Project 1 - Database - Design Document CSCE 315 - 903

Sandesh Neupane
Shane Poldervaart
Austin Offill

Phase 1

Section 1

We are solving the problem of looking up movies through information about the movies. We also want to get movies with similar attributes and display them to the user. This system will grant customers with easy access to information about movies and will help them find movies related to what they are looking up. It also provides a way for customers to find movies based on actors, genres, director, country, release date, runtime, rating, etc. This system will be used by everyday people, movie buffs, and anyone wanting information on movies. We want to provide them with the best system for finding movies in order to keep them using our product.

Section 2

First, we have the movie table. It takes information that is relevant to the movie. This way we have everything about a movie in one place. Production country is our second list and it will store the country name and abbreviation. Next, we have the movie company trailer which stores how it relates to the movie and company list. After that, we have the series table that stores the name of the series. The genre table will have similar data to the series table, but it's the genre name. The Language table stores the language that the movie is in. The movie language table stores how the language table relates to the movie table. The last table for movies will be the movie genre trailer table which stores how the movie table relates to the genre table. We need tables that store information about all the specified lists and tables for how these relate to each other. The movie list will serve as the main list. The other lists then have a list telling them how they relate to the main list. This is done using item ids to keep track of where in lists things are.

Section 3

3.1

MOVIE:-

The primary makeup of the overall movie table uses a GUID as there are lots of them. Utilizing this with the movie country, movie company, movie genre, movie language, and the series tables will gather all possible general information about a movie. This entity exists because it is a physical thing to relate to, and where a majority of the information resides. The benefit is that it makes up most of the data and holds relevant movie information. The main risk is that, if you want all the information about a movie, you need to utilize multiple tables.

PRODUCTION COUNTRIES:-

The relationship with the production country is the list of all countries ' names and abbreviations where films were produced. This entity is utilized in movie country table as it's a many to many relationship. The main benefit of this table is that it can have a list of movies made in countries and there is no risk of having it.

MOVIE COUNTRY:-

The relationship with the movie country table is a list of all countries and movies together as two foreign keys per object that indicates a country where the movie was filmed. The reason to come up with this table and relation with the production country is that one country can have many movies and one movie can have many countries. The risk of having movie country is that it needs to be called to have the full movie information

PRODUCTION COMPANY:-

The relationship of production company is the list of all production companies associated with movies with their names. The main reasoning behind this is that it is utilized in a movie company as it is a many to many relationship. The benefits are that it can have a list of production companies with the movies they have made and vice versa.

MOVIE COMPANY:-

The relationship of the movie company is the list of all production companies and movies together as two foreign keys per object that indicates a company that is associated with a movie. The reason this entity exists is that many to many relationships require this. As one production company can have many movies and one movie can have many production companies. The risk with having is that it needs to include in the call for full movie info.

SERIES:-

The main relationship between series and movie tables is that one movie can only belong to a single series. Thoughts process involved for the creation of the entity is because movie series is a concrete concept. The main benefit is that it is single link between all the movies in a series. There is no risk associated with it.

GENRES:-

This table is a list of all possible genres according to IMDB. This is more than just a string because it helps define a movie. The main relation is that movies can have multiple genres. The main benefit of this table is that it can have a list of genres and movies that are associated with each other.

MOVIE GENRE:-

The relationship with the movie genre table is a list of all genres with movies they are associated with two foreign keys. This table is necessary because this is a many to many relationships with movies because a movie can have many genres. The risk associated with this table is that it needs to be included to get the full movie information

LANGUAGES:-

The relationship listed in this table is all languages, names, and abbreviations. Languages are more than a simple string aspect of a movie. The benefit of this table is that we can easily search for movies only by languages and have a movie with a list of its languages. There is no risk associated with this table.

MOVIE LANGUAGE:-

The relationship of this table is a list of all languages and movies associated with two foreign keys. It is necessary because it is a many to many relationship. Many movies can be in a language, and one movie can be in multiple languages. The benefit of this table is that we can search for different movies of the same language. The risk associated with this table is that it needs to be included to get full movie information.

CREDITS:-

The relationship in this table is corresponding credits of a movie with the id that links the credits together. The benefit of this table is that it links together cast and crew who worked on movie then link to that movie. This is used as the overarching idea to compile the credits and is a concrete idea. There are no risks associated with this table.

PEOPLE:-

The relationship of this table is a list of all people associated with the movie world. This is used to compile people in the credits with a specifier for whether they are an actor or a crew member. The benefit of this table is that it links actors to multiple films and directors/other workers to their respective works. The reasoning behind this is that people are a concrete thing, and they work on films so we store them as one entity. Risk associated with this is there is no way to discern people of the same name, and also has two different people for someone who acts and also a crew for a film to help minify the first mentioned risk. An additional risk associated with this is that the table will be very large.

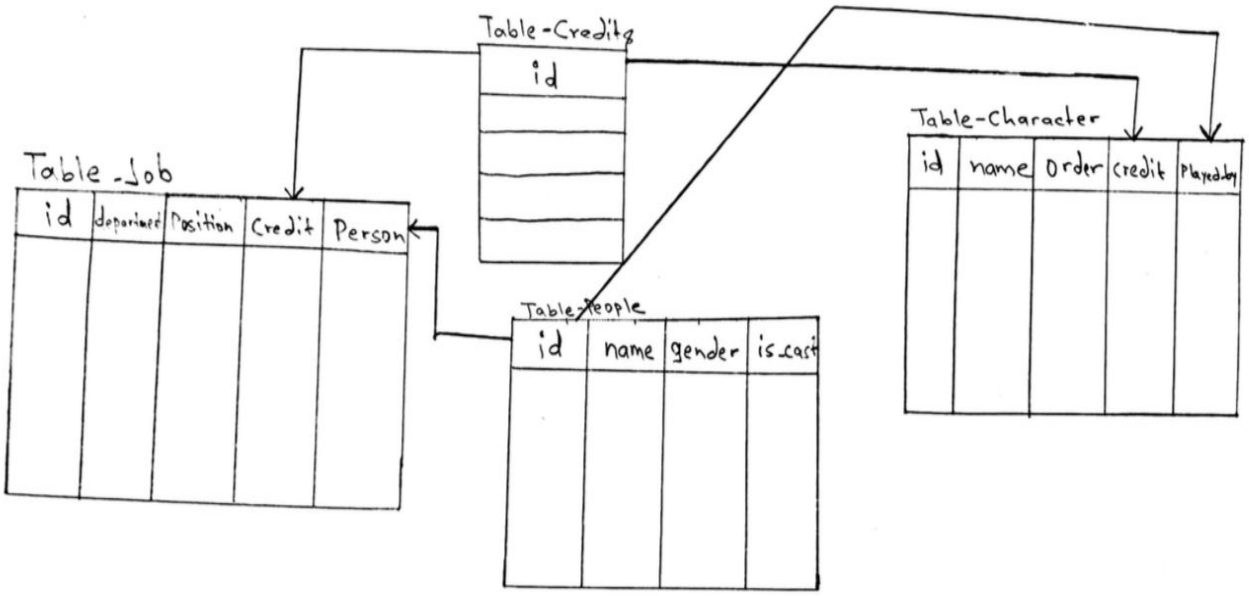
CHARACTERS: -

The relationship of this table is that a character can be in all movies. The stories, name, gender, order, what credit they belong to, and who plays them is very important. The benefit of this table is that we can get a list of characters in a movie after giving credit. We need a list of characters for movies and who played them. The risk associated is the size, which is going to belong.

