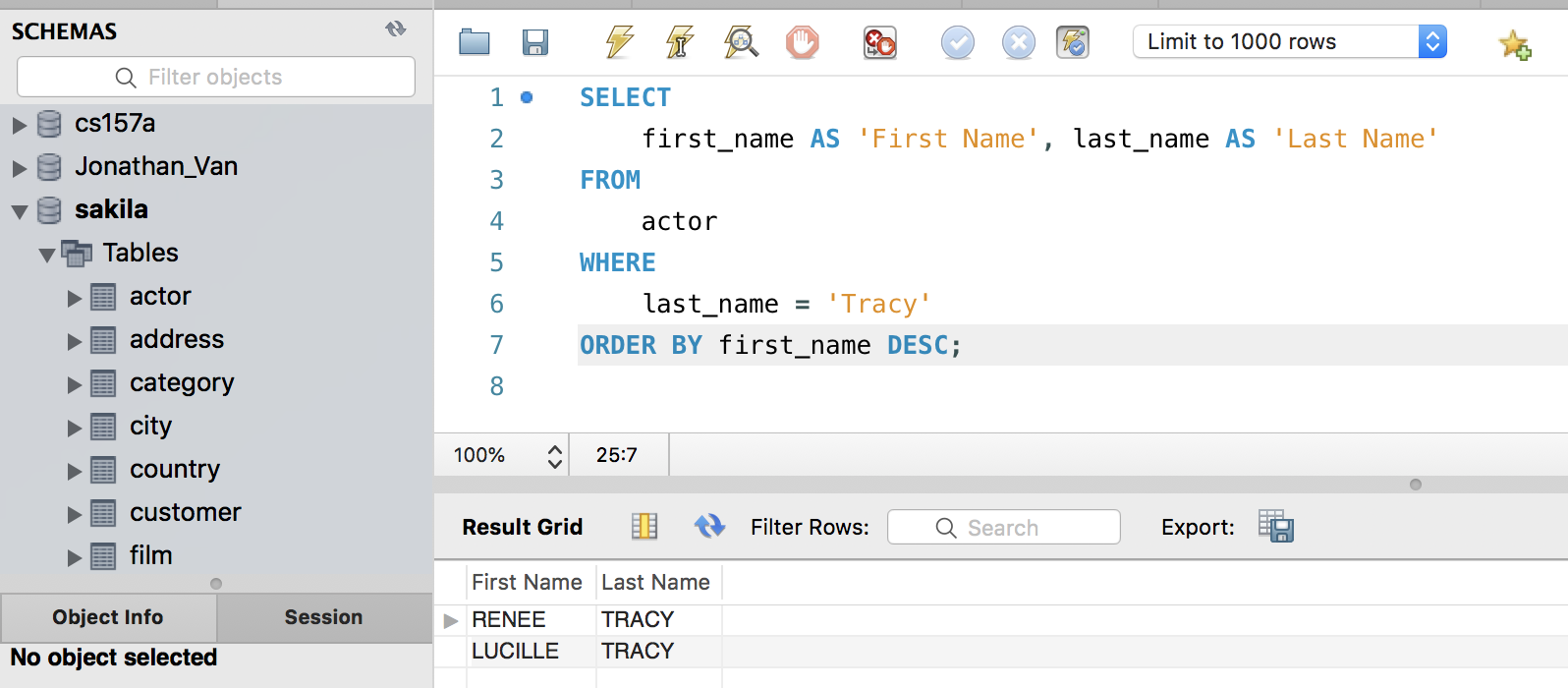
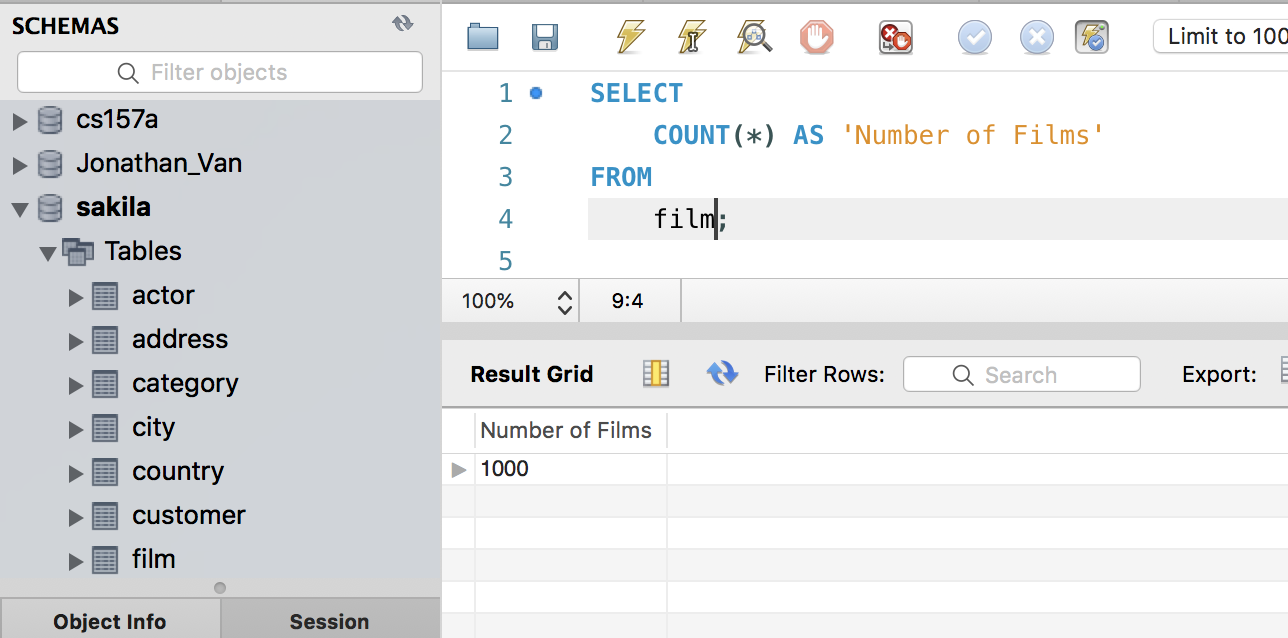
Questions:

1. (6%) Display the actor’s name (first\_name, last\_name) using alias name "First Name", "Last Name" whose last name is “Tracy” and display in descending order based on First Name.
   * I assumed the tables had those columns, and that the user was already in the database, and that case did not matter. (TRACY == tracy)I declared attributes whose data I wanted, reassigned the attribute names using “AS”, and selected the actor table, and I set a condition where it should be “Tracy” for the last name, and said to order it by the first name using DESC to indicate the descending order. I’m assuming using “AS” for the column names and that I was to retrieve it from actors table.

