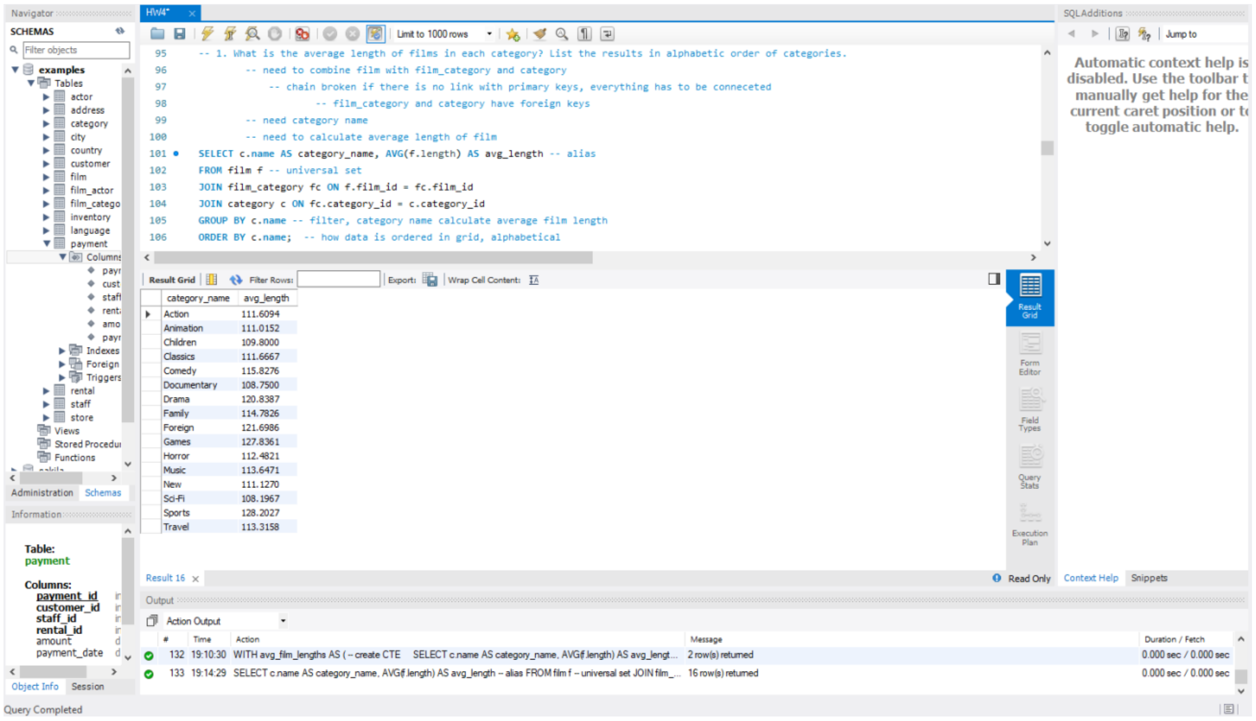
Christina DiMaggio

November 4, 2024

DB Assignment 4

1. **What is the average length of films in each category? List the results in alphabetic order of categories.**



```
-- 1. What is the average length of films in each category? List the results in alphabetic order of categories.