JOBS:-

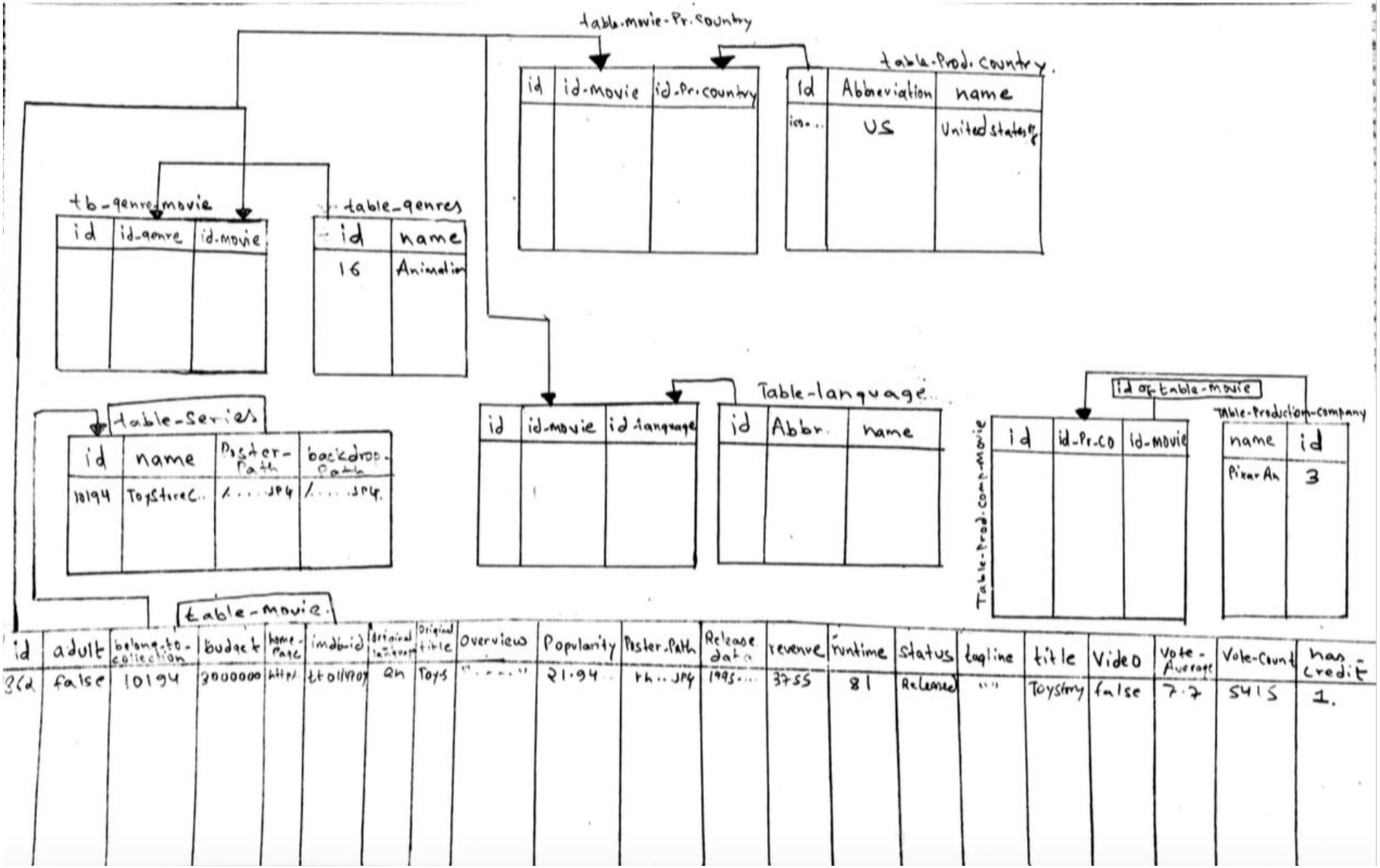
The relation of this table is a list of all jobs in every movie as a crew member. This table contains the department, position, and a link to the respective movie and the person who did that job. It is necessary to store this information for generating the staff that worked on a movie. The benefit of this table is that we can have a list of everyone who worked on a movie. Risk associated is size, as it is going to be a very large table due to it being compiled from the credits of every single movie.

