# Google BigQuery Lab



## 1 BigQuery background

General overview of BigQuery
https://cloud.google.com/bigquery/docs/introduction

Overview of BigQuery Data Storage
https://cloud.google.com/bigquery/docs/storage_overview

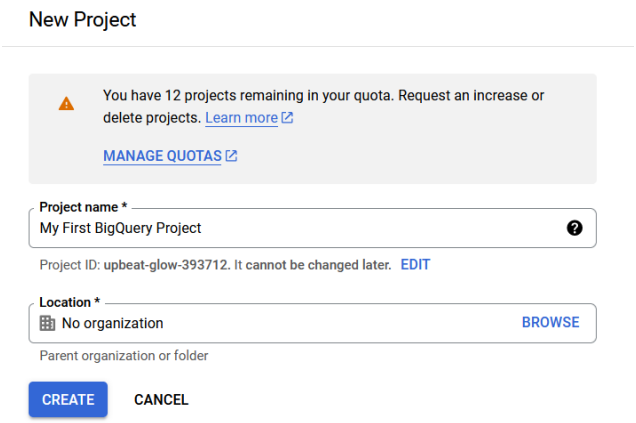
Overview of BigQuery analytics
https://cloud.google.com/bigquery/docs/query-overview

## 2 Lab setup

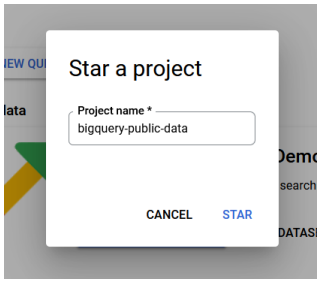
You can use the BigQuery sandbox to explore limited BigQuery capabilities without providing a credit card or creating a billing account for your project. If you already created a billing account, you can still use BigQuery at no cost in the free usage tier.

https://cloud.google.com/bigquery/docs/sandbox?hl=en

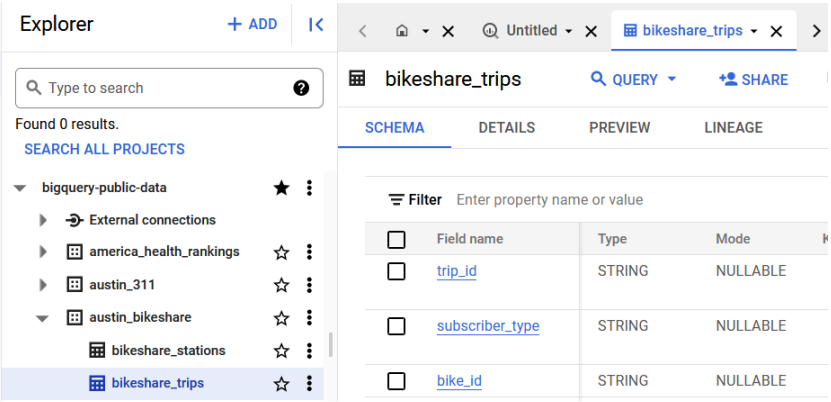
Create a new project with the name: `My First BigQuery Project`




Once done, you can set up access to one of the datasets in the public project `bigquery-public-data`.
https://cloud.google.com/bigquery/docs/quickstarts/query-public-dataset-console#open_a_public_dataset




Opening up any one of the datasets allows us to select a table which opens up a tab in the details pane, which we can then explore in more detail by selecting the appropriate tabs (for e.g. Schema, Details, Preview and Lineage for a normal table).



