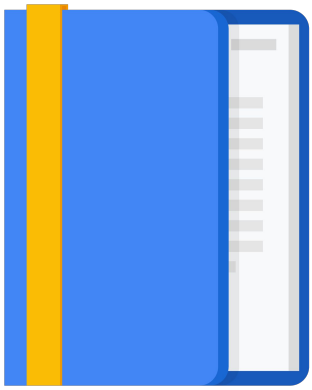
Advanced BigQuery Functionality

Agenda

- Analytic Window Functions
- Using With Clauses
- GIS Functions
- Performance Considerations



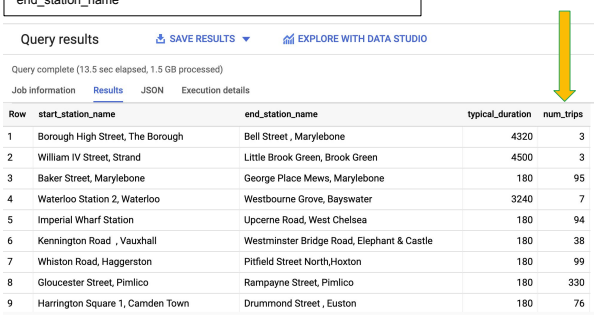
Google Cloud

Use analytic window functions for advanced analysis

- Standard aggregations
- Navigation functions
- Ranking and numbering functions

Example: The COUNT function (self-explanatory)

```
SELECT
  start_station_name,
  end_station_name,
  APPROX_QUANTILES(duration, 10)[OFFSET(5)] AS typical_duration,
  COUNT(*) AS num_trips
FROM
  `bigquery-public-data`.london_bicycles.cycle_hire
GROUP BY
  start_station_name,
  end_station_name
```



Query results

SAVE RESULTS

EXPLORE WITH DATA STUDIO

Query complete (13.5 sec elapsed, 1.5 GB processed)

Job information | Results | JSON | Execution details

Row	start_station_name	end_station_name	typical_duration	num_trips
1	Borough High Street, The Borough	Bell Street , Marylebone	4320	3
2	William IV Street, Strand	Little Brook Green, Brook Green	4500	3
3	Baker Street, Marylebone	George Place Mews, Marylebone	180	95
4	Waterloo Station 2, Waterloo	Westbourne Grove, Bayswater	3240	7
5	Imperial Wharf Station	Upcerne Road, West Chelsea	180	94
6	Kennington Road , Vauxhall	Westminster Bridge Road, Elephant & Castle	180	38
7	Whiston Road, Haggerston	Pitfield Street North,Hoxton	180	99
8	Gloucester Street, Pimlico	Rampayne Street, Pimlico	180	330
9	Harrington Square 1, Camden Town	Drummond Street , Euston	180	76

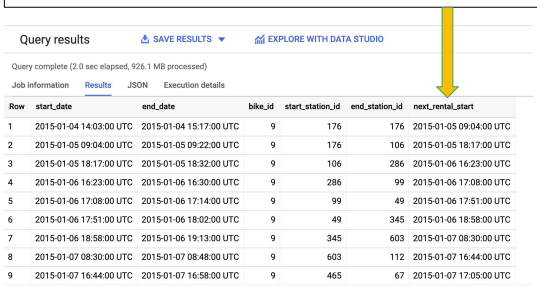
Some other “standard” aggregation functions

- SUM
- AVG
- MIN
- MAX
- BIT_AND
- BIT_OR
- BIT_XOR
- COUNTIF
- LOGICAL_OR
- LOGICAL_AND

https://cloud.google.com/bigquery/docs/reference/standard-sql/aggregate_functions

Example: The LEAD function returns the value of a row n rows ahead of the current row

```
SELECT
  start_date,
  end_date,
  bike_id,
  start_station_id,
  end_station_id,
  LEAD(start_date, 1) OVER(PARTITION BY bike_id ORDER BY start_date ASC ) AS next_rental_start
FROM
  `bigquery-public-data`.london_bicycles.cycle_hire
WHERE
  bike_id = 9
```



Query results SAVE RESULTS EXPLORE WITH DATA STUDIO

Query complete (2.0 sec elapsed, 926.1 MB processed)

Job information | Results | JSON | Execution details

Row	start_date	end_date	bike_id	start_station_id	end_station_id	next_rental_start
1	2015-01-04 14:03:00 UTC	2015-01-04 15:17:00 UTC	9	176	176	2015-01-05 09:04:00 UTC
2	2015-01-05 09:04:00 UTC	2015-01-05 09:22:00 UTC	9	176	106	2015-01-05 18:17:00 UTC
3	2015-01-05 18:17:00 UTC	2015-01-05 18:32:00 UTC	9	106	286	2015-01-06 16:23:00 UTC
4	2015-01-06 16:23:00 UTC	2015-01-06 16:30:00 UTC	9	286	99	2015-01-06 17:08:00 UTC
5	2015-01-06 17:08:00 UTC	2015-01-06 17:14:00 UTC	9	99	49	2015-01-06 17:51:00 UTC
6	2015-01-06 17:51:00 UTC	2015-01-06 18:02:00 UTC	9	49	345	2015-01-06 18:58:00 UTC
7	2015-01-06 18:58:00 UTC	2015-01-06 19:13:00 UTC	9	345	603	2015-01-07 08:30:00 UTC
8	2015-01-07 08:30:00 UTC	2015-01-07 08:48:00 UTC	9	603	112	2015-01-07 16:44:00 UTC
9	2015-01-07 16:44:00 UTC	2015-01-07 16:58:00 UTC	9	465	67	2015-01-07 17:05:00 UTC

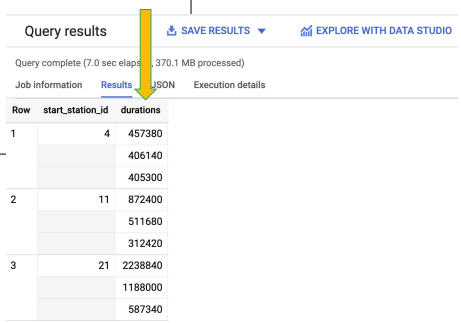
Some other navigation functions

- NTH_VALUE
- LAG
- FIRST_VALUE
- LAST_VALUE

https://cloud.google.com/bigquery/docs/reference/standard-sql/navigation_functions

Example: The RANK function returns the integer rank of a value in a group of values

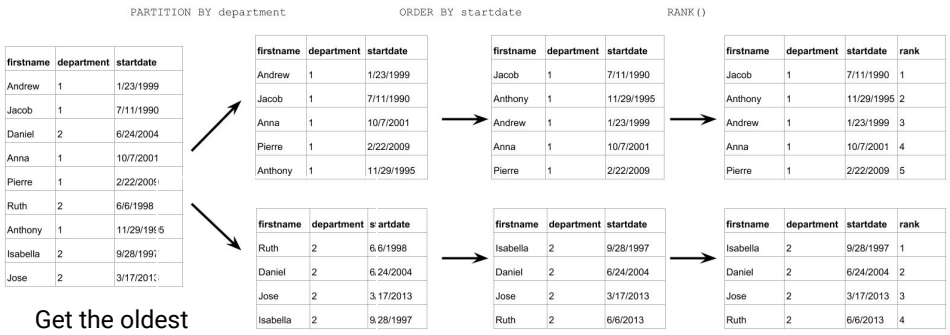
```
WITH
  longest_trips AS (
  SELECT
    start_station_id,
    duration,
    RANK() OVER(PARTITION BY start_station_id ORDER BY duration DESC) AS nth_longest
  FROM
    `bigquery-public-data`.london_bicycles.cycle_hire )
SELECT
  start_station_id,
  ARRAY_AGG(duration
  ORDER BY
    nth_longest
  LIMIT
    3) AS durations
FROM
  longest_trips
GROUP BY
  start_station_id
```



Row	start_station_id	durations
1	4	457380
		406140
		405300
2	11	872400
		511680
		312420
3	21	2238840
		1188000
		587340

Google Cloud

Example: RANK() function for aggregating over groups of rows



Get the oldest ranking employee ***by each department***

Google Cloud

Get the oldest ranking employee by each department

https://cloud.google.com/bigquery/docs/reference/standard-sql/functions-and-operators#supported-functions
RANK( )

In databases, an analytic function is a function that computes aggregate values over a group of rows. Unlike aggregate functions, which return a single aggregate value for a group of rows, analytic functions return a single value for each row by computing the function over a group of input rows.

Analytic functions are a powerful mechanism for succinctly representing complex analytic operations, and they enable efficient evaluations that otherwise would involve expensive self-JOINs or computation outside the SQL query.

Analytic functions are also called "(analytic) window functions" in the SQL standard and some commercial databases. This is because an analytic function is evaluated over a group of rows, referred to as a window or window frame. In some other databases, they may be referred to as Online Analytical Processing (OLAP) functions.

