVARCHAR(450),
3     @primaryName NVARCHAR(MAX),
4     @birthYear DATE,
5     @deathYear DATE
6 AS
7 BEGIN
8     UPDATE Persons
9     SET PrimaryName = @primaryName,
10        BirthYear = @birthYear,
11        DeathYear = @deathYear
12     WHERE Nconst = @nconst;
13 END
```

DeletePerson:

```
1 CREATE PROCEDURE DeletePerson
2     @nconst NVARCHAR(450)
3 AS
4 BEGIN
5     DELETE FROM Persons WHERE Nconst = @nconst;
6 END
```

Målet for mine STORED PROCEDURES har været at benytte dem, via Serviceklasserne, men mange af dem - heraf Add metoderne som har skulle tilføje et nyt automatiseret Id og properties til en klasse som består af andre klasser - været for besværligt, decideret at benytte da jeg ikke har fået det til at gå hånd i hånd med brugen af EF.

Dog har jeg testet mange af metoderne succesfuldt, direkte, med SQL Queries, hvilket har været en god øvelse. Delete metoderne bruges i det endelige produkt af servicesne.

## Custom Roles (hvis nogen og formål)

Jeg nåede desværre ikke så langt til at få dette implementeret i Visual Studio (ikke SSM)

## SQL-Statements og/eller funktioner til importering af .tsv

For importering/insertion fra .tsv til DB, benyttede jeg mig som nævnt tidligere en EF 'Code-First' approach. Jeg oprettede først min library fil, IMDbLib, hvori de classes som skulle agere som de kommende tables blev sat op sammen med en, DbContext opsætning med tilhørende og relationsopsætning.

```
//-----Konfiguration for forholdet mellem Person og MovieBase-----
modelBuilder.Entity<KnownForTitle>()
    .HasKey(bb => new { bb.Nconst, bb.Tconst });

modelBuilder.Entity<KnownForTitle>()
    .HasOne(bb => bb.Person)           // en BlockBuster har en Person
    .WithMany(p => p.KnownForTitles)   // en Person har mange KnownForTitles
    .HasForeignKey(bb => bb.Nconst);    // en BlockBuster har en fremmednøgle Nconst

modelBuilder.Entity<KnownForTitle>()
    .HasOne(bb => bb.MovieBase)        // en BlockBuster har en MovieBase
    .WithMany(mb => mb.KnownForTitles) // en MovieBase har mange KnownForTitles
    .HasForeignKey(bb => bb.Tconst);    // en BlockBuster har en fremmednøgle Tconst

//-----Konfiguration for forholdet mellem Person og Profession-----
modelBuilder.Entity<PersonalCareer>()
    .HasKey(pc => new { pc.Nconst, pc.PrimProf });

modelBuilder.Entity<PersonalCareer>()
    .HasOne(pc => pc.Person)           // en PersonalCareer har en Person
    .WithMany(p => p.PersonalCareers)   // en Person har mange PersonalCareers
    .HasForeignKey(pc => pc.Nconst);    // en PersonalCareer har en fremmednøgle Nconst

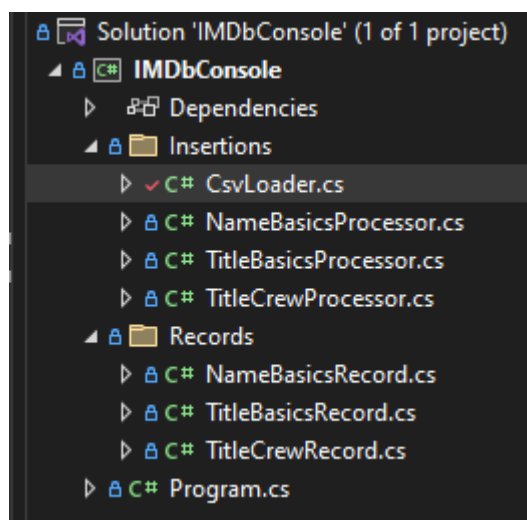
modelBuilder.Entity<PersonalCareer>()
    .HasOne(pc => pc.Profession)        // en PersonalCareer har en Profession
    .WithMany()                        // en PersonalCareer har mange Profession
    .HasForeignKey(pc => pc.PrimProf);  // en PersonalCareer har en fremmednøgle PrimProf