Notice that for some of these tables, the number of rows are in the millions and tens of millions. This exceeds the capacity of small scale data analytics tools such as Excel:
https://support.microsoft.com/en-gb/office/excel-specifications-and-limits-1672b34d-7043-467e-8e27-269d656771c3

Even popular relational databases such as MySQL have hard limits on their table column count and row size
https://dev.mysql.com/doc/refman/8.0/en/column-count-limit.html

The number of databases / tables are not limited by the system, but by the underlying file storage system and there are limits with modern SAN storage systems:
https://dev.mysql.com/doc/refman/8.0/en/database-count-limit.html

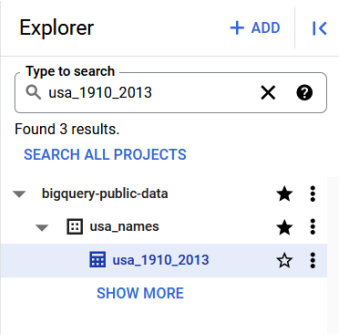
We will demonstrate some basic analytics that we can perform on a table using query statements (or Data Query Language (DQL) statements) written in GoogleSQL. The statements operate by scanning one or more specified tables and returns the computed result rows.

https://cloud.google.com/bigquery/docs/introduction-sql

# 3 Using the Count function

Search for this table, select and start it and view it in the details pane and explore it in more detail

```
usa_names.usa_1910_2013
```



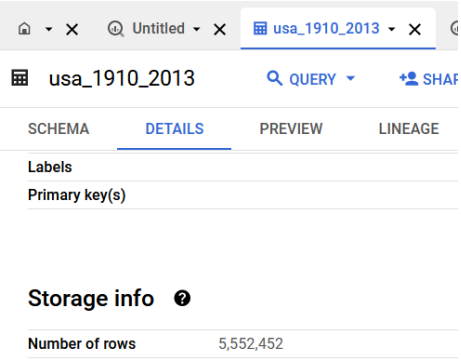

The most basic query (which you can type in a new query tab in the details pane) is to see the first 1000 (or whatever number you desire) number of rows / records in the table.

```
SELECT * FROM `bigquery-public-data.usa_names.usa_1910_2013` LIMIT 1000
```

You can count the total number of row / records with:

```
SELECT
  COUNT(*) AS num_records
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
```

Notice that this count tallies with the info about the table:




To get the count of the number of rows with the column name (which will be the same as the total number of rows in the table, since all rows have this column):

```
SELECT
  COUNT(name) AS cnt
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
```

To see how many distinct (unique) years there are:

```
SELECT
  COUNT(DISTINCT year) AS distinct_year_count
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
```

The result reflects the 104 distinct and different years between 1910 and 2013

To see how many distinct (unique) states there are:

```
SELECT
  COUNT(DISTINCT state) AS distinct_state_count
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
```

There are 51 states (50 states + Washington DC)

To see how many distinct (unique) genders there are:

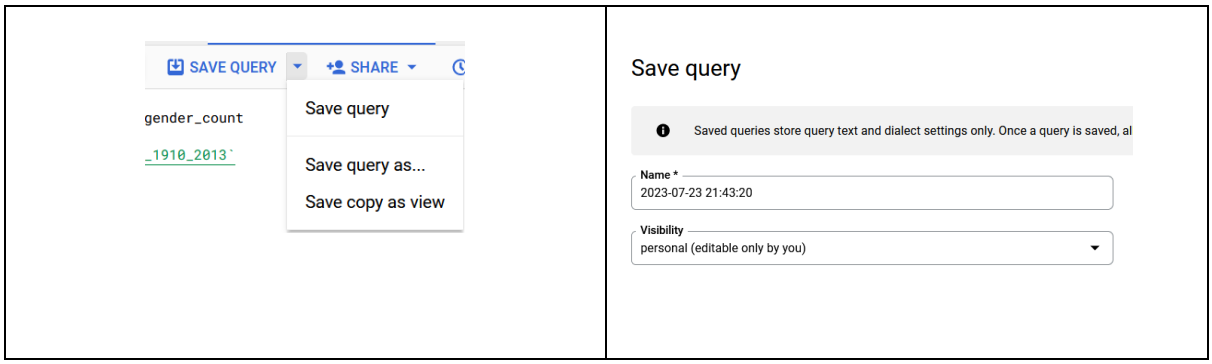
```
SELECT
  COUNT(DISTINCT gender) AS distinct_gender_count
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
```

We can combine all these separate queries into a single larger query if we wish:

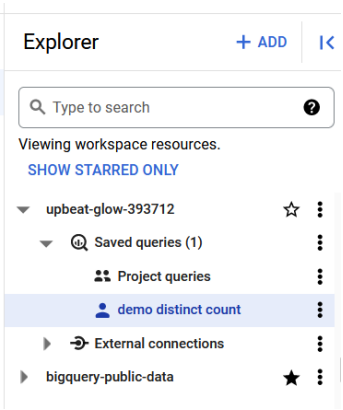
```
SELECT
  COUNT(DISTINCT gender) AS distinct_gender_count,
  COUNT(DISTINCT year) AS distinct_year_count,
```

```
  COUNT(DISTINCT state) AS distinct_state_count,
  COUNT(DISTINCT name) AS distinct_name_count,
  COUNT(*) AS num_records,
  COUNT(name) AS cnt
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
```

You can save any of these queries if you wish for future reference / reuse in the same project or to share with others:




The saved query should be visible in the Explorer Pane:




# 4 Using Count with Null records

Often, large datasets with have null values in some of their columns due to the various issues encountered when gathering the data.

Look for `new_york_mv_collisions.nypd_mv_collisions` in `bigquery-public-data` and star it in the Explorer Pane.

You can verify the number of rows with:

```
SELECT
  COUNT(*) AS num_rows
```

```
FROM                                                `bigquery-public-
data.new_york_mv_collisions.nypd_mv_collisions`
```

However, if you check again with:

```
SELECT
  COUNT(contributing_factor_vehicle_1)                          AS
contributing_factor_vehicle_1_count
FROM                                                `bigquery-public-
data.new_york_mv_collisions.nypd_mv_collisions`
```

You will notice the count is slightly less than the total number of rows, indicating the presence of NULL in some of the rows of this column.
This is even more evident in the next query:

```
SELECT
  COUNT(contributing_factor_vehicle_2)                          AS
contributing_factor_vehicle_2_count
FROM                                                `bigquery-public-
data.new_york_mv_collisions.nypd_mv_collisions`
```

This can also be seen in the preview tab of the details pane for this table.

We can also check the number of null values in the other columns with:

```
SELECT
  COUNT(borough) AS borough_count,
  COUNT(contributing_factor_vehicle_3)                          AS
contributing_factor_vehicle_3_count,
FROM
  `bigquery-public-data.new_york_mv_collisions.nypd_mv_collisions`
```

# 5 Retrieving a distinct set of values

You can retrieve a distinct set of values from a column and order the results as well using the ORDER BY clause

For example, if we wanted to return the distinct set of years, starting with the most recent, from the USA names data set we would run this query:

```
SELECT
  DISTINCT year
FROM
  `bigquery-public-data.usa_names.usa_1910_current`
ORDER BY
  year DESC
```

Or if we wanted to return a list of the first 50 names sorted in alphabetical order we could execute this query:

```
SELECT
```

```
  DISTINCT name
FROM
  `bigquery-public-data.usa_names.usa_1910_current`
ORDER BY
  name ASC
LIMIT
  50
```

If we wanted to see the last 100 names we could execute:

```
SELECT
  DISTINCT name
FROM
  `bigquery-public-data.usa_names.usa_1910_current`
ORDER BY
  name DESC
LIMIT
  100
```

You can select from the distinct values from a different number of columns at the same time and order the results based on one or more of these columns simultaneously:

```
SELECT
  DISTINCT
  borough,
  major_category,
  minor_category
FROM
  `bigquery-public-data.london_crime.crime_by_lsoa`
ORDER BY
  borough,
  major_category,
  minor_category
```

# 6 Basic use of WHERE

The WHERE clause allows you to specify a search condition for the rows returned by a query. The `search_condition` is a combination of one or more expressions using the logical operator AND, OR and NOT.

