

Media Database

Daniel Segal (dls148@case.edu)

Sam Fleckenstein (sef44@case.edu)

EECS 341- Introduction to Databases

1/27/13

Project Proposal

Given the massive amounts of media that exist in the current day and age, it can be overwhelming try to sort through everything that is available and make any kind of judgement about what seems the most interesting. Choosing which of your movies to watch next, what book to read, which CD to put on or what game to play can be a difficult decision. Keeping track of what you have loaned and to whom can also present a challenge.

In an effort to make it easier to keep all of these forms of entertainment in order and easily accessible, our project will be a categorized, sortable, and searchable library of the media you own. Utilizing the innate attributes of these different items, such as creators, genre, and even qualitative measurements such as ratings, we will attempt to break a large set of data into more accessible chunks, and show meaningful connections between them.

Our webapp will utilize SQL querying in order to create, update, and maintain a database over the different media types, pulling in information from outside sources when possible (either through API calls or manual entry). Using MySQL as a DBMS, we will be able to narrow the data down to meet precise conditions set by the user, making it easy and accessible to manage your media, both virtually and physically.

At this point in time most people have experienced the problem of having too much information at their fingertips. With a near infinite number of movies, books, or games that can be found and owned it is nearly impossible to keep track of them all. Whether you are trying to decide what to watch, or trying to remember who has your newest CD, our webapp will provide a solution.

Project Progress Report 1

1. Introduction: Overview

We live in an age of consumable media and content proliferation. No longer constrained to traditional content creation and distribution networks, artists, directors, and all manner of creative people are capable of producing something and making it accessible to a vast number of people. While this is undoubtedly great for people who seek out such works, it can make keeping track of all of your entertainment options a daunting task.

With our project, we plan to make a unified multimedia catalog, built from information supplied by the user. Rather than having to sift through a DVD binder or dig boxes of books out of the attic, our virtual construct of the user's media possessions will make sorting much easier through the use of MySQL as a DBMS. We are also looking to solve the eternal problem of keeping track of just who has your books and cds. These types of media get traded and passed between friends so often that it can be near impossible to track down. By making our database not just about the types of media, but the relations between people and their media, we will be able to offer a solution.

Our database will manage a number of types of media: books, movies (on multiple formats), music, and games (on various systems). The database will be built as one person's media catalog, but other people entities will be available as temporary borrowers of these items. On top of the MySQL database, we will be using HTML and PHP to dynamically interact with the data supplied by both the user and API calls to outside sources of information whenever possible.

2. Application Requirements Specifications

The core technologies behind our website will be HTML as the skeletal web structure used to create the various pages, CSS to provide styling to the HTML, preventing it from being just bare text and form fields, and PHP to act as an intermediary between the frontend of the website and the backend

MySQL database. PHP will provide a convenient way to make SQL calls without having to maintain a Java or .NET framework, and allow us the most flexibility when developing the site.

Our application will be built around the CRUD (Create, Read, Update, Delete) principles of interacting with data, and various pages of the website will be designed in accordance. All pages will fall into one of those four categories (note: wireframes intended only to depict functionality, exact fields and layouts are subject to change).

a) The Create Page

User's Media Collection

The wireframe shows a form titled "Enter a New Book" with a horizontal line underneath. On the left side, there are three text input fields labeled "Title", "Author", and "Number of Pages". Below these is a "Rating" section with five radio buttons labeled 1, 2, 3, 4, and 5; the radio button for "2" is selected. To the right of the text fields is a "Cover Picture" label above a square placeholder containing a grey silhouette of a person's head and shoulders. Below the entire form is a square button with a right-pointing arrow.

Figure A.1 - An example create page for a book

The create page will be how a user adds a new item to their collection. As seen in figure A.1, there will be fields for such information as "Title", "Author", etc. This will allow the user to add both qualitative information about the book as well as their own subjective opinion in the form of a rating which will later serve as a means of refining a search. Each type of media will have different fields, allowing for appropriate fields (such as a director for a movie) to only appear for their corresponding media type. If possible, these create pages will be tied into Amazon's Product Advertising API, which

allows information retrieval from Amazon's sizeable catalog, which would remove some of the necessity for the user to manually fill out non subjective information.

b) The Read Pages

User's Media Collection

Movies Books Games Music			
Title	Author	Pages	Rating
Item A	Author A	Pages A	★ ★ ☆ ☆ ☆
Item B	Author B	Pages B	★ ★ ★ ★ ★
Item C	Author C	Pages C	★ ★ ★ ☆ ☆
Item D	Author D	Pages D	★ ★ ★ ☆ ☆

Figure B.1 - An overall view of a media type

User's Media Collection

Book Title


Book Author

Book Pages

Rating

☐ 1
 ☒ 2
 ☐ 3
 ☐ 4
 ☐ 5

Cover Picture







Figure B.2 - A view page for a specific item

User's Media Collection

Search for an Item in your Catalog

Media Type  (Can Search over All Media Types)

Title

Books

Author

Pages


Rating

Movies

Director

Actor

Genre



- A search page

The read pages will make up the bulk of our application. While entering the data will be a crucial part of building the database initially, most interaction by the user will take place from one of three read pages shown above. Figure B.1 shows what will be the homepage of our application, a sortable table which will give the user an at-a-glance overview of what they have entered into the database. This table will be potentially be inherently sortable using JQuery, which will allow the user to view their media by alphabetic order, rating order, or others depending on the type of media and its attributes. There will be multiple tabs so that all forms of media can be presented in their most logical form, without needing to have columns for fields that don't apply.


The next figure, B.2 which is what you will get upon clicking on an item from B.1 will show all entered data. The only notable difference between B.2 and A.1 is that this information will be in a read only format (at least until the update view is entered, discussed in section C).

The last page shown, the search page, is where most of the intensive querying will be performed, and the main reason for this application existing. On this page, the user will be presented with a multitude of text fields and dropdowns to enter criteria which will be then converted to a SQL query in the PHP and sent to the MySQL database. Users will be able to limit the scope of their query to various forms of media or search the entirety of the database. It will also be possible to search for items by who is currently in possession of them.

c) The Update Page

User's Media Collection

Enter a New Book

<input type="text"/>	<div>Cover Picture</div> 
<input type="text"/>	
<input type="text"/>	
Rating <input type="radio"/> 1 <input checked="" type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5	




Figure C.1 - An update page

Update pages will not be appreciably different from create pages except that the information in the text boxes will already be supplied by the DBMS. This page will allow the user to update information about the item, such as fixing a typo or changing a rating.

d) The Delete Page

The delete page will not be a page as such, but a function accessible from the update page which will allow a user to remove an item from the database. It will simply take the form of a button that (upon confirmation) will issue the appropriate SQL query to drop the item from the database.

3. Database Requirements Specifications

As mentioned above, we will be using MySQL as a DBMS for this application, and all interaction with the data will be in the form of SQL queries processed and submitted by PHP, which will then be able to send that information back to the frontend of our application.

A) Data to be Maintained in the Database

a) Books

Books will store information such as title, author, year published, description, genre, rating, and comments.

Books can belong to a series, and authors can have written several books. A book can belong to several genres.

b) Movies

Movies will store information such as title, year released, director, lead actors/actresses, description, comments, and rating.

Movies can belong to a series, directors can have directed many movies, and actors/actresses can be in many movies. A movie can belong to several genres.

c) Music

Music will store information such as title, year released, principal musician/group, track listing, genre (distinct from book, movie, and game genres), rating, and comments.

Music can belong to multiple genres, and musicians can have put out multiple albums, and belong to multiple groups.

d) Games

Games will store information such as title, year released, genre, rating, and comments.

