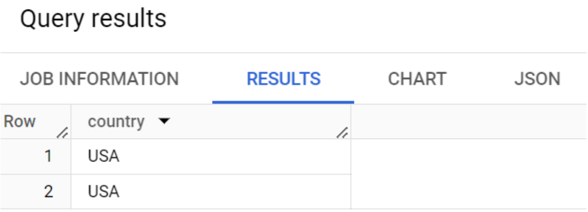
SQL Queries Used

1. GROUP BY & ORDER BY

```
SELECT
  language,
  title,
  SUM(views) AS views
FROM
  `bigquery-samples.wikipedia_benchmark.Wiki10B`
WHERE
  title LIKE '%Google%'
GROUP BY
  language,
  title
ORDER BY
  views DESC;
```

2. LENGTH

```
SELECT
  country
FROM
  my-first-project-418418.customer_data.customer_address
WHERE
  LENGTH(country) > 2
```



Row	country
1	USA
2	USA

3. Filter Using WHERE

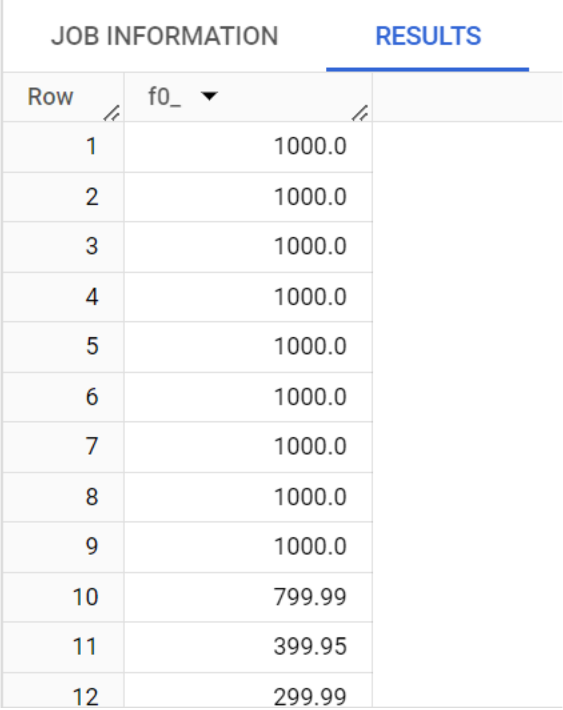
```
UPDATE
 my-first-project-418418.cars.car_info
SET
 num_of_doors = "four"
WHERE
 make = "dodge"
 AND fuel_type = "gas"
 AND body_style = "sedan";
```

4. Update Query

```
UPDATE
  my-first-project-418418.cars.car_info
SET
  price = 5118
WHERE
  price = 0
```

5. **CASTE**

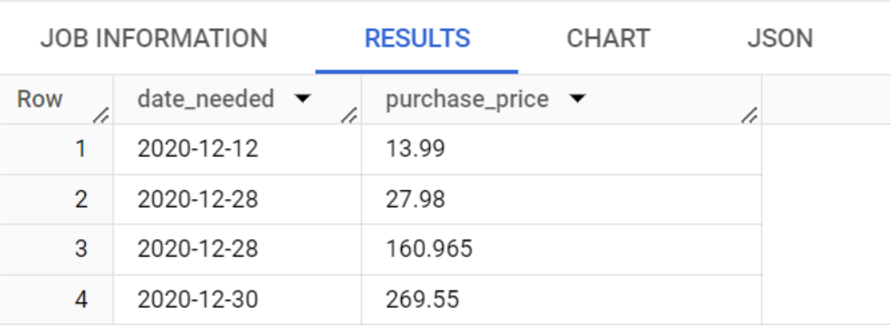
```
SELECT
  CAST(purchase_price AS float64)
  #purchase_price
FROM
  my-first-project-418418.customer_data.customer_purchase
ORDER BY
  CAST(purchase_price AS float64) DESC
```



Row	f0_
1	1000.0
2	1000.0
3	1000.0
4	1000.0
5	1000.0
6	1000.0
7	1000.0
8	1000.0
9	1000.0
10	799.99
11	399.95
12	299.99

6. **FILTER Using Between**

```
SELECT
  CAST (date AS date) AS date_needed,
  purchase_price
FROM
  my-first-project-418418.customer_data.customer_purchase
WHERE
  date BETWEEN '2020-12-01' AND '2020-12-31'
```



Row	date_needed	purchase_price
1	2020-12-12	13.99
2	2020-12-28	27.98
3	2020-12-28	160.965
4	2020-12-30	269.55

MOVIE DATESET SQL COMMANDS

7. Sort data by one column

```
SELECT *
FROM projectID.movie_data.movies
ORDER BY Release_date;
```

8. Sort data in descending order

```
SELECT *
FROM projectID.movie_data.movies
ORDER BY Release_Date DESC;
```

9. Filter and sort data in descending order

```
SELECT *
FROM projectID.movie_data.movies
WHERE Genre = "Comedy"
ORDER BY Release_Date DESC;
```

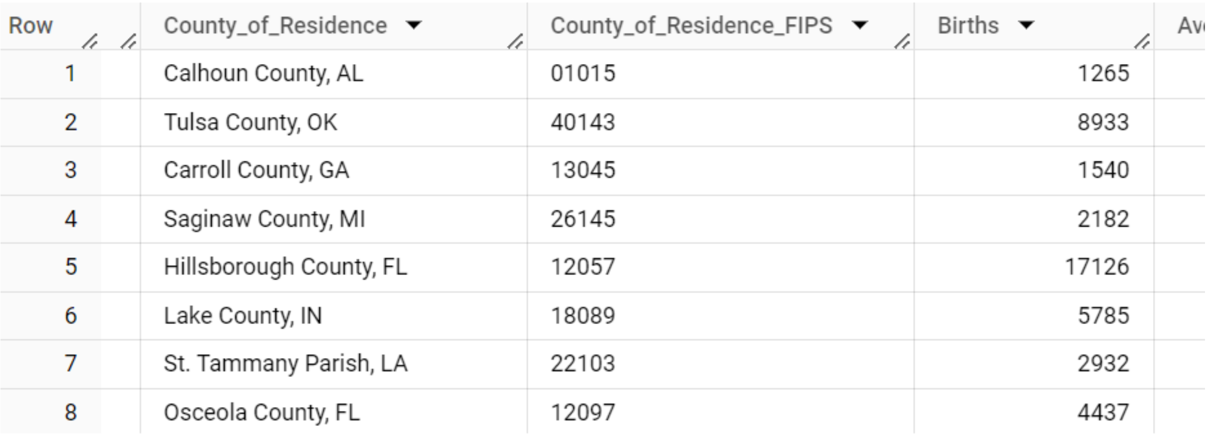
10. Filter on two conditions, then sort data in descending order

```
SELECT *
FROM projectID.movie_data.movies
WHERE Genre = "Comedy"
AND Revenue > 300000000
ORDER BY Release_Date DESC;
```

HANDON TASK

11. LOAD THE CBC DATASET

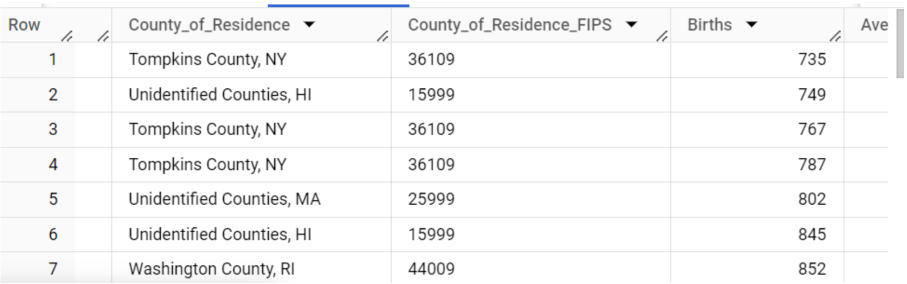
```
SELECT
  *
FROM
  bigquery-public-data.sdoh_cdc_wonder_natality.county_natality
LIMIT
  1000
```



Row	County_of_Residence	County_of_Residence_FIPS	Births	Av
1	Calhoun County, AL	01015	1265	
2	Tulsa County, OK	40143	8933	
3	Carroll County, GA	13045	1540	
4	Saginaw County, MI	26145	2182	
5	Hillsborough County, FL	12057	17126	
6	Lake County, IN	18089	5785	
7	St. Tammany Parish, LA	22103	2932	
8	Osceola County, FL	12097	4437	

12. ORDER BY Births

```
SELECT
  *
FROM
  bigquery-public-data.sdoh_cdc_wonder_natality.county_natality
ORDER BY
  Births
LIMIT
  1000
```



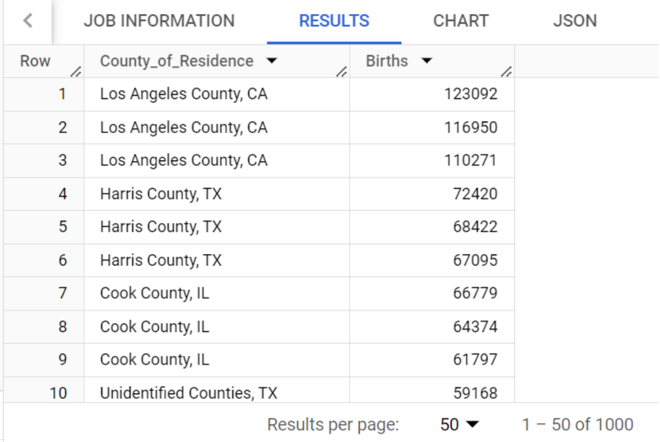
Row	County_of_Residence	County_of_Residence_FIPS	Births	Ave
1	Tompkins County, NY	36109	735	
2	Unidentified Counties, HI	15999	749	
3	Tompkins County, NY	36109	767	
4	Tompkins County, NY	36109	787	
5	Unidentified Counties, MA	25999	802	
6	Unidentified Counties, HI	15999	845	
7	Washington County, RI	44009	852	

13. ORDER BY Births in Descending order

```
SELECT
  *
FROM
  bigquery-public-data.sdoh_cdc_wonder_natality.county_natality
ORDER BY
  Births DESC
LIMIT
  1000
```

14. Country_of_residence grouped and sorted by Births in Desc

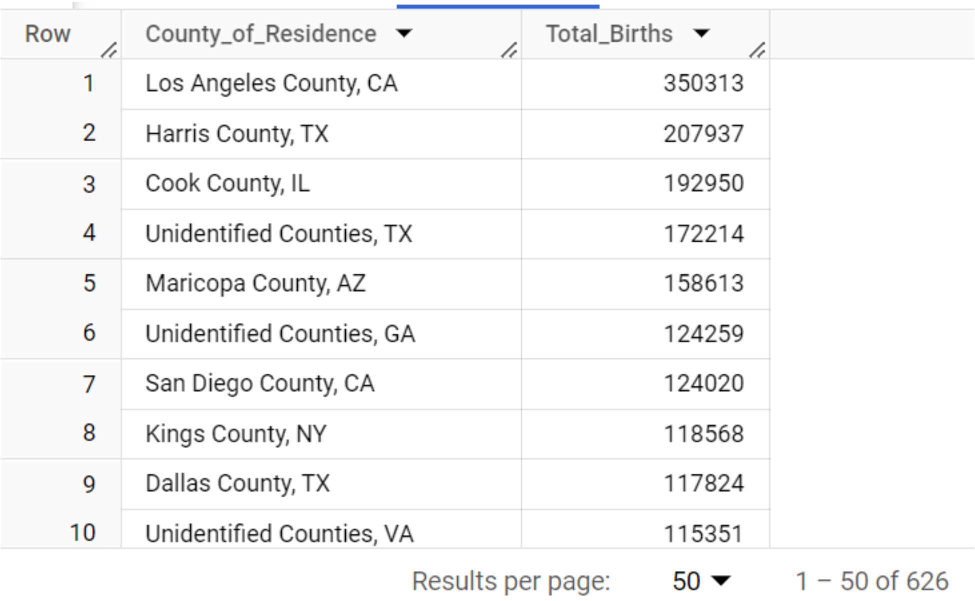
```
SELECT
  County_of_Residence,
  Births
FROM
  bigquery-public-data.sdoh_cdc_wonder_natality.county_natality
GROUP BY
  County_of_Residence,
  Births
ORDER BY
  Births DESC
LIMIT
  1000
```



Row	County_of_Residence	Births
1	Los Angeles County, CA	123092
2	Los Angeles County, CA	116950
3	Los Angeles County, CA	110271
4	Harris County, TX	72420
5	Harris County, TX	68422
6	Harris County, TX	67095
7	Cook County, IL	66779
8	Cook County, IL	64374
9	Cook County, IL	61797
10	Unidentified Counties, TX	59168

15. Used sum to find the total Births count GROUPED BY County_of_residence

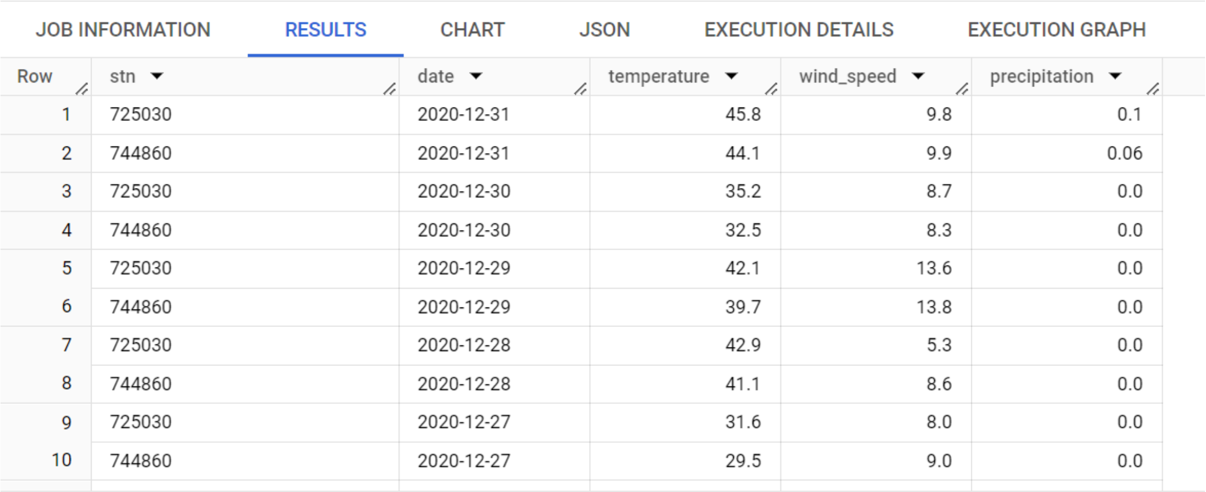
```
SELECT
  County_of_Residence,
  SUM(Births) AS Total_Births
FROM
  bigquery-public-data.sdoh_cdc_wonder_natality.county_natality
GROUP BY
  County_of_Residence
ORDER BY
  Total_Births DESC
LIMIT
  1000
```



Row	County_of_Residence	Total_Births
1	Los Angeles County, CA	350313
2	Harris County, TX	207937
3	Cook County, IL	192950
4	Unidentified Counties, TX	172214
5	Maricopa County, AZ	158613
6	Unidentified Counties, GA	124259
7	San Diego County, CA	124020
8	Kings County, NY	118568
9	Dallas County, TX	117824
10	Unidentified Counties, VA	115351

