Google publicly launched BigQuery, to outside users in 2012. Since then, BigQuery has expanded to become not just a query engine but a hosted, managed cloud-based structured storage provider, it is based on Dremel. Dremel run extremely fast SQL queries on large datasets.

The few aspects of BigQuery are:

SQL Queries over Big Data

The primary function of BigQuery is to enable interactive analytic queries over Big Data. It is scale-invariant, that is, whether you have a hundred rows in your table or a hundred billion, the mechanism to work with them should be the same. Of course running a megabyte and a terabyte will have a variant time delay but won't hit a brick wall when scale up.

BigQuery SQL

SQL is the query language, other systems, enable you to write code in your favorite language to perform analytics, but these languages make it difficult to interactively ask questions of your data. For instance if you use Java program to query your data, you'll end up spending a lot of time compiling, debugging, and uploading your program, rather than figuring out what data you need. SQL is a declarative language; that is, you declare what results you want, and it is up to the software to figure out how to get those results. It is a very common language you don't need to be a programmer to use it.

The Speed:

The Dremel query engine created a way to parallelize SQL execution across thousands if machines. If you double the size, it will take less than double the time to process the query, let's say if you have a 100 MB table that takes 3 seconds to query and you increase the size a thousand times to 100 GB, it might take only 5 seconds to query. It is a scale-out solution if you need your query to run faster you can throw more machine at the problem.

Cloud Storage System

BigQuery is also a place to store your structured data in the cloud. If your data didn't live in Google's cloud, then you couldn't query it. Your data is replicated to multiple geographically distinct locations for improved availability and durability. Data is also replicated within a cluster, so your data should be virtually immune to data loss due to hardware failure. Of course, the BigQuery service may not have perfect uptime, and if your data is important, you should make sure it is backed up.

Data Ingestion

There are three ways to get your data into BigQuery: streaming, direct upload, and through Google Cloud Storage. If your data is already in Google Cloud Storage, the load step is merely a transfer between two systems already within Google's cloud, so ingestion is very fast.

Structured Data Storage

BigQuery is a system that stores and operates on structured data; that is, data that follows a rigid *schema*. Collections of rows of data following a single schema are organized into *tables*. These tables are similar to tables in a typical relational database but have some restrictions. The only way to modify BigQuery tables is to append to them or rewrite them, there is no way to update individual rows. BigQuery also doesn't support table modification queries, like ALTER TABLE, DROP TABLE, or UPDATE TABLE.

We could also consider other aspects like:

Distributed Cloud Computing

Cloud Data Warehousing
offer fault-tolerance, geographic distribution, and automated backups.

Multitenancy and Parallel Execution

Analytics as a Service (AaaS?)
BigQuery is a service that you use to perform your analytics tasks. It operates at a higher level than most other Big Data analytics offerings. For example, tools such as Impala and Presto require you to manage your own virtual hardware and your own data. Even Amazon Redshift, although it is hosted, requires you to manage a database instance.

Global Data Namespace
One advantage to performing your analytics in the cloud is that it becomes easy to share data without moving it around. All BigQuery tables sit in the same namespace. This may seem like a minor detail, but it is actually extremely useful. Every table in BigQuery can be joined against every other table in BigQuery, as long as the user running the query has access to both tables. This means that if someone publishes a table with weather data, you can join that weather table against your sales data to determine how the weather affects your sales.

Web UI

You can do everything on your BigQuery from there, browse available tables, read their schema and data, share datasets with other users, load data, and export it to Google Cloud Storage. It also allows you to create and edit queries. There is no need to download any client-side software or install anything.

HTTP API

You can send request the same time of HTTP requests, there are client-side tools that make it easier to interact with the service, but if you're happier using the raw HTTP operations, that is an available option.

Asynchronous Job Execution

It is asynchronous, this is one of the most import piece of BigQuery has. In a synchrobnous model, where you start the query and wait for the response, which contains the query results. When running queries synchronously, if you hit a network error or the request times out, you have to retry the query from scratch. In asynchronous however, you can start the query and the poll until it is done.

BigQuery is not relational database, nor NoSQL, Not Even MapReduce, not open source, so how BigQurey relates to the Google infrastructure stack?

BigQuery table data is stored in Colossus. So far, Google has been secretive about the details of Colossus, other than to say that it is a successor to the Google File System (GFS). At a high level, Colossus is a distributed filesystem that stores data on an enormous number of disks and makes the data available over a network, which means that the storage is not physically attached to the machines requesting the data, and that data is distributed across the network. A distributed SQL query engine that can perform complex queries over data stored on Colossus, GFS, or elsewhere, called Dremel. Just to compare Dremel with others implementation, we could select Cloudera's Impala, allows to query data inside HDFS and Hive without extracting it. Amazon's Redshift is a fork of PostgreSQL which has been modified to scale out across multiple machines, Facebook's Presto that is like Impala and the Apache incubator project "Drill". Drill fills in the gaps of the Dremel paper to provide a similar open source version.

Although BigQuery runs the same types of SQL queries that you can run on a relational database, it executes them in a different way, for instance as BigQuery uses parallel architecture, there is no need of use index as in relational databases.

What BigQuery are:
Enterprise Data Warehouse
Sql queries – with Google storage underneath, supports standard and legacy SQL.
Fully-managed – no server, no resources deploys, no need to instantiate an instance, but has to leave in a project.
Access through: Web UI, REST API, clients
Third party tools

Schema Auto-Detection:

Available while

loading data

querying external data

BigQuery selects a random file in the data source and scans up to 100 rows of data to use as representative sample. Then examines each field and attempts to assign a data type to that field based on the values in the sample.

