Database Systems Class Project Mid-Eval.

CSCI 3700/6070 Database Systems Fall 2019

**Database Project Group 6**  
Brandon Litzinger, Kayla Luther, Alexander Maynard, Joanna Sumner

# Which database system are you using?

**MySQL**

# What programming language are you proposing to use?

**PHP: We have utilized PHP to load the database and are using it as we create a web application.**

**CSS, HTML: We are also utilizing CSS and HTML for formatting the web application.**

# How do you design your database schemas?

We utilized the descriptions of the IMDb Dataset to develop a database schema. For each table in the dataset, we created a corresponding table in our MySQL database. We were able to do this because the original dataset is already well organized. The schema below shows the one-to-one correspondence we utilized to create one table for each table in the IMDb dataset.

The types of each table attribute/column were determined from the dataset description, though we did have to do some trial and error to determine the correct field width for the columns.

We identified primary keys for each table as well as foreign keys to provide links between tables.

Many of the tables have columns which hold an array of values. For those columns where the values held are tconsts or nconsts, the unique identifiers used for titles and people, we considered two different methods for ensuring easy access.

1. We considered creating an interim table for each column containing an array of values in the title\_crew table and the name\_basics table. For example, the title\_crew table would be split into two tables: title\_director and title\_writer. Each record in each table would hold a tconst (identifying a title) and a nconst for either a director or writer as appropriate. The tables could hold multiple directors or writers for one title. This would enable us to constrain the information in the title\_crew and name\_basics tables to link to foreign keys. In addition, the values could all be indexed which would allow for more efficient access.
2. Though the first option would be more efficient in the long run, our group decided it wasn’t feasible for us given our time constraints in trying to setup our database. Thus, we chose to keep all values from an array in a single field with values separated by commas. This doesn’t allow the values to be used directly as foreign keys, but a SQL script can be utilized to extract the values from the array and used to fulfill necessary questions.

The tables below display the schema used for our IMDb. There are seven tables total.

|  |  |
| --- | --- |
| **title\_ratings** | |
| tconst | varchar(10) |
| averageRating | double |
| numVotes | mediumint(255) |
| PRIMARY KEY(tconst) | |
| FOREIGN KEY(tconst) REFERENCES title\_basics(tconst) | |

|  |  |
| --- | --- |
| **title\_episode** | |
| tconst | varchar(10) |
| parentTconst | varchar(10) |
| seasonNumber | int(2) |
| episodeNumber | mediumint(255) |
| PRIMARY KEY(tconst) | |
| FOREIGN KEY(tconst) REFERENCES title\_basics(tconst) | |
| FOREIGN KEY(parentTconst) REFERENCES title\_basics(tconst) | |

|  |  |
| --- | --- |
| **title\_akas** | |
| titleID | varchar(10) |
| ordering | int(2) |
| title | varchar(2000) |
| region | char(5) |
| language | char(25) |
| types | varchar(30) |
| attributes | varchar(200) |
| isOriginalTitle | boolean |
| PRIMARY KEY(titleId, ordering) | |
| FOREIGN KEY(titleId) REFERENCES title\_basics(tconst) | |

|  |  |
| --- | --- |
| **title\_crews** | |
| tconst | varchar(10) |
| directors | varchar(4500) |
| writers | varchar(11700) |
| PRIMARY KEY(tconst) | |
| FOREIGN KEY(tconst) REFERENCES title\_basics(tconst) | |

|  |  |
| --- | --- |
| **title\_basics** | |
| tconst | varchar(10) |
| titleType | varchar(255) |
| primaryTitle | varchar(500) |
| originalTitle | varchar(500) |
| isAdult | boolean |
| startYear | smallint(4) |
| endYear | smallint(4) |
| runtimeMinutes | double |
| genres | varchar(50) |
| PRIMARY KEY(tconst) | |

