Week 3: Data Aggregation in SQL

Parch and Posey Database

Contents

Introduction **1**

COUNT **3**

SUM **3**

MIN and MAX **3**

AVERAGE **4**

GROUP BY **5**

DISTINCT **7**

HAVING **7**

DATE Function with GROUP BY **10**

CASE **12**

Subqueries **14**

Recap: In the Parch & Posey database there are five tables:

- web_events
- accounts
- orders
- sales_reps
- region

Figure 1 shows the ERD (entity relationship diagram) for Parch and Posey.

Reference: These notes are based on material from Mode Analytics as well as w3schools.

Introduction

SQL is excellent at aggregating data the way you might in a pivot table in Excel. You will use aggregate functions all the time, so it's important to get comfortable with them. The functions themselves are the same ones you will find in Excel or any other analytics program. We'll cover them individually in the next few lessons. Here's a quick preview:

- **COUNT**: counts how many rows are in a particular column.
- **SUM**: adds together all the values in a particular column.

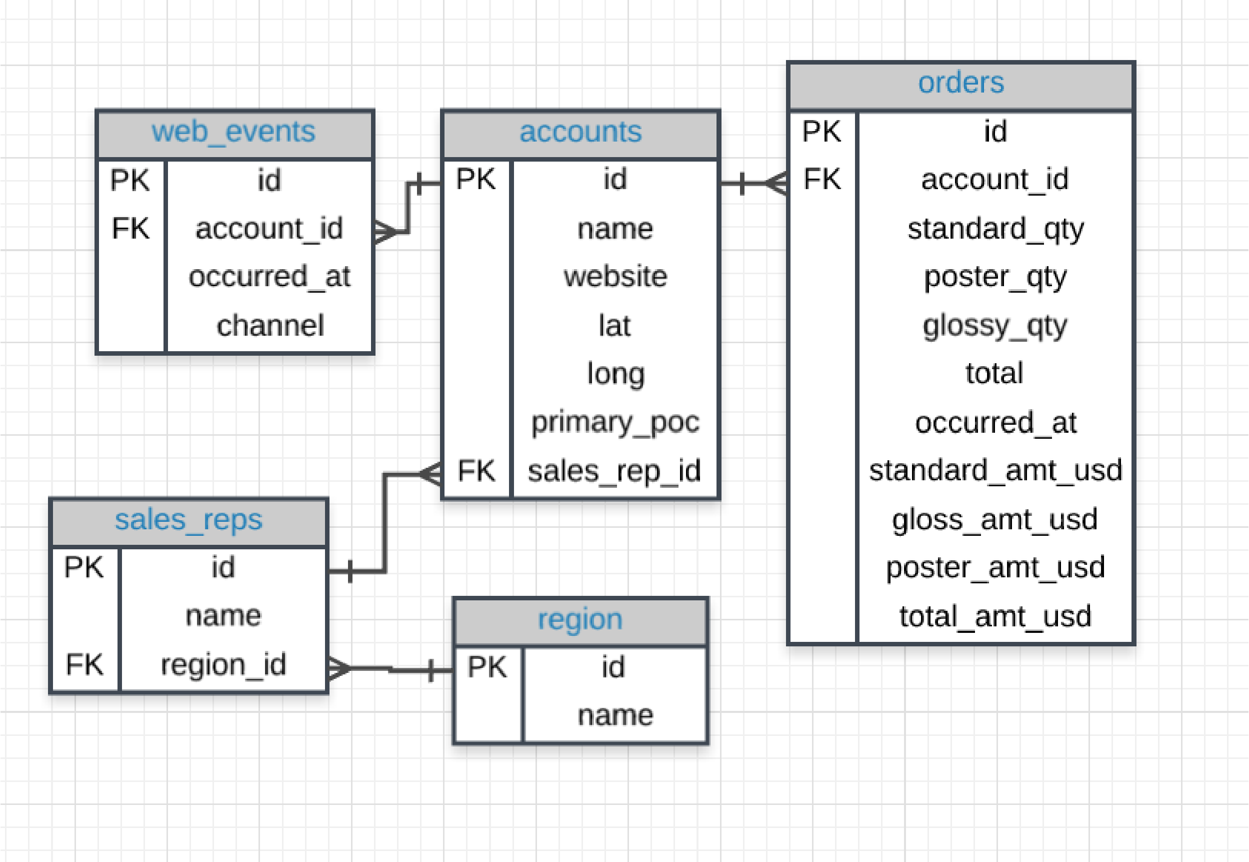


Figure 1: Parch and Posey ERD

- **MIN** and **MAX** return the lowest and highest values in a particular column, respectively.
- **AVG** calculates the average of a group of selected values.

COUNT

COUNT is a SQL aggregate function for counting the number of rows in a particular column.

1. Count the number of rows in the `accounts` table.

```sql
SELECT COUNT(*)
FROM accounts
```

count
351

When using COUNT to count the number of rows in an individual column, then it will return the number of rows that are not NULL in this row.

SUM

SUM is a SQL aggregate function that totals the values in a given column. Unlike COUNT, you can only use SUM on columns containing numerical values.

2. Find the total amount of `poster_qty` paper ordered in the `orders` table.

```sql
SELECT SUM(poster_qty) AS total_poster_sales
FROM orders;
```

total__poster__sales
723646

3. Find the total amount for each individual order that was spent on standard and gloss paper in the orders table. This should give a dollar amount for each order in the table.

```sql
SELECT standard_amt_usd + gloss_amt_usd AS total_standard_gloss
FROM orders
LIMIT 5;
```

total__standard__gloss
778.55
1255.19
776.18
958.24
756.13

MIN and MAX