Example: RANK() function for aggregating over groups of rows

```
SELECT firstname, department, startdate,
  RANK() OVER ( PARTITION BY department ORDER BY startdate )
AS rank
FROM Employees;
```

Google Cloud

Option to do demo with IRS dataset:

```
#standardSQL
# Largest employer per U.S. state per 2015 filing
WITH employer_per_state AS (
SELECT
  ein,
  name,
  state,
  noemplyeesw3cnt AS number_of_employees,
  RANK() OVER (PARTITION BY state ORDER BY noemplyeesw3cnt DESC ) AS
rank
FROM
  `bigquery-public-data.irs_990.irs_990_2015`
JOIN
  `bigquery-public-data.irs_990.irs_990_ein`
USING(ein)
GROUP BY 1,2,3,4 #remove duplicates
)

# Get the top employer per state and order highest to lowest states
SELECT *
FROM employer_per_state
WHERE rank = 1
```

```
ORDER BY number_of_employees DESC;
```

Some other ranking and numbering functions

- CUME_DIST
- DENSE_RANK
- ROW_NUMBER
- PERCENT_RANK

https://cloud.google.com/bigquery/docs/reference/standard-sql/numbering_functions

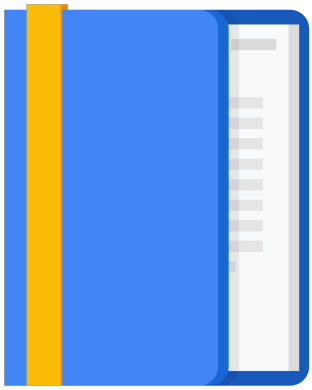
Agenda

Analytic Window Functions

Using With Clauses

GIS Functions

Performance Considerations



Google Cloud

Use WITH clauses and subqueries to modularize

```
WITH

  # 2015 filings joined with organization details
  irs_990_2015_ein AS (
  SELECT *
  FROM
    `bigquery-public-data.irs_990.irs_990_2015`
  JOIN
    `bigquery-public-data.irs_990.irs_990_ein` USING (ein)
    ),

  # duplicate EINs in organization details
  duplicates AS (
  SELECT
    ein AS ein,
    COUNT(ein) AS ein_count
  FROM
    irs_990_2015_ein
  GROUP BY
    ein
  HAVING
    ein_count > 1
    )

# return results to store in a permanent table
SELECT
  irs_990.ein AS ein,
  irs_990.name AS name,
  irs_990.noemployeesw3cnt AS num_employees,
  irs_990.grsrcptspublicuse AS gross_receipts
  # more fields commited for brevity
FROM irs_990_2015_ein AS irs_990
LEFT JOIN duplicates
ON
  irs_990.ein=duplicates.ein
WHERE
  # filter out duplicate records
  duplicates.ein IS NULL
```

- WITH is simply a named subquery (or Common Table Expression)
- Acts as a temporary table
- Breaks up complex queries
- Chain together multiple subqueries in a single WITH
- You can reference other subqueries in future subqueries

Google Cloud

```
#standardSQL
#CTEs
WITH

  # 2015 filings joined with organization details
  irs_990_2015_ein AS (
  SELECT *
  FROM
    `bigquery-public-data.irs_990.irs_990_2015`
  JOIN
    `bigquery-public-data.irs_990.irs_990_ein` USING (ein)
    ),

  # duplicate EINs in organization details
  duplicates AS (
  SELECT
    ein AS ein,
    COUNT(ein) AS ein_count
  FROM
    irs_990_2015_ein
  GROUP BY
    ein
  HAVING
    ein_count > 1
```

```
    )

# return results to store in a permanent table
SELECT
  irs_990.ein AS ein,
  irs_990.name AS name,
  irs_990.noemplyeesw3cnt AS num_employees,
  irs_990.grsrcptspublicuse AS gross_receipts
  # more fields ommited for brevity
FROM irs_990_2015_ein AS irs_990
LEFT JOIN duplicates
ON
  irs_990.ein=duplicates.ein
WHERE
  # filter out duplicate records
  duplicates.ein IS NULL

  LIMIT 10
```

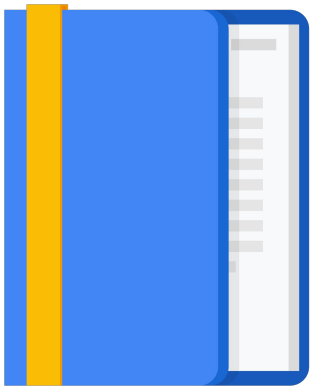
Agenda

Analytic Window Functions

Using With Clauses

GIS Functions

Performance Considerations



Google Cloud

BigQuery has built-in GIS functionality

- Example: Can we find the zip codes best served by the New York Citibike system by looking for the number of stations within 1 km of each zip code that have at least 30 bikes?

```
SELECT
  z.zip_code,
  COUNT(*) AS num_stations
FROM
  `bigquery-public-data.new_york_citibike.citibike_stations` AS s,
  `bigquery-public-data.geo_us_boundaries.zip_codes` AS z
WHERE
  ST_DWithin(z.zcta_geom,
    ST_GeogPoint(s.longitude, s.latitude),
    1000) -- 1km
  AND num_bikes_available > 30
GROUP BY
  z.zip_code
ORDER BY
  num_stations DESC
```

Query results SAVE RESULTS EXPLORE WITH DATA STUDIO

Query complete (1.1 sec elapsed, 128.3 MB processed)

Job information | Results | JSON | Execution details

Row	zip_code	num_stations
1	10003	21
2	10002	19
3	10026	17
4	10012	16
5	10029	16

Use ST_DWITHIN to check if two locations objects are within some distance

- ST_DWithin(geography_1, geography_2, distance)

in meters

- Can lessen resolution of geo coordinates with ST_SnapToGrid
  - ST_SnapToGrid(pt, 0.01) rounds off the latitude and longitude of pt to the second decimal place

```
SELECT
  z.zip_code,
  COUNT(*) AS num_stations
FROM
  `bigquery-public-data.new_york_citibike.citibike_stations` AS s,
  `bigquery-public-data.geo_us_boundaries.zip_codes` AS z
WHERE
  ST_DWithin(z.zip_code_geom,
    ST_GeogPoint(s.longitude, s.latitude),
    1000) -- 1km
  AND num_bikes_available > 30
GROUP BY
  z.zip_code
ORDER BY
  num_stations DESC
```

Google Cloud

Represent longitude and latitude points as Well Known Text (WKT) using the function ST_GeogPoint

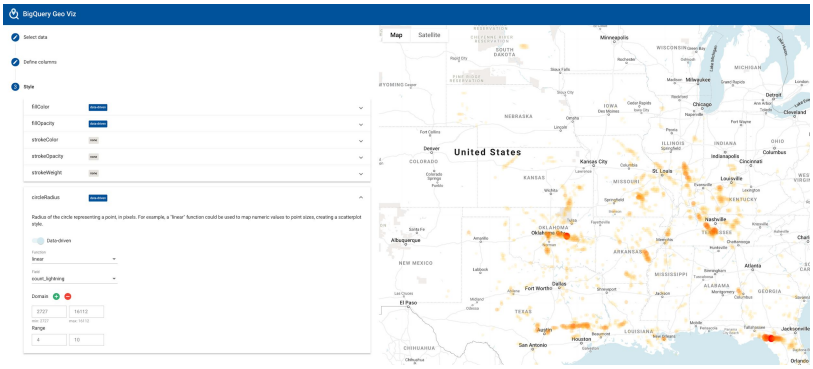
- Queries in BigQuery are much more efficient if geographic data is stored as geography types rather than as primitives

```
SELECT
  z.zip_code,
  COUNT(*) AS num_stations
FROM
  `bigquery-public-data.new_york_citibike.citibike_stations` AS s,
  `bigquery-public-data.geo_us_boundaries.zip_codes` AS z
WHERE
  ST_DWithin(z.zcta_geom,
    ST_GeogPoint(s.longitude, s.latitude),
    1000) -- 1km
  AND num_bikes_available > 30
GROUP BY
  z.zip_code
ORDER BY
  num_stations DESC
```

If your data is stored in JSON, use **ST_GeogFromGeoJSON**

```
SELECT ST_GeogFromGeoJSON('{"type": "Point", "coordinates": [-73.967416,40.756014]}')
```

Visualization with BigQuery Geo Viz



Google Cloud

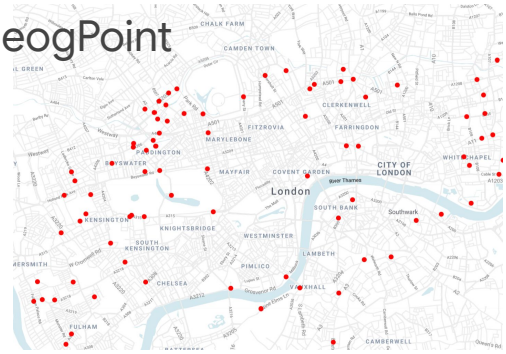
Demo:
https://github.com/GoogleCloudPlatform/training-data-analyst/tree/master/courses/data-engineering/demos

