Final Project Report

CSIS 340 - Dr. Hansen

December 13, 2021

Hannah Reynolds, Joshua Sills, Cassie Wischhoefer

The ER Diagram:

Initially, our database was conceived to be composed of two primary classes: *Person* and *Movie*. These two main classes would be connected through a variety of relations, and there would be additional supporting classes, (*Studio* and *Award*), which would help provide additional information and be defined by their relations to the primary classes. This initial conception can be visualized in figure 1.

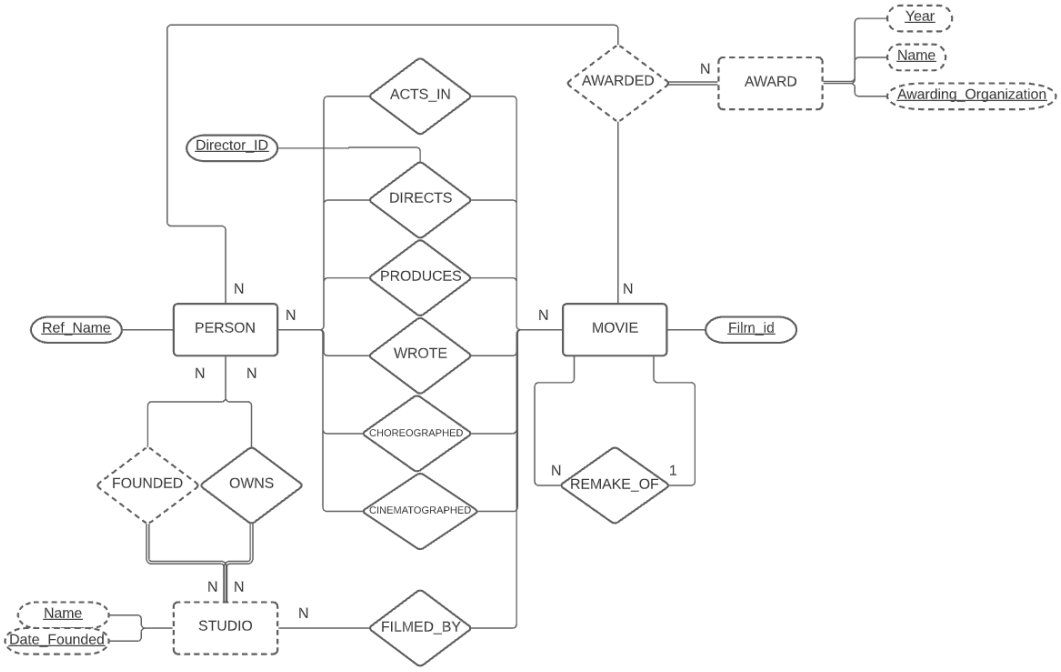

Figure 1. Initial ER diagram. Weak entities/relations are surrounded in dashed lines.

However, as soon as work on table parsing began, (where the structure of the data had to be looked at in depth), we soon realized that our structure was insufficient. Data about people was starkly divided between actors and directors/producers; found in the Actors/Casts and People/Main files respectively. From this we decided to create two "subclasses" from the *Person* entity: *Actor* and *Contributor*. Where each actor could participate acting in a movie, and each

director could participate in working on a movie. This limited redundancy in our database by allowing only one instance of a person to exist, while simultaneously allowing the flexibility to express more complex situations where a single person can both act in and direct a movie, or have multiple instantiations of these relations.

The *Studio* and *Award* entities were seen to be more independent than initially imagined, and this is reflected in changing them from weak to strong entities. The single *Awarded* relation was split into two separate relations: *Award_Person* where a person receives an award in relation to their performance in a movie, and *Award_Movie* where a movie is given an award on its own. The *Owns* and *Founded* relations were also removed between *Person* and *Studio* as very few studios actually had associated information for this, and those that did typically only had a last name which could not be easily mapped to existing *Persons*. The final ER diagram can be seen in figure 2.