We can select conditions involving single columns:

```
SELECT  *  FROM     `bigquery-public-data.usa_names.usa_1910_current`
WHERE   state = 'FL'

SELECT  *  FROM     `bigquery-public-data.usa_names.usa_1910_current`
WHERE   gender = 'M'
```

Or we can have conditions involving the combination of multiple columns

```
SELECT    * FROM    `bigquery-public-data.usa_names.usa_1910_current`
WHERE    state = 'FL'    AND gender = 'M'    AND year = 2000 ORDER BY
number DESC LIMIT   100
```

# 7 Using WHERE to filter on numerical columns with operators

We work with the table: `new_york_citibike.citibike_trips`

We can also filter on a numerical column using various combination of operators:

```
SELECT
  count(*) as num_bike_rides
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
    tripduration = 432
```

```
SELECT
  COUNT(*) AS num_bike_rides
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
  tripduration != 432
```

```
SELECT
  COUNT(*) AS num_bike_rides
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
  tripduration < 300
```

```
SELECT
  COUNT(*) AS num_bike_rides
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
  tripduration <= 300
```

```
SELECT
  COUNT(*) AS num_bike_rides
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
```

```
  tripduration > 24*60*60


SELECT
  COUNT(*) AS num_bike_rides
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
  tripduration >= 24*60*60
```

We can also combine multiple conditions using AND or OR operators.

```
SELECT
  COUNT(*) AS num_bike_rides
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
  tripduration >= 5*60*60 AND tripduration <= 9*60*60
```

Notice the use of parenthesis around the two statements separated by OR. You need these brackets when chaining together AND and OR conditions within the same clause.

```
SELECT
  COUNT(*) AS num_bike_rides
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
  (tripduration >= 1*60*60 AND tripduration <= 3*60*60)
  OR
  (tripduration >= 5*60*60 AND tripduration <= 8*60*60)
```

The IN statement can be used to provide a list within a condition. In this example we are counting the records where the tripduration is in the list (60,120).

```
SELECT
  COUNT(*) AS num_bike_rides
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
    tripduration IN (60,120)
```

The above query is the same as saying the tripduration is 60 or the trip duration is 120

Another similar example:

```
SELECT
  COUNT(*) AS num_bike_rides
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
```

```
WHERE
    tripduration = 60 or tripduration = 120 or tripduration = 180 or
tripduration = 240
```

can be replaced with a query using an IN statement that is easier to write/read.

```
SELECT
  COUNT(*) AS num_bike_rides
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
    tripduration IN (60,120,180,240)
```

You can also perform a negation of the above query, for e.g. count the number of records where the trip duration was not one of (60,120,180,240).

```
SELECT
  COUNT(*) AS num_bike_rides
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
WHERE
    tripduration NOT IN (60,120,180,240)
```

# 8 Using WHERE to filter on string columns with operators

We will work with the table: `london_crime.crime_by_lsoa`

We can also use the WHERE clause on the contents of text (string) columns for example:

```
SELECT
  COUNT(*) AS num_crimes
FROM
  `bigquery-public-data.london_crime.crime_by_lsoa`
WHERE
  minor_category != "Harassment"
```

```
SELECT
 COUNT(*) AS num_crimes
FROM
 `bigquery-public-data.london_crime.crime_by_lsoa`
WHERE
 minor_category in ("Harassment", "Assault with Injury")
```

```
SELECT
  COUNT(*) AS num_crimes
FROM
  `bigquery-public-data.london_crime.crime_by_lsoa`
```

```
WHERE
  minor_category = 'Harassment' or minor_category = 'Assault with
Injury'
```

A common technique is to search for a specific pattern within a string column. - You can look for a pattern anywhere in the string by using like '`%pattern%`' - You can look for a pattern at the end of the string by using like '`%pattern`' - You can look for a pattern at the start of the string by using like '`pattern%`' - The pattern given is any pattern of characters and it's case sensitive.

```
SELECT
  distinct minor_category
FROM
  `bigquery-public-data.london_crime.crime_by_lsoa`
WHERE
  minor_category like 'Drug%'
```

```
SELECT
  distinct minor_category
FROM
  `bigquery-public-data.london_crime.crime_by_lsoa`
WHERE
  minor_category like '%Drugs'
```

```
SELECT
  distinct minor_category
FROM
  `bigquery-public-data.london_crime.crime_by_lsoa`
WHERE
  minor_category like '%eh%'
```

