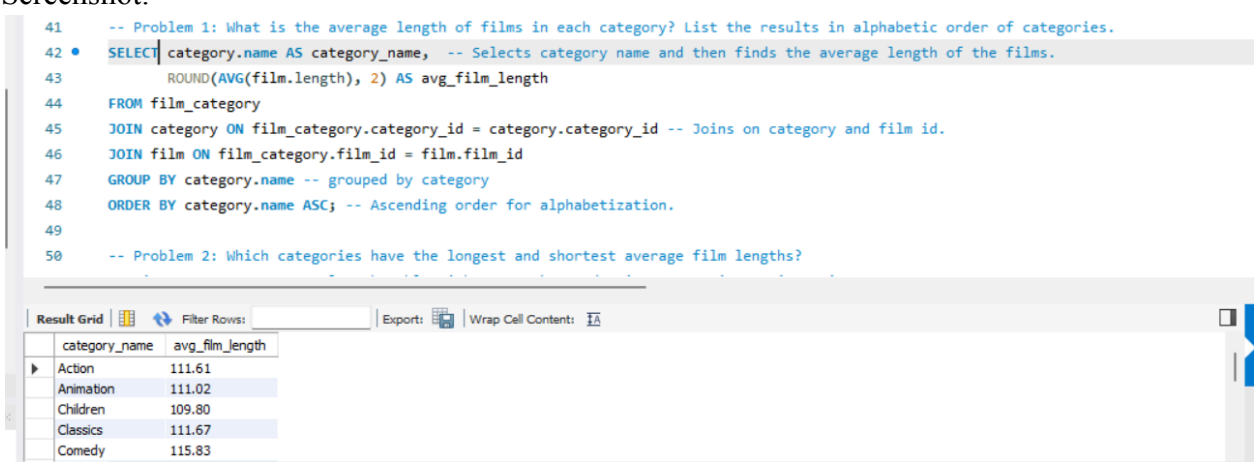
James Black

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

Query:
SELECT category.name AS category_name, -- Selects category name and then finds the average length of the films.
    ROUND(AVG(film.length), 2) AS avg_film_length
FROM film_category
JOIN category ON film_category.category_id = category.category_id -- Joins on category and film id.
JOIN film ON film_category.film_id = film.film_id
GROUP BY category.name -- grouped by category
ORDER BY category.name ASC; -- Ascending order for alphabetization.

Explanation:
First selects category name and finds the average film length by using the AVG function and rounding the result to two decimal places. Joins on category_id and film_id. Grouped by category name and ordered in ascending order for alphabetization.

Screenshot:



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

Query:
WITH avg_length AS (
  SELECT category.name AS category_name, -- Selects category and averages film length.
      ROUND(AVG(film.length), 2) AS avg_film_length
  FROM film_category

```
   JOIN category ON film_category.category_id = category.category_id -- Joins on category_id
and film_id
   JOIN film ON film_category.film_id = film.film_id
   GROUP BY category.name
)
-- Select the categories with the longest and shortest average lengths from the table avg_length
SELECT category_name, avg_film_length
FROM avg_length
WHERE avg_film_length = (SELECT MAX(avg_film_length) FROM avg_length)
  OR avg_film_length = (SELECT MIN(avg_film_length) FROM avg_length);
```

Explanation:
First a with statement is used to create a temporary table composed of the average lengths of the films, then essentially does the same process as problem one, without alphabetizing the tables. Then, the main query selects the category name, and the average film length from the table average length. It gets the longest and shortest length films by using a set of conditional statements that select the max and min length from the table.

Screenshot:

```
49
50      -- Problem 2: Which categories have the longest and shortest average film lengths?
51      -- First create a average length table with a WITH keyword, since we won't use it again.
52  WITH avg_length AS (
53      SELECT category.name AS category_name,  -- Selects category and averages film length.
54             ROUND(AVG(film.length), 2) AS avg_film_length
55      FROM film_category
56      JOIN category ON film_category.category_id = category.category_id -- Joins on category_id and film_id
57      JOIN film ON film_category.film_id = film.film_id
58      GROUP BY category.name
59  )
```