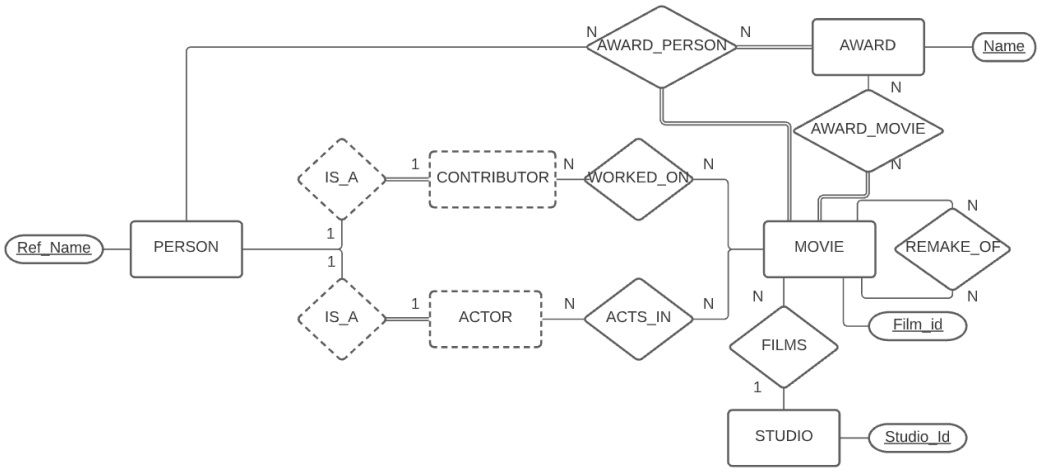

Figure 2. Final ER diagram. Weak entities/relations are surrounded in dashed lines.

Table Definitions:

Initial table definitions were based on the ER diagram in figure 1 (work on parsing had not been started so the updated version had yet to be made). Each relation between *Person* and *Movie* was given their own unique table. This was quickly pointed out to be a mistake as each of these relation tables had essentially the same structure, but different titles. This issue was fixed in creating the new tables *Works_On* and *Acts_In* which contained columns that could differentiate multiple relations of a single person to a movie. The *role_type* column in *Works_On* could differentiate between directed/produced/wrote relations as it is defined as part of the primary key. The *role_description* column in *Acts_In* could differentiate between people taking multiple roles within a single film.

The various dates within the database were initially all given type ‘date’. This was changed to type ‘int’ as all date information was decided to take the form “yyyy”. Those few areas where more precise dates were given, (e.g. July 3, 1998), were simplified to only include the year to maintain consistency across the database. As for the remaining columns, *gender* was given type ‘char’ (with values ‘M’ or ‘F’), *movie_id* was given type ‘varchar(9)’ (where 9 is the guaranteed max length of any movie id up to 9999 movies per director), and *person_id* was given type ‘int’ as it is a incremental synthetic key. Every remaining column was given type ‘text’.

An additional change that was made to the tables was assigning the *Studio* table its own synthetic key as attempts to create a natural key from its component parts was becoming too unwieldy. *Ref_name* was also changed to the aforementioned *person_id* as the reference name primary key only appeared in the People table and would thus not apply to actors. Aside from this, all final

table values reflect the ER diagram in figure 2, where each entity and relation (excluding the weak “is-a” relations) have their own table. Final table definitions can be seen in figure 3.

