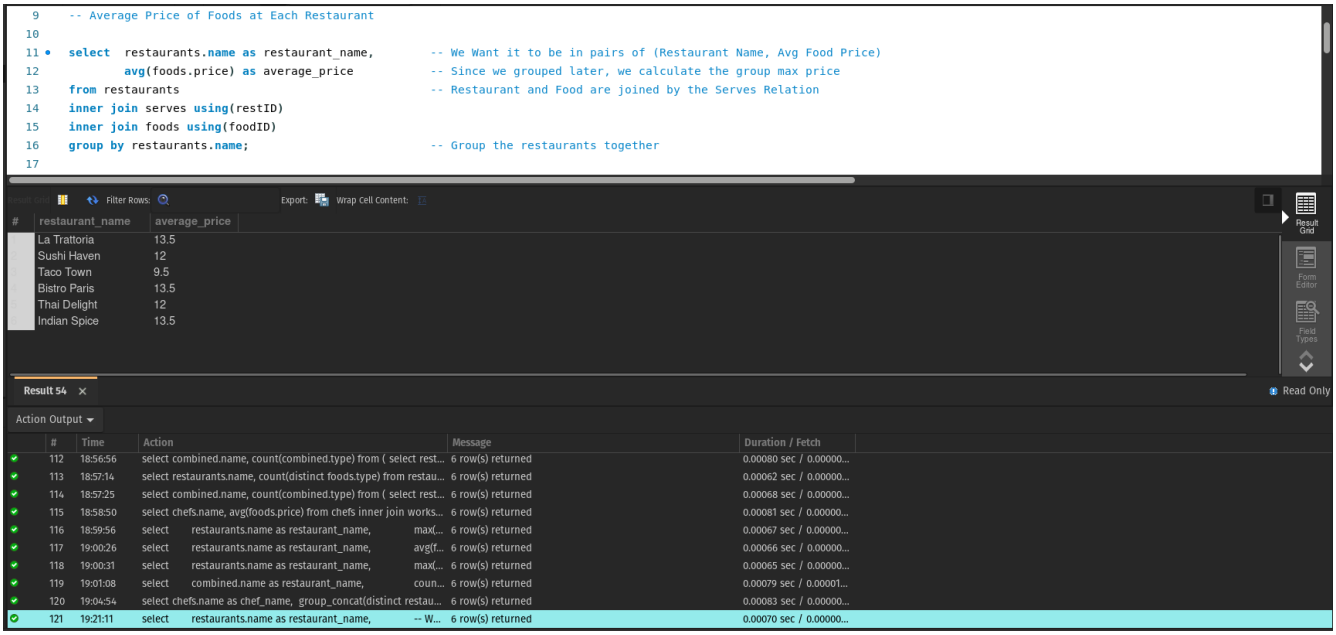
DB Assignment 2

Albert Adamson

September 30, 2025

Query 1: Average Price of Foods at Each Restaurant

```
select      restaurants.name as restaurant_name,
            avg(foods.price) as average_price
from restaurants
inner join serves using(restID)
inner join foods using(foodID)
group by restaurants.name;
```