Often when looking for a pattern, we lower case the column in which we are looking. This way we can just use lower case in our like statement. Google SQL provides many more of these functions that we can support complex query functionality or simplify existing ones.

```
SELECT
  distinct minor_category
FROM
  `bigquery-public-data.london_crime.crime_by_lsoa`
WHERE
  lower(minor_category) like '%motor%'
```

# 9 Using WHERE on TIMESTAMP columns

We will work with the table: `austin_bikeshare.bikeshare_trips`

A TIMESTAMP column will typically have the year, month, day, hour, minute and second. For example 2014-10-26 15:12:00 UTC. A DATE column will just have the year, month and day. For example, 2014-10-26. You can change a TIMESTAMP into a DATE by casting it i.e. `cast(TIMESTAMP AS DATE)`. This would convert 2014-10-26 15:12:00 UTC into 2014-10-26.

```
SELECT
  start_time as start_time_timestamp
FROM
  `bigquery-public-data.austin_bikeshare.bikeshare_trips`
LIMIT
  100
```

```
SELECT
  cast(start_time as date) as start_time_date
FROM
  `bigquery-public-data.austin_bikeshare.bikeshare_trips`
LIMIT
  100
```

```
SELECT
  cast(start_time as date) as start_time_date,
  extract(hour from start_time) as start_time_hour,
  extract(minute from start_time) as start_time_minute
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
LIMIT
  100
```

```
SELECT
  cast(start_time as date) as start_time_date,
  extract(day from start_time) as start_time_day,
  extract(year from start_time) as start_time_year,
  extract(month from start_time) as start_time_month,
  extract(week from start_time) as start_time_week
FROM
  `bigquery-public-data.austin_bikeshare.bikeshare_trips`
LIMIT
  100
```

You can also filter for records after a given date

```
SELECT
  cast(start_time as date) as start_time_date,
  extract(day from start_time) as start_time_day,
  extract(year from start_time) as start_time_year,
  extract(month from start_time) as start_time_month,
  extract(week from start_time) as start_time_week
FROM
  `bigquery-public-data.austin_bikeshare.bikeshare_trips`
```

```
WHERE  start_time > '2018-10-01'
LIMIT
  100
```

You can obtain records for a specific date

```
SELECT
  cast(start_time as date) as start_time_date,
  extract(day from start_time) as start_time_day,
  extract(year from start_time) as start_time_year,
  extract(month from start_time) as start_time_month,
  extract(week from start_time) as start_time_week
FROM
  `bigquery-public-data.austin_bikeshare.bikeshare_trips`
WHERE  cast(start_time as date) = '2018-10-01'
LIMIT
  100
```

You can obtain records between two dates:

```
SELECT
  cast(start_time as date) as start_time_date,
  extract(day from start_time) as start_time_day,
  extract(year from start_time) as start_time_year,
  extract(month from start_time) as start_time_month,
  extract(week from start_time) as start_time_week
FROM
  `bigquery-public-data.austin_bikeshare.bikeshare_trips`
WHERE  start_time >= '2018-09-01' and start_time <= '2018-09-30'
LIMIT
  100
```

Finally, you can also filter records in a given list of hours:

```
SELECT
  cast(start_time as date) as start_time_date,
  extract(hour from start_time) as start_time_hour,
  extract(minute from start_time) as start_time_minute,
FROM
  `bigquery-public-data.austin_bikeshare.bikeshare_trips`
where extract(hour from start_time) IN (17,18,19,20)
LIMIT
  100
```

## 10 Using WHERE To filter on Null / Not Null

We will work with the table: `new_york_mv_collisions.nypd_mv_collisions`

You can check the total number of records in this table from the Details pane, or with:

```
SELECT
  COUNT(*) AS total_records
FROM                                                `bigquery-public-
data.new_york_mv_collisions.nypd_mv_collisions`
```