3.2

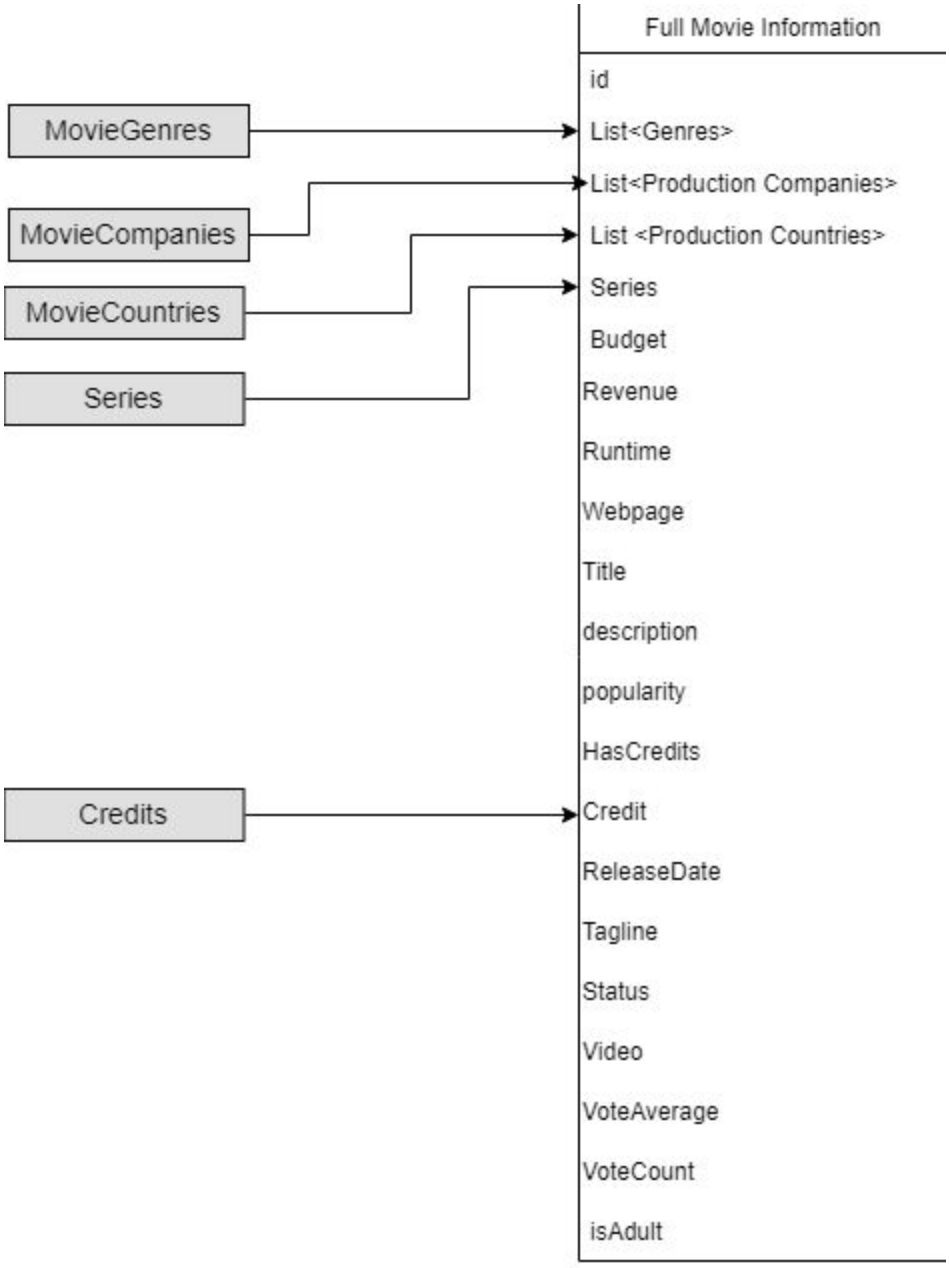


This diagram shows the breakdown of the credits section of the database. Here it visualizes the 4 tables associated. The credits, jobs, people, and characters table and how they all involve the credit id.

The next diagram shows the layout of the movie section of the database. This is made up of the 10 tables, the movie, genre, movie genre, production country, movie country, language, movie language, production company, movie company, and series. You can see in the compound tables (movie language, movie genre, movie company, movie country), how they are just made up of movies and their respective second piece.


table_movie_Pr_country
id
id_movie
id_Pr_country
table_Prod_country
id
Abbreviation
name
US
United states
tb_genre_movie
id
id_genre
id_movie
table_genres
id
name
16
Animation
table_Series
id
name
Poster_Path
backdrop_Path
10194
ToyStoreC...
/....JPG
/....JPG
Table_language
id
id_movie
id_language
id
Abbr.
name
id of table_movie
Table_Prod_com_Movie
id
id_Pr_CO
id_movie
Table_Production_company
name
id
Pixar An
3
table_movie
id
adult
belong_to_collection
budget
home_page
imdb_id
Original language
Original title
Overview
Popularity
Poster_Path
Release data
revenue
runtime
status
tagline
title
Video
Vote_Average
Vote_Count
has_credit
862
false
10194
3000000
http
tt0114709
en
Toys
"....."
21.94...
/h...JPG
1995...
3755
81
Released
""
Toystory
false
7.7
5415
1.

3.3



This Diagram shows what makes up the full movie information. There is a multitude of tables that need to be pulled from in order to generate this, and those tables themselves have underlying pieces that are seen in the diagrams above.

Section 4

Since we split up the movie into a cumulation of multiple tables to allow queries to be able to be more specific. This also allows searches for movies to be less taxing on the server. We will also have nullable entries such as indie films which will have no budget and also make no revenue. Unconfirmed release dates, movies without ratings, and standalone movies will also be nullable because of the missing information presented. We can query for specific things such as movies made in countries, or movies made by companies without getting the entire movies list or any irrelevant information to the search. The cast and crew both pull from people's table because some people both act and direct in the same movie. Using a single credits table as a way to link characters and jobs to the credits as a way to simplify the problem.

We are assuming that users are not going to be searching by image URLs based on IMDb posters and backdrops. Any time an ID will be used for credits it will be generated by our database for actors and crew members. Since we are creating our own unique IDs for movies and other entities, this will prevent users from searching by the IDs based on the IMDB database.

One of the big risks we run is people with the same names. The system will assume that people with the same name are the same person. This goes for actors, crew members directors, and anyone else apart from the movie. If we had more in-depth data we could find ways of avoiding it but as of right now we can't.

Our biggest issue is that our data doesn't contain enough information to make needed differentiations. We are getting around this marginally by utilizing a flag on the people table to say whether a person is a cast or crew member; splitting the amount of potential conflicts in half.

Phase 2

Changelog

9/28/19
All ints are instead of type serial, with the exception of characters, jobs, which are “bigserial”. Done as serial auto increments and is large enough to hold everything besides the 2 mentioned previously. Made concrete lengths of all varchars
Changed movie and people keys to be tconsts and nconsts respectively. char(9)’s to make inputting easier
Dropped credits table as not necessary anymore & made characters and jobs just directly reference the movie itself instead of a second credit id.
Dropped id primary key on the linking tables, making the primary key the movieid and other object it’s linking to
Moved ratings to their own tables
Created a title types table and a corresponding linking table

10/1/19
Changed all mentions of dates to be of type integer as only years are there.
Changed all tconsts and nconsts to varchar(16) as the length of the identifier is variable
Dropped productioncompany and associated linking table as those are not available in the new data set