        -- need to combine film with film_category and category
            -- chain broken if there is no link with primary keys, everything has to be conneceted
                    -- film_category and category have foreign keys
        -- need category name
        -- need to calculate average length of film
SELECT c.name AS category_name, AVG(f.length) AS avg_length -- alias
FROM film f -- universal set
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id
GROUP BY c.name -- filter, category name calculate average film length
ORDER BY c.name;  -- how data is ordered in grid, alphabetical
```

category_name	avg_length
Action	111.6094
Animation	111.0152
Children	109.8000
Classics	111.6667
Comedy	115.8276
Documentary	108.7500
Drama	120.8387
Family	114.7826
Foreign	121.6986
Games	127.8361
Horror	112.4821
Music	113.6471
New	111.1270
Sci-Fi	108.1967
Sports	128.2027
Travel	113.3158

```
#    Time      Action                                                                                      Message              Duration / Fetch
132  19:10:30  WITH avg_film_lengths AS ( -- create CTE    SELECT c.name AS category_name, AVG(f.length) AS avg_lengt...  2 row(s) returned    0.000 sec / 0.000 sec
133  19:14:29  SELECT c.name AS category_name, AVG(f.length) AS avg_length -- alias FROM film f -- universal set JOIN film_...  16 row(s) returned   0.000 sec / 0.000 sec
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

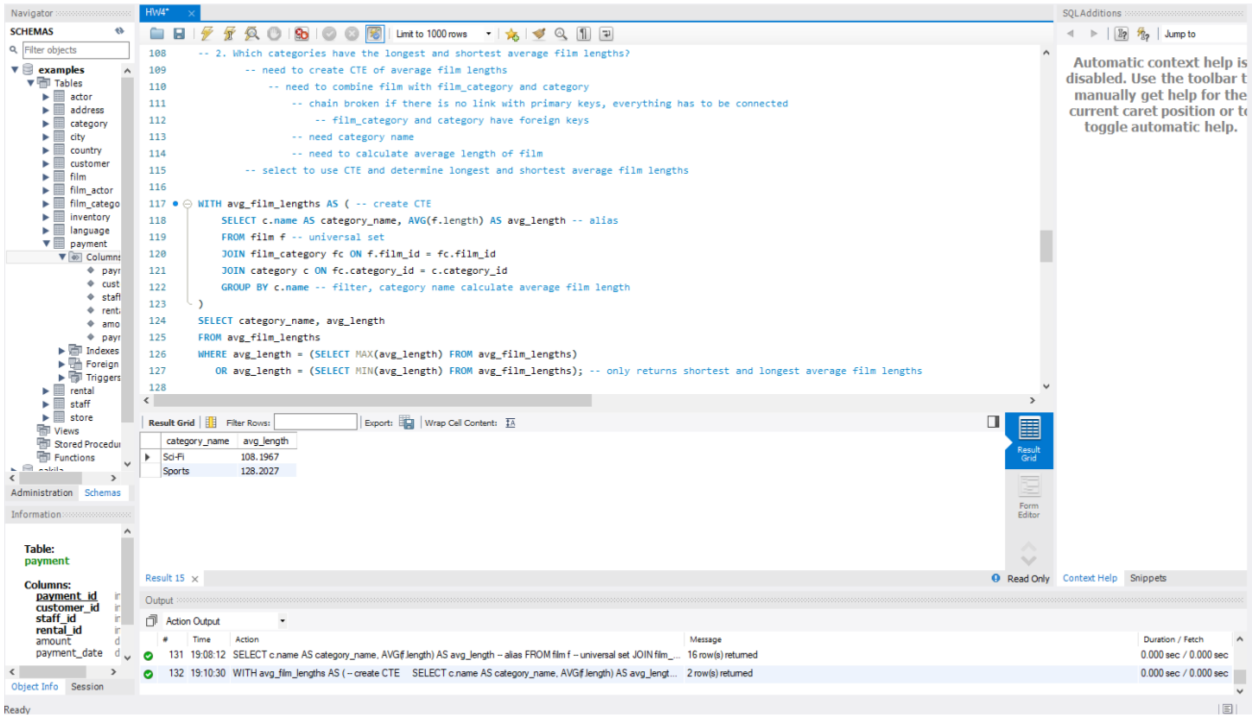
Query Completed

What does query do? How does it solve the problem?

This query can INNER join film with film_category and catagory by utilizing their primary keys and or foreign keys. Inner joins are for exactly which columns we need and are using. Comments on SQL queries also better define what codes represent.

The query nicely lays out the category name on the left and its respective average film length on the right of the table

2. Which categories have the longest and shortest average film lengths?



```sql
-- 2. Which categories have the longest and shortest average film lengths?