Querying and Viewing:

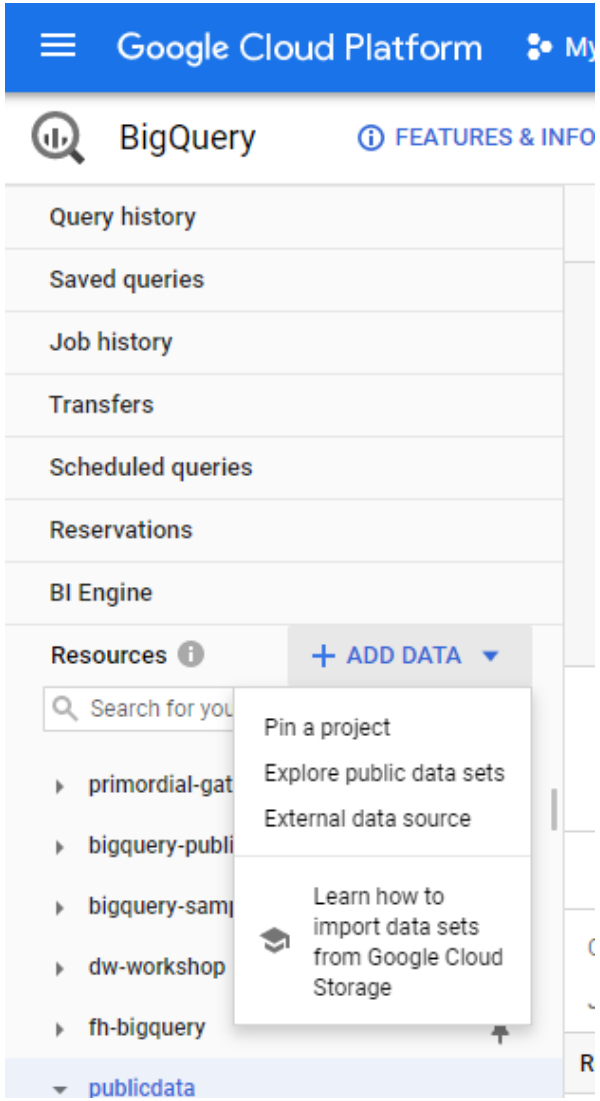
Interactive queries => Default mode (executed asap).

Batch queries => will schedule these to run whenever possible (idle resource).

Views => are logical – not materialized, execute each time view is accessed, can't export data from a view, can't use JSON API to retrieve data, can't mix standard and legacy SQL, no user-defined functions allowed, limit of 1000 views per dataset.

Partition tables => special table where data is portioned for you, no need to create partitions manually or programmatically. Need to declare tables as partitioned at creation time, no need specify schema, BigQuery automatically creates date partitions.

Add public datasets



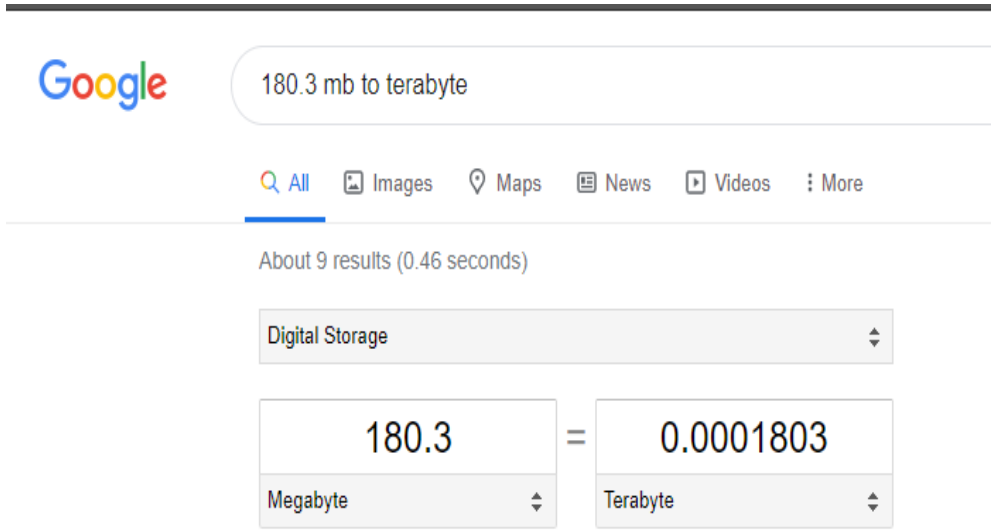
Pricing and pin public datasets:

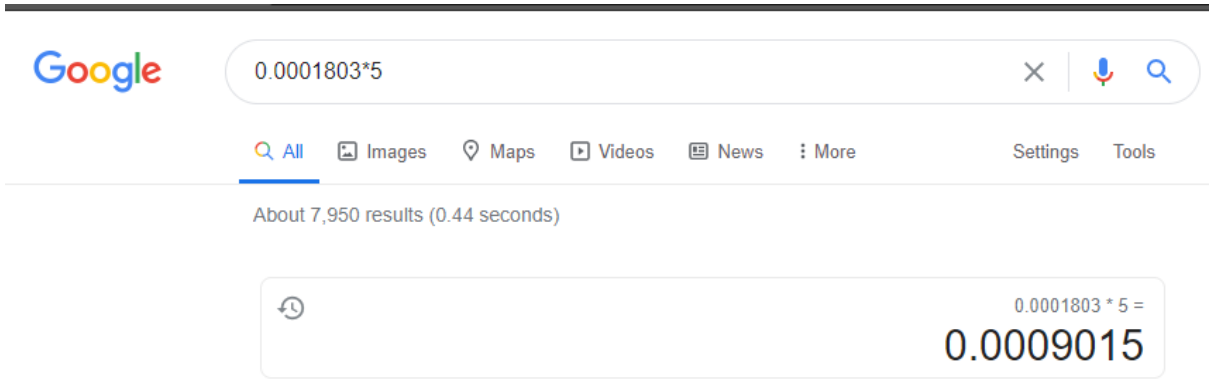
SELECT * FROM `bigquery-public-data.baseball.games_post_wide` LIMIT 1000

CHARGE

$5 for Terabyte

This query will process 180.3 MB when run





When datasets is very large, for instance:

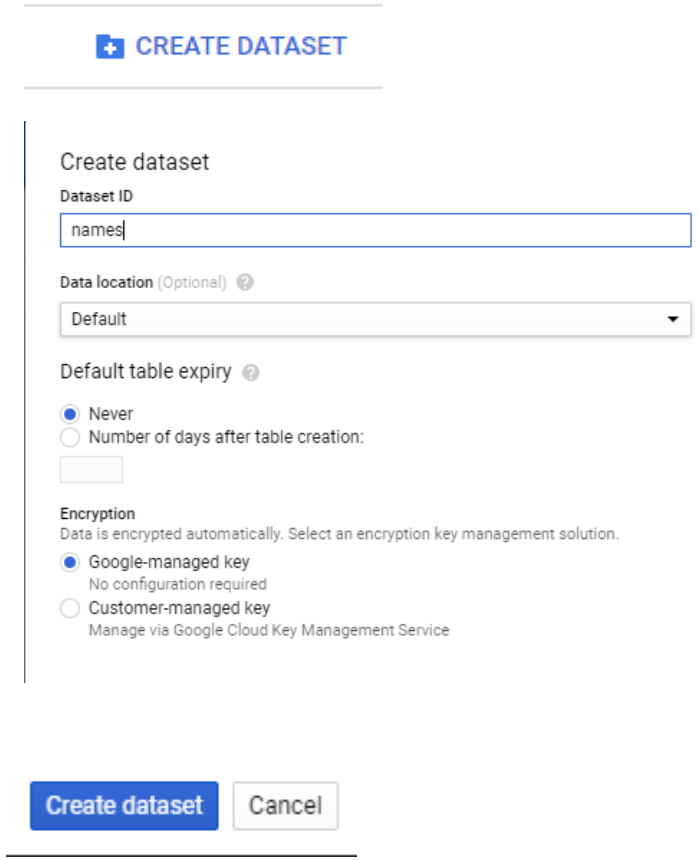
```
SELECT *
FROM `bigquery-public-data.stackoverflow.stackoverflow_posts`
```

Make sure query only columns you need.

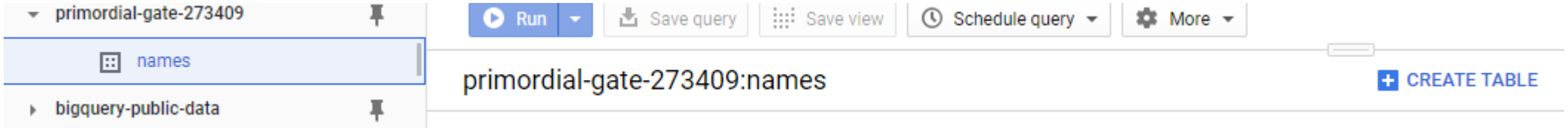
Loading data:

You can load csv, json and some other files to create your tables in BigQuery, this file does not need to be on GCS. Let's give some example:

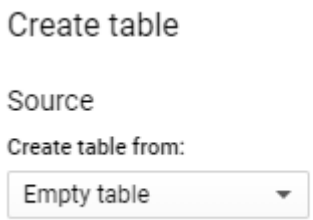
Start creating a new dataset and give any name you like it, for this example I will use names.