16. Query to Sort columns by dates in Desc and Station_Id in Ascending.

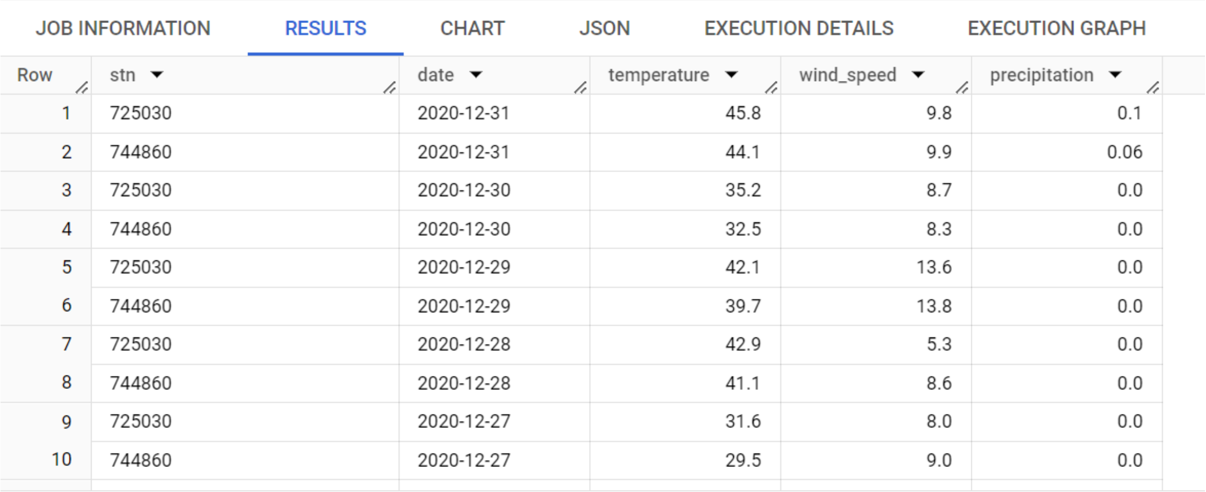
```
SELECT
  stn,
  date,
  IF(
    temp=9999.9,
    NULL,
    temp) AS temperature,
  IF(
    wdsp="999.9",
    NULL,
    CAST(wdsp AS float64)) AS wind_speed,
  IF(
    prcp=99.99,
    0,
    prcp) AS precipitation
FROM
  bigquery-public-data.noaa_gsod.gsod2020
WHERE
  stn="725030" -- La Guardia
  OR stn="744860" -- JFK
ORDER BY
  date DESC,
  stn ASC
```



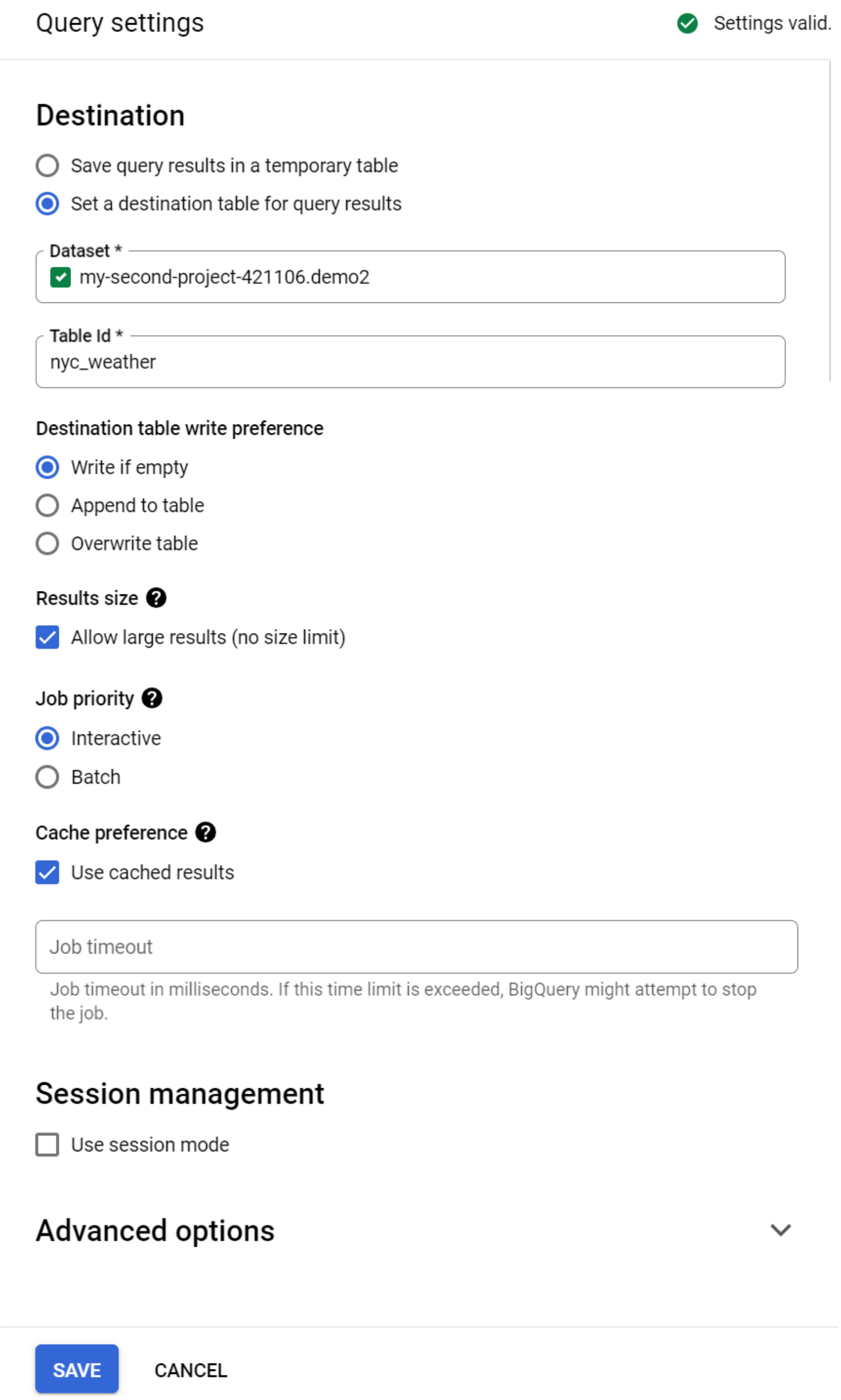
Row	stn	date	temperature	wind_speed	precipitation
1	725030	2020-12-31	45.8	9.8	0.1
2	744860	2020-12-31	44.1	9.9	0.06
3	725030	2020-12-30	35.2	8.7	0.0
4	744860	2020-12-30	32.5	8.3	0.0
5	725030	2020-12-29	42.1	13.6	0.0
6	744860	2020-12-29	39.7	13.8	0.0
7	725030	2020-12-28	42.9	5.3	0.0
8	744860	2020-12-28	41.1	8.6	0.0
9	725030	2020-12-27	31.6	8.0	0.0
10	744860	2020-12-27	29.5	9.0	0.0

17. Created a new table after Running a SQL query

```
SELECT
  stn,
  date,
  IF(
    temp=9999.9,
    NULL,
    temp) AS temperature,
  IF(
    wdsp="999.9",
    NULL,
    CAST(wdsp AS float64)) AS wind_speed,
  IF(
    prcp=99.99,
    0,
    prcp) AS precipitation
FROM
  `bigquery-public-data.noaa_gsod.gsod2020`
WHERE
  stn="725030" -- La Guardia
  OR stn="744860" -- JFK
ORDER BY
  date DESC,
  stn ASC
```



Row	stn	date	temperature	wind_speed	precipitation
1	725030	2020-12-31	45.8	9.8	0.1
2	744860	2020-12-31	44.1	9.9	0.06
3	725030	2020-12-30	35.2	8.7	0.0
4	744860	2020-12-30	32.5	8.3	0.0
5	725030	2020-12-29	42.1	13.6	0.0
6	744860	2020-12-29	39.7	13.8	0.0
7	725030	2020-12-28	42.9	5.3	0.0
8	744860	2020-12-28	41.1	8.6	0.0
9	725030	2020-12-27	31.6	8.0	0.0
10	744860	2020-12-27	29.5	9.0	0.0



Query settings ✓ Settings valid.

Destination

○ Save query results in a temporary table

● Set a destination table for query results

Dataset * ☑ my-second-project-421106.demo2

Table Id * nyc_weather

Destination table write preference

● Write if empty

○ Append to table

○ Overwrite table

Results size

☑ Allow large results (no size limit)

Job priority

● Interactive

○ Batch

Cache preference

☑ Use cached results

Job timeout

Job timeout in milliseconds. If this time limit is exceeded, BigQuery might attempt to stop the job.

Session management

□ Use session mode

Advanced options

After saving the above setting a new table inside "*demo2*" dataset will be created with the name "*nyc_weather*"

Combining data from different sources

A. INSERT INTO

Structure of the query

```
INSERT INTO [destination_table_name]
SELECT [column names, separated by commas, or * for all columns]
FROM [source_table_name]
WHERE [condition]
```

Example of Query

```
INSERT INTO customer_promotion
SELECT *
FROM customers
WHERE total_sales = 0 AND postal_code = '12345'
```

B. CONCAT

Structure of the query

```
SELECT CONCAT(field1, " ", field2)
FROM [table_name]
```

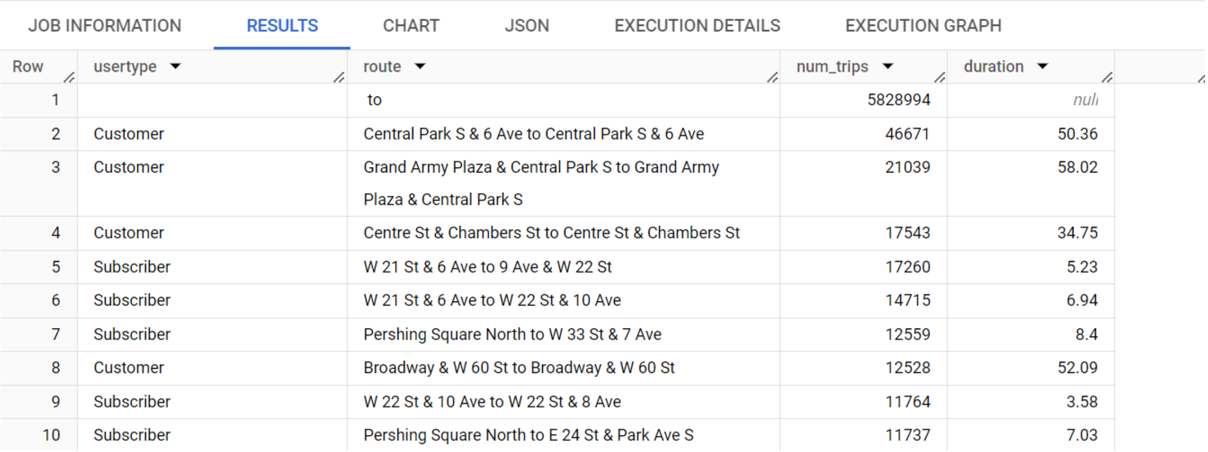
```
SELECT CONCAT(field1, " ", field2) AS alias
FROM [table_name]
```

Example of the query

```
SELECT CONCAT(first_name, " ", last_name) AS Customer_Name
FROM [table_name]
```

18. Merging using CONCAT

```
SELECT
  usertype,
  CONCAT (start_station_name," to ", end_station_name) AS route,
  COUNT (*) as num_trips,
  ROUND(AVG(cast(tripduration as int64)/60),2) AS duration
FROM
  `bigquery-public-data.new_york_citibike.citibike_trips`
GROUP BY
  start_station_name, end_station_name, usertype
ORDER BY
  num_trips DESC
LIMIT 10
```

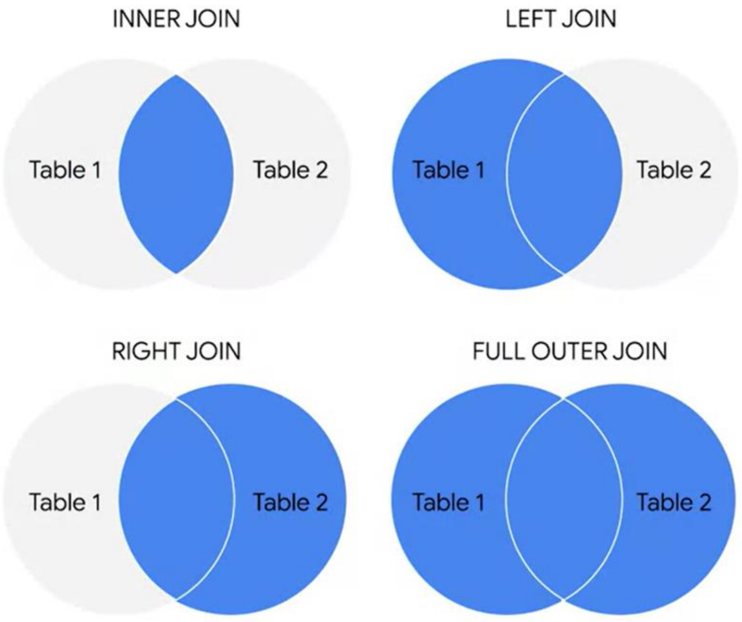


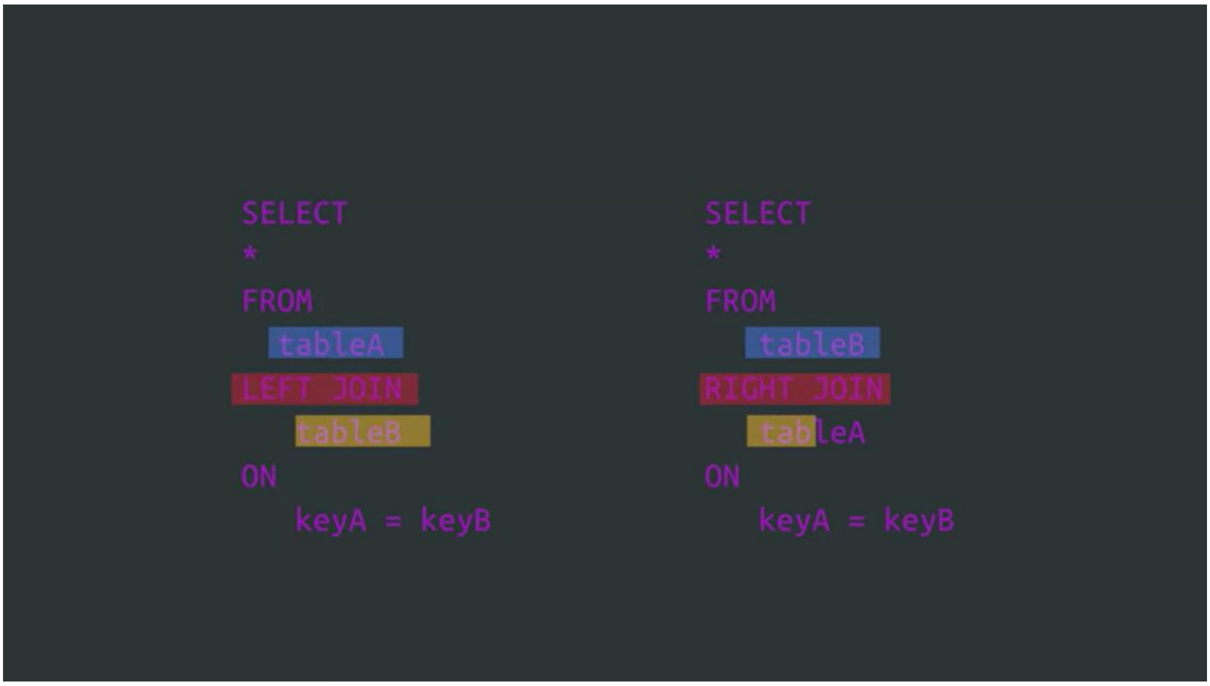
Row	usertype	route	num_trips	duration
1		to	5828994	null
2	Customer	Central Park S & 6 Ave to Central Park S & 6 Ave	46671	50.36
3	Customer	Grand Army Plaza & Central Park S to Grand Army Plaza & Central Park S	21039	58.02
4	Customer	Centre St & Chambers St to Centre St & Chambers St	17543	34.75
5	Subscriber	W 21 St & 6 Ave to 9 Ave & W 22 St	17260	5.23
6	Subscriber	W 21 St & 6 Ave to W 22 St & 10 Ave	14715	6.94
7	Subscriber	Pershing Square North to W 33 St & 7 Ave	12559	8.4
8	Customer	Broadway & W 60 St to Broadway & W 60 St	12528	52.09
9	Subscriber	W 22 St & 10 Ave to W 22 St & 8 Ave	11764	3.58
10	Subscriber	Pershing Square North to E 24 St & Park Ave S	11737	7.03

Joins Using SQL.

