Mastering SQL Through Real-World Business Problems

Learning SQL is often about syntax, but understanding how to apply it to real-world business challenges is equally important. While many tutorials focus on teaching SQL commands, they often miss explaining the thought process behind solving actual data problems.

In this blog, I share insights from my own SQL practice sessions, breaking down business problems into queries while illustrating the approach to visualizing and solving them.

We will start with some fundamental queries and gradually build complexity. To follow along, you can use the **Walmart sales dataset** for data analysis in **PostgreSQL**. [link is on GitHub]

Setting Up Your Database

Before diving into queries, the first step is to create a table in PostgreSQL and upload the dataset. Use the following **CREATE TABLE** command to structure your database with appropriate column names and data types.

```
CREATE TABLE walmart_sales(
                invoice_id VARCHAR(15),          -- string with maximum length of 15 characters.
                branch      CHAR(1),
                city  VARCHAR(25),
                customer_type       VARCHAR(15),
                gender      VARCHAR(15),
                product_line VARCHAR(55),
                unit_price  FLOAT,                  -- stores numeric values with decimal places.
                quantity    INT,                    -- stores integers
                vat         FLOAT,
                total       FLOAT,
                date        date,
                time time,
                payment_method VARCHAR(15),
                rating FLOAT
            );
```

Here is how the table will be created after the above query run:

invoice_id character varying (15)	branch character (1)	city character varying (25)	customer_type character varying (15)	gender character varying (15)	product_line character varying (55)	unit_price double precision	quantity integer	vat double precision

Now it is time to feed the data into the table. Once that is done, you can run the below query to select all records and it will display all the records. It can also verify that all the data has been transferred to the SQL Database.

```
SELECT * from walmart_sales;
```

---------- **Business Problems** ----------

Q.1 Find the total sales amount for each branch.

```
SELECT branch,                    -- filtering branch column.
    SUM(total) as branch_total    -- adding the sales total for each branch and naming it.
FROM walmart_sales
GROUP BY branch;                  -- now grouping the branch column to get the total.
```

Result:

	branch character (1)	branch_total double precision
1	E	60061.01850000001
2	D	85349.59650000003
3	A	39383.1165
4	C	69656.13900000002
5	B	68516.87849999996

Q.2 Calculate the average customer rating for each city.

```
SELECT city,
    AVG(rating) as Avg_rating     -- calculating the average rating for each city.
FROM walmart_sales
GROUP BY city;                    -- now grouping the city column to get distinct value for each city.
```

Result:

	city character varying (25)	avg_rating double precision
1	San Antonio	6.729166666666668
2	Dallas	6.967368421052632
3	Los Angeles	6.991818181818185
4	Chicago	7.000909090909089
5	Houston	7.066666666666666
6	New York	6.753846153846154
7	Philadelphia	6.891346153846153
8	San Jose	7.220212765957446
9	San Diego	6.999115044247786
10	Phoenix	7.145161290322578

Q.3 Count the number of sales transactions for each customer type.

```
SELECT customer_type,
    COUNT(*) as total_sales                    -- counting each sale made.
FROM walmart_sales
GROUP BY customer_type;          -- grouping by  customer_type column to get total sales made by each type.
```

Result:

	customer_type character varying (15)	total_sales bigint
1	Normal	499
2	Member	501

Q.4 Find the total quantity of products sold for each product line.

```
SELECT product_line,
    COUNT(quantity) as total_quantity              -- Counting total quantity in a new column.
FROM walmart_sales
GROUP BY product_line ;          -- grouping by product_line column to combine the quantity for each category.
```

Result:

	product_line character varying (55)	total_quantity bigint
1	Fashion accessories	178
2	Electronic accessories	170
3	Health and beauty	152
4	Food and beverages	174
5	Sports and travel	166
6	Home and lifestyle	160

Q.5 Calculate the total VAT collected for each payment method.

```
SELECT payment_method,
    SUM(vat) as total_vat                  -- adding the vat for each payment method.
FROM walmart_sales
GROUP BY payment_method ;        -- now grouping the payment_method column to get total by each method.
```

Result:

	payment_method character varying (15)	total_vat double precision
1	Credit card	4798.432000000001
2	Ewallet	5237.767000000001
3	Cash	5343.170000000006

