Outline for Final Project
Madeline Rosenberg

I made a database around a movie information dataset that included:

1. Movie title
2. Year of release
3. OMDb rating
4. Rotten Tomato rating
5. Whether a movie was on Netflix (1 or 0)
6. Whether a movie was on Prime (1 or 0)
7. Whether a movie was on Hulu (1 or 0)
8. Whether a movie was on Disney Plus (1 or 0)
9. Genre
10. Age Category (the intended audience)
11. Country of origin
12. Director first name (contained string formatted *first *space* last*)
13. Director last name (empty)
14. Available language(s) (comma separated listing of languages)
15. Runtime

I first normalized the database by splitting director names into first and last names. I did this because it satisfies the First Normal Form normalization principal that columns must contain values of atomic data types. I then put this information into a director table that contained a director primary key identifier, and distinct director first and last names. This will help reduce redundancy of string type values and will make it easier in the future to add new movie directors to the data set. For those general reasons, I created the age, country, and genre tables. All of those tables contain string values that repeat throughout the data set. I anticipate as movies come into the streaming platforms, including films with diverse places of origin, that information will need to be updated. This may include an extra age category or country for example. It is easier to reference and update these smaller normalized tables as the new movie information comes in.

I created the language table partly to reduce redundancy of string types, but also for more complicated normalization needs for language compared to previously mentioned tables. Languages were comma separated for each movie. If I included those data points as is in the table, it would make it very difficult to search for which movies are in English for example, because each English movie had English in different locations of the list. There would be lots of repetition of information. So to handle this, I first needed to split the locations by string. Unfortunately, MySQL does not have a string splitting function. This meant I had to use a creative process to normalize (in ONLY MySQL).

First, I created a temporary table called numbers. I included 1 through 15 in this temporary table (meaning a list of languages could have up to 15, which would be a large number of languages for a movie). I used a math equation using this temporary numbers table to take the last item out of a given list before the comma and keep going until the list had no more items. I

inserted the split-up data into a table called temp_lang which contained distinct language names, a primary key identifier, and the main table, film_info, identifier. Temp_lang made it easier for me to connect language to the main table and a connector table called link_lang_film without mistakes. I transferred the language name into temp_lang and populated the link_lang_film table with the language table's identifier (as a foreign key) and the film info identifier (as another foreign key) from temp_lang. I then dropped temp_lang.

Link_lang_film as mentioned is a connector between language and film info. This reduces the redundancy in the large main table since each film has multiple languages. Instead of repeating movie titles, year, etc. multiple times, just a couple numeric identifiers are repeated in the connector table.

Finally, film_info is the main table. At first, it contained all the original values (such as country as a string and language as a string list). After I created the other smaller tables, I added genre_id, director_id, age_id, and country_id as foreign key identifiers. In support of the Second Normal Form principal, I got rid of the multi-column keys by dropping the string values that the identifiers represented. Technically the platforms had repeating information. The years can technically repeat, but this information does not need its own table since it is an integer that takes up a relatively small amount of space and it would be more hassle than it is worth to make someone join year identifiers to get that basic movie information.

space: indicates a space sized separation between the words

Code for Creating Tables
Madeline Rosenberg

Note: comments are indicated with hashtags

```
create database movie;
use movie;
create table film_info
(
        id int auto_increment,
        name varchar(255) not null,
        age_category varchar(5) not null,
        imdb float not null,
        rotten_tomato int not null,
        netflix enum('1', '0') not null,
        hulu enum('1', '0') not null,
        prime enum('1', '0') not null,
        disney_plus enum('1', '0') not null,
        fname_director varchar(255) not null,
        lname_director varchar(255) not null,
        genre varchar(255) not null,
        country varchar(255) not null,
        language varchar(255) not null,
        runtime int not null,
        constraint film_info_pk
                primary key (id)
);


alter table film_info modify lname_director varchar(255) null;

 alter table film_info
        add year int null;




create table country
(
        id int auto_increment,
        name varchar(255) not null,
        constraint country_pk
                primary key (id)
);

create unique index country_name_uindex
        on country (name);
```