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

category_name	avg_film_length
Sports	128.20
Sci-Fi	108.20

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

Query:

```
WITH action_customers AS (
   -- Find customers who rented Action movies
   SELECT DISTINCT customer.customer_id, customer.first_name, customer.last_name
   FROM customer
   JOIN rental ON customer.customer_id = rental.customer_id -- Joins
   JOIN inventory ON rental.inventory_id = inventory.inventory_id
   JOIN film ON inventory.film_id = film.film_id
   JOIN film_category ON film.film_id = film_category.film_id
   JOIN category ON film_category.category_id = category.category_id
   WHERE category.name = 'Action' -- Conditional statement to find action.
```

```
),

excluded_customers AS (
   -- Find customers who rented Comedy or Classics
   SELECT DISTINCT customer.customer_id
   FROM customer
   JOIN rental ON customer.customer_id = rental.customer_id -- Joins
   JOIN inventory ON rental.inventory_id = inventory.inventory_id
   JOIN film ON inventory.film_id = film.film_id
   JOIN film_category ON film.film_id = film_category.film_id
   JOIN category ON film_category.category_id = category.category_id
   WHERE category.name IN ('Comedy', 'Classics') -- Comedy or classic conditional.
)

-- Select customers who rented Action movies but not Comedy or Classics
SELECT action_customers.customer_id, action_customers.first_name,
action_customers.last_name
FROM action_customers
LEFT JOIN excluded_customers ON action_customers.customer_id =
excluded_customers.customer_id -- Left Join to keep null values for excluded customers.
WHERE excluded_customers.customer_id IS NULL;
```

Explanation:
Using a with statement to create two temporary tables, first the action table is created by selecting customers from the customer table where the category id of the rental is action. Join on inventory_id, category_id, film_id, and customer_id. The second table, designed to show the customers to be excluded from the results, follows the same process, however it is slightly streamlined in the select statements to only include the id since all the other aspects will not be needed in the final joining of the tables. The other difference is that the conditional screens the rentals for the rental involving comedy or classic movies. The main query selects the customer's id, first name, and last name from the action table. A left join is used to preserve the null values in the excluded customers table, and lastly a conditional is used to find where the customer_id is null in the excluded table.

Screenshot:

```
-- SATA Customers 2.

WITH action_customers AS (
    -- Find customers who rented Action movies
    SELECT DISTINCT customer.customer_id, customer.first_name, customer.last_name
    FROM customer
    JOIN rental ON customer.customer_id = rental.customer_id -- Joins
    JOIN inventory ON rental.inventory_id = inventory.inventory_id
    JOIN film ON inventory.film_id = film.film_id
```

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

customer_id	first_name	last_name
139	AMBER	DIXON
350	JUAN	FRALEY
17	DONNA	THOMPSON
452	TOM	MILNER
232	CONSTANCE	REID

Result 9

Problem 4:Which actor has appeared in the most English-language movies?

Query:
SELECT actor.actor_id, actor.first_name, actor.last_name, COUNT(film.film_id) AS movie_count
FROM actor
JOIN film_actor ON actor.actor_id = film_actor.actor_id -- Joins
JOIN film ON film_actor.film_id = film.film_id
JOIN language ON film.language_id = language.language_id
WHERE language.name = 'English'
GROUP BY actor.actor_id, actor.first_name, actor.last_name
ORDER BY movie_count DESC;

Explanation:
Selects the actor id, first name, last name, and then selects the amount of films that the actor has appeared in from the actor table. Tables are joined on actor_id, film_id, and language_id to allow the conditional to be used that displays films with the language of English. The counts are grouped by the attributes of the actor, to ensure that each part is unique, and are ordered in descending order to go from the actor with the most appearances to the actor in the least. It can be adjusted with a LIMIT 1; statement if the user wants specifically only the actor who has appeared in the most English speaking movies.

Screenshot:

```sql
-- Problem 4: Which actor has appeared in the most English-language movies?
-- Find first the counts for who has appeared in the most movies grouped by language, then row number 1,  then select actor.
SELECT actor.actor_id, actor.first_name, actor.last_name, COUNT(film.film_id) AS movie_count
FROM actor
JOIN film_actor ON actor.actor_id = film_actor.actor_id -- Joins
JOIN film ON film_actor.film_id = film.film_id
JOIN language ON film.language_id = language.language_id
WHERE language.name = 'English'
GROUP BY actor.actor_id, actor.first_name, actor.last_name
ORDER BY movie_count DESC;
```