Represent points with ST_GeogPoint

- Represented as WKT (Well Known Text)

```
SELECT
  id,
  longitude,
  latitude,
  ST_GeogPoint(longitude, latitude) AS location
FROM
  `bigquery-public-data.london_bicycles.cycle_stations`
LIMIT
  100
```

id	longitude	latitude	location
171	-0.186753859	51.4916156	POINT(-0.186753859 51.4916156)
165	-0.183716959	51.517932	POINT(-0.183716959 51.517932)
261	-0.191351186	51.5134891	POINT(-0.191351186 51.5134891)
131	-0.136792671	51.53300545	POINT(-0.136792671 51.53300545)
467	-0.030556	51.523538	POINT(-0.030556 51.523538)
43	-0.157183945	51.52026	POINT(-0.157183945 51.52026)
212	-0.199004026	51.50658458	POINT(-0.199004026 51.50658458)
517	-0.033085	51.532513	POINT(-0.033085 51.532513)
704	-0.202802098	51.45682071	POINT(-0.202802098 51.45682071)
721	-0.026262677	51.53603947	POINT(-0.026262677 51.53603947)

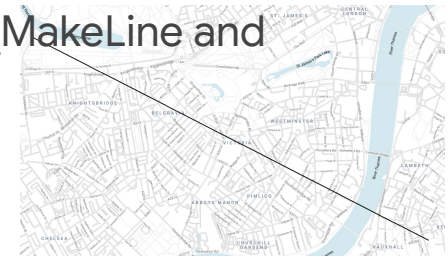


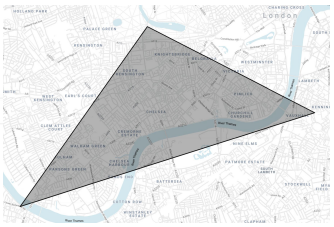
Google Cloud

Represent regions with ST_MakeLine and ST_MakePolygon

```
WITH stations AS (
SELECT
  (SELECT ST_GeogPoint(longitude, latitude) FROM `bigquery-public-da
      AS loc300,
  (SELECT ST_GeogPoint(longitude, latitude) FROM `bigquery-public-da
      AS loc302,
  (SELECT ST_GeogPoint(longitude, latitude) FROM `bigquery-public-da
      AS loc305
)
SELECT
  ST_MakeLine(loc300, loc305) AS seg1,
  ST_MakePolygon(ST_MakeLine([loc300, loc305, loc302])) AS poly
FROM
  stations
```

seg1	poly
LINESTRING(-0.17306032 51.505014, -0.115853961 51.48677988)	POLYGON((-0.216573 51.466907, -0.115853961 51.48677988, -0.17306032 ...





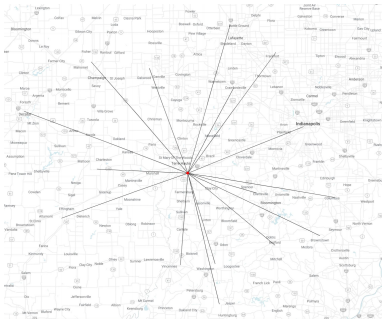
Google Cloud

Are Locations Within Some Distance?

- ST_DWithin

```
SELECT
  m.name AS city,
  m.int_point AS city_coords,
  ST_MakeLine(
    n.int_point,
    m.int_point) AS segs
FROM
  `bigquery-public-data.geo_us_boundaries.us_msa` AS n,
  `bigquery-public-data.geo_us_boundaries.us_msa` AS m
WHERE
  n.name='Terre Haute, IN'
  AND ST_DWithin(
    n.int_point,
    m.int_point,
    1.5e5) --150km
```

city	city_coords	segs
Crawfordsville, IN	POINT(-86.8927145 40.0402962)	LINESTRING(-87.3470958 39.392389, -86.892714...
Decatur, IL	POINT(-88.9615288 39.8602372)	LINESTRING(-87.3470958 39.392389, -88.961528...
Washington, IN	POINT(-87.076944 38.696089)	LINESTRING(-87.3470958 39.392389, -87.076944 ...
Indianapolis-Carmel-Anderson, IN	POINT(-86.2045408 39.7449323)	LINESTRING(-87.3470958 39.392389, -86.204540...
Lafayette-West Lafayette, IN	POINT(-86.9304747 40.5147171)	LINESTRING(-87.3470958 39.392389, -86.930474...
Bloomington, IN	POINT(-86.6717544 39.2417362)	LINESTRING(-87.3470958 39.392389, -86.671754...
Seymour, IN	POINT(-86.0425161 38.9119571)	LINESTRING(-87.3470958 39.392389, -86.042516...
Frankfort, IN	POINT(-86.4775665 40.305944)	LINESTRING(-87.3470958 39.392389, -86.477566...
Charleston-Mattoon, IL	POINT(-88.2422121 39.4231628)	LINESTRING(-87.3470958 39.392389, -88.242212...
Jasper, IN	POINT(-87.0359535 38.3841599)	LINESTRING(-87.3470958 39.392389, -87.035953...



Google Cloud

Other predicate functions

- Do locations intersect?
  - ST_Intersects
- Is one geometry contained inside another?
  - ST_Contains
- Does a geography engulf another?
  - ST_CoveredBy

```
WITH geos AS (
  SELECT
    (SELECT state_geom FROM `bigquery-public-data.geo_us_boundaries.us_states`
       WHERE state_name='Massachusetts') AS ma_poly,
    (SELECT msa_geom FROM `bigquery-public-data.geo_us_boundaries.us_msa`
       WHERE name='Boston-Cambridge-Newton, MA-NH') AS boston_poly,
    (SELECT msa_geom FROM `bigquery-public-data.geo_us_boundaries.us_msa`
       WHERE name='Seattle-Tacoma-Bellevue, WA') AS seattle_poly
)
SELECT
  ST_Intersects(boston_poly, ma_poly) boston_in_ma,
  ST_Intersects(seattle_poly, ma_poly) seattle_in_ma
FROM
  geos
```

boston_in_ma	seattle_in_ma
true	false

Google Cloud

Distance between points

- ST_Distance
- Can lessen resolution of geo coordinates with ST_SnapToGrid
  - ST_SnapToGrid(pt, 0.01) rounds off the latitude and longitude of pt to the second decimal place.

```
WITH geos AS (
  SELECT
    (SELECT int_point FROM `bigquery-public-data.geo_us_boundaries.us_msa`
     WHERE name='Ann Arbor, MI') AS ann_arbor_loc,
    (SELECT int_point FROM `bigquery-public-data.geo_us_boundaries.us_msa`
     WHERE name='Columbus, OH') AS columbus_loc
)
SELECT
  ST_Distance(ann_arbor_loc, columbus_loc)/1000 AS dist_in_km
FROM
  geos
```

dist_in_km
267.63049397704333

```
SELECT
  name,
  int_point as point_full,
  ST_SnapToGrid(int_point, .01) as point_2decs
FROM
  `bigquery-public-data.geo_us_boundaries.us_msa`
WHERE
  name='Ann Arbor, MI'
```

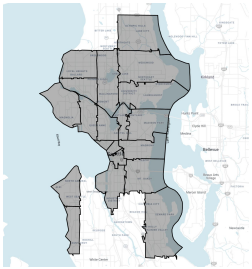
name	point_full	point_2decs
Ann Arbor, MI	POINT(-83.8446336 42.2523272)	POINT(-83.84 42.25)

Combine multiple polygons into a single unit

- ST_Union_Agg
- Find centroid of the union with ST_Centroid_Agg

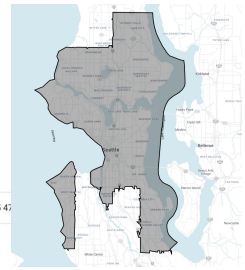
```
SELECT
  zip_code,
  zcta_geom AS loc
FROM
  `bigquery-public-data.geo_us_boundaries.us_zip_codes`
WHERE
  city = 'Seattle city'
  AND state_code = 'WA'
```

zip_code	loc
98195	POLYGON((-122.312412 47.647196, -122.310437 47.64712, -122.304698 47...
98121	POLYGON((-122.361829642306 47.6187252120307, -122.3568893920553 47...
98199	POLYGON((-122.429841 47.658915, -122.421367 47.662949, -122.421366...



```
SELECT
  ST_UNION_AGG(zcta_geom) as all_zips,
  ST_CENTROID_AGG(zcta_geom) as centroid
FROM
  `bigquery-public-data.geo_us_boundaries.us_zip_codes`
WHERE
  city = 'Seattle city'
  AND state_code = 'WA'
```

all_zips	centroid
MULTIPOLYGON(((-122.421139 47.57602, -122.409839 47.56892, -122.4097...	POINT(-122.3197801500555 47



