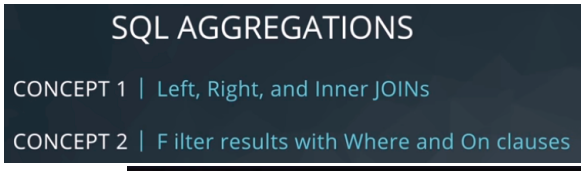
Bertelsmann Tech Scholarship - Data Track

LESSON 28 SQL Aggregation

In this lesson, you will learn how to aggregate data using SQL functions like SUM, AVG, and COUNT. Additionally, CASE, HAVING, and DATE functions provide you an incredible problem solving toolkit.




```
SELECT *
  FROM demo.accounts
 WHERE primary_poc IS NULL
```

28.1 Video: Introduction to Aggregation
- Aggregate functions that are used in SQL
- All SQL fns down to col

28.2 Video: Introduction to NULLs = NO DATA

NULLs = datatype that specifies where no data exists in SQL, often ignored in aggregation fns.

28.3 Video: NULLs and Aggregation
- NULLs different than a zero - they are cells where data does not exist.
- Identifying NULLs in a WHERE clause, we write IS NULL or IS NOT NULL, don't use =, because NULL isn't considered a value in SQL. Rather, it is a property of the data.
- NULLs frequently occur when performing a LEFT or RIGHT JOIN. Empty cells are NULL values in the result set.
- NULLs can also occur from simply missing data in db.

28.4 First Aggregation - COUNT

COUNT the Number of Rows in a Table >>

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

463

chosen a col to drop into aggreg fn:

```
SELECT COUNT(accounts.id)
FROM accounts;
```

These two statements are equivalent, but this isn't always the case

28.5 COUNT & NULLs
- COUNT does not consider rows that have NULL values. It is useful for quickly identifying which rows have missing data. Will learn GROUP BY in an upcoming concept, and then each of these aggregators will become much more useful.
- COUNT fn is count for non NULL data. > TEXT is not NULL

```
SELECT SUM(standard_qty) AS standard,
       SUM(gloss_qty) AS gloss,
       SUM(poster_qty) AS poster
  FROM demo.orders
```

28.6 SUM
- Unlike COUNT, use SUM on numeric columns, SUM will ignore NULL values, as other aggregation fns.
- Aggregators only aggregate vertically - the values of a col. To perform a calculation across rows, use simple arithmetic.

1- SELECT SUM(poster_qty) poster
   FROM orders

2- SELECT SUM(standard_qty) standard
   FROM orders

3- SELECT SUM(total_amt_usd) AS total_dollar_sales
   FROM orders;

4- SELECT standard_amt_usd + gloss_amt_usd
         AS total_standard_gloss
   FROM orders;

used both an aggregate and our mathematical operators

5- SELECT SUM(standard_amt_usd)/SUM(standard_qty)
AS standard_price_per_unit
FROM orders;

28.9 MIN & MAX
- We simultaneously obtaining MIN & MAX number of orders of each paper type. We could run each individually.
- MIN and MAX are aggregators that ignore NULL values.
- Functionally, MIN & MAX are 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 early in the alphabet as possible. MAX does the opposite—it returns the highest number, the latest date, or the non-numerical value closest alphabetically to "Z."

```
SELECT MIN(standard_qty) AS standard_min,
       MIN(gloss_qty) AS gloss_min,
       MIN(poster_qty) AS poster_min,
       MAX(standard_qty) AS standard_max,
       MAX(gloss_qty) AS gloss_max,
       MAX(poster_qty) AS poster_max
  FROM demo.orders
```

28.10 AVG >> ignores the NULL values in both the numerator and the denominator.
- To count NULLs as zero, will need to use SUM and COUNT.
- Median = measure of center for this data, but difficult thing to get using SQL alone.

simple arithmetic. https://mode.com/sql-tutorial/sql-operators/#arithmetic-in-sql

