Proprietary + Confidential

Advanced BigQuery Functionality and Performance

Proprietary + Confidential

Advanced BigQuery Functionality and Performance

01	Analytic window functions
02	GIS functions
03	Performance considerations

Proprietary + Confidential

Advanced BigQuery Functionality and Performance

01	Analytic window functions
02	GIS functions
03	Performance considerations

Proprietary + Confidential

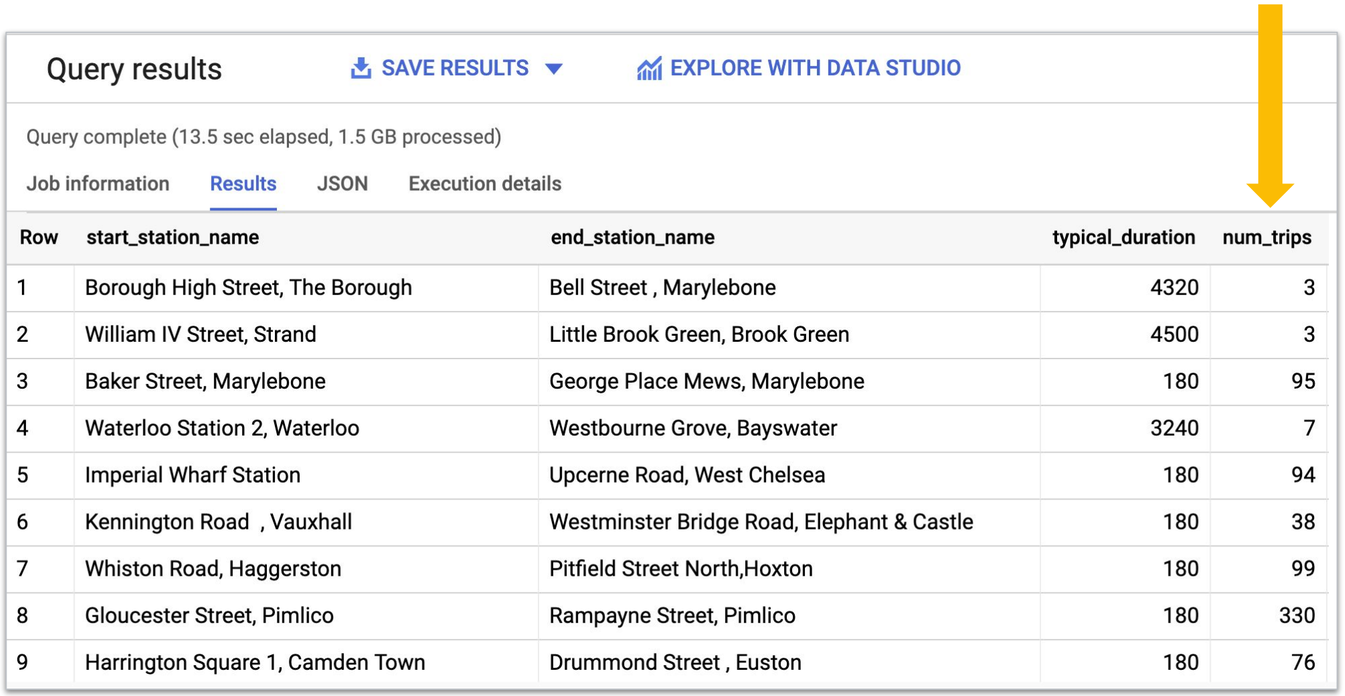
Use analytic window functions for advanced analysis