There are four types of JOIN in SQL. Below is the information regarding all the four.

- `INNER JOIN`: a function that returns records with matching values in both tables
- `LEFT JOIN`: a function that returns all the records from the left table (first mentioned) and only the matching records from the right table (second mentioned)
- `RIGHT JOIN`: a function that returns all records from the right table (second mentioned) and only the matching records from the left table (first mentioned).
- `OUTER JOIN`: a function that combines the `RIGHT JOIN` and `LEFT JOIN` to return all matching records in both tables.





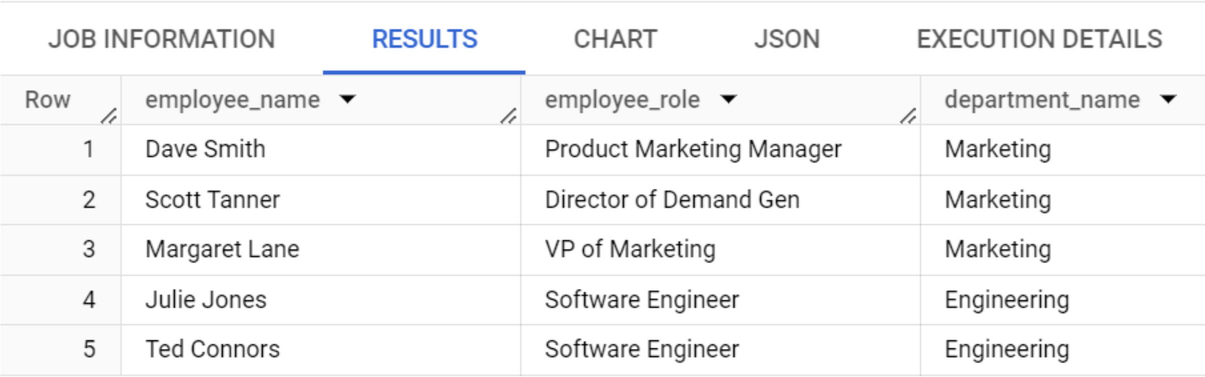
```
SELECT
*
FROM
  tableA
LEFT JOIN
    tableB
ON
    keyA = keyB
```

```
SELECT
*
FROM
  tableB
RIGHT JOIN
    tableA
ON
    keyA = keyB
```

In the above SS both the queries will return the same result.

19. INNER JOIN

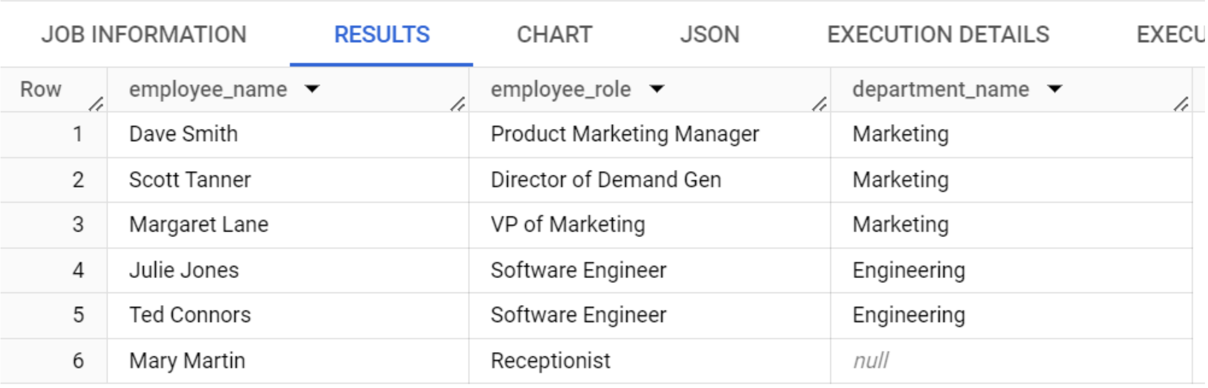
```
SELECT
  employees.name AS employee_name,
  employees.role AS employee_role,
  departments.name AS department_name
FROM
  my-first-project-418418.employee_data.employees AS employees
INNER JOIN
  my-first-project-418418.employee_data.departments AS departments
  ON employees.department_id = departments.department_id
```



Row	employee_name	employee_role	department_name
1	Dave Smith	Product Marketing Manager	Marketing
2	Scott Tanner	Director of Demand Gen	Marketing
3	Margaret Lane	VP of Marketing	Marketing
4	Julie Jones	Software Engineer	Engineering
5	Ted Connors	Software Engineer	Engineering

20. LEFT JOIN

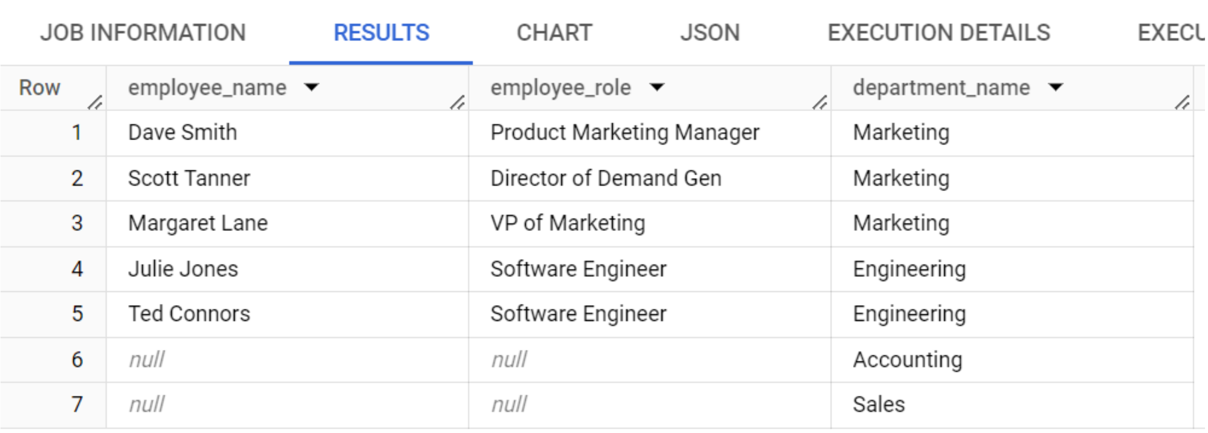
```
SELECT
  employees.name AS employee_name,
  employees.role AS employee_role,
  departments.name AS department_name
FROM
  my-first-project-418418.employee_data.employees AS employees
LEFT JOIN
  my-first-project-418418.employee_data.departments AS departments
  ON employees.department_id = departments.department_id
```



Row	employee_name	employee_role	department_name
1	Dave Smith	Product Marketing Manager	Marketing
2	Scott Tanner	Director of Demand Gen	Marketing
3	Margaret Lane	VP of Marketing	Marketing
4	Julie Jones	Software Engineer	Engineering
5	Ted Connors	Software Engineer	Engineering
6	Mary Martin	Receptionist	*null*

21. RIGHT JOIN

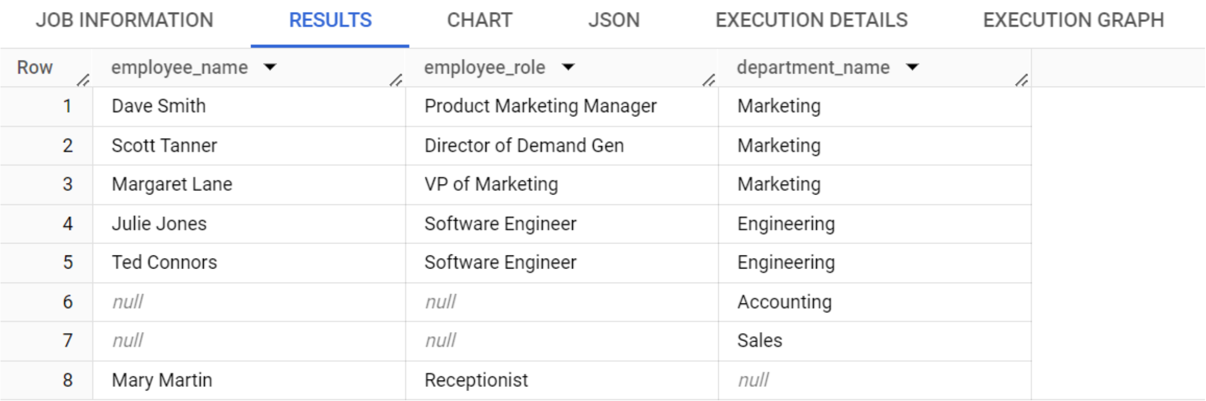
```
SELECT
  employees.name AS employee_name,
  employees.role AS employee_role,
  departments.name AS department_name
FROM
  my-first-project-418418.employee_data.employees AS employees
RIGHT JOIN
  my-first-project-418418.employee_data.departments AS departments
  ON employees.department_id = departments.department_id
```



Row	employee_name	employee_role	department_name
1	Dave Smith	Product Marketing Manager	Marketing
2	Scott Tanner	Director of Demand Gen	Marketing
3	Margaret Lane	VP of Marketing	Marketing
4	Julie Jones	Software Engineer	Engineering
5	Ted Connors	Software Engineer	Engineering
6	*null*	*null*	Accounting
7	*null*	*null*	Sales

22. OUTER JOIN

```
SELECT
  employees.name AS employee_name,
  employees.role AS employee_role,
  departments.name AS department_name
FROM
  my-first-project-418418.employee_data.employees AS employees
FULL OUTER JOIN
  my-first-project-418418.employee_data.departments AS departments
  ON employees.department_id = departments.department_id
```

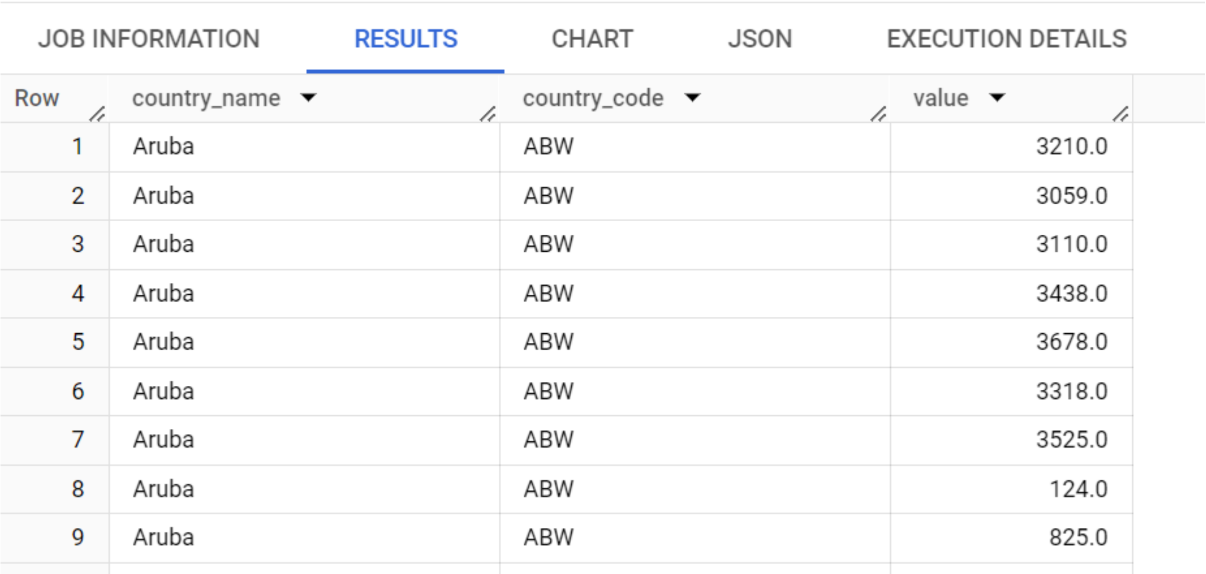


Row	employee_name	employee_role	department_name
1	Dave Smith	Product Marketing Manager	Marketing
2	Scott Tanner	Director of Demand Gen	Marketing
3	Margaret Lane	VP of Marketing	Marketing
4	Julie Jones	Software Engineer	Engineering
5	Ted Connors	Software Engineer	Engineering
6	*null*	*null*	Accounting
7	*null*	*null*	Sales
8	Mary Martin	Receptionist	*null*

Joins Hands On

23. INNER JOIN

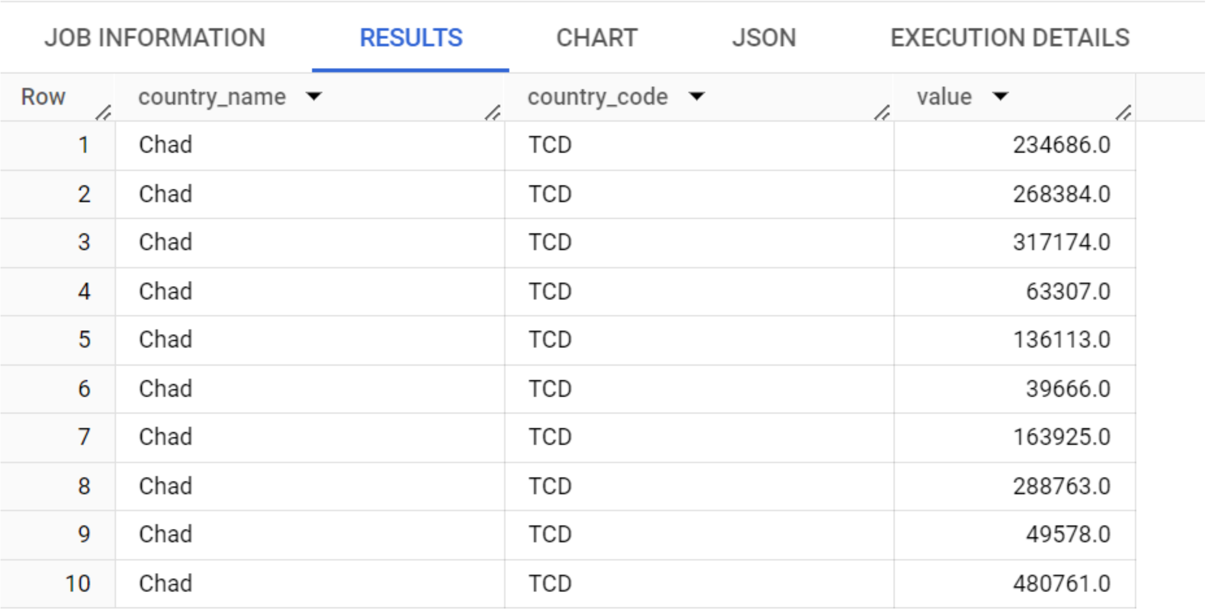
```
SELECT `bigquery-public-
data.world_bank_intl_education.international_education`.country_name,
    `bigquery-public-data.world_bank_intl_education.country_summary`.country_code,
    `bigquery-public-data.world_bank_intl_education.international_education`.value
FROM `bigquery-public-data.world_bank_intl_education.international_education`
INNER JOIN `bigquery-public-data.world_bank_intl_education.country_summary`
ON `bigquery-public-data.world_bank_intl_education.country_summary`.country_code =
`bigquery-public-
data.world_bank_intl_education.international_education`.country_code
```



Row	country_name	country_code	value
1	Aruba	ABW	3210.0
2	Aruba	ABW	3059.0
3	Aruba	ABW	3110.0
4	Aruba	ABW	3438.0
5	Aruba	ABW	3678.0
6	Aruba	ABW	3318.0
7	Aruba	ABW	3525.0
8	Aruba	ABW	124.0
9	Aruba	ABW	825.0