Outputs

Relations:

```
shanepoldervaart_db=> \dt team.*
                  List of relations
 Schema |         Name          | Type  |      Owner
--------+-----------------------+-------+------------------
 team   | characters            | table | shanepoldervaart
 team   | genre                 | table | shanepoldervaart
 team   | jobs                  | table | shanepoldervaart
 team   | languages             | table | shanepoldervaart
 team   | moviecountry          | table | shanepoldervaart
 team   | moviegenre            | table | shanepoldervaart
 team   | movielanguage         | table | shanepoldervaart
 team   | movies                | table | shanepoldervaart
 team   | movietype             | table | shanepoldervaart
 team   | people                | table | shanepoldervaart
 team   | productioncountries   | table | shanepoldervaart
 team   | ratings               | table | shanepoldervaart
 team   | titletypes            | table | shanepoldervaart
(13 rows)
```

```
shanepoldervaart_db=> \d team.movies
                                  Table "team.movies"
    Column     |         Type          |                          Modifiers
---------------+-----------------------+-------------------------------------------------------------
 id            | character varying(16) | not null default nextval('team.movies_id_seq'::regclass)
 adult         | boolean               | not null
 budget        | integer               |
 revenue       | integer               |
 runtime       | integer               | not null
 webpage       | character varying(64) |
 originaltitle | character varying(250) | not null
 title         | character varying(250) | not null
 description   | character varying(512) |
 popularity    | double precision      |
 hascredits    | boolean               | default false
 credits       | integer               | not null default nextval('team.movies_credits_seq'::regclass)
 startyear     | integer               | not null
 tagline       | character varying(64) |
 status        | character varying(16) |
 titletype     | character varying(16) | not null
Indexes:
    "movies_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "team.characters" CONSTRAINT "characters_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
    TABLE "team.jobs" CONSTRAINT "jobs_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
    TABLE "team.moviecompany" CONSTRAINT "moviecompany_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
    TABLE "team.moviegenre" CONSTRAINT "moviegenre_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
    TABLE "team.movielanguage" CONSTRAINT "movielanguage_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
    TABLE "team.movietype" CONSTRAINT "movietype_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
    TABLE "team.ratings" CONSTRAINT "ratings_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
```

```
shanepoldervaart_db=> \d team.genre
                                Table "team.genre"
 Column |         Type          |                       Modifiers
--------+-----------------------+-------------------------------------------------------
 id     | integer               | not null default nextval('team.genre_id_seq'::regclass)
 name   | character varying(32) |
Indexes:
    "genre_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "team.moviegenre" CONSTRAINT "moviegenre_genreid_fkey" FOREIGN KEY (genreid) REFERENCES team.genre(id)
```

```
shanepoldervaart_db=> \d team.languages
                              Table "team.languages"
 Column |         Type          |                        Modifiers
--------+-----------------------+-----------------------------------------------------------
 id     | integer               | not null default nextval('team.languages_id_seq'::regclass)
 name   | character varying(32) |
Indexes:
    "languages_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "team.movielanguage" CONSTRAINT "movielanguage_languageid_fkey" FOREIGN KEY (languageid) REFERENCES team.languages(id)
```

```
shanepoldervaart_db=> \d team.titletypes
                                Table "team.titletypes"
 Column |         Type          |                         Modifiers
--------+-----------------------+-------------------------------------------------------------------
 id     | integer               | not null default nextval('team.titletypes_id_seq'::regclass)
 name   | character varying(16) |
Indexes:
    "titletypes_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "team.movietype" CONSTRAINT "movietype_typeid_fkey" FOREIGN KEY (typeid) REFERENCES team.titletypes(id)
```

```
shanepoldervaart_db=> \d team.ratings
              Table "team.ratings"
  Column   |         Type          | Modifiers
-----------+-----------------------+-----------
 movieid   | character varying(16) | not null
 avgrating | double precision      |
 numvotes  | integer               | default 0
Indexes:
    "ratings_pkey" PRIMARY KEY, btree (movieid)
Foreign-key constraints:
    "ratings_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
```

```
shanepoldervaart_db=> \d team.moviecountry
           Table "team.moviecountry"
  Column   |         Type          | Modifiers
-----------+-----------------------+-----------
 movieid   | character varying(16) | not null
 countryid | integer               | not null
Indexes:
    "moviecountry_pkey" PRIMARY KEY, btree (movieid, countryid)
Foreign-key constraints:
    "moviecountry_countryid_fkey" FOREIGN KEY (countryid) REFERENCES team.productioncountries(id)
    "moviecountry_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
```

```
shanepoldervaart_db=> \d team.moviegenre
           Table "team.moviegenre"
 Column  |         Type          | Modifiers
---------+-----------------------+-----------
 movieid | character varying(16) | not null
 genreid | integer               | not null
Indexes:
    "moviegenre_pkey" PRIMARY KEY, btree (movieid, genreid)
Foreign-key constraints:
    "moviegenre_genreid_fkey" FOREIGN KEY (genreid) REFERENCES team.genre(id)
    "moviegenre_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
```

```
shanepoldervaart_db=> \d team.movielanguage
          Table "team.movielanguage"
   Column   |         Type          | Modifiers
------------+-----------------------+-----------
 movieid    | character varying(16) | not null
 languageid | integer               | not null
Indexes:
    "movielanguage_pkey" PRIMARY KEY, btree (movieid, languageid)
Foreign-key constraints:
    "movielanguage_languageid_fkey" FOREIGN KEY (languageid) REFERENCES team.languages(id)
    "movielanguage_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
```

```
shanepoldervaart_db=> \d team.movietype
           Table "team.movietype"
 Column  |         Type          | Modifiers
---------+-----------------------+-----------
 movieid | character varying(16) | not null
 typeid  | integer               | not null
Indexes:
    "movietype_pkey" PRIMARY KEY, btree (movieid, typeid)
Foreign-key constraints:
    "movietype_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
    "movietype_typeid_fkey" FOREIGN KEY (typeid) REFERENCES team.titletypes(id)
```

```
shanepoldervaart_db=> \d team.people
            Table "team.people"
  Column   |          Type          | Modifiers
-----------+------------------------+-----------
 name      | character varying(128) | not null
 gender    | integer                | default 0
 id        | character varying(16)  | not null
 birthdate | integer                |
 deathdate | integer                |
Indexes:
    "people_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "team.characters" CONSTRAINT "characters_personid_fkey" FOREIGN KEY (personid) REFERENCES team.people(id)
    TABLE "team.jobs" CONSTRAINT "jobs_personid_fkey" FOREIGN KEY (personid) REFERENCES team.people(id)
```

```
shanepoldervaart_db=> \d team.characters
                                        Table "team.characters"
   Column    |          Type          |                            Modifiers
-------------+------------------------+-------------------------------------------------------------------
 id          | bigint                 | not null default nextval('team.characters_id_seq'::regclass)
 name        | character varying(128) | not null
 gender      | integer                | default 0
 orderappear | integer                |
 personid    | character varying(16)  | not null default nextval('team.characters_personid_seq'::regclass)
 movieid     | character varying(16)  | not null
Indexes:
    "characters_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "characters_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
    "characters_personid_fkey" FOREIGN KEY (personid) REFERENCES team.people(id)
```

```
shanepoldervaart_db=> \d team.jobs
                                      Table "team.jobs"
   Column   |          Type          |                        Modifiers
------------+------------------------+---------------------------------------------------------
 id         | bigint                 | not null default nextval('team.jobs_id_seq'::regclass)
 department | character varying(32)  |
 position   | character varying(32)  |
 movieid    | character varying(16)  | not null
 personid   | character varying(16)  | not null
Indexes:
    "jobs_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "jobs_movieid_fkey" FOREIGN KEY (movieid) REFERENCES team.movies(id)
    "jobs_personid_fkey" FOREIGN KEY (personid) REFERENCES team.people(id)
```

Currently the only table fully populated with entries is the people table. Movies and ratings are both partially populated. This is due to the fact that we had immense trouble connecting to the database, in addition to having trouble with permission granting on the server. Despite granting all permissions multiple times, we still had to use the admin account to log into the server or it was impossible to make changes to columns in the database. These changes were necessary as we discovered the imdb datasets to be much more incomplete than expected, with nearly every entity type potentially having entirely null entries besides a single table. These all compounded to have us unable to complete the task on time, but are currently working through parsing the other datasets and amending tables to fit the task. Also of note, is that imdb considers the same movie translated into different languages as different movies, but still keep the associated ratings for said movie.