        -- need to create CTE of average film lengths
            -- need to combine film with film_category and category
                -- chain broken if there is no link with primary keys, everything has to be connected
                    -- film_category and category have foreign keys
                -- need category name
                -- need to calculate average length of film
        -- select to use CTE and determine longest and shortest average film lengths

WITH avg_film_lengths AS ( -- create CTE
    SELECT c.name AS category_name, AVG(f.length) AS avg_length -- alias
    FROM film f -- universal set
    JOIN film_category fc ON f.film_id = fc.film_id
    JOIN category c ON fc.category_id = c.category_id
    GROUP BY c.name -- filter, category name calculate average film length
)
SELECT category_name, avg_length
FROM avg_film_lengths
WHERE avg_length = (SELECT MAX(avg_length) FROM avg_film_lengths)
   OR avg_length = (SELECT MIN(avg_length) FROM avg_film_lengths); -- only returns shortest and longest average film lengths
```

category_name	avg_length
Sci-Fi	108.1967
Sports	128.2027

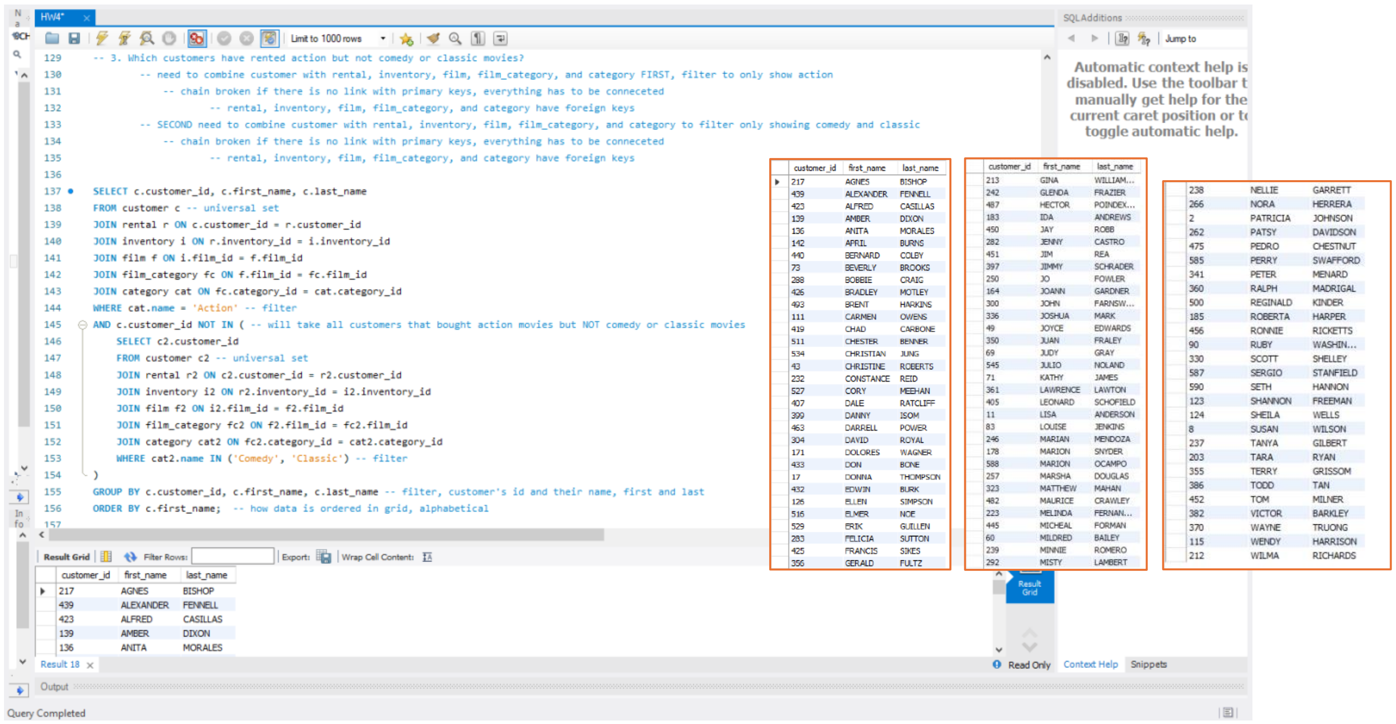
```
#    Time      Action                                                                                    Message            Duration / Fetch
131  19:08:12  SELECT c.name AS category_name, AVG(f.length) AS avg_length -- alias FROM film f -- universal set JOIN film_...  16 row(s) returned  0.000 sec / 0.000 sec
132  19:10:30  WITH avg_film_lengths AS ( -- create CTE    SELECT c.name AS category_name, AVG(f.length) AS avg_lengt...  2 row(s) returned  0.000 sec / 0.000 sec
```

What does query do? How does it solve the problem?