Click on dataset you've create and than create table

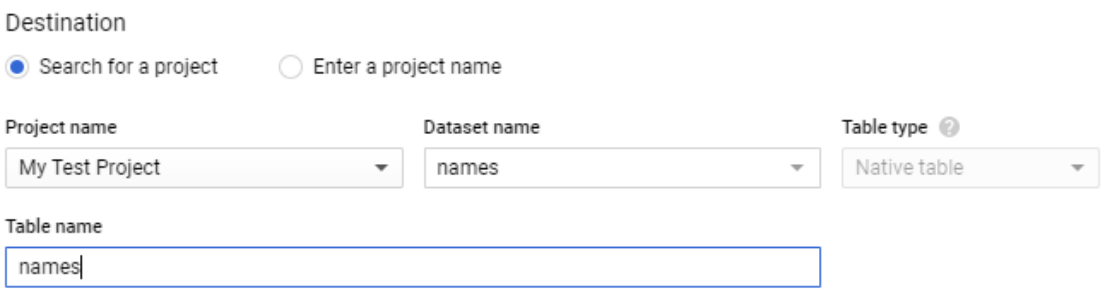


Click on create table, than Create table from choose upload



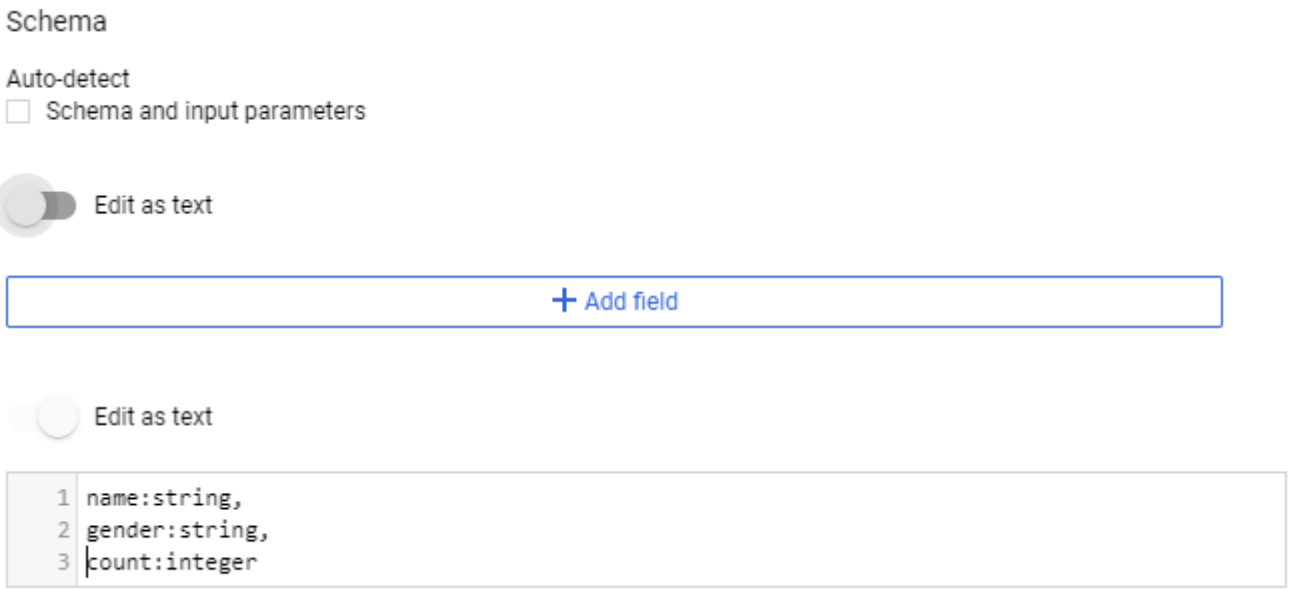
After choose upload you see this form

The default File format is avro, change for csv and click Browse, than select the file you like to upload, than give a table name

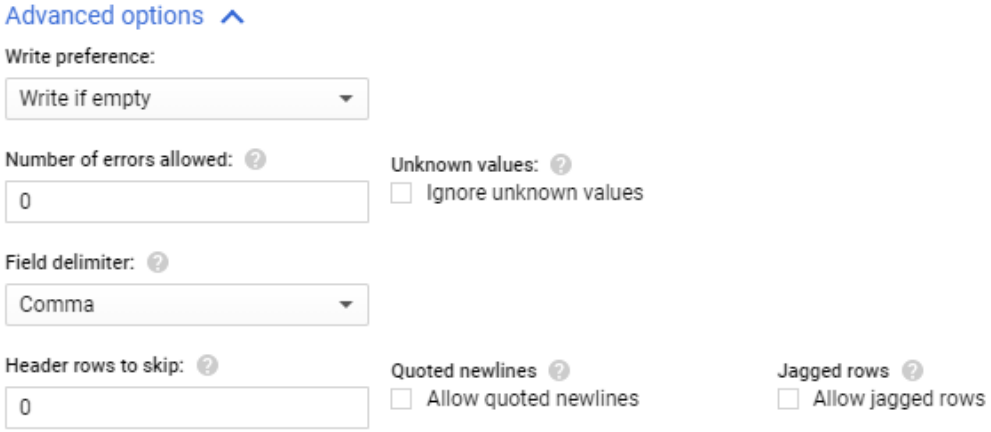


Next define the schema, you can edit each column, copy a json file our just type the columns name:type separate by comma.

Click on Edit as text, and edit your columns



Under advanced option you can specify the field delimiter, how many rows to skip in case you have header, etc



Press create table and you will see the table created under your dataset

You can now start query your recent create table.

So what are the 10 most given names for baby girl in USA in 2019?

```
SELECT  *
FROM `names.names`
WHERE gender = 'F'
ORDER BY count DESC
LIMIT 10
```

Row	name	gender	count
1	Olivia	F	18451
2	Emma	F	17102
3	Ava	F	14440
4	Sophia	F	13714
5	Isabella	F	13306
6	Charlotte	F	13138
7	Amelia	F	12862
8	Mia	F	12414
9	Harper	F	10442
10	Evelyn	F	10392

Let’s talk about one more thing before starting the action. Sometime we need to get help and one easy way to get that is by the console, you can open the console, and you can use some useful command there, for BigQuery all command starts with bq

```
gilson_bellon@cloudshell:~ (primordial-gate-273409)$ bq ls
  datasetId
 -----------
  names
gilson_bellon@cloudshell:~ (primordial-gate-273409)$
```

bq ls – list all dataset on the current project

Some useful commands are:

bq help
bq help show

If you want to use the interactive command shell: bq shell

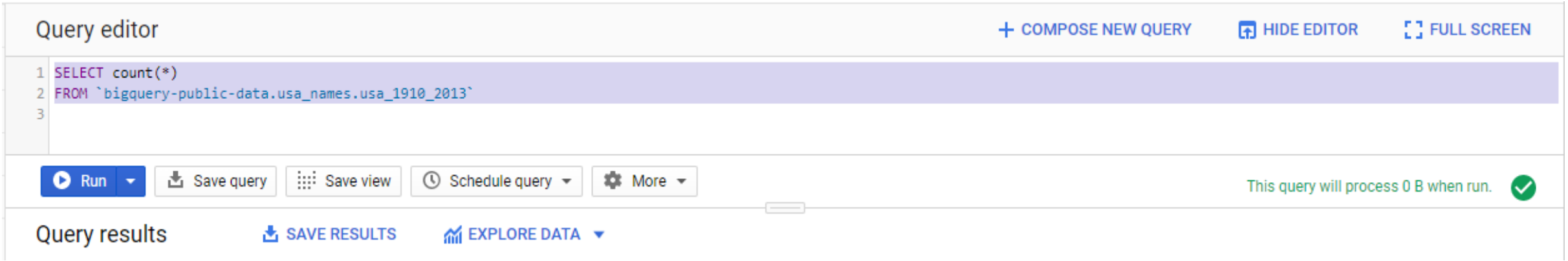
Let’s starting query

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

Now what if we want to count the number of rows on that table?

```
SELECT count(*)
FROM `bigquery-public-data.usa_names.usa_1910_2013`
```

This information is always get from the table details and is a very simple query, I just want to draw your attention, when you run this query the amount to be process is 0 B, that mean there is no cost for that query.



What if we want to counter genders?

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

Using the DISTINCT

```
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`
```

Row	distinct_gender_count	distinct_year_count	distinct_state_count	distinct_name_count	num_records	cnt
1	2	104	51	29828	5552452	5552452

The WHERE word

```
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
```

```
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
```

How many trip exceed a day?