MIN and MAX are SQL aggregation functions that return the lowest and highest values in a particular column.

They're similar to COUNT in that they can be used on non-numerical columns. Depending on the column type, MIN will return the lowest number, earliest date, or non-numerical value as close alphabetically to "A" as possible. As you might suspect, MAX does the opposite—it returns the highest number, the latest date, or the non-numerical value closest alphabetically to "Z".

4. What is the min and max order quantity for each poster papers in the database?

```
SELECT MIN(poster_qty) as poster_min, MAX(poster_qty) poster_max
FROM orders;
```

poster_min	poster_max
0	28262

5. When was the earliest order ever placed?

```
SELECT MIN(occurred_at)
FROM orders;
```

min
2013-12-04 04:22:44

6. Try performing the same query as in the previous without using an aggregation function.

```
SELECT occurred_at
FROM orders
ORDER BY occurred_at
LIMIT 1;
```

occurred_at
2013-12-04 04:22:44

AVERAGE

AVG is a SQL aggregate function that calculates the average of a selected group of values. It's very useful, but has some limitations. First, it can only be used on numerical columns. Second, it ignores nulls completely.

7. Find the mean (AVERAGE) amount spent per order on each paper type.

```
SELECT AVG(standard_amt_usd) mean_standard, AVG(gloss_amt_usd) mean_gloss,
           AVG(poster_amt_usd) mean_poster
FROM orders;
```

mean_standard	mean_gloss	mean_poster
1399.356	1098.547	850.1165

8. What is the MEDIAN `total_usd` spent on all orders?

```
SELECT *
FROM (SELECT total_amt_usd
      FROM orders
      ORDER BY total_amt_usd
      LIMIT 3457) AS Table1
ORDER BY total_amt_usd DESC
LIMIT 2;
```

total_amt_usd
2483.16
2482.55

Since there are 6,912 orders, we want the average of the 3,457 and 3,456 order amounts when ordered. This is the average of 2,483.16 and 2,482.55. This gives the median of 2,482.855.

GROUP BY

SQL aggregate functions like COUNT, AVG, and SUM have something in common: they all aggregate across the entire table. But what if you want to aggregate only part of a table? For example, you might want to count the number of orders for each account.

In situations like this, you'd need to use the GROUP BY clause. GROUP BY allows you to separate data into groups, which can be aggregated independently of one another.

