Web Design Project Specifications

Completion requirements

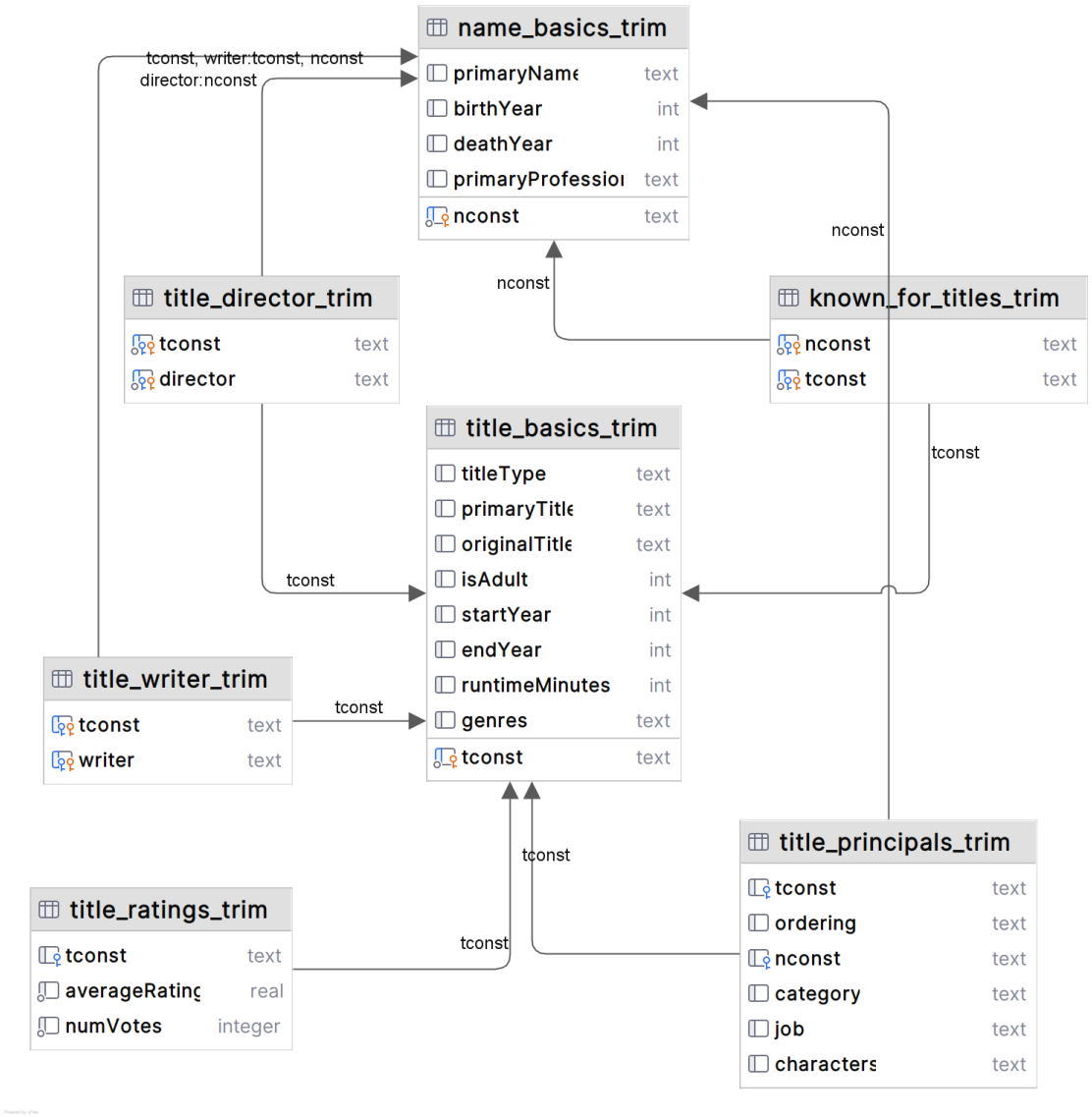
IMDB — The Movies Fan Page — IMDB-Pedia

Introduction

You are required to develop a website allowing to present and edit the movies and TV series metadata dataset found in the IMDB non-commercial datasets (https://developer.imdb.com/non-commercial-datasets/). While the project/solution functionality expectations are pretty basic — at the very minimum the solution must implement core features for retrieving and editing data in the database — plus, there also a Search Engine Optimisation (SEO) component to this project. The core functionality along with the SEO steps in your solution will ensure a reasonable grade for this project. And then, to get top marks you may implement some optional/advanced features, such as user registration and authentication, or JavaScript charts to visualise the data.