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

How may trip is between 5 and 9 hours?

```
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
```

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

```
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
```

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

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

```
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 lower(minor_category) like '%motor%'
```

Some useful TIMESTAMP functions

```
SELECT
  start_time as start_time_timestamp,
  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,
  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
```

Row	start_time_timestamp	start_time_date	start_time_hour	start_time_minute	start_time_day	start_time_year	start_time_month	start_time_week
1	2015-10-02 21:12:01 UTC	2015-10-02	21	12	2	2015	10	39
2	2014-10-26 15:12:00 UTC	2014-10-26	15	12	26	2014	10	43
3	2014-10-26 15:12:00 UTC	2014-10-26	15	12	26	2014	10	43
4	2014-10-26 15:12:00 UTC	2014-10-26	15	12	26	2014	10	43
5	2014-10-26 18:12:00 UTC	2014-10-26	18	12	26	2014	10	43
6	2014-10-26 18:12:00 UTC	2014-10-26	18	12	26	2014	10	43
7	2014-10-26 18:12:00 UTC	2014-10-26	18	12	26	2014	10	43

Filtering For Records After A Date

```
SELECT
  start_time as start_time_timestamp,
  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,
  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
```

Filtering For Records Equal To A Date

```
SELECT
  start_time as start_time_timestamp,
  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,
  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
```

Filtering For Records Between Two Dates

```
SELECT
  start_time as start_time_timestamp,
  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,
  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
```

Filtering For Records In Given List Of Hours

```
SELECT
  start_time as start_time_timestamp,
  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,
  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 extract(hour from start_time) IN (17,18,19,20)
LIMIT
  100
```

NULL / NOT NULL

```
SELECT
  COUNT(*) AS count1, -- the number of records
  COUNT(dropoff_location) AS count2 -- the number of records where dropoff_location is not null
FROM `bigquery-public-data.chicago_taxi_trips.taxi_trips`
WHERE dropoff_location IS NULL
```

```
SELECT
  COUNT(*) AS count1, -- the number of records
  COUNT(dropoff_location) AS count2 -- the number of records where dropoff_location is not null
FROM `bigquery-public-data.chicago_taxi_trips.taxi_trips`
WHERE dropoff_location IS NOT NULL
```

GROUP BY AND AGGREGATE FUNCTIONS

```
SELECT
  name,
  SUM(number) AS num_people
FROM `bigquery-public-data.usa_names.usa_1910_2013`
WHERE gender = 'F'
GROUP BY name
HAVING num_people > 500000
ORDER BY num_people DESC
LIMIT
  100
```

```
SELECT
  SUM(number) AS total,
  name,
  gender
FROM `bigquery-public-data.usa_names.usa_1910_current`
GROUP BY
   name,
  gender
ORDER BY total DESC
LIMIT 10;
```

```
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
```

```
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
```

```
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
```

JOIN

```
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
```

Some str functions

```
SELECT
  species_common_name,
  form,
  fall_color,
  tree_id
FROM `bigquery-public-data.new_york_trees.tree_species` ts
LEFT JOIN `bigquery-public-data.new_york_trees.tree_census_2015` tc
ON trim(lower(ts.species_common_name)) = trim(lower(tc.spc_common))
LIMIT 1000
```

```
SELECT
  species_common_name,
  form,
  fall_color,
  COUNT(DISTINCT tree_id) AS num_trees,
  AVG( tree_dbh) AS avg_tree_diam,
  MAX(tree_dbh) AS max_tree_diam
FROM `bigquery-public-data.new_york_trees.tree_species` ts
JOIN `bigquery-public-data.new_york_trees.tree_census_2015` tc
ON TRIM(LOWER(ts.species_common_name)) = TRIM(LOWER(tc.spc_common))
GROUP BY species_common_name, form, fall_color
ORDER BY species_common_name, form, fall_color
```

```
SELECT
  species_common_name,
  form,
  fall_color,
  health,
  COUNT(DISTINCT tree_id) AS num_trees,
  AVG( tree_dbh) AS avg_tree_diam,
  MAX(tree_dbh) AS max_tree_diam
FROM `bigquery-public-data.new_york_trees.tree_species` ts
JOIN `bigquery-public-data.new_york_trees.tree_census_2015`
tc
ON TRIM(LOWER(ts.species_common_name)) =
TRIM(LOWER(tc.spc_common))
WHERE health != 'Good'
GROUP BY
  species_common_name,
  form,
  fall_color,
  health
HAVING num_trees>100
ORDER BY species_common_name, form, fall_color, health
```

```
SELECT
  c.country_name,
  c.country_code,
  c.country_area,
  CAST(CONCAT(CAST(p.year AS string),'-01','-01') AS date) AS year,
  p.midyear_population AS population,
  b.crude_birth_rate,
  b.crude_death_rate,
  b.growth_rate,
  b.net_migration,
  m.infant_mortality,
  m.life_expectancy,
  f.total_fertility_rate
FROM `bigquery-public-data.census_bureau_international.country_names_area` c
LEFT JOIN `bigquery-public-data.census_bureau_international.midyear_population` p
ON c.country_code = p.country_code
LEFT JOIN `bigquery-public-data.census_bureau_international.birth_death_growth_rates` b
ON p.country_code = b.country_code
AND p.year = b.year
LEFT JOIN `bigquery-public-data.census_bureau_international.mortality_life_expectancy` m
ON m.country_code = b.country_code
AND b.year = m.year
LEFT JOIN `bigquery-public-data.census_bureau_international.age_specific_fertility_rates` f
ON f.country_code = m.country_code
AND f.year = m.year
ORDER BY c.country_name, p.year
```

If you have to query a few columns but not all, EXCEPT is a useful syntax

```
SELECT * EXCEPT (removal_date)
FROM `bigquery-public-data.london_bicycles.cycle_stations`
```

Row	id	installed	latitude	locked	longitude	name	bikes_count	docks_count	nbEmptyDocks	temporary	terminal_name	install_date
1	564	true	51.509943	false	-0.117619	Somerset House, Strand	41	41	0	false	200068	*null*
2	228	true	51.50742485	false	-0.134621209	St. James's Square, St. James's	38	40	1	false	1067	2010-07-20

```
SELECT * EXCEPT (removal_date, temporary)
FROM `bigquery-public-data.london_bicycles.cycle_stations`
```

