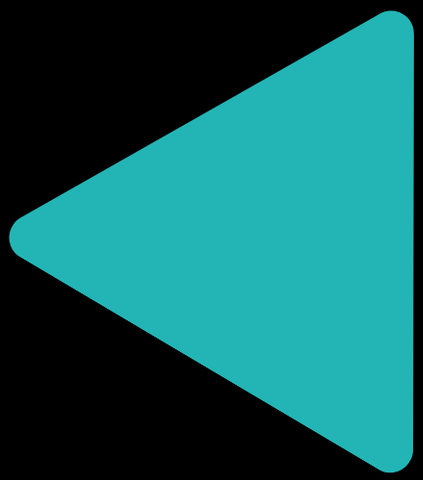
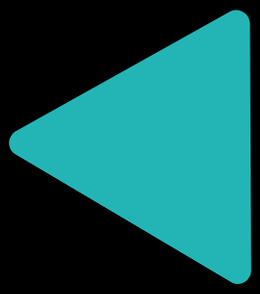


Day 4

Window Functions



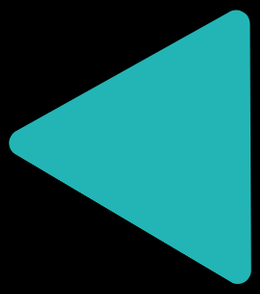
Aggregate Functions

Unlike aggregate functions, which aggregate all rows and output one single row value representing all data, window functions iterate over all the rows and return the value but do not aggregate them into one final output.

This is an aggregate function and its output.

```
SELECT SUM(list_price)
FROM production.products;
```

	sum_list_price
1	488109.84



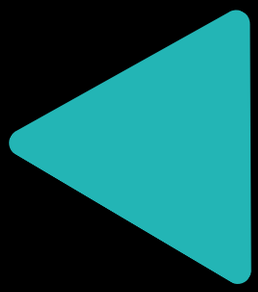
Window Functions

Window function on the other hand will return the total but not actually group all the fields into one.

This is the most basic window function and it uses the keyword **OVER**.

```
SELECT list_price
     , sum(list_price) over()
FROM production.products;
```

	list_price	sum_list_price
1	379.99	488109.84
2	749.99	488109.84
3	999.99	488109.84
4	2899.99	488109.84
5	1320.99	488109.84
6	469.99	488109.84
7	3999.99	488109.84
8	1799.99	488109.84
9	2999.99	488109.84
10	1549.00	488109.84
11	1680.99	488109.84

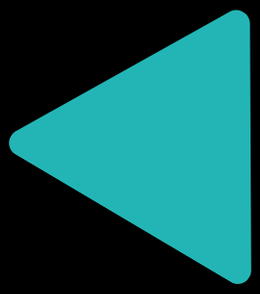


Syntax

window_function_name (expression) OVER (partition_clause order_clause frame_clause).

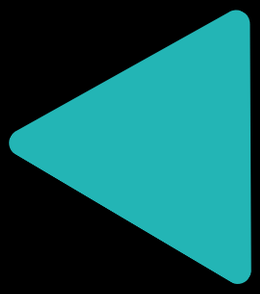
This is the syntax, I will not go into frame_clause at this moment.

- Window Function Name
  - **SUM**, **RANK**, **DENSE_RANK**, **ROW_NUMBER**
- Partition Clause
  - **PARTITION BY**
- Order Clause
  - **ORDER BY**



Function Name

- **SUM, AVG, COUNT, MAX, MIN**
  - Aggregates all rows together by the partition.
- **RANK**
  - Assigns a rank to all rows by the partition.
  - Equal values get the same rank and data could have gaps (1, 1, 3, 4, 5).
- **DENSE_RANK**
  - Same as rank, but there are no gaps (1, 1, 2, 3, 4).
-



Function Name

- **NTILE**
  - Allows you to specify how many buckets to split the results in.
- **ROW_NUMBER**
  - Will assign a sequential number to each row.

Partition

The partition is where the data gets rolled up to.

If you want to add all the values per brand then you would run this statement.

```
SELECT product_name
      , brand_id
      , list_price
      , SUM(list_price) OVER (PARTITION BY brand_id)
FROM production.products
```

	product_name	brand_id	list_price	total_per_brand
1	Electra Townie Original 21D - 2016	1	549.99	89798.73
2	Electra Cruiser 1 (24-Inch) - 2016	1	269.99	89798.73
3	Electra Girl's Hawaii 1 (16-inch) - 2015/2016	1	269.99	89798.73
4	Electra Moto 1 - 2016	1	529.99	89798.73
5	Electra Townie Original 7D EQ - 2016	1	599.99	89798.73

Order By

Once the data is partitioned you can order it so that it has some structure.

The example is partitioned by brand but then grouped by category within that.

```
SELECT product_name
      , brand_id
      , list_price
      , SUM(list_price) OVER (PARTITION BY brand_id ORDER BY category_id)
FROM production.products
```

113	Electra Townie Balloon 7i EQ - 2018	1	3	899.99	75798.78
114	Electra Loft Go! 8i - 2018	1	5	2799.99	89798.73
115	Electra Townie Go! 8i - 2017/2018	1	5	2599.99	89798.73
116	Electra Townie Commute Go! - 2018	1	5	2999.99	89798.73
117	Electra Townie Commute Go! Ladies' - 2018	1	5	2999.99	89798.73
118	Electra Townie Go! 8i Ladies' - 2018	1	5	2599.99	89798.73
119	Haro Shredder 20 - 2017	2	1	209.99	999.96
120	Haro Shredder 20 Girls - 2017	2	1	209.99	999.96
121	Haro Shredder Pro 20 - 2017	2	1	249.99	999.96