IMDB (SQLite) Database

The full non-commercial dataset is too huge for as to work with — there are 190,921,017 rows in the full dataset! It is also not entirely suitable for a pain-free import into an SQL database. Instead of using the full dataset the teaching team has prepared for our project a tidy little trim/extract of data which is still interesting and meaningful. We have also tidied this trimmed data for you just a bit — you will need to do a bit more of that, as per the specifications, along with the core tasks. Here is the schema diagram (generated automatically by the PHPStorm Database Tool, with the layout adjusted manually for clarity):



The two core tables are `title_basics` and `name_basics` (with the `_trim` suffix). Please have a read of the metadata descriptions provided by the IMDB Non-Commercial Datasets page for a better picture, but keep in mind that the structure of the data has been adjusted to turn it from a bunch of `.tsv` (tab-separated value files) into a proper relational (SQLite) database with primary and foreign keys and basic referential data integrity.

There are seven tables in total with `title_basics` table in the centre of them all. You may want/need to add more tables or views if you feel this is necessary to implement specific features required for your project. Here's the description of the main tables in the `imdb.2.sqlite3` database:

- `name_basics` — names of all people involved with one title or another,
- `title_director` and `title_writer` — these are the two associative tables linking titles and directors in the former, and titles and writers in the latter tables (these two tables were created from the original `title_crew` table, where the `directors` and `writers` columns contained comma-separated values, which could not be used as proper foreign keys to point to the appropriate people),

- `title_principals` — these are people (their IDs, really) involved with one title or another — here they are combined in this table along with their job type and possible roles/characters in the relevant movies/series,
- `known_for_titles` — this is another associative table created from the `knownForTitles` column in the original `title_basics` table, where they were all bunched up as comma-separated values, which would have made it impossible to implement any kind of references,
- `title_ratings` — the official IMDB ratings for each item in the `title_basics` table along with the number of votes.

If you noticed that there's some redundancy and overlap in these tables, you are right — this redundancy and overlap come from the original downloaded IMDB data.

To get a better feel of the data do go ahead and explore the databases in the PHPStorm Database/Query Console/Tool. It is essential that you examine the structure of the tables — this will give you a clear idea what data is expected when designing SQL insert queries and the pages for say, recording the `title_basics` data. Create a data source from the `imdb.2.sqilte3` file in the PHPStorm Database tool, following the explanations in Tutorial 4, and view the tables and columns. Please note that while we are using SQLite for this project, which is a server-less database, a more realistic scenario is a server-based database, e.g. MySQL, PostgeSQL or any others from the same shelf, but this should not stop you from improving your understanding of SQL which you'll have to get your hands a bit dirty with in this project.

When creating associative tables and breaking up columns containing comma-separated data, this code example demonstrates one approach to break a column of comma-separated values into a separate table:

```
-- First we create a playpen table with comma-separated values in one column.
create table tt (nconst, knownForTitles);
insert into tt values ('nm0000001', 'tt0072308,tt0050419,tt0027125,tt0031983'),
                      ('nm0000002', 'tt0037382,tt0075213,tt0038355,tt0117057');

-- Then we can run a query of that sort to break up the data into a separate table.
with recursive temp(nconst, tconst, remainder) as (
    select nconst,
           substr(knownForTitles, 1, instr(knownForTitles || ',', ',') - 1) as
tconst,
           substr(knownForTitles, instr(knownForTitles || ',', ',') + 1) as
remainder
    from tt
    union all
    select nconst,
           substr(remainder, 1, instr(remainder || ',', ',') - 1),
           substr(remainder, instr(remainder || ',', ',') + 1)
    from temp
    where remainder <> ''
)
select nconst, tconst, remainder
from temp
order by nconst;
```

Solution Requirements

The core task is to develop a website for showcasing and augmenting the movie and TV series titles and the associated people in their various capacities as listed in the IMDB

database. This task will require a bit of front-end work for designing HTML pages with forms and back-end functionality written in PHP code with some embedded SQL queries; and then, to give the website a proper look-and-feel you'll need to add some interactivity with JavaScript and CSS. There is a lot of freedom as to how you may approach this task — it is intentionally designed in an open-ended manner to encourage you to try your creative powers. At the very minimum your solution must include these basic features/properties:

- The start/home/index page for your solution, providing the visitors/users as well as search engine robots/crawlers with the basic information about your site. This page should include the navigation menu for accessing the rest of the pages in your solution. Don't forget about the appropriately implemented search functionality!
- The overview pages to display all titles (movies, short and series) and pagination — we don't want to burden our users and their web browsers with 211,339 titles in one page — and links to view all details for any of the items (rating, votes, etc.). A user should be able to view/click through to the directors, writers and principals, that associated with any given title — you may seek inspiration in the IMDB.com pages or come up with better solutions, which are well overdue!
- Improve the database structure by creating these tables:
    - `genres` — list of all (unique) genres, as extracted from `title_basics` — this would be a very short table,
    - `title_genre` — an associative table establishing the connections between `title_basics` and `genres` associated with any given title,
    - `professions` — list of all (unique) professional capacities, as extracted from `name_basics` table,
    - `name_professions` — an associative table establishing the connections between `name_basics` and `professions` associated with any given name,
- Clean up the tables/data in the database by replacing any occurrence of `'\N'` strings with `NULL` values where appropriate.
- The appearance and interactivity (the look and feel) of the website must use appropriate layout, graphics, and interaction flow, e.g.: make the images clickable to zoom-in/display a hi-res version to see the details. You may use suitable JavaScript/CSS libraries/frameworks to style and enhance your solution. For example, these are easy front-end frame works for designing: Bootstrap, Bulma, Materialize, Metro UI, Tailwind, etc.
- The SEO features/steps required for improving the ranking of your page on Google, whatever it is you deem possible to do for the on-site SEO steps.

For the advanced features you may choose to implement some or all of the following:

- Implementing login pages, features for creating user profiles/user registration, with appropriate features, or a feature for creating trivia sections for specific titles or people. This should use appropriate security measures and avoid storing plain-text passwords in the database. There are two types of users:
    - The administrator users may only be able add and edit the content on the website and overwrite absolutely anything.
    - The visitors or regular users should be allowed to rate titles and leave comments. Or, if desired titles and people should be added to favourites lists to be displayed via user profiles. And then, a user should be able to view the favourites of other users.

- Make a Single-Page application using ReactJS library. The structure of the project will have to be changed quite a bit from the start, where PHP will be used a backend and ReactJS as frontend, in an ideal all-stars-solution scenario consuming a PHP REST API. This means that the PHP will be used to access the database and will provide the API that ReactJS will use to visualise/change the data. There's a bit of ReactJS material in the textbook, one chapter, but it is quite light on details — you will likely have to look into ReactJS documentation online to see the examples (only for the interface ReactJS usage): https://react.dev/learn, https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started.

Your solution must be accompanied by a four-to-five-page documentation/report presenting/describing the details of your solution. This documentation is essential to the graders — please include the information to help the grader gain a clear understanding of your project solution. *The reality of this is such that a weaker solution with a clear guide for the grader is likely to earn a better grade than an advanced solution with no grader guidance. So, pretty please, no matter what, don't franchise this job to ChatGPT or any of its friends, because you'll do a disservice to yourself in the first place.*

Marking Schedule

The core tasks have 85% allocated to those, which is all the way up to the lower boundary of an A+, so to get into the A+ territory you'd want to think of the advanced features.

- Documentation/report — 25%. The documentation should be sufficient for testing and understanding your solution, presented in proper format and language. You can use any UC-approved GenAI tools to help you with the quality of the report and your code for that matter. *By all means, put GenAI to good use, which is learning new stuff, but don't outsource your brain to it*. Don't forget to state in your report what GenAI was useful for in your instance — a few paragraphs will be enough.
- Basic features — 35%. The index/home page with search features (1) along with the pages for showing the film/series and people overview (2), click through to each part (3), appropriate navigation, structure, presentation (4), which all should be supported by the improved database structure (5).
- SEO features — 25%. The work demonstrating your understanding of the SEO techniques applicable to the scope and context of this project.
- Advanced features — 15% (each). You may choose to implement just one of those advanced features (properly) or all or any subset of them — best solutions will be noted and acknowledged. And before you ask, no, it's impossible to get more than 100% for this project, although quite often the teaching team would like to make this happen.

Questions and Clarifications via Discussion Forum

Needless to say, you are invited to post questions via our Discussion Forum, and please be sure to discuss in the form anything related to this project, short of sharing your solutions.

Last modified: Wednesday, 16 April 2025, 11:48 AM