Row	id	installed	latitude	locked	longitude	name	bikes_count	docks_count	nbEmptyDocks	terminal_name	install_date
1	564	true	51.509943	false	-0.117619	Somerset House, Strand	41	41	0	200068	*null*
2	228	true	51.50742485	false	-0.134621209	St. James's Square, St. James's	38	40	1	1067	2010-07-20


```
SELECT *
FROM `bigquery-public-data.london_bicycles.cycle_stations`
WHERE CHARACTER_LENGTH(name) > 20;
```

VAR_POP() – variance / STDDEV_POP() – standard deviation

```
SELECT
VAR_POP(duration) AS variance_seconds,
STDDEV_POP(duration) AS standard_deviation_seconds,
VAR_POP(duration)/60 AS variance_minutes,
STDDEV_POP(duration)/60 AS standard_deviation_minutes,
FROM `bigquery-public-data.london_bicycles.cycle_hire`
```

```
SELECT *,
  IF(name LIKE '%Hackney%', true, false) station_in_hackney
FROM `bigquery-public-data.london_bicycles.cycle_stations`
WHERE name LIKE '%Hackney%'
```

```
SELECT *,
  IF(name LIKE '%Hackney%', true, false) AS station_in_hackney,
  IF(docks_count > 40, 'Large', NULL) AS is_large_docks,
FROM `bigquery-public-data.london_bicycles.cycle_stations`
WHERE name LIKE '%Hackney%'
```

```
SELECT
CASE
WHEN docks_count <= 10 THEN 'Small'
WHEN docks_count > 10 AND docks_count < 40 THEN
'Medium'
WHEN docks_count > 40 THEN 'Large'
ELSE NULL
END AS dock_size_desc
FROM `bigquery-public-data.london_bicycles.cycle_stations`
```

```
SELECT
  PARSE_TIMESTAMP ("%Y-%m-%d %T %p", CONCAT(CAST(d_date AS STRING), ' ', CAST(t_hour AS STRING), ':',
CAST(t_minute AS STRING), ':', CAST(t_second AS STRING), ' ',t_am_pm),"America/Los_Angeles") AS timestamp,
  s.ss_item_sk,
  i.i_product_name,
  s.ss_customer_sk,
  c.c_first_name,
  c.c_last_name,
  c.c_email_address,
  c.c_preferred_cust_flag,
  s.ss_quantity,
  s.ss_net_paid
FROM `dw-workshop.tpcds_2t_baseline.store_sales` AS s
JOIN `dw-workshop.tpcds_2t_baseline.date_dim` AS d
ON s.ss_sold_date_sk = d.d_date_sk
JOIN `dw-workshop.tpcds_2t_baseline.time_dim` AS t
ON s.ss_sold_time_sk = t.t_time_sk
JOIN `dw-workshop.tpcds_2t_baseline.item` AS i
ON s.ss_item_sk = i.i_item_sk
JOIN  `dw-workshop.tpcds_2t_baseline.customer` AS c
ON s.ss_customer_sk = c. c_customer_sk
WHERE d_date >= '2000-01-01' AND ss_customer_sk IS NOT NULL
ORDER BY ss_net_paid DESC
LIMIT 10
```

One Analytic Example:

Starting find the number of delayed flights by airline, still limited to La Guardia airport

```
SELECT
  airline,
  COUNT(departure_delay) as dep_delay
FROM `bigquery-samples.airline_ontime_data.flights`
WHERE departure_delay > 0
AND departure_airport = 'LGA'
GROUP BY airline
ORDER BY airline DESC
```

Filter results by a particular date

```
SELECT airline,
  COUNT(departure_delay) AS number_of_airline
FROM `bigquery-samples.airline_ontime_data.flights`
WHERE departure_delay > 0
AND departure_airport = 'LGA'
AND date = '2008-05-13'
GROUP BY airline
ORDER BY airline
```

In a single query find the “total number of flights”, “number of delayed flights”

```
SELECT f.airline,
  COUNT(f.departure_delay) AS total_flights,
  SUM(IF (f.departure_delay > 0, 1, 0)) AS num_delayed
FROM `bigquery-samples.airline_ontime_data.flights` AS f
WHERE f.departure_airport = 'LGA'
AND f.date = '2008-05-13'
GROUP BY airline
```

Now we start to think what could cause the delays, so what day was the raining in NYC?

```
SELECT year,
  month,
  day
FROM `bigquery-samples.weather_geo.gsod`
WHERE station_number = 725030
AND total_precipitation > 0
LIMIT 10
```

Get the dates for weather data in the same format as in the flights table
Yyyy-mm-dd using string functions and string concatination

```
SELECT CONCAT(CAST(year AS string), '-', LPAD(CAST(month AS
string), 2, '0'), '-', LPAD(CAST(day AS string), 2, '0'))
FROM `bigquery-samples.weather_geo.gsod`
WHERE station_number = 725030
AND total_precipitation > 0
LIMIT 10
```

Now we can perform joing operation and specify the desired in question

Delayed and total flights from earlier query, this time it is flights **arriving** at LGA

```
SELECT f.airline,
  COUNT(f.departure_delay) AS total_flights, SUM(IF(f.departure_delay > 0, 1, 0)) AS num_delayed
FROM `bigquery-samples.airline_ontime_data.flights` AS f
JOIN (SELECT CONCAT(CAST(year AS string), '-', LPAD(CAST(month AS string), 2, '0'), '-', LPAD(CAST(day AS string), 2, '0')) AS rainday
FROM `bigquery-samples.weather_geo.gsod`
WHERE station_number = 725030
AND total_precipitation > 0) AS w
ON w.rainday = f.date
WHERE f.arrival_airport = 'LGA'
GROUP BY f.airline
```

```
SELECT   airline,  num_delayed,  total_flights,  num_delayed / total_flights AS frac_delayed
FROM
  (SELECT  f.airline,  COUNT(f.departure_delay) AS total_flights, SUM(IF (f.departure_delay > 0, 1, 0)) AS num_delayed
FROM `bigquery-samples.airline_ontime_data.flights` AS f
JOIN (
   SELECT CONCAT(CAST(year AS string), '-', LPAD(CAST(month AS string), 2, '0'), '-', LPAD(CAST(day AS string), 2, '0')) AS
rainday
   FROM `bigquery-samples.weather_geo.gsod`
   WHERE station_number = 725030
   AND total_precipitation > 0) AS w
   ON w.rainday = f.date
   WHERE f.arrival_airport = 'LGA'
   GROUP BY f.airline
   )
ORDER BY frac_delayed DESC
```