//-----Konfiguration for forholdet mellem MovieBase og Genre-----
modelBuilder.Entity<MovieGenre>()
```

I en anden fil, IMDbConsole, benyttede jeg mig af mit library via en dll.

Udover de basale EF NuGet packages (EFCore, EFCore.SqlServer og EFCore.Tools) benyttede jeg mig som noget nyt, NuGet packages'ne [csvHelper](#) og [EFCore.BulkExtension](#) for lettere læselig kode, og for lettere at kunne separere min opsætningen. Jeg valgte at benytte mig af de første 100.000 linjer kode fra hver .tsv fil.

Arkitektur kan ses på billedet nedenfor som giver indblik for min tilgang



### CsvLoader:

Denne klasse, CsvLoader, indeholder en metode LoadCsv<T>. Denne metode læser en CSV-fil fra en given sti (filePath), og returnerer de første numberOfLines linjer som en liste af objekter af typen T. CSV-filen antages at være tabulatorsepareret, og eventuelle fejl i dataene ignoreres.

### <tsvFileName>Processor

Disse klasser har en metode, der tager en liste over dens tilsvarende Records objekter og transformerer dem til forskellige samlinger/lister. Denne transformation involverer opdeling af strenge, oprettelse af nye objekter, udfyldning af disse objekter med data fra record-objekterne, og håndtering af specielle tilfælde, hvor dataene kan være mangelfulde eller i et uventet format (som f.eks. ved håndtering af datoer)

```
public class NameBasicsProcessor
{
    public static (List<Person>, HashSet<Profession>, List<PersonalCareer>, List<BlockBuster>)
    ProcessNameBasicsRecords(List<NameBasicsRecord> nameRecords)
    {
        var persons = new List<Person>();
        var professions = new Dictionary<string, Profession>();
        var personalCareers = new List<PersonalCareer>();
        var personalBlockbusters = new List<BlockBuster>();

        foreach (var record in nameRecords)
        {
            var person = new Person
            {
                Nconst = record.nconst,
                PrimaryName = record.primaryName,
                BirthYear = TryParseDate(record.birthYear),
                DeathYear = TryParseDate(record.deathYear)
            };
            persons.Add(person);

            if (!string.IsNullOrEmpty(record.primaryProfession))
            {
                var professionTypes = record.primaryProfession.Split(',');
                foreach (var professionType in professionTypes)
                {
                    if (!professions.ContainsKey(professionType))
                    {
                        var profession = new Profession { PrimaryProfession = professionType };
                        professions.Add(professionType, profession);
                    }
                }
            }
        }
    }
}
```

```

    }

    var personalCareer = new PersonalCareer { Nconst = record.nconst, PrimProf =
    professionType };
    personalCareers.Add(personalCareer);
}
}

var tconsts = record.knownForTitles.Split(',');
foreach (var tconst in tconsts)
{
    var blockBuster = new BlockBuster { Nconst = record.nconst, Tconst = tconst };
    personalBlockbusters.Add(blockBuster);
}
}

//----- DateTime Converter
static DateOnly? TryParseDate(string dateValue)
{
    if (dateValue.Equals("\\N", StringComparison.OrdinalIgnoreCase))
    {
        return null;
    }

    // Hvis datoen er i formatet "yyyy-mm-dd"
    if (DateTime.TryParseExact(dateValue, "yyyy", null, System.Globalization.DateTimeStyles.None, out var
    dateTime))
    {
        return new DateOnly(dateTime.Year, 1, 1);
    }

    Console.WriteLine($"Fejl ved konvertering af dato: {dateValue}");
    return null;
}

return (persons, new HashSet<Profession>(professions.Values), personalCareers, personalBlockbusters);
}
}

```

Uddrag fra klassen NameBasicProecesser.  
 Idéen/udgangspunktet er den samme for de andre Processors

## <tsvFileName>Records

