Intro to GROUP BY

In this lesson we are going into some detail about what the **group by** statement does.

Lets consider this simple example which is just a small random sample from the table `bigquery-public-data.chicago_taxi_trips.taxi_trips` . Lets call this sample table `TABLE` for the purposes of this explanation.

unique_key	payment_type	trip_total	trip_
c68ac4d3ba984a5fba226329bbc7bf4678e13c3d	Cash	36	1140
4c9704f13c885f1e01e8a5db2e9fcf8014935e39	Cash	34	180
d961af189a638d0bddeb86f350e8ca8150be5d2e	Cash	26	1920
baeac2f014728809cf2e95cf3d4a89f0d114b733	Credit Card	65	1200
2185d653bf5b9cef15fe77cc9f9ed5f3b86510f2	Credit Card	54	242
a2fb1becd90b2db2e639cccddf6a6715a45e82a1	Credit Card	23	540
485b57285eec3cdb2bcf3545df6dc87bbda59d61	Mobile	55	1112
9068d21febca381a5f505ada1e69af62a1d5a7a8	Mobile	50	204

When I recorded this video I thought the trip_total column from `bigquery-public-data.chicago_taxi_trips.taxi_trips` was in dollars. It is actually in cents. You need to divide this column by 100 to convert it to dollars. I also put this note in the corresponding video.

Grouping By Column Returns Unique List

The query

```
SELECT
    payment_type
FROM
    TABLE
GROUP BY
    payment_type
ORDER BY
    payment_type
```

would return

payment_type
Cash
Credit Card
Mobile

So **GROUP BY** scans the table and returns a distinct list of the values in the column in which your grouping by (in this case *payment_type*). When we **GROUP BY** a column, we need to return that column within the **SELECT** portion of the query as well. So in this example, all the records with *Cash* go in one group, *Credit Card* in a second group, and *Mobile* in a third group.

Aggregate Functions

The real power comes when we apply an aggregate function over each group. This is known as aggregating. For example we may want to know:

- What was the maximum *trip_total* for each *payment_type* group?
- What was the minimum *trip_total* for each *payment_type* group?
- What was the average *trip_total* for each *payment_type* group?
- What was the total sum of *trip_total* for each *payment_type* group?
- How many taxi trips were there for each *payment_type* group?

We can use the **MAX**, **MIN**, **AVG**, **SUM**, and **COUNT** functions along with the **GROUP BY** statement to achieve these results.

MAX

For example, consider this query:

```
SELECT
    payment_type,
    MAX(trip_total) as max_trip_total
FROM
    TABLE
GROUP BY
    payment_type
ORDER BY
    payment_type
```

In the above query we find the **maximum** *trip_total* for each of the three *payment_type* groups. It would return:

payment_type	max_trip_total
Cash	36
Credit Card	65
Mobile	55

So the **GROUP BY** first creates these three groups (in the context of performing an aggregate on the *trip_total* column.)

- GROUP 1: 'Cash'
  - with values 26, 34, 36 from the *trip_total* column
  - the **MAX** function would return the maximum value i.e. **36**
- GROUP 2: 'Credit Card'
  - with values 23, 54, 65 from the *trip_total* column
  - the **MAX** function would return the maximum value i.e. **65**
- GROUP 3: 'Mobile'
  - with values 50, 55 from the *trip_total* column
  - the **MAX** function would return the maximum value i.e. **55**

MIN

- similar to the MAX except that it returns the **minimum** value of the numeric column.

```
SELECT
    payment_type,
    MIN(trip_total) as min_trip_total
FROM
    TABLE
GROUP BY
    payment_type
ORDER BY
    payment_type
```

would return

payment_type	min_trip_total
Cash	26
Credit Card	23
Mobile	50

AVG

The **AVG** function computes the **average** value for the numeric column.

```
SELECT
    payment_type,
    AVG(trip_total) as avg_trip_total
FROM
    TABLE
GROUP BY
    payment_type
ORDER BY
    payment_type
```

In the above query we find the **average** *trip_total* for each of the three *payment_type* groups. It would return:

payment_type	avg_trip_total
Cash	32
Credit Card	47.3333333333
Mobile	52.5

So the **GROUP BY** first creates these three groups and then takes the average of the values for the *trip_total* within each group.