24. INNER Join Using Alias. (The same above query using Alias)

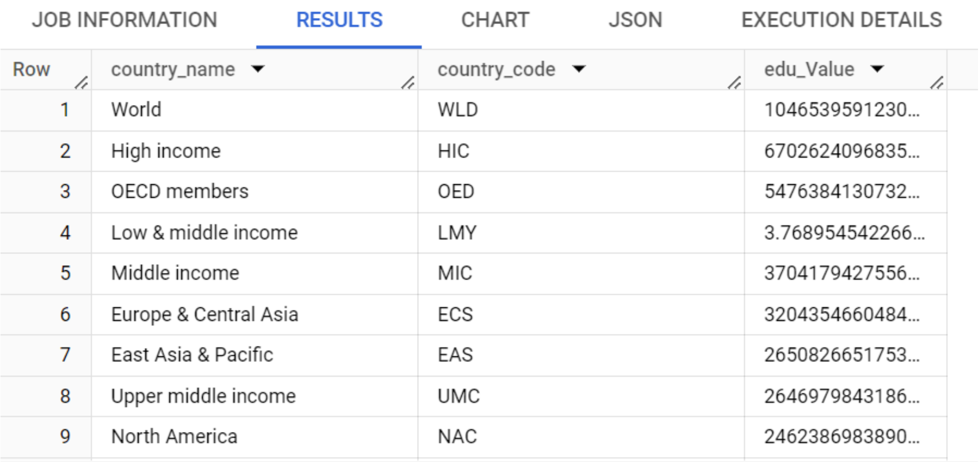
```
SELECT
    edu.country_name,
    summary.country_code,
    edu.value
FROM
    `bigquery-public-data.world_bank_intl_education.international_education` AS edu
INNER JOIN
    `bigquery-public-data.world_bank_intl_education.country_summary` AS summary
ON edu.country_code = summary.country_code
```



Row	country_name	country_code	value
1	Chad	TCD	234686.0
2	Chad	TCD	268384.0
3	Chad	TCD	317174.0
4	Chad	TCD	63307.0
5	Chad	TCD	136113.0
6	Chad	TCD	39666.0
7	Chad	TCD	163925.0
8	Chad	TCD	288763.0
9	Chad	TCD	49578.0
10	Chad	TCD	480761.0

25. Inner Join with Group By

```
SELECT
    edu.country_name,
    summary.country_code,
    SUM(edu.value) edu_Value
FROM
    `bigquery-public-data.world_bank_intl_education.international_education` AS edu
LEFT JOIN
    `bigquery-public-data.world_bank_intl_education.country_summary` AS summary
ON edu.country_code = summary.country_code
GROUP BY
    edu.country_name,
    summary.country_code
ORDER BY
    edu_Value DESC
```

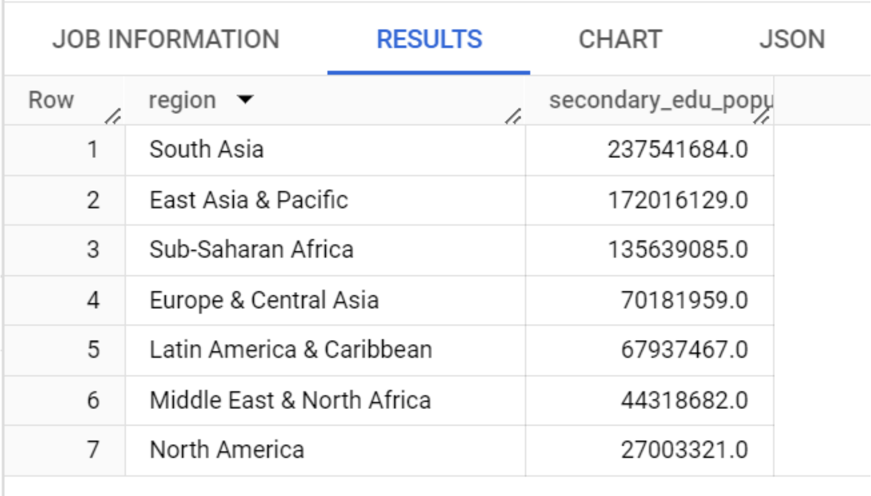


Row	country_name	country_code	edu_Value
1	World	WLD	1046539591230…
2	High income	HIC	6702624096835…
3	OECD members	OED	5476384130732…
4	Low & middle income	LMY	3.768954542266…
5	Middle income	MIC	3704179427556…
6	Europe & Central Asia	ECS	3204354660484…
7	East Asia & Pacific	EAS	2650826651753…
8	Upper middle income	UMC	2646979843186…
9	North America	NAC	2462386983890…

26. INNER JOIN with WHERE Condition

question: In 2015, how many people were of the official age for secondary education broken down by region of the world?

```
SELECT
summary.region,
SUM(edu.value) secondary_edu_population
FROM
    `bigquery-public-data.world_bank_intl_education.international_education` AS edu
INNER JOIN
    `bigquery-public-data.world_bank_intl_education.country_summary` AS summary
ON edu.country_code = summary.country_code --country_code is our key
    WHERE summary.region IS NOT NULL
    AND edu.indicator_name = 'Population of the official age for secondary
education, both sexes (number)'
    AND edu.year = 2015
GROUP BY summary.region
ORDER BY secondary_edu_population DESC
```

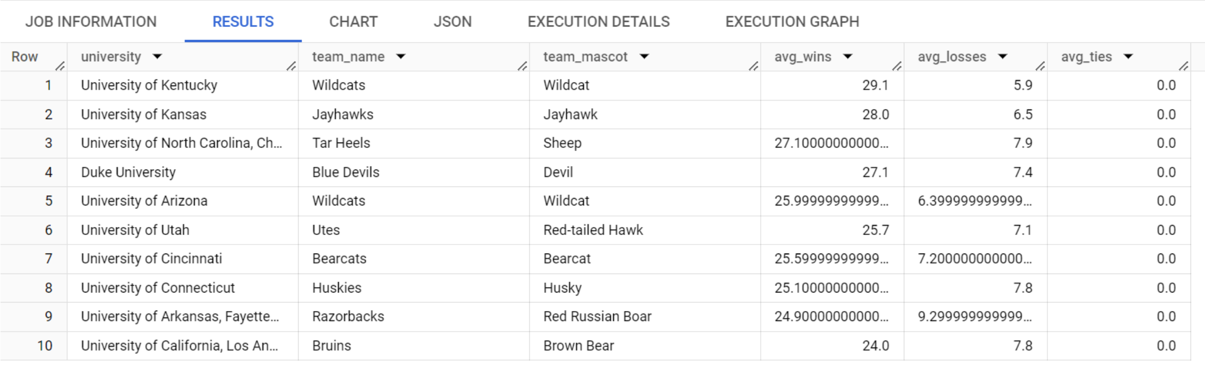


Row	region	secondary_edu_popu
1	South Asia	237541684.0
2	East Asia & Pacific	172016129.0
3	Sub-Saharan Africa	135639085.0
4	Europe & Central Asia	70181959.0
5	Latin America & Caribbean	67937467.0
6	Middle East & North Africa	44318682.0
7	North America	27003321.0

27. LEFT JOIN

Consider this scenario: You have been tasked to provide data for a feature sports article on NCAA basketball in the 1990s. The writer wants to include a funny twist about which Division 1 team mascots were the winningest.

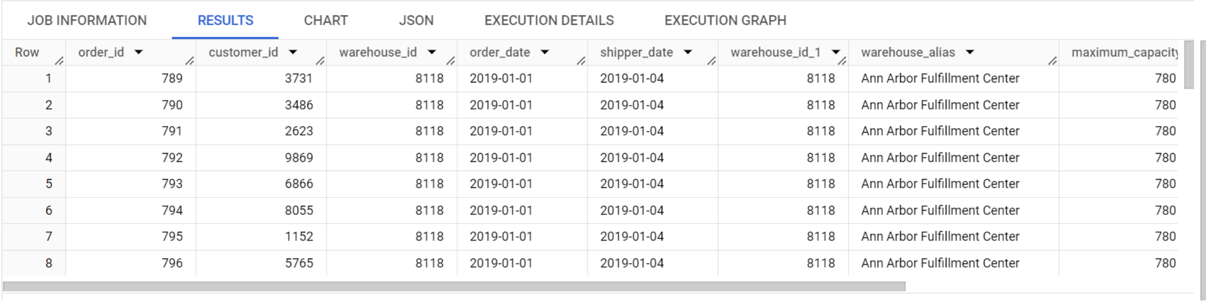
```
SELECT
 seasons.market AS university,
 seasons.name AS team_name,
 mascots.mascot AS team_mascot,
 AVG(seasons.wins) AS avg_wins,
 AVG(seasons.losses) AS avg_losses,
 AVG(seasons.ties) AS avg_ties
FROM `bigquery-public-data.ncaa_basketball.mbb_historical_teams_seasons` AS seasons
LEFT JOIN `bigquery-public-data.ncaa_basketball.mascots` AS mascots
ON seasons.team_id = mascots.id
WHERE seasons.season BETWEEN 1990 AND 1999
 AND seasons.division = 1
 GROUP BY 1,2,3
ORDER BY avg_wins DESC, university
```



Row	university	team_name	team_mascot	avg_wins	avg_losses	avg_ties
1	University of Kentucky	Wildcats	Wildcat	29.1	5.9	0.0
2	University of Kansas	Jayhawks	Jayhawk	28.0	6.5	0.0
3	University of North Carolina, Ch...	Tar Heels	Sheep	27.10000000000...	7.9	0.0
4	Duke University	Blue Devils	Devil	27.1	7.4	0.0
5	University of Arizona	Wildcats	Wildcat	25.99999999999...	6.399999999999...	0.0
6	University of Utah	Utes	Red-tailed Hawk	25.7	7.1	0.0
7	University of Cincinnati	Bearcats	Bearcat	25.59999999999...	7.200000000000...	0.0
8	University of Connecticut	Huskies	Husky	25.10000000000...	7.8	0.0
9	University of Arkansas, Fayette...	Razorbacks	Red Russian Boar	24.90000000000...	9.299999999999...	0.0
10	University of California, Los An...	Bruins	Brown Bear	24.0	7.8	0.0

28. INNER JOIN for Handson activity

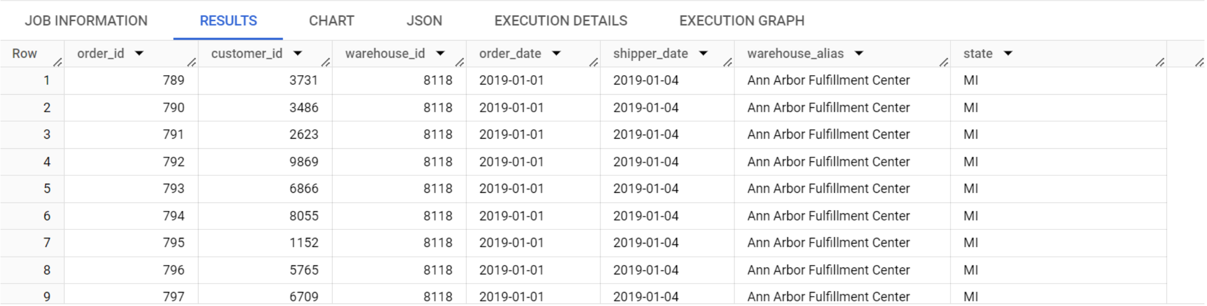
```
SELECT
  *
FROM
  my-first-project-418418.warehouse_orders.orders AS orders
JOIN
  my-first-project-418418.warehouse_orders.warehouse warehouse
ON
  orders.warehouse_id = warehouse.warehouse_id
```



Row	order_id	customer_id	warehouse_id	order_date	shipper_date	warehouse_id_1	warehouse_alias	maximum_capacity
1	789	3731	8118	2019-01-01	2019-01-04	8118	Ann Arbor Fulfillment Center	780
2	790	3486	8118	2019-01-01	2019-01-04	8118	Ann Arbor Fulfillment Center	780
3	791	2623	8118	2019-01-01	2019-01-04	8118	Ann Arbor Fulfillment Center	780
4	792	9869	8118	2019-01-01	2019-01-04	8118	Ann Arbor Fulfillment Center	780
5	793	6866	8118	2019-01-01	2019-01-04	8118	Ann Arbor Fulfillment Center	780
6	794	8055	8118	2019-01-01	2019-01-04	8118	Ann Arbor Fulfillment Center	780
7	795	1152	8118	2019-01-01	2019-01-04	8118	Ann Arbor Fulfillment Center	780
8	796	5765	8118	2019-01-01	2019-01-04	8118	Ann Arbor Fulfillment Center	780

29. Taking All the columns from Orders but only 2 columns from Warehouse

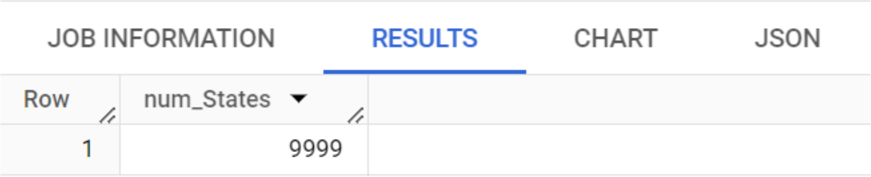
```
SELECT
  orders.*,
  warehouse.warehouse_alias,
  warehouse.state
FROM
  my-first-project-418418.warehouse_orders.orders AS orders
JOIN
  my-first-project-418418.warehouse_orders.warehouse warehouse
ON
  orders.warehouse_id = warehouse.warehouse_id
```



Row	order_id	customer_id	warehouse_id	order_date	shipper_date	warehouse_alias	state
1	789	3731	8118	2019-01-01	2019-01-04	Ann Arbor Fulfillment Center	MI
2	790	3486	8118	2019-01-01	2019-01-04	Ann Arbor Fulfillment Center	MI
3	791	2623	8118	2019-01-01	2019-01-04	Ann Arbor Fulfillment Center	MI
4	792	9869	8118	2019-01-01	2019-01-04	Ann Arbor Fulfillment Center	MI
5	793	6866	8118	2019-01-01	2019-01-04	Ann Arbor Fulfillment Center	MI
6	794	8055	8118	2019-01-01	2019-01-04	Ann Arbor Fulfillment Center	MI
7	795	1152	8118	2019-01-01	2019-01-04	Ann Arbor Fulfillment Center	MI
8	796	5765	8118	2019-01-01	2019-01-04	Ann Arbor Fulfillment Center	MI
9	797	6709	8118	2019-01-01	2019-01-04	Ann Arbor Fulfillment Center	MI

30. Using COUNT with INNER Join

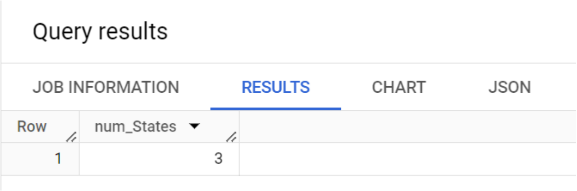
```
SELECT
  COUNT(warehouse.state) AS num_States
FROM
  my-first-project-418418.warehouse_orders.orders AS orders
JOIN
  my-first-project-418418.warehouse_orders.warehouse warehouse
ON
  orders.warehouse_id = warehouse.warehouse_id
```



Row	num_States
1	9999

31. Using DISTINCT COUNT with INNER Join

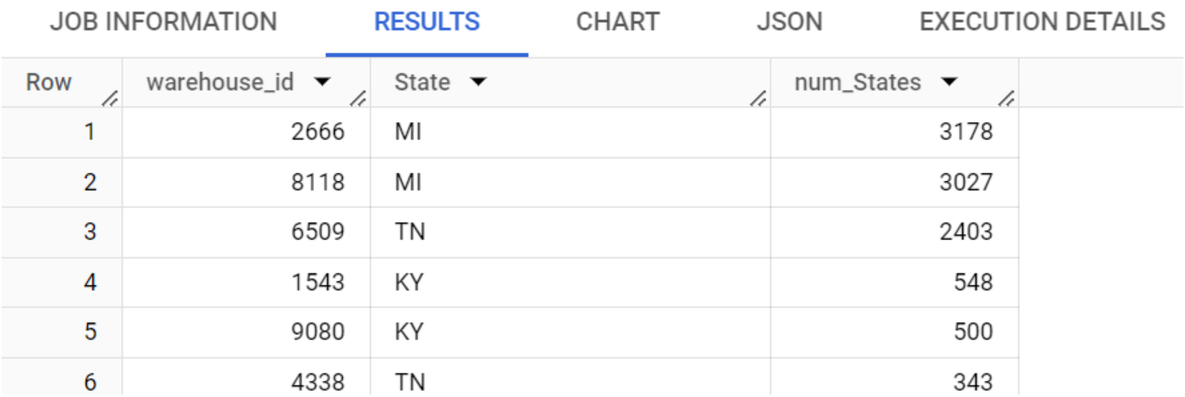
```
SELECT
  COUNT (DISTINCT warehouse.state) AS num_States
FROM
  my-first-project-418418.warehouse_orders.orders AS orders
JOIN
  my-first-project-418418.warehouse_orders.warehouse warehouse
ON
  orders.warehouse_id = warehouse.warehouse_id
```


