

# FilmReel

CMPS 3420-01 – Database Systems  
Fall 2020

Abraham Aldana  
Cesar Lara  
Adrian Telan

# Table of Contents

---

<b>Phase I: Conceptual Design .....</b>	<b>4</b>
1.1 - Fact-Finding Techniques and Information Gathering.....	4
1.1.1 – Introduction to the Organization .....	4
1.1.2 – Description of Fact-Finding Techniques .....	4
1.1.3 – The Miniverse of Interest.....	5
1.1.4 – Itemized Description of Entity Sets and Relationship Sets .....	5
1.1.5 – User Groups, Data Views, and Operations .....	9
1.2 - Conceptual Database Design.....	9
1.2.1 – Entity Type Descriptions.....	9
1.2.2 – Relationship Type Description.....	15
1.2.3 – Related Entity Types .....	15
1.2.4 – ER Diagram.....	16
 <b>Phase 2: Conversion from Conceptual to Relational Database .....</b>	 <b>17</b>
2.1 – The ER and Relational models .....	17
2.1.1 – Descriptions of ER and Relational Models .....	17
2.1.2 – Model Comparisons .....	18
2.2 – Conceptual to Logical Conversion Process .....	19
2.2.1 – Converting Entity Types to Relations .....	19
2.2.2 – Converting Relationship Types to Relations.....	19
2.2.3 – Converting Extended Types to Relations .....	20
2.2.4 – Database Constraints.....	21
3 – Results of ER to Relational Conversion.....	22
3.1 – Relation Schema.....	22
3.2 – Sample Data .....	25
4 – Sample Queries .....	45
4.1 – Design of Queries .....	45

4.2 – Relational Algebra Expressions .....	46
4.3 – Relational Calculus Expressions .....	47

## Phase 3: Relational Normalization and Physical Implementation .....49

5 – Normalization of the Relations .....	49
5.1 – Normalization Process .....	49
5.2 – Application to the Relational Model .....	51
6 – Database Implementation .....	51
6.1 – Background Information .....	51
6.2 – Schema and Hosting .....	52
7 – Query Implementation .....	61

## Phase 4: Database Programming (Stored Procedures, Functions, Views, Triggers).....69

8 – Programming Logic for SQL .....	69
8.1 – Introduction .....	69
8.2 – Syntax of Programming Logic .....	69
8.3 – Implementation.....	71

## Phase 5: Graphical User Interface (GUI) Application Development .....80

9 – GUI Development .....	80
9.1 – GUI Functionalities and User Groups.....	80
9.2 – GUI Programming .....	92

# **Phase I: Conceptual Design**

## **1.1 - Fact-Finding Techniques and Information Gathering**

### **1.1.1 – Introduction to the Organization**

FilmReel is our group-created movie-tracking website that will allow people to access its ever-growing database of films. This website is designed to cater to people with an interest in movie-watching. Although it is not necessary to simply browse the films stored in the database, the website will allow visitors to register for accounts. Upon completing registration, accountholders will have the ability to write reviews for films and organize films into lists that they can share with friends or other movie-watchers. A web-based interface will be developed to make accessing the database, the organizational features, and the reviews as intuitive of an experience as possible.

### **1.1.2 – Description of Fact-Finding Techniques**

The internet was an invaluable source of information in our fact-finding process and it helped us get an accurate idea of what we needed to include in our database. For attributes related to films, IMDb.com was the most useful website we could find. Because maintaining the record of films will be left entirely to the database administrator, it was important to establish exactly what information we needed to include for films. Other websites like MyAnimeList.com and PCPartPicker.com served as excellent examples of systems that allow their users a quick and simple method to create an account, search the various entities of their respective databases (anime and computer parts), the creation of lists to organize instances of such entities, and the ability to provide public feedback in the form of written reviews. These are all components that we wanted to consider when designing our database.

The internet was helpful for learning what data and operations we could consider implementing. However, FilmReel is ultimately meant for any person who has a significant interest in watching movies and therefore our team decided to inquire about what aspects such a typical user would desire. Between our three group members, we compiled a list of friends, relatives, and peers who we recognized to have a significant interest in watching movies. This was based on their frequent viewings of films as well as their often-vocal critiques of such films. Our team asked the people on this list various questions regarding their decision-making process for whether to watch a certain movie or not, their tendencies to watch unknown movies of their favorite genres or studios, and if they let either critic or user reviews affect their opinions on films. Both the internet and the survey of avid movie-watchers served as effective methods to design our database.

### 1.1.3 – The Miniverse of Interest

This database is designed to be used by sites which tracks what users have watched and will watch such as IMDb. The database will contain user data, film data, film list data, and film review data. The database will also involve handling how users view films, how users make lists, and how users write reviews.

### 1.1.4 – Itemized Description of Entity Sets and Relationship Sets

#### Entities

##### Users

- Account\_id
  - Unique ID for each user.
  - Integer
- First\_Name
  - User's first name.
  - Varchar
- Last\_Name
  - User's last name.
  - Varchar
- Username
  - User's username.
  - Varchar
- Email
  - User's email.
  - Varchar
- Password
  - User's password.
  - Varchar
- Birthdate
  - User's date of birth.
  - Date

##### Film

- Film\_ID
  - Unique ID for each film.
  - Integer
- Film\_Name
  - Name of film.
  - Varchar

- Release\_Date
  - Date film was released.
  - Date
- Synopsis
  - Description of film.
  - Varchar
- Duration
  - Length of film in minutes.
  - Integer
- Image
  - Cover image for film.
  - Varchar
- Rating
  - Age rating of film.
  - Varchar

### **Genre**

- Genre\_ID
  - Unique ID for each genre.
  - Integer
- Name
  - Name of the genre
  - Varchar

### **Film Genre**

- Film\_ID
  - ID for each film.
  - Integer
- Genre\_ID
  - ID of the associated genre
  - Integer

### **Review**

- Review\_ID
  - Unique ID for film review.
  - Integer
- Title
  - Title for review.
  - Varchar
- Score
  - Score for film (1/10)
  - Integer
- Description

- Description of review for film.
  - Varchar
- Review\_Date
  - Date of review.
  - Date

### **List**

- List\_ID
  - Unique ID for each list.
  - Integer
- List\_Title
  - Title of the list
  - Varchar

### **List Items**

- List\_ID
  - ID for each item.
  - Integer
- Film\_ID
  - ID of the enlisted film
  - Integer

Example: A user views a film.

## **Relationships**

### **Makes**

- Description – User **makes** List
- Entities Involved – User, List
- Attributes – none
- Cardinality – 1..N
- Participation/Constraints - Partial for User, Total for List

### **Views**

- Description – User **views** Film
- Entities Involved – User, Film
- Attributes – none
- Cardinality – M..N
- Participation/Constraints - Partial for both

### **References**

- Description – Film **references** Review
- Entities Involved – Film, Review
- Attributes – none
- Cardinality – 1..N
- Participation/Constraints - Partial for Film, Total for Review

### **Has**

- Description – List **has** List Items
- Entities Involved – List, List Items
- Attributes – none
- Cardinality – 1..N
- Participation/Constraints - Partial for List, Total for List Items

### **Enlists**

- Description – List Items **enlists** Films
- Entities Involved – List Items, Film
- Attributes – none
- Cardinality – 1..N
- Participation/Constraints - Total for both

### **Writes**

- Description – User **writes** Review
- Entities Involved – User, Review
- Attributes – none
- Cardinality – 1..N
- Participation/Constraints - Partial for User, Total for Review

### **Categorized as**

- Description – Film **categorized as** Film\_Genre
- Entities Involved – Film, Film\_Genre
- Attributes – none
- Cardinality – M..N
- Participation/Constraints – Total for both

### **Categorizes**

- Description – Genre **categorizes** Film\_Genre
- Entities Involved – Film\_Genre, Genre
- Attributes – none
- Cardinality – M..N
- Participation/Constraints - Partial for Genre, Total for Film\_Genre



Example: A user makes a list.

### 1.1.5 – User Groups, Data Views, and Operations

The sole user group will be individuals who want to browse data regarding the various films stored within our database. This group can therefore be known as ‘users’. Also, because there exists only a single user group, there will be one data view. Users must be able to view every instance of ‘Film’ entity and its attributes as well as all instances of the ‘Review’ entity that are directly referencing such instance of Film. The user must also be allowed to create an account with a user-created combination of a unique username and a password. After having established an account, users need to be able to create, populate, and save lists of films as well as leave feedback for films as they desire. Generating and manipulating instances of ‘List’ and ‘Review’ are the only other way users will be affecting any data in the database aside from the creation and personalization of their user accounts for FilmReel.

## 1.2 - Conceptual Database Design

### 1.2.1 – Entity Type Descriptions

#### Users

**Description:** The person using the database

**Candidate Keys:** Account\_id, First\_Name, Last\_Name, Username, Email

**Primary Key:** Account\_id

**Strong/Weak:** Strong

**Fields to be Indexed:** Account\_id, First\_Name, Last\_Name, Username, Email

Attribute	Account ID	First Name	Last Name	Username	Email	Password	Birth Date
Description	Unique Identifier	First Name	Last Name	Username	Email	Password	Date of Birth
Domain/ Type	Integer	Varchar	Varchar	Varchar	Varchar	Varchar	Date
Value/ Range	1,999999	255 chars	255 chars	255 chars	255 chars	255 chars	

<b>Default</b>	Auto Increment	N/A	N/A	N/A	N/A	N/A	N/A
<b>Null Allowed?</b>	No	Yes	Yes	No	No	No	No
<b>Unique</b>	Yes	No	No	No	No	No	No
<b>Single/ Multi</b>	Single	Single	Single	Single	Single	Single	Single
<b>Simple/ Composite</b>	Simple	Simple	Simple	Simple	Simple	Simple	Simple

## Film

**Description:** The movie or film in the database

**Candidate Keys:** Film\_ID, Film\_Name

**Primary Key:** Film\_ID

**Strong/Weak:** Strong

**Fields to be Indexed:** Film\_ID, Film\_Name, Release\_Date, Genre

Attribute	Film ID	Film Name	Release Date	Genre	Synopsis	Duration	Image	Rating
<b>Description</b>	Unique Identifier	Name of film	Film release date	Film genre	Film description	Film duration	Film poster image	Film age rating
<b>Domain/ Type</b>	Integer	Varchar	Date	Varchar	Varchar	Time	Image	Varchar
<b>Value/ Range</b>	1,999999	255 chars		255 chars	2000 chars			5 chars
<b>Default</b>	Auto Increment	N/A	N/A	N/A	N/A	N/A	N/A	N/A
<b>Null Allowed?</b>	No	No	No	No	No	No	Yes	Yes
<b>Unique</b>	Yes	No	No	No	No	No	No	No

<b>Single/ Multi</b>	Single	Single	Single	Multi	Single	Single	Single	Single
<b>Simple/ Composite</b>	Simple	Simple	Simple	Simple	Simple	Simple	Simple	Simple

## Genre

**Description:** A genre to categorize a film

**Candidate Keys:** Genre\_ID

**Primary Key:** Genre\_ID

**Strong/Weak:** Strong

**Fields to be Indexed:** Genre\_ID

Attribute	Genre ID	Name
<b>Description</b>	Unique Identifier	Name of Genre
<b>Domain/ Type</b>	Integer	Varchar
<b>Value/ Range</b>	1,999999	255 chars
<b>Default</b>	Auto Increment	N/A
<b>Null Allowed?</b>	No	No
<b>Unique</b>	Yes	No
<b>Single/ Multi</b>	Single	Single
<b>Simple/ Composite</b>	Simple	Simple

## Film\_Genre

**Description:** The films and their associated genres

**Candidate Keys:** Film\_ID, Genre\_ID

**Primary Key:** Film\_ID

**Strong/Weak:** Weak

**Fields to be Indexed:** Film\_ID, Genre\_ID

Attribute	Film_ID	Genre_ID
Description	ID of the Film	ID of the genre
Domain/ Type	Integer	Integer
Value/ Range	1,999999	1,999999
Default	N/A	N/A
Null Allowed?	No	No
Unique	No	No
Single/ Multi	Single	Single
Simple/ Composite	Simple	Simple

## Review

**Description:** Review left by users about the film

**Candidate Keys:** Review\_ID, Title, Score

**Primary Key:** Review\_ID

**Strong/Weak:** Weak

**Fields to be Indexed:** Review\_ID, Title, Score, Review\_Date

Attribute	Review ID	Title	Score	Description	Review Date
Description	Unique Identifier	Review Title	Review Score	Review Description	Review Date
Domain/ Type	Integer	Varchar	Integer	Varchar	Date
Value/ Range	1,999999	255 chars	1,10	255 chars	
Default	Auto Increment	N/A	N/A	N/A	Current Date
Null Allowed?	No	No	No	Yes	No
Unique	Yes	No	No	No	No
Single/ Multi	Single	Single	Single	Single	Single
Simple/ Composite	Simple	Simple	Simple	Simple	Simple

## List

**Description:** A list made by the user

**Candidate Keys:** List\_ID

**Primary Key:** List\_ID

**Strong/Weak:** Weak

**Fields to be Indexed:** List\_ID, List Title

Attribute	List ID	List Title
Description	Unique Identifier	Title of list
Domain/ Type	Integer	Varchar

<b>Value/ Range</b>	1,999999	255 chars
<b>Default</b>	Auto Increment	N/A
<b>Null Allowed?</b>	No	No
<b>Unique</b>	Yes	No
<b>Single/ Multi</b>	Single	Single
<b>Simple/ Composite</b>	Simple	Simple

## List\_Items

**Description:** The enlisted films of the lists.

**Candidate Keys:** List\_ID, Film\_ID

**Primary Key:** List\_ID

**Strong/Weak:** Weak

**Fields to be Indexed:** List\_ID, Film\_ID

Attribute	List_ID	Film_ID
<b>Description</b>	ID of the List	ID of the Film
<b>Domain/ Type</b>	Integer	Integer
<b>Value/ Range</b>	1,999999	1,999999
<b>Default</b>	N/A	N/A
<b>Null Allowed?</b>	No	No

<b>Unique</b>	No	No
<b>Single/ Multi</b>	Single	Single
<b>Simple/ Composite</b>	Simple	Simple

## 1.2.2 – Relationship Type Description

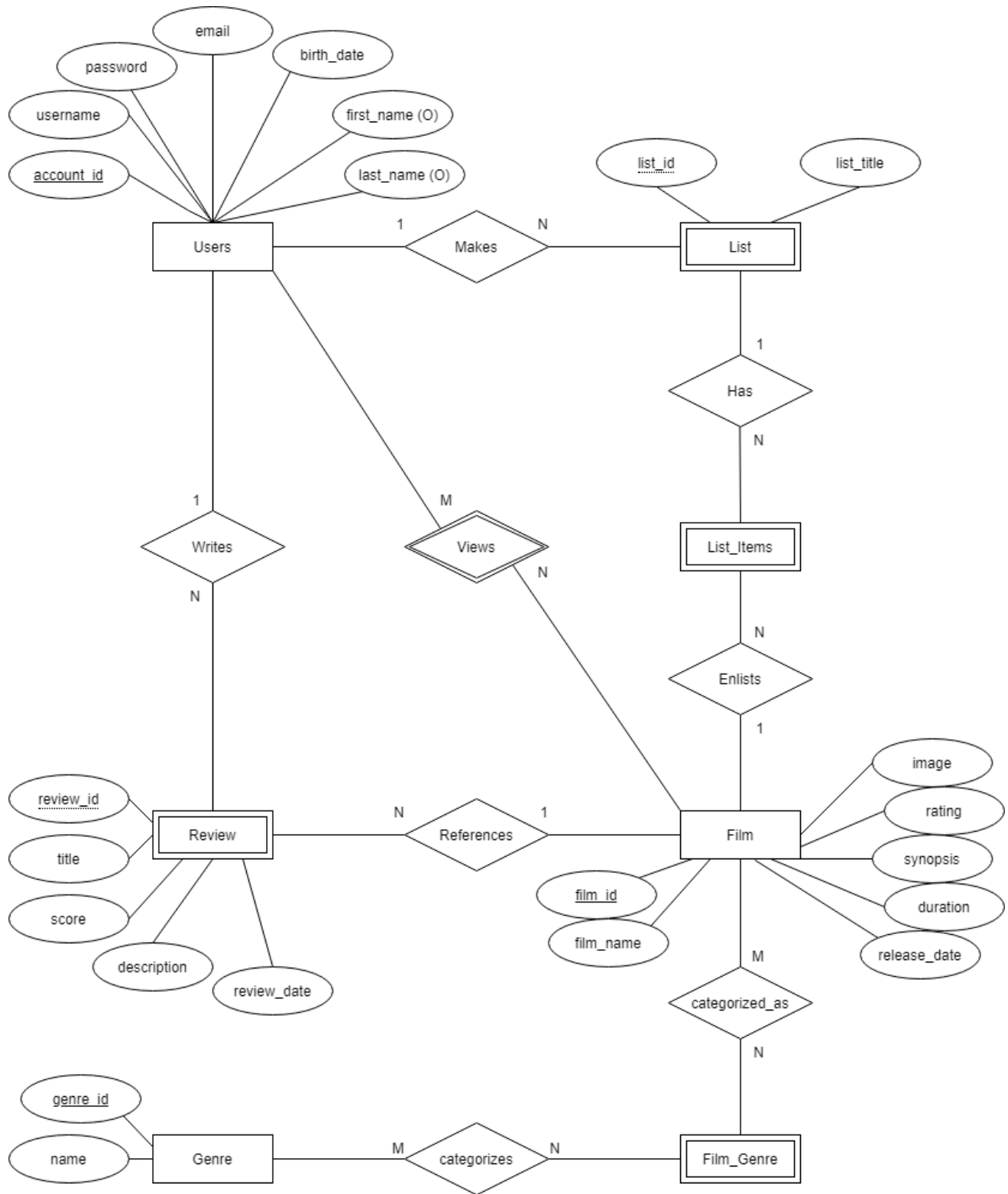
Relationship	Makes	Views	References	Has	Writes	Enlists	Categorized_ as	Categorizes
<b>Description</b>	User makes List	User views Film	Film references Review	List has List Items	User writes Review	List Items enlists Films.	Film categorized as a Film Genre	Genre categorizes Film Genre.
<b>Entities Involved</b>	User, List	User, Film	Film, Review	List, List Items	User, Review	List Items, Films	Film, Film_Genre	Genre, Film_Genre
<b>Cardinality</b>	1..N	M..N	1..N	1..N	1..N	1..N	M..N	M..N
<b>Attributes</b>	None	None	None	None	None	None	None	None
<b>Participation Constraint</b>	Partial for User, Total for List	Partial for both	Partial for Film, Total for Review	Partial for List, Total for List Items	Partial for User, Total for Review	Partial for Film, Total for List_Items	Total for both	Partial for Genre, Total for Film_Genre

## 1.2.3 – Related Entity Types

Specialization refers to the formation of subclasses of an entity type. Subclasses inherit the attributes of their parent class, and they can add attributes to specific subclasses. This database does not use any specializations since there are not any attributes which need to be replicated across entities.

Generalization refers to grouping related subclasses into a superclass. This was not used for this database since none of the entities have any significant similarities to justify the need for generalization.

## 1.2.4 – ER Diagram





# Phase 2: Conversion from Conceptual to Relational Database

## 2.1 – The ER and Relational models

This section will be an overview of the two types of database modeling techniques that will have been utilized by the completion of this second phase as we convert from a conceptual database to a relational one. Before we complete this conversion, we will first describe the history and the traits of both the entity-relationship model and the relational model. Afterwards, the two models will be compared as we highlight each one's advantages, disadvantages, similarities, and differences.

### 2.1.1 – Descriptions of ER and Relational Models

The now-common method of data diagramming known as entity-relationship (E-R) modeling was first introduced in 1976 by Harvard graduate and computer scientist Peter Chen in his paper "The Entity-Relationship Model - Toward a Unified View of Data". Chen developed this original model for the purpose of database design. The result was an intuitive way for anyone working with sets of related data to diagram it in such a way that others could read it and easily understand it as well. This is accomplished through the way that data entities and their respective attributes are shown as connected to each other through relationships. For example, attributes 'year', 'make', and 'model' will have actual connecting lines between them and the 'car' entity that they are networked to. Meanwhile, this 'car' entity could then be connected to a completely different entity called 'owner' to represent the relationship of a car belonging to an owner. What makes E-R modeling even easier to understand is how entities, attributes, and relationships are each shown as having different two-dimensional polygons representing them in the diagram. As a result, no labels are needed to distinguish between what type of element any instance of data is. Overall, the ER model is effective for getting information together quickly and making it easy for readers to understand the general layout of the database.