REGULAR EXPRESSIONS:

```
SELECT
  title,
  SUM(views) views
FROM `bigquery-samples.wikimedia_pageviews.201112`
WHERE wikimedia_project = 'wp'
AND REGEXP_CONTAINS(title, 'Red.*t')
GROUP BY title
ORDER BY views DESC
```

Query a metadata, information schema

```
SELECT
  dataset_id,
  table_id,
  -- Convert bytes to GB.
  ROUND(size_bytes/pow(10,9),2) as size_gb,
  -- Convert UNIX EPOCH to a timestamp.
  TIMESTAMP_MILLIS(creation_time) AS creation_time,
  TIMESTAMP_MILLIS(last_modified_time) as last_modified_time,
  row_count,
  CASE
    WHEN type = 1 THEN 'table'
    WHEN type = 2 THEN 'view'
  ELSE NULL
  END AS type
FROM
  -- Replace baseball with a different dataset:
  `bigquery-public-data.baseball.__TABLES__`
ORDER BY size_gb DESC;
```

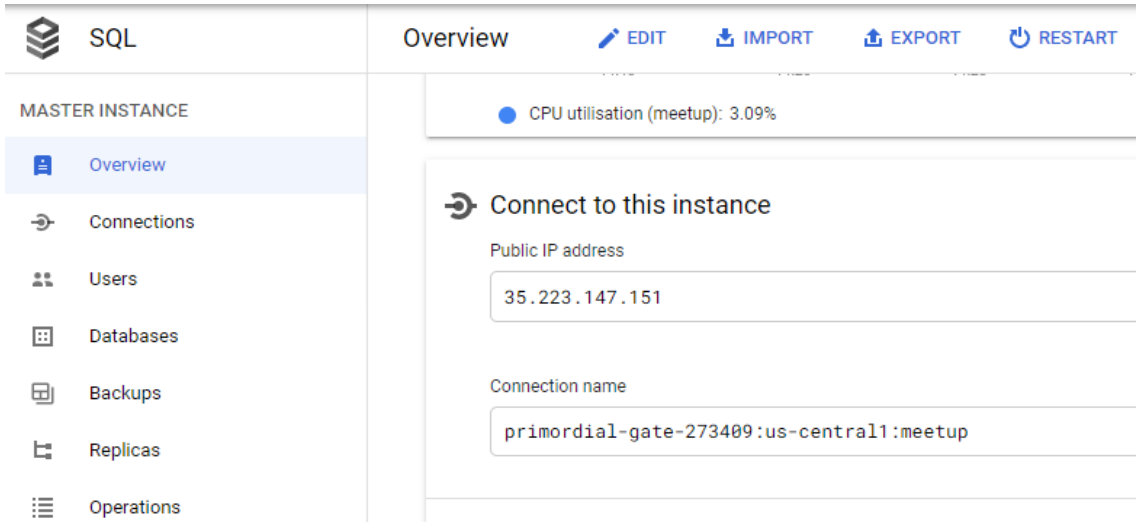
SELECT * FROM
`bigquery-public-data.baseball.INFORMATION_SCHEMA.COLUMNS`;

```
WITH ALL__TABLES__ AS (
  SELECT * FROM `bigquery-public-data.baseball.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.bls.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.census_bureau_usa.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.cloud_storage_geo_index.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.cms_codes.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.fec.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.genomics_cannabis.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.ghcn_d.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.ghcn_m.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.github_repos.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.hacker_news.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.irs_990.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.medicare.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.new_york.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.nlm_rxnorm.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.noaa_gsod.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.open_images.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.samples.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.san_francisco.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.stackoverflow.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.usa_names.__TABLES__` UNION ALL
  SELECT * FROM `bigquery-public-data.utility_us.__TABLES__`
)
SELECT *
FROM ALL__TABLES__
ORDER BY row_count DESC -- Top 10 tables with the most rows
LIMIT 10;
```

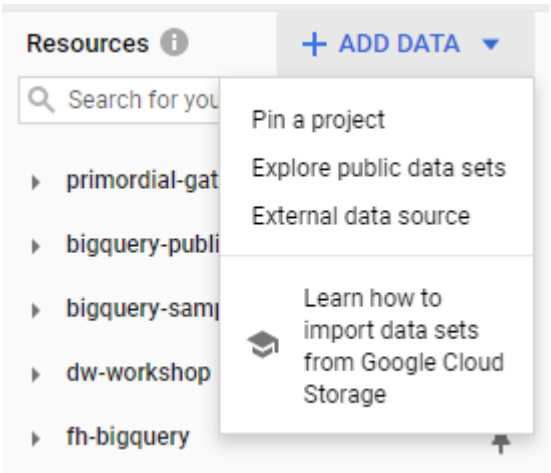
```
SELECT
  corpus,
  word,
  word_count,
  RANK() OVER (PARTITION BY corpus ORDER BY word_count DESC) rank
FROM
  `publicdata.samples.shakespeare`
WHERE
  LENGTH(word) > 10
  AND word_count > 10
ORDER BY rank DESC
LIMIT 40
```

Working with external connection.

