

# 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;
```

The screenshot shows a SQL IDE interface. At the top, a query is written in a text editor. Below the editor, the results of the query are displayed in a table. The table has two columns: 'restaurant\_name' and 'average\_price'. The results are as follows:

#	restaurant_name	average_price
1	La Trattoria	13.5
2	Sushi Haven	12
3	Taco Town	9.5
4	Bistro Paris	13.5
5	Thai Delight	12
6	Indian Spice	13.5

Below the results table, there is an 'Action Output' section showing a log of database actions. The log includes timestamps, the SQL actions performed, and the number of rows returned. The final action in the log is the execution of the query shown in the editor, which returned 6 rows.

## 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;
```

The screenshot shows a SQL query editor with the following code:

```
18 -- Maximum Food Price at Each Restaurant
19
20 • select restaurants.name as restaurant_name,      -- We Want it in pairs of (Restaurant Name, Max Price)
21      max(foods.price) as maximum_price            -- Since we grouped later, we calculate the group max price
22 from restaurants                                  -- Restaurant and Food are joined by the Serves Relation
23 inner join serves using(restID)
24 inner join foods using(foodID)
25 group by restaurants.name;                        -- Group the restaurants together
26
```

Below the editor, the results are displayed in a table:

#	restaurant_name	maximum_price
1	La Trattoria	15
2	Sushi Haven	14
3	Taco Town	11
4	Bistro Paris	18
5	Thai Delight	13
6	Indian Spice	15

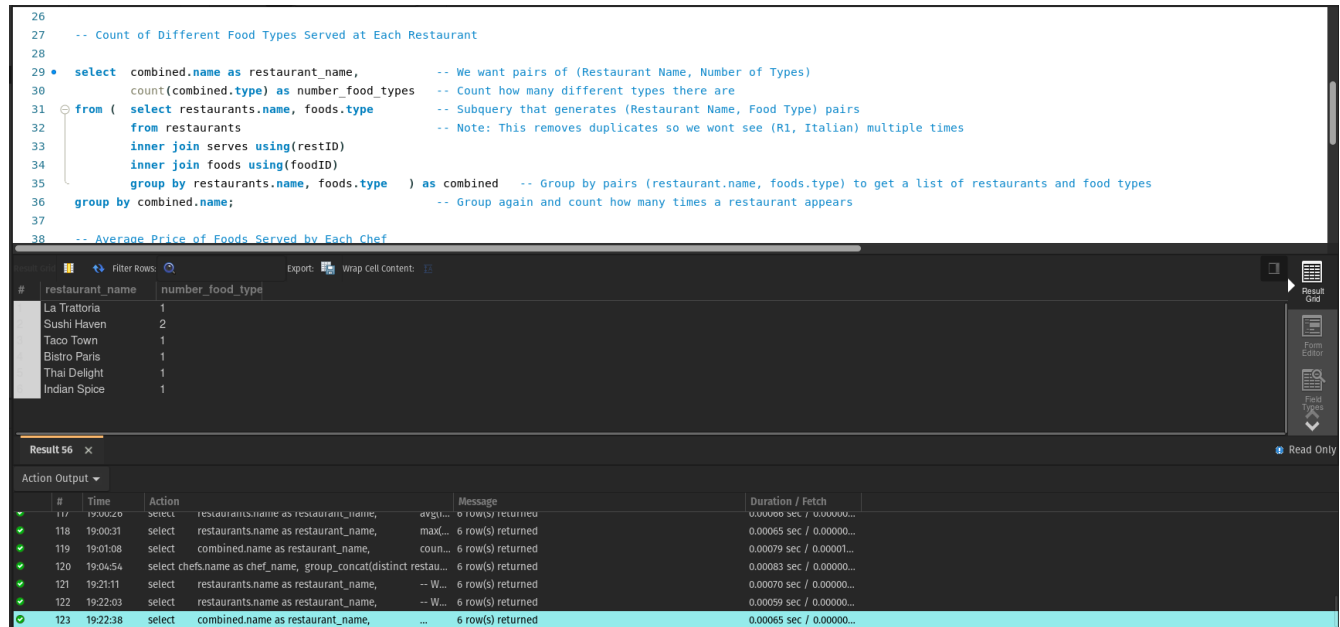
The bottom section of the screenshot shows the 'Action Output' log, which lists various SQL queries and their execution times. The last entry (122) is the query shown in the editor above, which returned 6 rows.