Notice that you can group by multiple columns, but you have to separate column names with commas just as with ORDER BY.

9. Which account (by name) placed the earliest order? Your solution should have the account name and the date (occurred_at) of the order.

```sql
SELECT a.name, o.occurred_at
FROM accounts a
JOIN orders o
ON a.id = o.account_id
ORDER BY occurred_at
LIMIT 1;
```

name	occurred_at
DISH Network	2013-12-04 04:22:44

10. **YOUR TURN** Find the total sales in usd for each account. You should include two columns: the total sales for each company's orders in usd and the company name (column names should be `name` and `total_sales`)

11. Via what channel did the most recent (latest) web_event occur, which account was associated with this web_event? Your query should return only three values - the date, channel, and account name.

```sql
SELECT w.occurred_at, w.channel, a.name
FROM web_events w
JOIN accounts a
ON w.account_id = a.id
ORDER BY w.occurred_at DESC
LIMIT 1;
```

occurred_at	channel	name
2017-01-01 23:51:09	organic	Molina Healthcare

12. **YOUR TURN** Find the total number of times each type of channel from the web_events was used. Your final table should have two columns - `channel` (the channel name) and `times_used` (the number of times the channel was used).

13. Who was the primary contact associated with the earliest web_event?

```sql
SELECT a.primary_poc
FROM web_events w
JOIN accounts a
ON a.id = w.account_id
ORDER BY w.occurred_at
LIMIT 1;
```

primary_poc
Leana Hawker

14. What was the smallest order placed by each account in terms of total usd. Provide only two columns - the account name and the total usd. Order from smallest dollar amounts to largest.

```
SELECT a.name, MIN(total_amt_usd) smallest_order
FROM accounts a
JOIN orders o
ON a.id = o.account_id
GROUP BY a.name
ORDER BY smallest_order
LIMIT 3;
```

name	smallest_order
Reynolds American	0
Monsanto	0
Gilead Sciences	0

15. **YOUR TURN** Find the number of sales reps in each region. Your final table should have two columns - `name` (the region name) and `num_reps` (the number of sales_reps). Order from fewest reps to most reps.

16. For each account, determine the average amount of each type of paper they purchased across their orders. Your result should have four columns - one for the account name and one for the average spent on each of the paper types.

```
SELECT a.name, AVG(o.standard_qty) avg_stand, AVG(o.gloss_qty) avg_gloss, AVG(o.poster_qty) avg_post
FROM accounts a
JOIN orders o
ON a.id = o.account_id
GROUP BY a.name
LIMIT 3;
```

name	avg_stand	avg_gloss	avg_post
Comcast	294.00000	18.28571	28.85714
Microsoft	88.46154	32.23077	62.30769
Monsanto	353.61404	42.71930	26.24561

17. Determine the number of times a particular channel was used in the web_events table for each sales rep. Your final table should have three columns - the name of the sales rep, the channel, and the number of occurrences. Order your table with the highest number of occurrences first.

```
SELECT s.name, w.channel, COUNT(*) num_events
FROM accounts a
JOIN web_events w
ON a.id = w.account_id
JOIN sales_reps s
ON s.id = a.sales_rep_id
GROUP BY s.name, w.channel
ORDER BY num_events DESC
LIMIT 5;
```

name	channel	num_events
Earlie Schleusner	direct	234
Vernita Plump	direct	232
Moon Torian	direct	194
Georgianna Chisholm	direct	188
Tia Amato	direct	185

18. **YOUR TURN** Determine the number of times a particular channel was used in the web_events table for each region. Your final table should have three columns - the region name, the channel, and the number of occurrences. Order your table with the highest number of occurrences first.

DISTINCT

You'll occasionally want to look at only the unique values in a particular column. You can do this using SELECT DISTINCT syntax.