Games can belong to a series and multiple genres.

e) Information Regarding Owners

Information will be stored as to who owns any particular item (assumed to be owner of the database/ admin), as well as who is in current possession of that item. People will be their own objects in the database, with a 1:m relationship from them to items in the database.

B) Queries and Transactions

Most active queries will occur as shown above from the search page (Figure B.3). However, the database will be queried whenever the main page (Figure B.1) is built. For this reason, should the database exceed a certain preselected number of items, the data will be paginated such that the entire table for a form of media does not need to be sent every time the page is viewed.

Aside from the page building queries, the other querying functionality will be user driven, as the user attempts to find items in their catalog according to their

attributes, or according to who is currently in possession of them. These queries will be able to contain a large number of criteria (aggregated from attributes, i.e. Author = "X", Genre="Y", Rating > "3") to allow the user to find their items by a very tight bound.

C) Events and Actions

The main events and actions of the database are as described above in the context of CRUD. Users will be able to Create new entries, Update them to have more recent information, and Delete them should they wish to remove an item from their catalog for whatever reason.

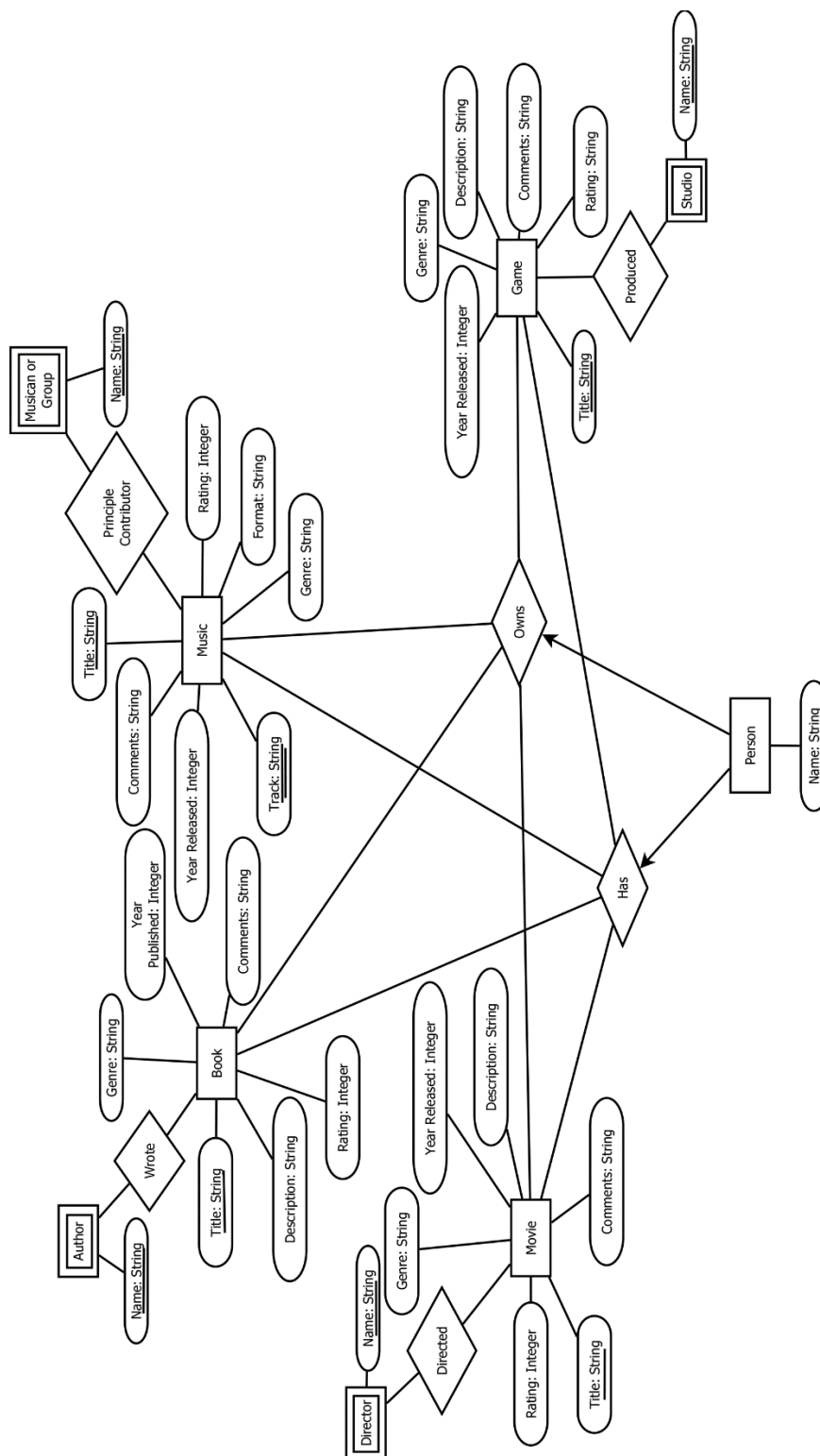
D) Integrity Constraints

In order to prevent the database from overgrowing and retaining old or inaccurate information, all groupings of items will require that at least one item exist. For example, an actor can be in several movies so upon creation of a movie entry in the database, an actor entry would also be created with the actor being in a 1:m relationship with his body of work. Many more movies can be added to the database featuring that one actor, but should the user remove all movies featuring that actor from his or her catalog, the actor entry would also be removed. This weak entity relationship will occur for all has-many relationships in the database as described in order to prevent database bloat.

In addition, all entries in the database will have to be unique so that they can be appropriately fetched by queries, also fitting the metaphor of this application as a catalog. It would make no sense to have multiple copies of the same item in your library.

Project Progress Report 2:

1. ER Model



2. Relational Model

Person (Name, Owned_Movie, Owned_Book, Owned_Music, Owned_Game, Loaned_Movie, Loaned_Book, Loaned_Music, Loaned_Game)

Primary key: Name

Foreign keys: Owned_Movie, Owned_Book, Owned_Music, Owned_Game, Loaned_Movie, Loaned_Book, Loaned_Music, Loaned_Game

Integrity constraints: People who were loaned books are deleted upon loan end

Movie (Title, Rating, Director, Genre, Year_Released, Description, Comments)

Primary keys: Title, Director

Foreign keys: None

Integrity constraints: None

Book (Title, Author, Year_Published, Comments, Rating, Genre, Description)

Primary keys: Title, Author

Foreign keys: None

Integrity constraints: None

Music (Title, Principle_Contributor, Rating, Format, Genre, Tracks, Year_Released, Comments)

Primary keys: Title, Principle_Contributor

Foreign keys: None

Integrity constraints: None

Game (Title, Year_Released, Genre, Description, Comments, Rating, Production_Studio)