- Standard aggregations
- Navigation functions
- Ranking and Numbering functions

Proprietary + Confidential

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

Proprietary + Confidential

Some other “standard” aggregation functions

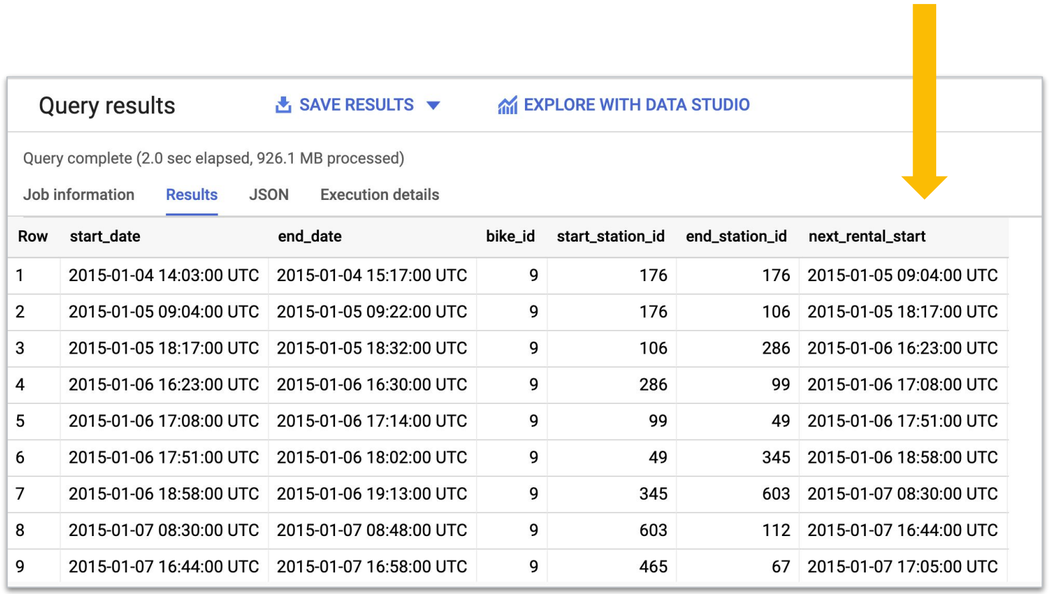
- SUM
- AVG
- MIN
- MAX
- BIT_AND
- BIT_OR
- BIT_XOR
- COUNTIF
- LOGICAL_OR
- LOGICAL_AND

More SQL aggregate functions

Proprietary + Confidential

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

Proprietary + Confidential

Some other navigation functions

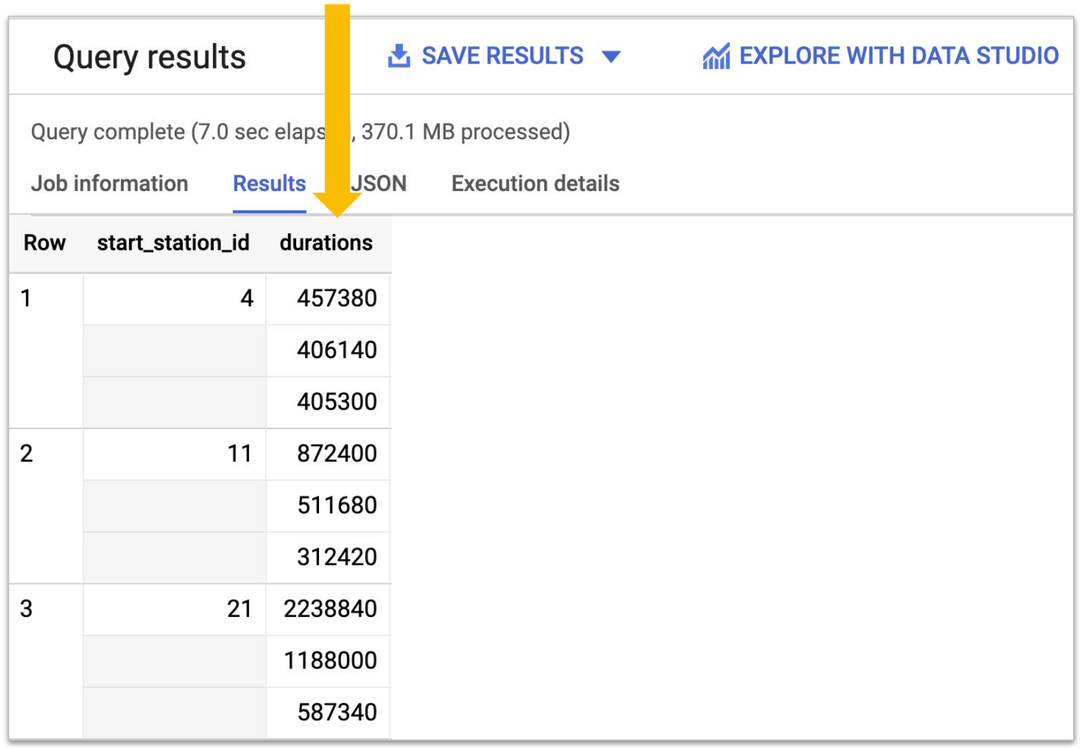
- NTH_VALUE
- LAG
- FIRST_VALUE
- LAST_VALUE