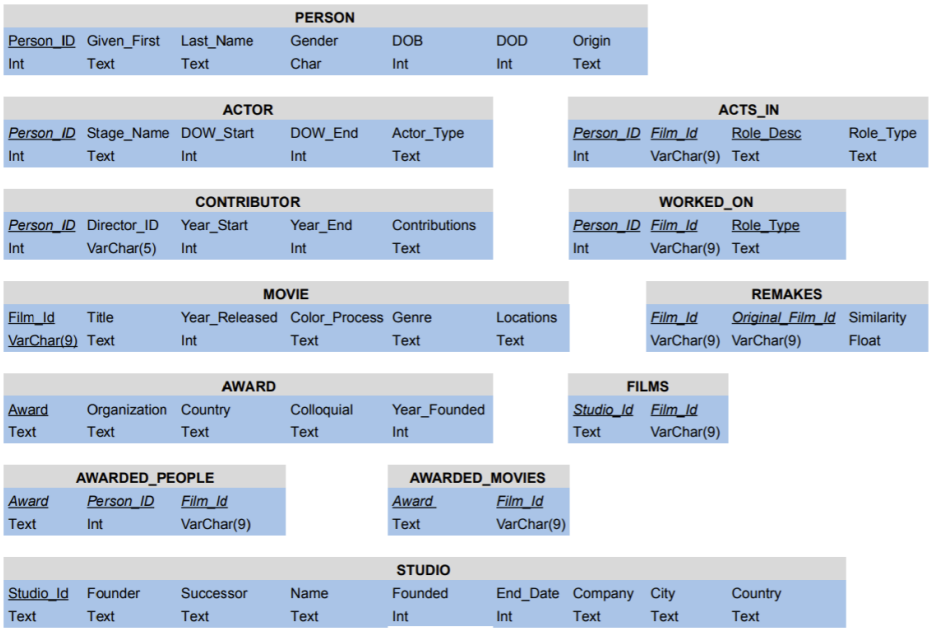

Figure 3. Final table definitions. Primary keys are underlined, and foreign keys are italicised.

Data Scraping and Loading the Database:

The initial plan to load our database was to scrape the movie database with Python scripts that parsed .xml files and threw the relevant information into .csv files. These output files could then be uploaded to PostgreSQL via the ‘copy’ command. This plan was quickly modified as there were no .xml versions of the ‘Awards’ and ‘Studios’ tables. Instead, Python .html parsers were written to load the relevant information from these tables. Additionally, the output files were adjusted to be .txt files in the form: “INSERT INTO <table_name> VALUES (‘val_1’,‘val_2’, …,‘val_n’);” where each line help info about a tuple to be inserted. Text values had to be enclosed in single quotes, numerical values had no quotes, and null values were simply written

as NULL without enclosing quotes. This change was made so that text data containing commas would not confuse the .csv file template and ruin the upload.

Writing the parsers themselves resulted in a myriad of problems; mostly due to structural inconsistencies and garbage data in the existing database. With structural errors, there were instances where the tabs for data did not exist in certain tuples, which complicated parsing and offset the data categorization in the .html files. Garbage data typically existed in the form of place holders. For example the *first name* column would often contain “fn”, “gn”, “bn”, etc., which most likely stands for “first name”, “given name” and “birth name”. Similar stand-ins could be found in *last name*, *dob*, and *dod*. For the most part, these issues were fixed by searching for all instances in the database where these character combinations existed in a given column and simply making them null. The most difficult part about this portion was that most errors only ranged from a couple instances to upwards of two-hundred. There was no consistency and many of the errors had to be caught by printing to the consul and examining the results.

Concerning dates, the database contained information where approximate dates were included, for example “19xx” and “1992+”. These approximate dates were replaced with null values as we did not want our database to be contaminated with guesswork.

Additional problems sprouted when trying to generate the relational tables *Works_On* and *Acts_In*. The way these relations were stored in the source database was based only on a person’s first and last name, (sometimes not even this). As these relations were not based off of primary keys, there is no guarantee that all of these relations are being assigned to the correct person. In

order to combat this, only complete matches of first name, last name, and dob/dod were initially allowed as valid relations. As this still left a multitude of relations unaccounted for, matches for only last name were also allowed given that the last name was unique in the database.

