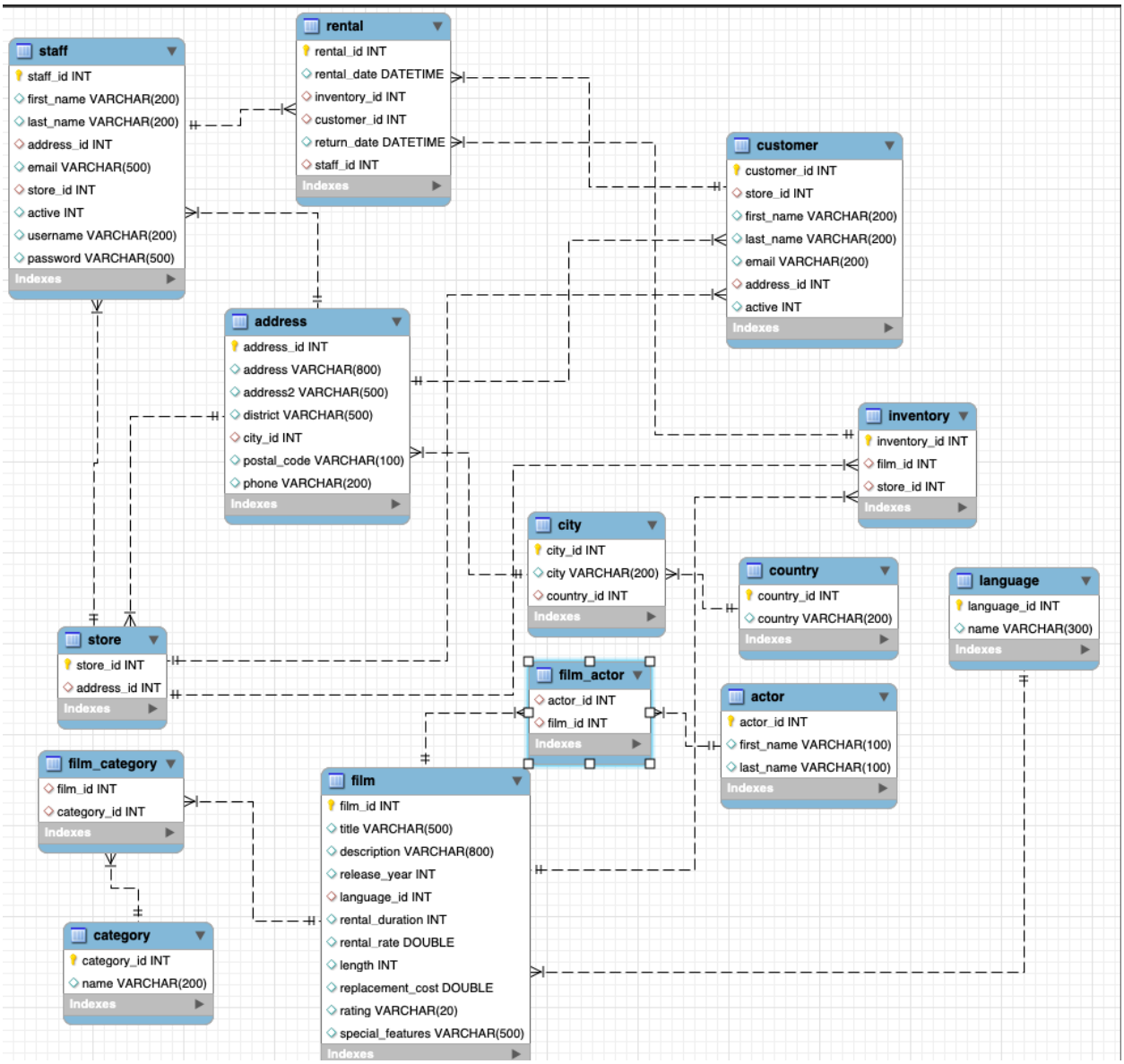
**Title:** DB Assignment 4

**Name:** Devon Reing

**Date:** 31 October 2024




The ER diagram for this picture illustrated above shows the relationships between the 14 tables and the attributes for each entity set. Primary keys are indicated with a gold key at the top of the attribute list next to the attribute in the corresponding table. Foreign keys that are used in creating relationships between the tables are indicated by the red diamond next to the

corresponding attribute. These foreign keys come from the connected entity's primary key. Regular attributes are indicated by the blue diamonds next to the corresponding attributes. In this diagram, relationships are shown through the dotted lines that connect tables.