Google Cloud

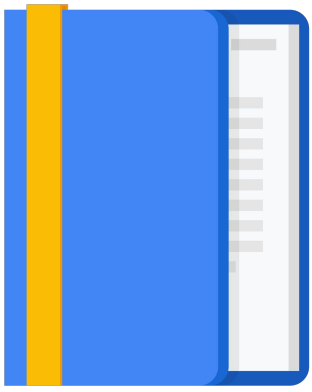
Agenda

Analytic Window Functions

Using With Clauses

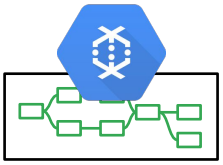
GIS Functions

Performance Considerations

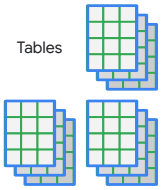


Google Cloud

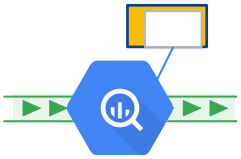
Best practices for fast, smart, data-driven decisions



Use Dataflow to do the processing and data transformations

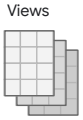


Create multiple tables for easy analysis



Use BigQuery for streaming analysis and dashboards

Store data in BigQuery for low cost long term storage

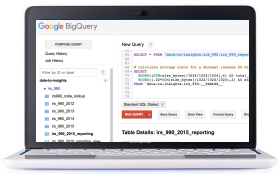
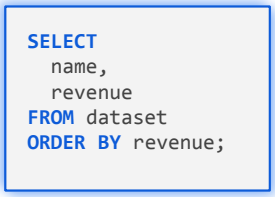


Create views for common queries

Google Cloud

This is the goal: fast smart decisions.

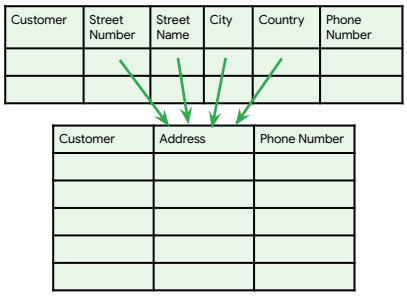
Best practices for analyzing data with BigQuery

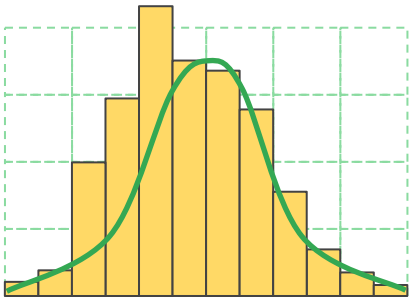
Dataset	Questions	SQL
		
Know your data	Ask good questions	Use queries effectively

Google Cloud

Exploring a dataset through SQL is more than just writing good code. You need to know what destination you're heading towards and the general layout of your data. Good data analysts will explore how the dataset is structured even before writing a single line of code.

How to optimize in production?
Revisit the schema. Revisit the data.





Stop accumulating work that could be done earlier.

Google Cloud

People often analyze data and develop a schema at the beginning of a project and never revisit those decisions. The assumptions they made at the beginning may have changed and are no longer true. So they attempt to adjust the downstream processes without ever reviewing and considering changing some of the original decisions.

Look at the data. Perhaps it was evenly distributed at the start of the project but as the work has grown, the data may have become skewed.

Look at the schemas. What were the goals then? Are those the same goals now? Is the organization of the data optimized for current operations?

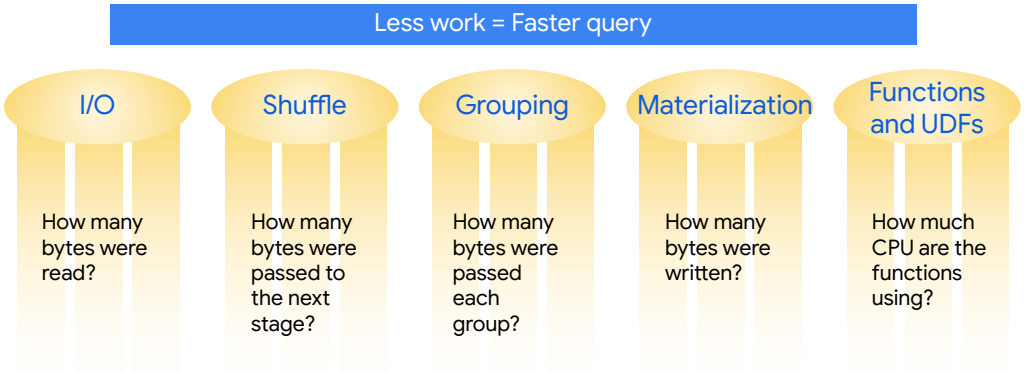
Stop accumulating work that could be done earlier. Analogy: dirty dishes. If you clean them as you use them, the kitchen remains clean. If you save them, you end up with a sink full of dirty dishes and a lot of work.

<INSTRUCTOR>
True story. I was working on an application for an organization. At first the data was exploratory. We went through a complicated schema design process that captured many of the nuances of the circumstance in a complex collection of relational tables. After weeks of work, the Data Analysts built a report generation system that was amazingly complex and took four days to run. Unfortunately, in production, the report was needed every two days. A consultant company was hired. After weeks of analysis they suggested replicating the entire system so that reports could be run in parallel to meet the schedule requirements.

I started experimenting with optimizing the system. I started with intermediate materialization of tables. And that made the report generation faster. Then I realized that we could get the report out much faster if we stored the RESULTS the report needed. I created intermediate tables with the fields that were needed for the last stages of the report. Eventually I rewrote the data ingress portion of the application. Instead of accumulating data and processing it, I pushed the processing back to data ingest.

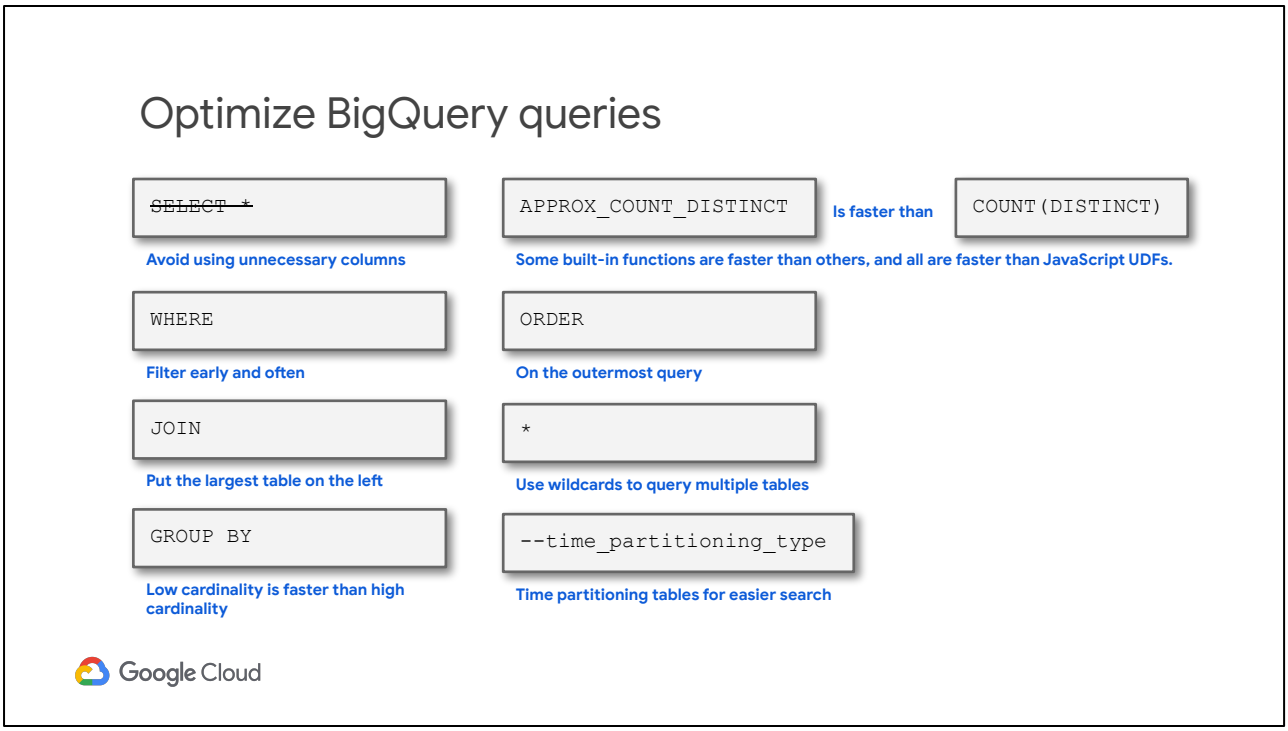
For example, we had a total number of items in the final report. And in the original application, this required counting several million items. I created a table with the final count of items in it. And wrapped the operator functions so that every time an object was added to the database this field in the intermediate table was incremented, and each time an object was removed this field was decremented. Now the report already had this sum total when it began its queries and didn't need to calculate it dynamically. The time to run the report was radically reduced without buying a second system.
</INSTRUCTOR>