This query creates a CTEs to be able to calculate the longest and shortest average film length by category by joining film with film_category and category. Select is important for determining which movie category has the shortest and longest film lengths. Comments on SQL queries also better define what codes represent.

The query nicely lays out the category name and average film length. It will only return the highest (first one listed) and the lowest (second one listed).

3. Which customers have rented action but not comedy or classic movies?



```sql
-- 3. Which customers have rented action but not comedy or classic movies?
        -- need to combine customer with rental, inventory, film, film_category, and category FIRST, filter to only show action
            -- chain broken if there is no link with primary keys, everything has to be conneceted
                    -- rental, inventory, film, film_category, and category have foreign keys
        -- SECOND need to combine customer with rental, inventory, film, film_category, and category to filter only showing comedy and classic
            -- chain broken if there is no link with primary keys, everything has to be conneceted
                    -- rental, inventory, film, film_category, and category have foreign keys

SELECT c.customer_id, c.first_name, c.last_name
FROM customer c -- universal set
JOIN rental r ON c.customer_id = r.customer_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN film f ON i.film_id = f.film_id
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category cat ON fc.category_id = cat.category_id
WHERE cat.name = 'Action' -- filter
AND c.customer_id NOT IN ( -- will take all customers that bought action movies but NOT comedy or classic movies
    SELECT c2.customer_id
    FROM customer c2 -- universal set
    JOIN rental r2 ON c2.customer_id = r2.customer_id
    JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
    JOIN film f2 ON i2.film_id = f2.film_id
    JOIN film_category fc2 ON f2.film_id = fc2.film_id
    JOIN category cat2 ON fc2.category_id = cat2.category_id
    WHERE cat2.name IN ('Comedy', 'Classic') -- filter
)
GROUP BY c.customer_id, c.first_name, c.last_name -- filter, customer's id and their name, first and last
ORDER BY c.first_name;  -- how data is ordered in grid, alphabetical
```