|  |  |
| --- | --- |
| **name\_basics** | |
| nconst | varchar(10) |
| primaryName | varchar(255) |
| birthYear | smallint(4) |
| deathYear | smallint(4) |
| primaryProfession | varchar(255) |
| knownForTitles | varchar(255) |
| PRIMARY KEY(nconst) |  |

|  |  |
| --- | --- |
| **title\_principals** | |
| tconst | varchar(10) |
| ordering | int(2) |
| nconst | varchar(10) |
| category | varchar(255) |
| job | varchar(1000) |
| characters | varchar(1000) |
| PRIMARY KEY(tconst, nconst) | |
| FOREIGN KEY(tconst) REFERENCES title\_basics(tconst) | |
| FOREIGN KEY(nconst) REFERENCES name\_basics(nconst) | |

# Individual tasks

Our group did not assign specific roles for each group member. We did identify Kayla as our group leader because of her experience developing websites. The table below shows the tasks each group member has completed thus far.

Note that our group still has a fair amount of work to do in developing our web application. Specific tasks for developing and refining the frontend have not yet been assigned, therefore, there is little frontend work demonstrated in the following assignments. However, the plan is for all group members to contribute to completing the application by designing and or programming different parts of the application.

|  |  |
| --- | --- |
| Student Name | Major tasks |
| Brandon Litzinger | Assist with writing the queries for the mid-term report.  Draft designs of the web application. |
| Kayla Luther | Group leader.  Determine database schema based on IMDb dataset.  Write PHP script to load database.  Load database on personal computer to host webserver.  Lead the writing of queries for the mid-term report.  Develop PHP scripts to run queries and provide results through webpage. |
| Alexander Maynard | Determine database schema based on IMDb dataset.  Assist with writing queries for the mid-term report. |
| Joanna Sumner | Determine database schema based on IMDb dataset.  Load database on personal computer to host webserver.  Write queries for the mid-term report.  Prepare mid-term report.  Prepare presentation for in-class mid-term report. |

# Query design (Section 2.3)

* 1. Show the information of movies (Title, Region, Language, Year, Rating, Votes) which have average rating greater than 7 and number of votes greater than 1000.

Statement:

/\* Note: Movies are rated by tconst, therefore different versions of the same movie for different regions all have the same rating.

The query below selects only the first version of a movie in the dataset by selecting titles where ordering=1.\*/

**SELECT title,region,language,startYear,averageRating,numVotes**

**FROM (title\_akas JOIN title\_basics ON title\_akas.titleId = title\_basics.tconst)**

**JOIN title\_ratings ON title\_ratings.tconst = title\_basics.tconst**

**WHERE averageRating > 7 AND numVotes > 1000 AND titleType ='movie' AND ordering = 1**

**ORDER BY averageRating DESC;**

/\* The below query includes all versions of the movie so that all versions are listed, though all versions of the same tconst title still have the same rating.\*/

**SELECT title,region,language,startYear,averageRating,numVotes**

**FROM (title\_akas JOIN title\_basics ON title\_akas.titleId = title\_basics.tconst)**

**JOIN title\_ratings ON title\_ratings.tconst = title\_basics.tconst**

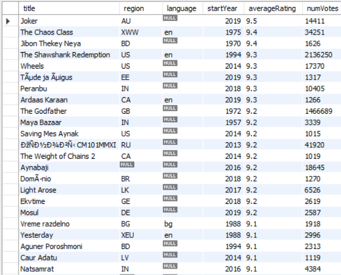
**WHERE averageRating > 7 AND numVotes > 1000 AND titleType ='movie'**

**ORDER BY averageRating DESC;**

Screenshot:

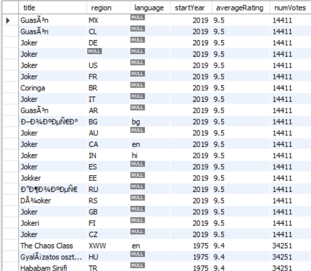
/\* Result of query for one version per unique tconst title.