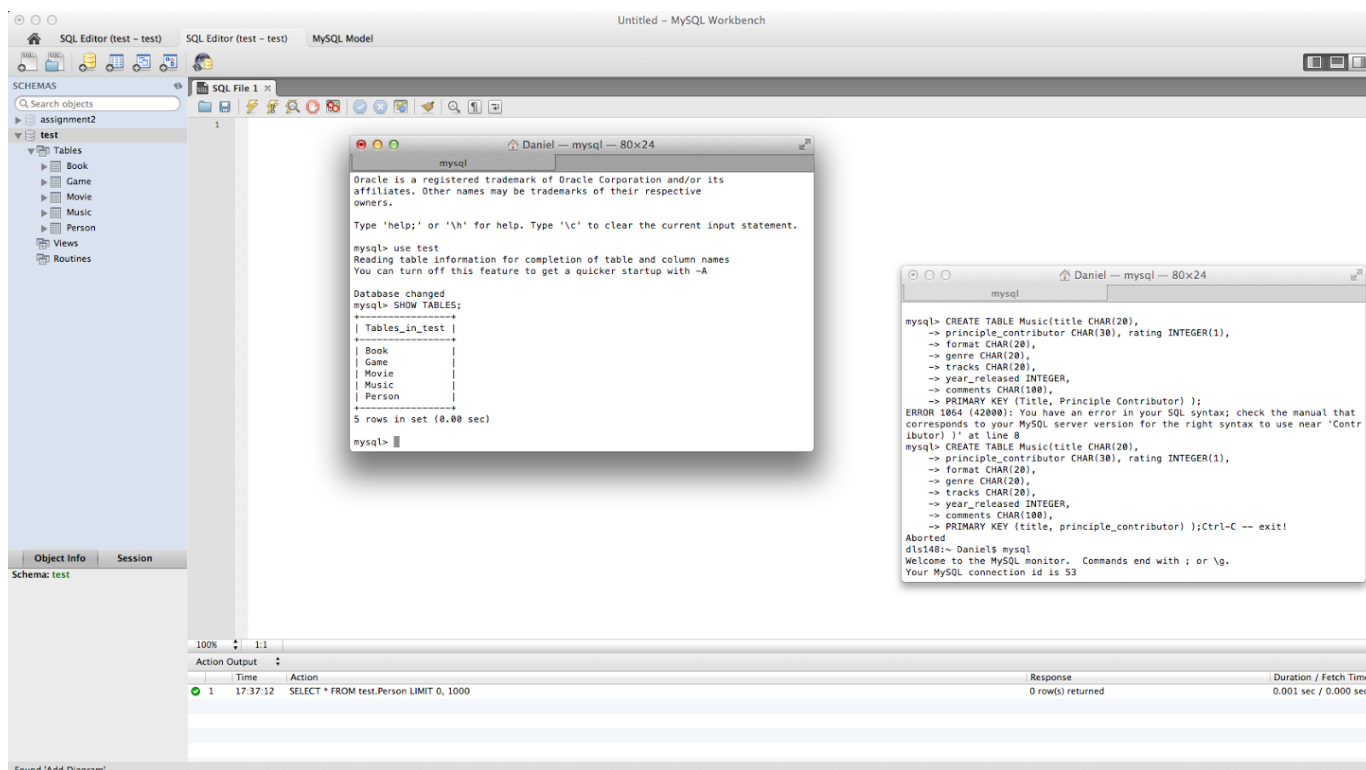
Primary keys: Title, Production_Studio

Foreign keys: None

Integrity constraints: None

2. Database Creation

For our project, we will be using MySQL as our DBMS. Daniel already had it set up on his computer from previous projects, and so had no difficulties with implementing the database schema detailed below. Sam does not yet have MySQL, but is in the process of downloading it. Using the command line to actually create the database and tables along with MySQL Workbench as a gui frontend to the database, all tables and attributes were created as seen below. The database will be populated using data from Daniel and Sam's personal media collections.



The following commands were used to create the database:

```
CREATE TABLE Person (
    Name CHAR(20) NOT NULL,
    Owned_Movie CHAR(20), Owned_Director CHAR(20) ,
    Owned_Book CHAR(20), Owned_Author CHAR(20),
    Owned_Music CHAR(20), Owned_Contributor CHAR(20),
    Owned_Game CHAR(20), Owned_Studio CHAR(20),
    Loaned_Movie CHAR(20), Loaned_Director CHAR(20),
    Loaned_Book CHAR(20), Loaned_Author CHAR(20),
    Loaned_Music CHAR(20), Loaned_Contributor CHAR(20),
    Loaned_Game CHAR(20), Loaned_Studio CHAR(20),
    PRIMARY KEY (Name),
    FOREIGN KEY (Owned_Movie, Owned_Director )
        REFERENCES (Movie.Title, Movie.Director),
    FOREIGN KEY (Owned_Book, Owned_Author )
        REFERENCES (Book.Title, Book.Author),
    FOREIGN KEY (Owned_Music, Owned_Contributor )
        REFERENCES (Music.Title, Music.Principle Contributor),
    FOREIGN KEY (Owned_Game, Owned_Studio )
        REFERENCES (Game.Title, Game.Production Studio)
    FOREIGN KEY (Loaned_Movie, Loaned_Director )
        REFERENCES (Movie.Title, Movie.Director),
    FOREIGN KEY (Loaned_Book, Loaned_Author )
```

```

REFERENCES (Book.Title, Book.Author),

FOREIGN KEY (Loaned_Music, Loaned_Contributor )

REFERENCES (Music.Title, Music.Principle_Contributor),

FOREIGN KEY (Loaned_Game, Loaned_Studio )

REFERENCES (Game.Title, Game.Production_Studio) );

CREATE TABLE Movie(

    Title CHAR(20) NOT NULL,

    Director CHAR(30) NOT NULL,

    Rating INTEGER,

    Genre CHAR(20),

    Year_Released INTEGER,

    Description CHAR(200),

    Comments CHAR(100),

    PRIMARY KEY (Title, Director) );

CREATE TABLE Book(

    Title CHAR(20) NOT NULL,

    Author CHAR(30) NOT NULL,

    Rating INTEGER,

    Genre CHAR(20),

    Year_Published INTEGER,

    Description CHAR(200),

    Comments CHAR(100),

    PRIMARY KEY (Title, Author) );

```

```
CREATE TABLE Music(  
    Title CHAR(20) NOT NULL,  
    Principle_Contributor CHAR(30) NOT NULL,  
    Rating INTEGER,  
    Format CHAR(20),  
    Genre CHAR(20),  
    Tracks CHAR(20),  
    Year_Released INTEGER,  
    Comments CHAR(100),  
    PRIMARY KEY (Title, Principle_Contributor) );
```

```
CREATE TABLE Game(  
    Title CHAR(20) NOT NULL,  
    Production_Studio CHAR(30) NOT NULL,  
    Rating INTEGER,  
    Genre CHAR(20),  
    Year_Released INTEGER,  
    Description CHAR(200),  
    Comments CHAR(100),  
    PRIMARY KEY (Title, Production_Studio) );
```


7. SQL Queries and an Exercise in RA and TRC

All of these queries are only taking into account the content creators (authors, artists, studios, etc) who have at least one piece of media in the database. For example, if there is a director with no movies, they will not be considered in any of these queries.

Another note: the following queries are examples. The same idea has been applied to each of the different types of media. For example, the first query is “Find the authors with three or more 3-star or better books”, but we have also applied this query to directors and movies for our application.

Find the authors with three or more 3-star or better books.

RA and TRC cannot be applied to this query because it requires keeping a count.

SQL

```
SELECT A.name, A.id
FROM Authors A, Books B, Authors_Books C
WHERE A.id = C.author_id AND C.book_id = B.id AND
      B.rating >= 3
GROUP BY A.id
HAVING COUNT(B.id) >= 3
```

Find the user with the most games.

RA and TRC cannot be applied to this query because it requires keeping a count.

SQL

```
SELECT P.name
FROM People P, Games G, (SELECT COUNT(G1.id) AS num_games, G1.person_id
                        FROM Games G1
                        GROUP BY G1.person_id) AS T1
WHERE NOT EXISTS (
    SELECT COUNT(G2.id) AS num_games, G2.person_id
    FROM Games G2
    GROUP BY G2.person_id
    HAVING COUNT(G2.id) > T1.num_games)
```

Find the artists who only produce classic rock.