customer_id	first_name	last_name
217	AGNES	BISHOP
439	ALEXANDER	FENNELL
423	ALFRED	CASILLAS
139	AMBER	DIXON
136	ANITA	MORALES
142	APRIL	BURNS
440	BERNARD	COLBY
73	BEVERLY	BROOKS
288	BOBBIE	CRAIG
426	BRADLEY	MOTLEY
493	BRENT	HARKINS
111	CARMEN	OWENS
419	CHAD	CARBONE
511	CHESTER	BENNER
534	CHRISTIAN	JUNG
43	CHRISTINE	ROBERTS
232	CONSTANCE	REID
527	CORY	MEEHAN
407	DALE	RATCLIFF
399	DANNY	ISOM
463	DARRELL	POWER
304	DAVID	ROYAL
171	DOLORES	WAGNER
433	DON	BONE
17	DONNA	THOMPSON
432	EDWIN	BURK
126	ELLEN	SIMPSON
516	ELMER	NOE
529	ERIK	GUILLEN
283	FELICIA	SUTTON
425	FRANCIS	SIKES
356	GERALD	FULTZ
213	GINA	WILLIAM...
242	GLENDA	FRAZIER
487	HECTOR	POINDEX...
183	IDA	ANDREWS
450	JAY	ROBB
282	JENNY	CASTRO
451	JIM	REA
397	JIMMY	SCHRADER
250	JO	FOWLER
164	JOANN	GARDNER
300	JOHN	FARNSW...
336	JOSHUA	MARK
49	JOYCE	EDWARDS
350	JUAN	FRALEY
69	JUDY	GRAY
545	JULIO	NOLAND
71	KATHY	JAMES
361	LAWRENCE	LAWTON
405	LEONARD	SCHOFIELD
11	LISA	ANDERSON
83	LOUISE	JENKINS
246	MARIAN	MENDOZA
178	MARION	SNYDER
588	MARION	OCAMPO
257	MARSHA	DOUGLAS
323	MATTHEW	MAHAN
482	MAURICE	CRAWLEY
223	MELINDA	FERNAN...
445	MICHEAL	FORMAN
60	MILDRED	BAILEY
239	MINNIE	ROMERO
292	MISTY	LAMBERT
238	NELLIE	GARRETT
266	NORA	HERRERA
2	PATRICIA	JOHNSON
262	PATSY	DAVIDSON
475	PEDRO	CHESTNUT
585	PERRY	SWAFFORD
341	PETER	MENARD
360	RALPH	MADRIGAL
500	REGINALD	KINDER
185	ROBERTA	HARPER
456	RONNIE	RICKETTS
90	RUBY	WASHIN...
330	SCOTT	SHELLEY
587	SERGIO	STANFIELD
590	SETH	HANNON
123	SHANNON	FREEMAN
124	SHEILA	WELLS
8	SUSAN	WILSON
237	TANYA	GILBERT
203	TARA	RYAN
355	TERRY	GRISSOM
386	TODD	TAN
452	TOM	MILNER
382	VICTOR	BARKLEY
370	WAYNE	TRUONG
115	WENDY	HARRISON
212	WILMA	RICHARDS

Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or to toggle automatic help.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

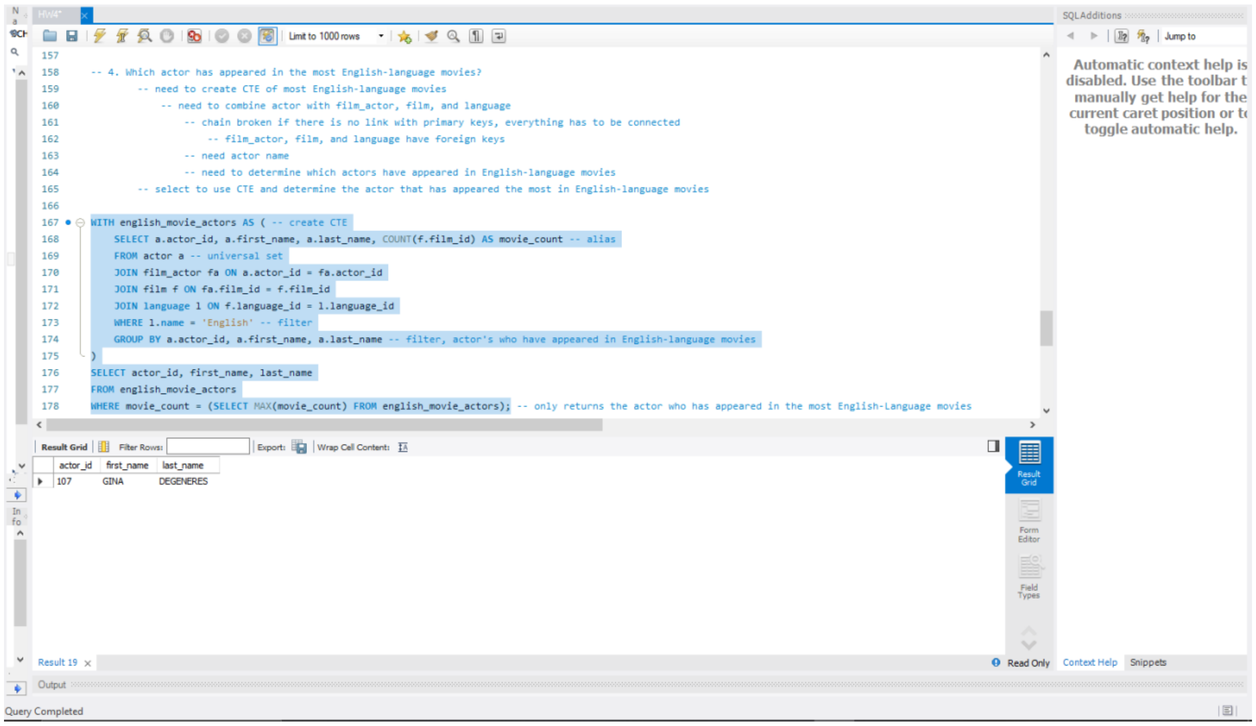
customer_id	first_name	last_name
217	AGNES	BISHOP
439	ALEXANDER	FENNELL
423	ALFRED	CASILLAS
139	AMBER	DIXON
136	ANITA	MORALES

What does query do? How does it solve the problem?