If you include two (or more) columns in a SELECT DISTINCT clause, your results will contain all of the unique pairs of those two columns:

DISTINCT can be particularly helpful when exploring a new data set. In many real-world scenarios, you will generally end up writing several preliminary queries in order to figure out the best approach to answering your initial question. Looking at the unique values on each column can help identify how you might want to group or filter the data.

19. Use DISTINCT to test if there are any accounts associated with more than one region.

We will run and compare the following two queries:

```sql
SELECT COUNT(*)
FROM accounts a
JOIN sales_reps s
ON s.id = a.sales_rep_id
JOIN region r
ON r.id = s.region_id;
```

count
351

and query:

```sql
SELECT COUNT(DISTINCT id)
FROM accounts;
```

count
351

20. **YOUR TURN** How many accounts assigned to sales reps in the Midwest region have placed an order? You should be returning one row, with one column called `midwest_accounts`.

HAVING

The WHERE clause doesn't allow you to filter on aggregate columns, that's where the HAVING clause comes in. The HAVING clause comes after GROUP BY and before ORDER BY.

21. How many of the sales reps have more than 5 accounts that they manage?

```sql
SELECT s.name, COUNT(*) num_accounts
FROM accounts a
JOIN sales_reps s
ON s.id = a.sales_rep_id
GROUP BY s.name
HAVING COUNT(*) > 5
ORDER BY num_accounts;
```

name	num_accounts
Elba Felder	6
Samuel Racine	6
Eugena Esser	6
Sibyl Lauria	6
Necole Victory	6
Debroah Wardle	6
Cliff Meints	7
Soraya Fulton	7
Elna Condello	7
Charles Bidwell	7
Gianna Dossey	7
Babette Soukup	7
Nelle Meaux	7
Derrick Boggess	7
Michel Averette	7
Tia Amato	8
Julia Behrman	8
Delilah Krum	8
Dawna Agnew	9
Elwood Shutt	9
Marquetta Laycock	9
Hilma Busick	10
Saran Ram	10
Brandie Riva	10
Moon Torian	10
Arica Stoltzfus	10
Dorotha Seawell	11
Earlie Schleusner	11
Maren Musto	11
Vernita Plump	11
Maryanna Fiorentino	11
Micha Woodford	11
Calvin Ollison	11
Georgianna Chisholm	15

Technically, we can get this using a SUBQUERY as shown below:

```sql
SELECT COUNT(*) num_reps_above5
FROM(SELECT s.name, COUNT(*) num_accounts
     FROM accounts a
     JOIN sales_reps s
     ON s.id = a.sales_rep_id
     GROUP BY s.name
     HAVING COUNT(*) > 5
     ORDER BY num_accounts) AS Table1;
```

num_reps_above5
34

22. **YOUR TURN** How many accounts have more than 20 orders?

23. **YOUR TURN** Which account has the most orders? Return one row, with columns `id` (the account ID), `name` (the account name), and `num_orders` (the number of orders for the account with the most orders.). Hint: Use a `LIMIT 1` in your query.

24. How many accounts spent more than 30,000 usd total across all orders?

```
SELECT a.id, a.name, SUM(o.total_amt_usd) total_spent
FROM accounts a
JOIN orders o
ON a.id = o.account_id
GROUP BY a.id, a.name
HAVING SUM(o.total_amt_usd) > 30000
ORDER BY total_spent
LIMIT 5;
```

id	name	total_spent
1661	American Airlines Group	30083.18
1431	PepsiCo	30095.72
3661	Group 1 Automotive	30708.92
1141	Costco	30741.01
1761	Oracle	31231.56

25. **YOUR TURN** Which account has spent the most with us? Provide three columns in your answer - `id` (the account ID), `name` (name of the account), and `total_spent`. Use `total_amt_usd` to calculate spend.