```
public class NameBasicsRecord
{
    3 references | Mikkel Nørfeldt Friberg, 16 days ago | 1 author, 1 change
    public string nconst { get; set; }
    1 reference | Mikkel Nørfeldt Friberg, 16 days ago | 1 author, 1 change
    public string primaryName { get; set; }
    1 reference | Mikkel Nørfeldt Friberg, 16 days ago | 1 author, 1 change
    public string birthYear { get; set; }
    1 reference | Mikkel Nørfeldt Friberg, 16 days ago | 1 author, 1 change
    public string deathYear { get; set; }
    2 references | Mikkel Nørfeldt Friberg, 16 days ago | 1 author, 1 change
    public string primaryProfession { get; set; }
    1 reference | Mikkel Nørfeldt Friberg, 16 days ago | 1 author, 1 change
    public string knownForTitles { get; set; }
}
```

Disse klasser fungerer som Data Transfer Objects (DTO'er) og repræsenterer datastrukturen for de tilsvarende .tsv filer. Hver instans af en Record-klasse repræsenterer en enkelt række i den tilsvarende TSV-fil.

Hver property i en Record-klasse svarer til en kolonne i .tsv filen.

Disse klasser bruges til at indlæse data fra .tsv filerne og gør det nemmere at arbejde med data sættet.

## Program

Dette er hovedklassen, der styrer dataindlæsning og -indsættelse. Den læser data fra TSV-filer, transformerer dataene til objekter ved hjælp af Processor-klasserne, og indsætter derefter objekterne i databasen i bulk. Hvis der opstår fejl, fanges de og udskrives til konsollen.

```
public class Program
{
    public static void Main(string[] args)
    {
        string nameBasicsTsv = @"C:\Users\mikkf\OneDrive\Dokumenter\Visual Studio
2022\TSVs\name.basics.tsv\data.tsv";
        string titleBasicsTsv = @"C:\Users\mikkf\OneDrive\Dokumenter\Visual Studio
2022\TSVs\title.basics.tsv\data.tsv";
        string titleCrewTsv = @"C:\Users\mikkf\OneDrive\Dokumenter\Visual Studio 2022\TSVs\title.crew.tsv\data.tsv";

        Console.WriteLine("Starting program...");
        Stopwatch stopwatch = Stopwatch.StartNew();

        try
        {
            using (var context = new IMDb_Context())
            {
                Console.WriteLine("Created context...");
                var loader = new CsvLoader();
                Console.WriteLine("Created CSV loader...");

                // Load the data into instances of the Record class
                var nameRecords = loader.LoadCsv<NameBasicsRecord>(nameBasicsTsv, 100000);
                Console.WriteLine($"Loaded {nameRecords.Count} records from {nameBasicsTsv} TSV file...");
                var titleRecords = loader.LoadCsv<TitleBasicsRecord>(titleBasicsTsv, 100000);
                Console.WriteLine($"Loaded {titleRecords.Count} records from {titleBasicsTsv} TSV file...");
                var titleCrewRecords = loader.LoadCsv<TitleCrewRecord>(titleCrewTsv, 100000);
                Console.WriteLine($"Loaded {titleCrewRecords.Count} records from {titleCrewTsv} TSV file...");

                var (persons, professions, personalCareers, personalBlockbusters) =
                    NameBasicsProcessor.ProcessNameBasicsRecords(nameRecords);
                var (movieBases, titleTypes, genres, movieGenres) =
```



```

TitleBasicsProcessor.ProcessTitleBasicsRecords(titleRecords);
var (directors, writers) = TitleCrewProcessor.ProcessTitleCrewRecords(titleCrewRecords);

// Use BulkInsert to insert the records for each table in bulk
context.BulkInsert(persons);
context.BulkInsert(professions);
context.BulkInsert(personalCareers);
context.BulkInsert(personalBlockbusters);
//movieBase.tsv
context.BulkInsert(movieBases);
context.BulkInsert(titleTypes);
context.BulkInsert(genres);
context.BulkInsert(movieGenres);
//titleCrew.tsv
context.BulkInsert(directors);
context.BulkInsert(writers);
    }
}
catch (Exception ex)
{
    Console.WriteLine($"An error occurred: {ex.Message}");
    Console.WriteLine($"Stack trace: {ex.StackTrace}");
}

stopwatch.Stop();
Console.WriteLine($"Total time elapsed: {stopwatch.Elapsed}");
Console.WriteLine("End of program...");
}
}

```

Uddrag fra class Program

## DDL/"Schema" i SQL-statements

