# **In-class practice – Day 14 (Summary Queries)**

**Aggregate Functions**

1. Select the count of all records in invoices
2. Add more columns to the query results that return the following
   1. Sum of invoice\_total
   2. Average invoice\_total
   3. Lowest invoice\_total
   4. Largest invoice\_total
   5. Count of distinct vendors
3. Expect that part of the output of the average will need formatting. Use a SQL *ROUND* function to format it.

**Aggregate Functions with Expression**

1. Update the query below to add a 3rd column called “amount\_due” that returns the sum of the amount due. NOTE: The “amount\_due” = (invoice\_total – payment\_total – credit\_total)

**SELECT COUNT(\*) AS number\_of\_invoices,**

**SUM(invoice\_total) AS sum\_of\_invoice\_totals**

**--add amount\_due here**

**FROM invoices;**

1. Update query to only consider invoices created on/after FEB 15th 2021.
   1. *Hint: Dates follow a “DD-MMM-YYYY” format (e.g. 07-Aug-2018).*

**GROUP BY and HAVING (Aggregate Filtering)**

1. Write a query that returns the vendor state and the count of vendors in that state
2. Update query to filter out vendor\_state equal to ‘CA’ (hint: This filter happens in WHERE)
3. Update query to only show states with more than 2 vendors (hint: This filter happens in HAVING)
4. Add in vendor\_city between the state and count columns so the count is by state and city
5. Sort by Count DESC

**ROLLUP and CUBE**

1. Pull vendor state and count of vendors like before but use a ROLLUP keyword in group by
2. Try adding in a 2nd column between state and count and update the rollup to be by state and city.
3. Try updating ROLLUP to CUBE discuss what is different about CUBE’s subtotaling.

## More practice if you like…

**HAVING compared to WHERE**

1. Figure out which Zilka Design record is being filtered out between these two statements and why

|  |  |
| --- | --- |
| **SELECT vendor\_name, COUNT(\*) AS invoice\_qty,**  **ROUND(AVG(invoice\_total),2) AS invoice\_avg**  **FROM vendors JOIN invoices**  **ON vendors.vendor\_id = invoices.vendor\_id**  **GROUP BY vendor\_name**  **HAVING AVG(invoice\_total) > 500 --aggregate filter**  **ORDER BY invoice\_qty DESC;** | **SELECT vendor\_name, COUNT(\*) AS invoice\_qty,**  **ROUND(AVG(invoice\_total),2) AS invoice\_avg**  **FROM vendors JOIN invoices**  **ON vendors.vendor\_id = invoices.vendor\_id**  **WHERE invoice\_total > 500 --row level filter**  **GROUP BY vendor\_name**  **ORDER BY invoice\_qty DESC;** |

1. Pull all invoices after 15-APR-2014 and give the average invoice\_total by vendor\_state. Only show states with avg > 2000.

**TIP: Follow these steps:**

* Select \* From table…
* Code the JOIN (if applicable)
* Code WHERE
* Specify the columns and add aggregate functions AND group by
* Code HAVING filter aggregate columns

**More practice with Functions and String values**

1. Pull the Last Name in vendor\_contacts closest to A (e.g. Adams). Pull the Last Name closest to Z (e.g.Zilker)
2. *Pop Quiz – Do you think we can pull the AVG of last\_name?*

## Above and Beyond…grouping without GROUP BY

For those of you that want to learn even more, here’s something a bit tricky that could show up on a technical interview. How do you group or summarize data without using GROUP BY clause? To do this you can use the ***OVER Partition*** syntax in SQL which will group a given column based on a partition you define as a column level. This syntax is called the Window Function and you can [learn more here](https://mode.com/sql-tutorial/sql-window-functions/). For starters, look at this SQL Statement and run it:

**SELECT DISTINCT vendor\_state,**

**vendor\_city,**

**COUNT(\*) AS invoice\_qty,**

**ROUND(sum(invoice\_total),2) AS invoice\_avg**

**FROM invoices JOIN vendors ON invoices.vendor\_id = vendors.vendor\_id**

**group by vendor\_state, vendor\_city**

**ORDER BY vendor\_state, vendor\_city;**

Notice that you’re grouping data into states and then cities in order to get the count of invoices and sum of invoice total. If you were remove the GROUP BY line of code and run this query, it would result in the common grouping error.

But SQL allows you to define grouping parameters in the SELECT with slightly different syntax. If you select the COUNT(\*) ***OVER (PARTITION BY vendor\_state, vendor\_city)*** this tells SQL to group the data by state and city for the count. Try this version of the query that returns the exact some things as above.

**SELECT DISTINCT vendor\_state,**

**vendor\_city,**

**COUNT(\*) OVER (PARTITION BY vendor\_state, vendor\_city) AS invoice\_qty,**

**SUM(invoice\_total) OVER (PARTITION BY vendor\_state, vendor\_city) AS invoice\_sum**

**FROM invoices JOIN vendors ON invoices.vendor\_id = vendors.vendor\_id**

**ORDER BY vendor\_state, vendor\_city;**

If you used GROUP BY, the grouping applies to all aggregates in the SELECT. In this case, that’s the Count and Sum. With the *Over Partition* feature you can group each aggregate differently to get subtotaling a different levels. For example, run this statement that only groups count by state while sum of invoice\_total is by state and city. See if you understand why the results are different:

**SELECT DISTINCT vendor\_state,**

**vendor\_city,**

**COUNT(\*) OVER (PARTITION BY vendor\_state) AS invoice\_qty,**

**SUM(invoice\_total) OVER (PARTITION BY vendor\_state, vendor\_city) AS invoice\_sum**

**FROM invoices JOIN vendors ON invoices.vendor\_id = vendors.vendor\_id**

**ORDER BY vendor\_state, vendor\_city;**