More SQL navigation functions

Proprietary + Confidential

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

Proprietary + Confidential

Example: **RANK() Function** for aggregating over groups of rows

firstname	department	startdate
Andrew	1	1/23/1999
Jacob	1	7/11/1990
Daniel	2	6/24/2004
Anna	1	10/7/2001
Pierre	1	2/22/2009
Ruth	2	6/6/2013
Anthony	1	11/29/1995
Isabella	2	9/28/1997
Jose	2	3/17/2013

PARTITION BY department

firstname	department	startdate
Andrew	1	1/23/1999
Jacob	1	7/11/1990
Anna	1	10/7/2001
Pierre	1	2/22/2009
Anthony	1	11/29/1995

firstname	department	startdate
Daniel	2	6/24/2004
Ruth	2	6/6/2013
Isabella	2	9/28/1997
Jose	2	3/17/2013

ORDER BY startdate

firstname	department	startdate
Jacob	1	7/11/1990
Anthony	1	11/29/1995
Andrew	1	1/23/1999
Anna	1	10/7/2001
Pierre	1	2/22/2009

firstname	department	startdate
Isabella	2	9/28/1997
Daniel	2	6/24/2004
Jose	2	3/17/2013
Ruth	2	6/6/2013

RANK ()

firstname	department	startdate	rank
Jacob	1	7/11/1990	1
Anthony	1	11/29/1995	2
Andrew	1	1/23/1999	3
Anna	1	10/7/2001	4
Pierre	1	2/22/2009	5

firstname	department	startdate	rank
Isabella	2	9/28/1997	1
Daniel	2	6/24/2004	2
Jose	2	3/17/2013	3
Ruth	2	6/6/2013	4

More SQL analytic functions

Google Cloud

Proprietary + Confidential

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;
```

Proprietary + Confidential

Some other ranking and numbering functions

- CUME_DIST
- DENSE_RANK
- ROW_NUMBER
- PERCENT_RANK

More SQL numbering functions

Proprietary + Confidential

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.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
```

- 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

More SQL query syntax

Proprietary + Confidential

Advanced BigQuery Functionality and Performance

01	Analytic window functions
02	GIS functions
03	Performance considerations

Proprietary + Confidential

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

Proprietary + Confidential

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

- ST_DWithin(geography_1, geography_2, distance)