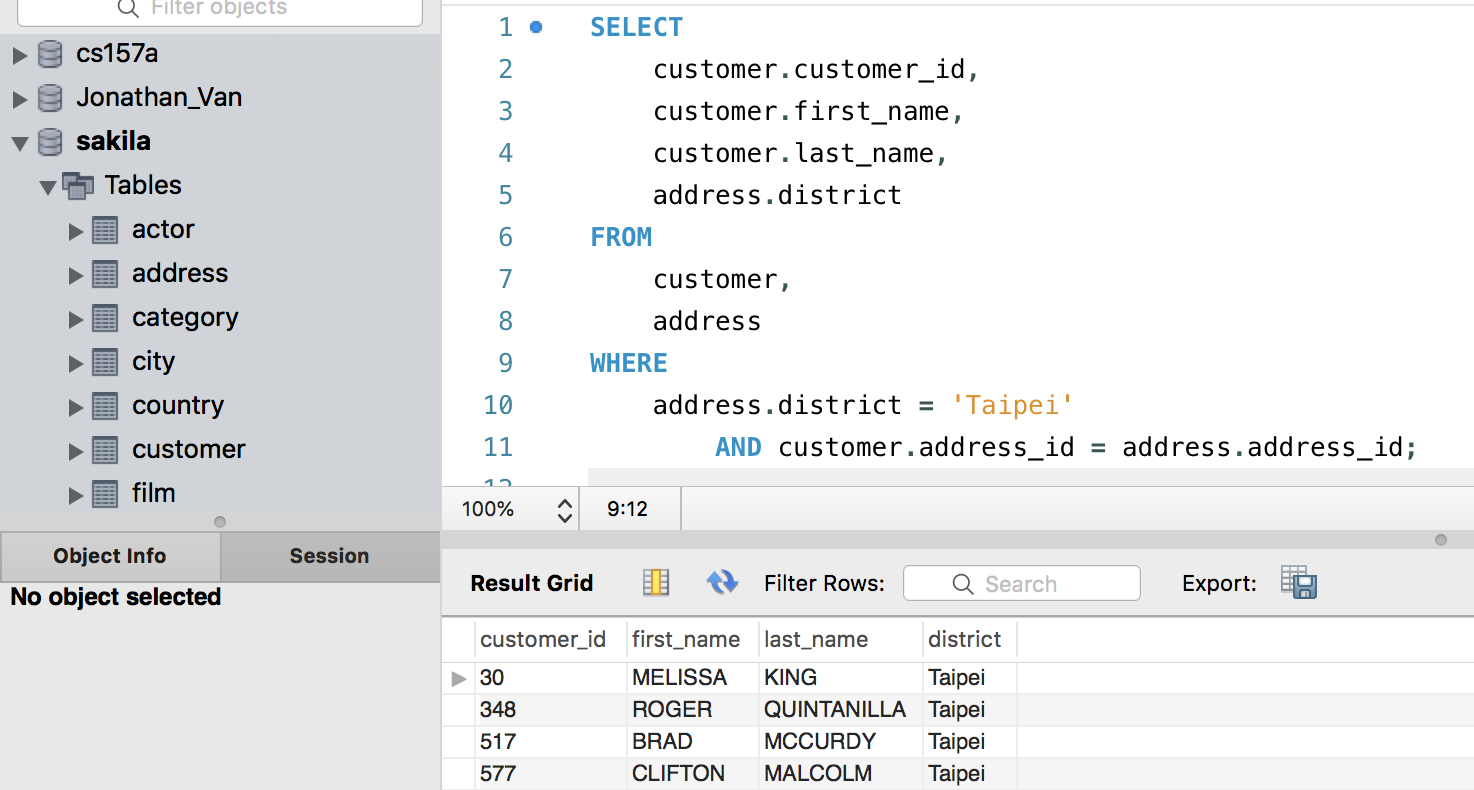
****

1. (5%) How many films in the database? Display the following exact screenshot.

