In-class practice – (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
    a. Sum of invoice_total
    b. Average invoice_total
    c. Lowest invoice_total
    d. Largest invoice_total
    e. 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

4. 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;
```

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

GROUP BY and HAVING (Aggregate Filtering)

6. Write a query that returns the vendor state and the count of vendors in that state
7. Update query to filter out vendor_state equal to 'CA' (hint: This filter happens in WHERE)
8. Update query to only show states with more than 2 vendors (hint: This filter happens in HAVING)
9. Add in vendor_city between the state and count columns so the count is by state and city
10. Sort by Count DESC

ROLLUP and CUBE

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

More practice if you like…

HAVING compared to WHERE

14. 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;
```

15. 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

16. Pull the Last Name in vendor_contacts closest to A (e.g. Adams). Pull the Last Name closest to Z (e.g.Zilker)
17. *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. 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;
```