The results are too many to show, so the below screenshot shows the first portion.\*/



/\* Result of query for all versions of each title.

The results are too many to show, so the below screenshot shows the first portion.\*/



* 1. Show the names of persons who have directed more than 10 movies. \*\*\*\* BUT RUNS SLOW

Statement:

**SELECT nconst, primaryName, count(\*) AS Total**

**FROM ( (SELECT tconst FROM title\_basics WHERE titleType='movie') AS titles**

**JOIN (SELECT tconst, nconst FROM title\_principals WHERE category='director') AS directors**

**USING (tconst) )**

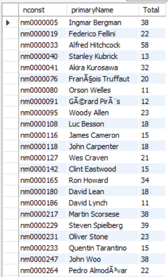
**JOIN name\_basics USING (nconst)**

**GROUP BY directors.nconst**

**HAVING Total >10;**

Screenshot:

/\* The results are too many to show, so the below screenshot shows the first portion.\*/



* 1. Show the top 10 TV series (more than 1 season) which most average ratings.

Statement:

**SELECT tconst, primaryTitle, rating FROM**

**title\_basics JOIN**

**(SELECT parentTconst, AVG(averageRating) AS rating FROM title\_episode JOIN title\_ratings USING(tconst)**

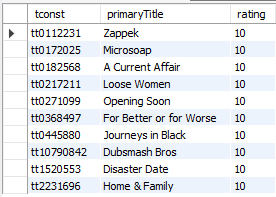
**GROUP BY parentTconst**

**HAVING SUM(DISTINCT seasonNumber) > 1) AS series\_ratings**

**ON title\_basics.tconst=series\_ratings.parentTconst**

**ORDER BY rating DESC LIMIT 10;**

Screenshot:



* 1. You need to design at least FOUR more queries which satisfy
     1. Use aggregate functions, “group by” and “order by”.
     2. Use set operations.
     3. Use nested/sub queries (either in SELECT clause, FROM clause or WHERE clause).
     4. Use JOIN operations.

1. Use aggregate functions, “group by” and “order by”.

**Query Objective: Determine the number of each titleType stored in the IMDB and list the types with the most titles first. In other words, determine how many movies, tv episodes, shorts, etc. are in the database.**

Statement:

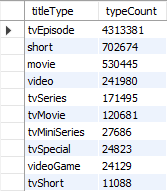
**SELECT titleType, COUNT(\*) AS typeCount**

**FROM title\_basics**

**GROUP BY titleType**

**ORDER BY typeCount DESC;**

Screenshot:



1. Use set operations.

Query Objective: Select all titles originally released for the US or Great Britain.

Note their language and release year.

Statement:

**SELECT title, region, language, startYear FROM**

**((SELECT titleId, title, region, language FROM title\_akas**

**WHERE region='US' AND isOriginalTitle=1)**

**UNION**

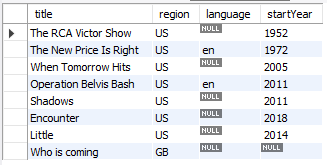
**(SELECT titleId, title, region, language FROM title\_akas**

**WHERE region='GB' AND isOriginalTitle=1)) AS title\_info**

**JOIN title\_basics ON title\_info.titleId=title\_basics.tconst;**

Screenshot:

The complete result is shown below.



1. Use nested/sub queries (either in SELECT clause, FROM clause or WHERE clause).

Query Objective: Select individuals who are only known for a single adult title and died after 2005.

Statement:

**SELECT primaryTitle, primaryName, deathYear FROM title\_basics**

**JOIN name\_basics ON tconst=knownForTitles**

**WHERE deathYear IN (SELECT deathYear FROM name\_basics WHERE deathYear > 2005)**

**AND isAdult=1**

**ORDER BY deathYear;**

Screenshot:



1. Use JOIN operations.

Query objective: Select all titles that were produced in or after 2000 for the US,

are in the English language, and are classified only in the fantasy genre.