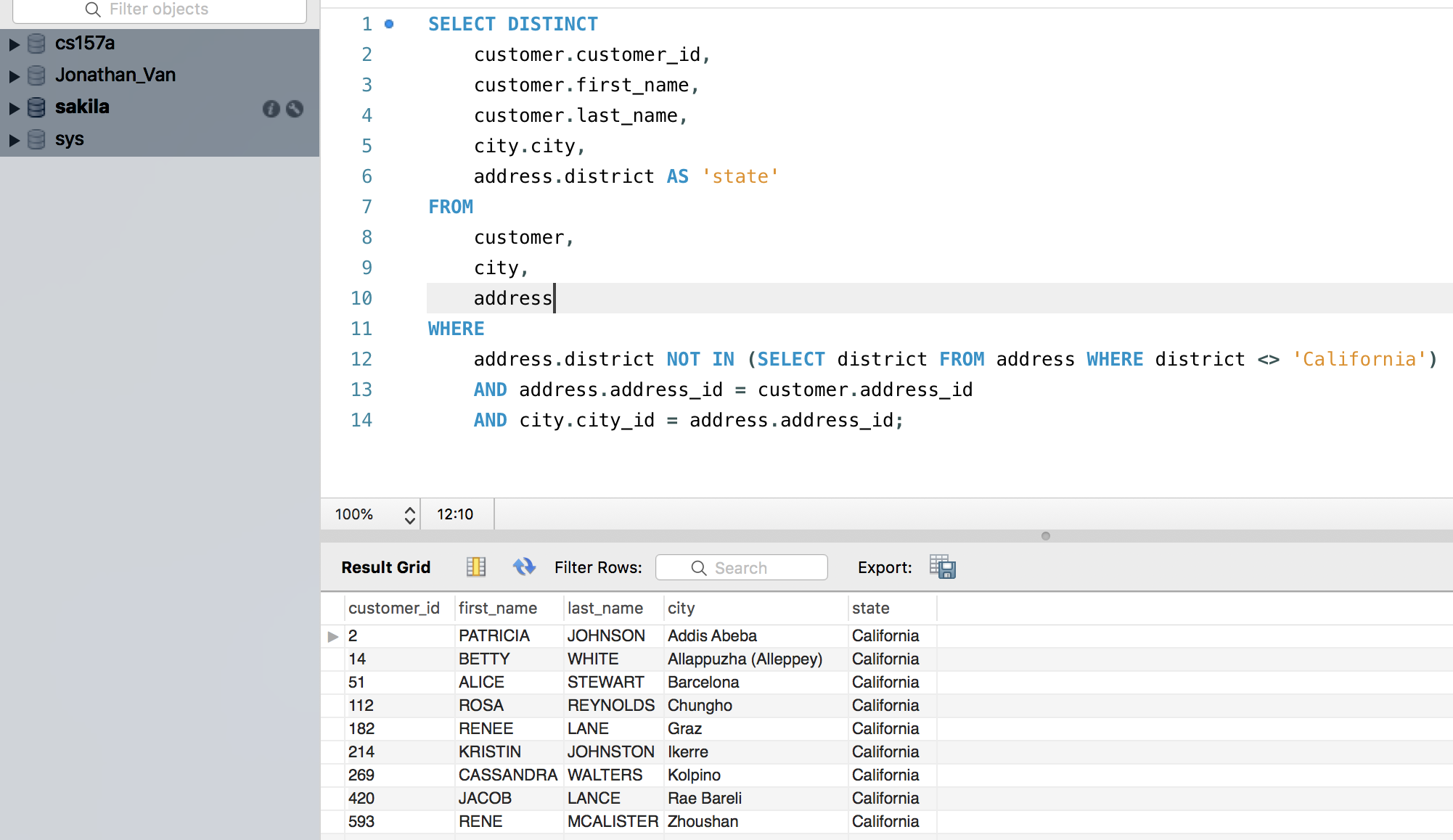


There are 1000 films according to the query, I basically used a select to get the count of all the tuples from the film table without any conditions. I assumed that there was data in the film table and that film table existed. I just assumed you were just counting all the films w/o conditions. What it does is that selects the count of all the tuples from the film table because there were no filters placed, and I used “AS” to replace the COUNT category with “Number of Films”.

1. (6%) Write a SQL statement to display customer\_id, first\_name, last\_name, and district information of all customers, who are living in “Taipei”.

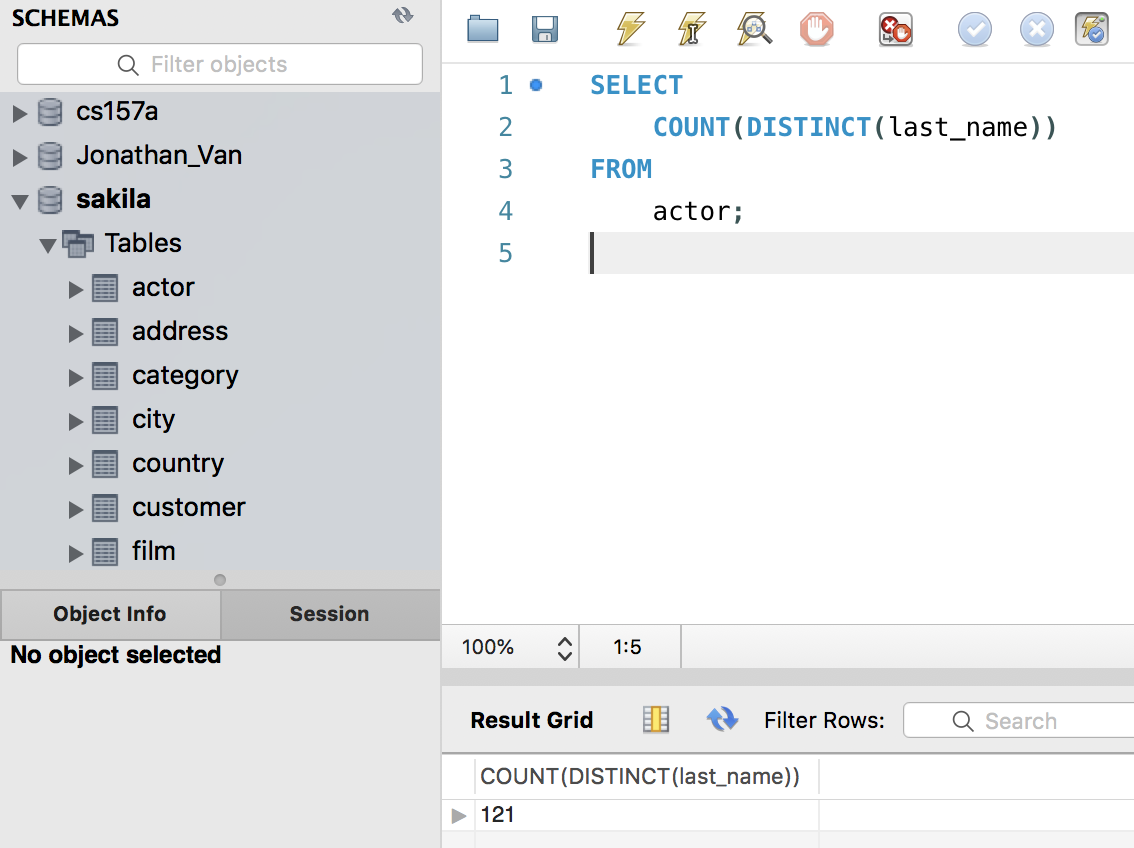
****I’m assuming we are using the sakila database again, and that the tables, as a well as their columns existed. I assumed we were to combine the customer and address tables together. And that when they registered the customer, the address was added appropriately. What it does is that it gets the tuple of the customer table’s id, first\_name, and last\_name, and the address table’s district where the address id found in the customer’s table, matches the tuple containing the address id found in the address table, and must have that address tuple’s district be “Taipei” .

1. (8%) Using NOT IN operator and subquery to write a query to produce a list of customer\_id, first\_name, last\_name, city, state of all customers who live in California.



I’m assuming that the person who asked this is an American, where states is used instead of districts, I’m going to have the column named state instead of district im assuming all the tables as well as those tuples exist. What this sql command does is that it selects the customer’s id, first name, last name, and the address’s city and district(shown as state), where the condition is that the district/state is not inside of a list returned from a subquery where it returned districts from the address table where the district was not California. It will return the tuples where the tuple from the customer table has the same address\_id as in the address table and in that same address\_id’s tuple, it will check that the city\_id of the address table matches in the city table’s city\_id.