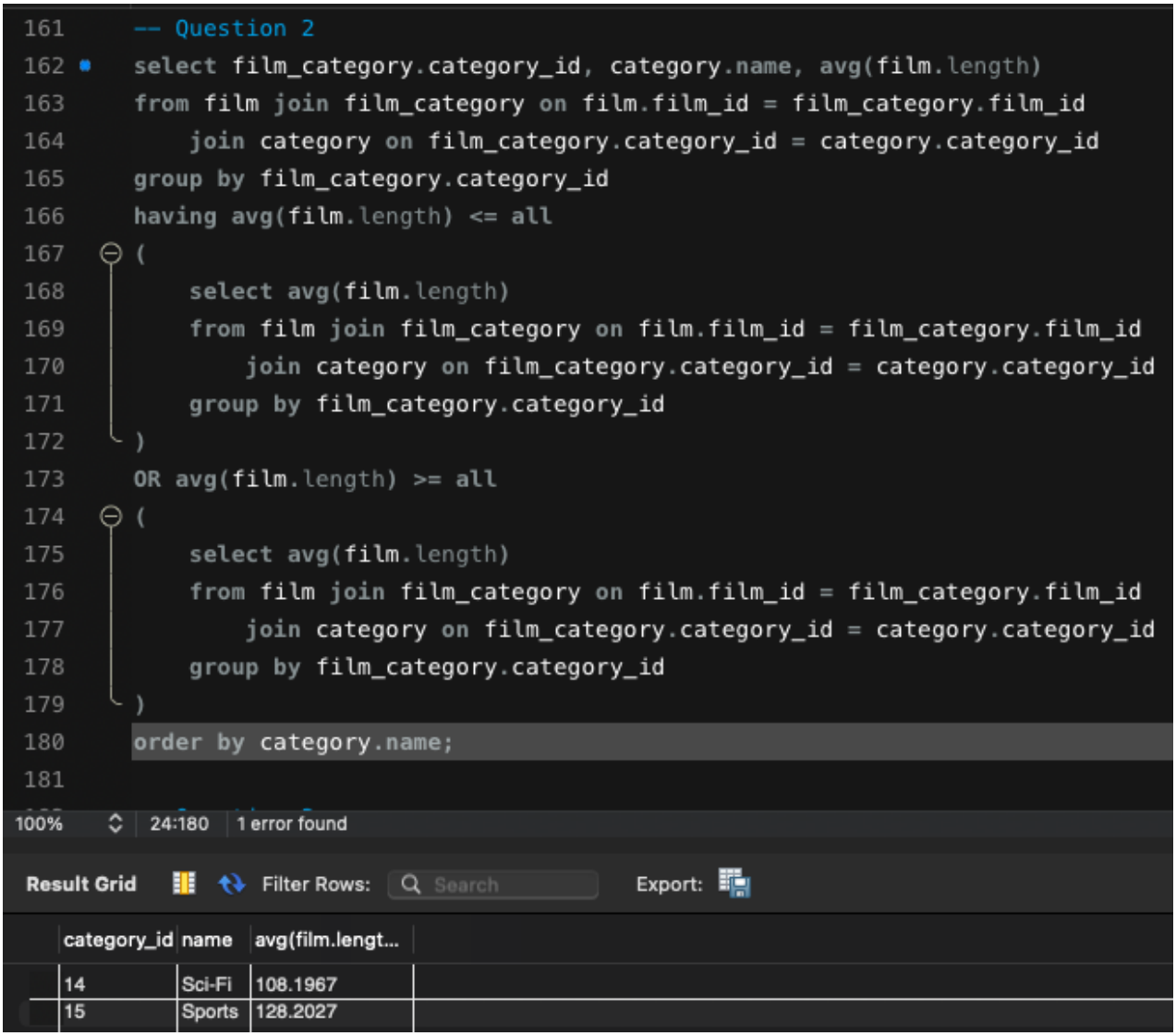
```
154     -- Question 1
155 •   select film_category.category_id, category.name, avg(film.length)
156     from film join film_category on film.film_id = film_category.film_id
157         join category on film_category.category_id = category.category_id
158     group by film_category.category_id
159     order by category.name;
```