Statement:

**SELECT DISTINCT title, language, region, genres, startYear**

**FROM title\_akas**

**JOIN title\_basics ON startYear>=2000 AND genres='Fantasy'**

**WHERE language='en' AND region='US';**

Screenshot:

/\* The results are too many to show, so the below screenshot shows the first portion.\*/



* 1. Others

The following are 5 queries which utilize either views or stored procedures.

1. Show the top 10 longest running tv series.

Run this query on a view which shows the tv series title and years running. \*/

Statement:

**/\* Create the view \*/**

**CREATE VIEW tvseries\_yearsRunning AS**

**SELECT tconst AS seriesTconst, primaryTitle, endYear-startYear AS yearsRunning**

**FROM title\_basics WHERE titleType='tvseries';**

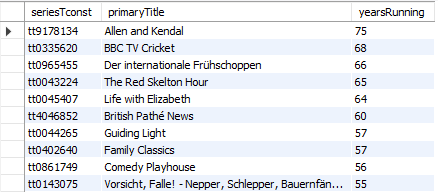
**/\* Run the example query \*/**

**SELECT \* FROM tvseries\_yearsRunning**

**ORDER BY yearsRunning DESC**

**LIMIT 10;**

Screenshot:



1. Select the movies titles and ratings for all movies released in a specified year with a rating at or above the specified minimum rating.

This query utilizes a procedure which accepts a year and minimum rating as parameters and selects all the specified titles.

Statement:

**/\* Create procedure\*/**

**CREATE PROCEDURE movies\_in\_year\_rating(IN yearSelect SMALLINT(4), IN ratingMin DOUBLE)**

**BEGIN**

**SELECT primaryTitle, averageRating**

**FROM (SELECT \* FROM title\_basics WHERE titleType='movie' AND startYear=yearSelect) AS movies\_in\_year**

**JOIN (SELECT \* FROM title\_ratings WHERE averageRating >= ratingMin) AS title\_with\_rating**

**USING (tconst);**

**END**

**/\* Call the procedure using example year and minimum rating.\*/**

**CALL movies\_in\_year\_rating(2005, 9.0);**

Screenshot:



1. Given a name, find all individuals in database with that name and list their information along with title details for one title they are known for.

This query utilizes a procedure that takes a person's name and finds all individuals with that name and titles details for one title each individual is known for.

Statement:

**/\* Create procedure\*/**

**CREATE PROCEDURE known\_for\_title(IN searchName VARCHAR(50))**

**BEGIN**

**WITH name\_title AS (SELECT nconst, primaryName, primaryProfession,**

**SUBSTRING\_INDEX(knownForTitles, ',', 1) AS tconst**

**FROM name\_basics WHERE primaryName=searchName)**

**SELECT nconst, primaryName, primaryProfession, titleType, primaryTitle, startYear, endYear**

**FROM name\_title JOIN title\_basics USING(tconst);**

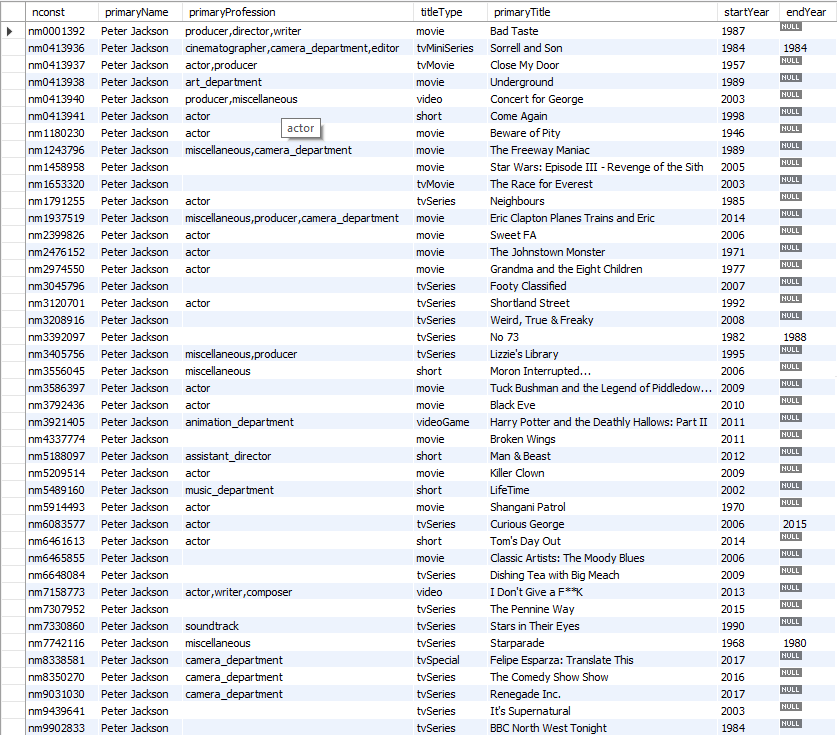
**END**

**/\* Call the procedure using an individual's name.\*/**

**CALL known\_for\_title('Peter Jackson');**

Screenshot:

As shown in the screenshot below, there are 42 individuals in the IMDb dataset with the name Peter Jackson who are known for certain titles. See them all below.



1. Find all documentaries that relate to the topics of war.

Create a view showing all documentaries. Then, run a query looking for titles with "War" in the title. Note that this query will result in some titles that are actually unrelated to war but contain the string “war”. This scenario cannot be realistically avoided. Like any search, you may see a few titles that aren’t exactly what you’re looking for.

Statement:

**/\* Create the view \*/**

**CREATE VIEW documentaries AS**

**SELECT \* FROM title\_basics WHERE genres LIKE '%Documentary%';**

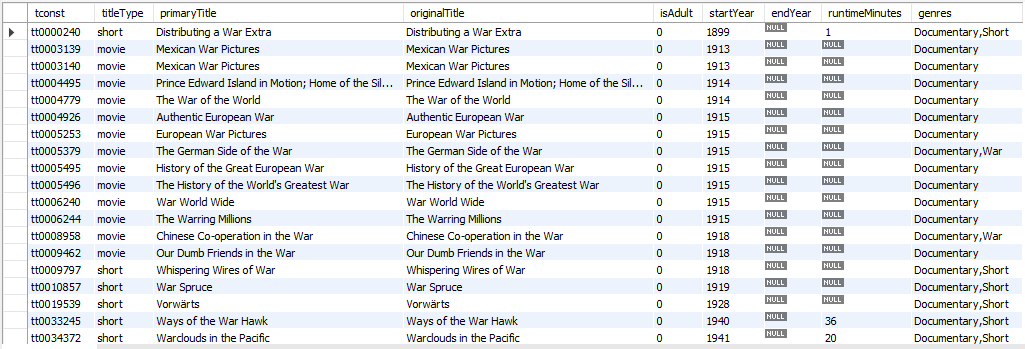
**/\* Run the query \*/**

**SELECT \* FROM documentaries WHERE**

**primaryTitle LIKE '%War%';**

Screenshot:

/\* The results are too many to show, so the below screenshot shows the first portion.\*/



1. Select all tv series that are comedies.

Use a procedure which will return information about all tv series that are in a specified genre.

Statement:

**/\* Create procedure**

**CREATE PROCEDURE tvseries\_genre(IN genre VARCHAR(20))**

**BEGIN**

**SELECT \* FROM title\_basics**

**WHERE titleType='tvseries' AND genres LIKE CONCAT('%',genre,'%');**

**END**

**/\* Call the procedure for the comedy genre.\*/**

**CALL tvseries\_genre('comedy');**

Screenshot:

/\* The results are too many to show, so the below screenshot shows the first portion.\*/