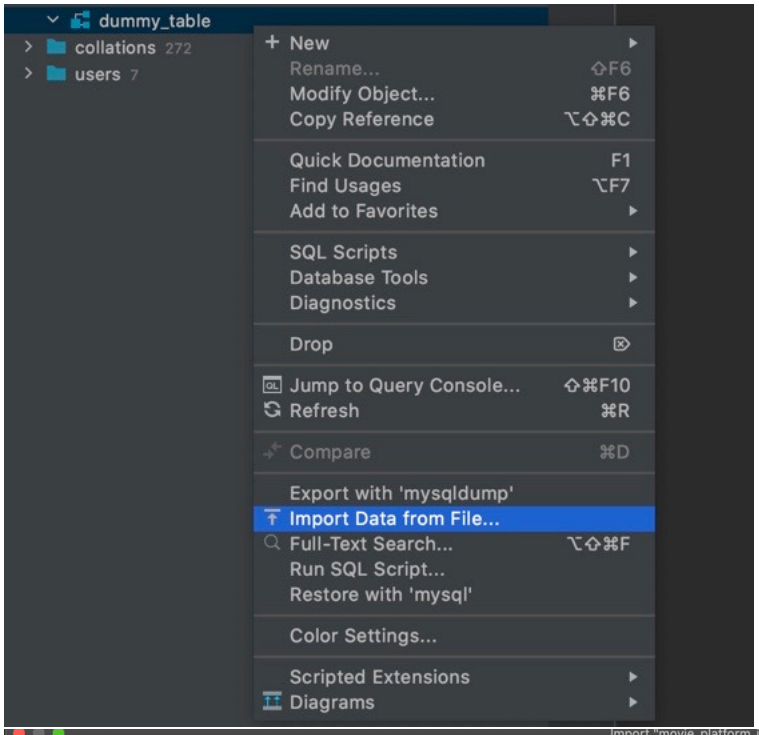
```
create table age
(
	id int auto_increment,
	category varchar(5) not null,
	constraint age_pk
		primary key (id)
);

create unique index age_category_uindex
	on age (category);

alter table age change category name varchar(5) not null;


create table genre
(
	id int auto_increment,
	name varchar(255) not null,
	constraint genre_pk
		primary key (id)
);

create unique index genre_name_uindex
	on genre (name);

create table lang
(
	id int auto_increment,
	name varchar(255) not null,
	constraint lang_pk
		primary key (id)
);

create unique index lang_name_uindex
	on lang (name);


create table link_lang_film
(
	id int auto_increment,
	film_id int not null,
	lang_id int not null,
	constraint link_lang_film_pk
		primary key (id)
);

create table director
(
```

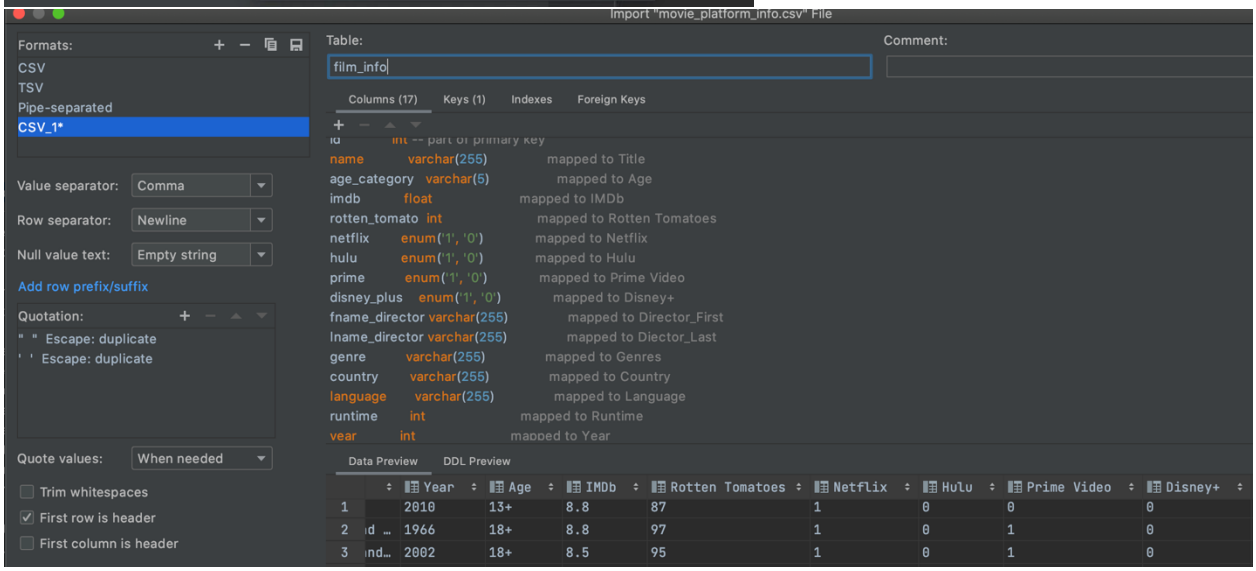
```
	id int auto_increment,
	fname varchar(255) not null,
	lname varchar(255) not null,
	constraint director_pk
		primary key (id)
);
```

########Since the table already has data, I do not believe director_id should be initiialized in the database excluding null values

```
alter table film_info
	add director_id int null;


alter table film_info modify age_category varchar(5) null;

alter table film_info modify imdb float null;

alter table film_info modify rotten_tomato int null;

alter table film_info modify netflix enum('1', '0') null;

alter table film_info modify hulu enum('1', '0') null;

alter table film_info modify prime enum('1', '0') null;

alter table film_info modify disney_plus enum('1', '0') null;

alter table film_info modify fname_director varchar(255) null;

alter table film_info modify genre varchar(255) null;

alter table film_info modify country varchar(255) null;

alter table film_info modify language varchar(255) null;

alter table film_info modify runtime int null;

alter table film_info modify name varchar(255) null;
```