Team.people

```
shanepoldervaart_db=> select * from team.people limit 10
;
      name        | gender |     id     | birthdate | deathdate
------------------+--------+------------+-----------+-----------
 Fred Astaire     |      0 | nm0000001 |      1899 |      1987
 Lauren Bacall    |      0 | nm0000002 |      1924 |      2014
 Brigitte Bardot  |      0 | nm0000003 |      1934 |
 John Belushi     |      0 | nm0000004 |      1949 |      1982
 Ingmar Bergman   |      0 | nm0000005 |      1918 |      2007
 Ingrid Bergman   |      0 | nm0000006 |      1915 |      1982
 Humphrey Bogart  |      0 | nm0000007 |      1899 |      1957
 Marlon Brando    |      0 | nm0000008 |      1924 |      2004
 Richard Burton   |      0 | nm0000009 |      1925 |      1984
 James Cagney     |      0 | nm0000010 |      1899 |      1986
(10 rows)
```

team.movies

```
    id     | adult | budget | revenue | runtime | webpage |               originaltitle                |
          title           | description | popularity | startyear | tagline | status | titletype
-----------+-------+--------+---------+---------+---------+--------------------------------------------+
-----------------------------+-------------+------------+-----------+---------+--------+-----------
 nm0000001 |     0 |        |         |       1 |         | Carmencita                                 |
 Carmencita - spanyol tánc |             |            |      1894 |         |        |
 nm0000002 |     0 |        |         |       1 |         | Carmencita                                 |
 Καρμενσίτα                |             |            |      1894 |         |        |
 nm0000003 |     0 |        |         |       5 |         | Le clown et ses chiens                     |
 Карменсита                |             |            |      1892 |         |        |
 nm0000004 |     0 |        |         |       5 |         | Le clown et ses chiens                     |
 Carmencita                |             |            |      1892 |         |        |
 nm0000005 |     0 |        |         |       4 |         | Pauvre Pierrot                             |
 Carmencita                |             |            |      1892 |         |        |
 nm0000006 |     0 |        |         |       0 |         | Un bon bock                                |
 Le clown et ses chiens    |             |            |      1892 |         |        |
 nm0000007 |     0 |        |         |       1 |         | Blacksmith Scene                           |
 A bohóc és kutyái         |             |            |      1893 |         |        |
 nm0000008 |     0 |        |         |       1 |         | Blacksmith Scene                           |
 Le clown et ses chiens    |             |            |      1893 |         |        |
 nm0000009 |     0 |        |         |       1 |         | Chinese Opium Den                          |
 Clovnul si cainii sai     |             |            |      1894 |         |        |
 nm0000010 |     0 |        |         |       1 |         | Corbett and Courtney Before the Kinetograph |
 Клоун и его собаки        |             |            |      1894 |         |        |
(10 rows)
```

```
shanepoldervaart_db=> SELECT * FROM team.ratings LIMIT 10;
  movieid   | avgrating | numvotes
-----------+-----------+----------
 nm0000001 | 5.6       |     1538
 nm0000002 | 5.6       |     1538
 nm0000003 | 6.1       |      185
 nm0000004 | 6.1       |      185
 nm0000005 | 6.5       |     1195
 nm0000006 | 6.2       |      113
 nm0000007 | 6.1       |     1909
 nm0000008 | 6.1       |     1909
 nm0000009 | 5.2       |      102
 nm0000010 | 5.4       |      610
(10 rows)
```

```
shanepoldervaart_db=> SELECT * from team.genre;
 id |  name
----+--------
  1 | Comedy
```

```
shanepoldervaart_db=> SELECT * FROM team.moviegenre ;
 movieid | genreid
---------+---------
 1       |       1
(1 row)
```

```
shanepoldervaart_db=> SELECT * FROM team.productioncountries
shanepoldervaart_db-> ;
 id |           name           | abbreviation
----+--------------------------+--------------
  1 | United States of America | USA
(1 row)
```

```
shanepoldervaart_db=> SELECT * FROM team.moviecountry ;
 movieid | countryid
---------+-----------
 1       |         1
(1 row)
```

```
shanepoldervaart_db=> SELECT * FROM team.languages;
 id |  name
----+---------
  1 | English
(1 row)

shanepoldervaart_db=> SELECT * FROM team.movielanguage ;
 movieid | languageid
---------+------------
 1       |          1
(1 row)

shanepoldervaart_db=> SELECT * FROM team.titletypes;
 id | name
----+------
  1 | dvd
(1 row)

shanepoldervaart_db=> SELECT * FROM team.movietype ;
 movieid | typeid
---------+--------
 1       |      1
(1 row)

shanepoldervaart_db=> SELECT * FROM team.characters;
 id |      name      | gender | orderappear | personid  | movieid
----+----------------+--------+-------------+-----------+---------
  1 | Test Character |      1 |             | nm0000001 | 1
(1 row)

shanepoldervaart_db=> SELECT * FROM team.jobs;
 id | department | position | movieid | personid
----+------------+----------+---------+-----------
  1 | Writing    | Writer   | 1       | nm0000002
(1 row)
```

Sizes:

```
shanepoldervaart_db=> SELECT COUNT(*) FROM team.people;
  count
---------
 1600930
(1 row)
```

Database Verification:

```
import java.sql.*;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;

public class sqlquerybenchmark {

    static final String JDBC_DRIVER = "org.postgresql.Driver";
    static final String DB_URL =
"jdbc:postgresql://db-315.cse.tamu.edu/shanepoldervaart_db";

    static final String USER = "shanepoldervaart";
    static final String PASS = "taeKwondo9";

    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        String sql;
        ResultSet rs = null;
        try {

            Class.forName("org.postgresql.Driver");

            System.out.println("Connecting to database...");
            conn = DriverManager.getConnection(DB_URL, USER, PASS);
            System.out.println("Opened database successfully");

            stmt = conn.createStatement();
```

```
            sql = String.format("CREATE TEMPORARY VIEW movie_master AS " +
"SELECT team.movies(id) team.");

            int runtime = 9999;
            String movieID = "0";
            String movieName, popularTitle, language, country, actor,
writer;
            double rating;
            int numvotes, actorbirthdate, writerbirthdate;
            sql = String.format("SELECT * FROM team.movies WHERE runtime =
%d;", runtime);
            rs = stmt.executeQuery(sql);
            if (rs.next()) {
                movieID = rs.getString(1);
                popularTitle = rs.getString(8);
                System.out.printf("Movie ID: %s\tPopularTitle: %s\n",
movieID, popularTitle);
            }
            // sql = String.format("SELECT avgrating, numvotes FROM
team.ratings WHERE
            // movieid = $$%s$$", movieID);
            // rs = stmt.executeQuery(sql);
            // rating = rs.getDouble(0);
            // numvotes = rs.getInt(1);

            // sql = String.format("SELECT countryid FROM
team.moviecountry WHERE movieid =
            // $$%s$$", movieID);
            // rs = stmt.executeQuery(sql);

            stmt.close();
            conn.close();
        } catch (SQLException se) {
            // Handle errors for JDBC
            se.printStackTrace();
        } catch (Exception e) {
```

```
            // Handle errors for Class.forName
            e.printStackTrace();
        } finally {
            // finally block used to close resources
            try {
                if (stmt != null)
                    stmt.close();
            } catch (SQLException se2) {
            } // nothing we can do
            try {
                if (conn != null)
                    conn.close();
            } catch (SQLException se) {
                se.printStackTrace();
            } // end finally try
        } // end try
        System.out.println(" Goodbye!");
    }// end main
}// end FirstExample
```

```
[shane.poldervaart]@linux2 ~/cs315> (23:42:44 10/01/19)
:: javac sqlquerybenchmark.java

[shane.poldervaart]@linux2 ~/cs315> (23:42:46 10/01/19)
:: java -cp .:postgresql-42.2.8.jar sqlquerybenchmark
Connecting to database...
Opened database successfully
Movie ID: 1     PopularTitle: The Real Title
 Goodbye!
```

Is successfully accessing database. Ran out of time to build out a full view to show how extensive relationship setups are.