100% 24:159 1 error found

Result Grid Filter Rows: Search Export:

| category_id | name | avg(film.lengt... |
|---|---|---|
| 1 | Action | 111.6094 |
| 2 | Animation | 111.0152 |
| 3 | Children | 109.8000 |
| 4 | Classics | 111.6667 |
| 5 | Comedy | 115.8276 |
| 6 | Documentary | 108.7500 |
| 7 | Drama | 120.8387 |
| 8 | Family | 114.7826 |
| 9 | Foreign | 121.6986 |
| 10 | Games | 127.8361 |
| 11 | Horror | 112.4821 |
| 12 | Music | 113.6471 |
| 13 | New | 111.1270 |
| 14 | Sci-Fi | 108.1967 |
| 15 | Sports | 128.2027 |
| 16 | Travel | 113.3158 |

The first query aims to find the average film length in minutes of each category. To achieve this, the film and film_category tables are joined using an inner join on the film_id attribute and the film_category and category tables are joined using an inner join on the category_id attribute. These three tables joined together create one table where the category_id, category name, and film length are all accessible. The query selects these attributes and performs an average on the film length for each category, which is indicated in the group by statement. Finally, the results tabled is organized alphabetically through the order by statement.



```
-- Question 2
select film_category.category_id, category.name, avg(film.length)
from film join film_category on film.film_id = film_category.film_id
    join category on film_category.category_id = category.category_id
group by film_category.category_id
having avg(film.length) <= all
(
    select avg(film.length)
    from film join film_category on film.film_id = film_category.film_id
        join category on film_category.category_id = category.category_id
    group by film_category.category_id
)
OR avg(film.length) >= all
(
    select avg(film.length)
    from film join film_category on film.film_id = film_category.film_id
        join category on film_category.category_id = category.category_id
    group by film_category.category_id
)
order by category.name;
```

100% 24:180 1 error found

Result Grid Filter Rows: Search Export:

| category_id | name | avg(film.lengt... |
|---|---|---|
| 14 | Sci-Fi | 108.1967 |
| 15 | Sports | 128.2027 |

The second query aims to find the shortest and longest average film length by category. This is achieved by altering the first query to include a having statement. While the initial structure of the first query is the same in order to find the average length of the films for each category, in order to find the min and max, additional components must be added. The having statement performs two checks joined by an or statement to ensure both the shortest and longest film length are returned. The first check finds the shortest average length. It selects only the average film length using the same table created by the three joins and groups the average by category. It then checks each result against each other to only return the shortest length in this case (or lengths if

two times happened to be the same and both the shortest). The second check finds the longest average length. It selects only the average film length using the same table created by the three joins and groups the average by category. It then checks each result against each other to only return the longest length in this case (or lengths if two times happened to be the same and both the longest). Both results are then returned in the results table.

