Brendan Gressel

Stephen Johnson

Tyler Yelverton

Stage 4: Databases Project Report

Introduction:

For this project, we created a database that sorts and compares book to movie adaptations and uses online resources to initially establish scores that the user will be able to view simultaneously. Our database contains books with their title, rating, and author, while our films have their title, rating, and year of production. All book ratings and places of publication were originally taken from Goodreads, while the films ratings and production companies were taken from IMDB. We chose this topic because we love books and movies and have yet to come across a website that does this comparison. We thought it would be interesting to see the scores compared from general audiences and finally answer the question, was the book better than the movie? This initially works as a one-stop website for people looking for easy comparisons. This website will also allow users to share their opinions of the adaptations. The key components for our website are laid out in three columns. The middle column holds the title which the user searched up, to the left is the book rating and publisher, and the right is the movie rating with its production company. Above these columns is a search bar where the user can make queries, and below is a submission form for the user to submit their own ratings.

Database Details:

We designed our database using three different entities: TV Show/Movie, Book, and Production Company. The TV Show/Movie entity has a Title key, as well as a Rating, Year, and Production Company Name attribute. It has an “adapted from” relationship with the Book entity and a “produced by” relationship with the Production Company entity. The Book entity has a Title key, as well as a Rating, Author, and Publisher Name attribute. It has an “adapted to” relationship with the TV Show/Movie entity and a “published by” relationship with the Production Company entity. The Production Company entity has a Name key as well as a Location attribute. It has a “published” relationship with the Book entity and a “produced” relationship with the TV Show/Movie entity. For the relationships, the TV Show/Movie “adapted from” Book relationship is many to one because we assumed all adaptations must come from one Book, however a Book can be adapted many times. The TV Show/Movie “produced by” Production Company relationship is many to one because we assumed each TV Show/Movie only has one Production Company, but a Production Company can produce many TV Shows/Movies. The Book “published by” Production Company relationship is the same, as we assumed a Book only has one publisher, but a Production Company can publish many Books.

ER Diagram:



\*We assumed there is only one publisher per book.

\*We assumed there is only one studio per TV Show/Movie.

Relational Schema:

TV Show/Movie = (T\_Title, Year, T\_Rating, B\_Title, P\_Name)

Book = (B\_Title, B\_Rating, Author, P\_Name)

Production Company = (P\_Name, Location)

We created three tables for our database. The first we named tvOrMovie, and this table holds all the values for the tv shows or movies that were adapted from books. This table has four values: rating and year, which are both not null integers, and the production company name and title, which are both not null varchars. This table has one primary key, title. The second table we made is called book, and this table holds all the book entries in our database. This table also has four values: rating, which is a not null integer, and title, author, and the publisher name, which are all not null varchars. This table has one primary key, title. Our third and final table is named pCompany, and this holds all the production companies for the tv shows and movies and publishers for the books. This table has two values: name and location, which are both not null varchars. This table has one primary key, name. Our functional dependencies rely on our relationships between our entities. Our relationships are all many to one, meaning you can always find the publisher if you know the book, movie, or tv show, however you may not always be able to find the book, tv show, or movie if you only know the production company or publisher. All of our attributes are always key attributes for our relationships, however they are not always super keys for the relationships, so our tables are in 3NF. We have quite a few constraints when working with our database, the first being the keys are mandatory for any contribution. In order INSERT, UPDATE, or DELETE into our tables, the title cannot be left blank or it will not submit. We also define all of our values in our SQL code as NOT NULL so they always have to be defined when contributing.

Functionality Details:

Our basic features involve searching, inserting, updating, and removing current records in the database. To insert records, we will require someone to have a movie and a book in order to validate the entry. They will need to provide the production companies, authors, years, and ratings, which they can get from IMDB and Goodreads or be personal, when entering their adaptations. We accomplished this by implementing SQL in our PHP and querying a new entry using the INSERT TO function and the table provided. In order to search the database, the user will need to know the title of the book and movie pair because those are the keys for our entities and are required for registration. We implemented this by using the SQL function SELECT and compared what the user typed in versus all the titles in our database. Some examples of our entries that users could search are: The Shining (Film, 8.4 on IMDB, Warner Bros.) vs The Shining (Book, 8.4 on Goodreads, New English Library). To update records, we will allow users to add their own ratings to the books or movies when searched, as well as update authors and production companies unless they are ever wrong. This was accomplished by using the SQL function UPDATE in our PHP code. Since titles do not change, they cannot update those. To delete an entry, the user will just need to enter the title, and both the book and movie will be deleted. This was implemented by using the SQL function DELETE FROM followed by both tables. Our advanced function is comparing entries based on whether they were good adaptations or not. We will allow the user to sort the entries by the difference between the movie rating (from those who have read the book) and book rating, thus showing if the adaptation was a good or bad one and if it exceeded in quality over the book. We accomplished this by writing algorithms that used SQL functions MIN, MAX, and INNER JOIN that compared each entry and spit out the one that matched the minimum and maximum.

Implementation Details:

Link to our code: <https://github.com/bgressel/DatabasesProject>

We used XAMPP as the platform for our local host database. For the website design, we used HTML and CSS to design the containers. We then used PHP to connect the backend SQLite3 database to our frontend HTML containers. All database logic was implemented using SQL.

Experiences:

We learned a lot from this project, especially how to organize databases. We learned how to code in SQL and PHP, as well as integrating the two together so that they work in unison. The structuring of the tables and how they are organized was a large part of our learning, as well as displaying these tables on the website. Adding in our basic features and our advanced features was another hurdle, as we had to use HTML to get the data from the user and then use PHP to parse it through the database. A challenge that occurred was when we were trying to implement our advance feature that separates the best and worst adaptation. We eventually accomplished this by finding the INNER JOIN SQL command and used MIN and MAX when finding the differences between the movie rating and the book rating. Once we compared all the data we collected, we were able to find the best and worst. We will use what we learned in this project to make more complicated systems in the future. Now that we understand how they are implemented and organized, we feel very comfortable making systems with more than three entities and even more relationships and attributes. We feel comfortable with the languages used, and granted they are used in whatever professional workplace we end up at, the experience will be a nice reminder of how to use them and we will easily be able to prompt ourselves on how to work with them.