Improve scalability by improving efficiency

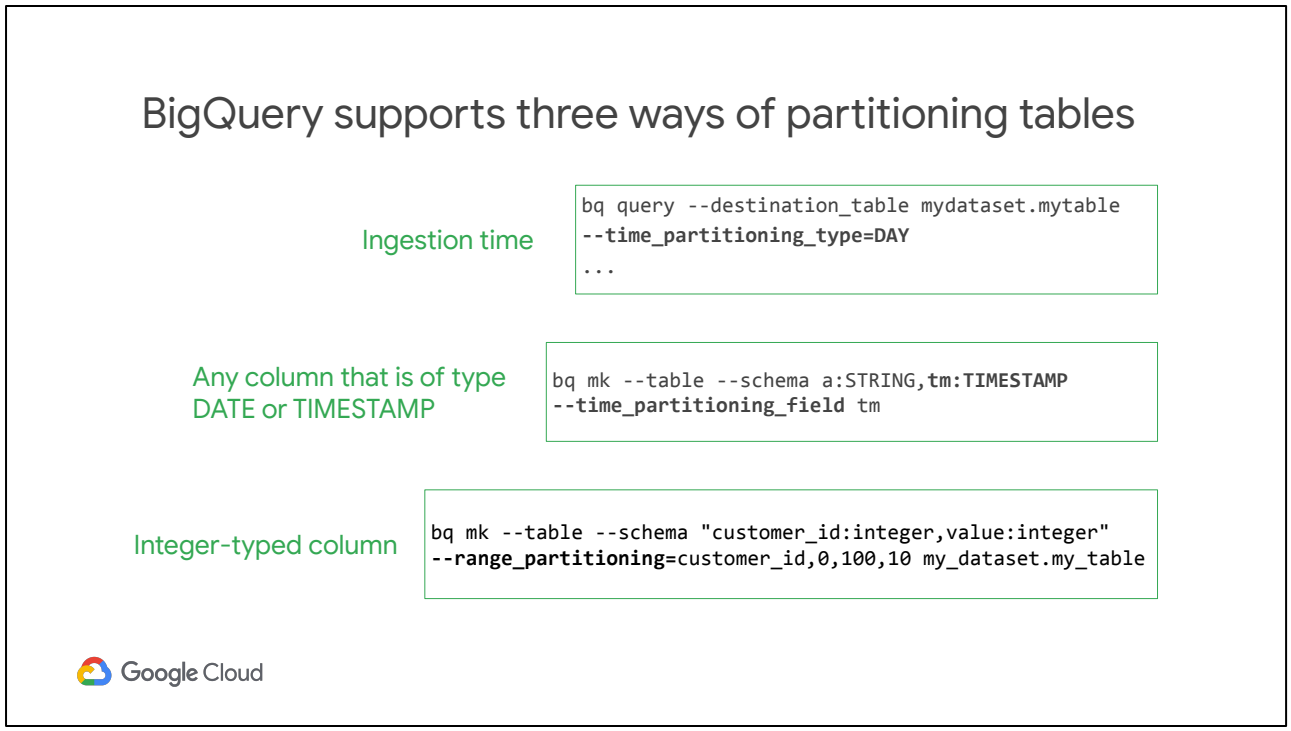


Google Cloud

Old Silicon Valley saying: "Don't scale up your problems. Solve them early while they are small."



See: https://cloud.google.com/bigquery/docs/best-practices-performance-compute



If you remember, earlier in this course we talked about how to build a data warehouse. We mentioned that you can optimize the tables in your data warehouse by reducing the cost and amount of data read. This can be achieved by partitioning your tables.

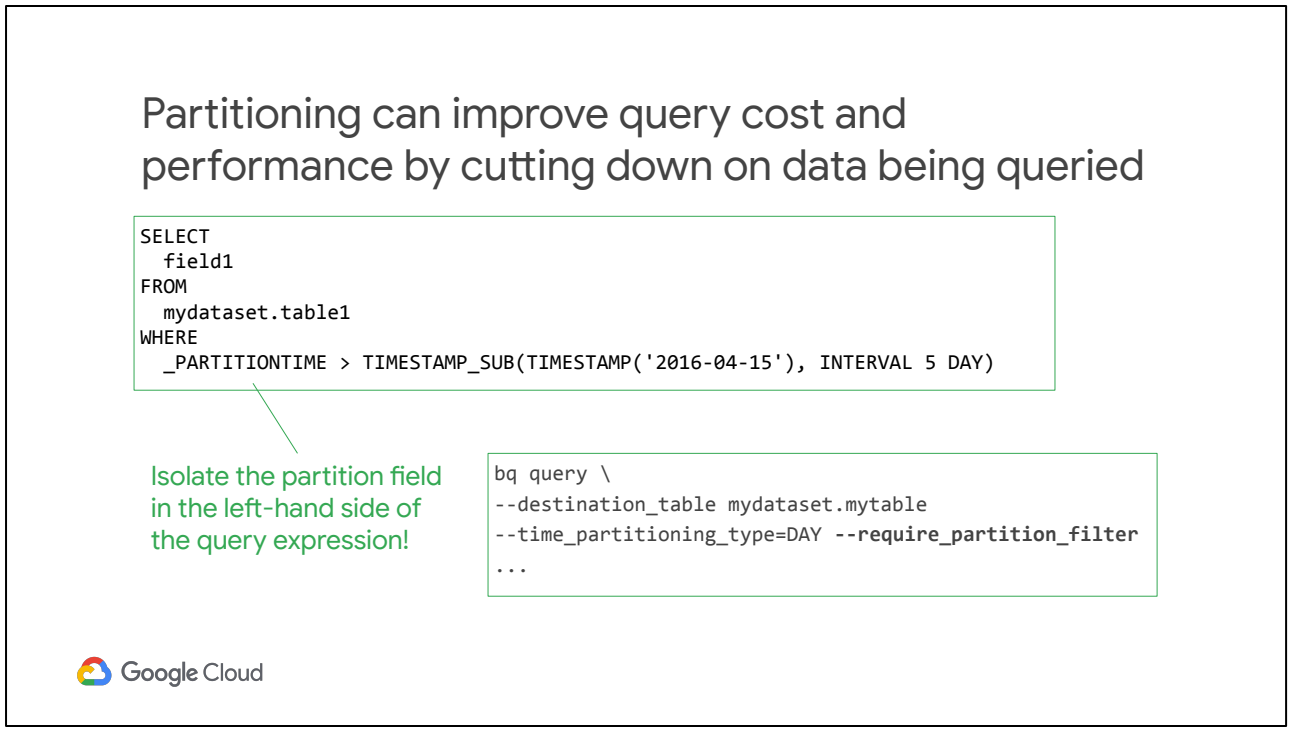
Partitioning tables is very relevant to performance so let's revisit in the next few slides some of the main points already covered.

You enable partitioning during the table-creation process.

The slide shows how to migrate an existing table to an ingestion-time-partitioned table: just use destination table. It will cost you one table scan.

BigQuery creates new date-based partitions automatically, with no need for additional maintenance. In addition, you can specify an expiration time for data in the partitions.

New data that is inserted into a partitioned table is written to the raw partition at the time of insert. To explicitly control which partition the data is loaded to, your load job can specify a particular date partition.