- GROUP 1: 'Cash'
  - with values 26, 34, 36 from the *trip_total* column
  - the **AVG** function would return `(26 + 34 + 36)/3 = 32`
- GROUP 2: 'Credit Card'
  - with values 23, 54, 65 from the *trip_total* column
  - the **AVG** function would return `(23 + 54 + 65)/3 = 47.3333333333`
- GROUP 3: 'Mobile'
  - with values 50, 55 from the *trip_total* column
  - the **AVG** function would return `(50 + 55)/2 = 52.5`

SUM

The **SUM** function computes the sum of all the values for the numeric column.

```
SELECT
    payment_type,
    SUM(trip_total) as sum_trip_total
FROM
    TABLE
GROUP BY
    payment_type
```

```
ORDER BY
    payment_type
```

In the above query we find the **SUM** of *trip_total* for each of the three *payment_type* groups. It would return:

payment_type	sum_trip_total
Cash	96
Credit Card	142
Mobile	105

So the **GROUP BY** first creates these three groups and then takes the **SUM** of the values for the *trip_total* within each group.

- GROUP 1: 'Cash'
  - with values 26, 34, 36 from the *trip_total* column
  - the **SUM** function would return `(26 + 34 + 36) = 96`
- GROUP 2: 'Credit Card'
  - with values 23, 54, 65 from the *trip_total* column
  - the **SUM** function would return `(23 + 54 + 65) = 142`
- GROUP 3: 'Mobile'
  - with values 50, 55 from the *trip_total* column
  - the **SUM** function would return `(50 + 55) = 105`

COUNT

The `count(*)` function counts the number of records/rows.
The `count(distinct column_name)` function counts the number of distinct records.
Basically, the same thing it did before when we first introduced it. However, when used with GROUP BY the **COUNT** function is applied to each group separately.

For example, the query

```
SELECT
    payment_type,
    count(distinct unique_key) as num_trips
FROM
    TABLE
GROUP BY
    payment_type
ORDER BY
    payment_type
```

would return

payment_type	num_trips
Cash	3
Credit Card	3
Mobile	2

Using Multiple Aggregation Functions in Single Query

- You can use multiple aggregate functions within a single query.
- When grouping by a single column, all the aggregate functions get applied across the same groups for the column.
- Run this query in Big Query and make sense of the results:

```
SELECT
  payment_type,
  COUNT(DISTINCT unique_key) AS num_trips,
  SUM(trip_total) AS sum_trip_total,
  AVG(trip_total) AS avg_trip_total,
  MAX(trip_total) AS max_trip_total,
  MIN(trip_total) AS min_trip_total
FROM
  `bigquery-public-data.chicago_taxi_trips.taxi_trips`
GROUP BY
  payment_type
ORDER BY
  payment_type
```

Group By and Where Clause Together

- The WHERE clause comes before the GROUP BY
- The WHERE clause gets applied first to the table to filter it, then the GROUP BY and Aggregates get applied.
- you can also order by one of the aggregated results. Here we order by *num_trips*

```
SELECT
  payment_type,
  COUNT(DISTINCT unique_key) AS num_trips,
  SUM(trip_total) AS sum_trip_total,
  AVG(trip_total) AS avg_trip_total,
  MAX(trip_total) AS max_trip_total,
  MIN(trip_total) AS min_trip_total
FROM
  `bigquery-public-data.chicago_taxi_trips.taxi_trips`
```

```
WHERE
  payment_type IN ('Cash', 'Credit Card')
GROUP BY
  payment_type
ORDER BY
  num_trips DESC
```

Applying a Filter on Top of the Aggregated Results Using HAVING

- the **HAVING** statement can only be used when performing a **GROUP BY**.
- It is essentially like performing a WHERE CLAUSE after the Aggregation is applied to the different groups.
- For example, suppose we wanted to calculate the number of trips for each *payment_type*. However, when returning the aggregated results we only want to return records for which the total number of trips was greater than 300,000. Then we could use a **HAVING** statement like this:

```
SELECT
  payment_type,
  COUNT(DISTINCT unique_key) AS num_trips
FROM
  `bigquery-public-data.chicago_taxi_trips.taxi_trips`
GROUP BY
  payment_type
HAVING
  num_trips > 300000
ORDER BY
  num_trips DESC
```

Run the above query with and without the **HAVING** statement so you can understand completely what is going on by comparing the results of the two queries.