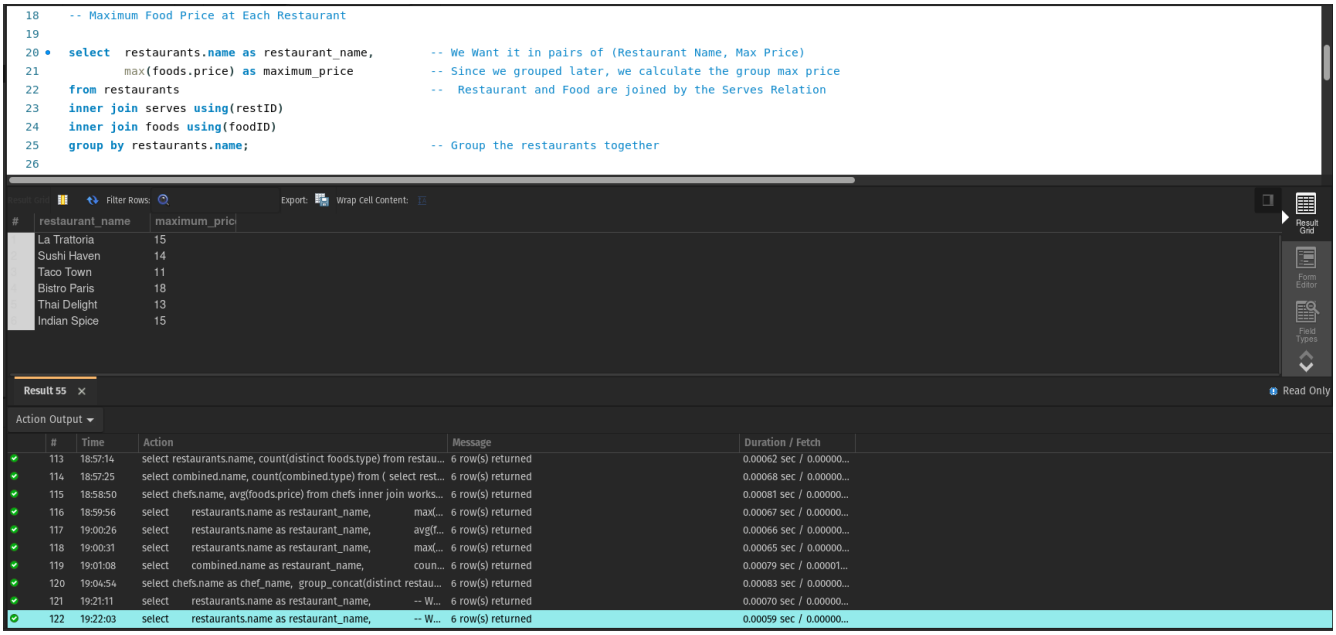


About the Query:

This SQL query works by first selecting the restaurants' name and the average of the food prices. Normally, adding "avg(foods.price)" would throw an error, but because we add a group by at the end SQL knows to apply the avg aggregate to each group. Furthermore, the "as" keywords are just used to format the name of the output. I then joined the different tables since I need to pull data from two tables with "serves" being the relation that connects the two. An inner join was used for simplicity to specify that we are using a restID and foodID. However, a cross join with a where statement would have worked fine. A natural join would NOT have worked since the restaurants and foods tables share the common "name" attribute. Lastly, the "group by" was added for the reason mentioned earlier.

Query 2: Maximum Food Price at Each Restaurant

```
select      restaurants.name as restaurant_name,
            max(foods.price) as maximum_price
from restaurants
inner join serves using(restID)
inner join foods using(foodID)
group by restaurants.name;
```

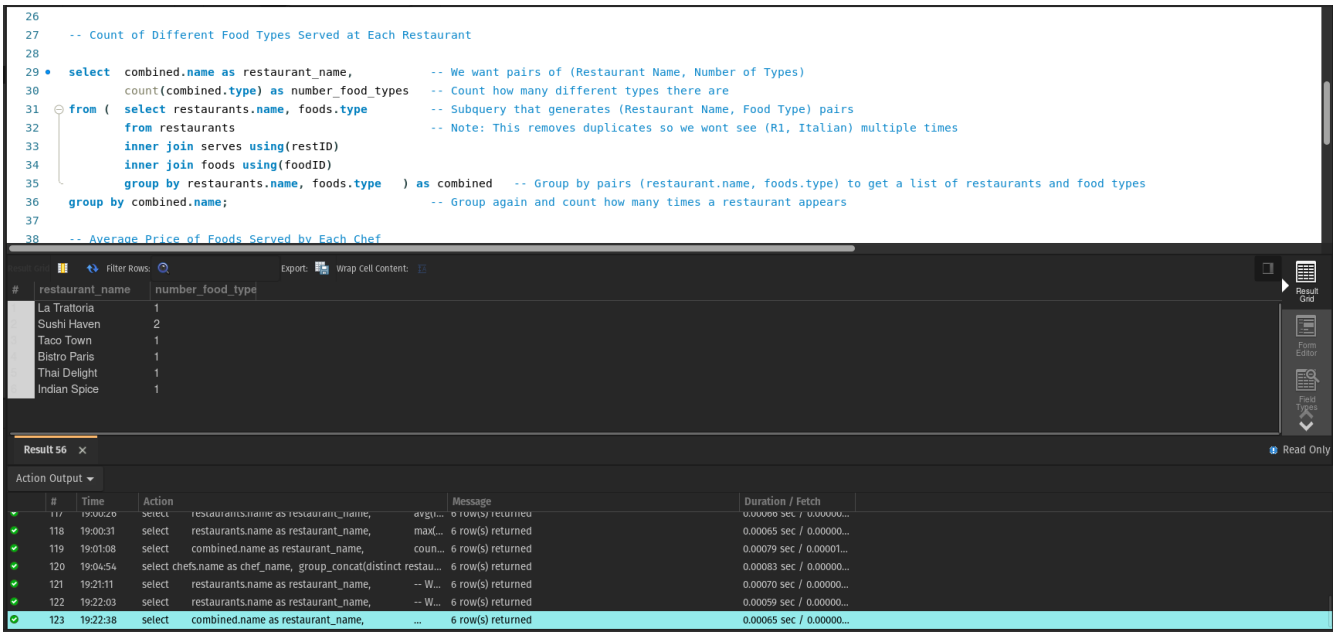


About the Query:

This SQL query is very similar to the previous one. It similarly selects the restaurants' name and the maximum of the food prices along with the "as" keywords to format the name of the output. I then joined the different tables since I need to pull data from two tables with "serves" being the relation that connects the two. An inner join was used for simplicity to specify that we are using a restID and foodID. Lastly, the "group by" was added to apply the "max" aggregate to the prices in each group.

Query 3: Count of Different Food Types Served at Each Restaurant

```
select      combined.name as restaurant_name,
            count(combined.type) as number_food_types
from (      select restaurants.name, foods.type
            from restaurants
            inner join serves using(restID)
            inner join foods using(foodID)
            group by restaurants.name, foods.type      ) as combined
group by combined.name;
```

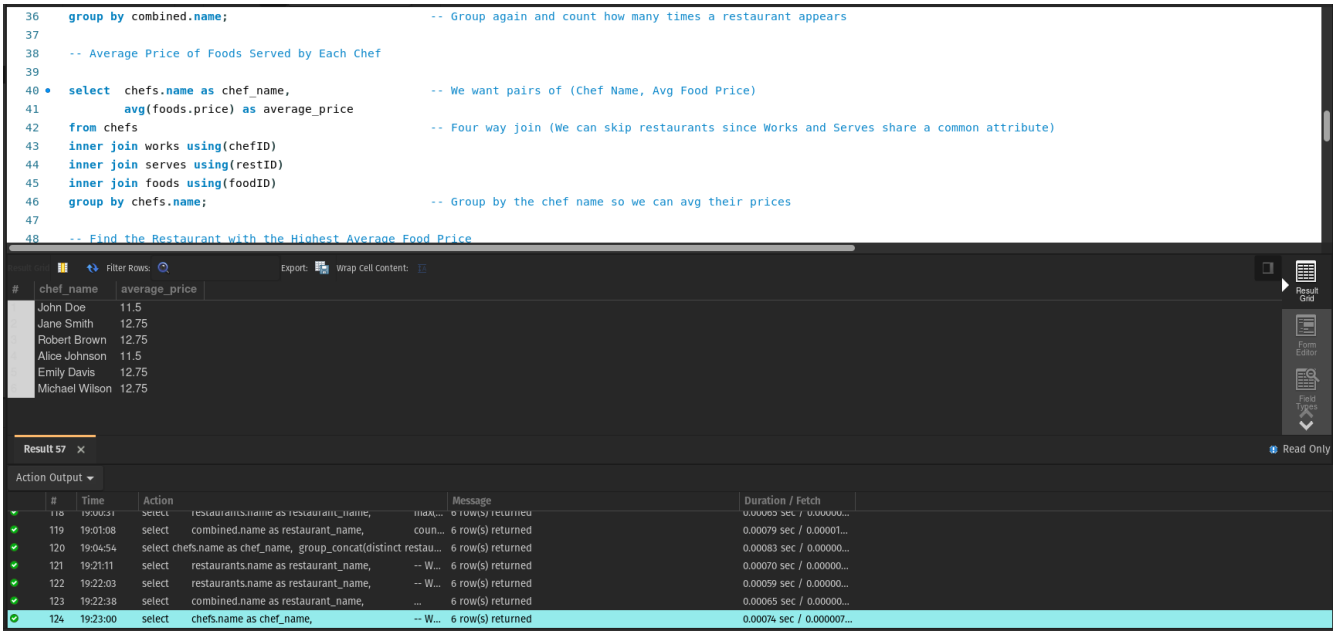


About the Query:

This SQL query works by first selecting two types “combined.name” and “count(combined.type)” and aliasing them for the output. This may seem strange at first because there shouldn’t be a table called combined, however, going into the “from” section there is a subquery embedded within it. The subquery looks at the restaurant names and the types of foods from a joined restaurant, serves, and foods table. At first it seems normal, but unlike the other two problems, this subquery is grouped by two attributes “restaurants.name” and “foods.type”. Since I grouped by two attributes this results in an ordered pair output of (restaurant.name, foods.type), furthermore, this grouping eliminates duplicates so if a restaurant serves multiple foods of the same type, that type will only get outputted once. The results of the subquery look something like this “(restaurant.name, food.type)”. Going back to the main query, by taking from this subquery, I can apply an additional grouping on it. In this case, I grouped by just the restaurant name since I can then apply the count aggregate on the number of times a different food type appears. As a result, the restaurant name and the number of food types gets outputted.