A final issue within the original data dealt with violation of primary key constraints. Around 50 movies violated the primary key constraint by being duplicates of other keys. In some instances, these were simply duplicate tuples which could be deleted without trouble. Others were typos where a sequence of movie ids “ID_1”, “ID_1”, “ID_3” simply had an id neglect to be incremented. If the date the film came out was between the films above and below it, this error was easy enough to fix. However, there were instances where there were duplicate ids that were not found in a sequence. In this case it was uncertain which the correct value for the ids should be. With this situation, one of the ids was changed to a valid id number and the value was recorded in a separate document with both the initial and changed values. In this way, the values could be evaluated and verified at a later date, (though this was not carried out for the scope of this project).

Writing the SQL Queries:

The SQL queries were tackled in an iterative manner, and split up among teammates evenly. Many of them were complex, and were best handled by testing small elements of them on the database, and on the company database. One of the ways used to test whether a complex query was working correctly was to use ‘select * from’ so that all the relations being joined could be shown. This method proved especially useful with queries like 14, 16, and 20, where it was found there was information missing by using test queries and ‘select *’. This also proved useful

for the infamous Kevin Bacon, where a smaller query showed accurate results for a smaller data set but resulted in exponential processing time when applied to a non-trivial data set. By minimizing the amount of data necessary and increasing the constraints on the joins, processing requirements were largely minimized and the query was able to complete in a less than infinite amount of time. When possible, relationships were joined together and then limited by clauses to enhance performance, and alternatively joined using subqueries only when necessary. The simplest way to think about writing the queries was to think as simplistically as possible about what information was needed, since writing a query is similar to tracing information through the database tables.

Final Queries:

Note: role_type and role_description meanings got flipped in our database, role_type is the description, not the generic code

Director ids are also not used, which is not ideal...

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

Row count: 4

select title, year_released from movie where title like '%george' or title like '%George' order by title;

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.**

row count: 11

SELECT og_actor.stage_name, ancestor.title, descendant.title as remake_title, ancestor.year_released, descendant.year_released as remake_year_released, og_cast.role_type as original_role, acts_in.role_type

from remakes as r

join movie as ancestor on r.original_film_id=ancestor.film_id

join remakes as s on r.film_id=s.original_film_id

join movie as descendant on descendant.film_id=s.film_id

join acts_in as og_cast on ancestor.film_id=og_cast.film_id

join actor as og_actor on og_cast.person_id=og_actor.person_id

join acts_in on descendant.film_id=acts_in.film_id and acts_in.person_id=og_cast.person_id

order by og_actor.stage_name, ancestor.year_released, ancestor.title

3. **Which movie immediate remake is most similar 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.**

Row count: 11

```
select original.title, original.year_released, director.first_name, director.last_name,

remake.title as remake_title, remake.year_released as remake_year_released, p.first_name,
p.last_name, remakes.similarity

from remakes

join movie as original on remakes.original_film_id=original.film_id

join movie as remake on remakes.film_id=remake.film_id

join worked_on on worked_on.film_id=original.film_id and worked_on.work_type='director'

join worked_on as w on w.film_id=remake.film_id and w.work_type='director'

join person as director on director.person_id=worked_on.person_id

join person as p on p.person_id=w.person_id

where remakes.similarity = (select max(similarity) from remakes)
```

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

Row count: 1

```
select max(totalRemakes)

  from (with recursive remakes_movies as

    (select m.film_id as movieId, rm.film_id from movie as m join remakes as rm on m.film_id =
rm.original_film_id

      union

     select re.film_id, r.film_id from remakes as re inner join remakes as r on r.film_id =
re.original_film_id)

   select count(movieId) as totalRemakes from remakes_movies group by movieId) as s;
```

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

Row count: 10,353

```
select * from movie

where not exists (select remakes.film_id from remakes where movie.film_id=remakes.film_id or
movie.film_id=remakes.original_film_id)
```

order by movie.title

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.**

Row count: 106

select a.stage_name, i.role_type, a.actor_type, m.title, m.year_released

from actor as a join awarded_people as p on a.person_id = p.person_id join movie as m on p.film_id = m.film_id join acts_in as i on a.person_id = i.person_id and m.film_id = i.film_id

where award_name = 'AA'

order by m.year_released;