Bertelsmann Tech Scholarship - Data Track

28.11 Quiz: MIN, MAX, & AVG

```
SELECT MIN(occurred_at)
FROM orders;
```
1

```
SELECT occurred_at
FROM orders
ORDER BY occurred_at
LIMIT 1;
```
2

```
SELECT MAX(occurred_at)
FROM web_events;
```
3

```
SELECT occurred_at
FROM web_events
ORDER BY occurred_at DESC
LIMIT 1
```
4

```
SELECT AVG(standard_qty) mean_standard,
       AVG(gloss_qty) mean_gloss,
       AVG(poster_qty) mean_poster,
       AVG(standard_amt_usd) mean_standard_usd,
       AVG(gloss_amt_usd) mean_gloss_usd,
       AVG(poster_amt_usd) mean_poster_usd
FROM orders;
```
5

```
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;
```
6

Since there are 6912 orders - we want the average of the 3457 and 3456 order amounts when ordered. This is the average of 2483.16 and 2482.55. This gives the median of 2482.855. This obviously isn't an ideal way to compute. If we obtain new orders, we would have to change the limit. SQL didn't even calculate the median for us. The above used a SUBQUERY, but you could use any method to find the two necessary values, and then you just need the average of them.

28.13 GROUP BY used to aggregate data within subsets of the data. Any col in SELECT state that is not within an aggregator must be in GROUP BY clause.

"GROUP BY" CLAUSE GOES IN-BETWEEN THE "WHERE" AND "ORDER" CLAUSE

GROUP BY always goes between WHERE and ORDER BY.

ORDER BY works like SORT in spreadsheet software.

- SQL evaluates the aggregations before the LIMIT clause. If you don't group by any columns, you'll get a 1-row result. If you group by a column with enough unique values that it exceeds the LIMIT number, the aggregates will be calculated, and then some rows will simply be omitted from the results.
- In next concept, use SQL env to removing the LIMIT and running it again to see what changes.

28.15 Solutions: GROUP BY

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

```
SELECT a.name, SUM(total_amt_usd) total_sales
FROM orders o
JOIN accounts a
ON a.id = o.account_id
GROUP BY a.name;
```
2

```
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;
```
3

```
SELECT w.channel, COUNT(*)
FROM web_events w
GROUP BY w.channel
```
4

```
SELECT a.primary_poc
FROM web_events w
JOIN accounts a
ON a.id = w.account_id
ORDER BY w.occurred_at
LIMIT 1;
```
5

```
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;
```
6

```
SELECT r.name, COUNT(*) num_reps
FROM region r
JOIN sales_reps s
ON r.id = s.region_id
GROUP BY r.name
ORDER BY num_reps;
```
7

Bertelsmann Tech Scholarship - Data Track

28.16 Video: GROUP BY Part II

- Can GROUP BY multiple cols at once. Order of columns listed in the ORDER BY clause does make a difference, from left to right.
- Any column not within an aggregation must show up in GROUP BY statement.

GROUP BY AND ORDER BY CAN BE USED WITH MULTIPLE COLUMNS IN THE SAME QUERY

THE ORDER IN THE ORDER BY DETERMINES WHICH COLUMN IS ORDERED ON FIRST

YOU CAN ORDER DESC FOR ANY COLUMN IN YOUR ORDER BY

28.17 Quiz: GROUP BY Part II

```
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;
```

1

```
SELECT a.name,
 AVG(o.standard_amt_usd) avg_stand,
     AVG(o.gloss_amt_usd) avg_gloss,
    AVG(o.poster_amt_usd) avg_post
FROM accounts a
JOIN orders o
ON a.id = o.account_id
GROUP BY a.name;
```

2

```
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;
```

3

```
SELECT r.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
JOIN region r
ON r.id = s.region_id
GROUP BY r.name, w.channel
ORDER BY num_events DESC;
```

4

28.19 Video: DISTINCT always used in SELECT statements, provides the unique rows for all cols written in SELECT. Use DISTINCT once in any particular SELECT statement.