Query 4: Average Price of Foods Served by Each Chef

```
select   chefs.name as chef_name,
                  avg(foods.price) as average_price
from chefs
inner join works using(chefID)
inner join serves using(restID)
inner join foods using(foodID)
group by chefs.name;
```

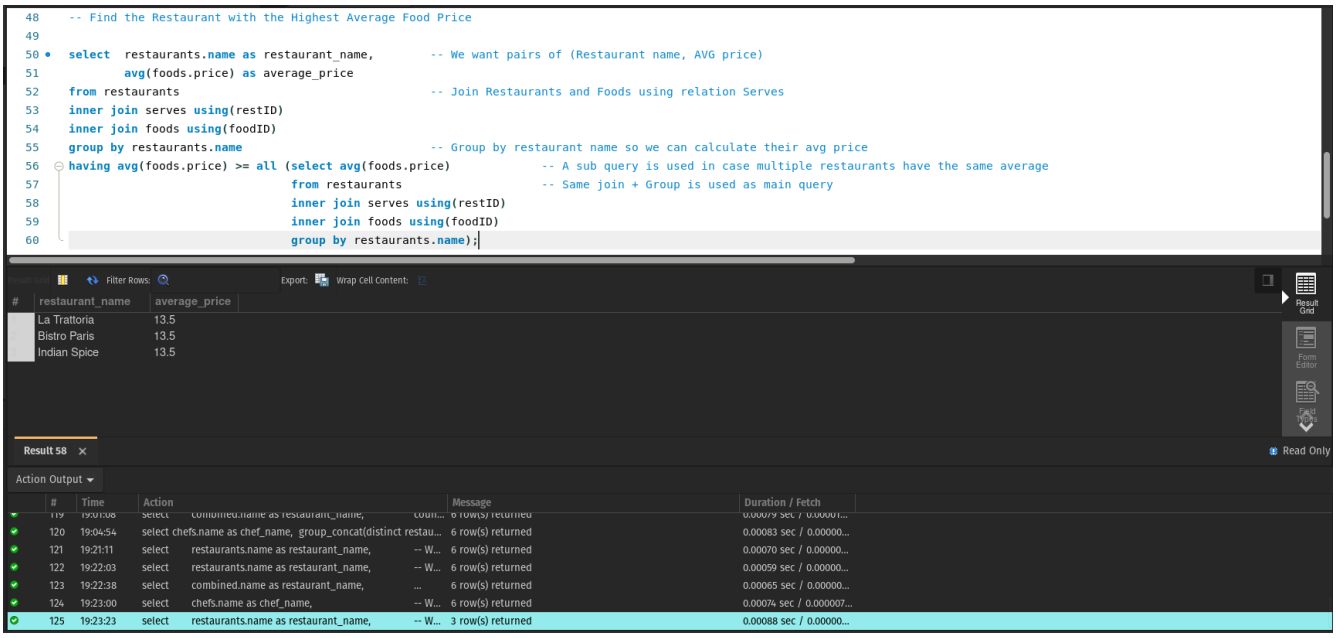


About the Query:

This query works by first selecting the chefs' names and the average food prices. These choices were also aliased to format the resulting output. Four tables were used for this join: "chefs", "works", "serves", and "foods". The reason "restaurants" wasn't used is because I didn't need any data for it and I can join the two relations "works" and "serves" together since they both share the same foreign key, "restID". Once the tables were joined together, I then grouped by the chefs' names so the "avg" aggregate would be applied to the food prices like I specified in the select statement.