in meters

```
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
```

Proprietary + Confidential

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

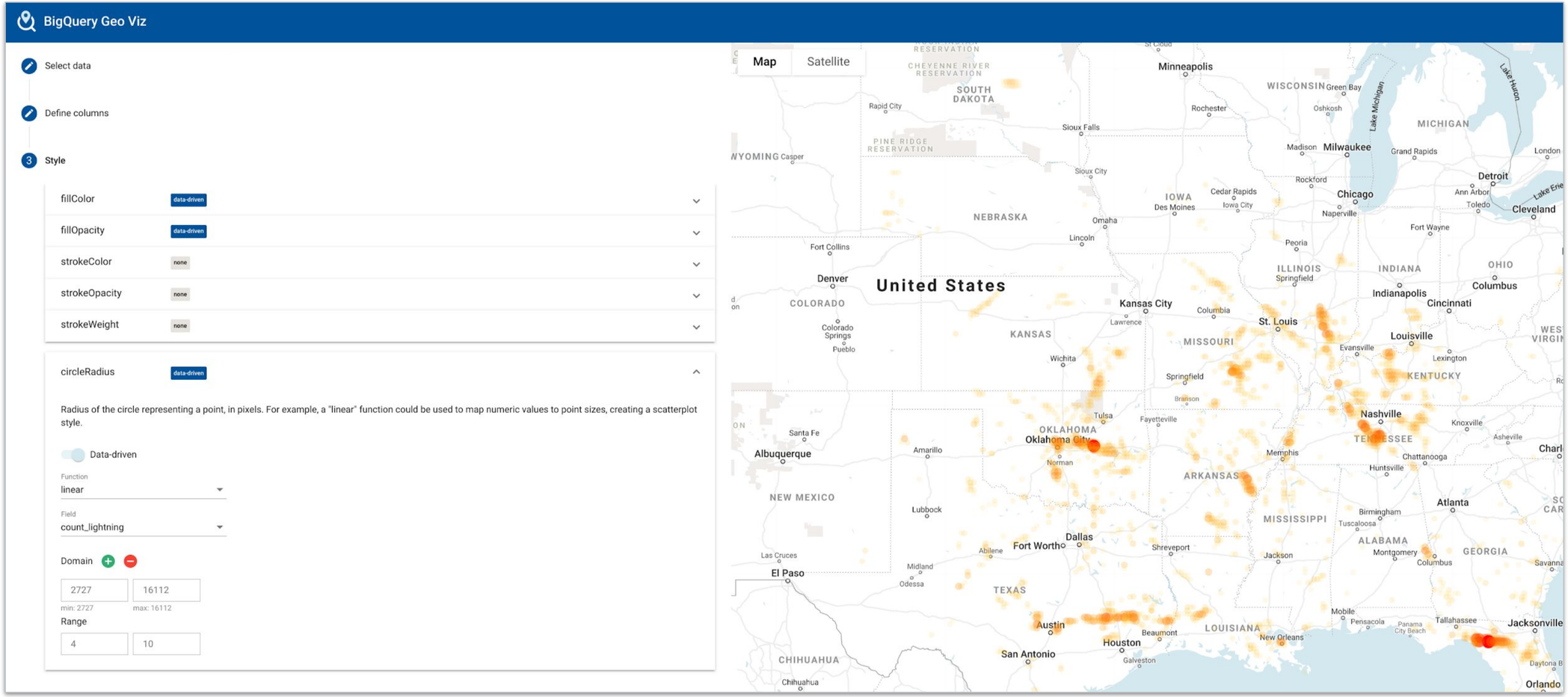
Queries in BigQuery are much more efficient if geographic data is stored as geography types rather than as basics.

```
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]}')
```

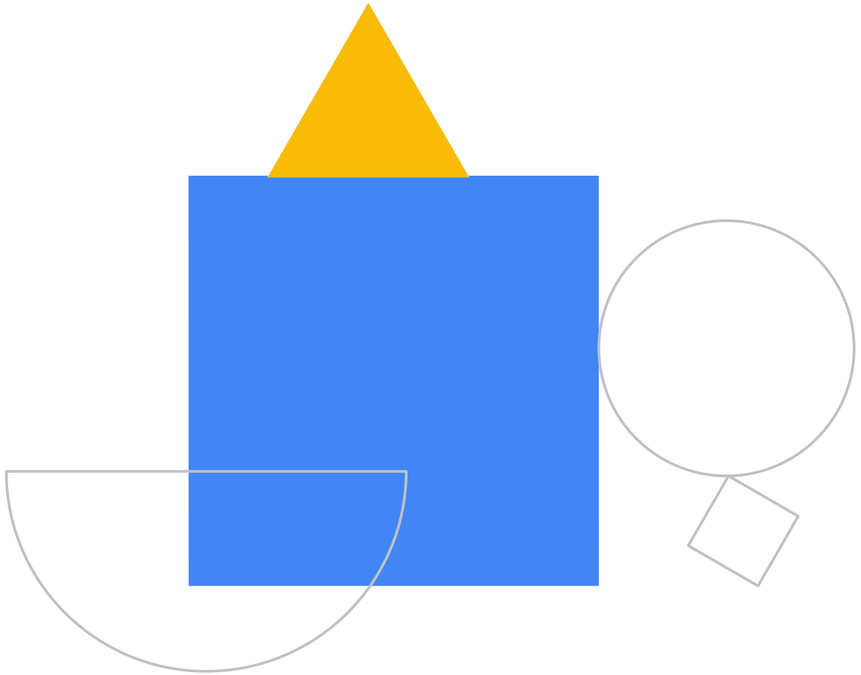
Visualization with BigQuery Geo Viz



Proprietary + Confidential

Demo

Mapping Fastest Growing Zip Codes with BigQuery GeoViz



Proprietary + Confidential

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)