SQL

```
SELECT P.name
FROM People P, Games G, (SELECT COUNT(G1.id) AS num_games, G1.person_id
                          FROM Games G1
                          GROUP BY G1.person_id) AS T1
WHERE NOT EXISTS (
    SELECT COUNT(G2.id) AS num_games, G2.person_id
    FROM Games G2
    GROUP BY G2.person_id
    HAVING COUNT(G2.id) > T1.num_games)
```

TRC

$$\prod_{name} Artists - \prod_{name} (Artists \bowtie Artists_Musics \bowtie (\sigma_{genre \neq Classic\ Rock}(Music)))$$

RA

```
{t(1) | (∃a) (Artists (a) ^ (t.name = a.name) ^ (~∃m) (Musics(m) ^
(∃c) (Artists_Musics(c) ^ (c.artist_id = a.id) ^ (c.music_id = m.id) ^
(m.genre <> 'Classic Rock'))))}
```

Find the directors whose movies are all rated 3-stars or better.

SQL

```
SELECT D.name
FROM Directors D
WHERE NOT EXISTS (
    SELECT *
    FROM Movies M, Directors_Movies A
    WHERE M.rating < 3 AND A.director_id = D.id
    AND A.movie_id = M.id)
```

TRC

$$\prod_{name} Directors - \prod_{name} (Directors \bowtie Directors_Movies \bowtie (\sigma_{rating < 3}(Movies)))$$

RA

```
{t(1) | (∃d) (Directors(d) ^ (t.name = d.name) ^ (~∃m) (Movies(m) ^
(∃a) (Directors_Movies(a) ^ (m.rating < 3) ^ (a.director_id = d.id) ^
(a.movie_id = m.id))))}
```

Find the users who own every movie directed by Michael Bay.

SQL

```
SELECT P.name
FROM People P
WHERE NOT EXISTS (
    SELECT M.title
    FROM Movies M, Directors D, Directors_Movies A
    WHERE M.id = A.movie_id AND D.id = A.director_id AND D.name = 'Michael
Bay'
    AND P.id NOT IN(
        SELECT P1.id
        FROM People P1
        WHERE P1.id = M.person_id))
```

RA

$$\prod_{name} (People \bowtie (\prod_{person_id, movie_id} (Owns))) \\ / (\prod_{name} (\sigma_{name='MichaelBay'}(Directors \bowtie Directors_{Movies} \bowtie Movies)))$$

TRC

```
{a(1) | (∃p) (People(p) ^ (a.name = p.name) ^ (~∃m) (Movies(m) ^
(∃d) (Directors(d) ^ (∃t) (Directors_Movies(t) ^ (m.director_id =
t.director_id) ^ (t.director_id = d.id) ^ (d.name = 'Michael Bay') ^
(~∃p1) (People(p1) ^ (m.person_id = p1.id)))))) }
```

The other queries were very basic. For example, on the games page, we used

```
SELECT * FROM Games
```

to display all of the games. The same sorts of queries were used for Books, Movies, and People.

8. Integrity Constraints

We needed to be careful of a few things as we were implementing this database. First, we needed to ensure that only a single person could own each item. This was because of the “perspective”

that the database was designed from. The database is designed so that a single instance of it will only be used by a single person. This means that everything in the database belongs to a single person. Second, every piece of media also must have a creator and title. The triggers that we implemented were as follows. If a user is deleted, all of the media belonging to them is also deleted. If an author is deleted, all of their books will be deleted as well. The same ideas apply for artists/music, directors/movies, and studios/games. We also assigned every media item a unique id (within that type of media).

9. Relational Database Design -- Applying the Dependency Theory

Our database is already in BCNF. The dependencies are as follows.

For movies:

{director_name, title -> rating, genre, year_released, description, comments}

This is not redundant because director_name and title are the only attributes whose closures contain any of these related attributes. Since there is only one functional dependency for this relation and the LHS of it is a superkey, it is in BCNF.

For books:

{author_name, title -> rating, genre, year_published, description, comments}

This is not redundant because author_name and title are the only attributes whose closures contain any of these related attributes. Since there is only one functional dependency for this relation and the LHS of it is a superkey, it is in BCNF.

For music:

{artist_name, title -> rating, genre, year_released, description, comments, format, track_listing}

This is not redundant because artist_name and title are the only attributes whose closures contain any of these related attributes. Since there is only one functional dependency for this relation and the LHS of it is a superkey, it is in BCNF.

For games:

{studio_name, title -> rating, genre, year_released, description, comments}

This is not redundant because author_name and title are the only attributes whose closures contain any of these related attributes. Since there is only one functional dependency for this relation and the LHS of it is a superkey, it is in BCNF.

For people:

The person's name functionally determines everything that they own, and have, so this is also in BCNF.

10. Revisiting the Relational Database Schema

The biggest change that we made to our database schema was removing the Has relation. We chose to do this because given that our database is intended to be used by a single person, we realized that it was unnecessary. A user will be able to tell who has what pieces of their media through the use of the Owns attribute for each entity. We chose not to revise any other parts of our schema for several reasons. First, given the size of the database, any modern processor will be able to execute any of the queries very quickly. This means that we did not need to combine any of the tables for faster execution. The next concern is storage size of the database. Even though it is small enough to evaluate queries on quickly, we still need to be concerned with the total disk space it takes up. This is the other reason that we chose not to combine any of the tables. This would lead to redundant data, which would take up more space than necessary. Our schema is as follows:

```
CREATE TABLE `artists` (`id` integer primary key NOT NULL,
                          `name` varchar(255) NULL,
                          `created_at` timestamp NOT NULL,
                          `updated_at` timestamp NOT NULL,
                          FOREIGN KEY (music_id)
                          REFERENCES ARTISTS_MUSICS(Music.id));
```

```
CREATE TABLE `artists_musics` (`music_id` integer NULL,
                                `artist_id` integer NULL);
```

```
CREATE TABLE `authors` (`id` integer primary key NOT NULL,
                          `name` varchar(255) NULL,
                          `created_at` timestamp NOT NULL,
                          `updated_at` timestamp NOT NULL,
                          FOREIGN KEY (book_id)
                          REFERENCES Authors_Books(Book.id));
```

```
CREATE TABLE `authors_books` (`book_id` integer NULL,
                                `author_id` integer NULL);
```

```
CREATE TABLE `books` (`id` integer primary key NOT NULL,
                        `title` varchar(255) NULL,
                        `year` integer NULL,
                        `comments` text NULL,
                        `rating` integer NULL,
                        `genre` varchar(255) NULL,
                        `description` text NULL,
                        `created_at` timestamp NOT NULL,
                        `updated_at` timestamp NOT NULL,
                        `person_id` integer NULL,
                        `author_id` integer NULL,
                        FOREIGN KEY (author_id)
                        REFERENCES Authors_Books(Author.id));
```

```
CREATE TABLE `directors` (`id` integer primary key NOT NULL,
                            `name` varchar(255) NULL,
                            `created_at` timestamp NOT NULL,
                            `updated_at` timestamp NOT NULL,
                            FOREIGN KEY (movie_id)
                            REFERENCES DIRECTORS_MOVIES(Movie.id));
```

```
CREATE TABLE `directors_movies` (`movie_id` integer NULL,
                                   `director_id` integer NULL);
```

```
CREATE TABLE `games` (`id` integer primary key NOT NULL,
                        `title` varchar(255) NULL,
                        `year` integer NULL,
                        `genre` varchar(255) NULL,
                        `description` text NULL,
                        `comments` text NULL,
                        `rating` integer NULL,
                        `created_at` timestamp NOT NULL,
                        `updated_at` timestamp NOT NULL,
                        `person_id` integer NULL,
                        `studio_id` integer NULL,
                        FOREIGN KEY (studio_id)
                        REFERENCES GAMES_STUDIOS(Studio.id));
```

```
CREATE TABLE `games_studios` (`studio_id` integer NULL,
                                `game_id` integer NULL);
```

```
CREATE TABLE `movies` (`id` integer primary key NOT NULL,
                         `title` varchar(255) NULL,
                         `rating` integer NULL,
                         `genre` varchar(255) NULL,
                         `year` integer NULL,
                         `description` text NULL,
                         `comments` text NULL,
                         `created_at` timestamp NOT NULL,
                         `updated_at` timestamp NOT NULL,
                         `person_id` integer NULL,
                         `director_id` integer NULL,
                         FOREIGN KEY (director_id)
                         REFERENCES DIRECTORS_MOVIES(Director.id));
```

```
CREATE TABLE `movies_directors` (`movie_id` integer NULL,
                                   `director_id` integer NULL);
```

```
CREATE TABLE `musics` (`id` integer primary key NOT NULL,
                         `title` varchar(255) NULL,
                         `rating` integer NULL,
```

```

`format` varchar(255) NULL,
`genre` varchar(255) NULL,
`tracks` text NULL,
`year` integer NULL,
`comments` text NULL,
`created_at` timestamp NOT NULL,
`updated_at` timestamp NOT NULL,
`person_id` integer NULL,
`artist_id` integer NULL,
FOREIGN KEY (artist_id)
REFERENCES ARTISTS_MUSICS(Artist.id));