Another method of modeling a database was introduced earlier in 1969 by IBM computer scientist Edgar Frank Codd in his research report for IBM titled *Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks* and explained in further detail his paper "A Relational Model of Data for Large Shared Data Banks" which was published a year later in 1970. This method, which is referred to as the relational model, was aimed at showing all components of the database as sets of tuples and referring to them as relations. This is similar to the basic concept of tables comprised of rows and columns. Such a table is created for each set of related data values and referred to as a relation. When working with the relational model, all entities and relationships would require their own relation with tuples composed of the attributes involved. Also, the relational model takes constraints like primary keys and foreign keys into consideration. Because of these traits and the fact that each relation can be diagrammed as a table, the relational model

allows for database designers and readers to create and read the data while understanding exactly what attributes will be associated with each data element.

## 2.1.2 – Model Comparisons

Both the E-R model and the relational model are useful tools for diagramming a database. Despite having some similarities, it can be beneficial to utilize both models during different parts of the design process because they each have unique advantages in their respective part of the database creation procedure.

One of the greatest strengths of the E-R model is that it is a graphical approach to designing a database and can therefore be a popular option to use during conceptual design. As mentioned earlier, an E-R model is used to create a visual representation of the interconnected entities of data as a network. Because each data element has its own space on the graph and a shape typically indicative of its nature (either a relationship, entity, or attribute), both the creators and readers of the model can have an easier time understanding what data exists and how it relates to each other. Another advantage of the E-R model is that it is not necessary for the designer to expend time to include redundant pieces of data in the cases of primary keys and foreign keys. Such is the case with our E-R model in section 1.2.4 where the attributes 'film\_id' and 'account\_number' will be associated with the entity 'List' as foreign-keys. However, they are only included once as primary-key attributes of their respective entities.

While opting to not include redundant data could save time during the conceptual database design process, the lack of such items is also one of the disadvantages of the E-R model. While again referencing the model in section 1.2.4, a reader would require additional information to recognize that the 'List' entity is going to be relying on two foreign keys from two different entities since it is not explicitly shown. The disadvantage to having a simplistic design for E-R model diagrams is that there is a limit to how much information can be displayed at once. While the E-R model is beneficial for showing what data is going to be included and how it immediately relates to the rest, it is likely not going to be the sole model used during the entirety of the of the database's creation process.

In the relational model, the database is showed as sets of tables called relations. Considering tables are used for representation, an advantage of the relational model is that the data is easy to understand and work with because of the simplicity of tables (composed of only rows and columns). Because the data is represented through tables, this also allows for the use of queries to access and navigate throughout the database even if multiple relations must be traversed. Despite the lack of structure, the relational model is still effective in allowing users to traverse as many tables as needed to access information. However, this is also one of the disadvantages of the relational model. Due to the simplicity of the design where the data is purely tables, it is not immediately apparent what and how certain pieces of data relate to each other. Another disadvantage of this model is that the difficulty in understanding the relations between data is heightened further as the database grows.

Both the E-R and relational models are methods of organizing large sets of information into a database. However, the E-R model displays the information through a visually

friendly graphical network showing attributes, entities, and relationships connected with each other to show relation while the relational model represents the data through tables. In some E-R models (such as the one we use in section 1.2.4), foreign keys are not explicitly shown to be tied to weak entities while the relational model factors in all keys and other constraints during the design of the relations. Overall, both models have their advantages and disadvantages along with some similarities. In the following section we will discuss converting from our E-R model to a relational one.

## 2.2 – Conceptual to Logical Conversion Process

This section will focus on explaining the actual process for converting an E-R model to a relational model. First, we will describe the methods for converting all entity types to relations. Next, we will explain the process for converting relationships of varying types to relations. This is followed by an explanation for how extended types, such as inherited attributes or superclass/subclass instances, are converted to relations. Finally, we will explain what database constraints are and how several types of constraints can impact a database.

### 2.2.1 – Converting Entity Types to Relations

To convert strong entities into the relational model, a relation with the same attributes and key attributes as the strong entity is created. In the event of multiple key attributes, one would be indicated as the primary key. Thus, in the case of the strong entity “User,” a relation would be created as follows:

Users(account\_id, username, password, email, birth\_date, first\_name, last\_name)

The process is similar when converting weak entities into the relational model, but since weak entities are derived from strong entities, the weak entity relations must have the primary key of the owning entity. Thus, in the case of the weak entity “Review” which is derived from the strong entities “User” and “Film,” a relation would be created as follows:

Review(review\_id, account\_id, film\_id, title, score, description, review\_date)

The above examples demonstrate simple conversions where the entities and relationships only possess single, simple attributes. If there were any composite attributes, then the composite attributes would be segmented into individual attributes. If there were any multi-value attributes, then there would be a new relation for each multi-value attribute with a foreign key connecting this new relation to the main relation. There were no entities or relationships in this project’s diagram where there were multi-value attributes, so there is no example demonstrating this process as it is irrelevant in this project’s context.

### 2.2.2 – Converting Relationship Types to Relations

There are three methods to convert binary relationships into relations. The first method would be to use foreign keys. Essentially, given that two entities are linked by a

relationship, the entity which has total participation would obtain the primary key of the other entity as well as the attributes of the relationship.

The second method would be to merge the entities which have total participation with one another. In essence, the entities' names would be merged into one relation, and the relation attributes would have all the attributes of both entities. The advantage of this method over the foreign key method is that it is easy to create a new relation which perfectly demonstrates the link between its constituent relations. However, the drawback is that it can only be used on relationships and entities which have one-to-one total participation relationships with one another.

The third method would be to cross-reference method where two entities create a new relation which would contain only the primary keys of these two entities. The advantage of this method is that all considerations when converting these relationships and entities into the relational model are covered: there is no oversight when using the cross-reference method. However, it may add unnecessary complexity to the relational model, especially when either a merge or the foreign key methods can be applied.

When converting one-to-many relationships into the relational model, the same methods, excluding the merge method, are available with the same advantages and drawbacks. When converting many-to-many relationships, only the cross-reference method is available for use.

If there were any multi-value attributes, then there would be a new relation for each multi-value attribute with a foreign key connecting this new relation to the main relation.

When converting n-ary relationships where multiple entities are linked by a single relationship, then when converting these entities and relationship into the relational model, a new relation which contains the primary keys and attributes of each entity is created and used as the relation for the relationship.

### **2.2.3 – Converting Extended Types to Relations**

When converting inherited types into the relational model, there are four methods of conversion. The first method is used to create a superclass with many subclasses. The superclass will be the central relation, and its primary key is shared among itself and the subclass relations. The subclass will only have its own primary key as an attribute. The subclasses will each have its own relation which has the primary key of the superclass along with all its own attributes. One advantage of this method is that it is the simplest way to convert superclass and subclasses into relations.

The second method is used to create only subclasses, and it only involves creating relations for each subclass with a shared primary key.

The third method is used to create one relation with an attribute which indicates which type it is. A relation which contains the attributes of all the subclasses and the superclass is created using this method along with an attribute appended to the end of the attribute list which indicates its type. This way, depending on what type is specified, only the relevant attributes would have values while the irrelevant attributes would be null. The obvious drawback of this method is that it will only work with disjointed subclasses: the

relation can only be one type. There is also the issue of having too many columns which will only contain null values if the type does not deem these attributes relevant.

The fourth method is used to create a relation with multiple types. The procedure is like the third method, but it can hold multiple type attributes. However, the problem of having too many attributes holding null values is magnified.

Lastly, when representing union types, each relation in the union simply shares a key attribute. This key attribute can be derived from the diagram, or a new key can be made in the absence of a sensible shared key attribute among the union.

## **2.2.4 – Database Constraints**

Rules called Constraints are used to maintain the consistency and integrity of the data in a database by restricting the type of data that can be entered. They apply to any scope, whether that scope be limited to a column, a table, or a model.

### **Entity Constraints**

Entity constraints are constraints where an entity in a database must have a primary key that cannot be null. The primary key is used to uniquely identify an entity which is why the primary key cannot be left null, however other attributes of the entity can be left null without causing any problems.

### **Primary Key and Unique Key Constraints**

Primary keys are used to uniquely distinguish a tuple in a relation. A candidate key is a set of one or more attributes that is unique for each tuple. A candidate key also cannot be set to null. A relation may have multiple candidate keys, but only one key can be designated as the primary key for the relation. The DBMS will verify during insert and update operations that the data does not violate the key constraint.

### **Referential Constraints**

Referential constraints are also known as foreign key constraints and they are constraints that state that if an attribute in one relation references a value in a different relation then that value that was referenced has to exist. During the insert and update operations the DBMS will verify that the foreign key being referenced exists. If it does not exist, then the query will fail. During the delete operation the DBMS will check to see if the primary key is being used as a foreign key. If it is then it will reject the deletion stating that there is a foreign key violation.

### **Check Constraints**

Check constraints are used to ensure that incorrect data is not entered into the database by specifying a range of acceptable values. For example, when inputting a date of birth, it does not make sense for someone to put in a date that is in the future since they would not be born yet. Therefore, it would make sense

to put a constraint that restricts putting in a value that is past today's date. The DBMS will enforce this check constraint during insert and update operations.

## 3 – Results of ER to Relational Conversion

Following the conceptual-to-logical conversion process outlined in section 2.2, we have now established a formal relation schema for our database. It was a relatively simple task to convert the strong entities into relations because they do not require any additional tuples other than the ones directly associated with the entity. It took more time to convert the rest of the database because every entity (both weak and strong) was involved in at least two relationships. Thus, we had to be careful to include the necessary tuples from every possible entity source when converting the relationships. With the schema complete, this section will display the full set of relations as well as sample data to populate the database and be used for query testing later.

### 3.1 – Relation Schema

#### Users

account\_id

Domain: positive auto incrementing integer, 1 – 999999999, cannot be NULL

username

Domain: Varchar 255, cannot be NULL

password

Domain: Varchar 255, cannot be NULL

email

Domain: Varchar 255, cannot be NULL

birth\_date

Domain: valid date

first\_name

Domain: Varchar[255]

last\_name

Domain: Varchar[255]

Constraints:

Primary Key: account\_id

Semantic: date of birth\_date must be greater than or equal 18 years to the date of initialization

## Film

film\_id

Domain: positive auto incrementing integer, 1 – 99999999, cannot be NULL

film\_name

Domain: Varchar[255], cannot be NULL

release\_date

Domain: valid date, cannot be NULL

duration

Domain: positive integer, 1-1439, cannot be NULL

synopsis

Domain: Varchar[2000]

rating

Domain: Varchar[5], cannot be NULL

Image

Domain: Varchar[255]

Constraints:

Primary Key: film\_id

## Genre

genre\_id

Domain: positive auto incrementing integer, 1 – 99999999, cannot be NULL

name

Domain: Varchar[255] cannot be NULL

Constraints:

Primary Key: genre\_id

## Film\_Genre

film\_id

Domain: positive integer, 1 – 99999999, cannot be NULL

genre\_id

Domain: positive integer, 1 – 99999999, cannot be NULL

Constraints:

Foreign Keys: film\_id, genre\_id

## Review

review\_id

Domain: positive auto incrementing integer, 1 – 999999999, cannot be NULL

account\_id

Domain: positive integer, 1 – 999999999, cannot be NULL

film\_id

Domain: positive integer, 1 - 999999999, cannot be NULL

title

Domain: Varchar[255], cannot be NULL

score

Domain: positive integer, 1-10, cannot be NULL

description

Domain: Varchar[2047]

review\_date

Domain: valid date

Constraints:

Primary Key: review\_id

Foreign Keys: account\_id, film\_id

## List

list\_id

Domain: positive auto incrementing integer, 1 – 999999999, cannot be NULL

account\_id

Domain: positive auto incrementing integer, 1 – 99999, cannot be NULL

list\_title

Domain: Varchar[255], cannot be NULL

Constraints:

Primary Key: list\_id



Foreign Keys: account\_id, film\_id

## List\_Items

list\_id

Domain: positive integer, 1 – 999999999, cannot be NULL

film\_id

Domain: positive integer, 1 – 999999999, cannot be NULL

Constraints:

Foreign Keys: list\_id, film\_id

## 3.2 – Sample Data

### User

account_id	username	password	email	birth_date	first_name	last_name
1	flivsey0	yCcVAbQBax9	flivsey0@blog.com	2002-10-08	Fleming	Livsey
2	mwoolham1	V5kieW	mwoolham1@va.gov	1990-12-26	Mariska	Woolham
3	jgerling2	5bCe8p9	jgerling2@over-blog.com	1998-01-15	Jannelle	Gerling
4	hateggart3	Idl5SE1HEiM6	hateggart3@reddit.com	1996-05-06	Honoria	Ateggart
5	cscough4	fdUgUAqam5j	cscough4@tiny.cc	1985-06-29	Chen	Scough
6	mcrunkhorn5	2qwmaJWMgeg	mcrunkhorn5@soup.io	1984-06-14	Michael	Crunkhorn
7	zquixley6	HIZHNXm1R	zquixley6@geocities.com	1987-02-25	Zandra	Quixley
8	glerway7	HTTDyzDapSKf	glerway7@dropbox.com	1992-03-13	Gerhardt	Lerway

9	awarland8	Wt9tsOBRqT	awarland8@illinois.edu	1997-02-22	Annette	Warland
10	emoulster9	nJ9yXcvR5uRm	emoulster9@facebook.com	1995-02-11	Ezra	Moulster
11	erosena	MViXXMI9	erosena@omniture.com	1998-06-19	Elsi	Rosen
12	egrindlayb	6OakxKc	egrindlayb@admin.ch	2000-01-26	Enrika	Grindlay
13	bspinellac	AS2Z6krLA2M	bspinellac@kickstarter.com	1986-02-20	Bernie	Spinella
14	amurnamed	6ahKGkfYb	amurnamed@elpais.com	1983-07-08	Alane	Murname
15	mbloxame	4i0GpGIYDIs	mbloxame@reverbnation.com	1983-12-10	Marcello	Bloxam
16	rmilbornf	1BYN0OPky	rmilbornf@mozilla.org	2001-07-12	Rowland	Milborn
17	tmcshaneg	WHxScNMaq5	tmcshaneg@cnbc.com	1995-04-06	Tyrone	McShane
18	tbarnwillh	MzGAN7nS0Eol	tbarnwillh@oracle.com	1985-03-19	Tracie	Barnwill
19	larnolli	GRMJml	larnolli@discuz.net	2002-03-01	Lion	Arnoll
20	cvaleroj	mkdQan6	cvaleroj@reddit.com	1981-07-20	Corinne	Valero

## Film

\* The values for each instance of the 'synopsis' attribute have been omitted from the table below to prevent any formatting issues. Such issues could occur because this attribute is meant to store the brief summary of the film's core concept and plot points which could result in several paragraphs of text. Considering the 'synopsis' attribute is not specifically mentioned in the queries listed in the next section and the fact that it is not a key, the contents for each instance will not be explicitly shown below. However, we still maintain the actual data in our mock data excel files separate from this report.

film_id	film_name	release_date	duration	rating	image	synopsis
---------	-----------	--------------	----------	--------	-------	----------

1	Faces of Schlock	2008-10-05	176	R	nisl_nunc.jpeg	*
2	Good Year, A	2019-08-21	90	NC-17	odio.tiff	*
3	Long Way Round	2013-01-10	179	G	ridiculus_mus_vivamus.jpeg	*
4	Cool It	2015-12-16	128	PG-13	sapien.tiff	*
5	Fuzz	2019-01-23	127	PG	vitae_quam_suspendisse.tiff	*
6	Princess Protection Program	2012-02-27	147	G	vel_ipsum_praesent.gif	*
7	Men with Guns	2015-01-12	103	PG-13	mi_sit.tiff	*
8	Left Behind II: Tribulation Force	2010-12-17	100	PG	ut_at_dolor.tiff	*
9	Armadillo	2011-04-23	150	PG	pede.tiff	*
10	Monster	2011-10-28	155	PG-13	consequat_ut.jpeg	*
11	Singles	2013-08-30	115	G	porta.gif	*
12	Adam Resurrected	2013-06-23	138	PG-13	eget_nunc.tiff	*
13	Innocent Blood	2018-07-30	153	R	erat_volutpat_in.jpeg	*
14	Doomsday Book	2018-10-31	169	PG	libero_rutrum.jpeg	*
15	Beautiful Girl	2019-08-13	179	G	odio_consequat_varius.jpeg	*
16	Italian for Beginners	2017-07-18	117	G	non_ligula_pellentesque.jpeg	*

17	Life 2.0	2018-12-02	136	PG-13	aliquam.png	*
18	King - Jari Litmanen, The (Kuningas Litmanen)	2011-07-02	142	PG-13	id_massa_id.jpeg	*
19	Homem Que Desafiou o Diabo, O	2017-04-18	138	PG-13	vel_nisl.jpeg	*
20	Karan Arjun	2009-11-12	154	R	proin_risus_praesent.gif	*

## Genre

genre_id	name
1	Action
2	Comedy
3	Crime
4	Drama
5	Fantasy
6	Historical
7	Horror
8	Romance
9	Science

10	Western
11	Adventure
12	War
13	Musical
14	Sports
15	Documentary
16	Children

## Film\_Genre

film_id	genre_id
1	2
1	4
1	1
2	8
2	4
3	11
3	1

3	5
4	15
4	6
5	4
5	9
5	10
6	16
6	7
7	1
7	8
8	3
8	11
9	15
9	9
10	7
10	5
11	2

11	8
12	7
13	7
13	1
14	12
14	9
15	4
15	13
16	2
16	10
17	15
17	6
18	10
18	5
19	11
19	14
20	12

20	13
20	14

## Review

review_id	account_id	film_id	title	score	description	review_date
1	14	11	Public-key	10	justo in blandit ultrices enim lorem ipsum dolor	2019-11-24
2	18	18	implementation	9	odio cras mi pede malesuada in	2020-07-05
3	4	3	Quality-focused	6	aenean fermentum donec ut mauris	2019-04-17
4	8	15	artificial intelligence	3	nibh fusce lacus purus aliquet	2019-09-16
5	3	20	Grass-roots	2	sodales sed tincidunt eu felis	2018-11-13
6	19	15	Programmable	8	suspendisse ornare consequat lectus in est risus auctor sed	2020-04-02
7	4	5	explicit	1	sociis natoque penatibus et magnis dis parturient montes	2019-02-13
8	17	19	User-centric	3	quis libero nullam sit amet turpis elementum	2020-01-02
9	5	15	secured line	10	accumsan tellus nisi	2020-06-24
10	18	16	time-frame	4	amet diam in magna bibendum imperdiet	2019-09-23
11	13	14	secondary	3	platea dictumst aliquam augue quam sollicitudin vitae consectetur eget	2020-05-18
12	17	13	Business-focused	7	ipsum primis in faucibus orci luctus et ultrices	2019-04-20



13	9	10	Multi-channelled	6	duis bibendum felis	2018-10-31
14	14	8	Robust	10	ultrices erat tortor sollicitudin mi sit amet	2019-10-01
15	7	11	tertiary	2	eros suspendisse accumsan tortor quis turpis sed ante vivamus	2019-09-12
16	12	3	asymmetric	4	curae donec pharetra magna vestibulum aliquet ultrices erat tortor	2020-09-11
17	10	18	Quality-focused	3	at nibh in hac habitasse	2019-10-31
18	20	16	paradigm	5	curae mauris viverra diam vitae quam suspendisse	2018-12-09
19	7	9	Fully-configurable	8	et ultrices posuere cubilia curae	2019-05-17
20	20	14	discrete	7	ultrices phasellus id sapien in sapien iaculis congue vivamus metus	2019-01-12
21	4	2	adapter	4	lacus morbi quis tortor id nulla ultrices	2020-01-16
22	4	10	mission-critical	9	nunc nisl duis bibendum felis sed interdum venenatis turpis enim	2019-04-13
23	1	1	solution-oriented	10	placerat praesent blandit nam nulla integer pede justo	2020-02-24
24	6	2	secured line	9	erat quisque erat eros viverra eget congue eget semper rutrum	2020-06-06
25	5	10	motivating	9	vitae nisi nam ultrices libero non	2020-01-16
26	18	17	Robust	9	sed tristique in tempus sit amet sem fusce consequat nulla	2019-09-12
27	4	7	matrices	1	varius ut blandit non interdum in ante vestibulum ante	2019-08-09
28	1	14	grid-enabled	3	nascetur ridiculus mus vivamus	2020-09-30
29	11	9	context-sensitive	9	ante nulla justo aliquam quis turpis eget elit sodales	2018-10-31