######Import Data to film info





DELETE FROM movie.film_info WHERE id = 1;

####EXAMPLE OF INSERTING VALUE

INSERT INTO film_info
(id,name,age_category,imdb,rotten_tomato,netflix,hulu,prime,disney_plus,fname_director,lname_director,genre,country,language,runtime,year) VALUES
(1,'Inception','13+',8.8,67,'1','0','0','0','Christopher Nolan','','Action','United States','English,Japanese,French',148,2010);

```
#director lastname
UPDATE film_info SET lname_director = SUBSTRING_INDEX(fname_director, ' ', -1);
```

fname_director	lname_dire... 1
Harald Zwart	Zwart
David Zucker	Zucker
Mark Williams	Williams
David Ward	Ward
Rupert Wainwright	Wainwright
Denis Villeneuve	Villeneuve
Hèctor Hernández Vicens	Vicens

```
#director firstname
UPDATE film_info SET fname_director = SUBSTRING_INDEX(fname_director, ' ', 1);


INSERT INTO director (fname, lname) SELECT DISTINCT fname_director, lname_director
FROM film_info;


UPDATE film_info fi  SET director_id = (SELECT d.id FROM director d WHERE
fi.fname_director = d.fname AND fi.lname_director = d.lname);


alter table film_info
        add age_id int null;


INSERT INTO age (name) SELECT DISTINCT age_category FROM film_info;


UPDATE film_info fi  SET age_id = (SELECT a.id FROM age a WHERE a.name =
fi.age_category);

INSERT INTO country (name) SELECT DISTINCT country FROM film_info;


alter table film_info
        add country_id int null;
```

```
UPDATE film_info fi  SET country_id = (SELECT c.id FROM country c WHERE c.name =
fi.country);


INSERT INTO genre (name) SELECT DISTINCT genre FROM film_info;

alter table film_info
        add genre_id int null;


UPDATE film_info fi  SET genre_id = (SELECT g.id FROM genre g WHERE g.name = fi.genre);




create temporary table numbers as (
  select 1 as n
  union select 2 as n
  union select 3 as n
  union select 4 as n
  union select 5 as n
  union select 6 as n
  union select 7 as n
  union select 8 as n
  union select 9 as n
  union select 10 as n
  union select 11 as n
  union select 12 as n
  union select 13 as n
  union select 14 as n
  union select 15 as n
);


create table temp_lang
(
        id int auto_increment,
        lang varchar(255) not null,
        num int not null,
        constraint temp_lang_pk
                primary key (id)
);
```

####### Takes the film info table id of each movie and language split by commas (each one n and each subsequent one after that) with a math equation. The equation takes language, gets a substring at the nth index, and finds all the list after that comma. Then, -1 grabs everything to the left of that comma. Each language is split up and every language with the same film is given the same id!

```
INSERT INTO temp_lang (num, lang)
```

```
SELECT id, substring_index(substring_index(language, ',', n),  ',', -1)
from film_info
join numbers
  on char_length(language)
    - char_length(replace(language, ',', ''))
    >= n - 1;

####### populating language table with unique languages from temp_lang table
INSERT INTO lang (name) SELECT DISTINCT lang FROM temp_lang;

####### populating link_lang_film table with language id's and num from temp_lang by joining
on the name from language and temp_lang tables : )
#set the film_id as num and lang_id as id
INSERT INTO link_lang_film (film_id, lang_id) SELECT tl.num AS film_id, l.id AS lang_id FROM
temp_lang tl JOIN lang l ON tl.lang = l.name;


######Drop temp_lang table
drop table temp_lang;



alter table film_info drop column age_category;

alter table film_info drop column fname_director;

alter table film_info drop column lname_director;

alter table film_info drop column genre;

alter table film_info drop column country;

alter table film_info drop column language;

alter table film_info modify name varchar(255) not null;

alter table film_info modify imdb float not null;

alter table film_info modify year int not null;

alter table film_info modify runtime int not null;

alter table film_info modify age_id int not null;

alter table film_info modify country_id int not null;

alter table film_info modify genre_id int not null;

alter table film_info modify director_id int not null;
```

```
alter table link_lang_film
        add constraint link_lang_film_film_info_id_fk
                foreign key (film_id) references film_info (id);

alter table link_lang_film
        add constraint link_lang_film_lang_id_fk
                foreign key (lang_id) references lang (id);

alter table film_info
        add constraint film_info_age_id_fk
                foreign key (age_id) references age (id);

alter table film_info
        add constraint film_info_country_id_fk
                foreign key (country_id) references country (id);

alter table film_info
        add constraint film_info_director_id_fk
                foreign key (director_id) references director (id);

alter table film_info
        add constraint film_info_genre_id_fk
                foreign key (genre_id) references genre (id);
```