Final Project Report

Team Members

- Rylan Schubkegel
- Sam Imlig
- Luke Havener

Database Name
death_to_hollywood

Table of Contents

Database Design

Milestone 1

ER diagram

Our team followed the schema definition of the movies database to decide which entities and relationships existed. We deviated only when something seemed completely useless or overly confusing. For example, our initial design combined actors and persons into the same entity. We thought this would make the database simpler if the *acts* relationship conveyed which persons were actors, just as directors and producers are indicated by their relationships.

We also thought that movie references were important so we created an entity called *REFERENCES*. However, this was later removed when we realized that movie references were not literature references in movies, but literature that the authors used to populate the database.

Another notable deviation from the existing schema definition was our omission of *special codes* in the ER diagram. We thought it would be better to put constraints on the data itself rather than create superfluous entities.

The initial ER diagram can be viewed in the appendix.

Milestone 2

Postgres table definitions

Milestone 2 was the creation of our Table Definitions. Essentially, we simply translated our ER Diagram schema into PostgreSQL Table Definitions, with only a few minor adjustments to encapture key attributes or relations. The first among these was the multi-valued attribute *movie_location*. We captured this information, because we thought it would be a common query to ask later on.

The second adjustment was to capture the specific roles an actor played in the movie. We called this relationship *role_types*. This became confusing towards the end of the project, because we realized role_type was the typical kind of role an actor played and not the actual role an actor played in a movie. At this period of time, we presumed the two were identical. Besides these minute changes, we simply followed our ER Diagram.

The initial table definitions can be viewed in the appendix.

Milestone 3

Database population evidence & loading plan

For Milestone 3, we didn't modify our Table Definitions from its state in Milestone 2. Our plan was to use our HTML parsers to capture and validate the data required to completely populate our tables. However, at this point, our parsers weren't fully functional, so we elected to personally load the first 5 entries into the database. This process went swimmingly, as we were still under the delusion that the HTML files contained all of the information required to properly populate the database.

Milestone 4

Most of our schema modifications happened when we started parsing and loading data into our database. We started by parsing the HTML but switched to XML to minimize the amount of file re-formatting we had to do (i.e. adding `<td>` back into table rows). In addition to this issue, we also noticed there was a lack of specific foreign key connections between a movie and a person. For example, there was no connection between a person and a author or between a person and a cinematographer in *main.html*, but there were foreign keys inside of *movies.xml*. Additionally, there was no HTML equivalent to *person.xml*, so we decided to use XML for the majority of our parsing.

Further down the line, we realized that there wasn't an XML equivalent to *STUDIOS.html*, so we actually did have to use HTML again. But instead of switching completely back to HTML, we decided that it would be simplest to manually clean up the formatting of *STUDIOS.html* and use our original HTML parser to load the data into our database.

Upon looking at the information in *cast.xml*, we discovered that there is no way to connect a *PERSON* to a *MOVIE* because *casts.xml* uses *person_id* as a foreign key whereas *actor.xml* does not have a *person_id* for an actor. As a result of this, we decided to seperate *PERSON* into an additional *ACTOR* table. We also moved *role_type* into *ACTOR* instead of having a seperate table, since there is only one role type per actor.

During the process of constructing queries, we discovered that movies had other types of persons that were not previously captured as relationships. In addition to directors and producers, movies can also have writers, authors, composers, and cinematographers. These relationships *supposedly* were captured in the original database with the *role code* attribute, but this is not a good relational design. While our final database schema still includes *role_code* as a multi-valued attribute of *person*, our intent moving forward would have been to completely remove this from our database. Instead, we would capture each type of role in a seperate relationship.

The final table definitions can be viewed in the appendix.

The final ER diagram can be viewed in the appendix.

Final Thoughts

Instead of making the Geography (*doc.html GEO*) or Category (*doc.html CATS*) files into distinct entities, we decided to use the information provided within each of these to validate the data of larger tables. In essence, these became our check constraints to avoid appending garbage data into the genre and background attributes of our database.

It quickly became apparent that using the MyWebSQL browser to populate the database was the worst way imaginable. This is because each foreign key violation, attribute missing or formatting error would cause the database to throw an error and restrict the previous valid information from being loaded. Only if every row of data was completely correct would the data be allowed into the database. Initially, this was actually quite helpful, because it allowed us to discover the additional constraints required in our parser. However, once we comprehended how jumbled and inconsistent the data was inside of the database, loading anything became a massive chore. It was much simpler to run it from command line and have the invalid inserts fail and the valid to be loaded into the database.

On Decemember 13th, we realized that we did not possess the information necessary in our database to answer query #6. However, instead of just jotting down our failture, we decided to modify our schema to fix this dilemma. To do so, we split the original table of actor_awarded into 2 tables. Initially, we had all of the information under the same table and had the capacity to find if an actor was awarded without a movie by checking to see if the movie was null. Because of this design, it quickly became apparent that *film_id* could not be a key, because it could be null. This meant that we couldn't have an actor recieve multiple awards, even if they were in multiple movies that each received an award. This new design should rectify this issue and allow us to answer the aforementioned query.

After completeing this project, we believe that we should have spent more time reading the data at the beginning. This way we could have formed a better schema, which would have stopped us from having to constantly modify it to make it congruent with the jumbled data we were parsing.

Queries

Our final queries can be viewed in the appendix, along with the number of rows they returned.

1. What movies (list title and year) have "george" (in any form) at the end of their titles. Order by the title.

   > To answer this query, we selected the title and release_year from the movie table and checked to see if it ended in george by using fuzzy matching.

2. Has any actor ever appeared in both a movie and an immediate remake of an immediate remake? If so, list the actor's stagename, the movie titles, the years, and the roles. Order by stagename, year, and movie title.