30	19	8	parallelism	1	nec sem duis aliquam convallis nunc	2019-07-08
31	19	1	Public-key	6	ut blandit non interdum in ante vestibulum ante ipsum	2019-04-29
32	7	3	Down-sized	10	mattis nibh ligula	2019-06-27
33	19	15	Advanced	8	neque aenean auctor gravida sem praesent id	2020-09-05
34	7	1	Vision-oriented	10	natoque penatibus et magnis dis parturient montes nascetur ridiculus mus	2019-09-19
35	6	19	alliance	4	vestibulum quam sapien varius	2019-09-16
36	11	20	attitude-oriented	6	massa volutpat convallis morbi odio odio elementum	2019-07-08
37	4	6	Virtual	3	lectus pellentesque at nulla suspendisse potenti cras in purus eu	2018-11-16
38	15	16	Enterprise-wide	9	id massa id nisl venenatis	2020-05-03
39	15	8	multi-tasking	9	lectus pellentesque at	2020-08-20
40	15	14	Ergonomic	2	dapibus augue vel accumsan tellus nisi	2020-01-01
41	1	6	Profit-focused	7	consequat nulla nisl nunc nisl duis bibendum felis sed	2019-07-30
42	3	16	Phased	10	a feugiat et	2020-02-14
43	14	20	directional	2	imperdiet nullam orci pede venenatis non sodales	2019-12-22
44	6	7	knowledge user	6	bibendum felis sed interdum venenatis turpis enim	2020-01-29
45	16	19	User-centric	4	ultrices posuere cubilia curae duis faucibus accumsan odio curabitur convallis	2018-10-27

46	10	9	collaboration	2	vivamus in felis eu sapien cursus vestibulum proin	2019-02-18
47	11	16	policy	7	feugiat non pretium quis lectus suspendisse potenti	2019-08-12
48	1	1	logistical	10	mi integer ac neque dui bibendum morbi non quam nec	2019-04-25
49	5	6	secured line	6	condimentum curabitur in libero ut massa	2019-08-22
50	14	5	Multi-tiered	6	ante ipsum primis in	2019-06-26
51	16	3	object-oriented	5	orci eget orci	2020-08-12
52	19	8	Streamlined	4	erat nulla tempus vivamus in felis	2019-08-20
53	11	18	support	1	sed nisl nunc rhoncus dui vel sem sed sagittis	2019-06-05
54	7	18	eco-centric	3	ut nulla sed accumsan felis ut at	2020-02-29
55	15	20	Phased	2	dis parturient montes nascetur ridiculus mus vivamus vestibulum	2020-04-20
56	5	3	encoding	1	aenean sit amet justo	2019-02-22
57	17	15	benchmark	10	sollicitudin ut suscipit a feugiat et eros vestibulum ac	2019-09-09
58	4	8	Realigned	9	vulputate luctus cum sociis natoque penatibus et magnis dis parturient	2020-09-03
59	18	8	Business-focused	5	cum sociis natoque penatibus et magnis	2020-06-28
60	13	18	executive	9	tincidunt in leo maecenas pulvinar lobortis	2019-12-22
61	7	3	moratorium	10	non ligula pellentesque ultrices phasellus id sapien in sapien	2020-10-06
62	14	4	Secured	4	dapibus nulla suscipit ligula in lacus curabitur	2020-09-12

63	15	10	solution	9	nulla sed vel enim sit amet nunc viverra	2020-07-14
64	10	18	intranet	9	odio porttitor id consequat in consequat ut nulla	2019-02-08
65	15	3	service-desk	3	maecenas tincidunt lacus at velit vivamus vel	2020-01-07
66	13	6	actuating	7	vivamus in felis eu sapien cursus vestibulum proin eu	2020-04-17
67	5	12	clear-thinking	6	aliquam erat volutpat	2020-05-12
68	4	18	installation	9	id nulla ultrices aliquet maecenas leo	2020-09-16
69	14	3	clear-thinking	7	id sapien in sapien iaculis congue vivamus	2020-07-11
70	3	18	4th generation	7	mauris ullamcorper purus sit amet nulla quisque	2020-10-15
71	1	2	Organic	3	nec nisi vulputate nonummy maecenas tincidunt lacus at velit vivamus	2019-02-24
72	16	6	Assimilated	5	interdum mauris ullamcorper	2018-10-31
73	3	2	Seamless	7	nunc viverra dapibus nulla suscipit ligula in lacus	2018-10-28
74	8	2	standardization	3	ac neque duis bibendum morbi non quam nec	2019-11-28
75	16	18	global	4	consequat morbi a ipsum integer a	2019-10-30
76	11	16	Focused	4	eleifend pede libero quis orci nullam molestie	2020-10-13
77	17	19	disintermediate	9	fusce congue diam id ornare imperdiet	2020-08-16
78	9	20	Fully-configurable	1	imperdiet nullam orci pede venenatis non sodales sed tincidunt	2019-12-31
79	9	3	dedicated	9	sem fusce consequat nulla nisl nunc nisl duis	2020-04-26

80	16	1	24/7	2	primis in faucibus orci luctus et	2020-06-09
81	9	16	Re-contextualized	6	id ligula suspendisse ornare consequat lectus in est risus	2019-06-29
82	14	13	alliance	2	vestibulum vestibulum ante	2020-02-28
83	10	11	Up-sized	2	nunc vestibulum ante ipsum primis in faucibus	2019-09-15
84	5	7	Vision-oriented	6	diam nam tristique tortor eu pede	2020-08-09
85	20	13	Multi-layered	8	augue vel accumsan tellus nisi eu orci mauris lacinia	2020-08-16
86	9	13	Automated	9	turpis nec euismod	2018-10-21
87	10	7	Horizontal	9	et ultrices posuere cubilia curae duis faucibus accumsan odio	2019-04-08
88	16	2	workforce	10	praesent lectus vestibulum quam sapien varius ut blandit	2019-11-26
89	20	16	definition	6	amet turpis elementum ligula	2019-03-10
90	16	7	Programmable	9	ipsum dolor sit amet consectetur adipiscing elit proin risus	2020-09-25
91	6	8	algorithm	3	turpis eget elit sodales scelerisque mauris sit	2019-06-27
92	6	19	Proactive	6	nullam molestie nibh in lectus pellentesque at nulla suspendisse potenti	2019-08-09
93	18	16	exuding	4	lobortis est phasellus sit amet erat nulla tempus vivamus	2020-08-04
94	17	13	definition	1	blandit nam nulla integer pede justo lacinia eget tincidunt eget	2019-12-23
95	17	5	project	1	justo sit amet sapien dignissim vestibulum vestibulum ante ipsum	2019-08-22
96	7	1	Intuitive	5	mi integer ac	2020-04-04

97	15	14	initiative	2	et magnis dis parturient montes nascetur	2019-06-20
98	11	10	needs-based	2	tellus nulla ut erat id mauris vulputate elementum	2019-03-31
99	9	12	web-enabled	4	porta volutpat quam pede lobortis ligula	2019-08-26
100	5	16	instruction set	2	dui nec nisi volutpat eleifend	2020-03-19

## List

list_id	account_id	list_title
1	16	With the homies
2	9	My Favorites
3	4	For Later
4	19	best action flicks
5	13	Summer 2k19
6	20	Dead Franchises
7	15	Stupid-High Budgets
8	6	Dave's Faves
9	16	Must-Watch
10	16	Must-Avoid
11	5	listoffilms.org

12	11	date night
13	1	disguising your 150 TB Homework Folder
14	9	after anime hours
15	7	On the way to iHop
16	4	Fortnite 2 releases on 12/31/2020
17	4	films to watch when your dog runs away

## List\_Items

list_id	film_id
1	1
1	7
1	10
1	11
1	9
1	2
2	9
2	13

2	5
2	8
2	11
2	19
3	13
3	14
3	4
3	17
4	7
4	10
4	3
4	9
4	14
5	2
5	11
5	4
5	14



5	3
6	8
6	7
6	12
6	15
6	6
7	9
7	17
7	18
7	10
7	3
7	4
7	12
7	20
8	4
8	17
8	18

8	1
9	2
9	3
9	11
9	7
9	14
9	13
9	10
9	16
9	1
10	6
10	9
10	15
10	12
10	20
10	17
10	8

10	5
11	2
11	3
11	1
11	11
11	20
11	12
12	1
12	2
12	20
12	5
12	16
12	4
13	3
13	13
13	17
13	9

13	6
13	15
13	19
14	4
14	5
14	12
14	7
14	18
14	13
15	4
15	11
15	12
15	17
16	12
16	3
16	4
16	17

17	5
17	15
17	7
17	18
17	3
17	2
17	6

## 4 – Sample Queries

Now that the relation schema has been created and a collection of sample data has been gathered, the functionality of the database can be demonstrated using queries. First, a set of queries will be introduced in plain English. Then these queries will be converted into relational algebra expressions. Finally, the queries will also be converted into relational calculus expressions before moving onto the next phase.

### 4.1 – Design of Queries

1. List all films released after the year 2009 that are not rated R or NC-17 and at most 150 minutes in length.
2. List the most popular film in terms of review score from films released from 2015 through 2019.
3. For every user, list their first name, last name, and all the lists they made.
4. Find the titles of film lists created by users who were born after 1995.
5. List which genre(s) categorizes the film with the highest score among all films.
6. List the films that have at least an 8-point rating and were released after 2010.
7. List the three films that have the highest ratings.
8. Find all the films that are either reviewed by or included in a list by a user whose last name is 'Warland' or 'Rosen'.
9. List users who have made a list containing all films in the database
10. List users who have made a list containing all films of the "Action" genre.

## 4.2 – Relational Algebra Expressions

List of Relations:

- Users(account\_id, username, password, email, birth\_date, first\_name, last\_name)
- Film(film\_id, film\_name, release\_date, duration, rating, image, synopsis)
- Genre(id, name)
- Film\_Genre(genre\_id, film\_id)
- Review(review\_id, account\_id, film\_id, title, score, description, review\_date)
- List(list\_id, account\_id, list\_title)
- List\_Items(list\_id, film\_id)

Answers to Queries:

1.  $R \leftarrow \sigma_{\text{duration} \leq 150 \ \&\& \ \text{release\_date} > 12/31/2009 \ \&\& \ \text{rating} \neq 'R' \ \&\& \ \text{rating} \neq 'NC-17'}(\text{Film})$
2.  $T1(\text{film\_id}, \text{best\_score}) \leftarrow \text{film\_id } G \max(\text{score}) (\text{Review})$   
 $F \leftarrow T1 \bowtie_{\text{film\_id} = \text{film\_id}} \text{Film}$   
 $RD \leftarrow \sigma_{\text{release\_date} \geq 2015, \text{release\_date} \leq 2019}(F)$   
 $R \leftarrow \pi_{\text{film\_id}, \text{film\_name}, \text{release\_date}, \text{duration}, \text{synopsis}, \text{rating}, \text{image}}(RD)$
3.  $R \leftarrow \pi_{\text{first\_name}, \text{last\_name}, \text{list\_id}, \text{list\_title}} (\text{User} * \text{List})$
4.  $BD \leftarrow \sigma_{\text{birth\_date} \geq 12/31/1995} (\text{Users})$
5.  $UF \leftarrow \text{List} \bowtie_{\text{account\_id} = \text{account\_id}} BD$   
 $R \leftarrow \pi_{\text{list\_title}}(U \bowtie_{\text{film\_id}} F \bowtie_{\text{score}} R) \leftarrow \text{film\_id } G \max(\text{score}) (\text{Review})$   
 $GR \leftarrow \text{Films} \bowtie_{\text{film\_id} = \text{film\_id}} FR$   
 $R \leftarrow \pi_{\text{genre\_id}, \text{name}} (\text{Genre} \bowtie_{\text{genre\_id} = \text{genre\_id}} (GR \bowtie_{\text{film\_id} = \text{film\_id}} \text{Film\_Genre}))$
6.  $RD \leftarrow \sigma_{\text{rating} \geq 8 \ \&\& \ \text{release\_date} > 12/31/2010} (\text{Film})$
7.  $FR(\text{film\_id}, \text{rating}) \leftarrow \text{film\_id } G \max(\text{rating}) (\text{Film})$   
 $FJ \leftarrow \text{Film} \bowtie_{\text{film\_id} = \text{film\_id}} FR$   
 $L1 \leftarrow \text{Film} - FJ$   
 $SR(\text{film\_id}, \text{rating}) \leftarrow \text{film\_id } G \max(\text{rating}) (L1)$   
 $SJ \leftarrow L1 \bowtie_{\text{film\_id} = \text{film\_id}} SR$   
 $L2 \leftarrow L1 - SJ$   
 $TR(\text{film\_id}, \text{rating}) \leftarrow \text{film\_id } G \max(\text{rating}) (L2)$

```

TJ <- L2 ⋈film_id = film_id TR
R <- πfilm_id, film_name, release_date, duration, synopsis, rating, image(TJ + SJ + FJ)
8. U <- σlast_name = 'Warland' || last_name = 'Rosen'(User)
RV <- Review ⋈account_id = account_id U
L <- List ⋈account_id = account_id U
F <- L ⋈film_id = film_id Film
RU <- RV * Film
R <- πfilm_id, film_name, release_date, duration, synopsis, rating, image(F + RU)
9. LF <- πlist_id, film_id(List ⋈film_id = film_id Film)
F <- πfilm_id(List)
R4 <- LF ÷ F
R3 <- R3 ⋈list_id = list_id List
R2 <- User ⋈account_id = account_id R3
R <- πaccount_id, username, password, email, birth_date, first_name, last_name(R2)
10. LF <- πlist_id, film_id(List ⋈film_id = film_id (Film ⋈film_id = film_id Film_Genre))
F <- πfilm_id(List)
R4 <- LF ÷ F
R3 <- R3 ⋈list_id = list_id List
R2 <- User ⋈account_id = account_id R3
R <- πaccount_id, username, password, email, birth_date, first_name, last_name(R2)

```

### 4.3 – Relational Calculus Expressions

1.  $\{f \mid \text{Film}(f) \wedge f.\text{duration} \geq 150 \wedge f.\text{release\_date} \geq 12/31/2009 \wedge f.\text{rating} \neq 'R' \wedge f.\text{rating} \neq 'NC-17'\}$
2.  $\{f \mid \text{Film}(f) \wedge \text{Review}(r) \wedge \neg \exists m (\text{Review}(m) \wedge m.\text{score} > r.\text{score} \wedge f.\text{film\_id} = r.\text{film\_id})\}$
3.  $\{u.\text{first\_name} \wedge u.\text{last\_name} \wedge t \mid \text{User}(u) \wedge \exists t (\text{List}(t) \wedge t.\text{account\_id} = u.\text{account\_id})\}$
4.  $\{t.\text{list\_title} \mid \text{List}(t) \wedge \exists u (\text{User}(u) \wedge u.\text{account\_id} = t.\text{account\_id} \wedge u.\text{age} \geq 18 \wedge u.\text{age} \leq 25)\}$
5.  $\{g \mid \text{Genre}(g) \wedge \text{Film}(f) \wedge \text{Genre\_Film}(n) \wedge \neg \exists m (\text{Film}(m) \wedge m.\text{rating} > f.\text{rating} \wedge n.\text{film\_id} = f.\text{film\_id} \wedge g.\text{genre\_id} = n.\text{genre\_id})\}$

6.  $\{f \mid \text{Film}(f) \wedge f.\text{rating} > 8 \wedge f.\text{release\_date} \geq 1/1/2010\}$
7.  $\{f, g, h \mid \text{Film}(f) \wedge \neg \exists m (\text{Film}(m) \wedge m.\text{rating} > f.\text{rating})$   
 $\wedge \text{Film}(g) \wedge g.\text{film\_id} \neq f.\text{film\_id} \wedge \neg \exists n (\text{Film}(n) \wedge n.\text{rating} > g.\text{rating}$   
 $\wedge n.\text{film\_id} \neq f.\text{film\_id})$   
 $\wedge \text{Film}(h) \wedge h.\text{film\_id} \neq f.\text{film\_id} \wedge h.\text{film\_id} \neq g.\text{film\_id}$   
 $\wedge \neg \exists o (\text{Film}(o) \wedge o.\text{rating} > h.\text{rating} \wedge o.\text{film\_id} \neq f.\text{film\_id}$   
 $\wedge o.\text{film\_id} \neq h.\text{film\_id})\}$
8.  $\{f \mid \text{Film}(f) \wedge \exists u (\text{User}(u) \wedge (u.\text{last\_name} = \text{'Smith'} \vee u.\text{last\_name} = \text{'Williams'})$   
 $\wedge \exists r (\text{Review}(r) \wedge r.\text{film\_id} = f.\text{film\_id}$   
 $\wedge \exists t (\text{List}(t) \wedge t.\text{account\_id} = u.\text{account\_id}))\}$
9.  $\{u \mid \text{User}(u) \wedge \forall f (\neg \text{Film}(f) \vee \exists t (\text{List}(t) \wedge t.\text{film\_id} = f.\text{film\_id}$   
 $\wedge t.\text{account\_id} = u.\text{account\_id}))\}$
10.  $\{u \mid \text{User}(u) \wedge \forall f (\neg \text{Film}(f) \vee \neg (f.\text{genre} = \text{"Action"})$   
 $\vee \exists t (\text{List}(t) \wedge t.\text{film\_id} = f.\text{film\_id} \wedge t.\text{account\_id} = u.\text{account\_id}))\}$



# Phase 3: Relational Normalization and Physical Implementation

## 5 – Normalization of the Relations

### 5.1 – Normalization Process

Database normalization is a formal process that consists of analyzing relation schemas' primary keys and the functional dependencies amongst the attributes within the schema. The goal of this process is to minimize the redundancy of the database as much as possible while also minimizing any anomalies that can occur with insertion, deletion, and update procedures. Benchmarks are performed on relation schemas to test what normal form they satisfy as well as to identify where a schema can be modified to better meet the criteria of the desired normal form. Four examples of the classifications used to represent the normalization of a database are the first, second, third, and Boyce-Codd normal forms.

#### First Normal Form

The first normal form (1NF) is used to identify relations that have only simple and indivisible attributes amongst its entities. In other words, the attributes should not be multivalued, and they should also not have nested relations. If a relation has multivalued attributes, there are several ways to modify it to achieve 1NF. One technique is to decompose it into separate 1NF relations while giving the newly created relations the primary key of the original. Another technique is to create additional tuples in the relation for each of the different attribute values. In this instance, the violating attribute becomes a part of the primary key after the additional tuples are created. The downside is that redundancy becomes an issue since the same row of data (aside from the one differing attribute value) is being stored several times. The third method is to create additional columns in the relation for each possible atomic value. It is a relatively straightforward solution if the maximum quantity of values is known. However, the downsides include introducing many NULL values to the relation as well as complications with ordering and querying.

#### Second Normal Form

The second normal form (2NF) is used to identify relations that have full functional dependency between every one of its nonprime attributes and the primary key. Full functional dependency  $X \twoheadrightarrow Y$  can be established when the removal of any attribute A from X causes the dependency to cease holding. If such a removal occurs and the dependency still holds, then there is a partial dependency. The existence of any partial dependencies within a relation means it is not in 2NF. Also, the primary key must be composed of multiple attributes for any 2NF testing to be done. If a relation has a single-attribute primary key, then it is already in at least 2NF. However, to normalize a

multi-attribute primary key relation that is not in 2NF, decomposition will be the solution. New relations for each of the keys associated with the partial dependencies and their dependent attributes will be created while also maintaining a relation with the original primary key and its full functionally dependent attributes.

### **Third Normal Form**

The third normal form (3NF) is used to identify relations that both satisfy 2NF and that have no nonprime attributes that are transitively dependent on its primary key.

Transitive dependency occurs when there is a set of attributes Z in the relation (where Z is neither a candidate key nor a subset of any key) and there are holding dependencies  $X \rightarrow Z$  and  $Z \rightarrow Y$ . Therefore, testing for 3NF involves studying the relation to determine if any non-key attributes are functionally determined by another non-key attribute. If such is the case, then there is a transitive dependency, and the relation is not in 3NF.

To normalize a relation that is not in 3NF, the relation can be decomposed to set up new relations that will include the non-key attributes that were involved in the transitive dependencies at fault. Once this is accomplished, the schema should be in 3NF by having every nonprime attribute have full functional dependency on every key while also having no transitive dependency on these keys.

### **Boyce-Codd Normal Form**

The Boyce-Codd normal form (BCNF) can be considered an alternative form of 3NF such that if a relation is in BCNF, then it is also in 3NF. However, the opposite is not always true because of the stricter criteria of BCNF that is intended to deal with potential anomalies that cannot be resolved solely with 3NF. To satisfy BCNF, a relation must already be in 3NF and have true that for any dependency  $A \rightarrow B$ , A cannot be a nonprime attribute if B is a prime attribute. Therefore, along with ensuring that such a statement holds true for all dependencies, any relation in BCNF must also satisfy the previous normal forms by having only atomic attributes, no partial dependencies, and no transitive dependencies. Once again, to normalize a relation to satisfy BCNF, the relation needs to be decomposed into separate tables. After the violations are dealt with, the decomposition can cease, and the relation can be BCNF.