26. Which accounts used facebook as a channel to contact customers more than 6 times?

```
SELECT a.id, a.name, w.channel, COUNT(*) use_of_channel
FROM accounts a
JOIN web_events w
ON a.id = w.account_id
GROUP BY a.id, a.name, w.channel
HAVING COUNT(*) > 6 AND w.channel = 'facebook'
ORDER BY use_of_channel
LIMIT 5;
```

id	name	channel	use_of_channel
1221	J.P. Morgan Chase	facebook	7
3231	Parker-Hannifin	facebook	7
3991	eBay	facebook	7
1261	Wells Fargo	facebook	7
4241	Laboratory Corp. of America	facebook	7

27. Which account used facebook most as a channel?

```sql
SELECT a.id, a.name, w.channel, COUNT(*) use_of_channel
FROM accounts a
JOIN web_events w
ON a.id = w.account_id
WHERE w.channel = 'facebook'
GROUP BY a.id, a.name, w.channel
ORDER BY use_of_channel DESC
LIMIT 1;
```

id	name	channel	use_of_channel
1851	Gilead Sciences	facebook	16

28. Which channel was most frequently used by most accounts?

```sql
SELECT a.id, a.name, w.channel, COUNT(*) use_of_channel
FROM accounts a
JOIN web_events w
ON a.id = w.account_id
GROUP BY a.id, a.name, w.channel
ORDER BY use_of_channel DESC
LIMIT 10;
```

id	name	channel	use_of_channel
3411	Leucadia National	direct	52
2731	Colgate-Palmolive	direct	51
1601	New York Life Insurance	direct	51
2051	Philip Morris International	direct	49
3491	BlackRock	direct	48
2871	FirstEnergy	direct	48
2351	AutoNation	direct	48
3471	ADP	direct	48
3911	Charter Communications	direct	48
2481	Altria Group	direct	47