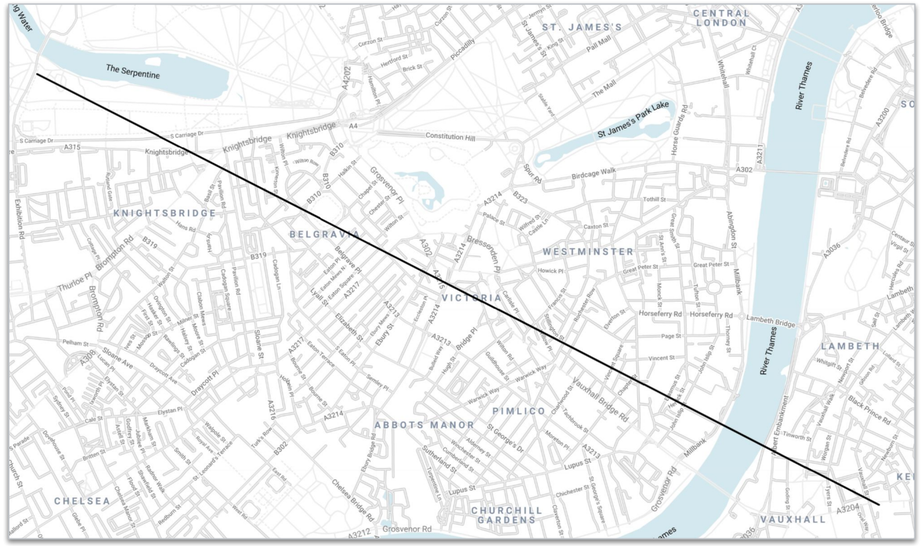
Google Cloud

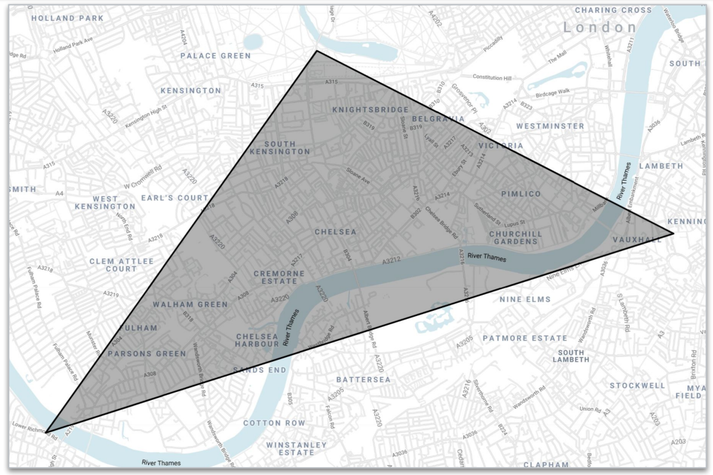
Proprietary + Confidential

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 …





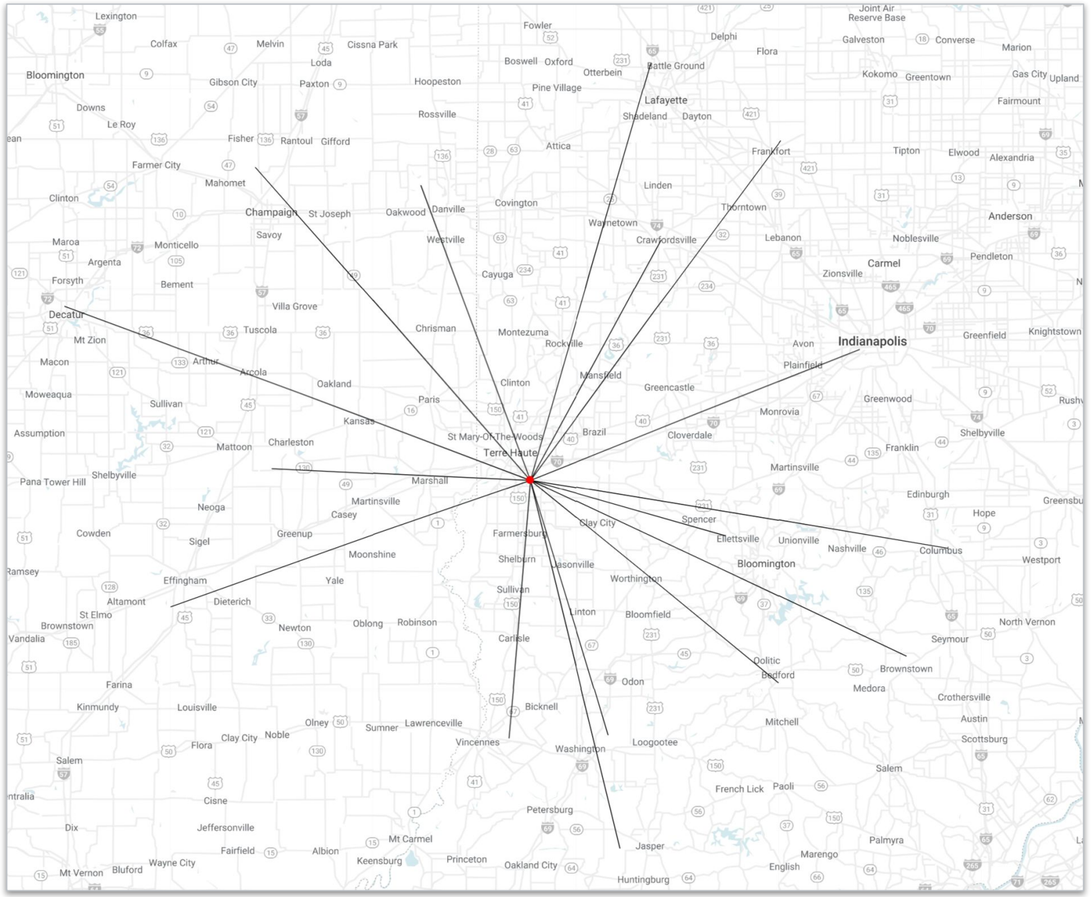
Google Cloud

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...



Proprietary + Confidential

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

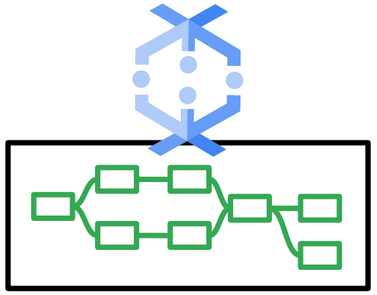
Proprietary + Confidential

Advanced BigQuery Functionality and Performance

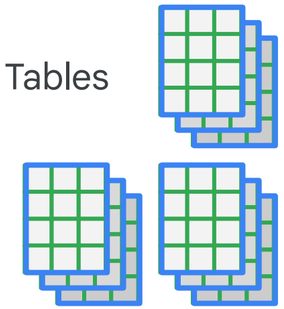
01	Analytic window functions
02	GIS functions