### **Insertion, Deletion, and Update Anomalies**

Several kinds of problems can arise in a database if the relational model is not normalized. These problems, which are otherwise known as data anomalies, can occur during insertion, deletion, and updating of the data. Insertion anomalies can happen when inserting data cannot successfully happen because other necessary data does not exist. For example, a database might have an insertion anomaly where a purchase cannot be recorded because it must be applied to an existing customer, but an instance of a customer cannot be created until they have recorded a purchase. Such a case cannot be solved with any combination of data insertions and must be fixed with changes to the relation schema. Deletion anomalies occur when deleting a piece of unwanted data causes other data to be deleted as well. An example of such a case is when deleting a review of a film causes the film itself to be removed from the database. Update anomalies happen when any updates are made to the database that involve redundantly stored data. Because it exists in several places, all instances of the data must be correctly updated to reflect the desired changes.

Proper normalization of a relation schema can prevent data anomalies from happening later during the implementation of the database. The normalization techniques for 2NF and 3NF are centered around breaking down relations into smaller ones to lower the amount of partial and transitive dependencies in the schema. With this tactic, the insertion anomaly example from earlier can be remedied by having separate tables for purchases and customers while also possibly adding a 'transactions' relation to link the two. Also, deletion anomalies can be minimized since changes made to the 'purchases' table will not affect the status of any data in the 'customers' table. Another benefit to normalization is that keeping the data redundancy as low as possible allows for lower risks of update anomalies since less overall changes will have to be made to the database with a lower number of redundant targets.

## **5.2 – Application to the Relational Model**

Based on your relational model, edit (or create) one relation that is not normalized, then list any anomalies that could be possible in this relation.

Since none of the relations in the relation schema have multiple records, our relations are in first normal form. Each attribute for the Reviews relation holds only atomic values which cannot be segmented into.

These relations also follow the second normal form paradigm since most of them have nonprime attributes which have full functional dependency with a single primary key. The List\_Items and Film\_Genre relations are the only relations where there exists two keys, but there are no nonprime attributes which are repeated with more than a single key pair.

In terms of following the third normal form process or the Boyce-Codd normal form process, no changes were made to any of the relations to be in third normal form nor to be in Boyce-Codd normal form.

# **6 – Database Implementation**

## **6.1 – Background Information**

The main purpose of using relational database management software (RDBMS) is to implement a database more easily without having to handle the lower-level considerations which would need to be managed more strictly. One specific way of handling a database is to use a client-server RDBMS to handle the database remotely on a server where clients would need permission to access the database.

Advantages to this method of implementing a database include better data integrity, easier data maintenance, and better security. Security is better in a client-server paradigm since several methods of authentication such as certificates, passwords, and permissions all help control who can access the database directly. Data maintenance is easier as well since data on a server can be shared with multiple database

administrators, and the data can be backed up on multiple local machines if multiple administrators have that responsibility. Easier data maintenance naturally leads to better data integrity. The only significant disadvantage of using a client-server database is that it is more complicated to implement than a database stored on a single local machine which can only be accessed if a network connects other machines to it directly.

MySQL was used as our RDBMS since it conforms to SQL standards. It is also supported by our ideal hosting service and has extensive official and third-party documentation on its use and applications.

## 6.2 – Schema and Hosting

### MySQL as a RDBMS

MySQL is an open-source RDBMS that conforms to the SQL standard and can run on Windows, macOS, Linux, Solaris, and BSD. It supports all the data types that we plan to incorporate into our database while also being very similar to the SQL command syntax of PostgreSQL (identical in many cases) which we are also familiar with. However, while it is slightly more restrictive to implement various kinds of check constraints in MySQL than in PostgreSQL, we have altered our database entities enough to implement our entire schema without having to use any check constraints within the the statements that create our tables. MySQL is also supported by our choice of hosting platform.

### Hosting

We have decided to implement and host our database schema through the cloud services of DigitalOcean. DigitalOcean supports MySQL 8 hosting, uses an easy-to-navigate website-based interface, and allows for the creation of Docker containers which will be beneficial for later phases. Despite its costs, albeit relatively low, DigitalOcean was a service that our group felt comfortable utilizing to host our project. Connecting to the database is done using a SSH-capable client to establish an SSH connection to our droplet on DigitalOcean. After the connection is made, the command 'sudo mysql' is used to connect to the MySQL server on our droplet. With a RDBMS and hosting plan in place, the database can finally be created and populated.

### MySQL Schema

#### Users

```
CREATE TABLE Users (  
    account_id INT unsigned NOT NULL AUTO_INCREMENT,  
    username VARCHAR(255) NOT NULL,  
    password VARCHAR(255) NOT NULL,  
    email VARCHAR(255) NOT NULL,  
    birth_date DATE NOT NULL CHECK (TIMESTAMPDIFF(YEAR, birth_date,  
        SYSDATE()) >= 18),  
    first_name VARCHAR(255),
```

```

        last_name VARCHAR(255),
        PRIMARY KEY (account_id)
    );

```

```

mysql> desc Users;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| account_id | int unsigned  | NO   | PRI | NULL    | auto_increment |
| username   | varchar(255)  | NO   |     | NULL    |                |
| password   | varchar(255)  | NO   |     | NULL    |                |
| email      | varchar(255)  | NO   |     | NULL    |                |
| birth_date | date          | NO   |     | NULL    |                |
| first_name | varchar(255)  | YES  |     | NULL    |                |
| last_name  | varchar(255)  | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql>

```

## Film

```

CREATE TABLE Film (
    film_id INT unsigned NOT NULL AUTO_INCREMENT,
    film_name VARCHAR(255) NOT NULL,
    release_date DATE NOT NULL,
    duration SMALLINT NOT NULL,
    rating VARCHAR(5) NOT NULL,
    image VARCHAR(255) NOT NULL,
    synopsis VARCHAR(1200),
    PRIMARY KEY (film_id)
);

```

```

mysql> desc Film;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| film_id    | int unsigned  | NO   | PRI | NULL    | auto_increment |
| film_name  | varchar(255)  | NO   |     | NULL    |                |
| release_date | date          | NO   |     | NULL    |                |
| duration   | smallint      | NO   |     | NULL    |                |
| rating     | varchar(5)    | NO   |     | NULL    |                |
| image      | varchar(255)  | NO   |     | NULL    |                |
| synopsis   | varchar(2000) | NO   |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql>

```

## Genre

```

CREATE TABLE Genre (
    genre_id SMALLINT unsigned NOT NULL AUTO_INCREMENT,
    name VARCHAR(20),

```

```

PRIMARY KEY (genre_id)
);

```

```

mysql> desc Genre;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| genre_id | smallint unsigned | NO   | PRI | NULL    | auto_increment |
| name     | varchar(20)      | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)

mysql>

```

## Film\_Genre

```

CREATE TABLE Film_Genre (
    film_id INT unsigned NOT NULL,
    genre_id SMALLINT unsigned NOT NULL,
    FOREIGN KEY(film_id)
        REFERENCES Film(film_id),
    FOREIGN KEY(genre_id)
        REFERENCES Genre(Genre_id)
);

```

```

mysql> desc Film_Genre;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| film_id | int unsigned   | NO   | MUL | NULL    |                |
| genre_id | smallint unsigned | NO   | MUL | NULL    |                |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)

mysql>

```

## Review

```

CREATE TABLE Review (
    review_id INT unsigned NOT NULL AUTO_INCREMENT,
    account_id INT unsigned NOT NULL,
    film_id INT unsigned NOT NULL,
    title VARCHAR(255) NOT NULL,
    score SMALLINT NOT NULL CHECK (score >= 1 AND score <= 10),
    description VARCHAR(2047),
    review_date DATE NOT NULL,
    PRIMARY KEY (review_id),

```

```

FOREIGN KEY(account_id)
    REFERENCES Users(account_id),
FOREIGN KEY(film_id)
    REFERENCES Film(film_id)
);

```

```

mysql> desc Review;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| review_id | int unsigned | NO   | PRI | NULL    | auto_increment |
| account_id | int unsigned | NO   | MUL | NULL    |                 |
| film_id    | int unsigned | NO   | MUL | NULL    |                 |
| title      | varchar(255) | NO   |     | NULL    |                 |
| score      | smallint     | NO   |     | NULL    |                 |
| description | varchar(2047) | YES  |     | NULL    |                 |
| review_date | date         | NO   |     | NULL    |                 |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql> |

```

## List

```

CREATE TABLE List (
    list_id INT unsigned NOT NULL AUTO_INCREMENT,
    account_id INT unsigned NOT NULL,
    list_title VARCHAR(255) NOT NULL,
    PRIMARY KEY (list_id),
    FOREIGN KEY(account_id)
        REFERENCES Users(account_id)
);

```

```

mysql> desc List;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| list_id | int unsigned | NO   | PRI | NULL    | auto_increment |
| account_id | int unsigned | NO   | MUL | NULL    |                 |
| list_title | varchar(255) | NO   |     | NULL    |                 |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>

```

## List Items

```

CREATE TABLE List_Items (
    list_id INT unsigned NOT NULL,
    film_id INT unsigned NOT NULL,
    FOREIGN KEY(list_id)
        REFERENCES List(list_id),
);

```

```

FOREIGN KEY(film_id)
REFERENCES Film(film_id)
);

```

```

mysql> desc List_Items;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| list_id | int unsigned | NO | MUL | NULL | |
| film_id | int unsigned | NO | MUL | NULL | |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>

```

## Data Insertion

To populate our database, we will be utilizing the tool on [convertcsv.com](http://convertcsv.com) that converts .csv files to functional SQL commands. Since all of our data is already stored on Excel files, we save a relatively large amount of time by exporting them to .csv sheets and generating the many 'INSERT' SQL commands needed to populate the database as opposed to handwriting them. Once the insertion commands are generated, they can be saved as text files for later use.

All seven tables of our database schema were populated with data using this method. After the conversion tool generated the insertion commands, they were copied and pasted into our client when connected to the database. Below is an example of when the 'Users' table was populated with the data of twenty users.

## Users

```

filmreel=> INSERT INTO Users(account_id,username,password,email,birth_date,first_name,last_name) VALUES
filmreel-> (1,'flivsey0','yCcVAbQ8ax9','flivsey0@blog.com','10/8/2002','Fleming','Livsey')
filmreel-> (2,'mwoolham1','V5kieW','mwoolham1@va.gov','12/26/1990','Mariska','Woolham')
filmreel-> (3,'jgerling2','5bCe8p9','jgerling2@over-blog.com','1/15/1998','Jannelle','Gerling')
filmreel-> (4,'hateggart3','ld15SE1HEiM6','hateggart3@reddit.com','5/6/1996','Honorio','Ateggart')
filmreel-> (5,'cscough4','fdUgUAqam5j','cscough4@tiny.cc','6/29/1985','Chen','Scough')
filmreel-> (6,'mcrunkhorn5','2qwmajWmgeg','mcrunkhorn5@soup.io','6/14/1984','Michael','Crunkhorn')
filmreel-> (7,'zquixley6','HI2HNxm1R','zquixley6@geocities.com','2/25/1987','Zandra','Quixley')
filmreel-> (8,'glerway7','HTTDyzDapSKF','glerway7@dropbox.com','3/13/1992','Gerhardt','Lerway')
filmreel-> (9,'awarland8','Wt9tsOBRqT','awarland8@illinois.edu','2/22/1998','Annette','Warland')
filmreel-> (10,'emoulster9','nJ9yXcvR5uRm','emoulster9@facebook.com','2/11/1995','Ezra','Moulster')
filmreel-> (11,'erosena','MViXXM19','erosena@omniture.com','6/19/1998','Elsi','Rosen')
filmreel-> (12,'egrindlayb','6OakxKc','egrindlayb@admin.ch','1/26/2000','Enrika','Grindlay')
filmreel-> (13,'bspinellac','AS2Z6krLA2M','bspinellac@kickstarter.com','2/20/1986','Bernie','Spinella')
filmreel-> (14,'amurnamed','6ahKGkfYb','amurnamed@elpais.com','7/8/1983','Alane','Murname')
filmreel-> (15,'mbloxame','4i0GpGlyDls','mbloxame@reverbnation.com','12/10/1983','Marcello','Bloxam')
filmreel-> (16,'rmilbornf','1BYN0OPky','rmilbornf@mozilla.org','7/12/2001','Rowland','Milborn')
filmreel-> (17,'tmcshane','WHxScNMaq5','tmcshane@cnbc.com','4/6/1995','Tyrone','McShane')
filmreel-> (18,'tbarnwillh','MzGAN7nS0EOI','tbarnwillh@oracle.com','3/19/1985','Tracie','Barnwill')
filmreel-> (19,'larnolli','GRMJmI','larnolli@discuz.net','3/1/2002','Lion','Arnoll')
filmreel-> (20,'cvaleroj','mkdQan6','cvaleroj@reddit.com','7/20/1981','Corinne','Valero');
INSERT 0 20
filmreel=>

```

The 'INSERT' SQL command ran without error given that the resulting statement was a simple 'INSERT 0 20'. To verify the contents of 'Users' table, the SQL command 'SELECT \* FROM Users' can be executed as shown below.



```
filmreel-> select * from users;
```

	account_id	username	password	email	birth_date	first_name	last_name
	1	flivsey0	yCcVAbQBax9	flivsey0@blog.com	2002-10-08	Fleming	Livsey
	2	mwoolham1	V5kieW	mwoolham1@va.gov	1990-12-26	Mariska	Woolham
	3	jgerling2	5bCe8p9	jgerling2@over-blog.com	1998-01-15	Jannelle	Gerling
	4	hateggart3	ld15SE1HEiM6	hateggart3@reddit.com	1996-05-06	Honorina	Ateggart
	5	cscough4	fdUgUAqam5j	cscough4@tiny.cc	1985-06-29	Chen	Scough
	6	mcrunkhorn5	2qumaJWMgeg	mcrunkhorn5@soup.io	1984-06-14	Michael	Crunkhorn
	7	zquixley6	HIZHNXm1R	zquixley6@geocities.com	1987-02-25	Zandra	Quixley
	8	glerway7	HTTDyzDapSKf	glerway7@dropbox.com	1992-03-13	Gerhardt	Lerway
	9	awarland8	Wt9ts0BRqT	awarland8@illinois.edu	1998-02-22	Annette	Warland
	10	emoulster9	nJ9yXcvR5uRm	emoulster9@facebook.com	1995-02-11	Ezra	Moulster
	11	erosena	MVIXXM19	erosena@omniture.com	1998-06-19	Elsi	Rosen
	12	egrindlayb	60akxKc	egrindlayb@admin.ch	2000-01-26	Enrika	Grindlay
	13	bspinellac	AS2Z6krLA2M	bspinellac@kickstarter.com	1986-02-20	Bernie	Spinella
	14	amurnamed	6ahKGkfYb	amurnamed@elpais.com	1983-07-08	Alane	Murname
	15	mbloxame	4i0GpGLVDls	mbloxame@everbnation.com	1983-12-10	Marcello	Bloxam
	16	rmilbornf	1BYN00Pky	rmilbornf@mozilla.org	2001-07-12	Rowland	Milborn
	17	tmcshaneq	WHxScnMaq5	tmcshaneq@cnbc.com	1995-04-06	Tyrone	McShane
	18	tbarnwillh	MzGAN7nS0EoI	tbarnwillh@oracle.com	1985-03-19	Tracie	Barnwill
	19	larnolli	GRMJmI	larnolli@discuz.net	2002-03-01	Lion	Arnoll
	20	cvaleroj	mkdQan6	cvaleroj@reddit.com	1981-07-20	Corinne	Valero

```
(20 rows)

filmreel->
```

As desired, all twenty rows were successfully inserted into the 'Users' table. The same methodology was applied to the remaining six tables with a priority on the other two strong entities 'Film' and 'Genre' since weak entities like 'List\_items', 'Review', and 'Film\_Genre' are dependent on their primary key's existence. Next, we will show the data insertion of the rest of the tables as well as a simple SELECT command to quickly verify that the table was correctly populated. However, to conserve space in this document, not all the data will be shown for the weak entities which have up to a hundred rows of data.

## Film

Once again, any instances of the 'synopsis' attribute for Film will be omitted to avoid having the formatting distorted by on-average 1000 characters for each tuple of such attribute. The rest of the attributes for 'Film' will be shown within the insertion command and the synopsis for each row will be entered afterward through an UPDATE command with a template as follows: UPDATE Film SET synopsis = 'data' WHERE film\_id = '%d';

INSERT INTO Film:

```
filmreel-> INSERT INTO Film(film_id,film_name,release_date,duration,rating,image) VALUES
filmreel-> (1,'Faces of Schlock','2008-10-05',176,'R','nisl_nunc.jpeg')
filmreel-> ,(2,'Good Year, A','2019-08-21',90,'NC-17','odio.tiff')
filmreel-> ,(3,'Long Way Round','2013-01-10',179,'G','ridiculus_mus_vivamus.jpeg')
filmreel-> ,(4,'Cool It','2015-12-16',128,'PG-13','sapien.tiff')
filmreel-> ,(5,'Fuzz','2019-01-23',127,'PG','vitae_quam_suspendisse.tiff')
filmreel-> ,(6,'Princess Protection Program','2012-02-27',147,'G','vel_ipsum_praesent.gif')
filmreel-> ,(7,'Men with Guns','2015-01-12',103,'PG-13','mi_sit.tiff')
filmreel-> ,(8,'Left Behind II: Tribulation Force','2010-12-17',100,'PG','ut_at_dolor.tiff')
filmreel-> ,(9,'Armadillo','2011-04-23',150,'PG','pede.tiff')
filmreel-> ,(10,'Monster','2011-10-28',155,'PG-13','consequat_ut.jpeg')
filmreel-> ,(11,'Singles','2013-08-30',115,'G','porta.gif')
filmreel-> ,(12,'Adam Resurrected','2013-06-23',138,'PG-13','eget_nunc.tiff')
filmreel-> ,(13,'Innocent Blood','2018-07-30',153,'R','erat_volutpat_in.jpeg')
filmreel-> ,(14,'Doomsday Book','2018-10-31',160,'PG','libero_rutrum.jpeg')
filmreel-> ,(15,'Beautiful Girl','2019-08-13',179,'G','odio_consequat_varius.jpeg')
filmreel-> ,(16,'Italian for Beginners','2017-07-18',117,'G','non_ligula_pellentesque.jpeg')
filmreel-> ,(17,'Life 2.0','2018-12-02',136,'PG-13','aliquam.png')
filmreel-> ,(18,'King - Jari Litmanen, The (Kuningas Litmanen)','2011-07-02',142,'PG-13','id_massa_id.jpeg')
filmreel-> ,(19,'Homem Que Desafiou o Diabo, O','2017-04-18',138,'PG-13','vel_nisl.jpeg')
filmreel-> ,(20,'Karan Arjun','2009-11-12',154,'R','proin_risus_praesent.gif');
```

SELECT film\_id, film\_name, release\_date, duration, rating, image) FROM Film:

```
filmreel-> SELECT film_id, film_name, release_date, duration, rating, image FROM Film;
film_id | film_name | release_date | duration | rating | image
-----|-----|-----|-----|-----|-----
1 | Faces of Schlock | 2008-10-05 | 176 | R | nisl_nunc.jpeg
2 | Good Year, A | 2019-08-21 | 90 | NC-17 | odio.tiff
3 | Long Way Round | 2013-01-10 | 179 | G | ridiculus_mus_vivamus.jpeg
4 | Cool It | 2015-12-16 | 128 | PG-13 | sapien.tiff
5 | Fuzz | 2019-01-23 | 127 | PG | vitae_quam_suspendisse.tiff
6 | Princess Protection Program | 2012-02-27 | 147 | G | vel_ipsum_praesent.gif
7 | Men with Guns | 2015-01-12 | 103 | PG-13 | mi_sit.tiff
8 | Left Behind II: Tribulation Force | 2010-12-17 | 100 | PG | ut_at_dolor.tiff
9 | Armadillo | 2011-04-23 | 150 | PG | pede.tiff
10 | Monster | 2011-10-28 | 155 | PG-13 | consequat_ut.jpeg
11 | Singles | 2013-08-30 | 115 | G | porta.gif
12 | Adam Resurrected | 2013-06-23 | 138 | PG-13 | eget_nunc.tiff
13 | Innocent Blood | 2018-07-30 | 153 | R | erat_volutpat_in.jpeg
14 | Domsday Book | 2018-10-31 | 169 | PG | libero_rutrum.jpeg
15 | Beautiful Girl | 2019-08-13 | 179 | G | odio_consequat_varius.jpeg
16 | Italian for Beginners | 2017-07-18 | 117 | G | non_ligula_pellentesque.jpeg
17 | Life 2.0 | 2018-12-02 | 136 | PG-13 | aliquam.png
18 | King - Jari Litmanen, The (Kuningas Litmanen) | 2011-07-02 | 142 | PG-13 | id_massa_id.jpeg
19 | Homem Que Desafiou o Diabo, O | 2017-04-18 | 138 | PG-13 | vel_nisl.jpeg
20 | Karan Arjun | 2009-11-12 | 154 | R | proin_risus_praesent.gif
(20 rows)
```