This query can join customer with rental, inventory, film, film_category, and category filter to only show action by utilizing their primary keys and or foreign keys. INNER join is completed twice, once with where clause Action, the other with where clause Comedy and Classic. Command ‘AND c.customer_id NOT IN’ takes the customer_id of someone who rented an action film and compares it to the table of customers who rented comedy and classic films. It will report customers who only bought action films and not comedy and classic films. Comments on SQL queries also better define what codes represent.

The query nicely lays out the customer id, with their respective first and last name, of who rented action but comedy or classic.

4. **Which actor has appeared in the most English-language movies?**



```sql
-- 4. Which actor has appeared in the most English-language movies?
        -- need to create CTE of most English-language movies
            -- need to combine actor with film_actor, film, and language
                -- chain broken if there is no link with primary keys, everything has to be connected
                    -- film_actor, film, and language have foreign keys
                -- need actor name
                -- need to determine which actors have appeared in English-language movies
        -- select to use CTE and determine the actor that has appeared the most in English-language movies

WITH english_movie_actors AS ( -- create CTE
    SELECT a.actor_id, a.first_name, a.last_name, COUNT(f.film_id) AS movie_count -- alias
    FROM actor a -- universal set
    JOIN film_actor fa ON a.actor_id = fa.actor_id
    JOIN film f ON fa.film_id = f.film_id
    JOIN language l ON f.language_id = l.language_id
    WHERE l.name = 'English' -- filter
    GROUP BY a.actor_id, a.first_name, a.last_name -- filter, actor's who have appeared in English-language movies
)
SELECT actor_id, first_name, last_name
FROM english_movie_actors
WHERE movie_count = (SELECT MAX(movie_count) FROM english_movie_actors); -- only returns the actor who has appeared in the most English-Language movies