   > To answer this query, we knew that we only needed to check the original movie and the second most remake from a remake, so we started at the second most remake. We connected this second remake to the first remake via its *film_id* and then checked to see if the first movie was the first remake's original and that the second movie was the second remake's remake id. Then we connected one *acts_in* relationship with the first movie and a second *acts_in* relationship with the second movie. By doing this, we now had a link to both movies and could check to see if the actor was the same in both movies.

3. Which movie immediate remake is most simmilar to the original (i.e., has the highest percentage)? Show the title, year, director for the original movie and the remake along with the percentage of similarity between them; in the case of a tie, display them all.

> To answer this query, we first joined the *movie* table and the *remake* table together on the original *film_id*. By doing this, we had a solid grasp on the first movie. Then we joined the *movie* table and the *remake* table together once again, but this time as the remake *film_id*. This allowed us to have a grasp of both films and after joining with *directs* and *person* tables for both the original and remake films, we could answer the designated questions required. Finally, in our where clause, we made sure that we were only selecting remakes that were most like the original via a nested max query.

4. Which movie has been remade (directly or indirectly) the most times over all (i.e., is the ancestor of the most remakes)?

> To answer this query, we used a recursive query. The non-recursive term was the *film_id_orig* and by including the *film_id_orig* and *film_id_remake*, those attributes could be used in the recursive term. In the recursive term, we joined the nested query with our recursive query foo, in order to make sure it is an ancestor of the original movie. After this, we had to repeat the same process again to get the maximum count.

5. Which movies are neither a remake nor have ever been remade? Order by title.

> To answer this query, we simply checked for films that did not exist in the remakes relationship

6. List the stagename of actors that have won an Academy Award for their role in a movie; include their name and role, the name of the movie they won the award for, and the year they won; order the list by the year the movie was made.

> To answer this query, we connected *actor_awarded_for_film* with *actor*, *movie* and *acts_in* and then checked to see if the movies and stage names were congruent and that they had won an Academy Award

7. Which movies, by name, won Academy Awards in 1970?

> To answer this query, we simply connected the *movie* table with the *film_awarded* relationship and checked to see if the movie was made in 1970 and that it received an Academy Award

8. Has any original movie and an immediate remake both won an Academy Award? If so, list the name of the original and the remake along with the year for each ordered by name of the original and remake.

> To answer this query, we connected the original *movie* table with the *remakes* table and then checked to see if both the original movie and the immediate remake were in the *film_awarded* relationship. After this, we finally checked if the award that they both recieved was an Academy Award.

9. Find the name and year of the movie that has the shortest title.

> To answer this query, we simply got the *title* and *release_year* from the *movie* table and checked to see if the length of characters in the title was equal with the shortest title from the *movie* table via a nested min sub query.

10. What movies did "George Fox" write?

> To answer this query, joined the *person* table, *writes* relationship and *movie* table. After we had done this, we could simply check to see if the person was named George Fox.

11. Are there any actors that have played more than one role in the same movie?! If so, list the movie title, the actor's name and the roles they played. Order by movie title, actor's name, and role.

> To answer this query, we joined the *movie* table with the *acts_in* relationship and the *actor* table. By doing this we had a firm grasp on actors who had acted in a movie. Next, we joined a seperate *acts_in* table instance with the previous movie. By doing this, we had a way to get multiple rows. Finally, we checked to make sure the actors were the same and the roles were not the same.

12. Are there any pairs of actors that appeared together in two different movies released in the same year? If so, list the movie titles, the years, the actor's names and the roles they played. Order by movie title, year, actor's names, and roles.

> To answer this query, we joined movies with the same release year together. We ensured that the movies were not the same (their *film_id*'s were not equal) and that a particular pair of movies only appeared once in the resulting relation (ensure first film's *film_id* is "less than" the other; that is, the row (m1, m2) is not followed by (m2, m1)).
> We joined these movies to actors who appeared in both films, ensuring that the actors were not the same (their stage names were not equal) using the *acts_in* relationship. Finally, to get the actors' names, we joined the *acts_in* table to *actor*.

13. List the title, year, and role for movies that "Tom Cruise" appeared in ordered by the year.

> To answer this query, we connected the *actor* table, *acts_in* relationship and the *movie* table. Then we simply checked to make sure the actor's name was Tom Cruise.

14. Is there an actor that has been a co-star with "Val Kilmer" and "Clint Eastwood" (not necessarily in the same movie)?!

> To answer this query, we joined one *acts_in* instance with another *acts_in* instance and checked to make sure their *stage_name*'s were identical. After this process, we joined with a nested sub-query. This nested sub-query joined the *actor* table with the *acts_in* table and checked that the actor's name was Val Kilmer. The unnested query joined this sub-query on it's *film_id* and it made sure that the two *stage_name*'s were not identical. Then we joined this with another sub-query that joined the *actor* table with the *acts_in* table and checked to see if the actor's name was Clint Eastwood. This unnested and nested queries were joined on *film_id* and were checked to make sure that the actor's *stage_name*'s were not identical.

15. Give me the names of all actors within "six degrees" of "Kevin Bacon". Specifically, Bacon's co-stars (1st degree), the co-star's co-stars (2nd degree), etc. out to "six degrees". List the actors ordered by stagename.

> To answer this query, we connected an *acts_in* table with another *acts_in* table on their *film_id*'s. Then we joined another *acts_in* table with the second *acts_in* table's *stage_name*. Next, we joined another *acts_in* table with the recently created *acts_in* table on their *film_id*'s. This process continued until 6 degrees were reached. Finally, we checked to see if the original *stage_name* was Kevin Bacon.

16. List the names of all actors that have ever appeared in a movie directed by "Clint Eastwood" along with a count of how frequently they've appeard in movies he directed ordered by the count (descending) and their name (ascending).

> To answer this query, we first joined the person whose first and last name are "Clint" and "Eastwood" to the movies they directed (they appear in the directs table). Those movies are then joined with the actors in that movie by the *acts_in* relationship. Finally, we count the results (aggregate function), grouping the actors by their first and last name.