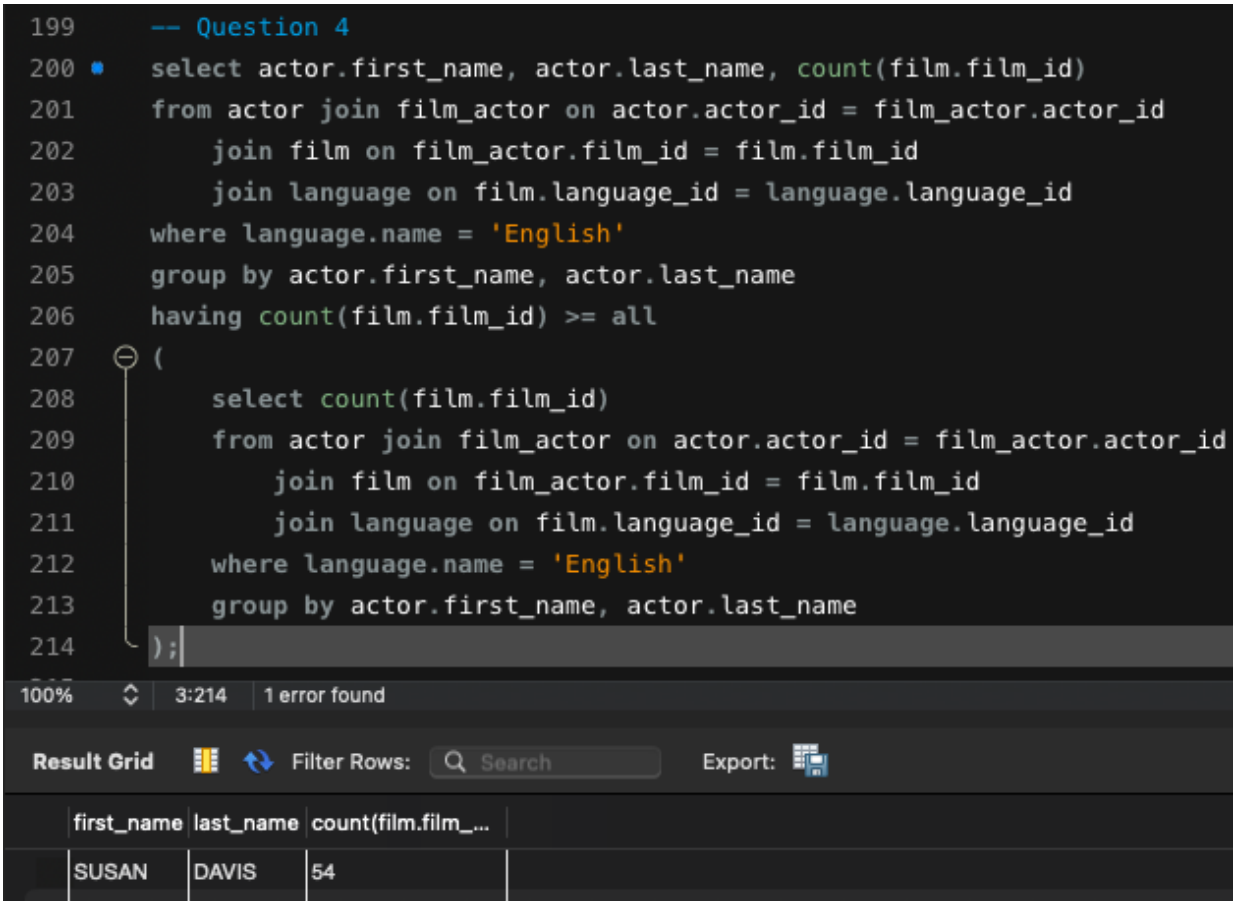
```
182     -- Question 3
183     select customer.first_name, customer.last_name
184     from customer join rental on customer.customer_id = rental.customer_id
185         join inventory on rental.inventory_id = inventory.inventory_id
186         join film on inventory.film_id = film.film_id
187         join film_category on film.film_id = film_category.film_id
188         join category on film_category.category_id = category.category_id
189     where category.name = 'Action'
190     EXCEPT
191     select customer.first_name, customer.last_name
192     from customer join rental on customer.customer_id = rental.customer_id
193         join inventory on rental.inventory_id = inventory.inventory_id
194         join film on inventory.film_id = film.film_id
195         join film_category on film.film_id = film_category.film_id
196         join category on film_category.category_id = category.category_id
197     where category.name = 'Comedy' or category.name = 'Classics';
```

100% 62:197 1 error found

Result Grid Filter Rows: Search Export:

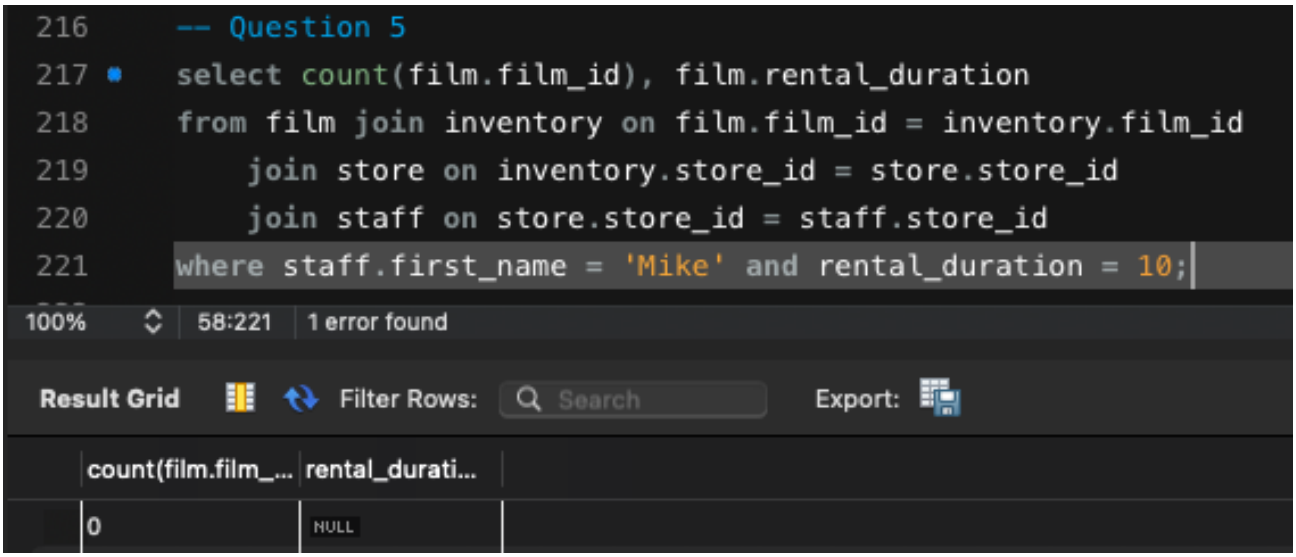
| first_name | last_name |
|---|---|
| LAWRENCE | LAWTON |
| MATTHEW | MAHAN |
| TOM | MILNER |
| JO | FOWLER |
| SCOTT | SHELLEY |
| EDWIN | BURK |
| JOANN | GARDNER |
| DONNA | THOMPSON |
| DON | BONE |
| JUAN | FRALEY |
| DOLORES | WAGNER |
| MICHEAL | FORMAN |
| AMBER | DIXON |
| MELINDA | FERNANDEZ |
| CONSTAN... | REID |
| RUBY | WASHING... |
| GINA | WILLIAMS... |

The third query aims to find the customers who rented an action movie but not a comedy or classics movie. This is achieved by joining the customer and rental table using an inner join on the customer_id attribute. Rental is then joined with inventory using an inner join on the inventory_id attribute. Film is then joined with inventory using an inner join on the film_id attribute. Film is also joined with film_category using an inner join on film_id. Category is then joined with film_category using an inner join on using the category_id attribute. Once this larger table is created through the joins, the rental history of each customer is accessible. Only the customers who rented movies will be included in this table due to the use of inner joins. The query selects the customers first and last name from the customer table if the customer has rented a movie where the category associated with that film's film_id is action. Once this results table is created, an except statement is utilized to remove any customers who also rented comedy or classics movies. Just like above, this portion of the query selects the customer's first and last name from the larger table that is created through inner joins as long as the customer has rented a movie whose film_id is associated with the category name comedy or classics. The customers are selected only for those categories through the where statement, which narrows down the results list. Once this list is created the except statement compares the first list with the second and removes any names that overlap. This narrowed down list is then returned in the results table.