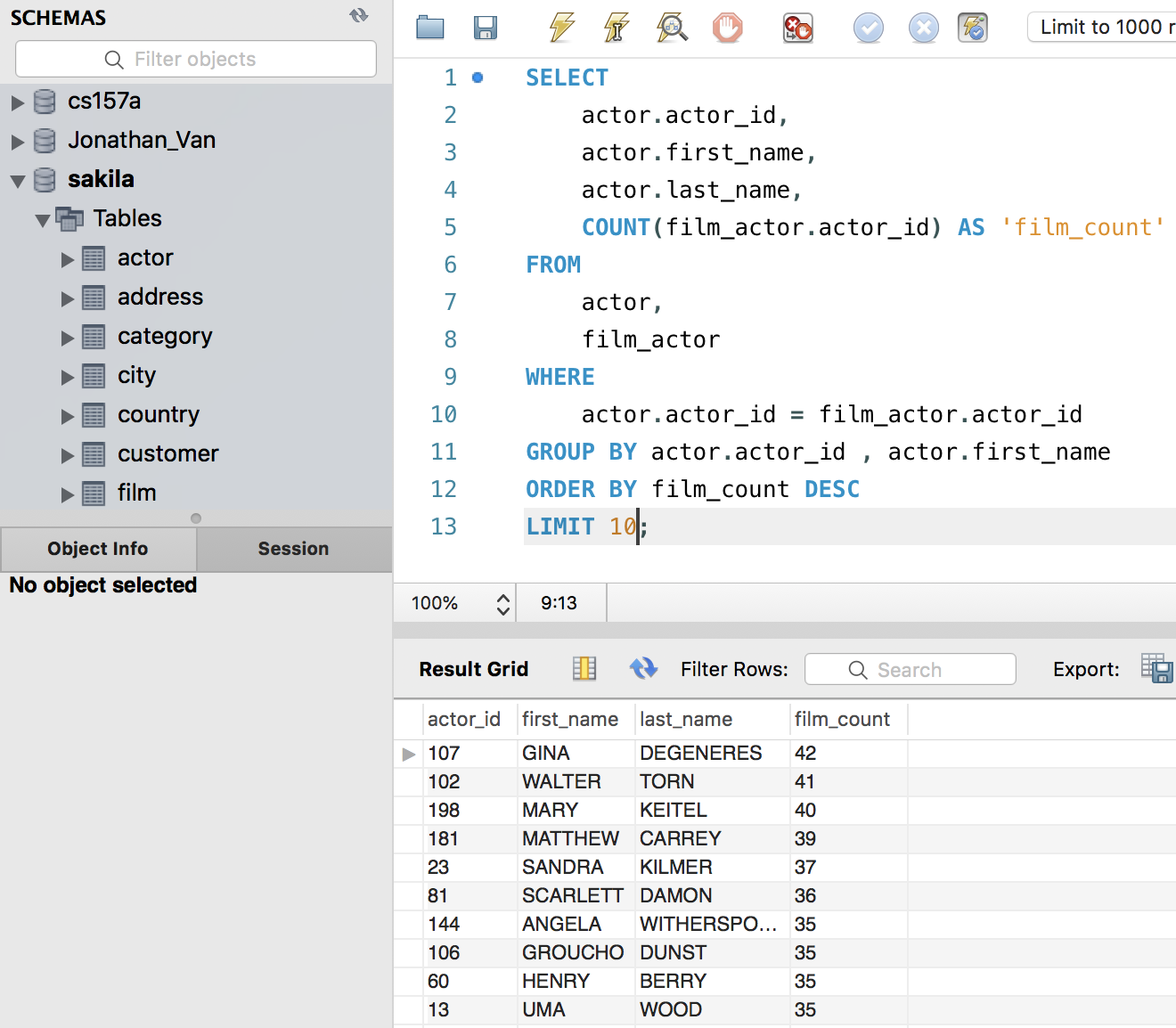
1. (5%) Display the number of distinct actors last names?

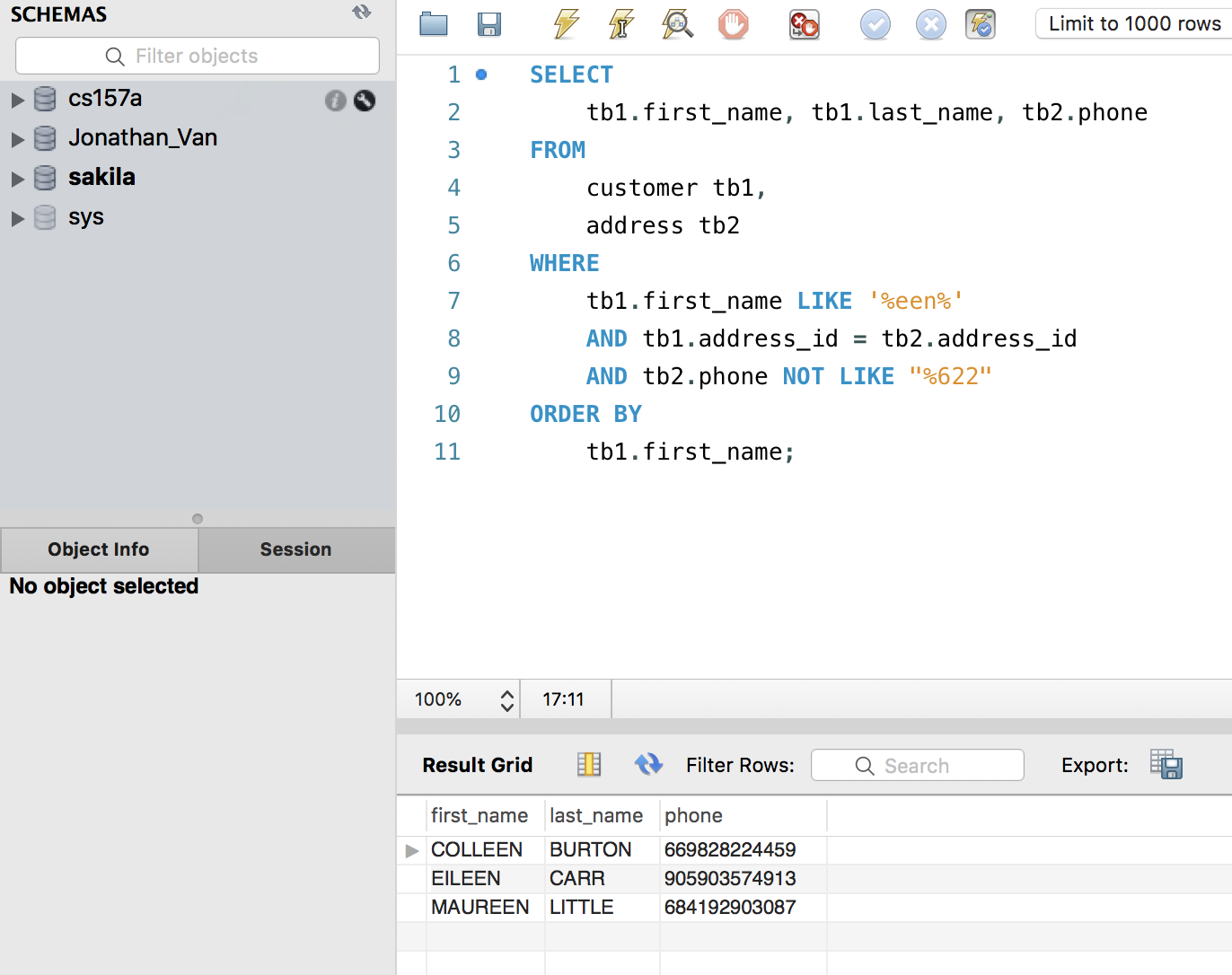


I’m assuming the sakila db is being used, and that it existed, and that there are entries within the actor table. I use the distinct call to get all the distinct last names from the actor table, and the select returns the count of those distinct values

1. (8%) Display the top 10 actors who have mostly appeared in most films as the exact screenshot below: (Hint: try “LIMIT 10” at the end of SQL statement)