Q.6 Find the total sales amount and average customer rating for each branch.

```
SELECT branch,
    SUM(total) as total_sales_amount,          -- adding the total sales for each branch.
        AVG(rating) as Average_rating          -- calculating the average rating for each branch.
FROM walmart_sales
GROUP BY branch;                               -- grouping by branch column to get combined data.
```

Result:

	branch character (1)	total_sales_amount double precision	average_rating double precision
1	E	60061.01850000001	7.022999999999997
2	D	85349.59650000003	6.995019157088127
3	A	39383.1165	6.918018018018014
4	C	69656.13900000002	6.850222222222216
5	B	68516.87849999996	7.0600985221674835

Q.7 Calculate the total sales amount for each city and gender combination.

```
SELECT city, gender,                           -- listing the columns : city is 1 and gender is 2
    SUM(total)                                 -- calculating the total amount of sales.
FROM walmart_sales
GROUP BY 1, 2;                                 -- grouping the data by city and gender columns to get desired results.
```

Result:

	city character varying (25)	gender character varying (15)	sum double precision
1	Los Angeles	Male	14327.649000000001
2	Chicago	Female	16599.009
3	San Jose	Female	17258.713499999998
4	New York	Male	14368.084499999999
5	San Jose	Male	17475.149999999998
6	Philadelphia	Male	15638.993999999999
7	New York	Female	13552.591500000002
8	Chicago	Male	18266.63999999999
9	Dallas	Male	15705.427499999998
10	San Diego	Female	18028.9515
11	Los Angeles	Female	26683.429500000002
12	Houston	Female	10744.8915
13	Phoenix	Male	13027.780500000003
14	Philadelphia	Female	15511.251
15	San Antonio	Female	17003.605499999998
16	San Antonio	Male	13262.025000000001
17	Houston	Male	14632.757999999994
18	Dallas	Female	15103.4625
19	San Diego	Male	18379.3155
20	Phoenix	Female	17397.019500000006

Q.8 Find the average quantity of products sold for each product line to female customers.

```
SELECT product_line,
        AVG(quantity)                    -- calculating the average quantity of product sold.
FROM walmart_sales
WHERE gender = 'Female'                  -- filtering only the female customers.
GROUP BY product_line;                   -- grouping by the category of the product.
```

Result:

	product_line character varying (55)	avg numeric
1	Fashion accessories	5.5208333333333333
2	Electronic accessories	5.8095238095238095
3	Health and beauty	5.3593750000000000
4	Sports and travel	5.6363636363636364
5	Food and beverages	5.7111111111111111
6	Home and lifestyle	6.3037974683544304

Q.9 Count the number of sales transactions for members in each branch.

```
SELECT branch,
    COUNT(invoice_id) as No_of_sales     -- counting the sales made by each branch.
FROM walmart_sales
WHERE customer_type = 'Member'           -- only by Walmart members.
GROUP BY branch;                         -- grouping by branch column to get combined data.
```

Result:

	branch character (1)	no_of_sales bigint
1	E	107
2	D	134
3	A	59
4	C	111
5	B	90

Q.10 Find the total sales amount for each day. (Return day name and their total sales order DESC by amt)

```
SELECT
    TO_CHAR (date, 'Day') AS Day_name,      -- extracting day from the time column.
        SUM(total) as total_sales_amount    -- calculating total sales made.
FROM walmart_sales
GROUP BY Day_name                           -- grouping by day.
```

```
ORDER BY total_sales_amount DESC;                    -- organising the total sales amount in descending order.
```

Result:

	day_name text	total_sales_amount double precision
1	Saturday	56120.80949999999
2	Tuesday	51482.24550000001
3	Thursday	45349.248000000014
4	Sunday	44457.892499999994
5	Friday	43926.34050000002
6	Wednesday	43731.135
7	Monday	37899.07799999999

Q.11 Calculate the total sales amount for each hour of the day.

```
SELECT
    EXTRACT ( HOUR FROM time) as hours, -- extracting hour from the time. Syntax: EXTRACT(part FROM date)
        SUM(total) as total_sales_amount          -- calculating total sales.
FROM walmart_sales
GROUP BY hours                                    -- grouping by day.
order by total_sales_amount;                      -- organising the total sales amount in ascending order(default).
```