```
1 CREATE TABLE [Genres] (
2     [GenreType] nvarchar(450) NOT NULL,
3     CONSTRAINT [PK_Genres] PRIMARY KEY ([GenreType])
4 );
5 GO
6
7
8 CREATE TABLE [Persons] (
9     [Nconst] nvarchar(450) NOT NULL,
10    [PrimaryName] nvarchar(max) NOT NULL,
11    [BirthYear] date NULL,
12    [DeathYear] date NULL,
13    CONSTRAINT [PK_Persons] PRIMARY KEY ([Nconst])
14 );
15 GO
16
17
18 CREATE TABLE [Professions] (
19     [PrimaryProfession] nvarchar(450) NOT NULL,
20     CONSTRAINT [PK_Professions] PRIMARY KEY ([PrimaryProfession])
21 );
22 GO
23
24
25 CREATE TABLE [TitleTypes] (
26     [Type] nvarchar(450) NOT NULL,
27     CONSTRAINT [PK_TitleTypes] PRIMARY KEY ([Type])
28 );
29 GO
30
```

```
31
32 CREATE TABLE [PersonalCareers] (
33     [Nconst] nvarchar(450) NOT NULL,
34     [PrimProf] nvarchar(450) NOT NULL,
35     CONSTRAINT [PK_PersonalCareers] PRIMARY KEY ([Nconst], [PrimProf]),
36     CONSTRAINT [FK_PersonalCareers_Persons_Nconst] FOREIGN KEY ([Nconst]) REFERENCES [Persons] ([Nconst]) ON DELETE CASCADE,
37     CONSTRAINT [FK_PersonalCareers_Professions_PrimProf] FOREIGN KEY ([PrimProf]) REFERENCES [Professions] ([PrimaryProfession]) ON DELETE CASCADE
38 );
39 GO
40
41
42 CREATE TABLE [MovieBases] (
43     [Tconst] nvarchar(450) NOT NULL,
44     [TitleTypeType] nvarchar(450) NOT NULL,
45     [PrimaryTitle] nvarchar(max) NOT NULL,
46     [OriginalTitle] nvarchar(max) NULL,
47     [IsAdult] bit NOT NULL,
48     [StartYear] date NULL,
49     [EndYear] date NULL,
50     [RuntimeHins] int NULL,
51     CONSTRAINT [PK_MovieBases] PRIMARY KEY ([Tconst]),
52     CONSTRAINT [FK_MovieBases_TitleTypes_TitleTypeType] FOREIGN KEY ([TitleTypeType]) REFERENCES [TitleTypes] ([Type]) ON DELETE CASCADE
53 );
54 GO
55
56
57 CREATE TABLE [Directors] (
58     [Tconst] nvarchar(450) NOT NULL,
59     [Nconst] nvarchar(450) NOT NULL,
60     CONSTRAINT [PK_Directors] PRIMARY KEY ([Tconst], [Nconst]),
61     CONSTRAINT [FK_Directors_MovieBases_Tconst] FOREIGN KEY ([Tconst]) REFERENCES [MovieBases] ([Tconst]) ON DELETE CASCADE,
62     CONSTRAINT [FK_Directors_Persons_Nconst] FOREIGN KEY ([Nconst]) REFERENCES [Persons] ([Nconst]) ON DELETE CASCADE
63 );
64 GO
65
```