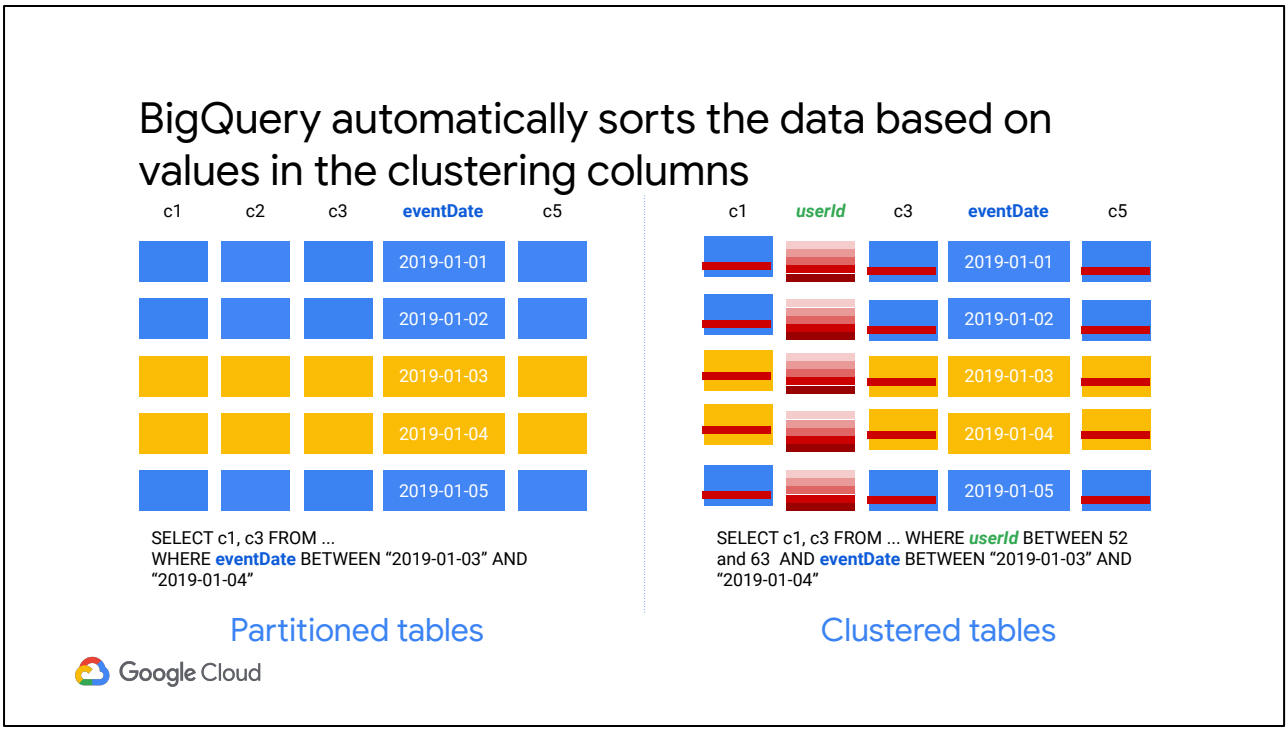


In a table partitioned by a date or timestamp column, each partition contains a single day of data. When the data is stored, BigQuery ensures that all the data in a block belongs to a single partition. A partitioned table maintains these properties across all operations that modify it: query jobs, Data Manipulation Language (DML) statements, Data Definition Language (DDL) statements, load jobs, and copy jobs. This requires BigQuery to maintain more metadata than a non-partitioned table. As the number of partitions increases, the amount of metadata overhead increases.

Although more metadata must be maintained, by ensuring that data is partitioned globally, BigQuery can more accurately estimate the bytes processed by a query before you run it. This cost calculation provides an upper bound on the final cost of the query.

A good practice is to require that queries always include the partition filter.

Make sure that the partition field is isolated on the left-hand-side since that's the only way BigQuery can quickly discard unnecessary partitions.

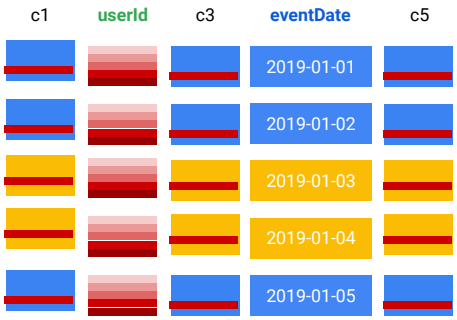


Clustering can improve the performance of certain types of queries, such as queries that use filter clauses and those that aggregate data. When data is written to a clustered table by a query or load job, BigQuery sorts the data using the values in the clustering columns. These values are used to organize the data into multiple blocks in BigQuery storage. When you submit a query containing a clause that filters data based on the clustering columns, BigQuery uses the sorted blocks to eliminate scans of unnecessary data.

Similarly, when you submit a query that aggregates data based on the values in the clustering columns, performance is improved because the sorted blocks co-locate rows with similar values.

In this example, the table is partitioned by eventDate and clustered by userId. Now, because the query looks for partitions in a specific range, only 2 of the 5 partitions are considered.
Because the query looks for userId in a specific range, BigQuery can jump to the row range and read only those rows for each of the columns needed.

Set up clustering at table creation time




```
CREATE TABLE mydataset.myclusteredtable
 (
   c1 NUMERIC,
   userId STRING,
   c3 STRING,
   eventDate TIMESTAMP,
   C5 GEOGRAPHY
  )
PARTITION BY DATE(eventDate)
CLUSTER BY userId
OPTIONS (
   partition_expiration_days=3,
   description="cluster")
AS SELECT * FROM mydataset.myothertable
```

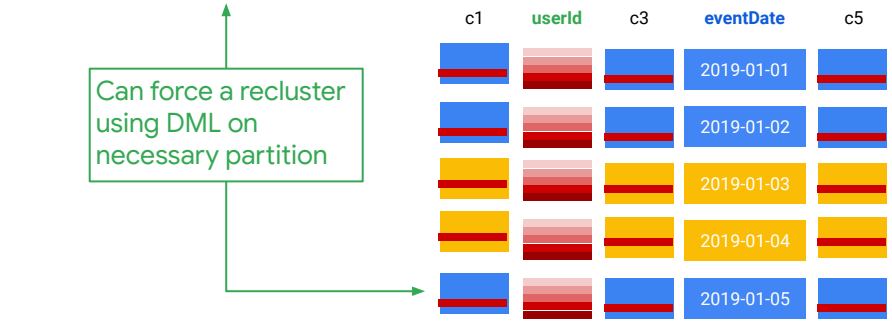
Google Cloud

You set up clustering at table creation time. Here, we are creating the table, partitioning by eventDate, and clustering by userId. We are also telling BigQuery to expire partitions that are more than 3 days old.

The columns you specify in the cluster are used to co-locate related data. When you cluster a table using multiple columns, the order of columns you specify is important. The order of the specified columns determines the sort order of the data.

In streaming tables, the sorting fails over time, and so BigQuery has to recluster

```
UPDATE ds.table
SET c1 = 300
WHERE c1 = 300
AND eventDate > TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 1 DAY)
```



Google Cloud

Over time, as more and more operations modify a table, the degree to which the data is sorted begins to weaken, and the table becomes only partially sorted. In a partially sorted table, queries that use the clustering columns may need to scan more blocks compared to a table that is fully sorted. You can re-cluster the data in the entire table by running a SELECT * query that selects from and overwrites the table, but guess what! You don't need to do that anymore.

BigQuery will automatically recluster your data

Automatic re-clustering

free

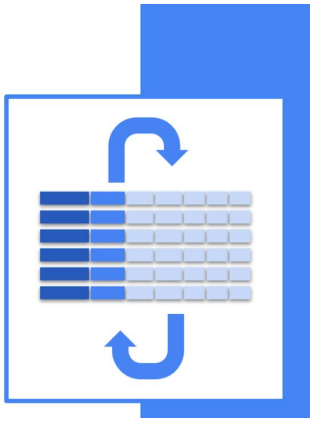
Doesn't consume your query resources

maintenance-free

Requires no setup or maintenance

autonomous

Automatically happens in the background



Google Cloud

The great news is that BigQuery now periodically does auto-reclustering for you so you don't need to worry about your clusters getting out of date as you get new data.

Automatic re-clustering is absolutely free and automatically happens in the background -- you dont need to do anything additional to enable it.

https://cloud.google.com/blog/products/data-analytics/whats-happening-bigquery-adding-speed-and-flexibility-10x-streaming-quota-cloud-sql-federation-and-more
https://cloud.google.com/bigquery/docs/clustered-tables#automatic_re-clustering

Organize data through managed tables

Partitioning

Filtering storage before query execution begins to reduce costs.

Reduces a full table scan to the partitions specified.

A single column results in lower cardinality (e.g., thousands of partitions).

- Time partitioning (Pseudocolumn)
- Time partitioning (User Date/Time column)
- Integer range partitioning

Clustering

Storage optimization within columnar segments to improve filtering and record colocation.

Clustering performance and cost savings can't be assessed before query begins.

Prioritized clustering of up to 4 columns, on more diverse types (but no nested columns).

Partitioning provides a way to obtain accurate cost estimates for queries and guarantees improved cost and performance. Clustering provides additional cost and performance benefits in addition to the partitioning benefits.