03	Performance considerations

Proprietary + Confidential

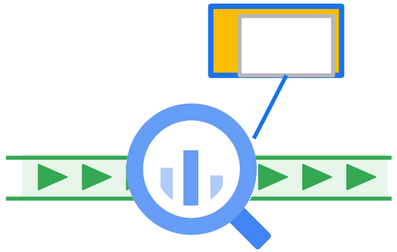
Best practices for fast, smart, data-driven decisions



Use Dataflow to do the processing and data transformations

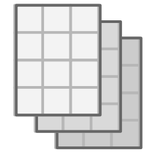


Create multiple tables for easy analysis



Use BigQuery for streaming analysis and dashboards

Store data in BigQuery for low cost long term storage

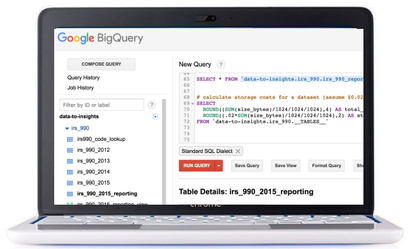


Create views for common queries

Proprietary + Confidential

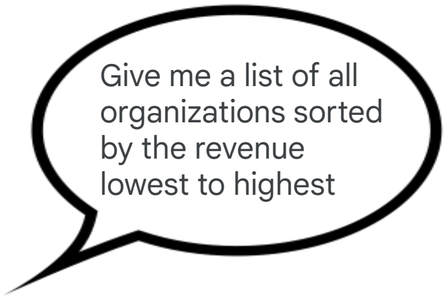
Best practices for analyzing data with BigQuery