```

```

CREATE TABLE `people` (`id` integer primary key NOT NULL,
`name` varchar(255) NULL,
`created_at` timestamp NOT NULL, `
updated_at` timestamp NOT NULL);

```

```

CREATE TABLE `studios` (`id` integer primary key NOT NULL,
`name` varchar(255) NULL,
`created_at` timestamp NOT NULL,
`updated_at` timestamp NOT NULL,
FOREIGN KEY (music_id)
REFERENCES ARTISTS_MUSICS(Artist.id));

```

```

CREATE TABLE `studios_games` (`game_id` integer NULL, `
studio_id` integer NULL);

```

11. DBMS Implementation

Based upon the overarching metaphor of creating a media library for one user to track all of their physical items, some changes were made as we began to build the actual database. The first, and largest, was a decision to change from building an application using MySQL and PHP to one based upon the Ruby on Rails framework and an SQLite3 database. The PHP to RoR design will be discussed below in the application implementation section.

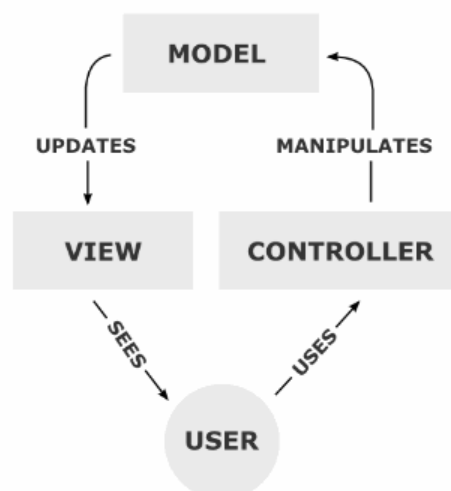
The decision to use SQLite3 instead of MySQL was predicated on several factors, both academic and practical. SQLite3, as its name would suggest, is a much more pared down DBMS than MySQL, for an

enterprise site or one expecting heavy traffic this would be a problem, but in our case the extra available overhead seemed inefficient and wasteful. SQLite3 is the default DBMS for a rails application, and efforts were made to migrate to a MySQL implementation but we encountered cross platform difficulty given that Daniel was developing on OS X, Sam was developing on Windows, and our deployment platform uses virtualized Linux machines. For an application intended to only have a single user, SQLite3 was more than enough to handle the workload.

For deployment to Heroku (a scalable cloud web-app service that will be described more in the application implementation section), PostgreSQL was used simply because that is what free Heroku accounts support. This did not effect our design or development strategy.

By default, Ruby on Rails uses Active Record, an abstraction placed on top of all querying to make data manipulation simpler (for example an SQL query to find the name of a book with id = 7 would be `SELECT B.name From Books B Where B.id = 7` whereas the equivalent Active Record query is `Book.find_by_id(7).name`), but in the spirit of this project almost all Active Record queries were overwritten (the only ones that were not are those which Rails generates automatically but where not actually used) and its functionality ignored. Instead, we used `Model_Name.find_by_sql("...")` which allows an SQL query to be passed in as a string and then executed.

Rails uses a Model-View-Controller (MVC) architecture pattern as seen in the diagram to the right. We had to violate this pattern slightly by putting SQL queries in the views for entities in order to practically overwrite the Active Record, but all of our “difficult” queries are located in the controllers, specifically in the



index function which drives the main pages of the site. We did not have any major difficulty implementing our non-trivial queries.

12. Application Implementation

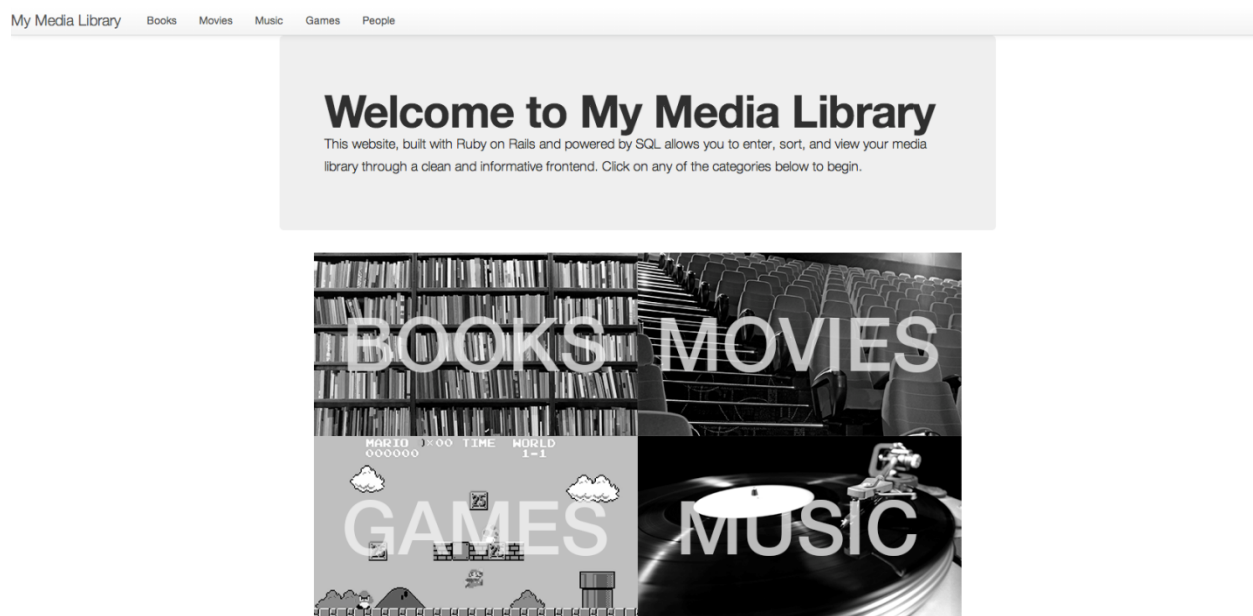
As mentioned in section 11, our application was implemented using Ruby on Rails. The Rails framework is an open source full-stack framework built around the language Ruby which allows for rapid, but powerful development of a fully featured website. RoR has many features that were useful for this project, namely the ability to scaffold (generate generic pages based on a model). By scaffolding entities, we could quickly create an index page for each type of media, pages to show each individual item, and forms to interact with them. Granted, the generated pages are very basic and needed extremely extensive reconfiguring to get to production level quality, but this ability to rapidly build pages made it easy to try out different ideas and make sure our database was working properly.

Rails encourages the use of CRUD (create, read, update, and destroy) actions with regard to all models involved, so these are what we implemented for all media types (creator entities such as artists and directors only have a create page available because frontend development was beginning to take a disproportionate amount of time, but non-styled RUD pages exist on the website).