```

66
67 CREATE TABLE [MovieGenres] (
68     [Tconst] nvarchar(450) NOT NULL,
69     [GenreType] nvarchar(450) NOT NULL,
70     CONSTRAINT [PK_MovieGenres] PRIMARY KEY ([Tconst], [GenreType]),
71     CONSTRAINT [FK_MovieGenres_Genres_GenreType] FOREIGN KEY ([GenreType]) REFERENCES [Genres] ([GenreType]) ON DELETE CASCADE,
72     CONSTRAINT [FK_MovieGenres_MovieBases_Tconst] FOREIGN KEY ([Tconst]) REFERENCES [MovieBases] ([Tconst]) ON DELETE CASCADE
73 );
74 GO
75
76
77 CREATE TABLE [PersonalBlockbusters] (
78     [Nconst] nvarchar(450) NOT NULL,
79     [Tconst] nvarchar(450) NOT NULL,
80     CONSTRAINT [PK_PersonalBlockbusters] PRIMARY KEY ([Nconst], [Tconst]),
81     CONSTRAINT [FK_PersonalBlockbusters_MovieBases_Tconst] FOREIGN KEY ([Tconst]) REFERENCES [MovieBases] ([Tconst]) ON DELETE CASCADE,
82     CONSTRAINT [FK_PersonalBlockbusters_Persons_Nconst] FOREIGN KEY ([Nconst]) REFERENCES [Persons] ([Nconst]) ON DELETE CASCADE
83 );
84 GO
85
86
87 CREATE TABLE [Writers] (
88     [Tconst] nvarchar(450) NOT NULL,
89     [Nconst] nvarchar(450) NOT NULL,
90     CONSTRAINT [PK_Writers] PRIMARY KEY ([Tconst], [Nconst]),
91     CONSTRAINT [FK_Writers_MovieBases_Tconst] FOREIGN KEY ([Tconst]) REFERENCES [MovieBases] ([Tconst]) ON DELETE CASCADE,
92     CONSTRAINT [FK_Writers_Persons_Nconst] FOREIGN KEY ([Nconst]) REFERENCES [Persons] ([Nconst]) ON DELETE CASCADE
93 );
94 GO
95

```

```

96
97 CREATE INDEX [IX_Directors_Nconst] ON [Directors] ([Nconst]);
98 GO
99
100
101 CREATE INDEX [IX_MovieBases_TitleTypeType] ON [MovieBases] ([TitleTypeType]);
102 GO
103
104
105 CREATE INDEX [IX_MovieGenres_GenreType] ON [MovieGenres] ([GenreType]);
106 GO
107
108
109 CREATE INDEX [IX_PersonalBlockbusters_Tconst] ON [PersonalBlockbusters] ([Tconst]);
110 GO
111
112
113 CREATE INDEX [IX_PersonalCareers_PrimProf] ON [PersonalCareers] ([PrimProf]);
114 GO
115
116
117 CREATE INDEX [IX_Writers_Nconst] ON [Writers] ([Nconst]);
118 GO

```

## SQL-Injection beskyttelse? Hvordan?

Mit overordnede mål for at beskytte mod SQL-injections har været ved brug af vores STORED PROCEDURES. EFs benyttelse og LINQ sikrer også mod angreb. Det er langt fra alle metoder som får benyttet STORED PROCEDURES i sidste ende, men min Delete-metoder består dog udelukkende deraf..

## Refleksion

Mega frustrerende og mega spændende projekt som jeg kan se komme til gavn i erhvervs livet - for evt kunder som skal have flyttet data og eller optimeret deres software.

Jeg kunne godt have tænkt mig det lykkedes med enums mht til de her conjunction tables som skildre en type (Genre, Profession og TitleType). Det tænker jeg ville have været en god måde at sikrer 'types'. Jeg vil også gerne have nået til at oprette forskellige brugere af systemet - det nåede jeg ikke - men jeg tror måske (7-9-13), at det ikke er så besværligt igen?

Ligeledes nåede jeg heller ikke i mål med en UI til min RESTful API. API'en er dog blevet OK testet, men kunne godt bruge flere exceptions.. Men det dog heller ikke det som jeg valgte at fokusere på.. Fokus var at blive klogere på DB opsætning, T-SQL, og normalisering med brug af EF. Hvilket jeg er ok tilfreds med.. Jeg nåede faktisk at få løst de to ekstra UserStories om at finde alt info til en MovieBase og en Person.

Fremtidigt vil jeg øve få sat en tilhørende UI op, og leget lidt med de andre T-SQL funktioner. Views, hovedsagligt da jeg tænker det er mest relevant inden for evt fremtidigt arbejde