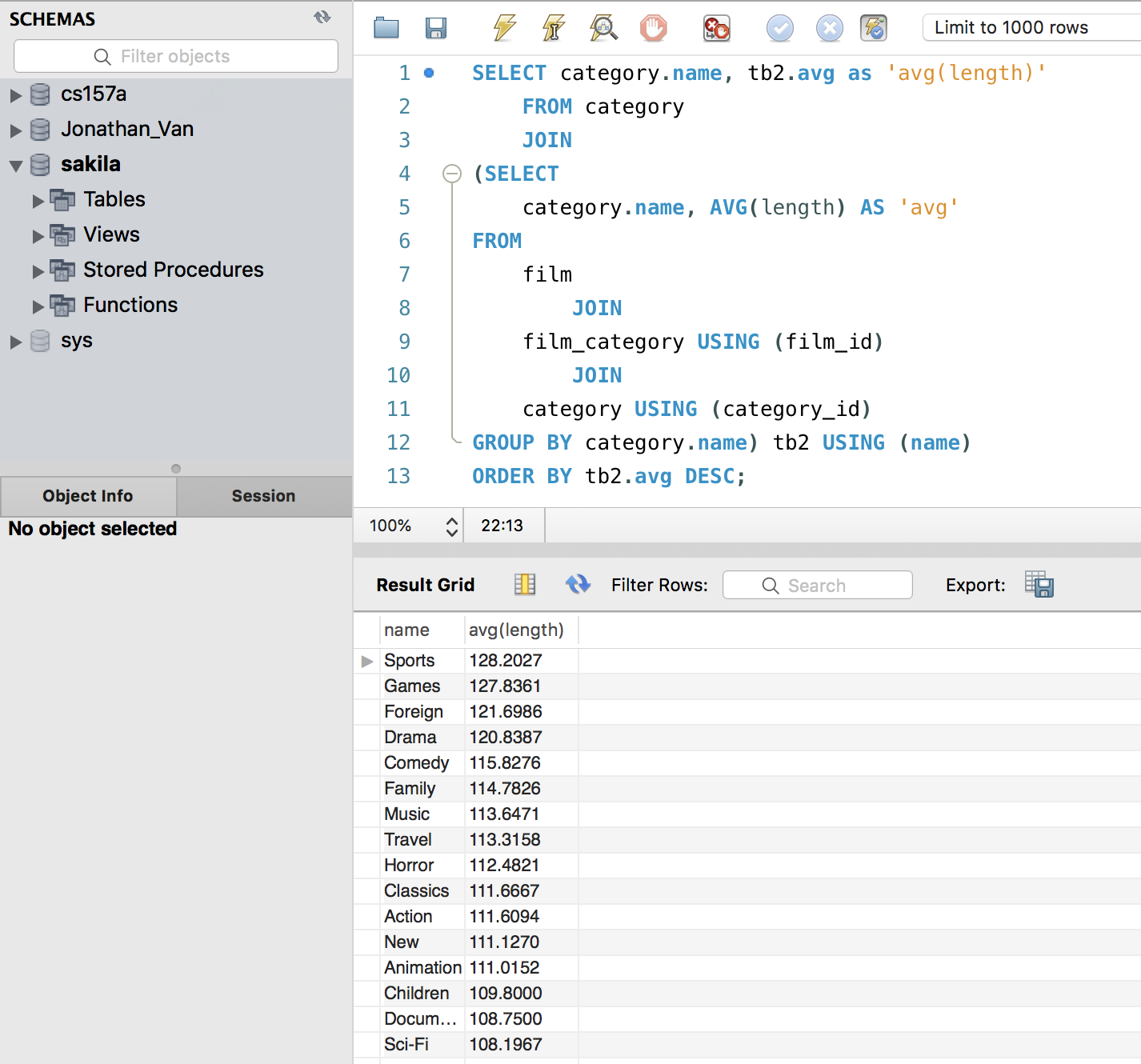
so I’m pretty sure the logic is there. I’m assuming the tables exist, and that the user is using the sakila table like earlier. I’m assuming there are the relevant tuples within the tables.What this command does is that it gets the actor table’s tuple’s id, first name, last name and a film count that I got from the film\_actor table’s id, but it will only display the ones where the actor table’s actor\_id matches the film\_actor table’s actor\_id, afterwards it’ll order in descending order by the film\_ count and the first name because that is what it looks like in the results and then limit it to the first 10. I rename the COUNT() function’s AS film\_count to match the answer.





I’m assuming the tables exist and that they are filled with the correct data, and they are not null for the first name. So what this query does is that it gets the first and last name from the customer table and the phone number from the address table, on the conditions that the first name must have a “een” somewhere, in the first name and that the address id on the customer tuple must match the address id on the address tuple that has the phone number, and then excludes the phone number that end s in “622” using the NOT LIKE syntax. I use LIKE to enable the usage of the % in the %een% check, and then I use NOT LIKE to say that no matter how many characters exist , it cannot end with 622. And then I order the output by the first name of the customer.