## Genre

INSERT INTO Genre:

```
filmreel-> INSERT INTO Genre(genre_id,name) VALUES
filmreel-> (1,'Action')
filmreel-> ,(2,'Comedy')
filmreel-> ,(3,'Crime')
filmreel-> ,(4,'Drama')
filmreel-> ,(5,'Fantasy')
filmreel-> ,(6,'Historical')
filmreel-> ,(7,'Horror')
filmreel-> ,(8,'Romance')
filmreel-> ,(9,'Science')
filmreel-> ,(10,'Western')
filmreel-> ,(11,'Adventure')
filmreel-> ,(12,'War')
filmreel-> ,(13,'Musical')
filmreel-> ,(14,'Sports')
filmreel-> ,(15,'Documentary')
filmreel-> ,(16,'Children');
INSERT 0 16
```

SELECT \* FROM Genre:

```
filmreel-> SELECT * FROM Genre;
genre_id | name
-----|-----
1 | Action
2 | Comedy
3 | Crime
4 | Drama
5 | Fantasy
6 | Historical
7 | Horror
8 | Romance
9 | Science
10 | Western
11 | Adventure
12 | War
13 | Musical
14 | Sports
15 | Documentary
16 | Children
(16 rows)

filmreel->
```

## Film\_Genre

## INSERT INTO Film\_Genre:

```
filmreel-> INSERT INTO Film_Genre(film_id,genre_id) VALUES
filmreel-> (1,2)
filmreel-> ,(1,4)
filmreel-> ,(1,1)
filmreel-> ,(2,8)
filmreel-> ,(2,4)
filmreel-> ,(3,11)
filmreel-> ,(3,1)
filmreel-> ,(3,5)
filmreel-> ,(4,15)
filmreel-> ,(4,6)
filmreel-> ,(5,4)
```

## SELECT \* FROM Film\_Genre:

```
filmreel-> SELECT * FROM Film_Genre;
film_id | genre_id
-----+-----
1       | 2
1       | 4
1       | 1
2       | 8
2       | 4
3       | 11
3       | 1
3       | 5
4       | 15
4       | 6
5       | 4
```

## Review

### INSERT INTO Review:

```
filmreel-> INSERT INTO Review(review_id,account_id,film_id,title,score,description,review_date) VALUES
filmreel-> (1,14,11,'Public-key',10,'justo in blandit ultrices enim lorem ipsum dolor','2019-11-24')
filmreel-> ,(2,18,18,'Implementation',9,'odio cras mi pede malesuada in','2020-07-05')
filmreel-> ,(3,4,3,'Quality-focused',6,'aenean fermentum donec ut mauris','2019-04-17')
filmreel-> ,(4,8,15,'artificial intelligence',3,'nibh fusce lacus purus aliquet','2019-09-16')
filmreel-> ,(5,3,20,'Grass-roots',2,'sodales sed tincidunt eu felis','2018-11-13')
filmreel-> ,(6,19,15,'Programmable',8,'suspendisse ornare consequat lectus in est risus auctor sed','2020-04-02')
filmreel-> ,(7,4,5,'explicit',1,'sociis natoque penatibus et magnis dis parturient montes','2019-02-13')
filmreel-> ,(8,17,19,'User-centric',3,'quis libero nullam sit amet turpis elementum','2020-01-02')
filmreel-> ,(9,5,15,'secured line',10,'accumsan tellus nisi','2020-06-24')
filmreel-> ,(10,18,16,'time-frame',4,'amet diam in magna bibendum imperdiet','2019-09-23')
filmreel-> ,(11,13,14,'secondary',3,'platea dictumst aliquam augue quam sollicitudin vitae consectetur eget','2020-05-18')
filmreel-> ,(12,17,13,'Business-focused',7,'ipsum primis in faucibus orci luctus et ultrices','2019-04-20')
filmreel-> ,(13,9,10,'Multi-channelled',6,'duis bibendum felis','2018-10-31')
filmreel-> ,(14,14,8,'Robust',10,'ultrices erat tortor sollicitudin mi sit amet','2019-10-01')
filmreel-> ,(15,7,11,'tertiary',2,'eros suspendisse accumsan tortor quis turpis sed ante vivamus','2019-09-12')
```

## SELECT \* FROM Review:

```
filmreel-> SELECT * FROM Review;
review_id | account_id | film_id | title | score | description | review_date
-----+-----+-----+-----+-----+-----+-----
1 | 14 | 11 | Public-key | 10 | justo in blandit ultrices enim lorem ipsum dolor | 2019-11-24
2 | 18 | 18 | Implementation | 9 | odio cras mi pede malesuada in | 2020-07-05
3 | 4 | 3 | Quality-focused | 6 | aenean fermentum donec ut mauris | 2019-04-17
4 | 8 | 15 | artificial intelligence | 3 | nibh fusce lacus purus aliquet | 2019-09-16
5 | 3 | 20 | Grass-roots | 2 | sodales sed tincidunt eu felis | 2018-11-13
6 | 19 | 15 | Programmable | 8 | suspendisse ornare consequat lectus in est risus auctor sed | 2020-04-02
7 | 4 | 5 | explicit | 1 | sociis natoque penatibus et magnis dis parturient montes | 2019-02-13
8 | 17 | 19 | User-centric | 3 | quis libero nullam sit amet turpis elementum | 2020-01-02
9 | 5 | 15 | secured line | 10 | accumsan tellus nisi | 2020-06-24
10 | 18 | 16 | time-frame | 4 | amet diam in magna bibendum imperdiet | 2019-09-23
11 | 13 | 14 | secondary | 3 | platea dictumst aliquam augue quam sollicitudin vitae consectetur eget | 2020-05-18
12 | 17 | 13 | Business-focused | 7 | ipsum primis in faucibus orci luctus et ultrices | 2019-04-20
13 | 9 | 10 | Multi-channelled | 6 | duis bibendum felis | 2018-10-31
14 | 14 | 8 | Robust | 10 | ultrices erat tortor sollicitudin mi sit amet | 2019-10-01
15 | 7 | 11 | tertiary | 2 | eros suspendisse accumsan tortor quis turpis sed ante vivamus | 2019-09-12
16 | 12 | 3 | asymmetric | 4 | curae donec pharetra magna vestibulum aliquet ultrices erat tortor | 2020-09-11
17 | 10 | 18 | Quality-focused | 3 | at nibh in hac habitasse | 2019-10-31
18 | 20 | 16 | paradigm | 5 | curae mauris viverra diam vitae quam suspendisse | 2018-12-09
19 | 7 | 9 | Fully-configurable | 8 | et ultrices posuere cubilia curae | 2019-05-17
20 | 20 | 14 | discrete | 7 | ultrices phasellus id sapien in sapien iaculis congue vivamus metus | 2019-01-12
21 | 4 | 2 | adapter | 4 | lacus morbi quis tortor id nulla ultrices | 2020-01-16
22 | 4 | 10 | mission-critical | 9 | nunc nisl duis bibendum felis sed interdum venenatis turpis enim | 2019-04-13
23 | 1 | 1 | solution-oriented | 10 | placerat praesent blandit nam nulla integer pede justo | 2020-02-24
24 | 6 | 2 | secured line | 9 | erat quisque erat eros viverra eget congue eget semper rutrum | 2020-06-06
25 | 5 | 10 | motivating | 9 | vitae nisl nam ultrices libero non | 2020-06-06
```

## List

INSERT INTO List:

```
filmreel=> INSERT INTO List(list_id,account_id,list_title) VALUES
filmreel-> (1,16,'With the homies')
filmreel-> ,(2,9,'My Favorites')
filmreel-> ,(3,4,'For Later')
filmreel-> ,(4,19,'best action flicks')
filmreel-> ,(5,13,'Summer 2k19')
filmreel-> ,(6,20,'Dead Franchises')
filmreel-> ,(7,15,'Stupid-High Budgets')
filmreel-> ,(8,6,'Daves Faves')
filmreel-> ,(9,16,'Must-Watch')
filmreel-> ,(10,16,'Must-Avoid')
filmreel-> ,(11,5,'listoffilms.org')
filmreel-> ,(12,11,'date night')
filmreel-> ,(13,1,'disguising your 150 TB Homework Folder')
filmreel-> ,(14,9,'after anime hours')
filmreel-> ,(15,7,'On the way to iHop')
filmreel-> ,(16,4,'Fortnite 2 releases on 12/31/2020')
filmreel-> ,(17,4,'films to watch when your dog runs away');
INSERT 0 17
filmreel=>
```

SELECT \* FROM List:

```
filmreel=> SELECT * FROM List;
list_id | account_id | list_title
-----|-----|-----
1 | 16 | With the homies
2 | 9 | My Favorites
3 | 4 | For Later
4 | 19 | best action flicks
5 | 13 | Summer 2k19
6 | 20 | Dead Franchises
7 | 15 | Stupid-High Budgets
8 | 6 | Daves Faves
9 | 16 | Must-Watch
10 | 16 | Must-Avoid
11 | 5 | listoffilms.org
12 | 11 | date night
13 | 1 | disguising your 150 TB Homework Folder
14 | 9 | after anime hours
15 | 7 | On the way to iHop
16 | 4 | Fortnite 2 releases on 12/31/2020
17 | 4 | films to watch when your dog runs away
(17 rows)
```

## List Items

INSERT INTO List\_Items:

```
filmreel=> INSERT INTO List_Items(list_id,film_id) VALUES
filmreel-> (1,1)
filmreel-> ,(1,7)
filmreel-> ,(1,10)
filmreel-> ,(1,11)
filmreel-> ,(1,9)
filmreel-> ,(1,2)
filmreel-> ,(2,9)
filmreel-> ,(2,13)
filmreel-> ,(2,5)
filmreel-> ,(2,8)
filmreel-> ,(2,11)
filmreel-> ,(2,19)
filmreel-> ,(3,13)
filmreel-> ,(3,14)
filmreel-> ,(3,4)
filmreel-> ,(3,17)
```

SELECT \* FROM List\_Items:

```
filmreel=> SELECT * FROM List_Items;
list_id | film_id
-----|-----
1 | 1
1 | 7
1 | 10
1 | 11
1 | 9
1 | 2
2 | 9
2 | 13
2 | 5
2 | 8
2 | 11
2 | 19
3 | 13
3 | 14
3 | 4
3 | 17
4 | 7
4 | 10
4 | 3
4 | 9
4 | 14
5 | 2
5 | 11
5 | 4
5 | 14
5 | 3
6 | 8
-- More --
```

## Database Backups

If the schema was destroyed for any reason, the recreation of the entire database is completely possible. All table-creation SQL commands are stored in a text file in our group's shared repository along with the insertion commands that populated the database in the first place. Even if those did not exist, our mock data is still stored in both Excel and CSV form. Overall, the database could be rebuilt relatively easily if it were lost. However, the process could still be simplified even further.

One of the useful features of using DigitalOcean as our database-hosting service is that it can automatically back up the entirety of a droplet once a week and store each backup for four weeks. However, we did not utilize this method to avoid the additional monetary costs needed to enable such a feature. Another method to easily backup the entirety of the database is by utilizing MySQL's dump command 'mysqldump'. By running this command (and preferably adding a timestamp to the filename for ease of use), a backup of the database can be created and set aside for later. The benefit to this is that such a dump file will exist as long as it isn't deleted. Since the dump is an actual file, we can move it to our local machines as well as onto our shared GitHub repository. This way, we have multiple copies of our backups.

## 7 – Query Implementation

Queries:

1. List all films released after the year 2009 that are not rated R or NC-17 and at most 150 minutes in length.

```
SELECT
```

```

        f.film_id,
        f.film_name,
        f.duration,
        f.rating,
        f.release_date
FROM Film f
WHERE f.release_date >= '2010-01-01'
AND f.rating != 'R' AND f.rating != 'NC-17'
AND f.duration <= 150;

```

```

mysql> SELECT
-> f.film_id,
-> f.film_name,
-> f.duration,
-> f.rating,
-> f.release_date
-> FROM Film f
-> WHERE f.release_date >= '2010-01-01'
-> AND f.rating != 'R' AND f.rating != 'NC-17'
-> AND f.duration <= 150;

```

film_id	film_name	duration	rating	release_date
4	Cool It	128	PG-13	2015-12-16
5	Fuzz	127	PG	2019-01-23
6	Princess Protection Program	147	G	2012-02-27
7	Men with Guns	103	PG-13	2015-01-12
8	Left Behind II: Tribulation Force	100	PG	2010-12-17
9	Armadillo	150	PG	2011-04-23
11	Singles	115	G	2013-08-30
12	Adam Resurrected	138	PG-13	2013-06-23
16	Italian for Beginners	117	G	2017-07-18
17	Life 2.0	136	PG-13	2018-12-02
18	King - Jari Litmanen, The (Kuningas Litmanen)	142	PG-13	2011-07-02
19	Homem Que Desafiou o Diabo, O	138	PG-13	2017-04-18

12 rows in set (0.00 sec)

- List the most popular film in terms of review score from films released from 2015 through 2019.

```

SELECT film_name as topRatedFilm
FROM
(
SELECT f.film_id, f.film_name, r.score
FROM Film f INNER JOIN Review r
ON f.film_id = r.film_id
WHERE f.film_id IN (SELECT film_id FROM Review)
AND f.release_date >= '2015-01-01' AND f.release_date <=
'2019-12-31'
ORDER BY 3 DESC
) g
LIMIT 1;

```

```

mysql> SELECT film_name as topRatedFilm
-> FROM
-> (
-> SELECT f.film_id, f.film_name, r.score
-> FROM Film f INNER JOIN Review r
-> ON f.film_id = r.film_id
-> WHERE f.film_id IN (SELECT film_id FROM Review)
-> AND f.release_date >= '2015-01-01' AND f.release_date <= '2019-12-31'
-> ORDER BY 3 DESC
-> ) g
-> LIMIT 1;

```

topRatedFilm
A Good Year

1 row in set (0.00 sec)

3. For every user, list their first name, last name, and all the lists they made.

```
SELECT u.first_name, u.last_name, l.list_title
FROM Users u INNER JOIN List l
ON u.account_id = l.account_id
ORDER BY u.first_name, u.last_name;
```

```
mysql> SELECT u.first_name, u.last_name, l.list_title
-> FROM Users u INNER JOIN List l
-> ON u.account_id = l.account_id
-> ORDER BY u.first_name, u.last_name;
```

first_name	last_name	list_title
Annette	Warland	My Favorites
Annette	Warland	after anime hours
Bernie	Spinella	Summer 2k19
Chen	Scough	listoffilms.org
Corinne	Valero	Dead Franchises
Elsi	Rosen	date night
Fleming	Livsey	disguising your 150 TB Homework Folder
Honorita	Ateggart	films to watch when your dog runs away
Honorita	Ateggart	Fortnite 2 releases on 12/31/2020
Honorita	Ateggart	For Later
Lion	Arnoll	best action flicks
Marcello	Bloxam	Stupid-High Budgets
Michael	Crunkhorn	Dave's Faves
Rowland	Milborn	Must-Avoid
Rowland	Milborn	Must-Watch
Rowland	Milborn	With the homies
Zandra	Quixley	On the way to iHop

```
17 rows in set (0.00 sec)
```

4. Find the titles of film lists created by users who were born after 1995.

```
SELECT l.list_title
FROM Users u INNER JOIN List l
ON u.account_id = l.account_id
WHERE u.birth_date >= '1995-12-31';
```

```
mysql> SELECT l.list_title
-> FROM Users u INNER JOIN List l
-> ON u.account_id = l.account_id
-> WHERE u.birth_date >= '1995-12-31';
```

list_title
disguising your 150 TB Homework Folder
For Later
Fortnite 2 releases on 12/31/2020
films to watch when your dog runs away
My Favorites
after anime hours
date night
With the homies
Must-Watch
Must-Avoid
best action flicks

```
11 rows in set (0.00 sec)
```

5. List which genre(s) categorizes the film with the highest score among all films.

```
SELECT g.name genre_of_top_film
FROM Genre g INNER JOIN Film_Genre fg
ON g.genre_id = fg.genre_id
INNER JOIN
(
```

```

SELECT film_id, film_name
FROM
(
  SELECT f.film_id, f.film_name, r.score
  FROM Film f INNER JOIN Review r
  ON f.film_id = r.film_id
  WHERE f.film_id IN (SELECT film_id FROM Review)
  ORDER BY 3 DESC
) t
LIMIT 1
) tf
ON fg.film_id = tf.film_id;

```

```

mysql> SELECT g.name genre_of_top_film
-> FROM Genre g INNER JOIN Film_Genre fg
-> ON g.genre_id = fg.genre_id
-> INNER JOIN
-> (
-> SELECT film_id, film_name
-> FROM
-> (
-> SELECT f.film_id, f.film_name, r.score
-> FROM Film f INNER JOIN Review r
-> ON f.film_id = r.film_id
-> WHERE f.film_id IN (SELECT film_id FROM Review)
-> ORDER BY 3 DESC
-> ) t
-> LIMIT 1
-> ) tf
-> ON fg.film_id = tf.film_id;
+-----+
| genre_of_top_film |
+-----+
| Comedy            |
| Drama             |
| Action            |
+-----+
3 rows in set (0.00 sec)

```

6. List the films that have at least an 8-point review score and were released after 2010.

```

SELECT
  DISTINCT f.film_id,
  f.film_name,
  f.duration,
  f.rating,
  f.release_date
FROM Film f INNER JOIN Review r
ON r.film_id = f.film_id
WHERE r.score >= 8
AND f.release_date >= '2011-1-1';

```



```
mysql> SELECT
-> DISTINCT f.film_id,
-> f.film_name,
-> f.duration,
-> f.rating,
-> f.release_date
-> FROM Film f INNER JOIN Review r
-> ON r.film_id = f.film_id
-> WHERE r.score >= 8
-> AND f.release_date >= '2011-1-1';
```

film_id	film_name	duration	rating	release_date
2	A Good Year	90	NC-17	2019-08-21
3	Long Way Round	179	G	2013-01-10
7	Men with Guns	103	PG-13	2015-01-12
9	Armadillo	150	PG	2011-04-23
10	Monster	155	PG-13	2011-10-28
11	Singles	115	G	2013-08-30
13	Innocent Blood	153	R	2018-07-30
15	Beautiful Girl	179	G	2019-08-13
16	Italian for Beginners	117	G	2017-07-18
17	Life 2.0	136	PG-13	2018-12-02
18	King - Jari Litmanen, The (Kuningas Litmanen)	142	PG-13	2011-07-02
19	Homem Que Desafiou o Diabo, O	138	PG-13	2017-04-18

12 rows in set (0.00 sec)

7. List the three films that have the highest ratings.

```
SELECT DISTINCT film_id, film_name
FROM
(
SELECT DISTINCT f.film_id, f.film_name, r.score
FROM Film f INNER JOIN Review r
ON f.film_id = r.film_id
WHERE f.film_id IN (SELECT film_id FROM Review)
ORDER BY 3 DESC
) t
LIMIT 3;
```

```
mysql> SELECT DISTINCT film_id, film_name
-> FROM
-> (
-> SELECT DISTINCT f.film_id, f.film_name, r.score
-> FROM Film f INNER JOIN Review r
-> ON f.film_id = r.film_id
-> WHERE f.film_id IN (SELECT film_id FROM Review)
-> ORDER BY 3 DESC
-> ) t
-> LIMIT 3;
```

film_id	film_name
1	Faces of Schlock
11	Singles
8	Left Behind II: Tribulation Force

3 rows in set (0.00 sec)

8. Find all the films that are either reviewed by or included in a list by a user whose last name is 'Warland' or 'Rosen'.

```
SELECT f.film_id, f.film_name
FROM Film f
WHERE f.film_id IN
(
SELECT DISTINCT r.film_id
FROM Users u INNER JOIN Review r
ON u.account_id = r.account_id
WHERE u.last_name = 'Warland' OR u.last_name = 'Rosen'
```

```

WHERE u.last_name = 'Warland' OR u.last_name = 'Rosen'
)
OR f.film_id IN
(
SELECT DISTINCT li.film_id
FROM Users u INNER JOIN List l
ON u.account_id = l.account_id
INNER JOIN List_Items li
ON l.list_id = li.list_id
WHERE u.last_name = 'Warland' OR u.last_name = 'Rosen'
);

```

```

mysql> SELECT f.film_id, f.film_name
-> FROM Film f
-> WHERE f.film_id IN
-> (
-> SELECT DISTINCT r.film_id
-> FROM Users u INNER JOIN Review r
-> ON u.account_id = r.account_id
-> WHERE u.last_name = 'Warland' OR u.last_name = 'Rosen'
-> )
-> OR f.film_id IN
-> (
-> SELECT DISTINCT li.film_id
-> FROM Users u INNER JOIN List l
-> ON u.account_id = l.account_id
-> INNER JOIN List_Items li
-> ON l.list_id = li.list_id
-> WHERE u.last_name = 'Warland' OR u.last_name = 'Rosen'
-> );