```

actor_id	first_name	last_name
107	GINA	DEGENERES

Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.

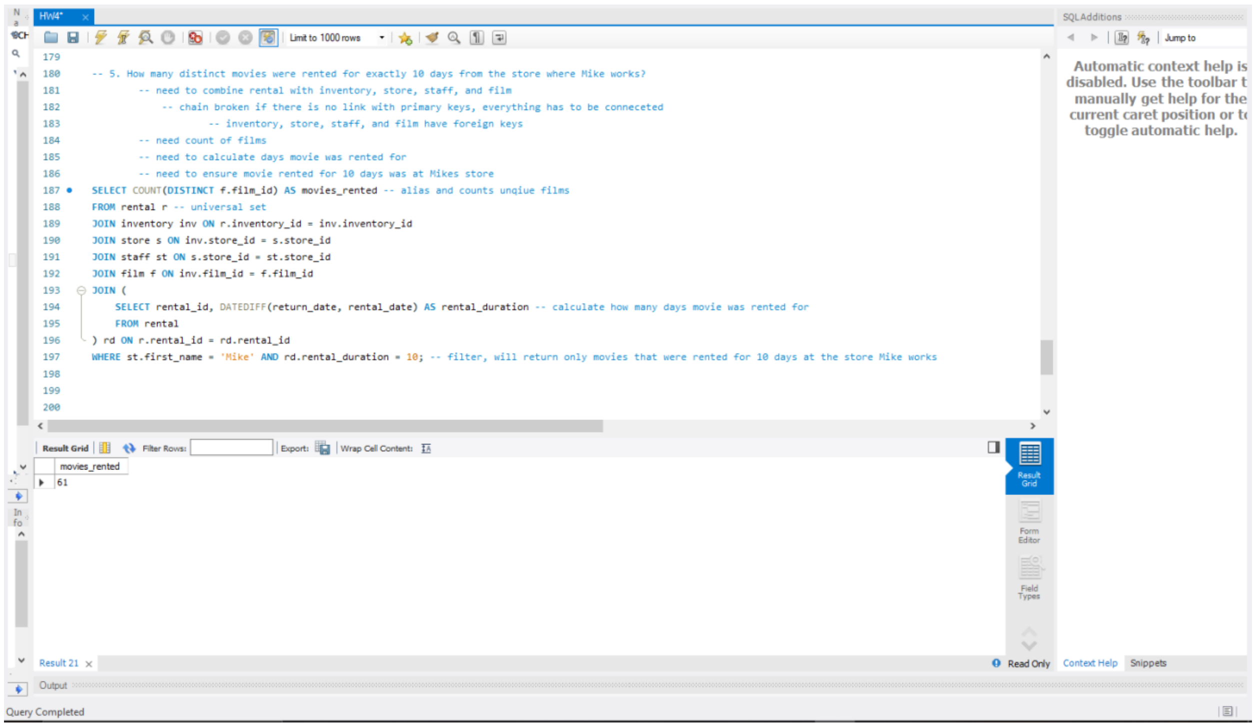
Query Completed

What does query do? How does it solve the problem?

This query creates a CTEs to be able to determine which actor has appeared in the most English-language films by joining actor with film_actor, film, and language. Select is important for determining which actor has the most appearances in these types of films. Comments on SQL queries also better define what codes represent.

The query nicely lays out the actor_id and their respective first and last name. It will only return the actor with the highest amount of appearances

5. **How many distinct movies were rented for exactly 10 days from the store where Mike works?**



```sql
-- 5. How many distinct movies were rented for exactly 10 days from the store where Mike works?
        -- need to combine rental with inventory, store, staff, and film
            -- chain broken if there is no link with primary keys, everything has to be conneceted
                    -- inventory, store, staff, and film have foreign keys
        -- need count of films
        -- need to calculate days movie was rented for
        -- need to ensure movie rented for 10 days was at Mikes store
SELECT COUNT(DISTINCT f.film_id) AS movies_rented -- alias and counts unqiue films
FROM rental r -- universal set
JOIN inventory inv ON r.inventory_id = inv.inventory_id
JOIN store s ON inv.store_id = s.store_id
JOIN staff st ON s.store_id = st.store_id
JOIN film f ON inv.film_id = f.film_id
JOIN (
    SELECT rental_id, DATEDIFF(return_date, rental_date) AS rental_duration -- calculate how many days movie was rented for
    FROM rental
) rd ON r.rental_id = rd.rental_id
WHERE st.first_name = 'Mike' AND rd.rental_duration = 10; -- filter, will return only movies that were rented for 10 days at the store Mike works
```

```
movies_rented
61
```

Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.