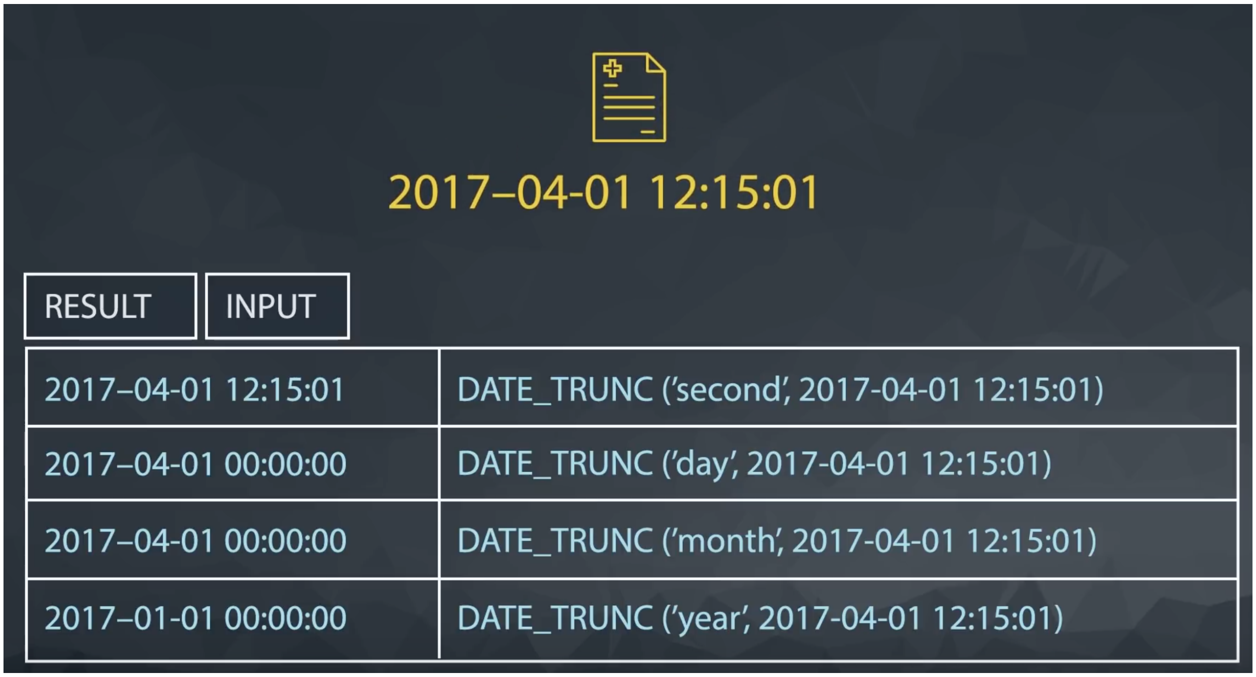
DATE Function with GROUP BY

GROUPing BY a date column is not usually very useful in SQL, as these columns tend to have transaction data down to a second. Keeping date information at such a granular data is both a blessing and a curse, as it gives really precise information (a blessing), but it makes grouping information together directly difficult (a curse).

Lucky for us, there are a number of built in SQL functions that are aimed at helping us improve our experience in working with dates:

- **DATE_TRUNC** allows you to truncate your date to a particular part of your date-time column. Common trunctions are day, month, and year.
- **DATE_PART** can be useful for pulling a specific portion of a date, but notice pulling month or day of the week (dow) means that you are no longer keeping the years in order. Rather you are grouping for certain components regardless of which year they belonged in.

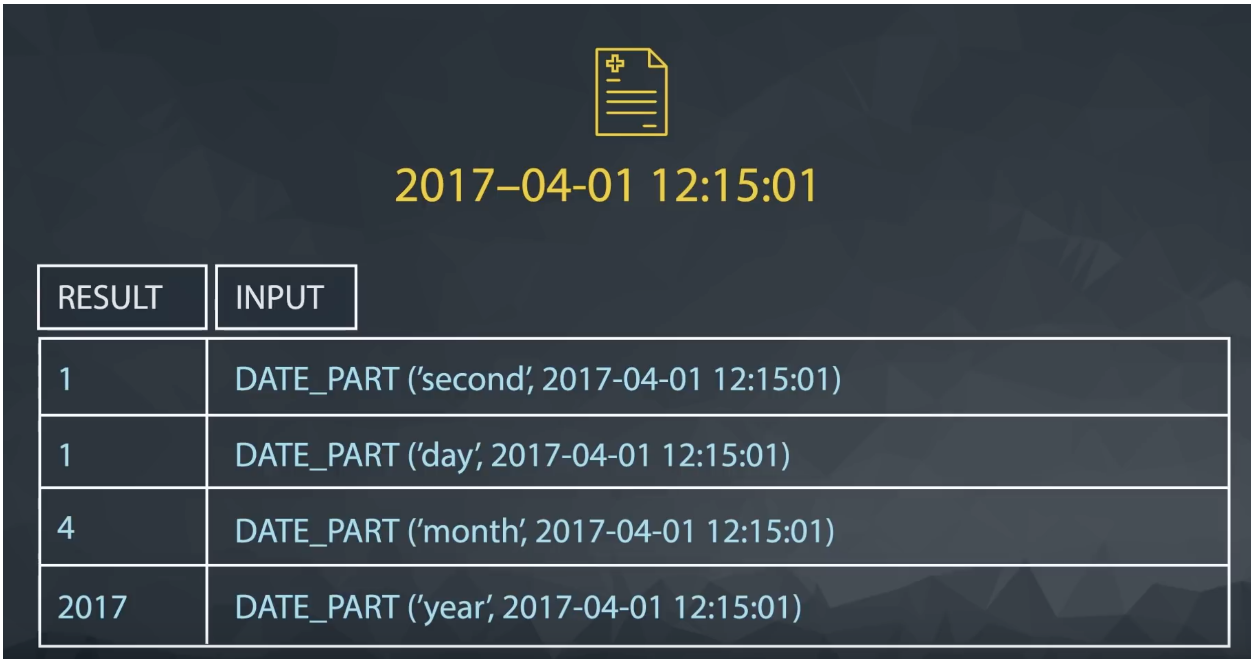
29. Find the sales in terms of total dollars for all orders in each year, ordered from greatest to least. Do you notice any trends in the yearly sales totals?