Query results

Row	num_States
1	3

32. USING GROUP BY & ORDER BY & JOIN & COUNT

```
SELECT
  warehouse.warehouse_id,
  warehouse.state AS State,
  COUNT (warehouse.state) AS num_States
FROM
  my-first-project-418418.warehouse_orders.orders AS orders
JOIN
  my-first-project-418418.warehouse_orders.warehouse warehouse
ON
  orders.warehouse_id = warehouse.warehouse_id
GROUP BY
  warehouse.state,
  warehouse.warehouse_id
ORDER BY
  num_States DESC
```



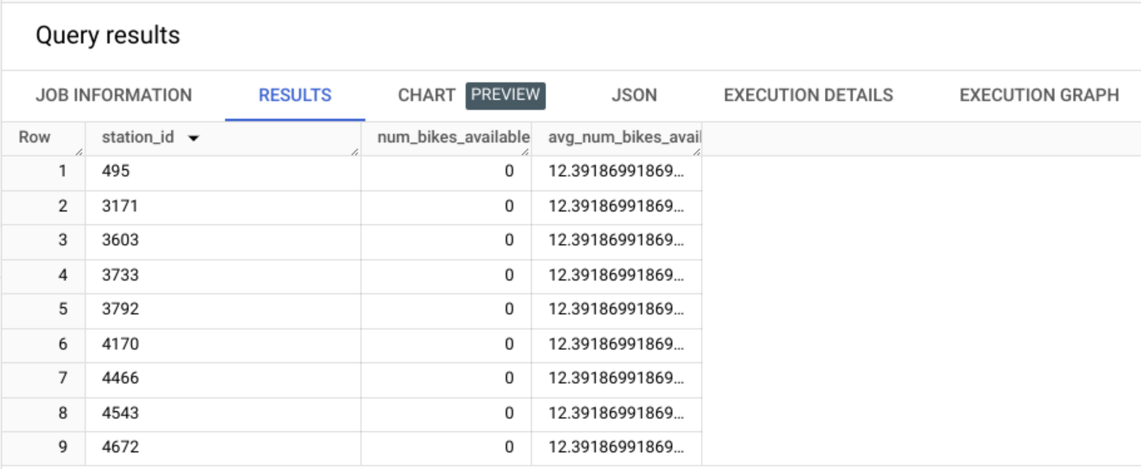
Row	warehouse_id	State	num_States
1	2666	MI	3178
2	8118	MI	3027
3	6509	TN	2403
4	1543	KY	548
5	9080	KY	500
6	4338	TN	343

NESTED QUERIES

Please find the explaining in the Hands-On Folder of Data Analytics. I have created a word file which has the explanation for each of the below three queries.

33. USE A SubQuery In a SELECT statement

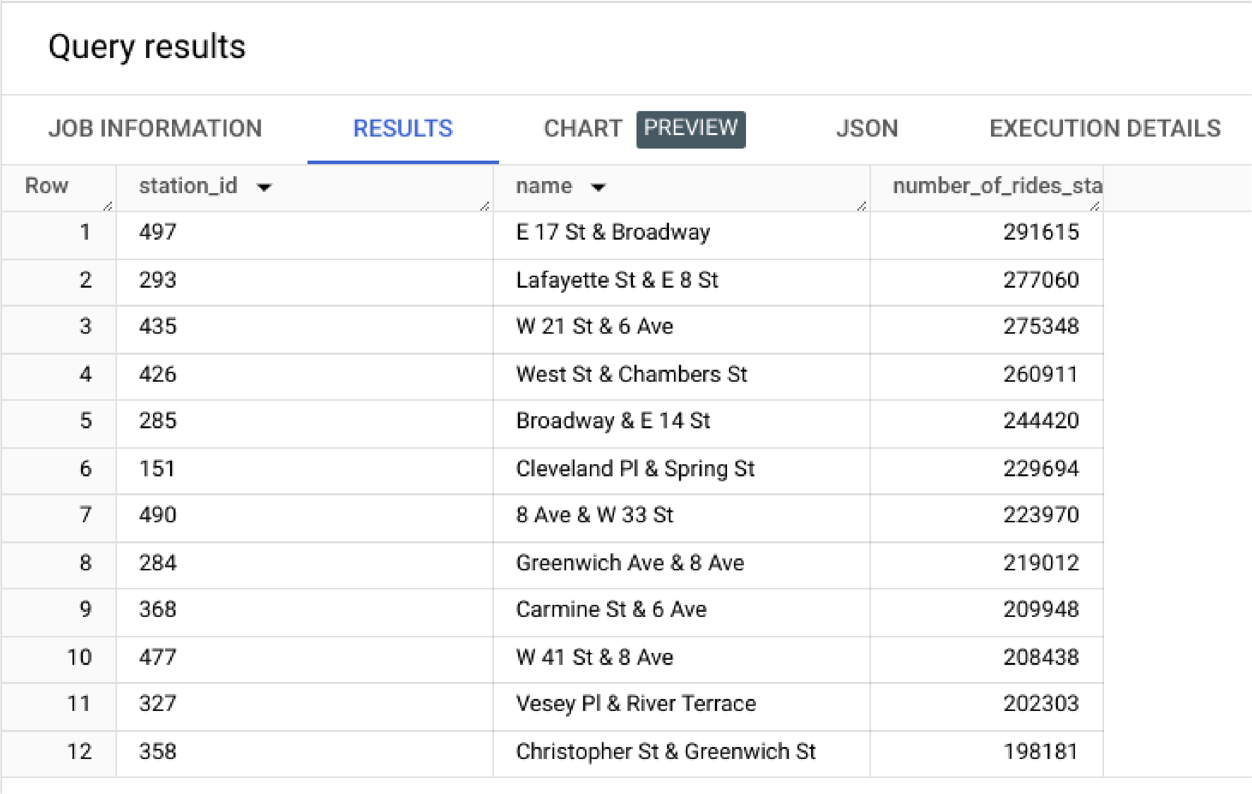
```
SELECT
    station_id,
    num_bikes_available,
    (SELECT
        AVG(num_bikes_available)
    FROM bigquery-public-
data.new_york.citibike_stations) AS avg_num_bikes_available
FROM bigquery-public-data.new_york.citibike_stations;
```



Row	station_id	num_bikes_available	avg_num_bikes_avail
1	495	0	12.39186991869...
2	3171	0	12.39186991869...
3	3603	0	12.39186991869...
4	3733	0	12.39186991869...
5	3792	0	12.39186991869...
6	4170	0	12.39186991869...
7	4466	0	12.39186991869...
8	4543	0	12.39186991869...
9	4672	0	12.39186991869...

34. Use a subquery in a FROM statement

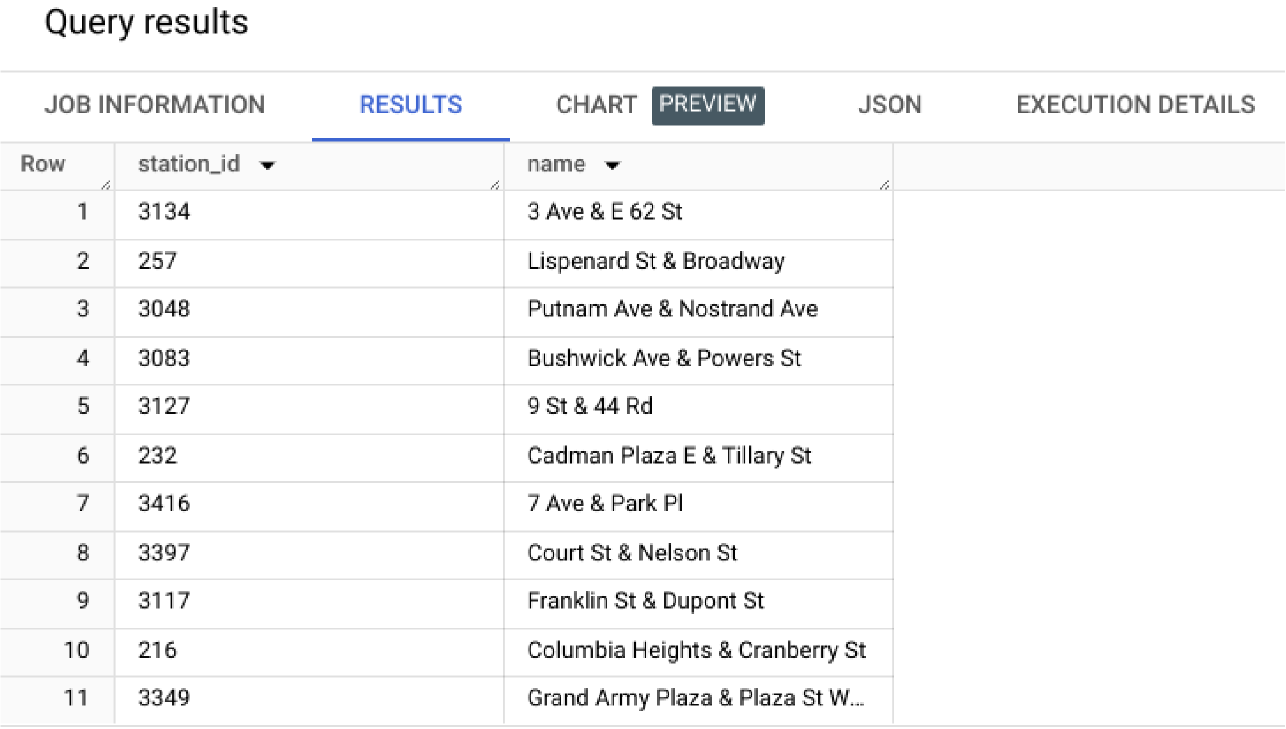
```
SELECT
  station_id,
  name,
  number_of_rides AS number_of_rides_starting_at_station #This column is not
present either in Citibike_trips or Citibike_station so we have created a temporary
table with the name as "station_num_trips" and from that table we are extracting
"number_of_rides".
FROM
  (
    SELECT
      CAST(start_station_id AS STRING) AS start_station_id_str,
      COUNT(*) AS number_of_rides
    FROM
      bigquery-public-data.new_york.citibike_trips
    GROUP BY
      CAST(start_station_id AS STRING)
  ) AS station_num_trips
  INNER JOIN
    bigquery-public-data.new_york.citibike_stations
  ON
    station_num_trips.start_station_id_str = station_id
  ORDER BY
    station_num_trips.number_of_rides DESC
```



Row	station_id	name	number_of_rides_sta
1	497	E 17 St & Broadway	291615
2	293	Lafayette St & E 8 St	277060
3	435	W 21 St & 6 Ave	275348
4	426	West St & Chambers St	260911
5	285	Broadway & E 14 St	244420
6	151	Cleveland Pl & Spring St	229694
7	490	8 Ave & W 33 St	223970
8	284	Greenwich Ave & 8 Ave	219012
9	368	Carmine St & 6 Ave	209948
10	477	W 41 St & 8 Ave	208438
11	327	Vesey Pl & River Terrace	202303
12	358	Christopher St & Greenwich St	198181

35. Use a subquery in a WHERE statement

```
SELECT
    station_id,
    name
FROM
    bigquery-public-data.new_york.citibike_stations
WHERE
    station_id IN
    (
        SELECT
            CAST(start_station_id AS STRING) AS start_station_id_str #**
        FROM
            bigquery-public-data.new_york.citibike_trips
        WHERE
            usertype = 'Subscriber'
    );
```


Query results

Row	station_id	name
1	3134	3 Ave & E 62 St
2	257	Lispenard St & Broadway
3	3048	Putnam Ave & Nostrand Ave
4	3083	Bushwick Ave & Powers St
5	3127	9 St & 44 Rd
6	232	Cadman Plaza E & Tillary St
7	3416	7 Ave & Park Pl
8	3397	Court St & Nelson St
9	3117	Franklin St & Dupont St
10	216	Columbia Heights & Cranberry St
11	3349	Grand Army Plaza & Plaza St W...

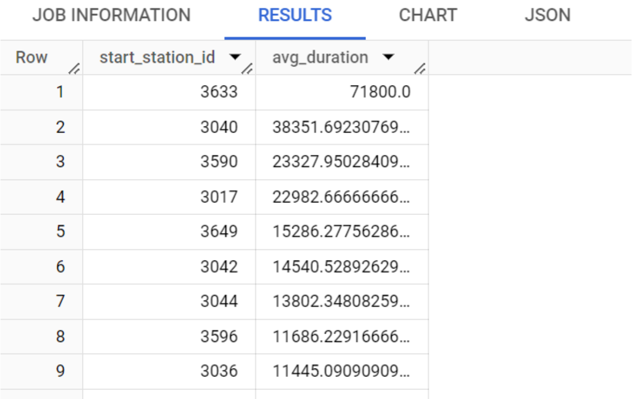
HANDS ON Activity

Data Set used : citibike_trips (Public Data Set)

Scenario : To complete this task, you will create three different subqueries, which will allow you to gather information about the average trip duration by station, compare trip duration by station, and determine the five stations with the longest mean trip durations.

36. Query 1 to calculate "average trip duration by station" (Nested SELECT inside FROM)

```
SELECT
    subquery.start_station_id,
    subquery.avg_duration
FROM
    (
    SELECT
        start_station_id,
        AVG(tripduration) as avg_duration
FROM bigquery-public-data.new_york_citibike.citibike_trips
GROUP BY start_station_id) as subquery
ORDER BY avg_duration DESC;
```

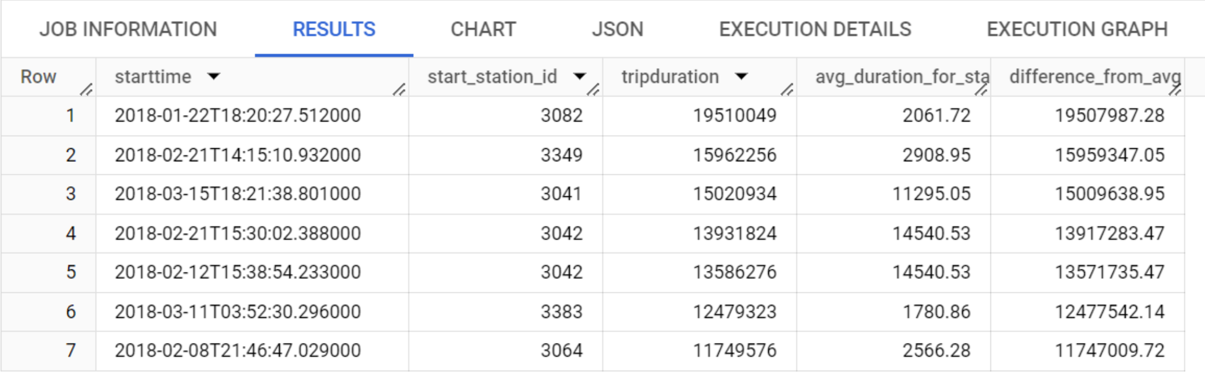


Row	start_station_id	avg_duration
1	3633	71800.0
2	3040	38351.69230769...
3	3590	23327.95028409...
4	3017	22982.66666666...
5	3649	15286.27756286...
6	3042	14540.52892629...
7	3044	13802.34808259...
8	3596	11686.22916666...
9	3036	11445.09090909...