In the interest of developing a polished project, we utilized a heavily modified version of Twitter's Bootstrap framework for styling. While Bootstrap makes rapid prototyping a decent interface much easier, the results tend to look very generic and awkward if used for anything other than one of Bootstrap's template pages. To this end 960.gs, a 960px wide grid system, along with custom CSS that can be found in the App > Assets > Stylesheets > Scaffolds.css file were used.

For ease of demonstration and to make sure that both of us were working from the most recent version, we deployed our application to Heroku (www.heroku.com). This service integrates with Github

to easily deploy to a virtualized service located on Amazon's elastic cloud. Doing so allowed us to not only host the application for free, but to supply a no-installation needed copy to the graders at <http://media-library341.herokuapp.com>.



The homepage of our application.

My Media Library Books Movies Music Games People

My Books (15)

View Quick Facts

Title	Author(s)	Year	Rating	Genre	Owner
American Gods	Neil Gaiman	2001	★★★★☆	Fantasy	Sam
Angels & Demons	Dan Brown	2009	★★★★☆	Mystery Thriller	Daniel
Catch 22	Joseph Heller	1960	★★★★★	Military Fiction	Daniel
Deception Point	Dan Brown	2006	★★★★☆	Science Fiction	Daniel
Ecotopia	Ernest Callenbach	1975	★★★☆☆	Utopian	Sam
Fahrenheit 451	Ray Bradbury	1953	★★★★☆	Dystopian	Sam
Good As Gold	Joseph Heller	1997	★★★☆☆	Fiction	Daniel
Something Wicked This Way Comes	Ray Bradbury	1962	★★★★☆	Fantasy Horror	Sam
The Algebraist	Ian M. Banks	2006	★★★★☆	Science Fiction	Daniel
The Bourne Supremacy	Robert Ludlum	1986	★★★★☆	Spy Fiction	Daniel
The Bourne Ultimatum	Robert Ludlum	1990	★★★★★	Spy Fiction	Daniel
The DaVinci Code	Dan Brown	2009	★★★★★	Mystery Thriller	Daniel
The Great Gatsby	F. Scott Fitzgerald	1953	★★★★☆	Classics	Daniel

The index page for Books. All media types follow the same generic format, which was designed to present the maximum amount of information without clutter.

View Quick Facts

Title	Author(s)	Year	Rating	Genre	Owner
American Gods	Neil Gaiman	2001	★★★★☆	Fantasy	Sam
<p>Description: The central premise of the novel is that gods and mythological creatures exist because people believe in them (similar to Tulpa). Immigrants to the United States brought with them dwarves, elves, leprechauns, and other spirits and gods. However, the power of these mythological beings has diminished as people's beliefs wane. New gods have arisen, reflecting America's obsessions with media, celebrity, technology, and drugs, among others.</p> <p>Comments: Won Hugo and Nebula awards</p> <p>Edit</p>					
Angels & Demons	Dan Brown	2009	★★★★☆	Mystery Thriller	Daniel

JQuery was used for all dynamic content.

My Books (15)

View Quick Facts

Authors with 3 or more 3 Star Books: Ray Bradbury Dan Brown

Person Who Owns the Most Books: Daniel

Authors Who Only Write Fantasy: Patrick Rothfuss Neil Gaiman

Authors Whose Books are All Rated > 3: Patrick Rothfuss Neil Gaiman Ray Bradbury F. Scott Fitzgerald Robert Ludlum Dan Brown Ian M. Banks

Users who own every Neil Gaiman Book: Sam

Title	Author(s)	Year	Rating	Genre	Owner
American Gods	Neil Gaiman	2001	★★★★☆	Fantasy	Sam

Our five non-trivial SQL queries are found at the top of each entity page

My Media Library Books Movies Music Games People

Editing book - American Gods

Title

Author:

☐ Patrick Rothfuss

☒ Neil Gaiman

☐ Ernest Cailenbach

☐ Ray Bradbury

☐ Joseph Heller

☐ F. Scott Fitzgerald

☐ Robert Ludlum

☐ Dan Brown

☐ Ian M. Banks

Year

Rating (1 - 5)

Genre

Person

Description

The central premise of the novel is that gods and mythological creatures exist because people believe in them (similar to Tulpa). Immigrants to the United States brought with them dwarves, elves, leprechauns, and other spirits and gods. However, the power of these mythological beings has diminished as people's beliefs wane. New gods have arisen, reflecting America's obsessions with media, celebrity, technology, and drugs, among others.

Comments

Won Hugo and Nebula awards

The page for editing a media item

My Media Library Books Movies Music Games People

New book

Title

Description

Comments

Author:

- ☐ Patrick Rothfuss
- ☐ Neil Gaiman
- ☐ Ernest Callenbach
- ☐ Ray Bradbury
- ☐ Joseph Heller
- ☐ F. Scott Fitzgerald
- ☐ Robert Ludlum
- ☐ Dan Brown
- ☐ Ian M. Banks

Year

Rating (1 - 5)

Genre

Person

Create Book

The editing form is also the creation form, for ease of use

New author

Name

Create Author

The creator editing pages are much simpler, but still functional

Daniel worked in Rails development over the past summer, so no major problems were encountered in creating the database. The initial setup of Rails on Sam's computer caused some problems as mentioned above, but all were eventually solved. Through extensive null and empty checking of passed variables, the application should be able to handle anything the user enters. All

inputs are also sanitized to prevent malicious users for modifying the database, though since the database is publically accessible through the front end, there would be no real point to this.

15. Conclusions

This project has been immensely helpful in learning both how to use SQL, how to create a useful database, and in learning how to get a number of different technologies to work together in a large project. Given that databases are used in almost every field relating to computer science, this information will be immensely helpful in the future.

Appendix 1. Installation

A. Installing Ruby on Rails

The Ruby on Rails installation process varies wildly between operating systems, so installing Rails will be left as an exercise to the reader. Information regarding Ruby on Rails can be found at <http://www.rubyonrails.org>, and all facets of it have been extensively documented on various websites. Our application was built using Rails 3.2.11 and Ruby 1.9.3p194. Using significantly older or newer builds of either may result in compatibility issues.

B. Installing My Media Library

Once Ruby on Rails has been successfully configured on your computer, installing our application is fairly trivial (it should be noted that a copy has been deployed to <http://media-library341.herokuapp.com> which can be used from any internet accessible device without any setup on that device). To begin using My Media Library:

- 1) From a command line terminal or shell, cd into the application directory
- 2) Run the command “bundle install” to install all application dependencies
- 3) Run command “rake db:create” to create a new SQLite3 database
- 4) Run command “rake db:migrate” to create all of the tables for the database