Query 5: Find the Restaurant with the Highest Average Food Price

```
select      restaurants.name as restaurant_name,
            avg(foods.price) as average_price
from restaurants
inner join serves using(restID)
inner join foods using(foodID)
group by restaurants.name
having avg(foods.price) >= all (    select avg(foods.price)
                                    from restaurants
                                    inner join serves using(restID)
                                    inner join foods using(foodID)
                                    group by restaurants.name);
```

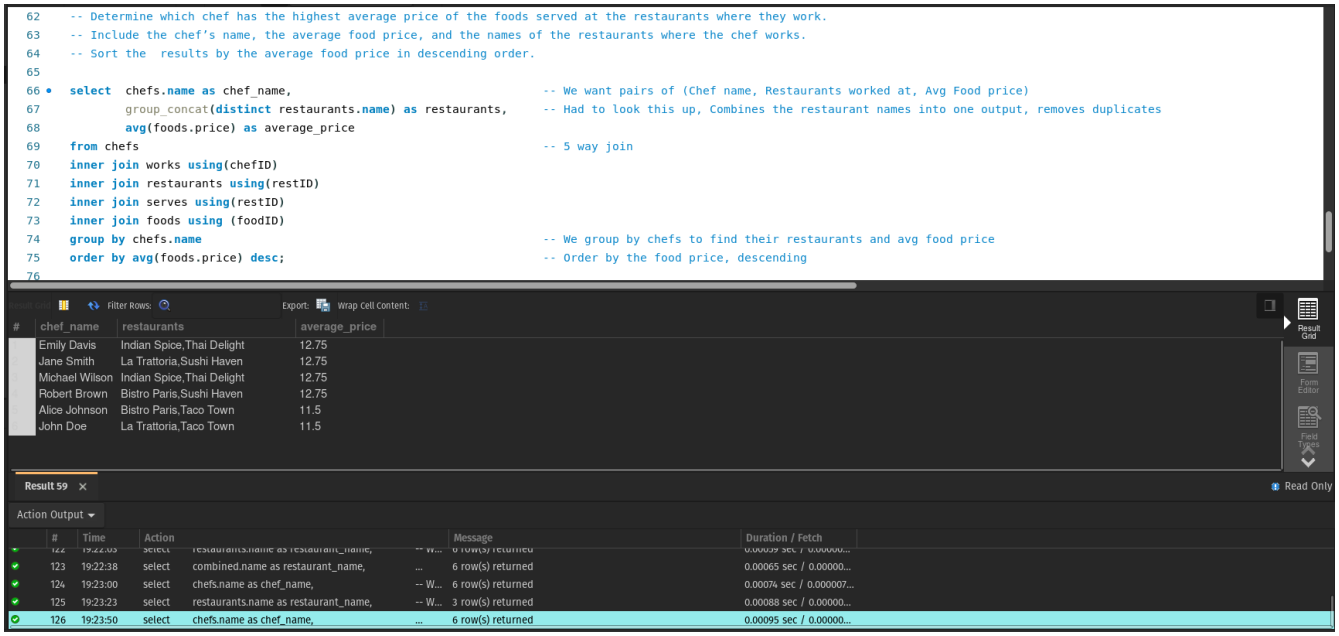


About the Query:

This query works by selecting both the restaurant name and the average food price. I then joined the “restaurant” and “foods” table by using their relation “serves”, and grouped the results by the restaurant name. However, one main difference is the additional “having” keyword. While I could have used the “order by” and a “limit 1” statements, using a “having” keyword instead allows me to output multiple restaurants if more than one have the same average food price. Within the “having” statement, I compare the average food price for the group to the entirety of an additional subquery. I used the “all” keyword since I wanted the maximum averages (hence the greater than or equal to signs). The subquery is almost the same as the main query, the only difference is the select statement only outputs the average food prices. The names aren’t needed since the comparison only looks at the average food prices and finds the maximum based on all of the food prices, so it doesn’t matter if the names are missing. As a result, multiple restaurants will be outputted if they all share the same max average food price.

Extra Credit: Determine which chef has the highest average price of the foods served at the restaurants where they work. Include the chef's name, the average food price, and the names of the restaurants where the chef works. Sort the results by the average food price in descending order.

```
select      chefs.name as chef_name,
            group_concat(distinct restaurants.name) as restaurants,
            avg(foods.price) as average_price
from chefs
inner join works using(chefID)
inner join restaurants using(restID)
inner join serves using(restID)
inner join foods using (foodID)
group by chefs.name
order by avg(foods.price) desc;
```



About the Query:

Within the select statement one thing stands out, the "group_concat(distinct)". Since the query needed to output all of the restaurants that each particular chef belongs to, this statement is needed. What it essentially does is take all of the rows from the grouping that belong to multiple restaurants and merges all of the restaurant names into a single column output. So for example, if Chef1 worked at rest1 and rest2, it would output Chef1; rest1,rest2 (Note: The comma doesn't separate the columns the ';' does in this case). The rest of the query is standard from the other ones. It joins all 5 tables since data is needed from the "chefs", "restaurants", and "foods" tables. In addition, the grouping is done with the name of the chef's and the results are ordered by the average food price in descending order.