## 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;
```



The screenshot shows a SQL IDE interface. The top pane displays the SQL query for counting food types per restaurant. The bottom pane shows the results of the query, which are as follows:

#	restaurant_name	number_food_type
1	La Trattoria	1
2	Sushi Haven	2
3	Taco Town	1
4	Bistro Paris	1
5	Thai Delight	1
6	Indian Spice	1

Below the results, there is an 'Action Output' section showing the execution details of the query, including the time taken and the number of rows returned for each step.

### 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;
```

The screenshot shows a SQL query editor with the following code:

```
36 group by combined.name; -- Group again and count how many times a restaurant appears
37
38 -- Average Price of Foods Served by Each Chef
39
40 • select chefs.name as chef_name, -- We want pairs of (Chef Name, Avg Food Price)
41        avg(foods.price) as average_price
42 from chefs -- Four way join (We can skip restaurants since Works and Serves share a common attribute)
43 inner join works using(chefID)
44 inner join serves using(restID)
45 inner join foods using(foodID)
46 group by chefs.name; -- Group by the chef name so we can avg their prices
47
48 -- Find the Restaurant with the Highest Average Food Price
```

Below the editor, the results are displayed in a table with two columns: **chef\_name** and **average\_price**.

#	chef_name	average_price
1	John Doe	11.5
2	Jane Smith	12.75
3	Robert Brown	12.75
4	Alice Johnson	11.5
5	Emily Davis	12.75
6	Michael Wilson	12.75

At the bottom, the 'Action Output' pane shows a log of database actions, including the final query execution at 19:23:00.

## 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);
```

```
48 -- Find the Restaurant with the Highest Average Food Price
49
50 • select  restaurants.name as restaurant_name,      -- We want pairs of (Restaurant name, AVG price)
51          avg(foods.price) as average_price
52 from restaurants                                  -- Join Restaurants and Foods using relation Serves
53 inner join serves using(restID)
54 inner join foods using(foodID)
55 group by restaurants.name                         -- Group by restaurant name so we can calculate their avg price
56 having avg(foods.price) >= all (select avg(foods.price) -- A sub query is used in case multiple restaurants have the same average
57                                from restaurants      -- Same join + Group is used as main query
58                                inner join serves using(restID)
59                                inner join foods using(foodID)
60                                group by restaurants.name);
```

#	restaurant_name	average_price
1	La Trattoria	13.5
2	Bistro Paris	13.5
3	Indian Spice	13.5

Result 58

#	Time	Action	Message	Duration / Fetch
119	19:20:38	select	columns.name as restaurant_name, count...	0 rows(s) returned
120	19:04:54	select	chefs.name as chef_name, group_concat(distinct restau...	6 row(s) returned
121	19:21:11	select	restaurants.name as restaurant_name, -- W...	6 row(s) returned
122	19:22:03	select	restaurants.name as restaurant_name, -- W...	6 row(s) returned
123	19:22:38	select	combined.name as restaurant_name, -- W...	6 row(s) returned
124	19:23:00	select	chefs.name as chef_name, -- W...	6 row(s) returned
125	19:23:23	select	restaurants.name as restaurant_name, -- W...	3 row(s) returned

## 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;
```

```
62 -- Determine which chef has the highest average price of the foods served at the restaurants where they work.
63 -- Include the chef's name, the average food price, and the names of the restaurants where the chef works.
64 -- Sort the results by the average food price in descending order.
65
66 • select chefs.name as chef_name, -- We want pairs of (Chef name, Restaurants worked at, Avg Food price)
67      group_concat(distinct restaurants.name) as restaurants, -- Had to look this up, Combines the restaurant names into one output, removes duplicates
68      avg(foods.price) as average_price
69 from chefs -- 5 way join
70 inner join works using(chefID)
71 inner join restaurants using(restID)
72 inner join serves using(restID)
73 inner join foods using (foodID)
74 group by chefs.name -- We group by chefs to find their restaurants and avg food price
75 order by avg(foods.price) desc; -- Order by the food price, descending
76
```

#	chef_name	restaurants	average_price
1	Emily Davis	Indian Spice,Thai Delight	12.75
2	Jane Smith	La Trattoria,Sushi Haven	12.75
3	Michael Wilson	Indian Spice,Thai Delight	12.75
4	Robert Brown	Bistro Paris,Sushi Haven	12.75
5	Alice Johnson	Bistro Paris,Taco Town	11.5
6	John Doe	La Trattoria,Taco Town	11.5

#	Time	Action	Message	Duration / Fetch
123	19:22:38	select	combined.name as restaurant_name, ... 6 row(s) returned	0.00065 sec / 0.00000...
124	19:23:00	select	chefs.name as chef_name, ... 6 row(s) returned	0.00074 sec / 0.000007...
125	19:23:23	select	restaurants.name as restaurant_name, ... 3 row(s) returned	0.00088 sec / 0.00000...
126	19:23:50	select	chefs.name as chef_name, ... 6 row(s) returned	0.00095 sec / 0.00000...

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.