Result:

	hours numeric	total_sales_amount double precision
1	20	22969.527000000002
2	17	24445.218
3	16	25226.323499999995
4	18	26030.339999999986
5	12	26065.882499999996
6	11	30377.329499999996
7	14	30828.399
8	15	31179.508499999996
9	10	31421.48100000001
10	13	34723.22700000001
11	19	39699.51300000002

Q.12 Find the total sales amount for each month. (return month name and their sales)

```
SELECT
    TO_CHAR (date, 'month') AS months,          --converts the date into strings, here in months.
        SUM(total) as total_sales_amount        -- calculating the total of sales amount.
FROM walmart_sales
GROUP BY months                                 -- grouping by months.
ORDER BY total_sales_amount DESC;               -- organising the total sales amount in descending order.
```

Result:

	months text	total_sales_amount double precision
1	january	116291.86800000005
2	march	109455.50700000004
3	february	97219.37399999997

Q.13 Calculate the total sales amount for each branch where the average customer rating is greater than 7.

```
SELECT branch,
    SUM(total) as total_sales_amount,          -- calculating the total of sales amount.
        AVG(rating)                            -- calculating the average of ratings.
FROM walmart_sales
GROUP BY branch                                -- grouping by branch.
HAVING AVG(rating) > 7;                        -- where average rating of the branch is greater than 7.
```

Result:

	branch character (1)	total_sales_amount double precision	avg double precision
1	E	60061.01850000001	7.022999999999997
2	B	68516.87849999996	7.0600985221674835

Q.14 Find the total VAT collected for each product line where the total sales amount is more than 500.

```
SELECT product_line,
    SUM(vat) as total_vat                      -- calculating the total of vat amount.
FROM walmart_sales
GROUP BY product_line                          -- grouping by product line.
HAVING SUM(vat) > 500 ;                        -- where sum of vat amount is greater than 500.
```

Result:

	product_line character varying (55)	total_vat double precision
1	Fashion accessories	2585.995
2	Electronic accessories	2587.5015000000017
3	Health and beauty	2342.5589999999993
4	Food and beverages	2673.5639999999994
5	Sports and travel	2624.8964999999994
6	Home and lifestyle	2564.853000000002

Q.15 Calculate the average sales amount for each gender in each branch.

```
SELECT branch,
```

```
    gender,
        AVG(total) as branch_total                    -- calculating the average of total sales amount.
FROM walmart_sales
GROUP BY branch, gender                               -- grouping by branch and gender both.
ORDER BY branch;                                      -- ordering the data by branch.
```

Result:

	branch character (1)	gender character varying (15)	branch_total double precision
1	A	Female	382.4696842105264
2	A	Male	325.5989722222222
3	B	Male	342.2781428571428
4	B	Female	333.08210000000014
5	C	Female	349.59261386138616
6	C	Male	276.9942338709676
7	D	Female	340.74136551724143
8	D	Male	309.84567672413795
9	E	Male	314.66203738317745
10	E	Female	283.78688709677414

Q.16 Count the number of sales transactions for each day of the week.

```
SELECT
    TO_CHAR (date, 'Day') AS Day_name,          --converts the date into strings, here in days.
        COUNT(payment_method) as no_of_sales    --calculating total sales made by different payment method on each day.
FROM walmart_sales
GROUP BY Day_name;                              --grouping by the day
```

Result:

	day_name text	no_of_sales bigint
1	Monday	125
2	Saturday	164
3	Tuesday	158
4	Friday	139
5	Thursday	138
6	Wednesday	143
7	Sunday	133

Q.17 Find the total sales amount for each city and customer type combination where the number of sales transactions is greater than 50.

```
SELECT city,
    customer_type,
        SUM(total) as total_sales                   --calculating sum of total sales amount.
```

```
FROM walmart_sales
GROUP BY city, customer_type                --grouping the data by city and customer_type both.
HAVING COUNT(payment_method) > 50           --providing the results for city where no. of payments are greater than 50.
ORDER BY city;                              --ordering the data by city.
```

Result:

	city character varying (25)	customer_type character varying (15)	total_sales double precision
1	Chicago	Normal	19891.735499999995
2	Dallas	Member	19981.57349999999
3	Los Angeles	Member	21068.407500000005
4	Los Angeles	Normal	19942.671000000002
5	New York	Member	17583.426
6	Philadelphia	Normal	18028.038
7	San Diego	Normal	17484.663000000008
8	San Diego	Member	18923.603999999996