```

film_id	film_name
1	Faces of Schlock
2	A Good Year
3	Long Way Round
4	Cool It
5	Fuzz
7	Men with Guns
8	Left Behind II: Tribulation Force
9	Armadillo
10	Monster
11	Singles
12	Adam Resurrected
13	Innocent Blood
16	Italian for Beginners
18	King - Jari Litmanen, The (Kuningas Litmanen)
19	Homem Que Desafiou o Diabo, O
20	Karan Arjun

16 rows in set (0.00 sec)

9. Find the average score of films after 2010 listed from lowest average score to highest.

```

SELECT f.film_name,
       round(avg(r.score),2) avg_score
FROM Film f INNER JOIN Review r
ON f.film_id = r.film_id
WHERE (f.film_id IN (SELECT film_id FROM Review))
AND f.release_date >= '2010-01-01'
GROUP BY f.film_id
ORDER BY avg_score ASC;

```

```
mysql> SELECT f.film_name,
-> round(avg(r.score),2) avg_score
-> FROM Film f INNER JOIN Review r
-> ON f.film_id = r.film_id
-> WHERE (f.film_id IN (SELECT film_id FROM Review))
-> AND f.release_date >= '2010-01-01'
-> GROUP BY f.film_id
-> ORDER BY avg_score ASC;
```

film_name	avg_score
Fuzz	2.67
Doomsday Book	3.40
Cool It	4.00
Singles	4.67
Adam Resurrected	5.00
Homem Que Desafiou o Diabo, O	5.20
Innocent Blood	5.40
Princess Protection Program	5.60
Italian for Beginners	5.70
Left Behind II: Tribulation Force	5.86
King - Jari Litmanen, The (Kuningas Litmanen)	6.00
A Good Year	6.00
Long Way Round	6.11
Men with Guns	6.20
Armadillo	6.33
Monster	7.00
Beautiful Girl	7.80
Life 2.0	9.00

18 rows in set (0.00 sec)

10. Save a temporary record of each user\_id, their username, age, the number of films they have reviewed, the average review score given, the number of lists they have made, and the average number of items per list.

```
CREATE TEMPORARY TABLE userinfo AS
SELECT u.account_id id, u.username,
       TIMESTAMPDIFF(YEAR, birth_date, SYSDATE()) age,
       count(DISTINCT l.list_id) lists,
       round(count(DISTINCT li.film_id) / NULLIF(count(DISTINCT
l.list_id), 0), 2) films_per_list,
       count(DISTINCT r.review_id) reviews,
       round(avg(r.score),2) avg_score
FROM Users u LEFT JOIN List l
ON u.account_id = l.account_id
LEFT JOIN Review r ON u.account_id = r.account_id
LEFT JOIN List_Items li ON l.list_id = li.list_id
GROUP BY u.account_id
ORDER BY u.account_id ASC;
```

```
mysql> CREATE TEMPORARY TABLE userinfo AS
-> SELECT u.account_id id, u.username,
-> TIMESTAMPDIFF(YEAR, birth_date, SYSDATE()) age,
-> count(DISTINCT l.list_id) lists,
-> round(count(DISTINCT li.film_id) / NULLIF(count(DISTINCT l.list_id), 0), 2) films_per_list,
-> count(DISTINCT r.review_id) reviews,
-> round(avg(r.score),2) avg_score
-> FROM Users u LEFT JOIN List l
-> ON u.account_id = l.account_id
-> LEFT JOIN Review r ON u.account_id = r.account_id
-> LEFT JOIN List_Items li ON l.list_id = li.list_id
-> GROUP BY u.account_id
-> ORDER BY u.account_id ASC;
Query OK, 26 rows affected (0.01 sec)
Records: 26 Duplicates: 0 Warnings: 0

mysql> |
```

```
mysql> select * from userinfo;
```

id	username	age	lists	films_per_list	reviews	avg_score
1	flivsey0	18	1	7.00	5	6.60
2	mwoolham1	29	0	NULL	0	NULL
3	jgerling2	22	0	NULL	4	6.50
4	hateggart3	24	3	4.00	8	5.25
5	cscough4	35	1	6.00	7	5.71
6	mcrunkhorn5	36	1	4.00	5	5.60
7	zquixley6	33	1	4.00	7	6.86
8	glerway7	28	0	NULL	2	3.00
9	awarland8	23	2	5.00	6	5.83
10	emoulster9	25	0	NULL	5	5.00
11	erosena	22	1	6.00	6	4.83
12	egrindlayb	20	0	NULL	1	4.00
13	bspinellac	34	1	5.00	3	6.33
14	amurnamed	37	0	NULL	7	5.86
15	mbloxame	37	1	8.00	7	5.14
16	rmilbornf	19	3	5.67	7	5.57
17	tmcshaneg	25	0	NULL	6	5.17
18	tbarnwillh	35	0	NULL	5	6.20
19	larnolli	18	1	5.00	5	5.40
20	cvaleroj	39	1	5.00	4	6.50
21	clara14	23	0	NULL	0	NULL
44	Darian	21	6	0.00	5	7.60
45	jtesty	40	2	0.00	0	NULL
46	jtestyss	40	1	0.00	0	NULL
47	dudebro	1019	1	0.00	0	NULL
48	mikeock23	20	1	0.00	0	NULL

```
26 rows in set (0.00 sec)
```

# Phase 4: Database Programming (Stored Procedures, Functions, Views, Triggers)

## 8 – Programming Logic for SQL

### 8.1 – Introduction

SQL is a programming language specifically designed to handle databases. Unlike other programming languages such as C or C++ which are more object-oriented or procedure-oriented, SQL centers around the concept of databases and tables. For instance, SQL cannot be used to make an operating system.

There are three tools in SQL which are widely used in SQL programming: views, procedures or functions, and triggers. Views are essentially tables, but they do not store or create data on the database. They are used to quickly display tables and to query the database without altering anything. Stored procedures or functions are mechanisms which allows users to query the database without using a database management system like MySQL directly, and they are “shortcuts” used to expedite querying. For example, a procedure could be implemented to immediately show all the lists a user has made instead of having to always type out the query. Some SQL languages have a key distinction between procedures and functions: MySQL functions return a result after a function is executed while procedures do not. Lastly, triggers are mechanisms which can also expedite querying similar to stored procedures and function. However, they are designed to happen automatically in case of a change occurring to a specified table. For example, a trigger can be created to delete rows in table ‘Lists’ that use the primary id of a row in ‘Users’ as soon as it is deleted from ‘Users’. The trigger is executed immediately after the deletion without the database manager ever having to manually command it. This is useful for tasks such as updating multiple tables when inserting something into a table.

### 8.2 – Syntax of Programming Logic

#### Views

##### Syntax

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = user]
    [SQL SECURITY { DEFINER | INVOKER }]
```

```
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

In MySQL syntax, after the “CREATE” keyword, users would have to provide the name of the table being created. Afterwards, the user would query the database, and the resulting table from the query would be named as a new view. The syntax is identical with MariaDB’s syntax for creating views. In contrast, PostGres does not have any fields to handle security, but it is otherwise identical, as well.

## Procedures/ Functions

### Syntax: Procedure

```
CREATE
    [DEFINER = user]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

proc_parameter:      [ IN | OUT | INOUT ] param_name type
type:                Any valid MySQL data type
characteristic: {
    COMMENT 'string'
    | LANGUAGE SQL
    | [NOT] DETERMINISTIC
    | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
    | SQL SECURITY { DEFINER | INVOKER }
}
routine_body:        Valid SQL routine statement
```

### Syntax: Function

```
CREATE
    [DEFINER = user]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

func_parameter:      param_name type
type:                Any valid MySQL data type
characteristic: {
    COMMENT 'string'
```

```

| LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
}
routine_body: Valid SQL routine statement

```

In MySQL syntax, the general way of creating functions or procedures is to name the function or procedure after the “CREATE” key word and state the return type if it is a function. If it is a procedure, then this step is skipped. Then the characteristics of the function or procedure, such as the function or procedure’s programming language, is stated. Local variables would then be declared, and in between the “BEGIN” and “END” keywords, the function or procedure’s instructions would be defined. Functions and procedures can be invoked using the “CALL” keyword, and this is the also the case in MariaDB and PostGres.

## Triggers

### Syntax

```

CREATE
    [DEFINER = user]
    TRIGGER trigger_name    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]    trigger_body

trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
trigger_order: { FOLLOWS | PRECEDES } other_trigger_name

```

Triggers in PostGres are created by first specifying the trigger’s name followed by a keyword which indicates when the trigger will occur, such as “BEFORE” or “AFTER.” After specifying when it will happen, an event is specified. Then a procedure can be executed. The syntax for triggers in MariaDB is nearly the same as the syntax in PostGres. In addition to all the features of MariaDB’s and MySQL’s functions and procedures provide, PostGres also allows “TRUNCATE” to be a valid event for a trigger.

## 8.3 – Implementation

### 8.3.1 – Views

- A view that shows each user's id, username, and age who are between 18-25 years of age.

```
CREATE OR REPLACE VIEW young_users AS
SELECT      u.account_id id,
            u.username,
            TIMESTAMPDIFF(YEAR, birth_date, SYSDATE()) age
FROM Users u
WHERE (TIMESTAMPDIFF(YEAR, birth_date, SYSDATE()) <= 25)
AND (TIMESTAMPDIFF(YEAR, birth_date, SYSDATE()) >= 18);
```

Query:

```
SELECT * FROM young_users;
```

Results:

```
mysql> select * from young_users;
+----+-----+-----+
| id | username | age |
+----+-----+-----+
| 1  | flivsey0 | 18  |
| 3  | jgerling2 | 22  |
| 4  | hateggart3 | 24  |
| 9  | awarland8 | 22  |
| 10 | emoulster9 | 25  |
| 11 | erosena | 22  |
| 12 | egrindlayb | 20  |
| 16 | rmilbornf | 19  |
| 17 | tmcshang | 25  |
| 19 | larno11i | 18  |
+----+-----+-----+
10 rows in set (0.00 sec)

mysql>
```

- A view that shows each film's id, name, release date, duration, rating, image, synopsis, number of reviews, average review score, and number of list inclusions.

```
CREATE OR REPLACE VIEW film_stats AS
SELECT f.film_id, f.film_name,
       f.release_date, f.duration,
       f.rating, f.image,
       count(DISTINCT r.review_id) reviews,
       round(avg(r.score),2) avg_score,
       count(DISTINCT li.list_id) listings,
       GROUP_CONCAT(DISTINCT g.name SEPARATOR ', ') genre,
       f.synopsis
FROM Film f LEFT JOIN Review r
ON f.film_id = r.film_id
LEFT JOIN List_Items li ON f.film_id = li.film_id
INNER JOIN Film_Genre fg ON f.film_id = fg.film_id
INNER JOIN Genre g ON fg.genre_id = g.genre_id
GROUP BY f.film_id
```



```
ORDER BY f.film_id ASC;
```

Query:

```
SELECT film_id, film_name, release_date, duration, rating,
reviews, avg_score, listings FROM film_stats;
```

Results:

```
mysql> SELECT film_id, film_name, release_date, duration, rating, reviews, avg_score, listings FROM film_stats;
```

film_id	film_name	release_date	duration	rating	reviews	avg_score	listings
1	Faces of Schlock	2008-10-05	176	R	7	7.29	6
2	A Good Year	2019-08-21	90	NC-17	7	6.14	7
3	Long Way Round	2013-01-10	179	G	9	6.11	9
4	Cool It	2015-12-16	128	PG-13	1	4.00	8
5	Fuzz	2019-01-23	127	PG	4	3.50	5
6	Princess Protection Program	2012-02-27	147	G	6	6.17	4
7	Men with Guns	2015-01-12	103	PG-13	5	6.20	6
8	Left Behind II: Tribulation Force	2010-12-17	100	PG	7	5.86	3
9	Armadillo	2011-04-23	150	PG	3	6.33	6
10	Monster	2011-10-28	155	PG-13	6	7.17	4
11	Singles	2013-08-30	115	G	3	4.67	6
12	Adam Resurrected	2013-06-23	138	PG-13	2	5.00	7
13	Innocent Blood	2018-07-30	153	R	5	5.40	5
14	Doomsday Book	2018-10-31	169	PG	5	3.40	4
15	Beautiful Girl	2019-08-13	179	G	5	7.80	4
16	Italian for Beginners	2017-07-18	117	G	10	5.70	2
17	Life 2.0	2018-12-02	136	PG-13	1	9.00	7
18	King - Jari Litmanen, The (Kuningas Litmanen)	2011-07-02	142	PG-13	9	6.00	4
19	Homem Que Desafiou o Diabo, O	2017-04-18	138	PG-13	5	5.20	2
20	Karan Arjun	2009-11-12	154	R	5	2.60	4

20 rows in set (0.00 sec)

- A view that shows each user's id, username, age, birthdate, the number of films they have reviewed, and the number of lists they have created.

```
CREATE OR REPLACE VIEW user_stats AS
SELECT u.account_id id, u.username,
       TIMESTAMPDIFF(YEAR, birth_date, SYSDATE()) age,
       count(DISTINCT l.list_id) lists,
       round(count(DISTINCT li.film_id) / NULLIF(count(DISTINCT
l.list_id), 0), 2) films_per_list,
       count(DISTINCT r.review_id) reviews,
       round(avg(r.score),2) avg_score
FROM Users u LEFT JOIN List l
ON u.account_id = l.account_id
LEFT JOIN Review r ON u.account_id = r.account_id
LEFT JOIN List_Items li ON l.list_id = li.list_id
WHERE u.user_type = 0
GROUP BY u.account_id
ORDER BY u.account_id ASC;
```

Query:

```
SELECT * FROM user_stats;
```

Results:

```
mysql> select * from user_stats;
```

id	username	age	lists	films_per_list	reviews	avg_score
1	flivsey0	18	1	7.00	5	6.60
2	mwoolham1	29	0	NULL	0	NULL
3	jgerling2	22	0	NULL	4	6.50
4	hateggart3	24	3	4.00	8	5.25
5	cscough4	35	1	6.00	7	5.71
6	mcrunkhorn5	36	1	4.00	5	5.60
7	zquixley6	33	1	4.00	7	6.86
8	glerway7	28	0	NULL	2	3.00
9	awarland8	23	2	5.00	6	5.83
10	emoulster9	25	0	NULL	5	5.00
11	erosena	22	1	6.00	6	4.83
12	egrindlayb	20	0	NULL	1	4.00
13	bspinellac	34	1	5.00	3	6.33
14	amurnamed	37	0	NULL	7	5.86
15	mbloxame	37	1	8.00	7	5.14
16	rmilbornf	19	3	5.67	7	5.57
17	tmcshaneg	25	0	NULL	6	5.17
18	tbarnwillh	35	0	NULL	5	6.20
19	larnolli	18	1	5.00	5	5.40
20	cvaleroj	39	1	5.00	4	6.50
44	Darian	21	6	0.00	5	7.60
45	jtesty	40	2	0.00	0	NULL
46	jtestyss	40	1	0.00	0	NULL
47	dudebro	1019	1	0.00	0	NULL
48	mikeock23	20	1	0.00	0	NULL

```
25 rows in set (0.01 sec)
```

### 8.3.2 – Stored procedures/functions

- A stored procedure for inserting a new review with information supplied by the user.

```
delimiter //
CREATE PROCEDURE insert_review(
    IN id INT,
    IN film INT,
    IN t VARCHAR(255),
    IN s INT,
    IN descr VARCHAR(2047)
)
BEGIN
    DECLARE rID INT;
    SELECT MAX(review_id)+1 INTO rID FROM Review;
    INSERT INTO Review(
        review_id,account_id,film_id,title,score,description,review_date
    )
        VALUES (rID, id, film, t, s, descr, SYSDATE());
END //
delimiter;
```

## Procedure Call:

```
mysql> call insert_review(21,10,'pretty watchable',7,'i feel like the flower could have rotted');
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> SELECT * from Review ORDER BY 1 DESC;
```

review_id	account_id	film_id	title	score	description	review_date
101	21	10	pretty watchable	7	i feel like the flower could have rotted	2020-11-29
100	5	16	instruction set	5	dui nec nisi volutpat eleifend	2020-11-29
99	9	12	web-enabled	4	porta volutpat quam pede lobortis ligula	2019-08-26
98	11	10	needs-based	2	tellus nulla ut erat id mauris vulputate elementum	2019-03-31
97	15	14	initiative	2	et magnis dis parturient montes nascetur	2019-06-20
96	7	1	Intuitive	5	mi integer ac	2020-04-04
95	17	5	project	1	justo sit amet sapien dignissim vestibulum vestibulum ante ipsum	2019-08-22
94	17	13	definition	1	blandit nam nulla integer pede justo lacinia eget tincidunt eget	2019-12-23
93	18	16	exuding	4	lobortis est phasellus sit amet erat nulla tempus vivamus	2020-08-04
92	6	19	Proactive	6	nullam molestie nibh in lectus pellentesque at nulla suspendisse potenti	2019-08-09
91	6	8	algorithm	3	turpis eget elit sodales scelerisque mauris sit	2019-06-27
90	16	7	Programmable	9	ipsum dolor sit amet consectetur adipiscing elit proin risus	2020-09-25
89	20	16	definition	6	amet turpis elementum ligula	2019-03-10
88	16	2	workforce	10	praesent lectus vestibulum quam sapien varius ut blandit	2019-11-26
87	10	7	Horizontal	9	et ultrices posuere cubilia curae duis faucibus accumsan odio	2019-04-08
86	9	13	Automated	9	turpis nec euismod	2018-10-21
85	20	13	Multi-layered	8	augue vel accumsan tellus nisi eu orci mauris lacinia	2020-08-16
84	5	7	Vision-oriented	6	diam nam tristique tortor eu pede	2020-08-09
83	10	11	Up-sized	2	nunc vestibulum ante ipsum primis in faucibus	2019-09-15
82	14	13	alliance	2	vestibulum vestibulum ante	2020-02-28
81	9	16	Re-contextualized	6	id ligula suspendisse ornare consequat lectus in est risus	2019-06-29
80	16	1	24/7	2	primis in faucibus orci luctus et	2020-06-09
79	9	3	dedicated	9	sem fusce consequat nulla nisl nunc nisl duis	2020-04-26
78	9	20	Fully-configurable	1	imperdiet nullam orci pede venenatis non sodales sed tincidunt	2019-12-31

- A stored procedure for deleting a review based on the primary key of the Review table.

```
delimiter //
CREATE PROCEDURE remove_review(
    IN rid INT
)
BEGIN
    DELETE FROM Review WHERE review_id = rid;
END //
delimiter;
```

## Procedure Call:

```
mysql> call remove_review(101);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * from Review ORDER BY 1 DESC;
```

review_id	account_id	film_id	title	score	description	review_date
100	5	16	instruction set	5	dui nec nisi volutpat eleifend	2020-11-29
99	9	12	web-enabled	4	porta volutpat quam pede lobortis ligula	2019-08-26
98	11	10	needs-based	2	tellus nulla ut erat id mauris vulputate elementum	2019-03-31
97	15	14	initiative	2	et magnis dis parturient montes nascetur	2019-06-20
96	7	1	Intuitive	5	mi integer ac	2020-04-04
95	17	5	project	1	justo sit amet sapien dignissim vestibulum vestibulum ante ipsum	2019-08-22
94	17	13	definition	1	blandit nam nulla integer pede justo lacinia eget tincidunt eget	2019-12-23
93	18	16	exuding	4	lobortis est phasellus sit amet erat nulla tempus vivamus	2020-08-04
92	6	19	Proactive	6	nullam molestie nibh in lectus pellentesque at nulla suspendisse potenti	2019-08-09
91	6	8	algorithm	3	turpis eget elit sodales scelerisque mauris sit	2019-06-27
90	16	7	Programmable	9	ipsum dolor sit amet consectetur adipiscing elit proin risus	2020-09-25
89	20	16	definition	6	amet turpis elementum ligula	2019-03-10
88	16	2	workforce	10	praesent lectus vestibulum quam sapien varius ut blandit	2019-11-26
87	10	7	Horizontal	9	et ultrices posuere cubilia curae duis faucibus accumsan odio	2019-04-08
86	9	13	Automated	9	turpis nec euismod	2018-10-21
85	20	13	Multi-layered	8	augue vel accumsan tellus nisi eu orci mauris lacinia	2020-08-16
84	5	7	Vision-oriented	6	diam nam tristique tortor eu pede	2020-08-09
83	10	11	Up-sized	2	nunc vestibulum ante ipsum primis in faucibus	2019-09-15
82	14	13	alliance	2	vestibulum vestibulum ante	2020-02-28
81	9	16	Re-contextualized	6	id ligula suspendisse ornare consequat lectus in est risus	2019-06-29
80	16	1	24/7	2	primis in faucibus orci luctus et	2020-06-09
79	9	3	dedicated	9	sem fusce consequat nulla nisl nunc nisl duis	2020-04-26
78	9	20	Fully-configurable	1	imperdiet nullam orci pede venenatis non sodales sed tincidunt	2019-12-31