37. **Query 2 compare "trip duration by station"** (Nested SELECT inside SELECT)

```
SELECT
    starttime,
    start_station_id,
    tripduration,
    (
        SELECT ROUND(AVG(tripduration),2) #the nested ones are runned when a row
for 1st three columns are ready
        FROM bigquery-public-data.new_york_citibike.citibike_trips
        WHERE start_station_id = outer_trips.start_station_id
    ) AS avg_duration_for_station,
    ROUND(outer_trips.tripduration - (
        SELECT AVG(tripduration)
        FROM bigquery-public-data.new_york_citibike.citibike_trips
        WHERE start_station_id = outer_trips.start_station_id), 2) AS
difference_from_avg
FROM bigquery-public-data.new_york_citibike.citibike_trips AS outer_trips
ORDER BY difference_from_avg DESC
LIMIT 25;
```

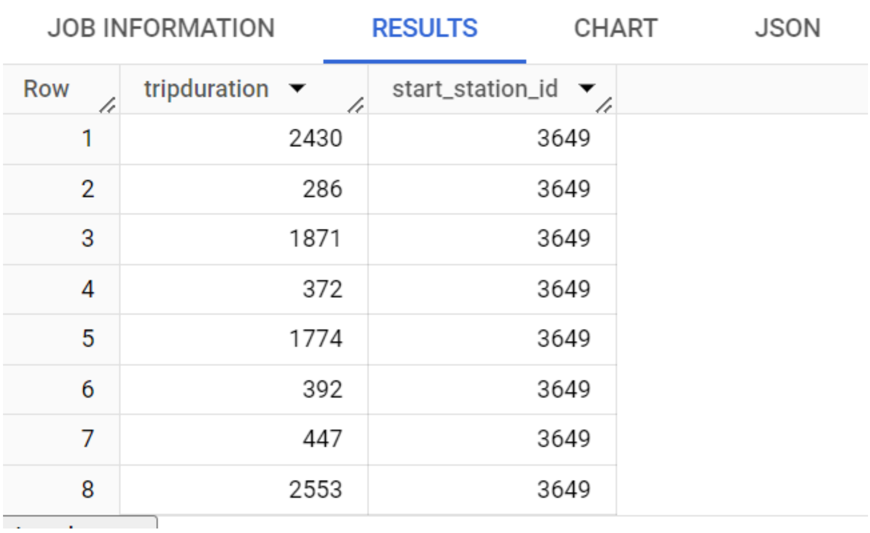
Regarding 1st Subquery : This subquery is used to calculate the average trip duration for trips starting at the same station as the current trip (outer_trips.start_station_id).



Row	starttime	start_station_id	tripduration	avg_duration_for_sta	difference_from_avg
1	2018-01-22T18:20:27.512000	3082	19510049	2061.72	19507987.28
2	2018-02-21T14:15:10.932000	3349	15962256	2908.95	15959347.05
3	2018-03-15T18:21:38.801000	3041	15020934	11295.05	15009638.95
4	2018-02-21T15:30:02.388000	3042	13931824	14540.53	13917283.47
5	2018-02-12T15:38:54.233000	3042	13586276	14540.53	13571735.47
6	2018-03-11T03:52:30.296000	3383	12479323	1780.86	12477542.14
7	2018-02-08T21:46:47.029000	3064	11749576	2566.28	11747009.72

38. **QUERY 3** compose a new query to filter the data to include only the trips from the five stations with the longest mean trip duration.

```
SELECT
    tripduration,
    start_station_id
FROM bigquery-public-data.new_york_citibike.citibike_trips
WHERE start_station_id IN
    (
        SELECT
            start_station_id
        FROM
        (
            SELECT
                start_station_id,
                AVG(tripduration) AS avg_duration
            FROM bigquery-public-data.new_york_citibike.citibike_trips
            GROUP BY start_station_id
        ) AS top_five
        ORDER BY avg_duration DESC
        LIMIT 5
    );
```

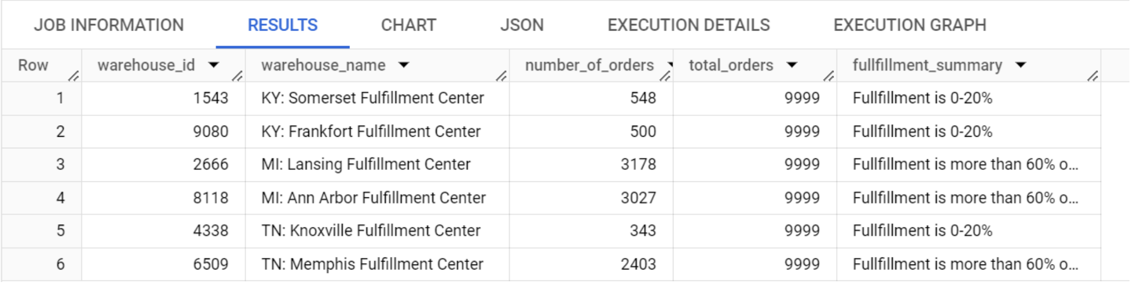


Row	tripduration	start_station_id
1	2430	3649
2	286	3649
3	1871	3649
4	372	3649
5	1774	3649
6	392	3649
7	447	3649
8	2553	3649

39. Using CASE to categorize the data.

The use of CASE is very similar to using IF ELSE or SWITCH CASE in java/python programming.

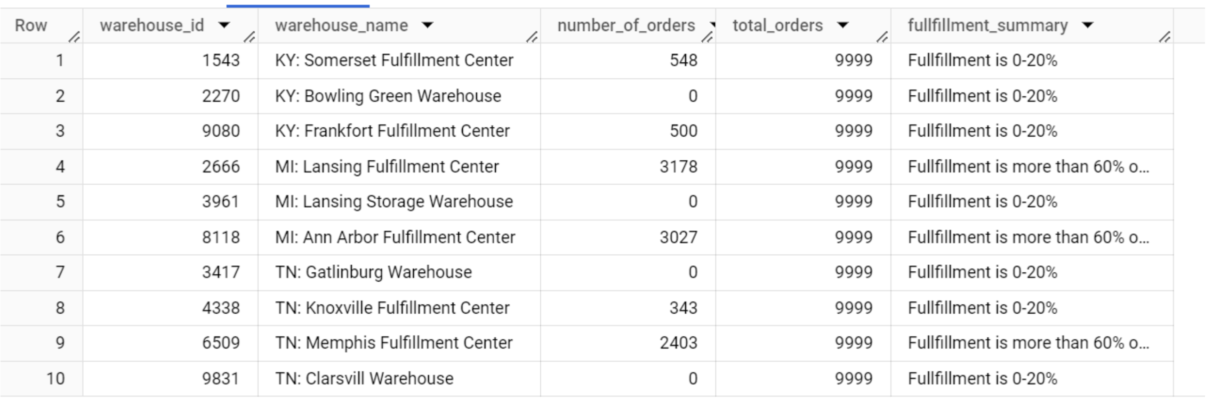
```
SELECT
  warehouse.warehouse_id,
  CONCAT(warehouse.state,': ',warehouse.warehouse_alias) AS warehouse_name,
  COUNT(orders.order_id) AS number_of_orders,
  (SELECT COUNT(*) FROM my-first-project-418418.warehouse_orders.orders AS orders)
AS total_orders,
  CASE
    WHEN COUNT(orders.order_id)/(SELECT COUNT(*) FROM my-first-project-
418418.warehouse_orders.orders AS orders) <= 0.2
    THEN 'Fullfillment is 0-20%'
    WHEN COUNT(orders.order_id)/(SELECT COUNT(*) FROM my-first-project-
418418.warehouse_orders.orders AS orders) > 20
    AND COUNT(orders.order_id)/(SELECT COUNT(*) FROM my-first-project-
418418.warehouse_orders.orders AS orders) <= 60
    THEN 'Fullfillment is 21-60%'
    ELSE 'Fullfillment is more than 60% of orders'
  END AS fullfillment_summary
FROM my-first-project-418418.warehouse_orders.warehouse AS warehouse
LEFT JOIN my-first-project-418418.warehouse_orders.orders AS orders
ON orders.warehouse_id = warehouse.warehouse_id
GROUP BY
  warehouse.warehouse_id,
  warehouse_name
HAVING
  COUNT(orders.order_id) > 0
```



Row	warehouse_id	warehouse_name	number_of_orders	total_orders	fullfillment_summary
1	1543	KY: Somerset Fulfillment Center	548	9999	Fullfillment is 0-20%
2	9080	KY: Frankfort Fulfillment Center	500	9999	Fullfillment is 0-20%
3	2666	MI: Lansing Fulfillment Center	3178	9999	Fullfillment is more than 60% o...
4	8118	MI: Ann Arbor Fulfillment Center	3027	9999	Fullfillment is more than 60% o...
5	4338	TN: Knoxville Fulfillment Center	343	9999	Fullfillment is 0-20%
6	6509	TN: Memphis Fulfillment Center	2403	9999	Fullfillment is more than 60% o...

40. Using CASE to categorize the data without HAVING

```
SELECT
  warehouse.warehouse_id,
  CONCAT(warehouse.state,': ',warehouse.warehouse_alias) AS warehouse_name,
  COUNT(orders.order_id) AS number_of_orders,
  (SELECT COUNT(*) FROM my-first-project-418418.warehouse_orders.orders AS orders)
AS total_orders,
  CASE
    WHEN COUNT(orders.order_id)/(SELECT COUNT(*) FROM my-first-project-
418418.warehouse_orders.orders AS orders) <= 0.2
    THEN 'Fullfillment is 0-20%'
    WHEN COUNT(orders.order_id)/(SELECT COUNT(*) FROM my-first-project-
418418.warehouse_orders.orders AS orders) > 20
    AND COUNT(orders.order_id)/(SELECT COUNT(*) FROM my-first-project-
418418.warehouse_orders.orders AS orders) <= 60
    THEN 'Fullfillment is 21-60%'
    ELSE 'Fullfillment is more than 60% of orders'
  END AS fullfillment_summary
FROM my-first-project-418418.warehouse_orders.warehouse AS warehouse
LEFT JOIN my-first-project-418418.warehouse_orders.orders AS orders
ON orders.warehouse_id = warehouse.warehouse_id
GROUP BY
  warehouse.warehouse_id,
  warehouse_name
#HAVING
#  COUNT(orders.order_id) > 0
```



Row	warehouse_id	warehouse_name	number_of_orders	total_orders	fullfillment_summary
1	1543	KY: Somerset Fulfillment Center	548	9999	Fullfillment is 0-20%
2	2270	KY: Bowling Green Warehouse	0	9999	Fullfillment is 0-20%
3	9080	KY: Frankfort Fulfillment Center	500	9999	Fullfillment is 0-20%
4	2666	MI: Lansing Fulfillment Center	3178	9999	Fullfillment is more than 60% o...
5	3961	MI: Lansing Storage Warehouse	0	9999	Fullfillment is 0-20%
6	8118	MI: Ann Arbor Fulfillment Center	3027	9999	Fullfillment is more than 60% o...
7	3417	TN: Gatlinburg Warehouse	0	9999	Fullfillment is 0-20%
8	4338	TN: Knoxville Fulfillment Center	343	9999	Fullfillment is 0-20%
9	6509	TN: Memphis Fulfillment Center	2403	9999	Fullfillment is more than 60% o...
10	9831	TN: Clarsvill Warehouse	0	9999	Fullfillment is 0-20%

Here the number_of_orders column is blank because there are some warehouse_id numbers which are not present in the orders table.

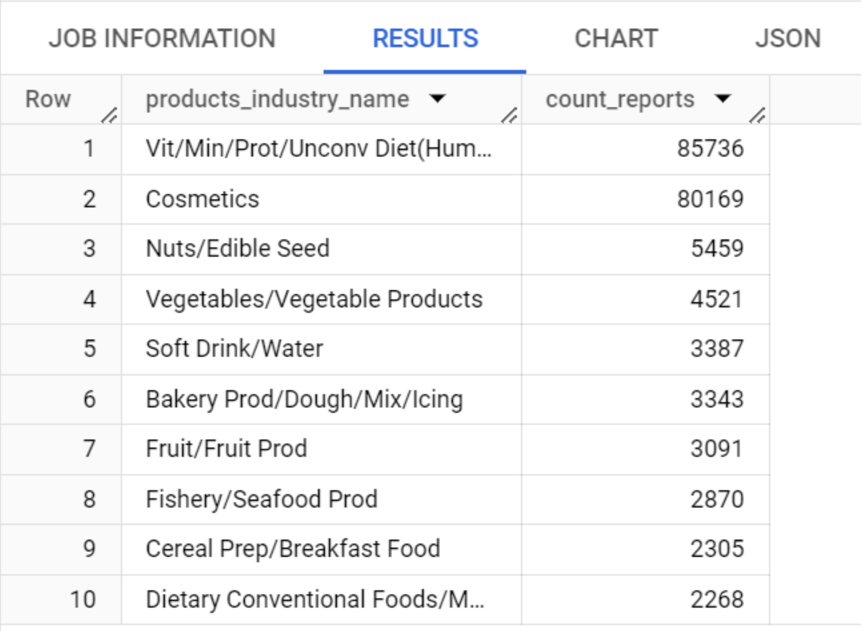
HANDS ON SCENARIO :

In this scenario, you are a junior data analyst for a multinational food and beverage manufacturer. You and your team are responsible for maintaining the safety of a wide array of food products. Because of the overwhelming number of products on the market, you have been asked to prioritize which products need to be reviewed by your stakeholders.

While it's useful to know which food industries receive the most complaints, the more critical aspect to consider is identifying the complaints that lead to severe health consequences, such as hospital visits.

41. Getting the top 10 Products having most Report Counts.

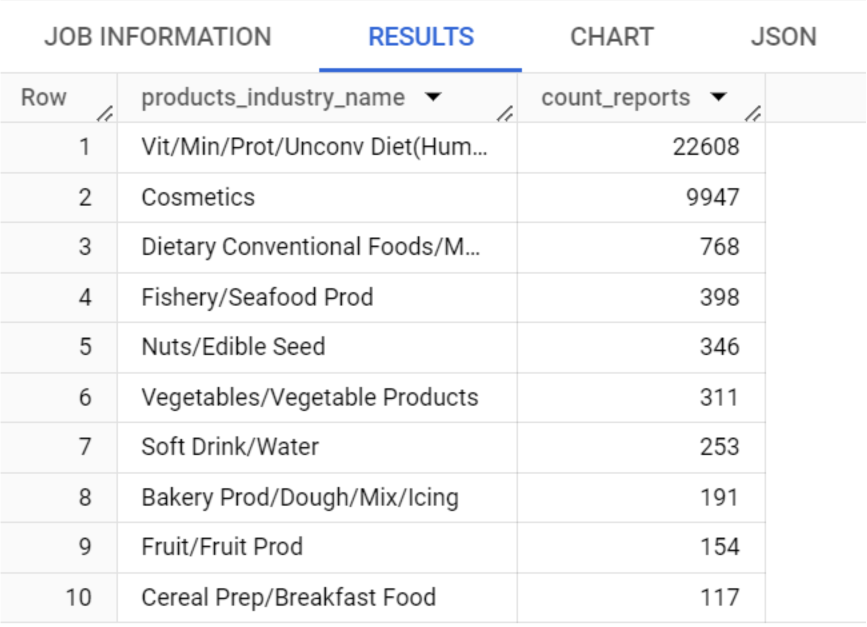
```
SELECT
  products_industry_name,
  COUNT(report_number) AS count_reports
FROM
  `bigquery-public-data.fda_food.food_events`
GROUP BY
  products_industry_name
ORDER BY
  count_reports DESC
LIMIT 10;
```



Row	products_industry_name	count_reports
1	Vit/Min/Prot/Unconv Diet(Hum...	85736
2	Cosmetics	80169
3	Nuts/Edible Seed	5459
4	Vegetables/Vegetable Products	4521
5	Soft Drink/Water	3387
6	Bakery Prod/Dough/Mix/Icing	3343
7	Fruit/Fruit Prod	3091
8	Fishery/Seafood Prod	2870
9	Cereal Prep/Breakfast Food	2305
10	Dietary Conventional Foods/M...	2268