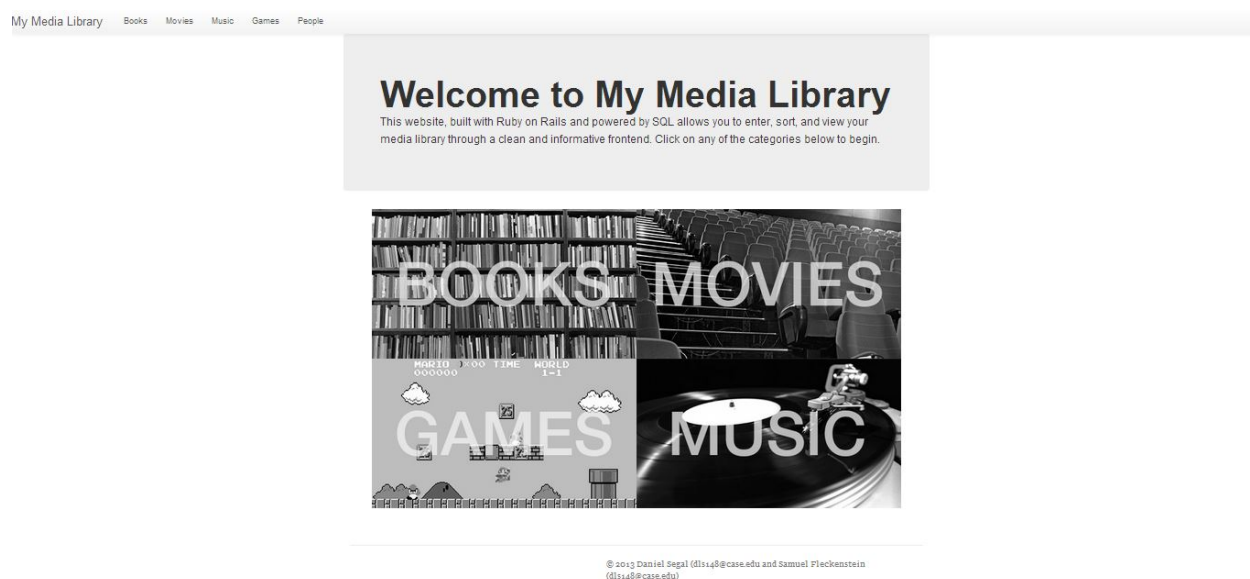
At this point the application will be ready for use in accordance with instructions found in **Section 17**, the user manual. There will be no information in the database, but a prepopulated db file can be downloaded at

www.daniel-segal.com/development.sqlite3

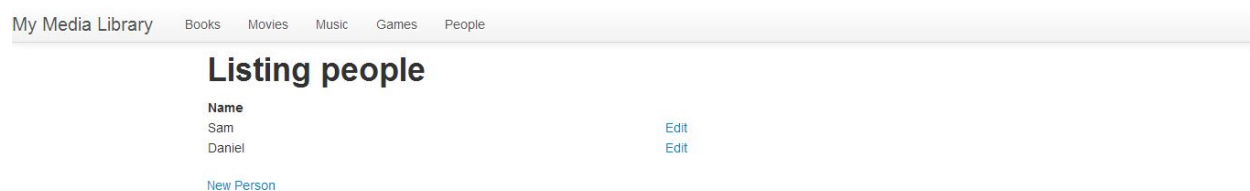
Overwriting the file with the same name in the /db folder of our application will populate the DB as we had it for testing.

Appendix 2: Users manual

To start the database, either go to <http://media-library341.herokuapp.com/> or follow the directions given in **Appendix 1** That will take you to this page:



To populate your database, first create a user by clicking on the People tab. You will now be at this page:



Where Sam and Daniel are the users we have entered in the database for testing purposes. Next, click on the New Person button and enter the name of the user. Now that you have a user, you will want to populate your database. Start by clicking on the Books tab. This will take you to a page like this:

Title	Author(s)	Year	Rating	Genre	Owner
American Gods	Neil Gaiman	2001	★★★★★	Fantasy	Sam
Angels & Demons	Dan Brown	2009	★★★★★	Mystery Thriller	Daniel
Catch 22	Joseph Heller	1960	★★★★★	Military Fiction	Daniel
Deception Point	Dan Brown	2006	★★★★★	Science Fiction	Daniel
Ecotopia	Ernest Callenbach	1975	★★★★★	Utopian	Sam
Fahrenheit 451	Ray Bradbury	1953	★★★★★	Dystopian	Sam
Good As Gold	Joseph Heller	1997	★★★★★	Fiction	Daniel
Something Wicked This Way Comes	Ray Bradbury	1962	★★★★★	Fantasy Horror	Sam
The Algebraist	Ian M. Banks	2006	★★★★★	Science Fiction	Daniel
The Bourne Supremacy	Robert Ludlum	1986	★★★★★	Spy Fiction	Daniel
The Bourne Ultimatum	Robert Ludlum	1990	★★★★★	Spy Fiction	Daniel
The DaVinci Code	Dan Brown	2009	★★★★★	Mystery Thriller	Daniel
The Great Gatsby	F. Scott Fitzgerald	1953	★★★★★	Classics	Daniel
The Martian Chronicles	Ray Bradbury	1950	★★★★★	Science Fiction	Sam
The Name of the Wind	Patrick Rothfuss	2007	★★★★★	Fantasy	Sam

The first step to creating a book is to create an author. Click the appropriate button at the bottom of the page enter the authors name. When the author has been created, click on the New Book button to create a book with a title, author, year, rating, genre, description and comments. It is essential that you create an author before attempting to create a book. This same process applies to Movies, Music, and Games.

To edit any field of an already existing item, click on the anywhere on the entry for the item. This will bring up a page similar to the following:

My Media Library Books Movies Music Games People

My Games (15)

[View Quick Facts](#)

Title	Studio(s)	Year	Rating	Genre	Owner
BioShock	Irrational Games	2007	★★★★★	Shooter	Daniel
<p>Description: BioShock is a first-person shooter video game developed by Irrational Games (at the time, named 2K Boston), and published by 2K Games.[12] The game was released for Microsoft Windows and Xbox 360 platforms in August 2007; a PlayStation 3 port by Irrational, 2K Marin, 2K Australia and Digital Extremes was released in October 2008, and a Mac OS X port by Feral Interactive in October 2009. A mobile version was developed by IG Fun.[13] The game's concept was developed by Irrational's creative lead, Ken Levine, and was based on the ideas of Objectivism as highlighted by Ayn Rand, while incorporating influences from other authors such as George Orwell. The game is considered a spiritual successor to the System Shock series, which many of Irrational's team including Levine had worked on previously.</p> <p>Comments: BioShock received critical acclaim and was particularly praised by critics for its "morality-based" storyline, immersive environment and its unique setting. It received several Game of the Year awards from different media outlets, including from BAFTA, Game Informer, Spike TV, and X-Play. Since its release a direct sequel has been released, BioShock 2 by 2K Marin, as well as a third game entitled BioShock Infinite by Irrational Games.</p> <p>Edit</p>					
Borderlands	Gearbox	2009	★★★★★	Shooter	Daniel
Borderlands 2	Gearbox	2012	★★★★★	Shooter	Daniel
Dark Souls	Namco Bandai	2011	★★★★★	Medieval Fighting	Daniel
Halo: Combat Evolved	Bungie	2001	★★★★★	Science Fiction	Daniel
Madden NFL 11	EA	2010	★★★★★	Sports	Sam
Madden NFL 12	EA	2011	★★★★★	Sports	Sam
Mass Effect	Biossage	2007	★★★★★	Science Fiction	Sam

Click on the Edit button and make whatever changes you want, then click on the Update button at the bottom of the page to save your changes.

To delete an item, follow the same process as with editing, only click on the Delete button at the bottom of the page, rather than the Update button.

To see the Quick Facts for each type of media, click on View Quick Facts. To view more information about any item, simply click on the entry you want more information on.

Programmers Manual

Before attempting to modify any portion of this code, any programmer is recommended to have a good working knowledge of Ruby on Rails. While Rails simplifies many aspects of building a website, it can be fairly opaque without a good understanding of its underlying architecture.

The following classes and commands use the Books & Authors models, but can be generalized to apply to any other set of Media & Creator (the only subtle change being that the Games_Studios table has the two in reversed order from Authors_Books due to alphabetic order naming conventions in Rails).

To modify any part of the execution, the code will be located in the controller for that model (i.e. books_controller).

