This READ\_ME file will tell you how to get the .tsv files and how to import them into SQL Server as tables. This also includes some basic info on the solution 😊

First of all where to get IMDB’s latest database dump:

<https://datasets.imdbws.com/>

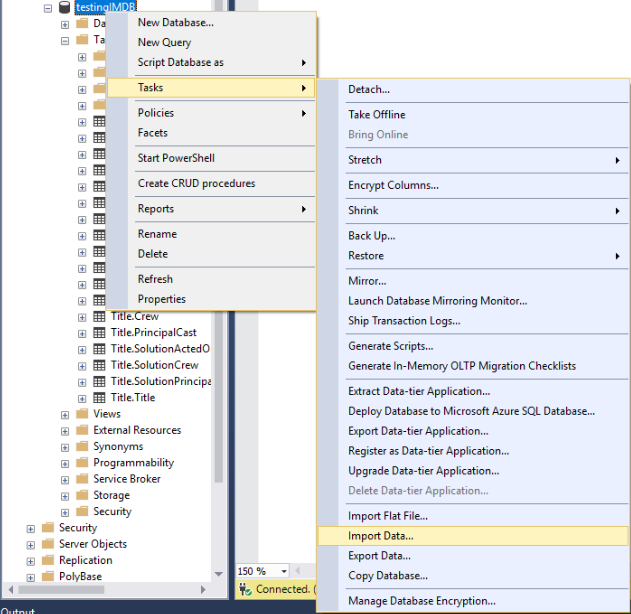
Here is their very simple documentation as to what each table contains and what their attributes mean:

<http://www.imdb.com/interfaces/>

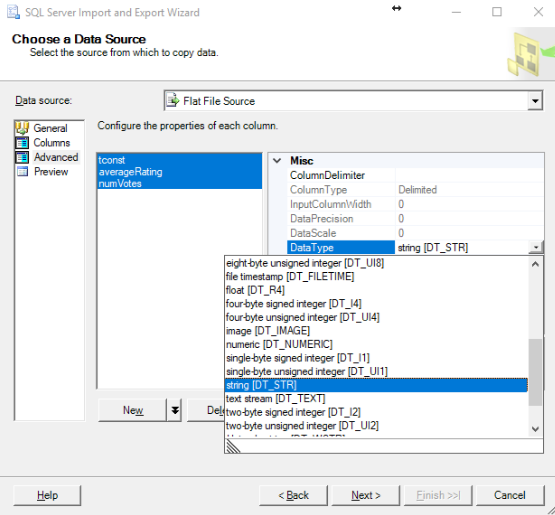
Unfortunately (although justified) the dump is not their full database. So, you will come across some weird issues when importing.

First of all HOW to import the tsv files onto SQL Server as actual tables. Keep in mind this Database is roughly around 5Gb after we’re done with our solution, this is not accounting to other testing tables you might do. So, keep at least 10Gb free on your PC for this project.

First, create a testing database, it can be a tempDB it can be an actual database, it’s up to you.



Although you are importing a flat file, in this case tsv (tab separated value), we will go with the ‘Import Data…’ option. Reason being is that Import Flat File option does not allow you to change how SQL Server will deal with the strings that you are going to import. And if you try to do it without changing that, it will always give you an error and won’t work.



After you choose Flat File in the Data Source, and browse to the file you want to import. Go into Advanced and mark all columns. The Columns by default are on string[DT\_STR], we don’t want that. Change it to the option [DT\_TEXT] (not Unicode) or [DT\_WSTR] (Preferably WSTR for Unicode/UTF8). The reason is that STR is varchar, and is in ANSI (also has a different size limit). This is a database in UTF8 format, with titles in different languages. So, you don’t want weird characters like these popping up.

C:\Users\Chessbrain\Desktop\IMDB_SOLUTION\LEL.png

After that you choose your testing database and import it there as a table (done by default).

Now the interesting stuff…

This solution is probably extremely inefficient, but it was the way I solved it, so if you have any other solutions by all means, share them 😊

The biggest problem with the database is arrays, if you want to normalize the database you can’t have string arrays. The **SplitterGitHub script** solves this issue, and I have included comments explaining some commands.

Depending on how you will solve this, you might be able to convert certain strings to their respective values. But in my solution dates and pretty much all other attributes except for Ratings are done in nvarchar. So most of your tables will have nvarchar as attribute types.

The IMDB\_SCHEMA\_TABLES script creates all tables and their respective schemas, this will be your “production” database, where your final product will be. The Title.ActedOn table was somewhat confusing for me, didn’t know if I should put it in the Cast schema or Title schema. You can change this if you want.

***Piece of advice*** Some scripts will run for a very long time (specifically talking about the script to split the title\_crew table, which took almost 2h on my rig to finish, my PC isn’t really all that strong, 8GB of RAM and an older generation i7 processor with an SSD. So, you might have better luck)

I have also included two scripts for Title.Type’s and Cast.Profession’s data, so you don’t have to import them.

***Now for the biggest problem while importing this database to your real production database.***

As mentioned before, this is not the full database… And I am not talking about the tables, but the data itself. For example: there are more than one million titles included in the name\_basics table in the “knownForTitle” attribute that don’t exist in the actual title\_basics table. So, when you try import data it will never work, it will give you an error that you’re violating the FOREIGN KEY CONSTRAINT.

The way I solved this is probably not the best way possible, but it works. I made in the testDB copies of the production tables but without any of the constraints, joined them together by putting the FOREIGN KEY constraint on the IDENTITY field… But after importing data, I inner joined them on their tconst and nconst fields. This resulted in consistent data that only exists in all tables in the given database dump.

I have included this dummy copy of the production database in the FAKE\_COPY script. It’s practically the same as the main production databse but without the constraints and with dummy tables in between to join them together.

After you import into the dummy tables the data from the tsv tables all you have to do now is write simple SELECT queries with INNER JOIN on the tconst and nconst fields instead of the generated ID… And all that’s left to do is to import that data into the production database and you’re done :D

This project took me 4 days in total of constant work (literally ate breakfast using one hand, wrote SQL queries using the other hand). Hope you enjoy it, and please improve anything you think needs improvement 😊

**Thank you for joining this project ~ Chessbrain**