17. What are the categories (i.e., genre) of movies that "Ronald Reagan" has appeared in as an actor?

> To answer this query, we just selected the distinct categories from *movie_category* whose *film_id* is the same as a *film_id* that Ronald Reagan has acted in; that is, an actor with the stage name "Ronald Reagan".

18. Was there any year where more movies were made in Denmark than in the US? If so, give the years.

> To answer this query, we selected the count of all movies made by Denmark (grouped by release year) using fuzzy matching on the *film_location* attribute of the *movie_location* table. We also selected the count of all movies made by USA (grouped by release year) using fuzzy matching of *film_location* to "USA" **or** checking if *film_location* is in the set of US state abbreviations (i.e. "AL", "AK", "AZ", etc.). Finally, these queries were combined on matching release years and the counts of movies made that year were compared.

19. "Paramount" is a famous studio. What category (i.e., genre) of movie was most commonly made by "Paramount"?

> To answer this query, we joined two sub-queries that selected the *category* and *counts of movies with that category* of films made by Paramount. The second sub-query selected the maximum of the category counts on which to join the first sub-query.

20. Has any person directed and produced a movie they've also acted in? If so, give their stagename, the title of the movie they directed and produced, and the role(s) they played.

> To answer this query, we selected the distinct results of a union of two queries. The first query selects actors whose stage names are the same as their last name, whereas the second query selects actors whose first name and last name are the same as the first name and last name of a person in the person table.
> The reason we unioned two queries is because the data did not provide a key on which to join *person*s and *actor*s. Sometimes actors' names are the same as persons', but not always. Sometimes the actor's last name is the *person_id* from the *person* table. This inconsistency led us to union two seperate queries to capture all relavent actors/persons.

21. For all of the generic roletypes for actors, list the name of the roletype and a count of how many actors are classified by that type in descending order of the counts.

> Unfortunately, our database cannot answer this question, because we did not add the *type* attribute from Cast into our database. Instead, we added the roletype from *actors.html*. This means that we don't have the 3 character generic role types that the query is asking for.

22. For all of the generic categories for movies (e.g., "drama", "mystery"), list the name of the category (long form, if possible) and a count of how many movies are in that category for movies from 1961 in descending order of the counts.

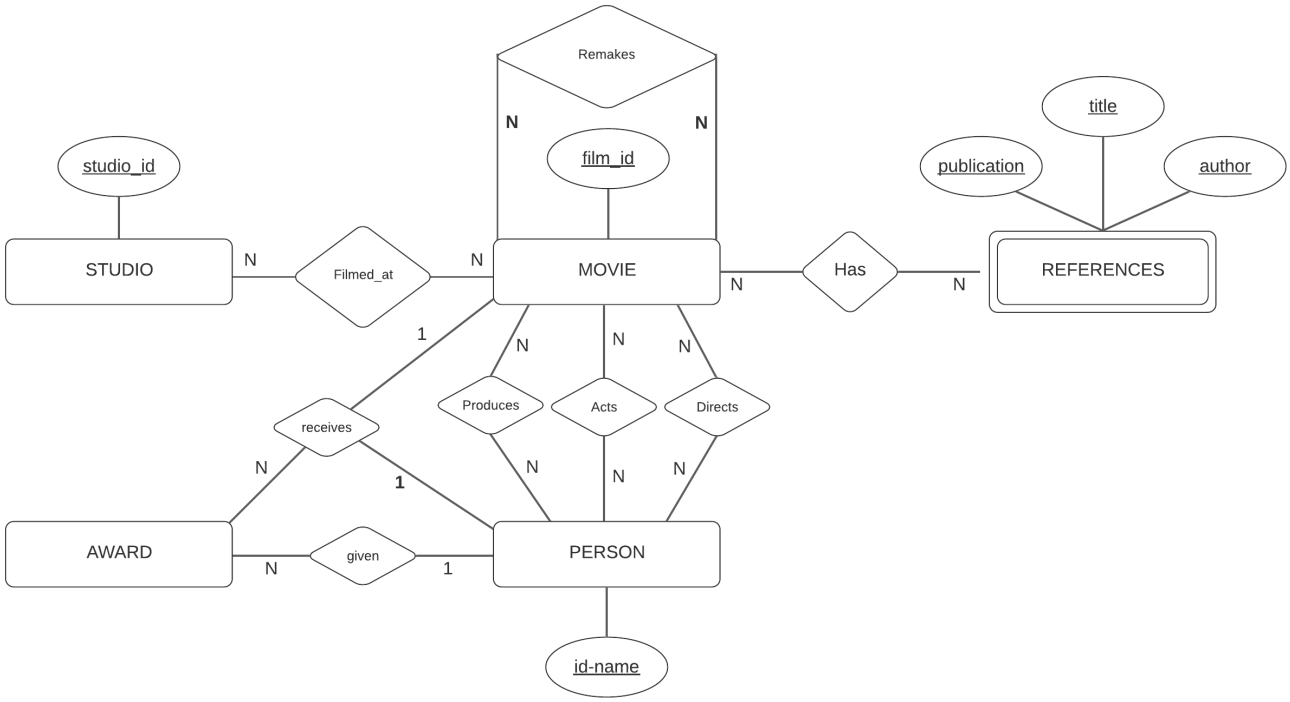
> To answer this query, we did a natural join on *movie* and *movie_category* to connect movies to their categories and used the *count* aggregate function to determine the number of movies in that category. Using the *where* clause, we selected only the category and count of that category for movies with the release year 1961.

23. Who was the oldest actor to appear in a movie? I.e., has the largest difference between their date of birth and the release of a movie they appeared in. Give their name, rough age at the time, title of the movie, and the role they played; in the case of a tie, display them all.