2017–04-01 12:15:01

RESULT	INPUT
2017–04-01 12:15:01	DATE_TRUNC (‘second’, 2017-04-01 12:15:01)
2017–04-01 00:00:00	DATE_TRUNC (‘day’, 2017-04-01 12:15:01)
2017–04-01 00:00:00	DATE_TRUNC (‘month’, 2017-04-01 12:15:01)
2017–01-01 00:00:00	DATE_TRUNC (‘year’, 2017-04-01 12:15:01)

Figure 2: DATE_TRUNC



2017–04-01 12:15:01

RESULT	INPUT
1	DATE_PART (‘second’, 2017-04-01 12:15:01)
1	DATE_PART (‘day’, 2017-04-01 12:15:01)
4	DATE_PART (‘month’, 2017-04-01 12:15:01)
2017	DATE_PART (‘year’, 2017-04-01 12:15:01)

Figure 3: DATE_PART

```sql
SELECT DATE_PART('year', occurred_at) ord_year,  SUM(total_amt_usd) total_spent
FROM orders
GROUP BY 1 -- 1 refers to the first argument in the SELECT statement (ord_year)
ORDER BY 2 DESC -- 2 refers to the second argument (total_spent)
LIMIT 5;
```

ord_year	total_spent
2016	12864917.92
2015	5752004.94
2014	4069106.54
2013	377331.00
2017	78151.43

30. Which month did Parch & Posey have the greatest sales in terms of total dollars?

```sql
SELECT DATE_PART('month', occurred_at) ord_month, SUM(total_amt_usd) total_spent
FROM orders
GROUP BY 1
ORDER BY 2 DESC
LIMIT 5;
```

ord_month	total_spent
12	3129412
10	2427506
11	2390034
9	2017217
7	1978731

31. **YOUR TURN** Which year did Parch & Posey have the greatest sales in terms of total number of orders? Are all years evenly represented by the dataset? Return two columns - `ord_year` for the order year, and `total_sales`, the number of orders that occurred in that year.
32. **YOUR TURN** In which month of which year did Walmart spend the most on gloss paper in terms of dollars? Return only one row, with two columns - the `order_month` and `tot_spent`(total spent on gloss in USD).

CASE

The CASE statement is SQL's way of handling if/then logic. The CASE statement is followed by at least one pair of WHEN and THEN statements—SQL's equivalent of IF/THEN in Excel. Because of this pairing, you might be tempted to call this SQL CASE WHEN, but CASE is the accepted term.

Every CASE statement must end with the END statement. The ELSE statement is optional, and provides a way to capture values not specified in the WHEN/THEN statements.

- The CASE statement always goes in the SELECT clause
- CASE must include the following components: WHEN, THEN, and END. ELSE is an optional component.
- You can make any conditional statement using any conditional operator (like WHERE) between WHEN and THEN. This includes stringing together multiple conditional statements using AND and OR.
- You can include multiple WHEN statements, as well as an ELSE statement to deal with any unaddressed conditions.

33. **YOUR TURN** We would like to understand 3 different branches of customers based on the amount associated with their purchases. The top branch includes anyone with a Lifetime Value (total sales

of all orders) greater than 200,000 usd. The second branch is between 200,000 and 100,000 usd. The lowest branch is anyone under 100,000 usd. Provide a table that includes the level associated with each account. You should provide a column called `name` (account name), `total_spent`, the total sales of all orders for the customer, and `customer_level` (the level categorization of the customer). Order with the top spending customers listed first.