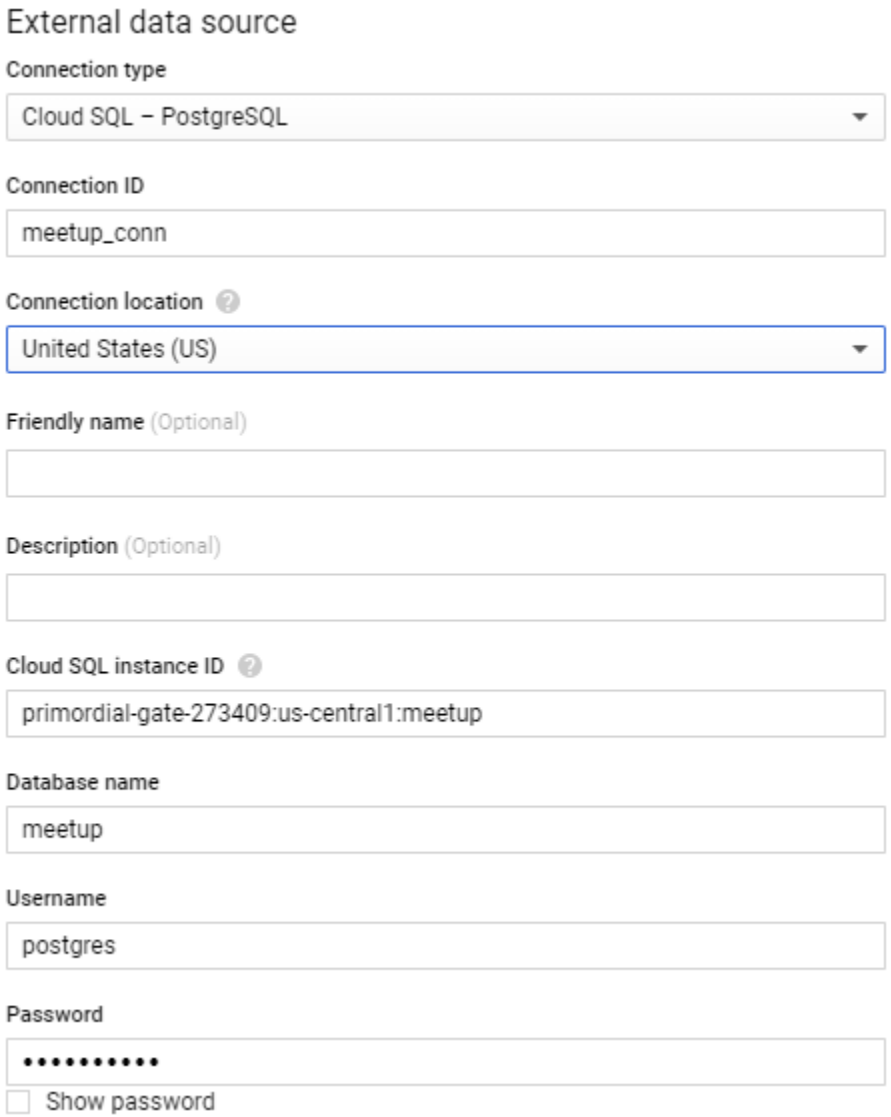
First, we need create a connection, to do so, copy your connection name from SQL overview



Than, from the BigQuery editor click on ADD DATA, External data source



Fill the external data source form



We are now ready to integrate BigQuery with external data source

```
SELECT *
FROM EXTERNAL_QUERY('us.meetup_conn', '''
SELECT *
FROM regions
''')
```

Row	region_id	region	country
1	1	Southwest	United States
2	2	Northeast	United States
3	3	Northwest	United States
4	4	Central	Asia
5	5	East Asia	Asia
6	6	Quebec	Canada
7	7	Nova Scotia	Canada

```
SELECT *
FROM `meetup_ds.department` LIMIT 1000
```

Employees who works on Movies

```
SELECT ex.*
FROM EXTERNAL_QUERY('us.meetup_conn', '''
SELECT *
FROM employees
''') AS ex
INNER JOIN `primordial-gate-273409.meetup_ds.department` AS d
ON ex.department = d.department
AND d.department = 'Movies'
```

Row	employee_id	first_name	last_name	email	hire_date	department	gender	salary	region_id
1	16	Merell	Yakovliv	myakovlivf@ucsd.edu	2008-08-16	Movies	M	78141	7
2	26	Frasquito	Cawson	*null*	2006-06-24	Movies	M	78881	1
3	27	Niles	Chawkley	nchawkleyq@flavors.me	2013-09-22	Movies	M	156303	3
4	37	Wilfrid	Sainer	wsainer10@gizmodo.com	2015-09-30	Movies	M	147235	3
5	96	Cal	Lowre	clowre2n@marketwatch.com	2010-12-30	Movies	F	47412	2
6	102	Bryna	Tarply	btarply2t@scientificamerican.com	2010-08-20	Movies	F	71440	7
7	108	Dionysus	Dumpleton	ddumpleton2z@typepad.com	2009-12-30	Movies	M	145973	5
8	110	Roarke	Sully	rsully31@army.mil	2004-10-26	Movies	M	86223	7
9	113	Myrtice	Emmens	*null*	2011-11-21	Movies	F	36919	6
10	118	Roxane	Raftery	rraftery39@arstechnica.com	2016-08-05	Movies	F	43199	1
11	132	Olivie	Issett	oissett3n@purevolume.com	2004-05-31	Movies	F	155316	4

Witch department has more employees?

```
SELECT ex.department,
  count(ex.department) num_empl_per_department
FROM EXTERNAL_QUERY('us.meetup_conn', '''
SELECT *
FROM employees
''') AS ex
INNER JOIN `primordial-gate-273409.meetup_ds.department` AS d
ON ex.department = d.department
GROUP BY ex.department
ORDER BY num_empl_per_department DESC
```

Row	department	num_empl_per_department
1	First Aid	58
2	Movies	56
3	Device Repair	51
4	Clothing	49
5	Computers	47
6	Toys	47
7	Children Clothing	47
8	Beauty	45
9	Furniture	43
10	Jewelry	41
11	Garden	41