- A function for returning the list of films, their number of reviews, and average review scores (rounded to two decimal places) over a specified number of months.

```
delimiter //
CREATE PROCEDURE filmsts(IN _months INT)
BEGIN
    SELECT f.film_id as id,
           f.film_name as name,
           count(r.score) as num_of_reviews,
           round(AVG(r.score),2) as avg_score
    FROM Film f LEFT JOIN Review r
    ON f.film_id = r.film_id
    WHERE r.review_date >= DATE_SUB(CURDATE(), INTERVAL _months
MONTH)
    GROUP BY f.film_id
    ORDER BY f.film_id ASC;
END //
delimiter ;
```

Procedure Call:

```
mysql> call film_stats(20);
```

id	name	num_of_reviews	avg_score
1	Faces of Schlock	6	7.17
2	A Good Year	4	6.50
3	Long Way Round	8	6.75
4	Cool It	1	4.00
5	Fuzz	2	3.50
6	Princess Protection Program	3	6.67
7	Men with Guns	5	6.20
8	Left Behind II: Tribulation Force	7	5.86
9	Armadillo	1	8.00
10	Monster	4	7.25
11	Singles	3	4.67
12	Adam Resurrected	2	5.00
13	Innocent Blood	4	4.50
14	Doomsday Book	4	2.50
15	Beautiful Girl	5	7.80
16	Italian for Beginners	8	6.13
17	Life 2.0	1	9.00
18	King - Jari Litmanen, The (Kuningas Litmanen)	8	5.63
19	Homem que Desafiou o Diabo, o	4	5.50
20	Karan Arjun	4	2.75

```
20 rows in set (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
```

### 8.3.3 – Triggers

- Trigger for inserting a row into a table: insert default list template upon the creation of a new user.

```
delimiter //
CREATE TRIGGER defaultlist
    AFTER INSERT
    ON Users FOR EACH ROW
BEGIN
    DECLARE lid INT;
    SELECT MAX(list_id)+1 INTO lid FROM List;
    INSERT INTO List(list_id, account_id, list_title)
    VALUES(lid, NEW.account_id, 'My First List');
END //
```

delimiter ;

### Trigger Example:

```
mysql> insert into Users(username, password, email, birth_date, first_name, last_name) VALUES ('clara14','jeloce','clara14@csu.edu','1997-12-03','cesar','lara');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select * from Users;
```

	account_id	username	password	email	birth_date	first_name	last_name
1		flivsey0	yccvabqax9	flivsey0@blog.com	2002-10-08	Fleming	Livsey
2		mwoolham1	V5kiew	mwoolham1@va.gov	1990-12-26	Mariska	woolham
3		jgerling2	SbCe8p9	jgerling2@over-blog.com	1998-01-15	Jannelle	Gerling
4		hateggart3	Id15SElHEIm6	hateggart3@reddit.com	1996-05-06	Honorla	Ateggart
5		cscough4	FduigUaqam5J	cscough4@tiny.cc	1985-06-29	Chen	Scough
6		mcrunkhorn5	ZqumaJWggeg	mcrunkhorn5@soup.io	1984-06-14	Michael	Crunkhorn
7		zquixley6	HIZHNMxM1R	zquixley6@geocities.com	1987-02-25	Zandra	Quixley
8		glerway7	HTTDyZdapSKf	glerway7@dropbox.com	1992-03-13	Gerhardt	Lerway
9		awarland8	wt9ts0BRqT	awarland8@illinois.edu	1998-02-22	Annette	warland
10		emoulster9	n39ycvR5uRm	emoulster9@facebook.com	1995-02-11	Ezra	Moulster
11		erosena	MV1XOM19	erosena@omniture.com	1998-06-19	Elsi	Rosen
12		egrindlayb	60akxkc	egrindlayb@admin.ch	2000-01-26	Enrika	Grindlay
13		bspinellac	ASZ2ZdRLA2M	bspinellac@kickstarter.com	1986-02-20	Bernie	Spinella
14		amurnamed	6ahkgkfyb	amurnamed@elpais.com	1983-07-08	Alane	Murname
15		mbloxame	4i0GpgLYD1s	mbloxame@reverbnation.com	1983-12-10	Marcello	Bloxam
16		rmilbornf	1BYN0OPky	rmilbornf@mozilla.org	2001-07-12	Rowland	Milborn
17		tmcshane	WHXSCNMaq5	tmcshane@cnbc.com	1995-04-06	Tyrone	McShane
18		tbarnwillh	MzGAN7ns0EoI	tbarnwillh@oracle.com	1985-03-19	Tracie	Barnwill
19		larnolli	GRWJMI	larnolli@discuz.net	2002-03-01	Lion	Arnoll
20		cvaleroj	gk0Qan6	cvaleroj@reddit.com	1981-07-20	Corinne	Valero
21		clara14	jeloce	clara14@csu.edu	1997-12-03	cesar	Tara

21 rows in set (0.00 sec)

```
mysql> select * from List;
```

	list_id	account_id	list_title
1	16		With the homies
2	9		My Favorites
3	4		For Later
4	19		best action flicks
5	13		Summer 2k19
6	20		Dead Franchises
7	15		Stupid-High Budgets
8	6		Dave's Faves
9	16		Must-Watch
10	16		Must-Avoid
11	5		listofFilms.org
12	11		date night
13	1		disguising your 150 TB Homework Folder
14	9		after anime hours
15	7		On the way to iHop
16	4		Fortnite 2 releases on 12/31/2020
17	4		films to watch when your dog runs away
18	21		My First List

18 rows in set (0.00 sec)

- Trigger for updating a row in a table: update Review's review date to be the current date whenever a modification is made to any aspect of the review.

```
delimiter //
CREATE TRIGGER update_review
    BEFORE UPDATE
    ON Review FOR EACH ROW
BEGIN
    SET new.review_date = CURDATE();
END //
delimiter ;
```

### Trigger Example:

```
mysql> select * from Review where account_id = 7;
```

review_id	account_id	film_id	title	score	description	review_date
15	7	11	tertiary	2	eros suspendisse accumsan tortor quis turpis sed ante vivamus	2019-09-12
19	7	9	Fully-configurable	8	et ultrices posuere cubilia curae	2019-05-17
32	7	3	Down-sized	10	mattis nibh ligula	2019-06-27
34	7	1	Vision-oriented	10	natoque penatibus et magnis dis parturient montes nascetur ridiculus mus	2019-09-19
54	7	18	eco-centric	3	ut nulla sed accumsan felis ut at	2020-02-29
61	7	3	moratorium	10	non ligula pellentesque ultrices phasellus id sapien in sapien	2020-10-06
96	7	1	Intuitive	5	mi integer ac	2020-04-04

```
7 rows in set (0.00 sec)
```

```
mysql> select * from Review where review_id = 7;
```

review_id	account_id	film_id	title	score	description	review_date
7	4	5	explicit	1	sociis natoque penatibus et magnis dis parturient montes	2019-02-13

```
1 row in set (0.00 sec)
```

```
mysql> update Review set score = 5 where review_id = 7;
```

Query OK, 1 row affected (0.01 sec)  
Rows matched: 1 changed: 1 Warnings: 0

```
mysql> select * from Review where review_id = 7;
```

review_id	account_id	film_id	title	score	description	review_date
7	4	5	explicit	5	sociis natoque penatibus et magnis dis parturient montes	2020-11-29

```
1 row in set (0.01 sec)
```

- Trigger for deleting a row from a table: Before deleting a user, drop all Reviews and Lists with their account\_id, which will result in a CASCADE DELETE and remove all List\_items for each deleted List\_id.

```
delimiter //
```

```
CREATE TRIGGER deactivate_user
```

```
    BEFORE DELETE
```

```
    ON Users FOR EACH ROW
```

```
BEGIN
```

```
    DELETE FROM Review WHERE Review.account_id =
```

```
OLD.account_id;
```

```
    DELETE FROM List WHERE List.account_id = OLD.account_id;
```

```
END //
```

```
delimiter ;
```

### Trigger Example:

```
mysql> SELECT u.account_id, u.username, l.list_title, f.film_name
-> FROM Users u INNER JOIN List l
-> ON u.account_id = l.account_id
-> INNER JOIN List_Items li ON l.list_id = li.list_id
-> INNER JOIN Film f ON li.film_id = f.film_id
-> WHERE u.account_id = 21;
```

account_id	username	list_title	film_name
21	clara14	My First List	A Good Year
21	clara14	My First List	Fuzz
21	clara14	test list tocks 1	Fuzz
21	clara14	test list tocks 1	Armadillo
21	clara14	test list tocks 1	Monster
21	clara14	test list tocks 2	Singles
21	clara14	test list tocks 2	Adam Resurrected

7 rows in set (0.00 sec)

```
mysql> delete from Users where account_id = 21;
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT u.account_id, u.username, l.list_title, f.film_name
-> FROM Users u INNER JOIN List l
-> ON u.account_id = l.account_id
-> INNER JOIN List_Items li ON l.list_id = li.list_id
-> INNER JOIN Film f ON li.film_id = f.film_id
-> WHERE u.account_id = 21;
```

Empty set (0.00 sec)

# Phase 5: Graphical User Interface (GUI) Application Development

## 9 – GUI Development

### 9.1 – GUI Functionalities and User Groups

Our project used a web graphical user interface (GUI). Work was done remotely by accessing the `/var/www/filmreel/` directory through a digital ocean droplet which contained our MySQL database and LAMP stack. We used PHP, HTML, and JavaScript for our GUI.

#### 9.1.1 – Itemized Descriptions of the GUI

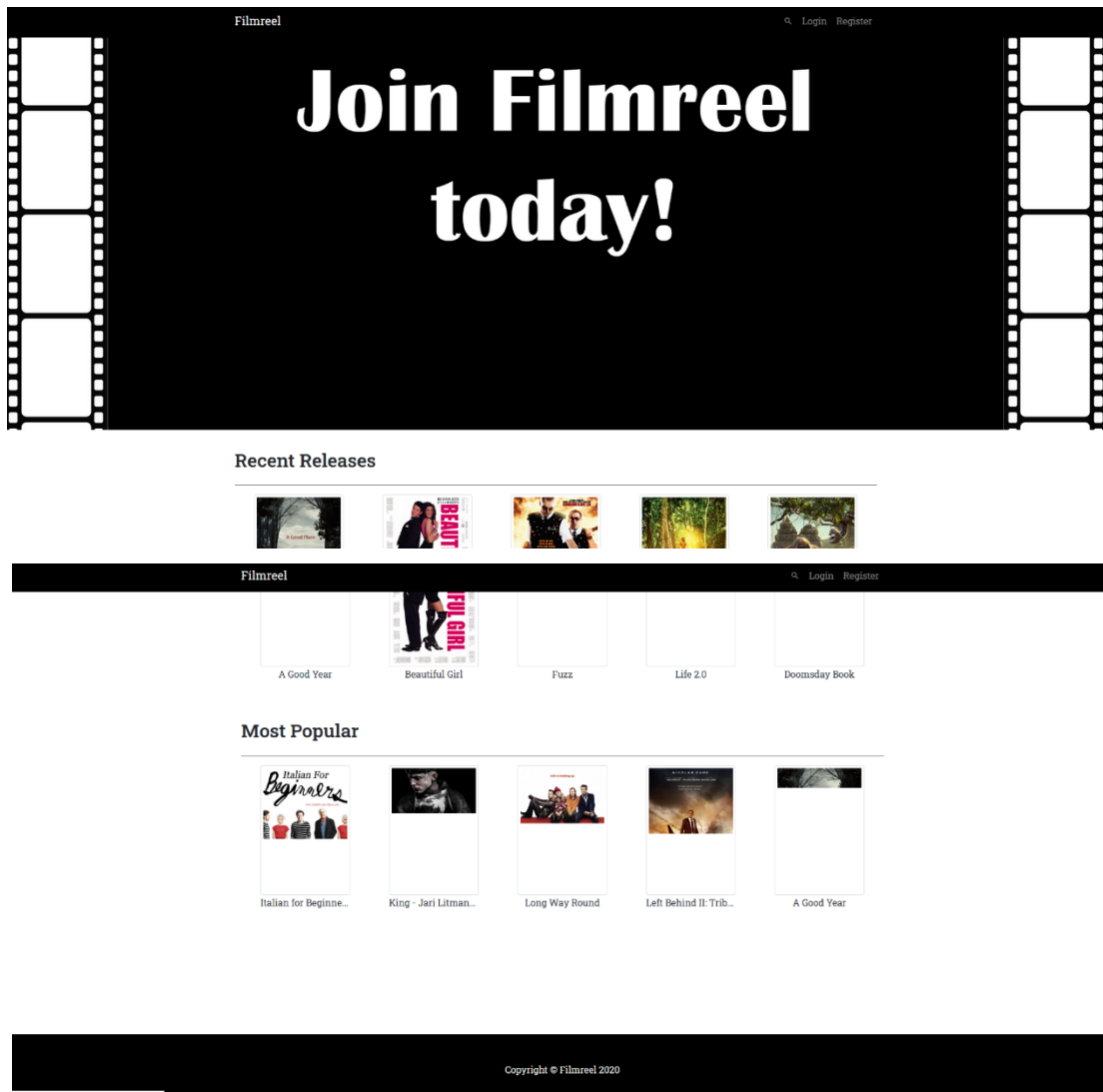
There are three user groups that the GUI supports: users, administrators, and guests. Through the GUI, users can create lists, add films to those lists, and make reviews for films. They can also remove reviews, but only ones that they have written themselves. Administrators have the same abilities as users, but they cannot make lists. However, they can remove any review, regardless of whether they have written it themselves. Lastly, guests can only search for films.

#### 9.1.2 – Screenshots and Walkthrough

##### Guest

This is the opening page of the website which has a flashy screen which urges the guest to join the site. This index page is the same regardless of which user group has it open: there will always be this screen as well as the “most popular” and “recent releases” sections below it which displays the most popular and the most recent movie releases contained in the database.





There is also a header which acts as a navigation bar, and this is present in all pages and user groups. The “Filmreel” text at the left brings the guest back to the index page, the magnifying glass reveals the search bar, and the Login and Register buttons brings the guest to the Login and Register pages respectively.




After inputting text into the search bar, the guest is brought to a results page which shows all films which somewhat matches the input.

Filmreel



[Login](#)
[Register](#)

## Search Results

1 results were found for the search for **lif**



**Life 2.0**


 Release Date: December 2, 2018
  Rated: PG-13

**Synopsis:** Life two syn - In tempor, turpis nec euismod scelerisque, quam turpis adipiscing lorem, vitae mattis nibh ligula nec sem. Duis aliquam convallis nunc. Proin at turpis a pede posuere nonummy. Integer non velit. Donec diam neque, vestibulum eget, vulputate ut, ultrices vel, augue. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec pharetra, magna vestibulum aliquet ultrices, erat tortor sollicitudin mi, sit amet lobortis sapien sapien non mi. Integer ac neque. Duis bibendum. Morbi non quam nec dui luctus rutrum. Nulla tellus. In sagittis dui vel nisl. D

Guests can also view the film page of a movie by clicking on either the film name or the film's image. The film page shows more detailed movie information as well as user-written reviews.

Filmreel

[Login](#)
[Register](#)



**Life 2.0**

Release Date: December 2, 2018  
 Rating: PG-13  
 Genre: Documentary Historical  
 Duration: 136 minutes

**Synopsis:**  
 Life two syn - In tempor, turpis nec euismod scelerisque, quam turpis adipiscing lorem, vitae mattis nibh ligula nec sem. Duis aliquam convallis nunc. Proin at turpis a pede posuere nonummy. Integer non velit. Donec diam neque, vestibulum eget, vulputate ut, ultrices vel, augue. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec pharetra, magna vestibulum aliquet ultrices, erat tortor sollicitudin mi, sit amet lobortis sapien sapien non mi. Integer ac neque. Duis bibendum. Morbi non quam nec dui luctus rutrum. Nulla tellus. In sagittis dui vel nisl. Duis ac nibh. Fusce lacus purus, aliquet at, feugiat non, pretium quis, lectus. Suspendisse potenti. In eleifend quam a odio. In hac habitasse platea dictumst. Maecenas ut massa quis augue luctus tincidunt. Nulla mollis molestie lorem.

## Reviews for Life 2.0

[Log in](#) or [Register](#) to start writing reviews

**Robust**

posted by [tharriwillh](#) on September 12, 2019

**Score:** 9/10

sed tristique in tempus sit amet sem fusce consequat nulla

The Login page prompts the guest to enter login credentials. It also proposes the guest to register, if the guest does not have an account.

The screenshot shows the 'Sign In' page of the Filmreel application. At the top, a black header bar contains the 'Filmreel' logo on the left and a search icon followed by 'Login' and 'Register' links on the right. The main content area is white and features a central 'Sign In' form. The form has a title 'Sign In' at the top. Below the title are two input fields: 'Username' and 'Password'. A blue 'Login' button is positioned below the password field. At the bottom of the form, there is a link that says 'No account? Register here'.

The Register prompts the guest to enter information needed to create a user account as shown below. It also proposes to be redirected to the login page if the guest already has an account

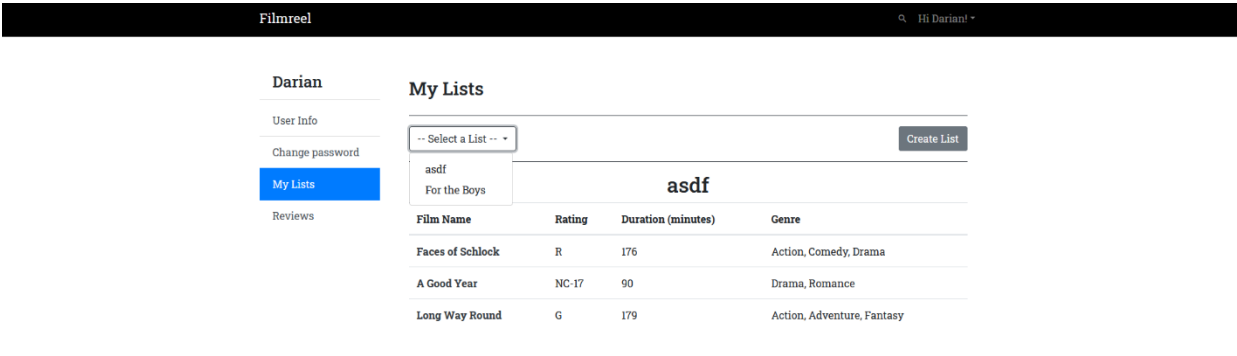
The screenshot shows the 'Register' page of the Filmreel application. The header is identical to the previous page. The main content area is white and features a central 'Register' form. The form has a title 'Register' at the top. Below the title are several input fields: 'Email', 'First Name', 'Last Name', 'Date of Birth' (with a hint 'mm / dd / yyyy'), 'Username', and 'Password'. A blue 'Register' button is positioned below the password field. At the bottom of the form, there is a link that says 'Already have an account? Login here!'.

## User

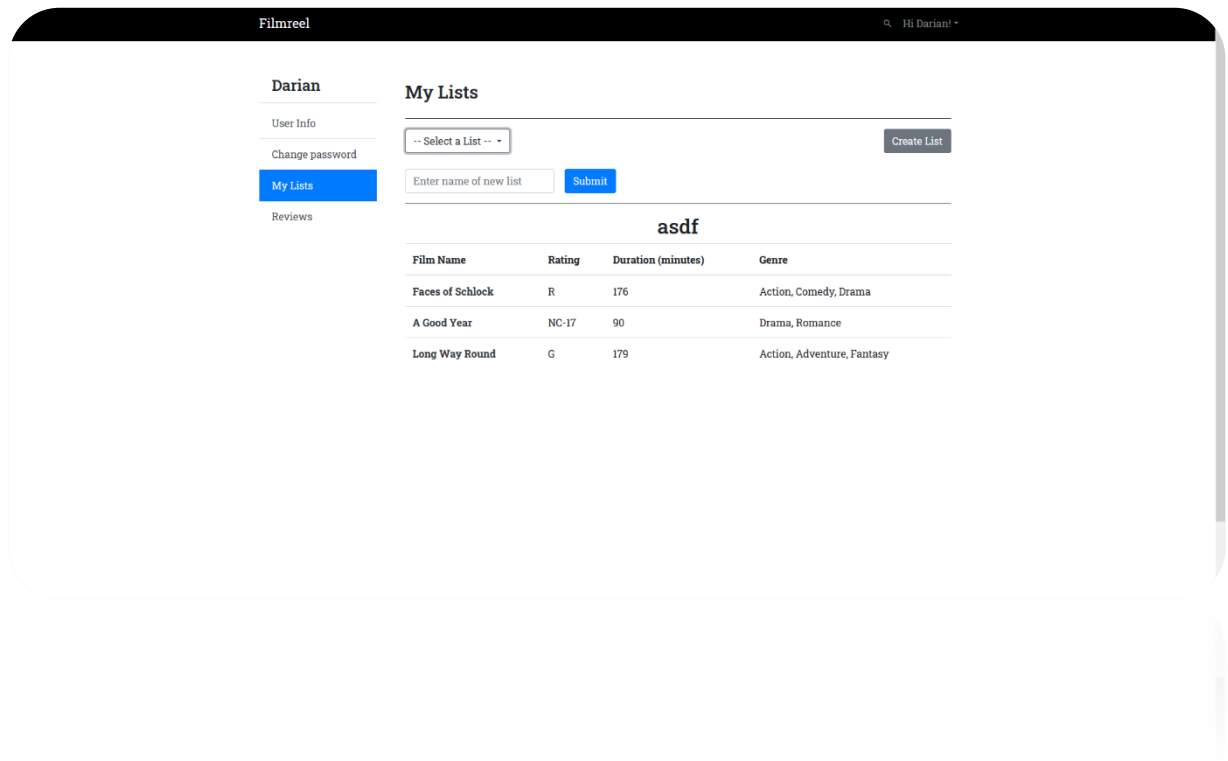
Users have access to all the same pages as guests with minor differences. In addition, they also have access to their own user page.