34. We would now like to perform a similar calculation to the first, but we want to obtain the total amount spent by customers only in 2016 and 2017. Keep the same levels as in the previous question. Order with the top spending customers listed first.

```
SELECT a.name, SUM(total_amt_usd) total_spent,
     CASE WHEN SUM(total_amt_usd) > 200000 THEN 'top'
     WHEN  SUM(total_amt_usd) > 100000 THEN 'middle'
     ELSE 'low' END AS customer_level
FROM orders o
JOIN accounts a
ON o.account_id = a.id
WHERE occurred_at > '2015-12-31'
GROUP BY 1
ORDER BY 2 DESC
LIMIT 5;
```

name	total_spent	customer_level
Pacific Life	255319.2	top
Mosaic	172180.0	middle
CHS	163471.8	middle
Core-Mark Holding	148105.9	middle
Disney	129157.4	middle

35. We would like to identify top performing sales reps, which are sales reps associated with more than 200 orders. Create a table with the sales rep name, the total number of orders, and a column with top or not depending on if they have more than 200 orders. Place the top sales people first in your final table.

```
SELECT s.name, COUNT(*) num_ords,
     CASE WHEN COUNT(*) > 200 THEN 'top'
     ELSE 'not' END AS sales_rep_level
FROM orders o
JOIN accounts a
ON o.account_id = a.id
JOIN sales_reps s
ON s.id = a.sales_rep_id
GROUP BY s.name
ORDER BY 2 DESC
LIMIT 5;
```

name	num_ords	sales_rep_level
Earlie Schleusner	335	top
Vernita Plump	299	top
Tia Amato	267	top
Georgianna Chisholm	256	top
Moon Torian	250	top

36. **YOUR TURN** The previous query didn't account for the middle, nor the dollar amount associated with the sales. Management decides they want to see these characteristics represented as well. We would like to identify top performing sales reps, which are sales reps associated with more than 200

orders or more than 750000 in total sales. The middle group has any rep with more than 150 orders or 500000 in sales. Create a table with the sales rep name, the total number of orders, total sales across all orders, and a column with top, middle, or low depending on this criteria. Place the top sales people based on dollar amount of sales first in your final table. There should be 4 columns - `name`, `num_orders`, `total_spent`, and `sales_rep_level`.

Subqueries

Subqueries (also known as inner queries or nested queries) are a tool for performing operations in multiple steps. For example, if you wanted to take the sums of several columns, then average all of those values, you'd need to do each aggregation in a distinct step.

37. Show all orders (all columns) that have an above average total USD amount value. Return the order ID, account ID, and total_amt_usd columns.

```
SELECT id, account_id, total_amt_usd
FROM orders
WHERE total_amt_usd > (SELECT AVG(total_amt_usd)
FROM orders)
LIMIT 5
```

id	account_id	total_amt_usd
24	1031	7474.32
37	1071	4393.44
48	1081	3405.62
88	1101	3585.33
129	1141	26055.46

38. What is the total amount of revenue (calculated as the sum of total_amt_usd) generated by Parch and Posey's top 3 accounts? Top 3 accounts here is defined as the accounts that have the largest number of orders. Hint - first find the accounts that have the largest number of orders.

```
SELECT SUM(total_amt_usd) AS total_revenue
FROM orders
WHERE account_id IN (
    SELECT account_id
    FROM orders
    GROUP BY account_id
    ORDER BY COUNT(DISTINCT id) DESC
    LIMIT 3);
```

total_revenue
844911.2

Note that you can also write this using a part of a nested table subquery:

```
SELECT SUM(revenue) AS total_revenue
FROM (  SELECT account_id, SUM(total_amt_usd) revenue
    FROM orders
    GROUP BY account_id
    ORDER BY COUNT(DISTINCT id) DESC
    LIMIT 3) revenue_totals;
```

total_revenue
844911.2

39. **YOUR TURN** Parch and Posey employees are often evaluated based on how many web events they are able to generate for their accounts. Compute the average number of web events generated per account. Hint - first compute the number of web events for each account (so you should have one number for each account). Then use this result to compute the average number of web events overall per account. You should return only one row, with one column (a scalar). The column name should be avg_num_web_events.

40. **YOUR TURN** How many total orders have been processed by sales reps who are assigned to the Midwest region and have more than 50 orders processed? Hint - first find the number of orders for sales reps assigned to the Midwest region. Then filter the grouped rows for only those with 2 or more orders to get the sales reps who have made 50 or more orders.

Your final result should be one row, one column (a scalar) titled `total_orders`.