To count the total number of records with non-null values in any particular column, we can type:

```
SELECT
  COUNT(*)
  FROM                                              `bigquery-public-
data.new_york_mv_collisions.nypd_mv_collisions`
WHERE
contributing_factor_vehicle_1 IS NOT NULL
```

Similarly, to count the total number of records with null values in any particular column, we can type:

```
SELECT
COUNT(*)
FROM                                                `bigquery-public-
data.new_york_mv_collisions.nypd_mv_collisions`
WHERE
contributing_factor_vehicle_1 IS NULL
```

```
SELECT
COUNT(*)
FROM                                                `bigquery-public-
data.new_york_mv_collisions.nypd_mv_collisions`
WHERE
contributing_factor_vehicle_2 IS NULL
```

## 11 Introducing GROUP BY

We will work with the table: `usa_names.usa_1910_2013`

The GROUP BY clause groups a set of rows into a set of summary rows by values of columns or expressions. The GROUP BY clause returns one row for each group. In other words, it reduces the number of rows in the result set.

To get the total count of males and females:

```
SELECT
      gender,
      COUNT(gender) AS gender_count
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
GROUP BY
  gender
```

To get the total count of people with a distinct name

```
SELECT
      name,
      COUNT(name) AS name_count
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
GROUP BY
  name
```

To get the total count of people from a distinct state

```
SELECT
      state,
      COUNT(state) AS state_count
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
GROUP BY
  state
```

To filter the results returned from the count, we use the HAVING clause

```
SELECT
      state,
      COUNT(state) AS state_count
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
GROUP BY
  state
HAVING
 state_count > 100000
```

To filter and sort the results returned from the count we can use the ORDER BY clause

```
SELECT
      state,
      COUNT(state) AS state_count
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
GROUP BY
  state
HAVING
 state_count > 100000
ORDER BY
  state_count DESC
```

To filter before grouping, we can use the WHERE clause which comes before the GROUP BY. This gets applied first to the table to filter it: (for e.g. finding the count of all females in all the states)

```
SELECT
      state,
```

```
      COUNT(state) AS state_count
FROM
  `bigquery-public-data.usa_names.usa_1910_2013`
WHERE
  gender = 'F'
GROUP BY
  state
ORDER BY
  state_count DESC
```

## 12 Using GROUP BY with aggregate functions

We will work with the table: `chicago_taxi_trips.taxi_trips`

Typically, the MAX, MIN, AVG, SUM, and COUNT functions are applied along with the GROUP BY statement to perform aggregation over the grouped records.

These aggregate functions can be applied individually:

```
SELECT
  payment_type,
  MIN(trip_total) AS min_trip_total
FROM
  `bigquery-public-data.chicago_taxi_trips.taxi_trips`
GROUP BY
  payment_type
ORDER BY
  payment_type
```

```
SELECT
  payment_type,
  MAX(trip_total) AS min_trip_total
FROM
  `bigquery-public-data.chicago_taxi_trips.taxi_trips`
GROUP BY
  payment_type
ORDER BY
  payment_type
```

They can also be combined into a single query:

```
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
```

You can also add in the WHERE clause to filter before performing the GROUP by and applying the aggregation function:

```
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', 'Mobile')
GROUP BY
  payment_type
ORDER BY
  num_trips DESC
```

# 13 Performing JOINS

Joining tables means to JOIN columns from one table onto another table given some join condition/criteria. This is typically done when there is a column or set of columns in common between tables.

Tables to use:
`census_bureau_international.midyear_population`
`census_bureau_international.country_names_area`

Query to use:

```
SELECT
  m.year,
  m.country_name AS country,
  m.midyear_population AS population,
  a.country_area AS area
FROM
  `bigquery-public-
data.census_bureau_international.midyear_population` m
LEFT JOIN
  `bigquery-public-
data.census_bureau_international.country_names_area` a
ON
```

```
  m.country_code = a.country_code
ORDER BY
  year,
  country
```

# 14 Looker Studio background

General Overview of Looker Studio and its use in BI
https://cloud.google.com/looker-studio

You can browse through and experiment with some of the templates available
https://lookerstudio.google.com/navigation/templates