> To answer this query, we first connected actors to the movies they acted in by joining *actor*, *acts_in*, and *movie*. From that relation, we calculated the difference between the actor's birth year and the movie's release year (both integers). We then selected the maximum (aggregate function) of the results in a nested query and re-joined the maximum to the same original query to find the actor's first & last name, stage name, age, the movie title, and the role the actor played in that film.

Appendix

ER Diagram Version 1



ER Diagram Version 2

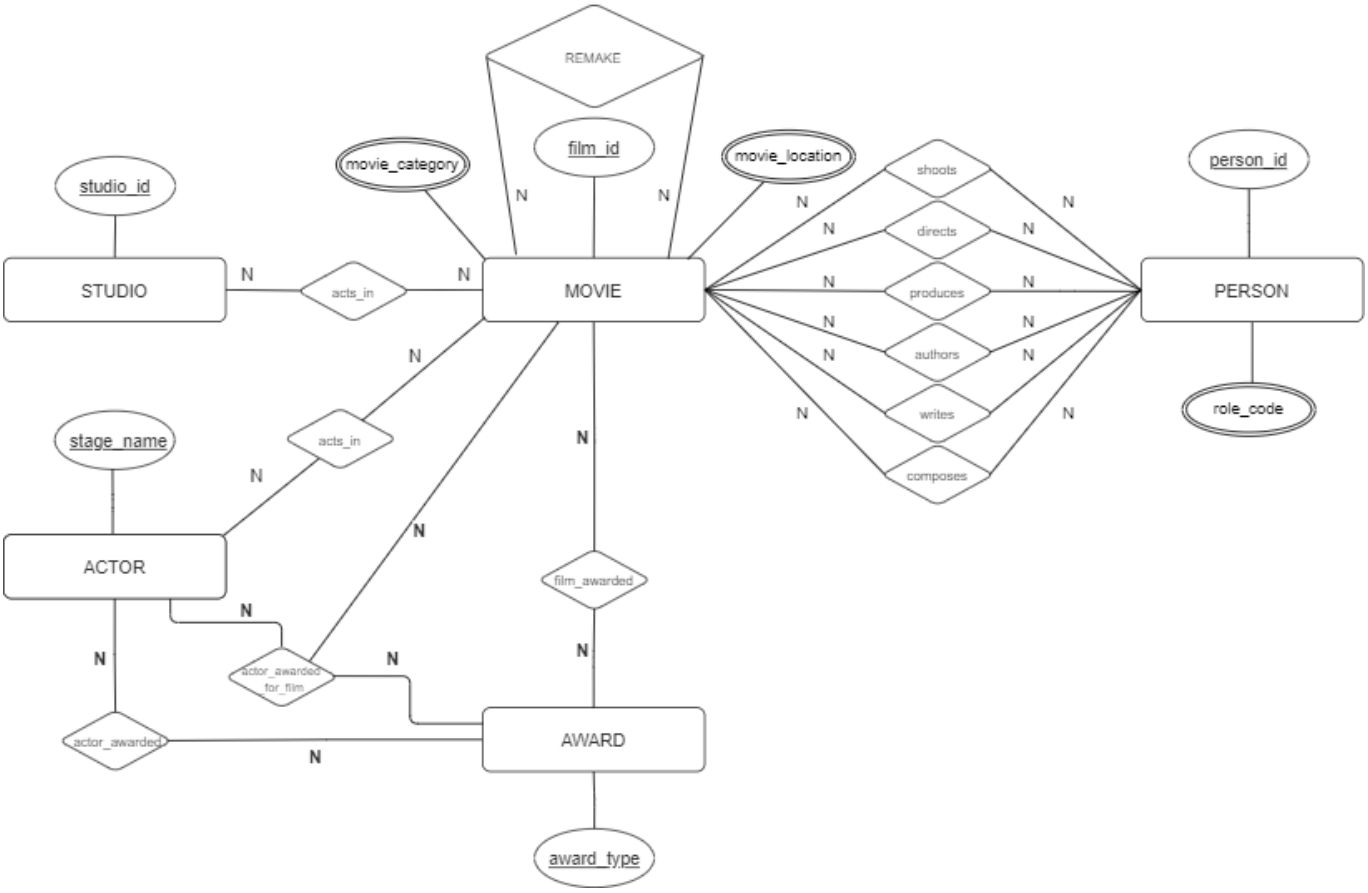


Table Definitions Version 1

```
create database movie;


create table movie(
    film_id char(8),
    title varchar(256),
    release_year date,
    notes, varchar(10000),
    primary key (film_id));


create table person(
    id_name char(7),
    p_code varchar(7) not null,
    year_start date not null,
    year_end date not null,
    name_first varchar(32) not null,
    name_last varchar(32) not null,
    name_og varchar(32),
    date_of_birth date,
    date_of_death date,
    birth_country varchar(32),
    notes varchar(10000),
    primary key (id_name));
```

```
create table studio(
    studio_id char(8),
    company varchar(32) not null,
    city varchar(32),
    country varchar(32),
    fd_date date,
    end_date date not null,
    founder varchar(32),
    successor varchar(32),
    notes varchar(10000),
    primary key (studio_id));


create table award(
    award_type varchar(64),
    award_year date,
    agency varchar(32),
    place varchar(128),
    reason varchar(32),
    notes varchar(10000),
    film_id char(8),
    id_name char(7),
    primary key(award_type, award_year, agency));


create table reference(
    film_id char(8),
    author varchar(32),
    title varchar(26,021),
    pub_info varchar(128),
    primary key (film_id, author, title, pub_info));


-- the weak entity's identifying relationship is N-N,
-- therefore we created another table to capture the relationship
create table movie_references(
    film_id char(8),
    author varchar(32),
    title varchar(26,021),
    pub_info varchar(128),
    primary key (film_id, author, title, pub_info));


create table movie_location(
    film_id char(8) not null,
    location varchar(128),
    primary key (film_id, location));


create table filmed_at(
    studio_id char(8),
    film_id char(8),
    primary key (studio_id, film_id));
```

```
create table remakes(
    film_id_orig char(8),
    film_id_remake char(8),
    part decimal,
    primary key (film_id_orig, film_id_remake));


create table produces(
    film_id char(8),
    id_name char(7),
    primary key (film_id, id_name));


create table acts(
    film_id char(8),
    id_name char(7),
    stage_name varchar(32),
    date_of_work_start date,
    date_of_work_end date,
    gender char(1),
    origin varchar(4),
    primary key (film_id, id_name));


create table role_types(
    film_id char(8),
    id_name char(7),
    role_type varchar(32),
    primary key (film_id, id_name));


create table directs(
    film_id char(8),
    id_name char(7),
    primary key (film_id, id_name));


--We decided to remove our alter table/foreign key constraints for the purpose of general readability and brevity
```