MEMO

Date: 10/01/2019

To: Robert Lightfoot

From: Shane Poldervaart

Subject: Phase 2 of Project 1 Database progress and plan going forwards

---

Due to unforeseen circumstances with the A&M hosted database, changing used files from the provided jsons to the imdb datasets, and the unexpected variance of the data in the imdb datafiles, we were unable to complete deliverable 2 on time. With much more missing information to potentially account for in the data set, it required reworking of parts of the database, and ultimately set us behind schedule more so than just the issues with connecting to the database should have. Following is our plan for how to catch up and ultimately complete this deliverable.

Our process for getting the data into the format is to begin with the title.akas and title.basics in order to create our basic movie entity. We then populated the ratings table with the title.ratings dataset. In addition to that, information from those 2 files will help populate the genres, languages, and types tables. Utilizing that information, we will make a second loop through the 2 files in order to generate the linking tables, moviegenres, movielanguages, and movitype in order to properly link the keys for the multiple many to many relationships that are present in this database, ultimately concluding the main makeup of the movie side of the database. Next we will use the name.basics to populate the people table. Finally we utilized the title.crew and the title.principals to generate the jobs and characters tables. These are used to generate a list of all characters in any movie, associated with the respective person and movie, in addition to generating the jobs table through associating crew members such as directors and writers with their respective movies.

My role in this section was to entirely design and implement the database. creating all the tables and relations. There were 2 fairly related iterations of the database, the first one was finished the previous Sunday to work following the two json files, then reworked again to work with the multiple data files given from the imdb interfaces page. This followed with guiding the parsing to input the correct values into the tables and also doing the parsing of the complex items and setting up of linking tables. In addition, I documented the relations of the tables and produced the writing that makes up the deliverable 2 section.

MEMO

Date: 10/01/2019

To: Robert Lightfoot

From: Sandesh Neupane

Subject: Phase 2 of Project 1 Database

My role in this team project was to connect PostgreSQL database using Java Database Connectivity (JDBC), read given .tsv file using java and write data into the specific column of database table. I started this project by drawing table and their relation designed by my project team. After that, I downloaded and installed PostgreSQL JDBC driver and successfully connected and opened database. I wrote the java code to read given .tsv file row by row and stored data into listArray. After Parsing data and creating sql query, I started writing into database table. I faced a huge difficulty to complete this task because I was unable to alter the datatype of table even though my team member authorized me to work on it. It was the technical and account problem that we did not expect. Finally, I had to login with my team member login information who had created all tables, to solve this authorization problem. The second problem I faced was the database connectivity problem and database shutting down problem. Because of these technical problems, we were not able to complete all tasks on time. Apart from technical problem, I was also struggling to understand the various relations between table and data. Big issue with the data was that it was not clean. There was no data in some fields and some data contained single quotation mark, which created parsing error. In a nutshell, this project was very fruitful to me which gave me a huge opportunity to learn and implement huge relational database. I also improved my skills to work with java programming language and SQL queries.

MEMO

Date: 10/01/2019

To: Robert Lightfoot

From: Austin Offill

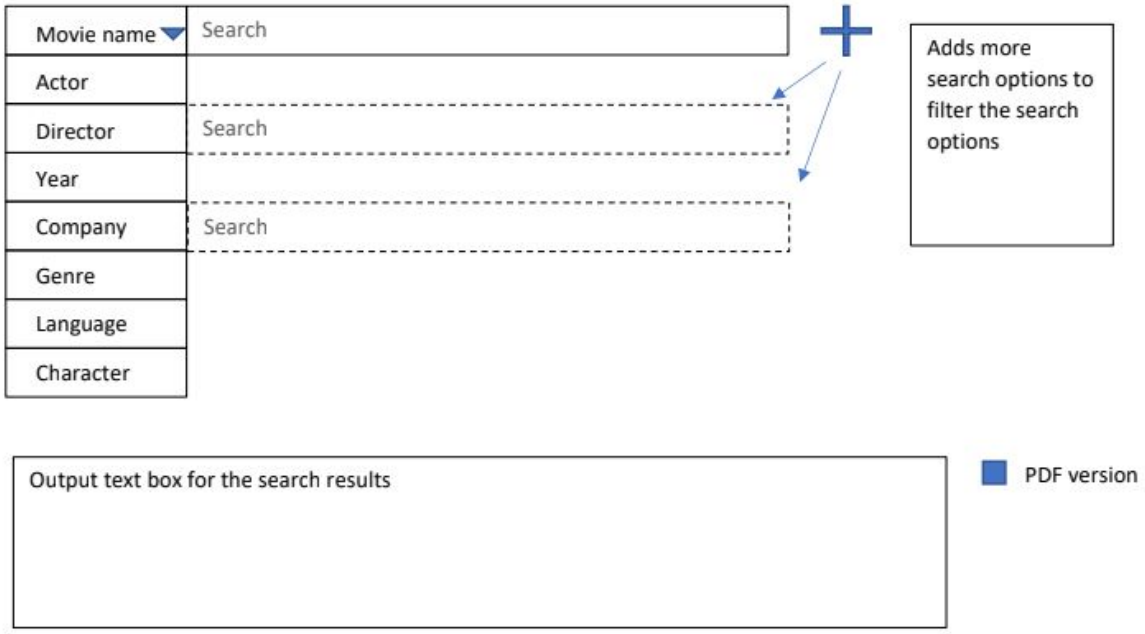
Subject: Phase 2 of Project 1 Database

In this part of the project, we started with 16 tables to load the data from IMBD into. When we started to load the data into our data tables we immediately ran into problems. We ran into an error that said we didn't have permission to write information into the data tables. The account we were using was granted full permissions but for some reason still wasn't allowed. We also ran into problems where we couldn't have more than one person connected to the data server. Since then we noticed that we only needed 13 of the 16 tables so we took out the extraneous ones. Those 3 were not used in our datasets.