Q.18 Calculate the average unit price for each product line and payment method combination.

```
SELECT product_line,                        --1
    payment_method,                         --2
        AVG(unit_price) as avg_unit_price   --calculating average unit price for each product line.
FROM walmart_sales
GROUP BY 1 , 2                              --grouping by 1 & 2.
ORDER BY product_line;                      --ordering by product line.
```

Result:

	product_line character varying (55)	payment_method character varying (15)	avg_unit_price double precision
1	Electronic accessories	Cash	52.039859154929566
2	Electronic accessories	Ewallet	55.823773584905666
3	Electronic accessories	Credit card	53.266956521739125
4	Fashion accessories	Cash	61.1159649122807
5	Fashion accessories	Ewallet	54.78553846153846
6	Fashion accessories	Credit card	55.8692857142857
7	Food and beverages	Credit card	54.57770491803279
8	Food and beverages	Cash	60.8580701754386
9	Food and beverages	Ewallet	52.631964285714275
10	Health and beauty	Cash	53.39551020408163
11	Health and beauty	Ewallet	55.230566037735834
12	Health and beauty	Credit card	55.885600000000004
13	Home and lifestyle	Cash	57.483333333333334
14	Home and lifestyle	Credit card	49.39155555555556
15	Home and lifestyle	Ewallet	57.756875000000015
16	Sports and travel	Cash	56.435762711864406
17	Sports and travel	Credit card	56.44490566037735
18	Sports and travel	Ewallet	58.14055555555555

Q.19 Find the total sales amount for each branch and hour of the day combination.

```
SELECT branch,
    EXTRACT( HOUR FROM time) AS hour_of_the_day,      --extracting hour from time column.
        SUM(total) AS total_sales                     --calculating sum of total sales amount.
FROM walmart_sales
GROUP BY branch, hour_of_the_day                      --grouping by branch and hour.
ORDER BY branch, hour_of_the_day;                     --ordering ths data by branch and hour.
```

Result:

	branch character (1)	hour_of_the_day numeric	total_sales double precision
1	A	10	7163.625000000002
2	A	11	1664.8589999999997
3	A	12	2518.1730000000002
4	A	13	3262.2765
5	A	14	4190.865
6	A	15	3225.3795
7	A	16	1087.6109999999999
8	A	17	5019.483
9	A	18	2681.1645000000003

Q.20 Calculate the total sales amount and average customer rating for each product line where the total sales amount is greater than 1000.

```
SELECT product_line,
        SUM(total) as total_sales,                    --calculating sum of total sales amount.
        AVG(rating) as avg_rating                     --calculating the average of rating for each product line.
FROM walmart_sales
GROUP BY product_line                                 --grouping the data by product line.
HAVING SUM(total)  > 1000                             --where total sales are greater than 1000.
ORDER BY product_line;                                --ordering by product line.
```

Result:

	product_line character varying (55)	total_sales double precision	avg_rating double precision
1	Electronic accessories	54337.531500000005	6.92470588235294
2	Fashion accessories	54305.895	7.029213483146067
3	Food and beverages	56144.844000000005	7.113218390804598
4	Health and beauty	49193.739000000016	7.003289473684212
5	Home and lifestyle	53861.91300000001	6.8375
6	Sports and travel	55122.826499999996	6.916265060240964

Q.21 Calculate the total sales amount for morning (6 AM to 12 PM), afternoon (12 PM to 6 PM), and evening (6 PM to 12 AM) periods using the time condition.

```
SELECT * from walmart_sales;

WITH new_table                                  --creating a new temporary table to convert time into shifts.

AS

(SELECT *,

CASE

    WHEN EXTRACT(HOUR FROM time) BETWEEN 6 AND 12 THEN 'Morning'

        WHEN EXTRACT(HOUR FROM time) > 12 AND EXTRACT(HOUR FROM time) <= 18 THEN 'Afternoon'

        ELSE 'Evening'

END as shifts

FROM walmart_sales)          --extracting morning, afternoon and evening shift from the time column.



SELECT shifts,

    SUM(total) as total_sales                   --calculating sum of total sales amount.

FROM new_table

GROUP BY shifts;                                --grouping the total sales made at each shift.
```

Result:

	shifts text	total_sales double precision
1	Afternoon	172433.01599999997
2	Evening	62669.04000000002
3	Morning	87864.69299999996