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

Row count: 2

select m.title from movie as m join awarded_movies as a on m.film_id = a.film_id where a.award_name = 'AA' and m.year_released = '1970';

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.**

Row count: 1

select m.title, m.year_released, f.title, f.year_released

from movie as m join awarded_movies as a on m.film_id = a.film_id join remakes as r on m.film_id = r.original_film_id join awarded_movies as s on s.film_id = r.film_id join movie as f on f.film_id = r.film_id

where a.award_name = 'AA' and s.award_name = 'AA';

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

Row count: 5

select title, year_released from movie where length(title)=(select min(length(title)) from movie)

10. **What movies did "George Fox" write?**

row count: 0

We didn’t parse the ‘notes’ file so George Fox isn’t in our database

```
select title from person as p
join worked_on as w on p.person_id = w.person_id
join movie as m on m.film_id = w.film_id
where first_name = 'George' and last_name = 'Fox' and w.work_type = ‘writer’;
```

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.**

Row count: 120

```
select movie.title, actor.stage_name, string_agg(acts_in.role_type, ',') as role_descriptions  from acts_in

join (select person_id, film_id from acts_in group by person_id, film_id having count(*) > 1) as many_acts

on acts_in.person_id=many_acts.person_id and acts_in.film_id=many_acts.film_id

 join movie on many_acts.film_id=movie.film_id

 join actor on many_acts.person_id=actor.person_id

 group by movie.title, actor.stage_name
```

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.**

Row count: 298

```
select c.stage_name, d.stage_name, a.year_released, string_agg(movie.title, ', ') as titles from

 (select a.person_id as actor, b.person_id as coactor, a.film_id as fid, year_released

 from acts_in as a

 join movie on a.film_id=movie.film_id

 join acts_in as b on movie.film_id=b.film_id

 where not a.person_id=b.person_id) as a

 join

 (select a.person_id as actor, b.person_id as coactor, a.film_id as fid, year_released
```

```
from acts_in as a
join movie on a.film_id=movie.film_id
join acts_in as b on movie.film_id=b.film_id
where not a.person_id=b.person_id) as b
on a.year_released=b.year_released and a.actor=b.actor and a.coactor=b.coactor
join actor as c on a.actor=c.person_id
join actor as d on a.coactor=d.person_id
join movie on a.fid=movie.film_id
where not a.actor=b.coactor and not b.actor=a.coactor and not a.fid=b.fid
group by c.stage_name, d.stage_name, a.year_released
```

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

Row count: 22

```
select movie.title, movie.year_released, acts_in.role_description
from actor
join acts_in on actor.person_id=acts_in.person_id
join movie on acts_in.film_id=movie.film_id
where actor.stage_name='Tom Cruise'
order by year_released;
```

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

Row count: 1

```
select clint.stage_name
  from (select d.stage_name from actor as a
  join acts_in as b on a.person_id=b.person_id
  join acts_in as c on b.film_id=c.film_id
```

```
  join actor as d on c.person_id=d.person_id
  where a.stage_name='Clint Eastwood') as clint
  join (select h.stage_name from actor as e
     join acts_in as f on e.person_id=f.person_id
     join acts_in as g on f.film_id=g.film_id
     join actor as h on g.person_id=h.person_id
     where e.stage_name='Val Kilmer') as val
    on clint.stage_name=val.stage_name
```