- Think of DISTINCT same way we think of the statement "unique".
- Using DISTINCT, particularly in aggregations, can slow your queries down quite a bit.

```
SELECT DISTINCT account_id,
       channel
  FROM demo.web_events
 ORDER BY account_id
```

28.20 Quiz: DISTINCT

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

```
SELECT a.id as "account id", r.id as "region id",
a.name as "account name", r.name as "region name"
FROM accounts a
JOIN sales_reps s
ON s.id = a.sales_rep_id
JOIN region r
ON r.id = s.region_id;
```

1

and

```
SELECT DISTINCT id, name
FROM accounts;
```

Have any sales reps worked on more than one account?

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

2

and

```
SELECT DISTINCT id, name
FROM sales_reps;
```

28.22 Video: HAVING

is the "clean" way to filter a query has been aggregated, but this is also commonly done using a subquery. Essentially, any time you want to perform a WHERE on an element of your query that was created by an aggregate, you need to use HAVING instead.

```
SELECT account_id,
       SUM(total_amt_usd) AS sum_total_amt_usd
  FROM demo.orders
 GROUP BY 1
 HAVING SUM(total_amt_usd) >= 250000
```

- [x] **WHERE** subsets the returned data based on a logical condition.
- [x] **WHERE** appears after the **FROM**, **JOIN**, and **ON** clauses, but before **GROUP BY**.
- [x] **HAVING** appears after the **GROUP BY** clause, but before the **ORDER BY** clause.
- [x] **HAVING** is like **WHERE**, but it works on logical statements involving aggregations.

28.23 Questions: HAVING

1. How many of the sales reps have more than 5 accounts that they manage?
and technically, we can get this using a SUBQUERY as shown below. This same logic can be used for the other queries, but this will not be shown ->>

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

1

and

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

1

2. How many accounts have more than 20 orders?

```
SELECT a.id, a.name, COUNT(*) num_orders
FROM accounts a
JOIN orders o
ON a.id = o.account_id
GROUP BY a.id, a.name
HAVING COUNT(*) > 20
ORDER BY num_orders;
```

2

3. Which account has the most orders?

```
SELECT a.id, a.name, COUNT(*) num_orders
FROM accounts a
JOIN orders o
ON a.id = o.account_id
GROUP BY a.id, a.name
ORDER BY num_orders DESC
LIMIT 1;
```

3

4. Which accs spent > 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;
```

4

5. Which accs spent < 1,000 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) < 1000
ORDER BY total_spent;
```

5

6. Which account has spent the most with us?

```
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
ORDER BY total_spent DESC
LIMIT 1;
```

6

7. Which account has spent the least with us?

```
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
ORDER BY total_spent
LIMIT 1;
```

7

8. 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;
```

8

9. Which account used facebook most as a channel?

```
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;
```

9

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