The fourth query aims to find the actor who has been in the most films where the language is English. This is achieved by joining the actor and film_actor table using an inner join on the actor_id attribute. Film_actor is then joined with film using an inner join on the film_id attribute, and finally language and film are joined using an inner join on the language_id attribute. Once this larger table is created, the actors name and what films they have been in as well as the film's language are accessible for the query. The main query select the actor's full name and the number of film's they are in. The where statement makes sure that only the films where the language is English are included in the count. The group by statement ensures that the count is done for each individual actor and not the total number of films where the language is English. The having statement is where each actor is compared against each other to see who has the most English films. In the subquery within the having statement, the same larger table is created using

inner joins and only the number of English films is included in the select statement. The where statement once again checks that only English films are included and the group by statement once again ensures that the count is done for each actor. The having statement compares all the results from the subquery and only selects the max, which in this case is only one value.




The fifth query aims to find the number of rentals that were rented from the store that Mike works at for exactly 10 days. This is achieved through joining film and inventory using an inner join on film_id, store and inventory using an inner join on store_id, and staff and store using an inner join on store_id. These joins create a larger table where the staff names, films, and rental durations are all accessible. The select statement selects each film and the film's rental duration and the where statement narrows this list down to the films rented from the store where Mike works and only the films rented for 10 days exactly. The count aggregate in the select statement then counts the number of films returned.

```sql
-- Question 6
with largest_movie as (
    select film.film_id, count(actor.actor_id)
    from film join film_actor on film.film_id = film_actor.film_id
        join actor on film_actor.actor_id = actor.actor_id
    group by film.film_id
    having count(actor.actor_id) >= all
    (
    select count(actor.actor_id)
    from film join film_actor on film.film_id = film_actor.film_id
        join actor on film_actor.actor_id = actor.actor_id
    group by film.film_id
    )
)

select actor.first_name, actor.last_name
from actor join film_actor on actor.actor_id = film_actor.actor_id
    join film on film_actor.film_id = film.film_id
    join largest_movie on largest_movie.film_id = film.film_id
order by actor.first_name, actor.last_name;
```

100% 44:242 1 error found

Result Grid Filter Rows: Search Export:

| first_name | last_name |
|---|---|
| BURT | POSEY |
| CAMERON | ZELLWEGER |
| CHRISTIAN | NEESON |
| FAY | WINSLET |
| JAYNE | NOLTE |
| JULIA | BARRYMORE |
| JULIA | ZELLWEGER |
| LUCILLE | DEE |
| MENA | HOPPER |
| MENA | TEMPLE |
| REESE | KILMER |
| SCARLETT | DAMON |
| VAL | BOLGER |
| WALTER | TORN |
| WOODY | HOFFMAN |

The sixth query aims to list all the actors involved in the movie with the largest cast of actors in alphabetical order. This is achieved through the use of a CTE named largest_movie that finds the largest movie and a main query that then lists the actors involved in that movie. Starting with the CTE, film and film_actor are joined using an inner join on film_id, and then actor and film_actor

are joined using an inner join on actor_id to create a larger table where the film ids and actor ids are accessible. The count aggregate on the actor_id in the select statement counts the number of actors in each movie. The group by statement ensures that the count is done for each film and not the total number of actors for every movie. The having statement then compares the counts for each film to find the movie with the most actors. This is achieved through a subquery in the having statement that selects the count for each movie using the same join statements to create the larger table. The counts are again grouped by film. The having statement only returns the film_id where the number of actors is the max for each film, which in this case is only one movie. The CTE holds the results of the film_id with the max number of actors to then be used in the main query. The main query then performs an inner joins on actor and film_actor on actor_id, film and film_actor on film_id, film and largest_movie (the CTE from above) on film_id. This larger table created through joins makes the actors names for the film that matches the film id from the CTE results accessible. Therefore, only the actors names that are in the largest movie are included in this larger table created through joins. The select statement selects these actors full names and then orders them alphabetically by first name and then last name if the first name is a duplicate.