Table Definitions Version 2

```
create table movie(
    film_id varchar(8),
    title varchar(256),
    release_year int,
    notes text,
    primary key (film_id)
);

create table movie_category(    -- multivalued attribute of movie
```

```
    film_id varchar(8),
    category varchar(4) check (category in ('Ctxx', 'Actn', 'Camp', 'Comd', 'Disa', 'Epic', 'Horr', 'Noir', 'ScFi',
'West', 'Advt', 'Cart', 'Docu', 'Faml', 'Musc', 'Porn', 'Surl', 'AvGa', 'CnR', 'Dram', 'Hist', 'Myst', 'Romt', 'Susp')),
    primary key (film_id, category)
);

create table movie_location(    -- multivalued attribute of movie
    film_id varchar(8) not null,
    film_location varchar(128),
    primary key (film_id, film_location)
);

create table person(            -- can be director, producer, writer, etc.
    person_id varchar(32),
    year_start int,
    year_end int,
    name_first varchar(32),
    name_last varchar(32),
    year_of_birth int,
    year_of_death int,
    birth_country varchar(32) check (birth_country in ('Am', 'Ar', 'Au', 'Be', 'Bz', 'Ca', 'Ch', 'Cz', 'Da', 'Gr', 'Du',
'Hu', 'In','Ir', 'Me', 'Os', 'Pe', 'Ru', 'Sp', 'SA', 'Yu', 'Zw')),
    primary key (person_id)
);

create table role_code(         -- multi-valued attribute of person
    person_id varchar(32),
    role_code char(1),
    primary key (person_id, role_code)
);

create table actor(
    stage_name varchar(32),
    year_start int,
    year_end int,
    name_first varchar(32),
    name_last varchar(32),
    gender char(1),
    year_of_birth int,
    year_of_death int,
    role_type varchar(128),
    birth_country varchar(32),
    primary key (stage_name)
);

create table actor_awarded(
    award_type varchar(64),
    stage_name varchar(32),
    primary key(award_type, stage_name)
);

create table actor_awarded_for_film(
    award_type varchar(64),
    stage_name varchar(32),
```

```
    film_id varchar(8),
    primary key(award_type, stage_name, film_id)
);

create table studio(
    studio_name varchar(32),
    company varchar(64),
    city varchar(32),
    country varchar(32),
    fd_date int,
    end_date int,
    founder varchar(128),
    successor varchar(64),
    notes text,
    primary key (studio_name)
);

create table award(
    award_type varchar(64),
    agency varchar(128),
    country varchar(32),
    primary key(award_type)
);

create table film_awarded(
    award_type varchar(64),
    film_id varchar(8),
    primary key (award_type, film_id)
);

create table actor_awarded(
    award_type varchar(64),
    stage_name varchar(32),
    film_id varchar(8),             -- what movie were they given this award for?
    primary key(award_type, stage_name)
);

create table filmed_at(
    studio_name varchar(32),
    film_id varchar(8),
    primary key (studio_name, film_id)
);

create table remake(
    film_id_orig char(8),
    film_id_remake char(8),
    part decimal,
    primary key (film_id_orig, film_id_remake)
);

create table directs(               -- directors
    film_id varchar(8),
    person_id varchar(32),
    primary key (film_id, person_id)
```

```
);

create table produces(              -- producers
    film_id varchar(8),
    person_id varchar(32),
    primary key (film_id, person_id)
);

create table writes(                -- writers
    film_id varchar(8),
    person_id varchar(32),
    primary key (film_id, person_id)
);

create table shoots(                -- cinematographers
    film_id varchar(8),
    person_id varchar(32),
    primary key (film_id, person_id)
);

create table composes(              -- composers
    film_id varchar(8),
    person_id varchar(32),
    primary key (film_id, person_id)
);

create table authors(               -- athors
    film_id varchar(8),
    person_id varchar(32),
    primary key (film_id, person_id)
);


-- Specify foreign keys seperately to avoid potential chicken/egg problems

-- The categories multivalued attribute
alter table movie_category add constraint film_idfk_mc foreign key (film_id) references movie(film_id);

-- The movie location multivalued attribute
alter table movie_location add constraint film_idfk_ml foreign key (film_id) references movie(film_id);

-- NOT SURE IF NEEDED!!!!!!!!!!!!!!!!!!
alter table role_code add constraint person_idfk_rc foreign key (person_id) references person(person_id);

-- The cast relationship between ACTOR and MOVIE
alter table acts_in add constraint stage_namefk_ai foreign key (stage_name) references actor(stage_name);
alter table acts_in add constraint film_idfk_ai foreign key (film_id) references movie(film_id);

-- The awarded relationship between MOVIE and AWARD
alter table film_awarded add constraint award_typefk_fa foreign key (award_type) references
award(award_type);
alter table film_awarded add constraint film_idfk_fa foreign key (film_id) references movie(film_id);

-- The awarded relationship btween ACTOR and AWARD
```

```
alter table actor_awarded add constraint award_typefk_aa foreign key (award_type) references
award(award_type);
alter table actor_awarded add constraint stage_namefk_aa foreign key (stage_name) references
actor(stage_name);
alter table actor_awarded add constraint film_idfk_aa foreign key (film_id) references movie(film_id);

-- The filmed_at relationship between STUDIO and MOVIE
alter table filmed_at add constraint studio_namefk_fiat foreign key (studio_name) references
studio(studio_name);
alter table filmed_at add constraint film_idfk_fiat foreign key (film_id) references movie(film_id);

-- The remake_of recursive relationship with MOVIE
alter table remake add constraint film_id_origfk_r foreign key (film_id_orig) references movie(film_id);

-- The directs relationship between PERSON and MOVIE
alter table directs add constraint film_idfk_d foreign key (film_id) references movie(film_id);
alter table directs add constraint person_idfk_d foreign key (person_id) references person(person_id);

-- The produces relationship between PERSON and MOVIE
alter table produces add constraint film_idfk_p foreign key (film_id) references movie(film_id);
alter table produces add constraint person_idfk_p foreign key (person_id) references person(person_id);

-- The writes relationship between PERSON and MOVIE
alter table writes add constraint film_idfk_w foreign key (film_id) references movie(film_id);
alter table writes add constraint person_idfk_w foreign key (person_id) references person(person_id);

-- The shoots relationship between PERSON and MOVIE
alter table shoots add constraint film_idfk_s foreign key (film_id) references movie(film_id);
alter table shoots add constraint person_idfk_s foreign key (person_id) references person(person_id);

-- The composes relationship between PERSON and MOVIE
alter table composes add constraint film_idfk_c foreign key (film_id) references movie(film_id);
alter table composes add constraint person_idfk_c foreign key (person_id) references person(person_id);

-- The authors relationship between PERSON and MOVIE
alter table authors add constraint film_idfk_a foreign key (film_id) references movie(film_id);
alter table authors add constraint person_idfk_a foreign key (person_id) references person(person_id);
```