```
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;
```

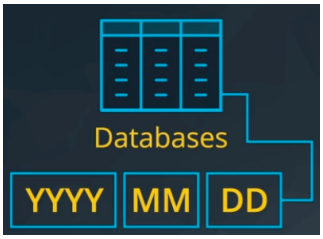
10

Bertelsmann Tech Scholarship - Data Track

28.25 Video: DATE Functions

GROUPing BY a date col is not very useful in SQL, as these columns tend to have transaction data down to a second. Keeping date info at such is blessing and a curse, as it gives really precise info (a blessing), but it makes grouping info together directly difficult (a curse). > Lucky, a number of built in SQL fn, aimed at help us improve in working with dates.> Dates are stored in year, month, day, hour, minute, second, which helps us in truncating.

- Next 'll see some fns in SQL to take advand of this functionality.

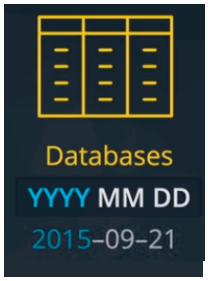


28.26 Video: DATE Functions II

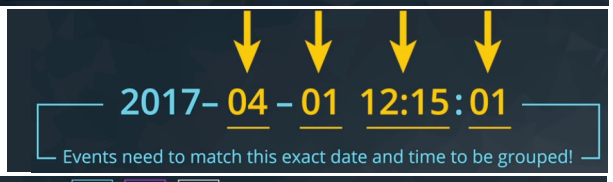
DATE_TRUNC fn allows to truncate date to a particular part of your date-time column. Common trunctions are day, month, and year.

- A blog post by Mode Analytics on the power of this function.

https://mode.com/blog/date-trunc-sql-timestamp-function-count-on



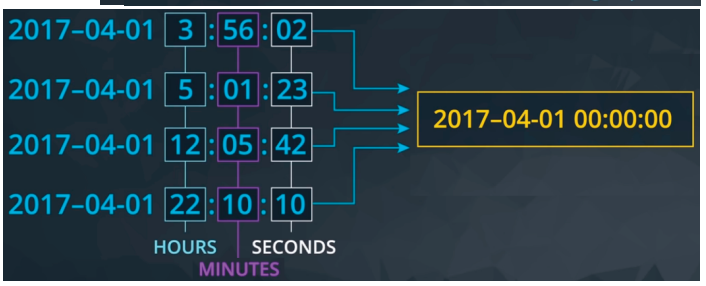
DATE_PART useful for pulling a specific portion of a date, but pull month/ day of week (dow) means not keep year in order. We group certain components regardless of which year they belonged in.



Additional fns use w/ dates:

> Reference cols in select state in GROUP BY and ORDER BY clauses w/ numbers, follow the order they appear in select statement.

```
SELECT standard_qty, COUNT(*)
FROM orders
GROUP BY 1 (1 refers to standard_qty since it is the 1st of the cols included in select statement)
ORDER BY 1 (1 refers to standard_qty since it is the 1st of the cols included in select statement)
```




```
SELECT DATE_TRUNC('day',occurred_at) AS day,
       SUM(standard_qty) AS standard_qty_sum
  FROM demo.orders
 GROUP BY DATE_TRUNC('day',occurred_at)
 ORDER BY DATE_TRUNC('day',occurred_at)
```

'DOW' PULLS THE DAY OF THE WEEK WITH 0 AS SUNDAY AND 6 AS SATURDAY

```
SELECT DATE_PART('dow',occurred_at) AS day_of_week,
       SUM(total) AS total_qty
  FROM demo.orders
 GROUP BY 1
 ORDER BY 2 DESC
```

THE 1 AND 2 HERE IDENTIFY THESE COLUMNS IN THE SELECT STATEMENT

28.27 Quiz: DATE Functions

1. 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?

```
SELECT DATE_PART('year', occurred_at) ord_year,
          SUM(total_amt_usd) total_spent
 FROM orders
 GROUP BY 1
 ORDER BY 2 DESC;
```

2. Which month did Parch & Posey have the greatest sales in terms of total dollars? Are all months evenly represented by the dataset?

```
SELECT DATE_PART('month', occurred_at) ord_month,
          SUM(total_amt_usd) total_spent
FROM orders
WHERE occurred_at BETWEEN '2014-01-01'
          AND '2017-01-01'
GROUP BY 1
ORDER BY 2 DESC;
```

3. Which year did Parch & Posey have the greatest sales in terms of total number of orders? Are all years evenly represented by the dataset?

```
SELECT DATE_PART('year', occurred_at) ord_year,
          COUNT(*) total_sales
FROM orders
GROUP BY 1
ORDER BY 2 DESC;
```

4. Which month did Parch & Posey have the greatest sales in terms of total number of orders? Are all months evenly represented by the dataset?

```
SELECT DATE_PART('month', occurred_at) ord_month,
          COUNT(*) total_sales
FROM orders
WHERE occurred_at BETWEEN '2014-01-01'
          AND '2017-01-01'
GROUP BY 1
ORDER BY 2 DESC;
```