After logging in using a user account, the user is directed to the user page. Basic user information is displayed initially, and the tabs on the left will display the appropriate content. The “Change Password” tab will lead to a way to change the account’s password. The “My Lists” tab leads to the account’s user-made lists, and after selecting

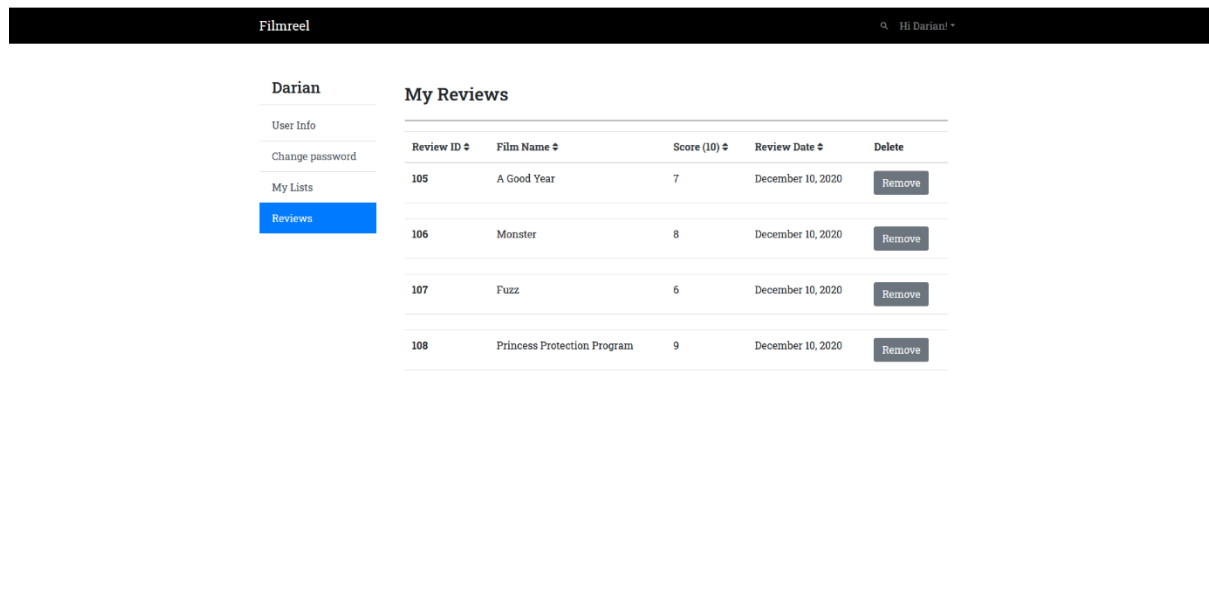
a list from the dropdown button, the films belonging to that list will be shown along with some information about them.



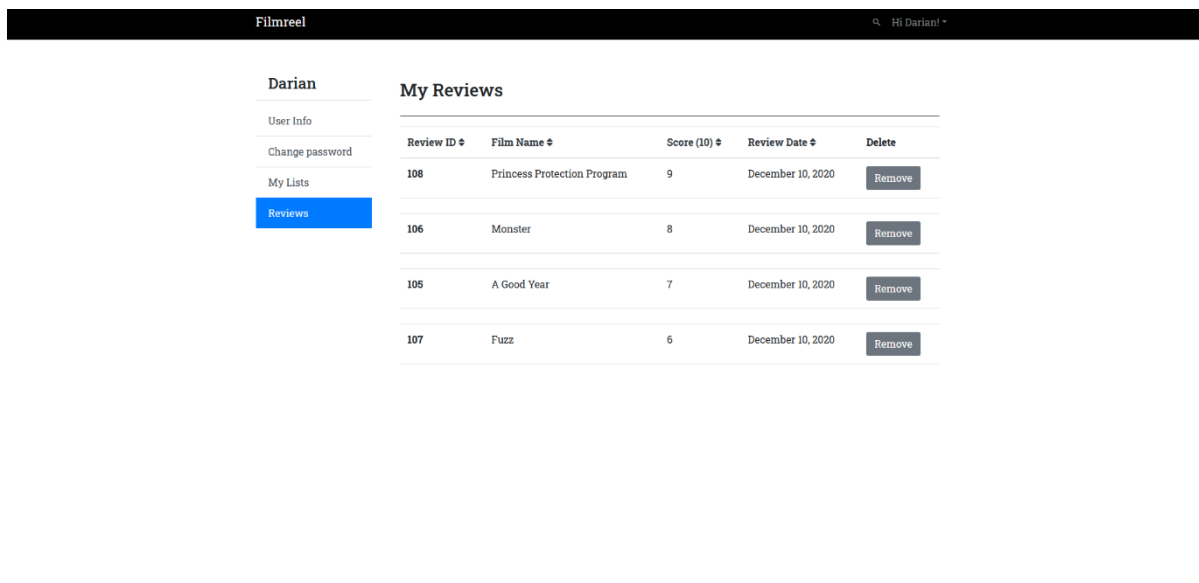
There is also a “Create List” button which will reveal a prompt to name the new list the user desires to create.



The last item on the user page is the “Reviews” tab which will show the reviews submitted by the user as well as the option to remove whichever review they choose.



Clicking on the arrows will sort the table in descending or ascending order based on the attribute beside the arrows.



The header is also different between users and guests: instead of the “Login” and “Register” buttons, there is a dropdown button which gives the option to open the user page or to log out.


The search page is also different from guests: there is now an “Add to List” dropdown button which will add the film to the selected list.

Filmreel

Hi Darian!

## Search Results

1 results were found for the search for 'lif'



### Life 2.0

📅 Release Date: December 2, 2018 ⚠️ Rated: PG-13

**Add To List**

asdf


For the Boys

cing lorem, vitae mattis nibh ligula nec sem. Duis aliquam convallis nunc. Proin at turpis a pede posuere nonummy. Integer non velit. Donec diam neque, vestibulum eget, vulputate ut, ultrices vel, augue. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec pharetra, magna vestibulum aliquet ultrices, erat tortor sollicitudin mi, sit amet lobortis sapien sapien non mi. Integer ac neque. Duis bibendum. Morbi non quam nec dui luctus rutrum. Nulla tellus. In sagittis dui vel nisl. D

Lastly, the film page will now also contain an interface to allow the user to write and submit a review.

Filmreel

Hi Darian!



ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec pharetra, magna vestibulum aliquet ultrices, erat tortor sollicitudin mi, sit amet lobortis sapien sapien non mi. Integer ac neque. Duis bibendum. Morbi non quam nec dui luctus rutrum. Nulla tellus. In sagittis dui vel nisl. Duis ac nibh. Fusce lacus purus, aliquet at, feugiat non, pretium quis, lectus. Suspendisse potenti. In eleifend quam a odio. In hac habitasse platea dictumst. Maecenas ut massa quis augue luctus tincidunt. Nulla mollis molestie lorem.

## Reviews for Life 2.0

### Leave a Review

Review Title

--Select List--

Share your thoughts on the film

Submit

### Robust

posted by tbarnwillh on September 12, 2019

Score: 9/10

They are also able to remove reviews they have written.

Filmreel

Hi Darian!

--Select List--

Share your thoughts on the film

Submit

it was more than good

posted by Darian on December 10, 2020

Score: 7/10

but not that good

Remove

## Administrators

Administrators have all the same functions as users except for list creation. The user page is now called an “Administrator Portal,” and they can remove any review they wish.

The Administrator Portal is essentially the same as a user page, but with an additional “Statistics” tab which displays user statistics detailing how a user uses the site. For instance, hattegart3 uses the site’s list and review features while jgerling2 seems to only be interested in writing film reviews.

User ID ^	Username ⇅	Age ⇅	Lists ⇅	Films per List ⇅	Reviews ⇅	Avg Score ⇅
1	flivsey0	18	1	7.00	5	6.60
2	mwoolham1	29	0		0	
3	jgerling2	22	0		4	6.50
4	hateggart3	24	3	4.00	8	5.25
5	cscough4	35	1	6.00	7	5.71
6	mcrunkhorn5	36	1	4.00	5	5.60
7	zquixley6	33	1	4.00	7	6.86
8	glerway7	28	0		2	3.00
9	awarland8	23	2	5.00	6	5.83
10	emoulster9	25	0		5	5.00
11	erosena	22	1	6.00	6	4.83
12	egrindlayb	20	0		1	4.00
13	bspinellac	34	1	5.00	3	6.33
14	amurnamed	37	0		7	5.86

The “Users” tab displays user information such as birth date.

User ID ▲	Username ◆	Email ◆	Date of Birth ◆	First Name ◆	Last Name ◆
1	flivsey0	flivsey0@blog.com	October 8, 2002	Fleming	Livsey
2	mwoolham1	mwoolham1@va.gov	December 26, 1990	Mariska	Woolham
3	jgerling2	jgerling2@over-blog.com	January 15, 1998	Jannelle	Gerling
4	hateggart3	hateggart3@reddit.com	May 6, 1996	Honorita	Ateggart
5	cscough4	cscough4@tiny.cc	June 29, 1985	Chen	Scough
6	mcrunkhorn5	mcrunkhorn5@soup.io	June 14, 1984	Michael	Crunkhorn
7	zquixley6	zquixley6@geocities.com	February 25, 1987	Zandra	Quixley
8	glerway7	glerway7@dropbox.com	March 13, 1992	Gerhardt	Lerway
9	awarland8	awarland8@illinois.edu	February 22, 1997	Annette	Warland
10	emoulster9	emoulster9@facebook.com	February 11, 1995	Ezra	Moulster
11	erosena	erosena@omniture.com	June 19, 1998	Elsi	Rosen
12	egrindlayb	egrindlayb@admin.ch	January 26, 2000	Erinka	Grindlay
13	bspinellac	bspinellac@kickstarter.com	February 20,	Bernie	Spinella



The “Films” tab displays all the films in the database and some information about each one.

Filmreel									
ADMIN									
clara14									
Admin Info									
Change password									
Statistics									
Users									
Films									
Films									
ID	Name	Release Date	Duration (minutes)	Rating	Reviews	Avg Score	Listings	Genre	
1	Faces of Schlock	October 5, 2008	176	R	6	7.17	7	Action, Comedy, Drama	
2	A Good Year	August 21, 2019	90	NC-17	7	6.14	7	Drama, Romance	
3	Long Way Round	January 10, 2013	179	G	9	6.11	9	Action, Adventure, Fantasy	
4	Cool It	December 16, 2015	128	PG-13	1	4.00	8	Documentary, Historical	
5	Fuzz	January 23, 2019	127	PG	4	3.50	5	Drama, Science, Western	
6	Princess Protection Program	February 27, 2012	147	G	6	6.17	4	Children, Horror	

Lastly, administrators can remove any review from the film page as shown below.

Filmreel

ADMIN

Focused

Remove

posted by erosena on October 13, 2020

Score: 4/10

eleifend pede libero quis orci nullam molestie

exuding

Remove

posted by tbarnwillh on August 4, 2020

Score: 4/10

lobortis est phasellus sit amet erat nulla tempus vivamus

Enterprise-wide

Remove

posted by mbloxame on May 3, 2020

Score: 9/10

id massa id nisl venenatis

instruction set

Remove

posted by cscough4 on March 19, 2020

Score: 2/10

dui nec nisi volutpat eleifend

Phased

Remove

### 9.1.3 – Demonstration of Programming Logic

#### Guests

Guests were only able to interface with the database in a very limited manner. As shown in the previous section, they had access to the index, search, login, film, and register pages.

The index page had two views which displayed “Recent Releases” and “Most Popular” films. The tables only showed the films’ image and title.

After guests enter an input to the search bar, the search page displayed view containing the table of films and all its attributes except the films’ film id and their duration.

The login page does not have any views, but it does execute a procedure which checks to see if the guest entered the correct login credentials stored in the database

The film page uses a view which displays all a film’s attributes onto the page. It also uses a view which shows reviews at the bottom.

Lastly, after the guest enters the appropriate information, a trigger is activated which increments the largest account id followed by a procedure which inserts a new user into a database based on the information the guest entered.

## Users

Other than not having a login or register page to access, users have and use the same procedures, functions, and views as guests.

The user page which users can access has various tabs which uses various views. The “User Info” tab uses a view which displays a user’s information.

The “My Lists” tab uses a view which displays a table of films within a list the user made.

Lastly, the “Reviews” tab uses a view which shows a table of all the reviews the user wrote.

An additional procedure which inserts a new film into a list is used in the search page as a user through the now available “Add to List” button.

There is also an additional procedure which inserts a new review which is used in the film page since users can also write reviews.

## Administrators

Other than not being able to create lists, Administrators have and use the same procedures, functions, and views as users.

The admin page which administrators can access has three new tabs. The “Statistics” tab uses a view which shows various user statistics.

The “Users” tab uses a view which displays all the users of the site and their information apart from users’ passwords.

The “Film” tab uses a view which displays all the films of the site and their attributes apart from the film’s image and synopsis. The view also includes the how many reviews a film has.

## 9.2 – GUI Programming

### 9.2.1 – Server-side Programming

The “film\_stats” view from section 8.3.1 was used to show administrators all the films from the database under the “Films” tab from the Administrator Portal

The “user\_stats” view from section 8.3.1 was used to show administrators user statistics under the “Statistics” tab from the Administrator Portal.

The “insert\_review” procedure from section 8.3.2 was used by users and administrators to allow both users and administrators to make and insert their reviews into the database.

The “remove\_review” procedure from section 8.3.2 was used by users and administrators to allow both users and administrators to remove reviews from the database. Users can only remove their own reviews. Administrators can remove any review.

### 9.2.2 – Middle-tier Programming

Code for database connection:

```
?php
$dbhost = 'localhost';
$dbname = 'filmreel';
$dbuser = 'root';
$dbpass = 'JeLoCe2713!';

function get_connection() {

    static $connection;

    if (!isset($connection)) {
        $connection = mysqli_connect('localhost','root','JeLoCe2713!','filmreel') or die(mysqli_connect_error());
    }
    if ($connection === false) {
        echo "Unable to connect to database<br>";
        echo mysqli_connect_error();
    }

    return $connection;
}

?>
```

Code example for displaying views:

```
$lists = $db->prepare("SELECT l.list_id, l.list_title FROM List l WHERE l.account_id=?");
$lists->bind_param('i', $userid);
if ($lists->execute()) {
    if (mysqli_stmt_bind_result($lists, $lid, $lname)) {
        while($lists->fetch()) {

echo <<<_RECENT
    <form method="post" action="addtoList.php">
        <input name="fid" id="fid" value="$fid" type="hidden">
        <input name="url" id="url" value="search.php?search=$search" type="hidden">
        <input name="lid" id="lid" value="$lid" type="hidden">
        <input type="submit" class="dropdown-item btn" name="Add" value="$lname">
    </form>
_RECENT;
        }
    }
    $lists->close();
} else {
    echo "Error executing query: " . mysqli_error($db);
    die();
}
```

Code example of calling stored procedures and functions:

```
if ($result_count > 0) {
    $_SESSION["error"] = "Error: User has already submitted a review for this film";
    header("Location: film.php?id=$fid");
} else {
    $insert = $db->prepare("call insert_review(?,?,?,?)");
    $insert->bind_param('iisis', $userid, $fid, $rtitle, $rscore, $rdesc);
    if ($insert->execute()) {
        echo "Review added!";
        header("Location: film.php?id=$fid");
    } else {
        echo "Error getting result: " . mysqli_error($db);
        header("Location: register.php");
        die();
    }
}
```

### 9.2.3 – Client-side Programming

JavaScript was used to allow users to organize various views in ascending or descending order based on an attribute on the table as shown in the “Users” section of 9.1.2. Here is how it was implemented:

First is the issue of organizing the data in ascending or descending order. This is handled server-side using PHP.

```

<?php
$columns = array('review_id', 'film_name', 'score', 'review_date');
$column = isset($_GET['column']) && in_array($_GET['column'], $columns) ? $_GET['column'] : $columns[0];
$sort_order = isset($_GET['order']) && strtolower($_GET['order']) == 'desc' ? 'DESC' : 'ASC';

$up_or_down = str_replace(array('ASC', 'DESC'), array('up', 'down'), $sort_order);
$asc_or_desc = $sort_order == 'ASC' ? 'desc' : 'asc';
?>

<div class="tab-pane fade" id="user-reviews">
<div class="card-body pb-2">
<div class="row mx-auto">
<h3>My Reviews</h3>
</div>
<hr class="border-top border-dark">
<table class="table">
<thead>
<tr>
<th scope="col">
<a class="text-dark" href="myaccount.php?column=review_id&order=<?php echo $asc_or_desc; ?>">Review ID
<i class="fas fa-sort<?php echo $column == 'account_id' ? '-' : $up_or_down : '' ; ?>"></i></a></th>
<th scope="col">
<a class="text-dark" href="myaccount.php?column=film_name&order=<?php echo $asc_or_desc; ?>">Film Name
<i class="fas fa-sort<?php echo $column == 'username' ? '-' : $up_or_down : '' ; ?>"></i></a></th>
<th scope="col">
<a class="text-dark" href="myaccount.php?column=score&order=<?php echo $asc_or_desc; ?>">Score (10)
<i class="fas fa-sort<?php echo $column == 'email' ? '-' : $up_or_down : '' ; ?>"></i></a></th>
<th scope="col">
<a class="text-dark" href="myaccount.php?column=review_date&order=<?php echo $asc_or_desc; ?>">Review Date
<i class="fas fa-sort<?php echo $column == 'birth_date' ? '-' : $up_or_down : '' ; ?>"></i></a></th>
<th scope="col">Delete</th>
</tr>
</thead>
<tbody>

<?php
$db = get_connection();
$rinfo = $db->prepare("SELECT r.review_id, f.film_name, r.score, r.review_date, r.title, r.description FROM Review r
INNER JOIN film_stats f ON r.film_id = f.film_id WHERE r.account_id=$userid ORDER BY " . $column . " " . $sort_order);
if ($rinfo->execute()) {
    if (mysqli_stmt_bind_result($rinfo, $rid, $filname, $rscore, $rdate, $rtitle, $rdesc)) {
        while ($rinfo->fetch()) {
            $longdate = date('F j, Y', strtotime($rdate));
echo <<<_TROWS
<tr class="accordian-toggle collapsed" id="accordian1" data-toggle="collapse" data-parent="#accordian1" href="#collapse$rid">
<th scope="row">$rid</th>
<td>$filname</td>
<td>$rscore</td>
<td>$longdate</td>
<td>
<form method="post" action="removeReview.php" class="ml-auto">
<input name="rid" id="rid" value="$rid" type="hidden">
<input name="url" id="url" value="myaccount.php" type="hidden">
<input type="submit" class="btn btn-secondary ml-auto" name="Remove" value="Remove">
</form>
</td>
</tr>

```

240,1-4

79%

Then a script is used to ensure that the page reloads to the tab the user or administrator was viewing. Otherwise, the user or administrator would be reloaded back to the “User Info” or “Admin Info” tab.

```
script>
    $('a[data-toggle="tab"]').on('show.bs.tab', function(e) {
        localStorage.setItem('activeTab', $(e.target).attr('href'));
    });

    var activeTab = localStorage.getItem('activeTab');
    if(activeTab){
        $('.list-group a[href="' + activeTab + '"]').tab('show');
    }
/script>
</script>
```