Final Queries

```
-- General Format:
-- 1. Problem #
-- 2. Question Description
-- 3. Query
-- 4. # of rows returned


-- 1
-- What movies (list title and year) have "george" (in any form) at the end of their titles. Order by the title.

select title, release_year
```

```
    from movie
    where lower(title) like '%george'
    order by title;

-- Rows returned: 4


-- 2
-- Has any actor ever appeared in both a movie and an immediate remake of an immediate remake? If so, list the actor's stagename, the movie titles, the years, and the roles. Order by stagename, year, and movie title.

select actor.stage_name,
    a1.actor_role as original_role, a2.actor_role as remake_role,
    m1.title as original_movie_title,
    m3.title as second_remake_title,
    m1.release_year as  original_movie_release_year,
    m3.release_year as second_remake_release_year

    from movie m1, movie m3, remake r1, remake r2, acts_in a1, acts_in a2, actor

    where r2.film_id_orig = r1.film_id_remake and
    m1.film_id = r1.film_id_orig and
    m3.film_id = r2.film_id_remake and
    a1.film_id = m1.film_id and
    a2.film_id = m3.film_id and
    a1.stage_name = a2.stage_name and
    a1.stage_name = actor.stage_name

    order by actor.stage_name, m1.release_year, m1.title, m3.title;

-- Rows returned: 12


-- 3
-- Which movie immediate remake is most simmilar to the original (i.e., has the highest percentage)? Show the title, year, director for the original movie and the remake along with the percentage of similarity between them; in the case of a tie, display them all.

select mold.title, mold.release_year, pold.name_first, pold.name_last, mnew.title, mnew.release_year,
pnew.name_first, pnew.name_last, part
    from movie mold join remake on film_id_orig=mold.film_id
    join movie mnew on film_id_remake=mnew.film_id
    join directs dold on mold.film_id=dold.film_id
    join person pold on dold.person_id=pold.person_id
    join directs dnew on mnew.film_id=dnew.film_id
    join person pnew on dnew.person_id=pnew.person_id
    where remake.part = (select max(part) from remake);

-- Rows returned: 17


-- 4
-- Which movie has been remade (directly or indirectly) the most times over all (i.e., is the ancestor of the most remakes)?
```

```
select title, count as num_remakes from

    -- recursive sub-query
    (with recursive foo as (

        -- non-recursive term
        select film_id_orig as og_id, film_id_orig, film_id_remake from remake as og

        union

        -- recursive term
        select og_id, rr.film_id_orig, rr.film_id_remake from remake as rr
        join foo on rr.film_id_orig = foo.film_id_remake

    ) select og_id, count(*) from foo
    group by og_id) as remake_nums

    join movie on movie.film_id = remake_nums.og_id

    -- repeat above query in order to get max of count
    where count = (select max(count) from (with recursive foo as (
        select film_id_orig as og_id, film_id_orig, film_id_remake from remake as og
        union
        select og_id, rr.film_id_orig, rr.film_id_remake from remake as rr
        join foo on rr.film_id_orig = foo.film_id_remake
    ) select og_id, count(*) from foo
    group by og_id) as remake_counts);

-- Rows returned: 1


-- 5
-- Which movies are neither a remake nor have ever been remade? Order by title.

select distinct title
    from movie
    where film_id not in (select film_id_orig from remake)
    and film_id not in (select film_id_remake from remake)
    order by title;

-- Rows returned: 10030


-- 6
-- List the stagename of actors that have won an Academy Award for their role in a movie; include their name
and role, the name of the movie they won the award for, and the year they won; order the list by the year the
movie was made.

select name_first, name_last, actor_role, title, release_year
    from actor_awarded_for_film aa
    join actor on actor.stage_name = aa.stage_name
    join movie on movie.film_id = aa.film_id
    join acts_in on acts_in.stage_name = aa.stage_name
```

```
    and acts_in.film_id = movie.film_id
    where aa.award_type = 'AA'
    order by release_year;

-- Rows returned: 209


-- 7
-- Which movies, by name, won Academy Awards in 1970?

SELECT movie.title
    from movie join film_awarded on movie.film_id = film_awarded.film_id
    where release_year = '1970' and award_type = 'AA';

-- Rows returned: 1


-- 8
-- Has any original movie and an immediate remake both won won an Academy Award? If so, list the name of
the original and the remake along with the year for each ordered by name of the original and remake.

select orig.title as movie, orig.release_year as movie_year, remk.title as remake, remk.release_year as
remake_year
    from movie orig
    join remake on remake.film_id_orig = orig.film_id
    join movie remk on remake.film_id_remake = remk.film_id
    join film_awarded orig_awrd on orig_awrd.film_id = orig.film_id
    join film_awarded remk_awrd on remk_awrd.film_id = remk.film_id
    where orig_awrd.award_type = 'AA' and remk_awrd.award_type = 'AA'
    order by orig.title, remk.title

-- Rows returned: 1


-- 9
-- Find the name and year of the movie that has the shortest title.