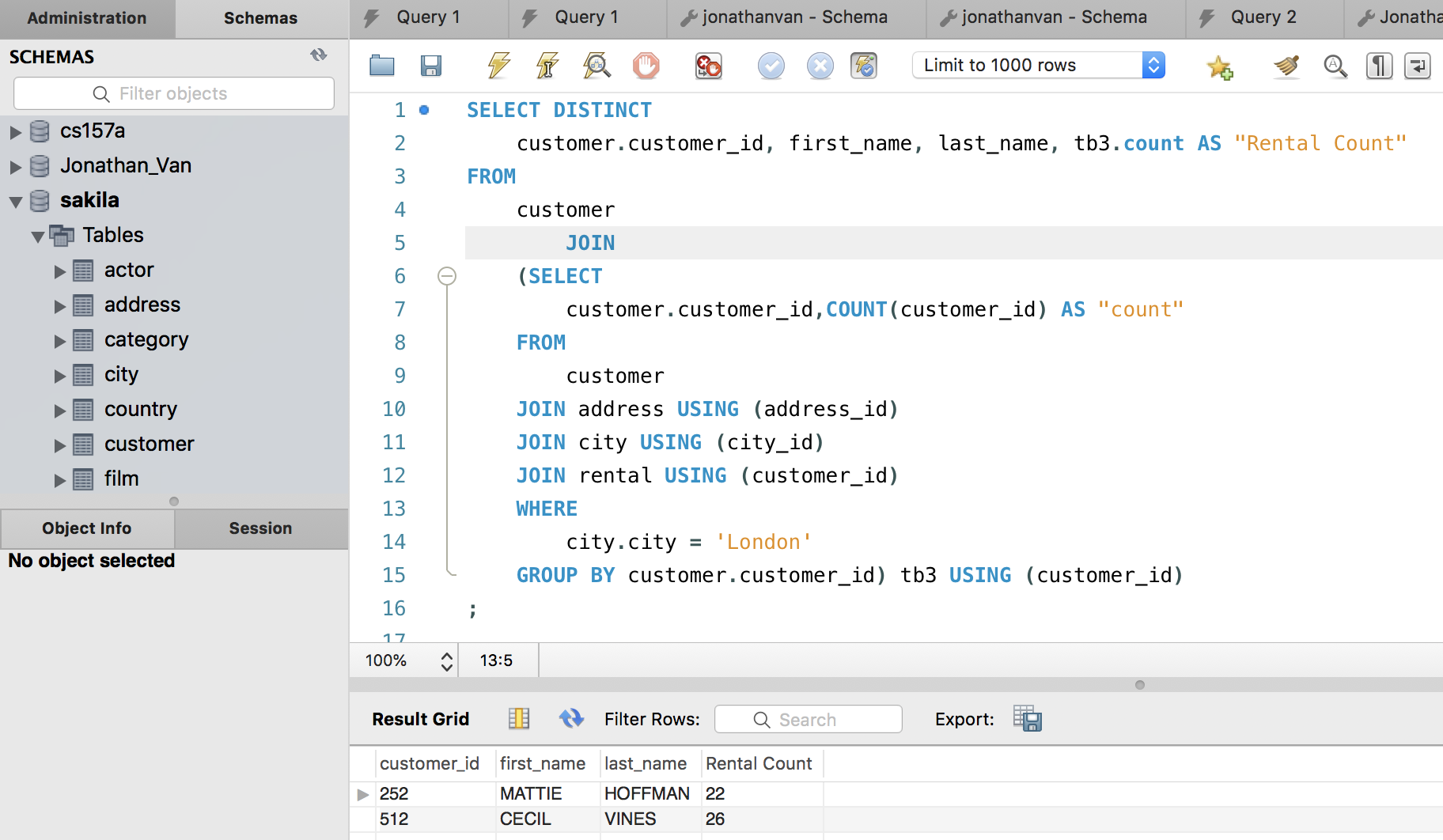
1. (8%) Display the average length of films by category using aggregation and JOIN operator? Must show the exact screenshot below: (hint: in MYSQL, you may use the syntax for NATURAL JOIN operator such as “FROM film JOIN film\_category USING (film\_id)”)



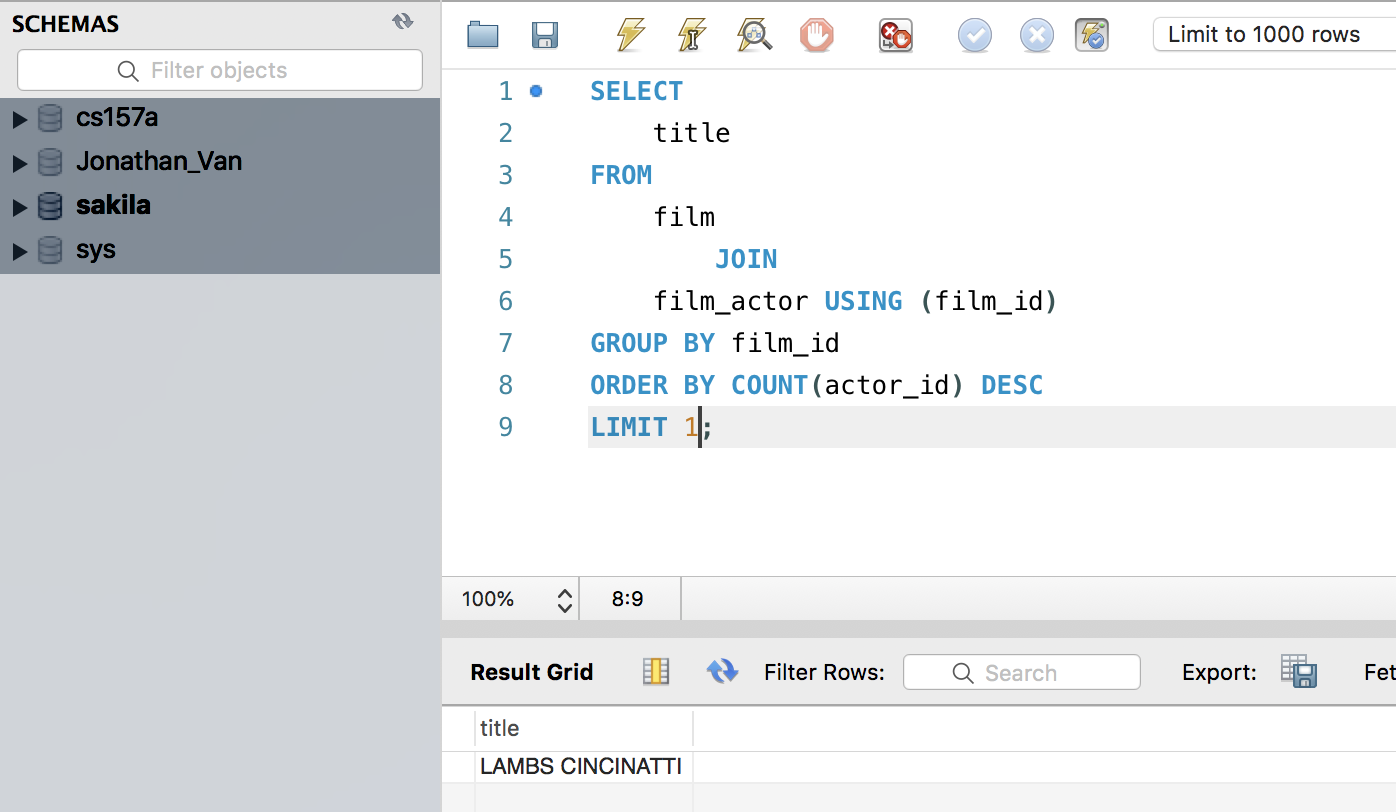
I assumed we were in the sakila database and that all the information was up to date and that I was allowed to use subqueries. I also assumed that it was okay to just rename the column to be ‘avg(length’) because I could not preserve the order if I put the order by outside. Basically, I have a subquery that pre processes the averages of the corresponding categories, and then attached a name to each row so they’re accessible by the outer query. I did a natural join with film and film category, using the film\_id, and then a natural join with category and film\_category by using the category\_id, I used a group by with the name so that it was a valid format, and then afterwards, I natural joined the data from the subquery with the category table called in the main query using the name (as they both have that in common) and attached the corresponding avg I found after renaming the column to “avg(length)”, and then had it ordered by descending order based on the average. The AVG() command has a column of AVG(length), but the answer wants avg(length).