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.**

Row count: 3750

```
select distinct b.stage_name from actor
  join acts_in as movies_in on actor.person_id=movies_in.person_id
  join acts_in as costars on movies_in.film_id=costars.film_id
  join acts_in as their_movies on costars.person_id=their_movies.person_id
  join acts_in as costars2 on their_movies.film_id=costars2.film_id
  join acts_in as co2movies on costars2.person_id=co2movies.person_id
  join acts_in as costars3 on co2movies.film_id=costars3.film_id
  join acts_in as co3movies on costars3.person_id=co3movies.person_id
  join acts_in as costars4 on co3movies.film_id=costars4.film_id
  join acts_in as co4movies on costars3.person_id=co4movies.person_id
  join acts_in as costars5 on co4movies.film_id=costars5.film_id
  join acts_in as co5movies on costars4.person_id=co5movies.person_id
  join acts_in as costars6 on co5movies.film_id=costars6.film_id
  join actor as b on costars6.person_id=b.person_id
```

```
where actor.stage_name='Kevin Bacon'
and not movies_in.person_id=costars.person_id
and not their_movies.person_id=costars2.person_id
and not co2movies.person_id=costars3.person_id
and not co3movies.person_id=costars4.person_id
and not co4movies.person_id=costars5.person_id
and not co5movies.person_id=costars6.person_id
```

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 appeared in movies he directed ordered by the count (descending) and their name (ascending).**

Row count: 47

```
select actor.stage_name, count(*) as num_appearances
 from person as p
 join worked_on as w on p.person_id=w.person_id
 join acts_in as a on w.film_id=a.film_id
 join actor on a.person_id=actor.person_id
 where p.first_name='Clint' and p.last_name='Eastwood' and work_type='director'
 group by actor.stage_name
 order by count(*) desc, actor.stage_name asc
```

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

Row count: 7

```
select distinct genre from movie join acts_in on movie.film_id=acts_in.film_id
join actor on acts_in.person_id=actor.person_id
where stage_name = 'Ronald Reagan' and genre is not null;
```

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

Row count:1

```
select d.year_released from

  (select count(*) as num, year_released from movie as m join films as f on m.film_id = f.film_id
join studio as s on s.studio_id = f.studio_id where s.country = 'USA' group by m.year_released)
as u

full outer join

  (select count(*) as num, year_released from movie as m join films as f on m.film_id = f.film_id
join studio as s on s.studio_id = f.studio_id where s.country = 'Denmark' group by
m.year_released) as d on d.year_released = u.year_released

where d.year_released is not null;
```

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

Row count: 1

The genre most commonly made by "Paramount" is comedy, or "Comd"

```
select genre, count(genre) as genre_count from studio join films on
studio.studio_id=films.studio_id

  join movie on films.film_id=movie.film_id

  where studio.studio_name='Paramount'

  group by genre

  having count(genre)=(select max(genre_count)

    from (select genre, count(genre) as genre_count

      from studio join films on studio.studio_id=films.studio_id

      join movie on films.film_id=movie.film_id

      where studio.studio_name='Paramount'

      group by genre) as sub_a)
```

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.**

Row count: 10

```
select actor.stage_name, movie.title, string_agg(acts_in.role_type, '; ')

from (select person_id, film_id from worked_on where work_type='producer') as producers

join (select person_id, film_id from worked_on where work_type='director') as directors on
producers.film_id=directors.film_id and producers.person_id=directors.person_id

join acts_in on directors.person_id=acts_in.person_id and directors.film_id=acts_in.film_id

join movie on acts_in.film_id=movie.film_id

join actor on acts_in.person_id=actor.person_id

group by actor.stage_name, movie.title
```

21. For all of the generic role types 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.

Row count: 90

```
select role_description, count(*) from acts_in where role_description is not null group by
role_description order by count(*) desc;
```

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.**

Row count: 19

```
select genre, count(*)  from movie  where year_released = '1961'  group by genre  order by count
desc;
```

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.**

Row count: 1

```
select stage_name, (movie.year_released-person.dob) from person

join actor on person.person_id=actor.person_id

        join acts_in on actor.person_id=acts_in.person_id

        join movie on acts_in.film_id=movie.film_id

where (movie.year_released-person.dob) is not null
```

```
and (movie.year_released-person.dob)=
(select max(movie.year_released-person.dob) from person
join acts_in on person.person_id=acts_in.person_id
join movie on acts_in.film_id=movie.film_id)
order by (movie.year_released-person.dob) desc;
```