select title, release_year from movie where char_length(title) = (select min(char_length(title)) from movie)

-- Rows returned: 5


-- 10
-- What movies did "George Fox" write?

select movie.title
    from person join writes on person.person_id = writes.person_id join movie on writes.film_id = movie.film_id
    where person.person_id = 'George Fox'

-- Rows returned: 1


-- 11
-- Are there any actors that have played more than one role in the same movie?! If so, list the movie title, the
```

```
actor's name and the roles they played. Order by movie title, actor's name, and role.

select distinct movie.title, actor.name_first, actor.name_last, first_role.actor_role as role
    from movie join acts_in first_role on movie.film_id = first_role.film_id join actor on first_role.stage_name =
actor.stage_name
    join acts_in second_role on movie.film_id = second_role.film_id
    where first_role.stage_name = second_role.stage_name and first_role.actor_role != second_role.actor_role
    order by movie.title, actor.name_first, actor.name_last, first_role.actor_role

-- Rows returned: 225


-- 12
-- Are there any pairs of actors that appeared together in two different movies released in the same year? If so,
list the movie titles, the years, the actor's names and the roles they played. Order by movie title, year, actor's
names, and roles.

select m1.title,
    m2.title,
    m2.release_year,
    a1.name_first,
    a1.name_last,
    a2.name_first,
    a2.name_last,
    ai1m1.actor_role as actor_1_role_1,
    ai1m2.actor_role as actor_1_role_2,
    ai2m1.actor_role as actor_2_role_1,
    ai2m2.actor_role as actor_2_role_2

    from movie m1
    join movie m2 on m1.release_year = m2.release_year
    and m1.film_id != m2.film_id
    and m1.film_id < m2.film_id
    join acts_in ai1m1 on ai1m1.film_id = m1.film_id
    join acts_in ai1m2 on ai1m2.film_id = m2.film_id
    and ai1m1.stage_name = ai1m2.stage_name
    join acts_in ai2m1 on ai2m1.film_id = m1.film_id
    join acts_in ai2m2 on ai2m2.film_id = m2.film_id
    and ai2m1.stage_name = ai2m2.stage_name
    and ai1m1.stage_name != ai2m1.stage_name
    join actor a1 on a1.stage_name = ai1m1.stage_name
    join actor a2 on a2.stage_name = ai2m1.stage_name

    order by m1.title,
    m2.title,
    m2.release_year,
    a1.name_first,
    a1.name_last,
    a2.name_first,
    a2.name_last,
    ai1m1.actor_role,
    ai1m2.actor_role,
    ai2m1.actor_role,
    ai2m2.actor_role;
```

```
-- Rows returned: 346


-- 13
-- List the title, year, and role for movies that "Tom Cruise" appeared in ordered by the year.

select title, release_year, acts_in.actor_role
   from actor join acts_in on actor.stage_name = acts_in.stage_name join movie on acts_in.film_id =
movie.film_id
   where actor.stage_name = 'Tom Cruise'
   order by release_year;

-- Rows returned: 20


-- 14
-- Is there an actor that has been a co-star with "Val Kilmer" and "Clint Eastwood" (not necessarily in the same
movie)?!

select distinct ai1.stage_name
   from acts_in ai1
   join acts_in ai2 on ai1.stage_name = ai2.stage_name

   join
   (   -- select films that Val Kilmer acted in
      select acts_in.film_id, acts_in.stage_name
      from actor
      join acts_in on actor.stage_name = acts_in.stage_name
      where actor.stage_name = 'Val Kilmer') as foo
   on ai1.film_id = foo.film_id
   and ai1.stage_name != foo.stage_name

   join
   (   -- select films that Clint Eastwood acted in
      select acts_in.film_id, acts_in.stage_name
      from actor
      join acts_in on actor.stage_name = acts_in.stage_name
      where actor.stage_name = 'Clint Eastwood') as bar
   on ai2.film_id = bar.film_id
   and ai2.stage_name != bar.stage_name

   order by ai1.stage_name;

-- Rows returned: 0


-- 15
-- Give me the names of all actors within "six degrees" of "Kevin Bacon". Specifically, Bacon's co-stars (1st
degree), the co-star's co-stars (2nd degree), etc. out to "six degrees". List the actors ordered by stagename.

select distinct a5.stage_name
   from acts_in ak
   join acts_in a1 on a1.film_id = ak.film_id
```

```
    join acts_in m1 on m1.stage_name = a1.stage_name
    join acts_in a2 on a2.film_id = m1.film_id
    join acts_in m2 on m2.stage_name = a2.stage_name
    join acts_in a3 on a3.film_id = m2.film_id
    join acts_in m3 on m3.stage_name = a3.stage_name
    join acts_in a4 on a4.film_id = m3.film_id
    join acts_in m4 on m4.stage_name = a4.stage_name
    join acts_in a5 on a5.film_id = m4.film_id
    where ak.stage_name = 'Kevin Bacon';

-- Rows returned: 3773


-- 16
-- List the names of all actors that have ever appeared in a movie directed by "Clint Eastwood" along with a count of how frequently they've appeard in movies he directed ordered by the count (descending) and their name (ascending).

select actor.name_first, actor.name_last, count(*)
    from person dir join directs on directs.person_id = dir.person_id
    join movie on movie.film_id = directs.film_id
    join acts_in on movie.film_id = acts_in.film_id
    join actor on acts_in.stage_name = actor.stage_name

    where dir.name_first = 'Clint' and dir.name_last = 'Eastwood' and actor.name_first is not null
    group by actor.name_first, actor.name_last