In this portion of the project, I was in charge of wrighting out the name of the tables and the components of each of the tables. I had to wright which tables had foreign keys and which had primary keys. If the table had a foreign key I needed to track where it came from. Another thing I needed to keep track of was what type each column was and if the column could be left null. After I found all the useful information for a table I had to track which tables needed to be implemented first. If a table had a foreign key it had to be put in once the table containing the primary version of that key was implemented. So once the tables only containing primary keys were implemented, we worried about tables that had both primary and secondary keys, and lastly, we put in tables containing only foreign keys.

Phase 3

Deliverable #1.Database GUI Sketch:



The sketch only includes the top half of the GUI, with the bottom half being blank to leave room for the future. This GUI utilizes a modular structure, starting with a single dropdown to select the type of parameter that will be searched. This ranges from Movie name to Character as seen in the list. Next to that is the search bar that is where a user provides the exact term they want to filter by as related to the previous parameter. Next the + sign is there to create more filters to search by.

Deliverable #2. Validate the GUI

We are going to demonstrate this part in lab 10/08/2019.

Deliverable #3. Process and Contribution (approx. 250 words)

MEMO

Date: 10/07/2019

To: Robert Lightfoot

From: Shane Poldervaart

Subject: Phase 3 of Project 1 Database, GUI

The process for designing the GUI was straightforward, as we knew that it needed to have at bare minimum option with room to expand, and we wanted to make it modular in design in order to make the search more robust for the users. I designed the initial sketch for the GUI, then individually implemented the GUI in Java. This implementation was done utilizing Swing, as it is a simple library to utilize, and can fairly directly map to what is inside the initial sketch through the use of components. In addition to GUI implementation, I did all the Java PostgreSQL implementation of the queries themselves. The querying of the database is done through the use of a custom created view, utilizing parsing of the checkboxes and a specific query to search around as it is built modularly. As of this moment it is only implemented with Actors, but will be fully fleshed out to include, "Movie Title", "Release Year", "Actor", "Producer", "Director", "Average Rating", "Number of Votes", "Country (Abbreviation)", and "Genre", at a later date. This delay is due to the limited amount of time to test the database functionality given the lack of data to utilizeI had to further issues cropping up such as many repeats of data inside the IMDB given .tsv files, caused by the issues with data insertion. In addition, I provided description for

the Database GUI sketch. Finally, I aided the insertion of data into the database. This was done by providing guidance and pseudocode to Sandesh over what data files correlate to what columns in the table, and ultimately what is necessary to populate many to many linking tables such as the movie genres table. Moving forwards, I hope to attain full functionality from this Java application through completion of the modularity aspect, and the implementation of other options.

MEMO

Date: 10/07/2019

To: Robert Lightfoot

From: Sandesh Neupane

Subject: Phase 3 of Project 1 Database, GUI

The Texas A&M PostgreSQL version available to us is older than the current version, where ON CONFLICT query doesn't work. I solved this issue by adding more java code. Since the data was not clean, we were not able to add enough data in the database. In this situation, I spent some time making a python data-clean module to remove duplicate data from the given .tsv file. After I successfully removed duplicate data, we faced another foreign key conflict issue. Finally, I solved this conflict issue and inserted enough data in the database. Most of my time was spent on cleaning data and inserting to the database. Every time conflict occurred and the program got stopped, I had to delete all data from the database, solve the issue with data and parsing error, and write to the database again. If I don't delete data from the database there will be primary key conflict issues. I struggled a lot in this process and solved by adding some java code which can ignore the conflict and go to the next step. I also contributed to my team helping to design GUI and by creating a phase delivery report. This project is helping me a lot to brush up my skills writing SQL queries, parsing in java and cleaning data. Also, I got an opportunity to learn more about full-stack development.

MEMO

Date: 10/07/2019

To: Robert Lightfoot

From: Austin Offill

Subject: Phase 3 of Project 1 Database, GUI

For this phase of the project, I was in charge of making the GUI and the sketch of the GUI. I started off with the sketch of the GUI so I could have an idea of what all was needed for the GUI. After hand drawing the first sketch I took it to my team and asked them if I should put anything else in. once we got to a point that we were all good with I started the final design of the sketch on the computer. Once the final design was done I presented it to my teammates for final approval. After the GUI sketch was done I downloaded our java file off of GitHub and started adding in the elements needed for the GUI. To start off I only implemented the basics for the GUI. This way it looked the way we wanted it to. Next, I added the actual mechanics of the GUI. This way our GUI wasn't just for show anymore. I gave the dropdown menu eight options, along with a search bar that would search in the category of the dropdown. There’s a plus button that adds more search bars and plus buttons in order to narrow your search. A checkbox to get the output as a .txt file. Lastly, there is an output box that outputs the list of all the movies that correlate with the search.

Phase 4

Deliverable 1

1. Code Turned into Ecampus
2. Demonstrated in Lab

Deliverable 2

Retrospective Survey

Shane Poldervaart:

- **Was the project fun and interesting?**
  Overall, I definitely think the project was very interesting as we build it from the ground up. There were interesting constraints and issues that came up throughout the project that were not initially foreseen, such as what actually makes up the data sets. As for fun the project itself was entertaining, but there was an overbearing stress throughout the project.
- **Did it provide opportunities to innovate?**
  Yes, we had free reign on much of what we wanted to do programming wise, giving us flexibility to try new tools or those we are already familiar with to expand our skill sets.
- **What went well? What didn't go well?**
  In chronological order, the designing of the database itself went very well as there were only minor changes necessary. Even once we changed datasets, there were only a couple additional tables to create and a couple to drop. The part that went the absolute worst was the inputting of data. It took weeks to get all the data input in addition to not being entirely accurate due to the methods taken to do so. GUI design and implementation went very smoothly overall. Next and most importantly, during the final deliverable the database was completely down for 2 days in which was the time I and other team members were specifically planning on completing this project. This resulted in being unable to implement SQL queries for the final deliverable. Finally, during last minute changes to the project version control became a very real issue as we were quickly attempting to produce changes in the master branch.
- **What Lessons did you learn that you would share with your team?**
  Make sure to set up standards for version control and how it works in the beginning, as it makes a lot of issues that come up near deadlines not occur.
- **What one topic do you want to make sure we address in the retrospective meeting?**
  Version control importance. Going forwards there must be a great amount of importance placed upon proper github usage to limit merge conflicts and organization of the repository. In addition, having a more defined layout of the database to help explain the implementation of a relational database for others to use.

Sandesh Neupane:

- **Was the project fun and interesting?**
  Yes, the project was very fun and interesting. This was my first database project and it went very well. I got a chance to learn and brush up my skills designing and handling database including query language.
- **Did it provide opportunities to innovate?**
  Yes, this project provided me an opportunity to innovate database design and operation. I used BFS algorithm to get expected answer from database.
- **What went well? What didn't go well?**
  Me as well as my team successfully created attractive GUI. I successfully completed question 1 using BFS algorithm. The worst thing that happened was the database connecting issues and database was down most of the time.
- **What Lessons did you learn that you would share with your team?**
  The lesson I learned was to design relational database and showing output in java swing GUI. I shared some of my friends how user can interact with database using GUI and SQL.
- **What one topic do you want to make sure we address in the retrospective meeting?**
  Product analysis and future work topic.