When to use clustering

Your data is already partitioned on a `DATE` or `TIMESTAMP` or Integer Range

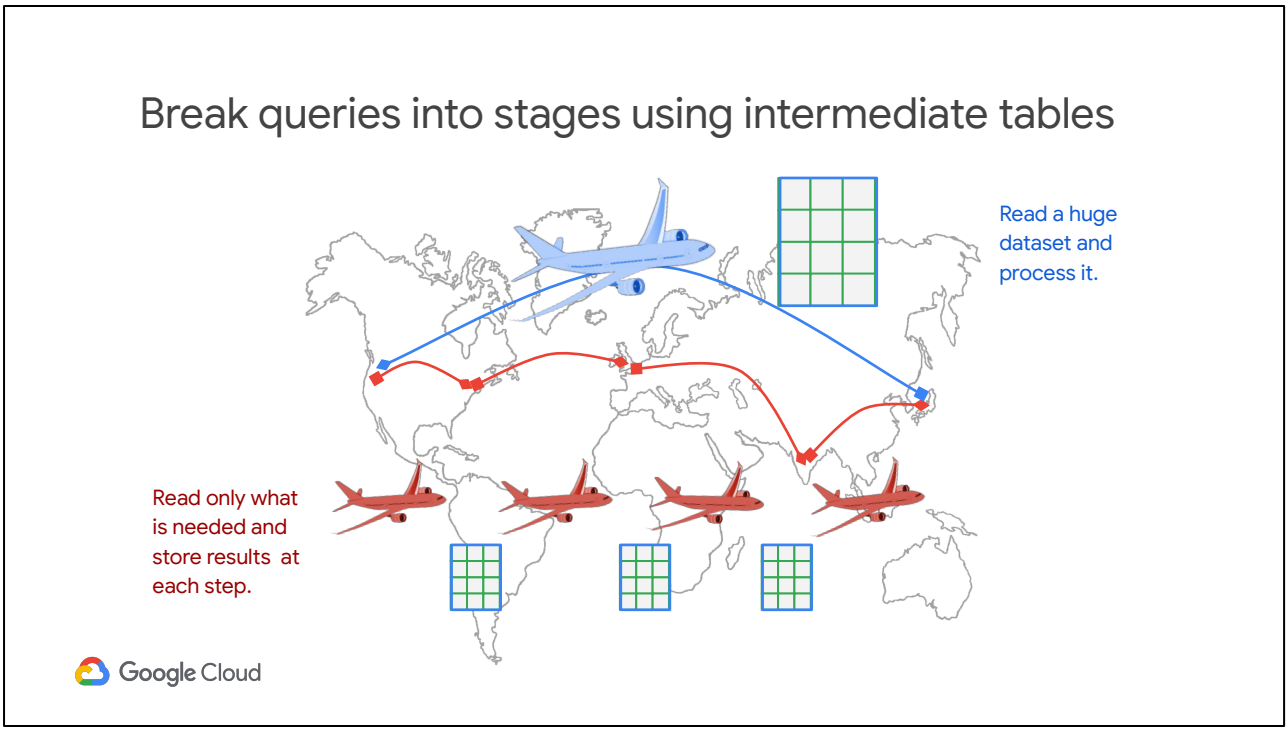
You commonly use filters or aggregation against particular columns in your queries.

Google Cloud

BigQuery supports clustering for both partitioned and non-partitioned tables.

When you use clustering and partitioning together, the data can be partitioned by a date or timestamp column and then clustered on a different set of columns. In this case, data in each partition is clustered based on the values of the clustering columns. Partitioning provides a way to obtain accurate cost estimates for queries.

Keep in mind if you don't have partitioned columns and you want the benefits of clustering you can create a fake_date column of type DATE and have all the values be NULL.



If you create a large, multi-stage query, each time you run it, BigQuery reads all the data that is required by the query. All the data that is read each time the query is run.

Intermediate table materialization is where you break the query into stages. Each stage materializes the query results by writing them to a destination table.

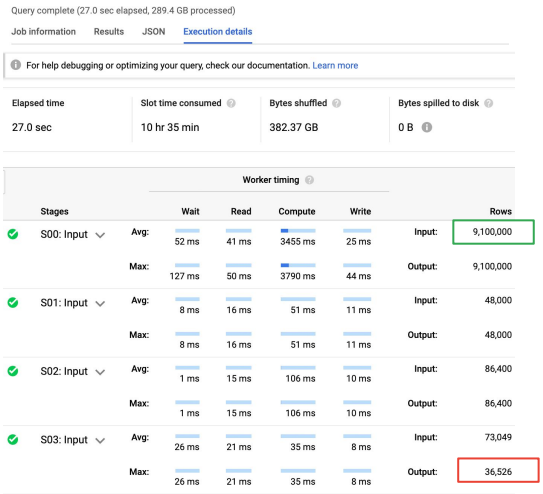
Querying the smaller destination table reduces the amount of data that is read. In general, storing the smaller materialized results is more efficient than processing the larger amount of data.

The analogy is air travel from Sunnyvale, California, USA to Japan. There is one direct flight. Or a series of four shorter connecting flights. The direct flight has to carry the fuel for the entire journey. The connecting flights only need enough fuel for each leg of the trip. The total fuel used in landing and taking off (an analogy for storing the intermediate tables) was less than the total fuel used for carrying everything in the entire journey.

Here is a tip: Compare costs of storing the data with costs of processing the data. Processing the large dataset will use more processing resources. Storing the intermediate tables will use more storage resources. In general, processing data is more expensive than storing data. But you can do the calculations yourself to establish a breakeven for your particular use case.

https://pixabay.com/vectors/airplane-jet-aircraft-plane-flight-309503/

Track input and output counts with Execution details tab



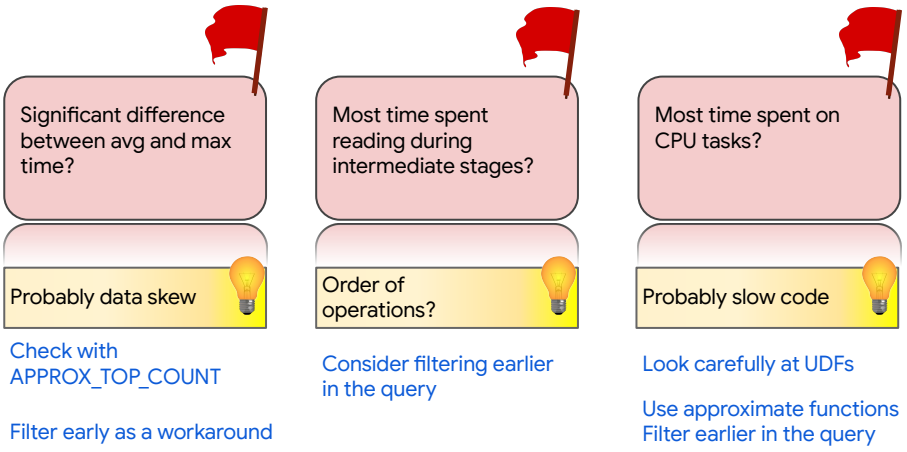
Google Cloud

A different way to check how many records are being processed is by clicking on the Explanation tab in the BigQuery UI after running a query.

We started with 9.1 Million rows and filtered down to roughly 36k

The query stages represent how BigQuery mapped out the work required to perform the query job.

Using BigQuery plans to optimize



Google Cloud

Approximate Functions: APPROX_COUNT_DISTINCT(expression) Description. Returns the approximate result for COUNT(DISTINCT expression). The result is less accurate but it performs much more effiicently.

Analyze BigQuery performance in Cloud Monitoring

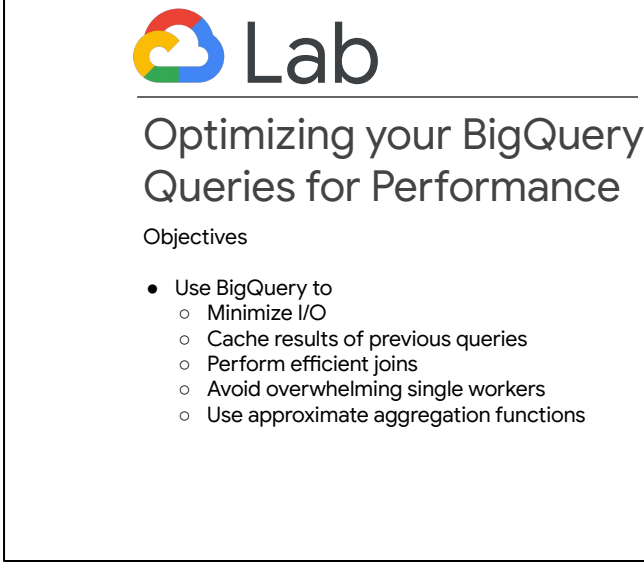
Custom Dashboards

Metrics include:

- slots utilization
- queries in flight
- upload bytes
- stored bytes

Google Cloud

These charts show Slot Utilization, Slots available and queries in flight for a 1 hr period.



Optimizing your BigQuery Queries for Performance
https://gcpstaging.qwiklabs.com/labs/24799/edit

Once your data is loaded into BigQuery, you are charged for storing it

Active Storage Pricing

Storage pricing is prorated per MB, per second. For example, if you store:

- 100 MB for half a month, you pay $0.001 (a tenth of a cent)
- 500 GB for half a month, you pay $5 ($0.02/GB per month)
- 1 TB for a full month, you pay $20

Long-term Storage Pricing

- Table or partition not edited 90+ consecutive days
- Pricing drops ≈ 50%
- No degradation (performance, durability, availability, functionality)
- Applies to BigQuery storage only

Google Cloud

Storage pricing is based on the amount of data stored in your tables when it is uncompressed. The size of the data is calculated based on the data types of the individual columns. Active storage pricing is prorated per MB, per second.

If a table is not edited for 90 consecutive days it is considered long-term storage. The price of storage for that table automatically drops by approximately 50 percent. There is no degradation of performance, durability, availability, or any other functionality.