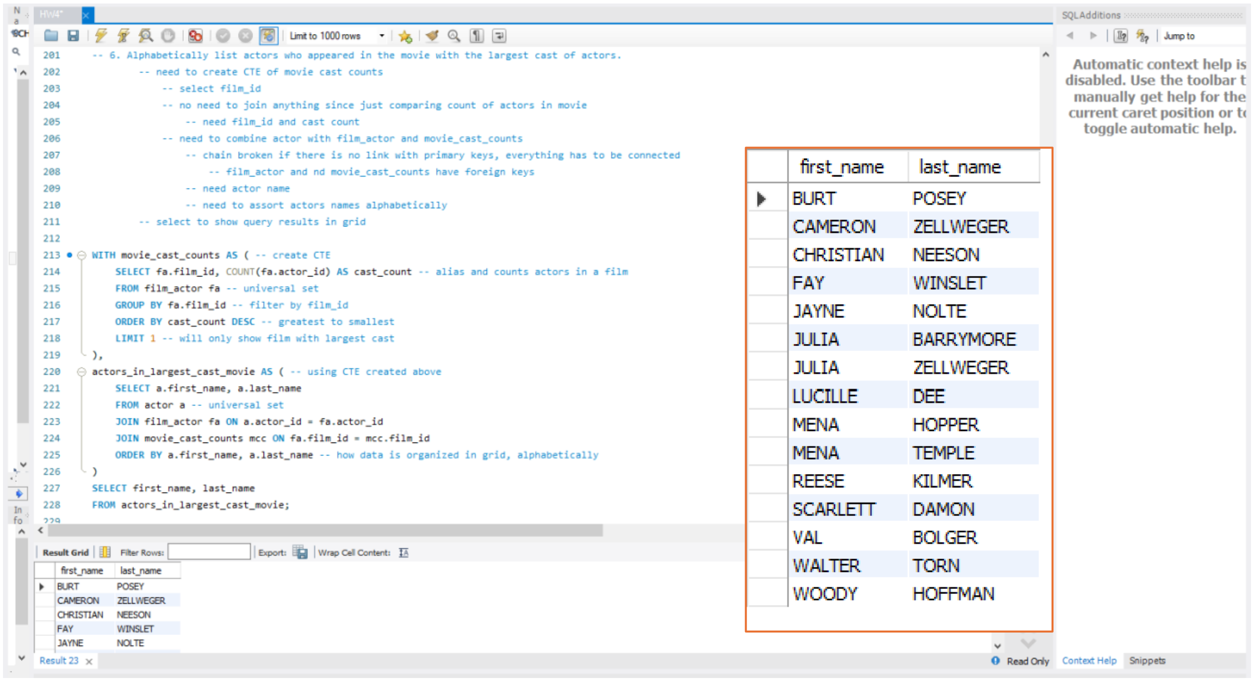
Query Completed

What does query do? How does it solve the problem?

This query can join rental with inventory, store, staff, and film by utilizing their primary keys and or foreign keys. Inner joins are for exactly which columns we need and are using. Comments on SQL queries also better define what codes represent.

The query nicely lays out the movies rented for exactly 10 days from the store Mike works at.

6. **Alphabetically list actors who appeared in the movie with the largest cast of actors.**



```sql
-- 6. Alphabetically list actors who appeared in the movie with the largest cast of actors.
      -- need to create CTE of movie cast counts
         -- select film_id
         -- no need to join anything since just comparing count of actors in movie
            -- need film_id and cast count
         -- need to combine actor with film_actor and movie_cast_counts
            -- chain broken if there is no link with primary keys, everything has to be connected
               -- film_actor and nd movie_cast_counts have foreign keys
            -- need actor name
            -- need to assort actors names alphabetically
      -- select to show query results in grid

WITH movie_cast_counts AS ( -- create CTE
    SELECT fa.film_id, COUNT(fa.actor_id) AS cast_count -- alias and counts actors in a film
    FROM film_actor fa -- universal set
    GROUP BY fa.film_id -- filter by film_id
    ORDER BY cast_count DESC -- greatest to smallest
    LIMIT 1 -- will only show film with largest cast
),
actors_in_largest_cast_movie AS ( -- using CTE created above
    SELECT a.first_name, a.last_name
    FROM actor a -- universal set
    JOIN film_actor fa ON a.actor_id = fa.actor_id
    JOIN movie_cast_counts mcc ON fa.film_id = mcc.film_id
    ORDER BY a.first_name, a.last_name -- how data is organized in grid, alphabetically
)
SELECT first_name, last_name
FROM actors_in_largest_cast_movie;
```

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

What does query do? How does it solve the problem?

This query creates a CTEs to be able to calculate a movies cast count by calculating the amount of actors in each film. Inner joins are utilized for joining exactly which columns we need and are using. It uses the CTE to determine the names of actors in the largest cast. Select is important to return the first and last name of the actors in the largest movie cast to be returned. Comments on SQL queries also better define what codes represent.

The query nicely lays out the first and last name in alphabetical order of the actors in the movie with the largest cast.