1. (10%) Write a query containing a subquery to display the total number of rentals for the customer (unknown before query) who belongs to the city London as the following exact display:

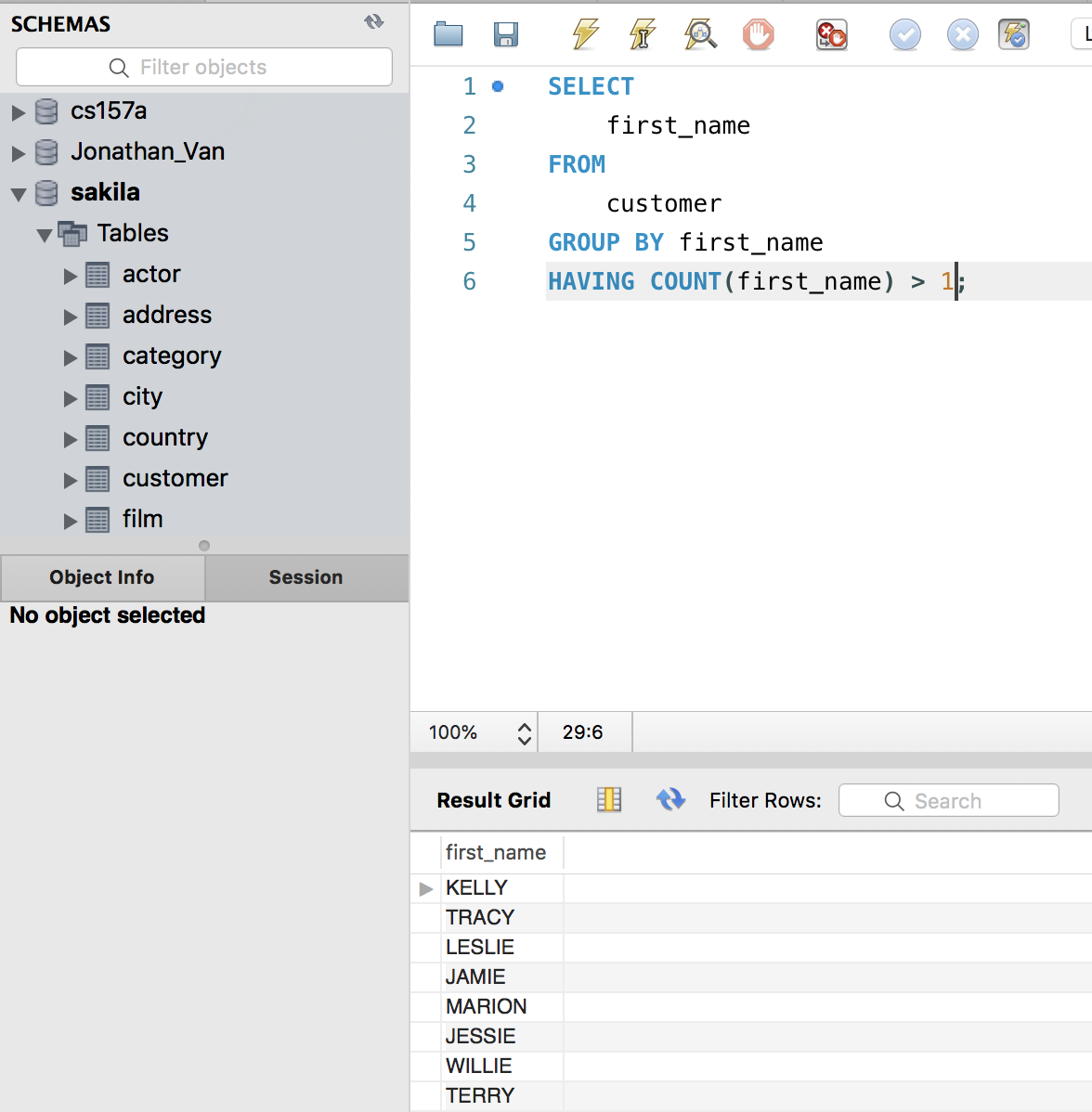
Honestly I was just assuming if the tables had the right/corresponding columns. And that the Sakila database was being used. I was mostly hoping/assuming the data was right, and that it was okay to use city.city instead of city.city\_id, and that to pass it to the query outside of it, I used the subquery to perform the count and grouping instead of having it done on the outside. I’m assuing the natural joins should be on address\_id, city\_id and rental\_id for the subquery. And I passed the customer\_id so that it would have something to naturally join with in the customers\_table. In the subquery, I return the customer id, the count that I got from joining the address,city and rental tables with the customer table where the city is London. I group it by the customer id, which I don’t think matters much. Afterwards, it’ll do a natural join (which I assume to be on the customer.id) and then do a select and returns the id, first name, last name and the count from the table I created.



1. (8%) Display the title of the film with the largest number of actors.

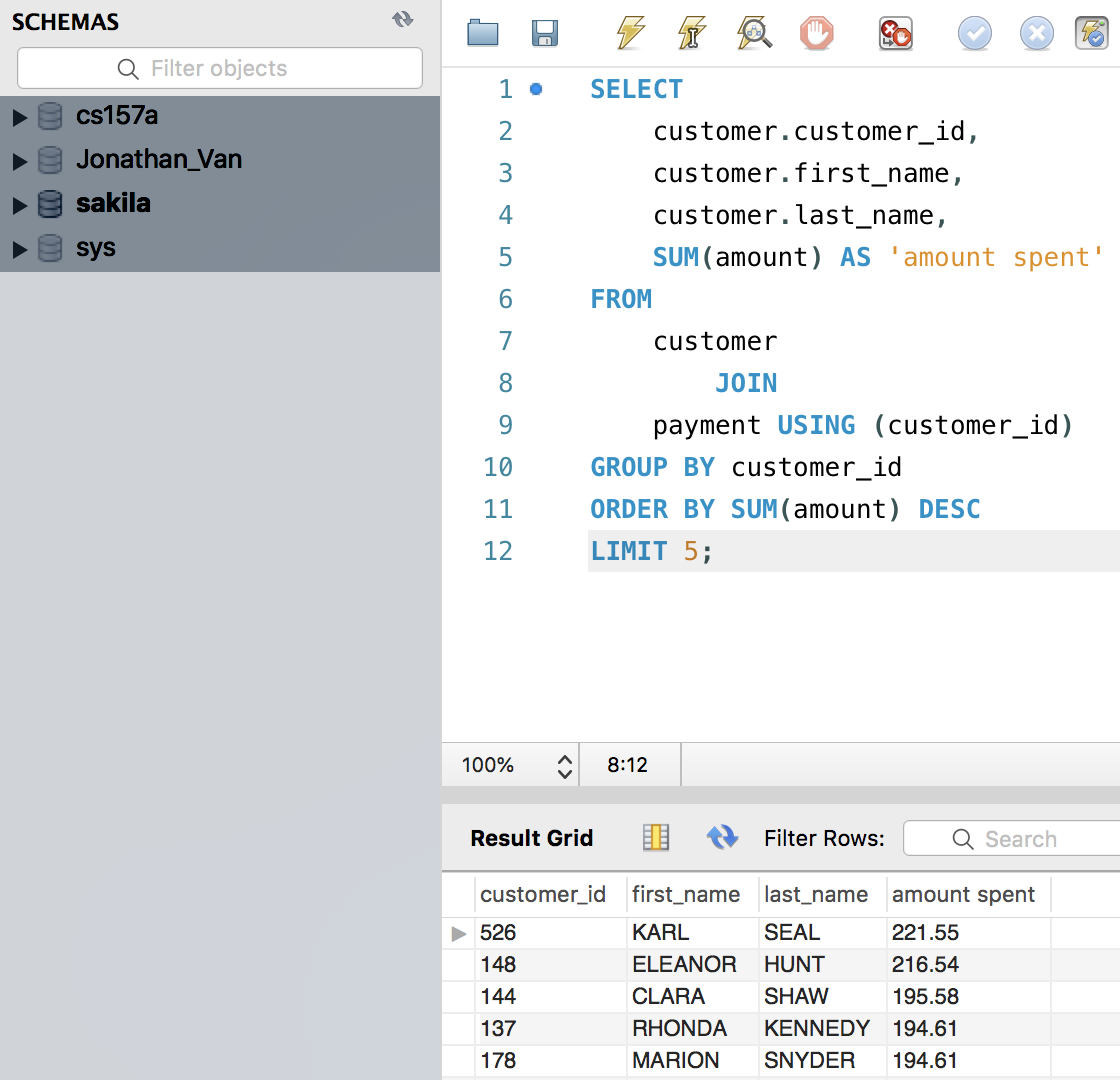
My assumption was to join the film and film\_actors table, and since each tuple in the film\_actor table represents an actor in the movie, I ordered the select by the count of the actor\_id in descending order and then limit it to the first one, that way itll be the “most” and then it’ll return only the title of that tuple, which is technically saying “Get the title of the first movie from a list of movies after sorting them by the count of actors and then sorting it by the count of actors in descending order” .