If the table is edited, the price reverts back to the regular storage pricing, and the 90-day timer starts counting from zero. Anything that modifies the data in a table resets the timer, including:
- Loading data into a table
- Copying data into a table
- Writing query results to a table
- Using the Data Manipulation Language
- Using Data Definition Language
- Streaming data into the table

All other actions do not reset the timer, including:
- Querying a table
- Creating a view that queries a table
- Exporting data from a table
- Copying a table (to another destination table)
- Patching or updating a table resource

https://cloud.google.com/bigquery/pricing

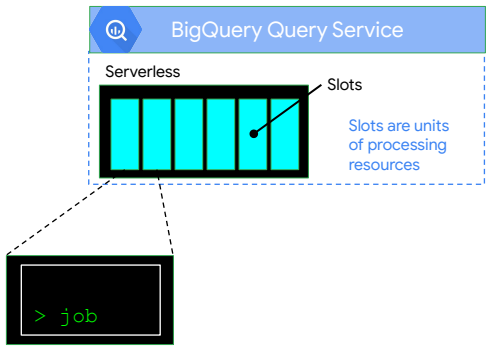
Which BigQuery pricing model to pick?

On-Demand Pricing	Flat-Rate Pricing
• $5/TB of data processed • Quota limit of 2,000 slots • Slots shared amongst all on demand users • 1st TB of data processed is free each month	• Fixed rate pricing is $10k* per 500 slots per month • Slots are dedicated 24/7 • Starting at 500 slots • Unlimited use of slots
Good for **exploratory work** and discovery	Good for **multiple large workloads** and **consistent monthly billing**

Google Cloud

Fixed rate pricing is $10k* per 500 slots per month. A 25% discount is offered for customers choosing a term length of a least 1-year ($7.5k per 500 slots). Capacity is sold in increments of 500 slots with a current minimum of 500 slots.

Slots are units of resources that are consumed when a query is run



Google Cloud

Because you don't get to see the VMs, BigQuery exposes slots to help you manage resource consumption and costs. Slots are units of resources such as processing power and memory that are consumed when a query is run. BigQuery automatically calculates how many slots are required by each query, depending on query size and complexity. The default slot capacity and allocation work well in most cases. You can monitor slot usage in Stackdriver.

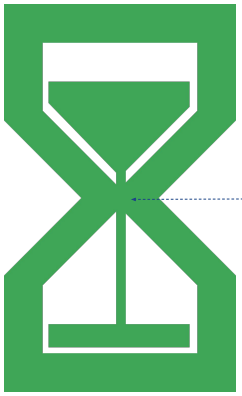
Candidate circumstances where additional slot capacity might improve performance are solutions with very complex queries on very large datasets with highly concurrent workloads, You can read more about slots in the online documentation or contact a sales representative.

Estimating the right BigQuery slots allocation is critical

Guideline: It is recommended to plan for 2,000 BigQuery slots for every 50 medium-complexity queries simultaneously

50 queries = 2,000 Slots

(quota: 100 concurrent queries, but this can be raised)



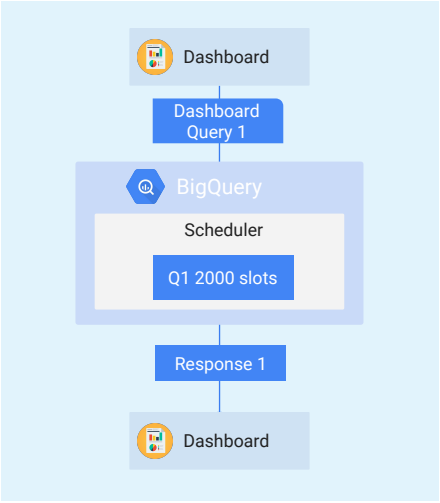
Slots Available

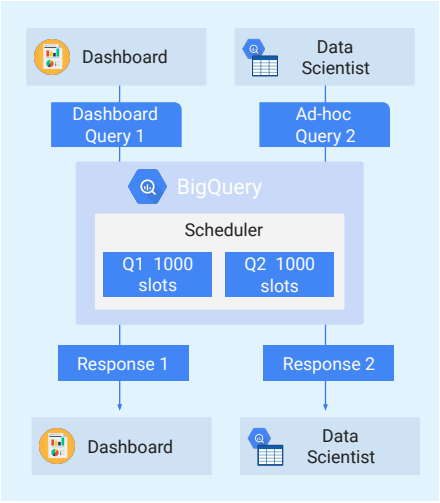
More slots	=	Faster depletion rate
More concurrent queries	=	Slower depletion rate

Google Cloud

BigQuery doesn't support fine-grained prioritization of interactive or batch queries. To avoid a pile up of BigQuery jobs and timely execution, estimating the right BigQuery slots allocation is critical. Currently, BQ times out any query taking longer than 6 hours.

BigQuery has a fair scheduler

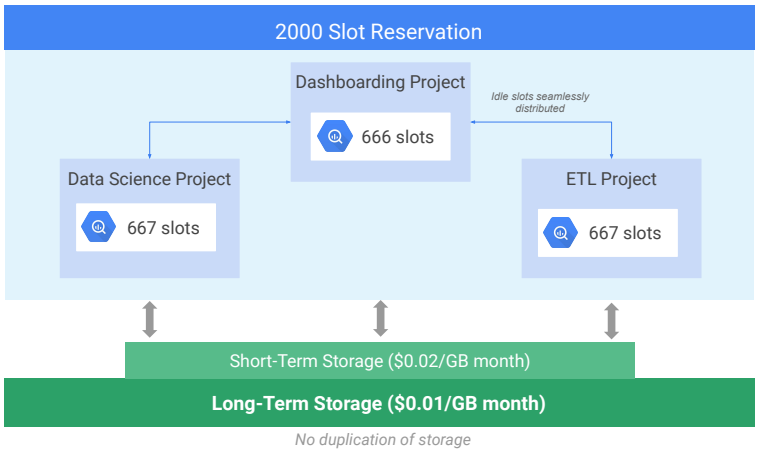




Google Cloud

If one query is executing within BigQuery, it has full access to the amount of slots available to the project or reservation, by default 2000
If we suddenly execute a second query, BigQuery will split the slots between the 2 queries with each getting half the total amount of slots available, in this case 1000 each
This subdividing of compute resources will continue to happen as more queries are executed
This is a long way of saying, it's unlikely that one resource-heavy query will overpower the system and steal resources from other running queries

Concurrency is fair across projects, users, and queries, with unused capacity fairly divided among existing tasks

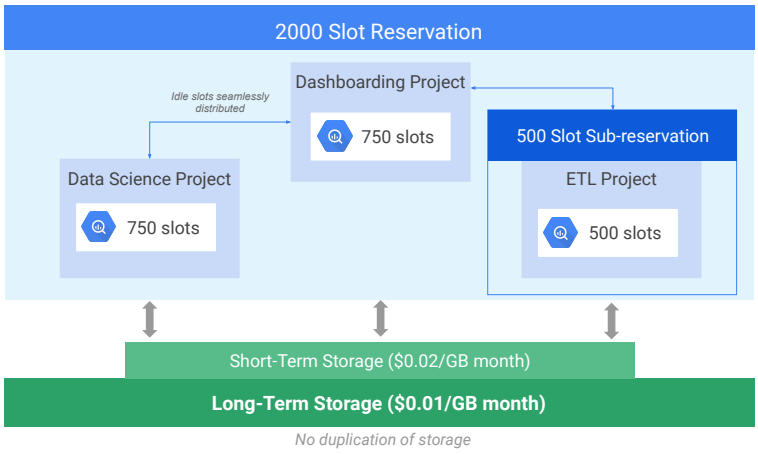


Google Cloud

In flat rate pricing they will have a fixed number of slots. Concurrency is fair across projects, users, and queries. That is if you have 2k slots and 2 projects, each project can get up to 1k slots. If one project uses less, the other project will be able to use all of the remainder. If you have 2 users in each project, each user will be able to get 500 slots. And if each of the two users each of the two projects, runs two queries, they'll each get 500 slots.

This is a long way of saying they won't likely degrade performance by adding projects.

To prioritize projects, set up a hierarchical reservation



Google Cloud

Note that if they want to prioritize one project over another, they can set up a hierarchical reservation. Let's say you have an ETL project that is somewhat lower priority than your dashboarding project. You can give the ETL project 500 slots as a sub-reservation and the dashboarding project will be in the outer one. If both projects are fully using their reservations, the ETL project can never get more than 500 slots. When one project is lightly used, the other project will be able to take the remaining slots.

Creating Date-Partitioned Tables in BigQuery

Objectives

- Query a partitioned dataset
- Create dataset partitions to improve query performance and reduce cost

OPTIONAL LAB
Creating Date-Partitioned Tables in BigQuery

Module Summary

- Use BigQuery's analytic window functions for advanced analysis
- BigQuery has built in GIS functionality
- Use BigQuery's WITH clause to help modularize your queries
- There are many ways to optimize your query performance