42. Getting the top 10 Products having Most Reports as well as those that have most hospitalizations.

```
SELECT
  products_industry_name,
  COUNT(report_number) AS count_reports
FROM
  bigquery-public-data.fda_food.food_events
WHERE
  products_industry_name IN
  (
    SELECT
      products_industry_name,
      -- COUNT(report_number) AS count_reports
    FROM
      `bigquery-public-data.fda_food.food_events`
    GROUP BY
      products_industry_name
    ORDER BY
      COUNT(report_number) DESC
    LIMIT 10
  ) AND outcomes LIKE '%Hospitalization%'
GROUP BY
  products_industry_name
ORDER BY
  count_reports DESC
```

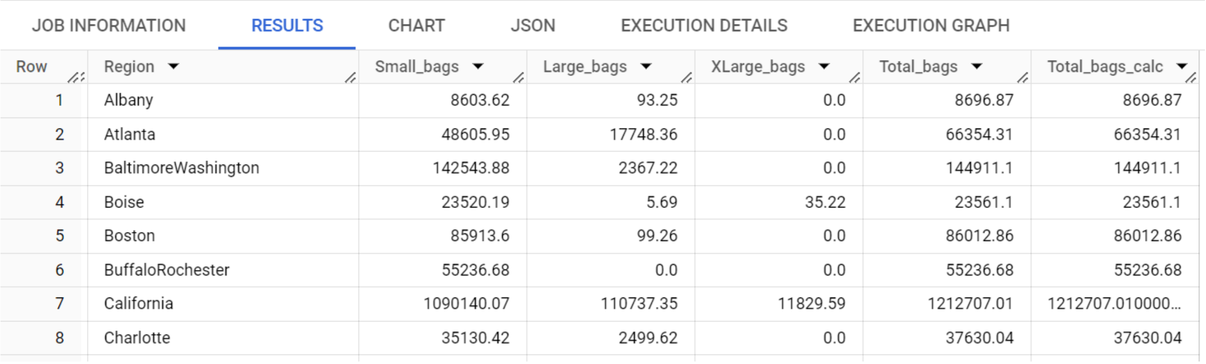


Row	products_industry_name	count_reports
1	Vit/Min/Prot/Unconv Diet(Hum...	22608
2	Cosmetics	9947
3	Dietary Conventional Foods/M...	768
4	Fishery/Seafood Prod	398
5	Nuts/Edible Seed	346
6	Vegetables/Vegetable Products	311
7	Soft Drink/Water	253
8	Bakery Prod/Dough/Mix/Icing	191
9	Fruit/Fruit Prod	154
10	Cereal Prep/Breakfast Food	117

SIMPLE Calculations using SQL

43. Simple Addition

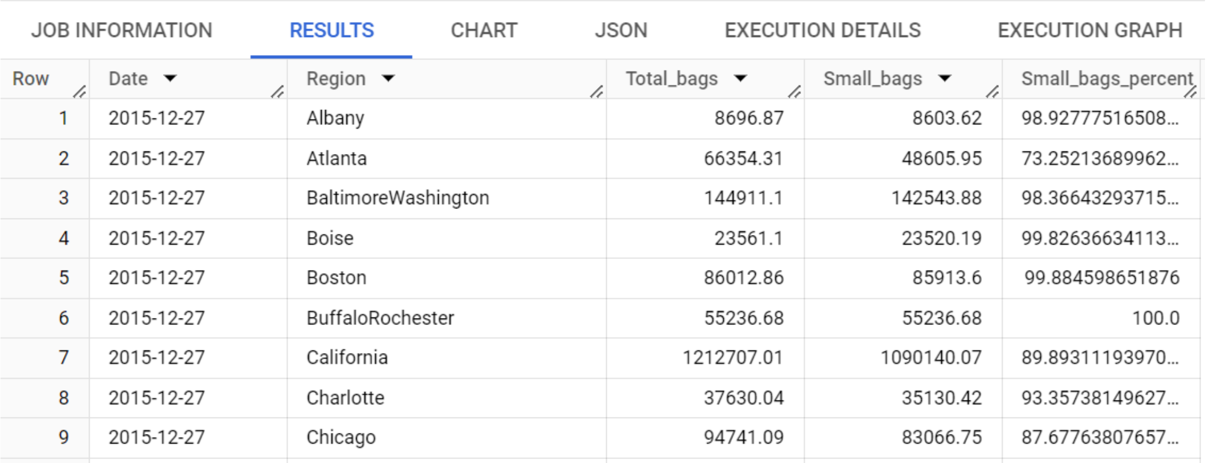
```
SELECT
  Date,
  Region,
  Small_bags,
  Large_bags,
  XLarge_bags,
  Total_bags,
  Small_bags + Large_bags + XLarge_bags AS Total_bags_calc
FROM
  `my-first-project-418418.avocado_data.avocado_prices`
```



Row	Region	Small_bags	Large_bags	XLarge_bags	Total_bags	Total_bags_calc
1	Albany	8603.62	93.25	0.0	8696.87	8696.87
2	Atlanta	48605.95	17748.36	0.0	66354.31	66354.31
3	BaltimoreWashington	142543.88	2367.22	0.0	144911.1	144911.1
4	Boise	23520.19	5.69	35.22	23561.1	23561.1
5	Boston	85913.6	99.26	0.0	86012.86	86012.86
6	BuffaloRochester	55236.68	0.0	0.0	55236.68	55236.68
7	California	1090140.07	110737.35	11829.59	1212707.01	1212707.010000...
8	Charlotte	35130.42	2499.62	0.0	37630.04	37630.04

44. Simple Division to calculate percentage

```
SELECT
  Date,
  Region,
  Total_bags,
  Small_bags,
  (Small_bags / Total_bags)*100 AS Small_bags_percent
FROM
  `my-first-project-418418.avocado_data.avocado_prices`
WHERE
  Total_bags <> 0
```

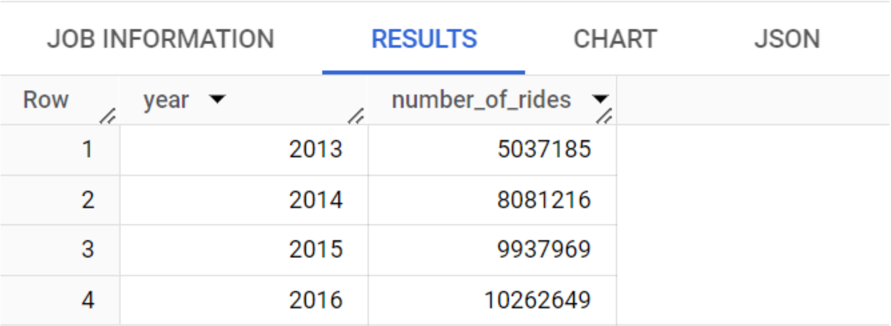


Row	Date	Region	Total_bags	Small_bags	Small_bags_percent
1	2015-12-27	Albany	8696.87	8603.62	98.92777516508...
2	2015-12-27	Atlanta	66354.31	48605.95	73.25213689962...
3	2015-12-27	BaltimoreWashington	144911.1	142543.88	98.36643293715...
4	2015-12-27	Boise	23561.1	23520.19	99.82636634113...
5	2015-12-27	Boston	86012.86	85913.6	99.884598651876
6	2015-12-27	BuffaloRochester	55236.68	55236.68	100.0
7	2015-12-27	California	1212707.01	1090140.07	89.89311193970...
8	2015-12-27	Charlotte	37630.04	35130.42	93.35738149627...
9	2015-12-27	Chicago	94741.09	83066.75	87.67763807657...

45. Use of EXTRACT

In the below query we have used EXTRACT to get the year from the "starttime" column.

```
SELECT
  EXTRACT(YEAR FROM starttime) AS year,
  COUNT(*) AS number_of_rides
FROM
  `bigquery-public-data.new_york.citibike_trips`
GROUP BY
  year
ORDER BY
  year
```

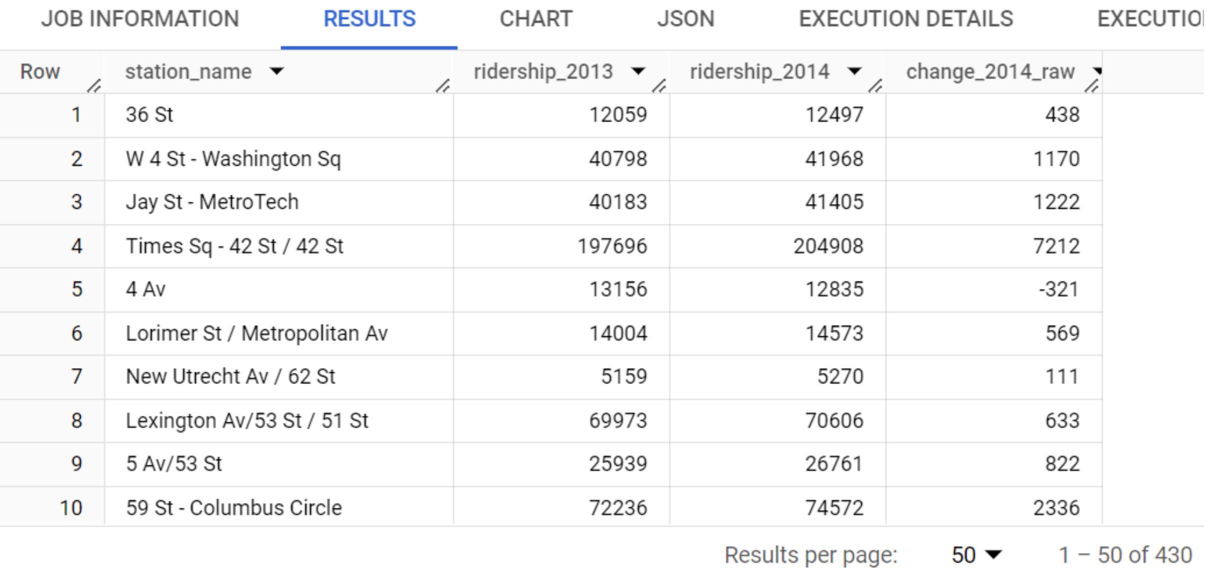


Row	year	number_of_rides
1	2013	5037185
2	2014	8081216
3	2015	9937969
4	2016	10262649

46. HANDSON For basic calculations using SQL.

Subtraction

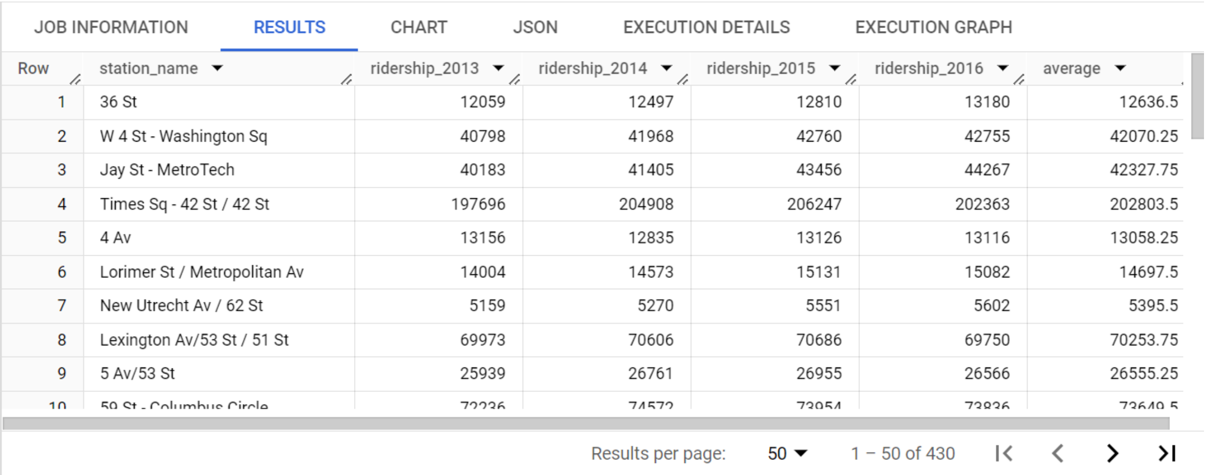
```
SELECT
  station_name,
  ridership_2013,
  ridership_2014,
  ridership_2014 - ridership_2013 AS change_2014_raw
FROM
  `bigquery-public-data.new_york_subway.subway_ridership_2013_present`
```



Row	station_name	ridership_2013	ridership_2014	change_2014_raw
1	36 St	12059	12497	438
2	W 4 St - Washington Sq	40798	41968	1170
3	Jay St - MetroTech	40183	41405	1222
4	Times Sq - 42 St / 42 St	197696	204908	7212
5	4 Av	13156	12835	-321
6	Lorimer St / Metropolitan Av	14004	14573	569
7	New Utrecht Av / 62 St	5159	5270	111
8	Lexington Av/53 St / 51 St	69973	70606	633
9	5 Av/53 St	25939	26761	822
10	59 St - Columbus Circle	72236	74572	2336

47. Calculating AVERAGE

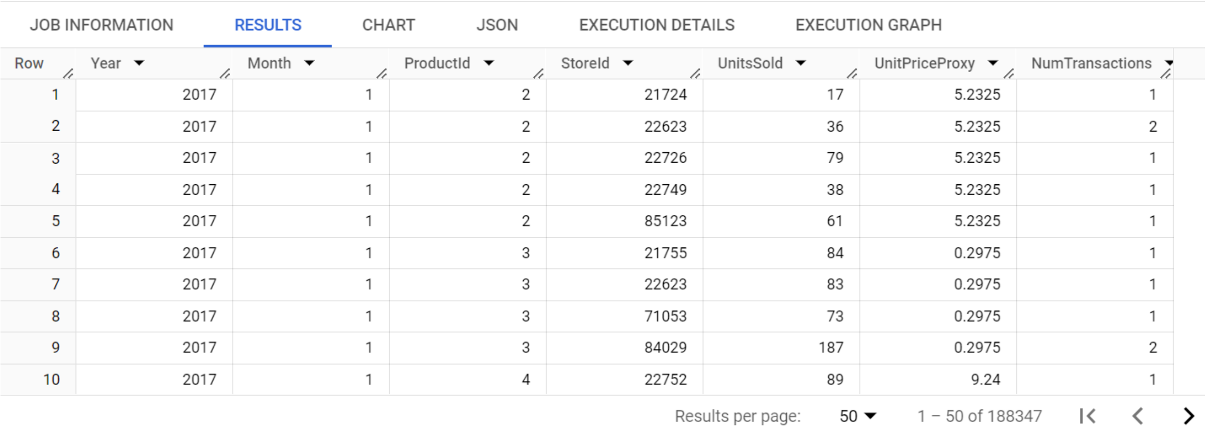
```
SELECT
  station_name,
  ridership_2013,
  ridership_2014,
  ridership_2015,
  ridership_2016,
  (ridership_2013 + ridership_2014 + ridership_2015 + ridership_2016)/4 AS average
FROM
  `bigquery-public-data.new_york_subway.subway_ridership_2013_present`
```



Row	station_name	ridership_2013	ridership_2014	ridership_2015	ridership_2016	average
1	36 St	12059	12497	12810	13180	12636.5
2	W 4 St - Washington Sq	40798	41968	42760	42755	42070.25
3	Jay St - MetroTech	40183	41405	43456	44267	42327.75
4	Times Sq - 42 St / 42 St	197696	204908	206247	202363	202803.5
5	4 Av	13156	12835	13126	13116	13058.25
6	Lorimer St / Metropolitan Av	14004	14573	15131	15082	14697.5
7	New Utrecht Av / 62 St	5159	5270	5551	5602	5395.5
8	Lexington Av/53 St / 51 St	69973	70606	70686	69750	70253.75
9	5 Av/53 St	25939	26761	26955	26566	26555.25
10	59 St - Columbus Circle	72236	74572	73954	73836	73649.5