1. (6%) Using “HAVING” to display which customer’s first names have appeared more than once?

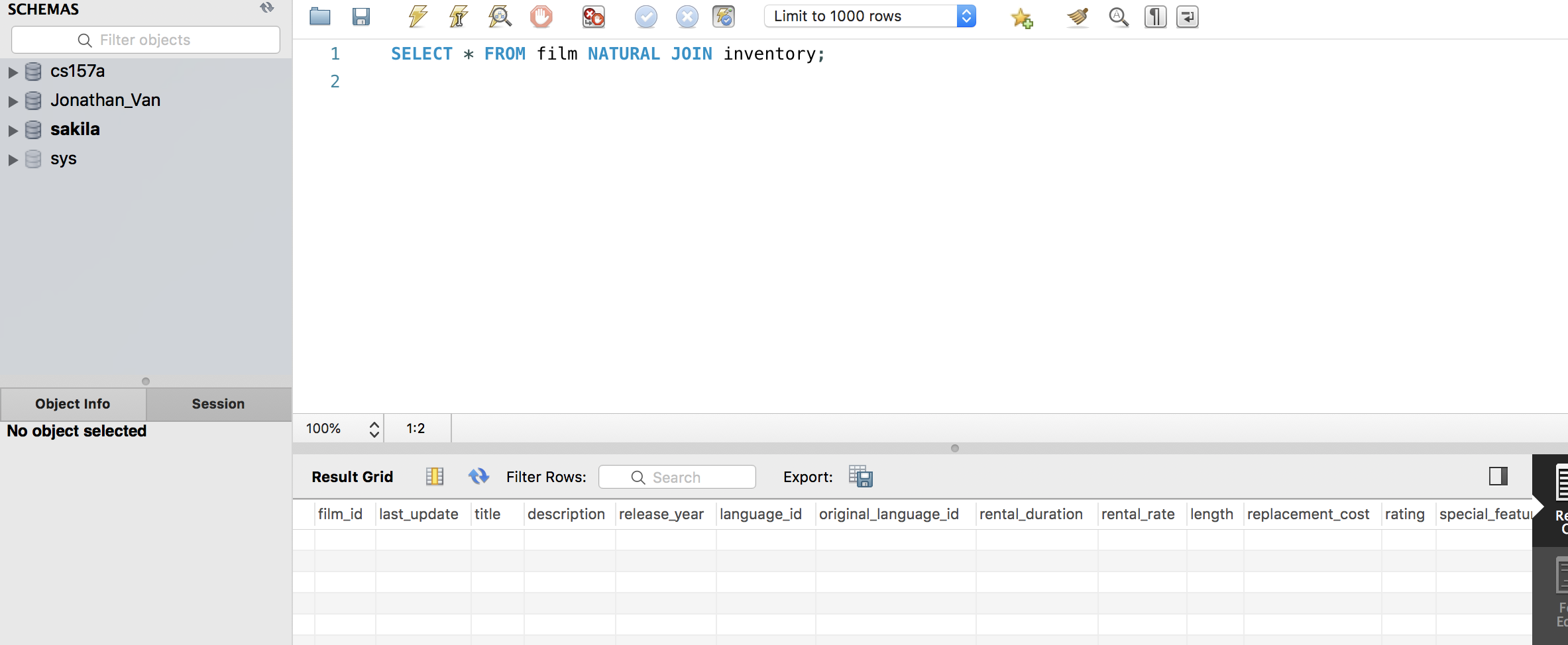
 I assume that I was counting the first\_name using the having and group by. I mean I hope it was okay to use the customer and grouping like this. What this does is that it will select the customer first names from the customers table and will group by customers’s first names where the count of the first name is greater than 1. WHERE clauses cannot be used with aggregation functions.

1. (8%) Query the top 5 customer’s names who spent the most on rental films. Display customer id, name and the amount spent.

I assume that we join the payment and customers table. I join the payment and customer table on customer\_id, and return the first 5 tuple’s id, first name, last name and sum as amount spent after sorting it by that sum in descending order. I group it by customer\_id to support the aggregation. This is technically “top 5 tuples based on sums”



1. (6%) What’s the result of this query? And Why? SELECT \* FROM film NATURAL JOIN inventory;



The result seems to be an empty set, or no tuples that match the query, because when doing natural join, it was comparing the film\_id (which does have a match) between the two tables, AND the last\_update. Even if the film id’s are equal, all of the last\_update dates between each table respectively are different. So it will never return a tuple.

1. (8%) Write an interesting query statement of your own, not shown in the questions. The query should include at least one NATURAL JOIN operator, HAVING clause and aggregation operation. Explain your query statement and the answer.
   * My statement gets the number of rentals, first name and last name of the employee whose staff\_id is 2. It natural joins the rental and staff table on the staff\_id and then only counts the rentals that were made on or before May 26th, 2005. IT groups it by the staff id. So yeah. I use aggregation function with the count and balanced it with the group by and set the parameter/staff id. The answer is Jon Stephens, who rented out 70 items before or on 05-26-2005.