The Index Page- Controller

```
def index
```

```
    //Controls the count located next to the media type on the index page
```

```
    @count = Book.find_by_sql('SELECT COUNT(*) FROM Books B GROUP BY B.id').size
```

```
    //Explanations of these SQL queries can be found in section 7
```

```
    @threeOrMore = Author.find_by_sql(" SELECT A.name, A.id
```

```
        FROM Authors A, Books B, Authors_Books C
```

```
        WHERE A.id = C.author_id AND C.book_id = B.id AND
```

```
        B.rating >= 3
```

```
        GROUP BY A.id
```

```
        HAVING COUNT(B.id) >= 3")
```

```
    @hasMost = Person.find_by_sql("SELECT P.name
```

```
        FROM People P, (SELECT COUNT(B1.id) AS num_books, B1.person_id AS id
```

```
        FROM Books B1
```

```
        GROUP BY B1.person_id) AS B1
```

```
        WHERE B1.id = P.id
```

```
        AND NOT EXISTS (SELECT COUNT(B2.id), B2.person_id
```

```
        FROM Books B2
```

```
        GROUP BY B2.person_id
```

```
        HAVING COUNT(B2.id) > B1.num_books)").first()
```

```
    @genreOnly = Author.find_by_sql("
```

```
        SELECT A.name
```

```

FROM Authors A

WHERE NOT EXISTS (

  SELECT *

  FROM Books B, Authors_Books C

  WHERE C.author_id = A.id AND C.book_id = B.id AND B.genre <> 'Fantasy')") )

```

```
@good = Author.find_by_sql("
```

```

  SELECT A.name

  FROM Authors A

  WHERE NOT EXISTS (

    SELECT *

    FROM Books B, Authors_Books C

    WHERE B.rating < 3 AND A.id = C.Author_id

    AND C.book_id = B.id) ")

```

```
@fan = Person.find_by_sql("SELECT P.name
```

```

  FROM People P

  WHERE NOT EXISTS (

    SELECT B.title

    FROM Books B, Authors D, Authors_Books A

    WHERE B.id = A.book_id AND D.id = A.author_id AND D.name = 'Neil Gaiman'

    AND P.id NOT IN(

      SELECT P1.id

      FROM People P1

      WHERE P1.id = B.person_id)))")

```

```

@books = Book.find_by_sql("SELECT * FROM BOOKS ORDER BY title")

//This tells the program which page is associated with this action
respond_to do |format|

    format.html # index.html.erb

    format.json { render json: @books }

end

end

```

To display the above queries on the index page, the following code is used (NOTE anything in `<% %>` is Ruby. File types for webpages in this application are `*.html.erb`, or hypertext markup language with embedded ruby. If the brackets have `<%= %>`, their contents will be printed. Anything with `@*` is an instance variable).

The Index- View

// If the variable is set to nil (no values in database), or contains nothing (no author/ book meets the criteria, print N/A

//To return an author with 3 or more 3 star books

```

<% if @threeOrMore.nil? || @threeOrMore.empty? %>

    N/A

<% else %>

<% @threeOrMore.each do |a| %>

    <%= a.name %>

<% end %>

<% end %>

```



```
//To return the name of the person who has the most books
```

```
<% if @hasMost.nil?%>
```

```
  N/A
```

```
<% else %>
```

```
<%= @hasMost.name %>
```

```
<% end %>
```

```
//To return the name of an author who only writes fantasy
```

```
<% if @genreOnly.nil? || @genreOnly.empty? %>
```

```
  N/A
```

```
<% else %>
```

```
<% @genreOnly.each do |a| %>
```

```
<%= a.name %>
```

```
<% end %>
```

```
<% end %>
```

```
//To return the name of an author who only writes books with >3 star reviews
```

```
<% if @good.nil? || @good.empty? %>
```

```
  N/A
```

```
<% else %>
```

```
<% @good.each do |a| %>
```

```
<%= a.name %>
```

```
<% end %>
```

```
<% end %>
```

```
//To return an author whose every work is owned by one person
```

```
<% if @fan.nil? %>
```

```
N/A
```

```
<% else %>
```

```
<% @fan.each do |a| %>
```

```
<%= a.name %>
```

```
<% end %>
```

```
<% end %>
```

The Edit Page- Controller

```
def edit
```

```
@book = Book.find_by_sql("SELECT * FROM Books B WHERE B.id = " + params[:id]).first()
```

```
@people = Person.find_by_sql("SELECT * FROM PEOPLE")
```

```
@authors = Author.find_by_sql("SELECT * FROM AUTHORS")
```

```
end
```

The Edit Page- View

```
<div class="container_16">
```

```
<div class="grid_16">
```

```
<h1>Editing book - <%= Book.find_by_sql('SELECT title FROM Books B WHERE B.id = ' +  
params[:id] + ')[0]['title'] %></h1>
```

```
<%= render 'form' %>
```

```
<div class="centered">
```

```
<%= button_to "Delete Book", { :controller => :books, :action => 'destroy', :id =>  
@book.id }, :method => :delete, :confirm => "Are you sure ?"%>
```

```
</div></div></div>
```

The Edit/ New Form

```

<%= form_for(@book) do |f| %>

//If any of the values violate integrity constraints alert the user

<% if @book.errors.any? %>

  <div id="error_explanation">

    //List violating errors

    <h2><%= pluralize(@book.errors.count, "error") %> prohibited this book from being saved:</h2>

    <ul>

      <% @book.errors.full_messages.each do |msg| %>

        <li><%= msg %></li>

      <% end %>

    </ul>

  </div>

<% end %>

<div class="grid_7">

  <div class="field">

    <%= f.label :title, :class => "form_field" %><br />

    <%= f.text_field :title %>

  </div>

  Author:

  <% for author in Author.find_by_sql("SELECT * FROM AUTHORS") %>

<div>

//Render a list of checkboxes to select the author

```

```

    <%= check_box_tag "book[author_ids][]", author.id, @book.authors.include?(author) %>

    <%= author.name %>

</div>

<% end %>

//Define the Year of the Book

<div class="field">

    <%= f.label :year, :class => "form_field" %><br />

    <%= f.number_field :year %>

</div>

//Define the Rating of the Book

<div class="field">

    <%= f.label "Rating (1 - 5)", :class => "form_field"%><br />

    <%= f.number_field :rating %>

</div>

//Define the Genre of the Book

<div class="field">

    <%= f.label :genre, :class => "form_field" %><br />

    <%= f.text_field :genre %>

</div>

//Define the Owner of the Book

<div class="field">

    <%= f.label :person, :class => "form_field" %><br />

    <%= f.collection_select :person_id, @people, :id, :name %>

</div>

</div>

```

```

<div class="grid_8">

//Describe the book

<div class="field">

  <%= f.label :description, :class => "form_field" %><br />

  <%= f.text_area :description, :class => "text" %>

</div>

  <div class="field">

//Coments on the book

  <%= f.label :comments, :class => "form_field" %><br />

  <%= f.text_area :comments, :class => "text lower" %>

</div>

</div>


<div class="actions submit">

  <%= f.submit %>

</div>


<% end %>

```

New – Controller

```

def new

  @book = Book.new

  @people = Person.all

  @authors = Author.all

  respond_to do |format|

```

```
format.html # new.html.erb

format.json { render json: @book }

end

end
```

Routes

If the programmer decides to add any additional pages to the website, a new route will need to be configured. Documentation for this exists in the Config > routes.rb file. The following routes currently are

```
MediaLibrary::Application.routes.draw do
```

```
  resources :people
  resources :studios
  resources :artists
  resources :authors
  resources :directors
  resources :games
  resources :musics
  resources :books
  resources :movies
  root :to => 'static_pages#index'
```

```
end
```

The resources: symbol supplies each with needed HTTP requests. All routes can be viewed using rake routes.

No password or username is needed for the database.

The application was designed to follow the RESTful strategy of architecture. Rails handles enough of the underlying actions that a comprehensive knowledge of the architecture will allow for easy manipulation.