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

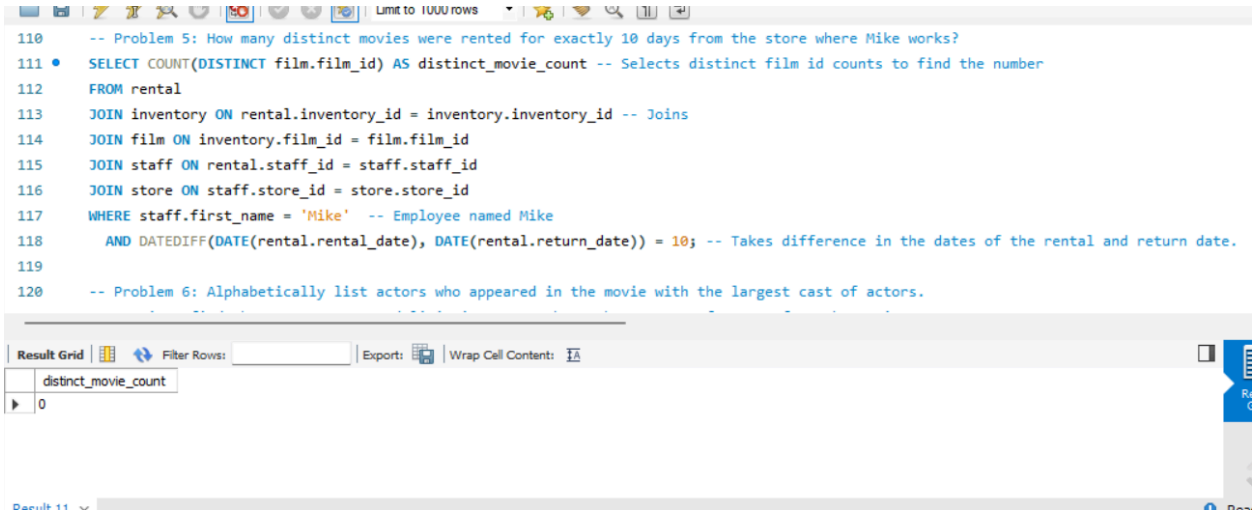
actor_id	first_name	last_name	movie_count
107	GINA	DEGENERES	42
102	WALTER	TORN	41
198	MARY	KEITEL	40
181	MATTHEW	CARREY	39
23	SANDRA	KILMER	37

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

Query:
SELECT COUNT(DISTINCT film.film_id) AS distinct_movie_count -- Selects distinct film id counts to find the number
FROM rental
JOIN inventory ON rental.inventory_id = inventory.inventory_id -- Joins
JOIN film ON inventory.film_id = film.film_id
JOIN staff ON rental.staff_id = staff.staff_id
JOIN store ON staff.store_id = store.store_id
WHERE staff.first_name = 'Mike' -- Employee named Mike
  AND DATEDIFF(DATE(rental.rental_date), DATE(rental.return_date)) = 10; -- Takes difference in the dates of the rental and return date.

Explanation:
First selects the distinct film id's, then adds up the counts, from rental. Joins inventory_id, film_id, staff_id, and store_id. Conditional statements where the staff first name is Mike and a date difference statement that finds the total difference between the rental date and the return date.

Screenshot:



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

Query:

```
WITH movie_actor_counts AS (
   SELECT film.film_id, COUNT(film_actor.actor_id) AS actor_count
   FROM film
   JOIN film_actor ON film.film_id = film_actor.film_id
   GROUP BY film.film_id -- Grouped and ordered to put the films in descending order, then
limit shows only the number one film in the list.
   ORDER BY actor_count DESC
   LIMIT 1
)

-- Select actor names and display them in alphabetical order.
SELECT actor.first_name, actor.last_name
FROM actor
JOIN film_actor ON actor.actor_id = film_actor.actor_id -- Joins
JOIN movie_actor_counts ON film_actor.film_id = movie_actor_counts.film_id
ORDER BY actor.last_name, actor.first_name; -- Order by actors' names. Alphabetically focuses
on last name first
```

Explanation:

First a temporary table for counting the number of actors in each movie is created using the with statement. This table selects the film id, takes the count of actors in each film, with a join on film_id. The actors are grouped by film_id and are ordered in descending order with a LIMIT 1 statement to have the table only contain the film with the most actors. In the main statement, the first and last names of the actors are selected from the actor table, with natural joins on the film_id and actor_id to ensure only the actors that are present in the film with the most actors are shown. Lastly they are ordered with last name and first name alphabetically.

Screenshot:

```sql
-- Problem 6: Alphabetically list actors who appeared in the movie with the largest cast of actors.
-- -- First find the actor counts and limit it to 1 to have the one set of actors from the movie.
WITH movie_actor_counts AS (
    SELECT film.film_id, COUNT(film_actor.actor_id) AS actor_count
    FROM film
    JOIN film_actor ON film.film_id = film_actor.film_id
    GROUP BY film.film_id -- Grouped and ordered to put the films in descending order, then limit shows only the number one film in the list
    ORDER BY actor_count DESC
    LIMIT 1
)
```

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

first_name	last_name
JULIA	BARRYMORE
VAL	BOLGER
SCARLETT	DAMON
LUCILLE	DEE
WOODY	HOFFMAN

Result Grid