Austin Offill:

- **Was the project fun and interesting?**
  The concept of the project was fun, it was also fun creating a working GUI that interacted with data.
- **Did it provide opportunities to innovate?**
  This project provided a lot of time to critically think and solve concepts new to me, forcing me to think outside the box.
- **What went well? What didn't go well?**
  Our team performed well in getting done what we chose as our parts and combining our parts to create one solid product. We also knew how to assign parts to our strengths. What didn't go well was our communication. We did not communicate to a degree where we knew what each other were doing or when we were getting things done.
- **What Lessons did you learn that you would share with your team?**
  That debugging java strings that contained SQL code is a lot more time consuming because you don't have a shell or IDE telling you where there's a problem.
- **What one topic do you want to make sure we address in the retrospective meeting?**
  Communicating to the team to make it clear what is needed from them.

Retrospective Document

- **Project overview (A summary of the work and the team)**

This project was designed to be a tool for film analysis through creative uses of IMDB data. We imported the information available from https://www.imdb.com/interfaces/ and created an easy to use GUI in swing that provides the user the ability to search based on degrees of separation, minimum spanning years of actors and directors, and the ability to modularly search for pairs between actors, directors, and producers that have worked together on films. These film names are then displayed to the user in descending order based on ratings.

The composition of the team as the project continued had each team member take some conventional roles. Shane became the pseudo team lead, as he provided primary architecture for both the database and GUI, and general direction for the team to work in. Sandesh was primarily in charge of managing the database through inputting the data, and Austin worked on much of the front end and writing.

Creation of the project was completed in a waterfall method with multiple stages of deliverables. First we created a database based off of the initial json files structures which was later altered to match the imdb database. Then we inserted data into the database by parsing the .tsv files. Afterwards we designed and implemented a GUI that would communicate with the database to accomplish the requested queries previously mentioned.

- **Engagement analysis (How was the engagement as a whole? Was the team satisfied? What were the challenges? Did we have the opportunity to innovate?)**

This was the great project and database topic was completely new for the two of us (Sandesh and Austin). Shane was familiar with database project before so, at the beginning shane explained and discuss to us and lead the design part. Our engagement for this project was well managed and well communicated. Atleast, 3 times a week we used to communicate and update status of the part we were working. We used to discuss the challenge with TA, who explained to us better algorithm for fast processing of data and accurate result. We all are fully satisfied with the involvement and effort of each team member. The challenges we faced at the beginning was database connection issue, followed by database login, and authorized. Database server was down most of the time, so we were not able to complete all tasks on time. The great thing that we want to add is, we all got great opportunity to innovate database design, implementation and GUI development. Our database design consists of multi relationship. Almost all table we designed and created had relation with multiple table.

- **Product analysis (How was our overall final product? What did we learn that we could apply to future work? How could we have done better?)**

Our overall final product had partial functionality. We had all of part one working, part 2 worked for the most part but every now and then would come up with problems. Part 3 needed dummy code because it was developed during the surver blackout, and part 4 partially worked because is was also developed during the blackout but did not have sufficient time to get it fully operational.

We learned how to create a database and upload information to it. We also now know how to create a GUI with swing and have that code communicate with a database. We had to also use java to send SQL commands that worked with the database.

Our time management could have been better getting things done sooner than we did. We needed to put more emphasis on version control, we ran into problems with that late into the project. Our Communication between what was required from teammates, sometimes left people confused on what was needed from them. Vocabulary used when talking to teammates so its at a level they understand.

- **Work analysis (Are we proud of what we accomplished? What went well? What could have gone better?)**

One part where we are proud is that we have created a full stack application from the ground up. This is a major undertaking as there are many things that need to be both coordinated and finished in order to effectively interact. With the wide spanning design of the database, it made completion of these complex tasks much simpler to execute searching for things such as degrees of separation.

We are proud that we were able to fix so much of our code that we wrote during the blackout of the server. The majority of our code was written during that time, so once it came back online we had a lot of debugging on our hands. Building the GUI and having the first part run worked really well for us. Our work during the blackout could have been done better and would have made it easier to get done right if the blackout never happened.

- **Key points to remember for future projects (What did we learn? What would we do differently the next time? What lessons would we share back with our individual teams? The class as a whole?)**

  For the future project**,** we have pointed the following suggestions:-

  1) Before starting the project we all need to make sure all team members are familiar with the topic and design idea. And if not try to get them up to speed as soon as possible to ensure the ability to effectively work in the future.
  2) Try to finish project before deadline so that we can get enough testing and update time.
  3) Don’t wait till last minute because server and connection crash may occur because of the huge number of active connections in the last hour.
  4) Emphasize version control, and set it up as soon as possible so that issues are not encountered late in the development

- **Work effort analysis (How'd we do our time together and time individually? The timeline? What was our breakdown of task per member and how did that differ from our initial estimates?)**

We spent most of our time together discussing design approach and future plans on who would accomplish what tasks.

Shane Poldervaart:

My time spent individual of the group was in both database work and GUI implementation. I created the database on the server and modified tables as necessary when changes were requested by Sandesh. I created pseudo code for inserting data into the correct tables. In addition, I created the initial back end connection for the GUI app of deliverable 2 along with implementing the team choice query front to back. This team choice query was a modular implementation of searching between any mix of two actors, producers, or directors to find their common movies all sorted by highest rating; ultimately providing a model for others on how to utilize the java postgres, SQL commands, and general backend connections. As a result I did a majority of the documentation over both the database and description of the GUI. Finally, I took on a team lead role as I tasked out objectives and deliverables with team members. In addition to this tasking out, I maintained our GitHub repo. This included the initial creation of the repo along with generating things such as the gitignore, review of all pull requests, the resolution of merge conflicts, and ultimately took charge whenever there were emergencies and last minute changes such as demo prep. Overall the amount of time I spent on this project was immensely larger than initially expected.

Sandesh Neupane:

During my individual time, Most of the work I did was cleaning data and inserting to respective table and columns. To clean data, I created a python module which cleaned big data in less than 10 seconds. After cleaning data, I started inserting into database using java. Because of connection issues and cisco VPN issue from my home, I spent most of my time in college during midnight. During midnight database server was not very busy so I inserted most of the data into the database during mid-night. I also worked for GUI using java swing, where I created GUI for question 1 having input 2 actors. Our timeline to complete all tasks was always before deadline but, due to database server issue we were not able to complete in time. The database was down for a couple of days just before the deadline.

Austin Offill:

Our time together was spent talking about what we need to get done and sometimes we spent time mapping out what we needed to get done. Individually we spent our time actually coding or creating the database. I worked on creating the GUI and then making the section for between 2 years. A lot more time was spent on the project from all of the teammates than we originally expected. This is mainly in part to the blackout. Once the server came back on we were all scrambling to fix our code