    order by count(*) desc, actor.name_first, actor.name_last;

-- Rows returned: 27


-- 17
-- What are the categories (i.e., genre) of movies that "Ronald Reagan" has appeared in as an actor?

select distinct category
    from acts_in
    join movie_category on acts_in.film_id = movie_category.film_id
    where actor.stage_name = 'Ronald Reagan';

-- Rows returned: 6


-- 18
-- Was there any year where more movies were made in Denmark than in the US? If so, give the years.

select dn.release_year
from (
    select release_year, count(*)
    from movie
    join movie_location on movie_location.film_id = movie.film_id
    where film_location like '%Denmark'
    group by release_year
) as dn
join (
```

```
    select release_year, count(*)
    from movie
    join movie_location on movie_location.film_id = movie.film_id
    where film_location like '%USA'
    or film_location in ('AL', 'AK', 'AZ', 'AR', 'CA', 'CO', 'CT', 'DE', 'FL', 'GA', 'HI', 'ID', 'IL', 'IN', 'IA', 'KS', 'KY',
'LA', 'ME', 'MD', 'MA', 'MI', 'MN', 'MS', 'MO', 'MT', 'NE', 'NV', 'NH', 'NJ', 'NM', 'NY', 'NC', 'ND', 'OH', 'OK',
'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VT', 'VA', 'WA', 'WV', 'WI', 'WY')
    group by release_year
) as us on dn.release_year = us.release_year
where dn.count > us.count;

-- Rows returned: 0


-- 19
-- "Paramount" is a famous studio. What category (i.e., genre) of movie was most commonly made by
"Paramount"?

select category, count from

    -- selects categories and counts made by Paramount
    (select category, count(film_id) from
        (select mc.film_id, mc.category
        from movie_category mc
        join filmed_at fa on mc.film_id = fa.film_id
        where studio_name = 'Paramount'
    ) as foo group by category
    ) as bar

    -- select max from counts of categories by Paramount
    join (select max(count) from
        (select category, count from
            (select category, count(film_id) from (
                select mc.film_id, mc.category
                from movie_category mc
                join filmed_at fa on mc.film_id = fa.film_id
                where studio_name = 'Paramount'
            ) as foo group by category
        ) as bar) as foobar) as hoo
    on bar.count = hoo.max;

-- Rows returned: 105


-- 20
-- Has any person directed and produced a movie they've also acted in? If so, give their stagename, the title of
the movie they directed and produced, and the role(s) they played.

select distinct * from
(
    -- join on stage_name = last_name
    select acts_in.stage_name, title, movie.film_id, acts_in.actor_role
    from actor
    join acts_in on actor.stage_name = acts_in.stage_name
```

```
    join movie on movie.film_id = acts_in.film_id
    join role_code on role_code.person_id = name_last
    join directs on directs.film_id = movie.film_id
    and directs.person_id = role_code.person_id
    join produces on produces.film_id = movie.film_id
    and produces.person_id = role_code.person_id
    where role_code.role_code = 'A'
) as foo
union
(
    -- join on actor name = person name
    select acts_in.stage_name, title, movie.film_id, acts_in.actor_role
    from actor
    join person on actor.name_first = person.name_first
    and actor.name_last = person.name_last
    join acts_in on actor.stage_name = acts_in.stage_name
    join movie on movie.film_id = acts_in.film_id
    join directs on directs.film_id = movie.film_id
    and directs.person_id = person.person_id
    join produces on produces.film_id = movie.film_id
    and produces.person_id = person.person_id
    join role_code on role_code.person_id = person.person_id
    where role_code.role_code = 'A'
);

-- Rows returned: 48


-- 21
-- For all of the generic roletypes for actors, list the name of the roletype and a count of how many actors are classified by that type in descending order of the counts.

select role_type, count(role_type)
    from actor
    group by role_type
    order by count(role_type) desc;

-- Rows returned: 1937


-- 22
-- For all of the generic categories for movies (e.g., "drama", "mystery"), list the name of the category (long form, if possible) and a count of how many movies are in that category for movies from 1961 in descending order of the counts.

select category, count(category)
    from movie natural join movie_category
    where release_year = 1961
    group by category
    order by count(category) desc;

-- Rows returned: 14
```

```
-- 23
-- Who was the oldest actor to appear in a movie? I.e., has the largest difference between their date of birth and
the release of a movie they appeared in. Give their name, rough age at the time, title of the movie, and the role
they played; in the case of a tie, display them all.

select actor.name_first, actor.name_last, release_year-year_of_birth as age, movie.title, acts_in.actor_role
    from actor join acts_in on actor.stage_name = acts_in.stage_name
    join movie on acts_in.film_id = movie.film_id
    where movie.release_year - actor.year_of_birth = (select max(movie.release_year - actor.year_of_birth)
        from actor join acts_in on actor.stage_name = acts_in.stage_name
        join movie on acts_in.film_id = movie.film_id);

-- Rows returned: 1
```