48. Filtering based on Year, Month, ProductID and StoreID

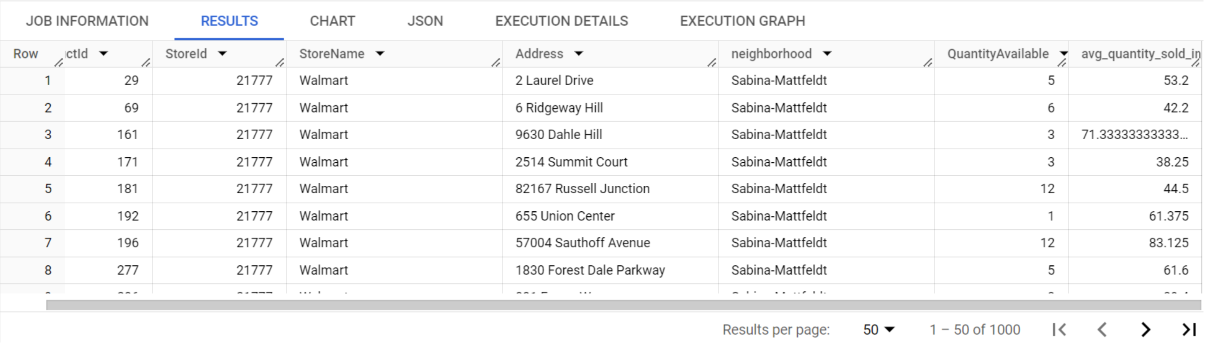
```
SELECT
  EXTRACT(YEAR FROM Date) AS Year
, EXTRACT(MONTH FROM Date) AS Month
, ProductId
, StoreId
, SUM(Quantity) AS UnitsSold
, AVG(UnitPrice) AS UnitPriceProxy
, COUNT(DISTINCT SalesId) AS NumTransactions
FROM
  my-second-project-421106.sales.sales
GROUP BY
  Year, Month, ProductId, StoreId
ORDER BY
  Year, Month, ProductId, StoreId
```



Row	Year	Month	ProductId	StoreId	UnitsSold	UnitPriceProxy	NumTransactions
1	2017	1	2	21724	17	5.2325	1
2	2017	1	2	22623	36	5.2325	2
3	2017	1	2	22726	79	5.2325	1
4	2017	1	2	22749	38	5.2325	1
5	2017	1	2	85123	61	5.2325	1
6	2017	1	3	21755	84	0.2975	1
7	2017	1	3	22623	83	0.2975	1
8	2017	1	3	71053	73	0.2975	1
9	2017	1	3	84029	187	0.2975	2
10	2017	1	4	22752	89	9.24	1

49. NESTED Query

```
with sales_history as (
  SELECT
    EXTRACT(YEAR FROM Date) AS YEAR --time grouping
  , EXTRACT(MONTH FROM Date) AS MONTH --time grouping
  , ProductId --need to know which products are sold
  , StoreID --need to know which stores are selling
  , SUM(quantity) AS UnitsSold --how many (impacts inventory)
  , AVG(UnitPrice) AS UnitPriceProxy --can be interesting
  , COUNT(DISTINCT salesID) AS NumTransactions --unique transactions can be
interesting
  FROM my-second-project-421106.sales.sales
  GROUP BY   YEAR,   MONTH, ProductId, StoreID
)
SELECT
 inventory.*
 , (SELECT AVG(UnitsSold) FROM sales_history
     WHERE inventory.ProductID=sales_history.ProductID
     AND inventory.StoreID=sales_history.StoreID) AS avg_quantity_sold_in_a_month
FROM my-second-project-421106.sales.Inventory AS inventory
```

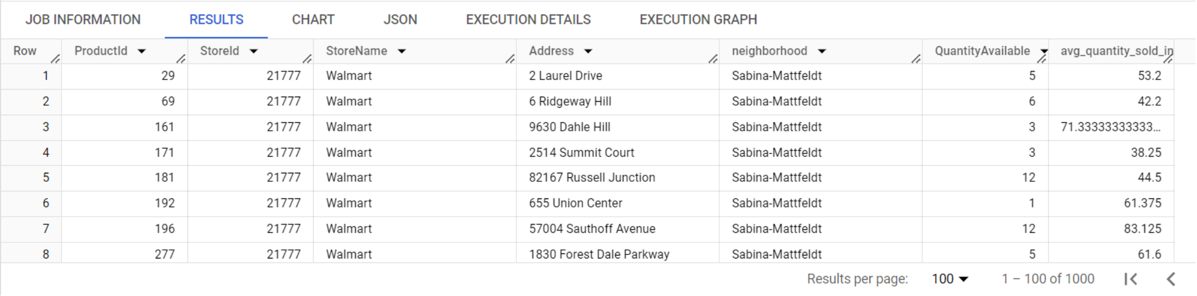


Row	ctId	StoreId	StoreName	Address	neighborhood	QuantityAvailable	avg_quantity_sold_in
1	29	21777	Walmart	2 Laurel Drive	Sabina-Mattfeldt	5	53.2
2	69	21777	Walmart	6 Ridgeway Hill	Sabina-Mattfeldt	6	42.2
3	161	21777	Walmart	9630 Dahle Hill	Sabina-Mattfeldt	3	71.33333333333...
4	171	21777	Walmart	2514 Summit Court	Sabina-Mattfeldt	3	38.25
5	181	21777	Walmart	82167 Russell Junction	Sabina-Mattfeldt	12	44.5
6	192	21777	Walmart	655 Union Center	Sabina-Mattfeldt	1	61.375
7	196	21777	Walmart	57004 Sauthoff Avenue	Sabina-Mattfeldt	12	83.125
8	277	21777	Walmart	1830 Forest Dale Parkway	Sabina-Mattfeldt	5	61.6

50. Revised Query using ChatGPT with LEFT JOIN

```
WITH sales_history AS (
  SELECT
    EXTRACT(YEAR FROM Date) AS YEAR, -- Extracts the year from the date
    EXTRACT(MONTH FROM Date) AS MONTH, -- Extracts the month from the date
    ProductId, -- Product ID
    StoreID, -- Store ID
    SUM(quantity) AS UnitsSold, -- Total units sold
    AVG(UnitPrice) AS UnitPriceProxy, -- Average unit price
    COUNT(DISTINCT salesID) AS NumTransactions -- Number of unique transactions
  FROM my-second-project-421106.sales.sales
  GROUP BY YEAR, MONTH, ProductId, StoreID
)

SELECT
  inventory.*,
  COALESCE(avg_sales.avg_quantity_sold_in_a_month, 0) AS
avg_quantity_sold_in_a_month
FROM
  my-second-project-421106.sales.Inventory AS inventory
LEFT JOIN (
  SELECT
    ProductId,
    StoreID,
    AVG(UnitsSold) AS avg_quantity_sold_in_a_month
  FROM sales_history
  GROUP BY ProductId, StoreID
) AS avg_sales
ON inventory.ProductID = avg_sales.ProductID
AND inventory.StoreID = avg_sales.StoreID;
```



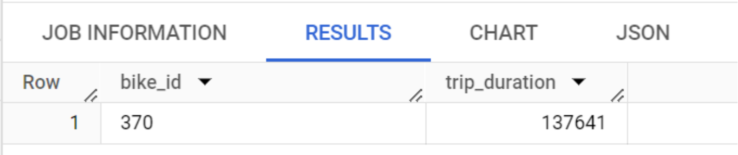
Row	ProductId	StoreId	StoreName	Address	neighborhood	QuantityAvailable	avg_quantity_sold_in
1	29	21777	Walmart	2 Laurel Drive	Sabina-Mattfeldt	5	53.2
2	69	21777	Walmart	6 Ridgeway Hill	Sabina-Mattfeldt	6	42.2
3	161	21777	Walmart	9630 Dahle Hill	Sabina-Mattfeldt	3	71.33333333333...
4	171	21777	Walmart	2514 Summit Court	Sabina-Mattfeldt	3	38.25
5	181	21777	Walmart	82167 Russell Junction	Sabina-Mattfeldt	12	44.5
6	192	21777	Walmart	655 Union Center	Sabina-Mattfeldt	1	61.375
7	196	21777	Walmart	57004 Sauthoff Avenue	Sabina-Mattfeldt	12	83.125
8	277	21777	Walmart	1830 Forest Dale Parkway	Sabina-Mattfeldt	5	61.6

51 . Hands-On to find the bikeId having the highest trip duration (Using TEMP Table).

```
WITH longest_bike_duration AS (
  SELECT
    bike_id,
    SUM(duration_minutes) AS trip_duration
  FROM
    `bigquery-public-data.austin_bikeshare.bikeshare_trips`
  GROUP BY
    bike_id
  ORDER BY
    trip_duration DESC  LIMIT 1
)

## Finding out which bikeid has the highest tripduration

SELECT * FROM longest_bike_duration
```



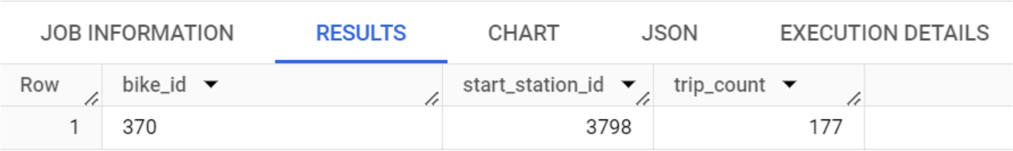
Row	bike_id	trip_duration
1	370	137641

52. SQL query to find the name of the station where that bike can most likely be found, so they ask you to determine which bike is used most often.

```
WITH longest_bike_duration AS (
  SELECT
    bike_id,
    SUM(duration_minutes) AS trip_duration
  FROM
    `bigquery-public-data.austin_bikeshare.bikeshare_trips`
  GROUP BY
    bike_id
  ORDER BY
    trip_duration DESC  LIMIT 1
)

## Finding out which bikeid has the highest tripduration

SELECT
  trips.bike_id,
  trips.start_station_id,
  COUNT(*) AS trip_count
FROM longest_bike_duration AS longest
INNER JOIN
  `bigquery-public-data.austin_bikeshare.bikeshare_trips` AS trips
ON trips.bike_id = longest.bike_id
GROUP BY start_station_id, trips.bike_id
ORDER BY trip_count DESC
LIMIT 1
```

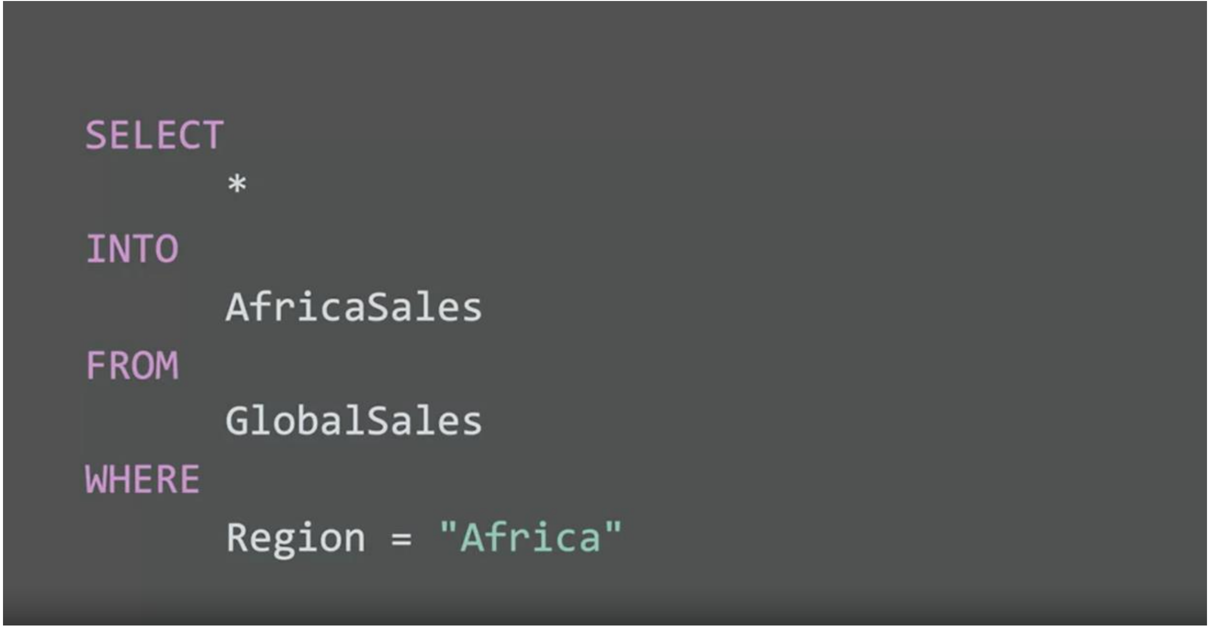


Row	bike_id	start_station_id	trip_count
1	370	3798	177

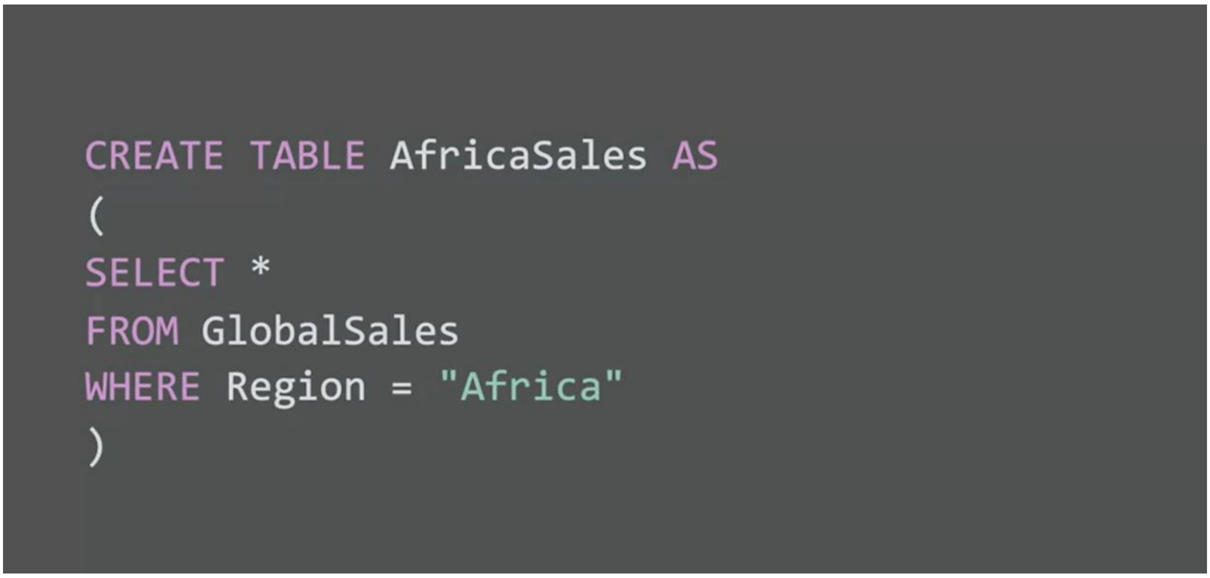
All the below queries actually create a temporary tables into the backend but the WITH ... AS Command doesn’t actually create a temporary table In the back but mimics the temporary table.

53. SELECT INTO

(BigQuery currently doesn’t recognize the SELECT INTO command currently below is the example of how the SELECT INTO might look for other RDBMS system)

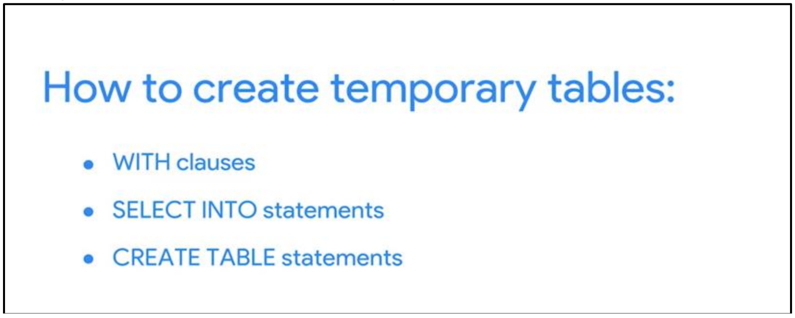


```
SELECT
       *
INTO
       AfricaSales
FROM
       GlobalSales
WHERE
       Region = "Africa"
```



```
CREATE TABLE AfricaSales AS
(
SELECT *
FROM GlobalSales
WHERE Region = "Africa"
)
```

3 Ways to create a Temporary Table



How to create temporary tables:

- WITH clauses
- SELECT INTO statements
- CREATE TABLE statements