Dataset



Know your data

Questions



Ask good questions

SQL

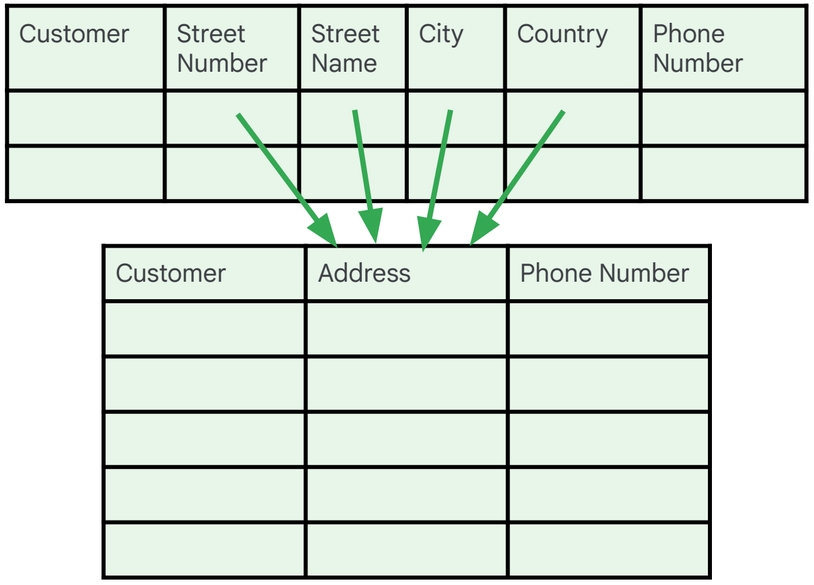
```
SELECT
  name,
  revenue
FROM dataset
ORDER BY revenue;
```

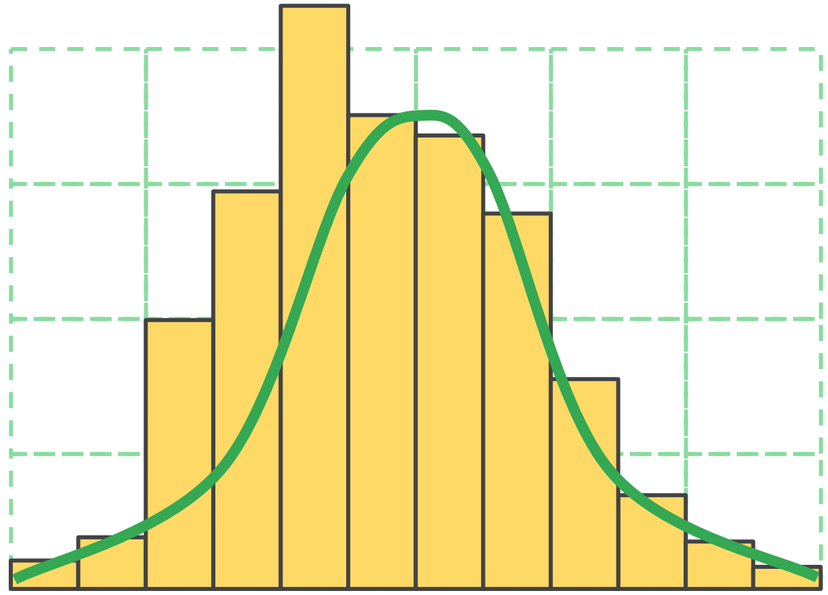
Use queries effectively

Google Cloud

Proprietary + Confidential

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





Stop accumulating work that could be done earlier.

Proprietary + Confidential

Improve scalability by improving efficiency

1	2	3	4	5
I/O	Shuffling	Grouping	Materialization	Functions and UDFs
How many bytes were read?	How many bytes were passed to the next stage?	How many bytes were passed through to each group?	How many bytes were written?	How much CPU are the functions using?

Less work = Faster query

Proprietary + Confidential

Optimize BigQuery queries

~~SELECT *~~

Avoid using unnecessary columns

WHERE

Filter early and often

JOIN

Put the largest table on the left

GROUP BY

Low cardinality is faster than high cardinality

APPROX_COUNT_DISTINCT **Is faster than** COUNT(DISTINCT)

Some built-in functions are faster than others, and all are faster than JavaScript UDFs.

ORDER

On the outermost query

*

Use wildcards to query multiple tables

--time_partitioning_type

Time partitioning tables for easier search

Proprietary + Confidential

BigQuery supports three ways of partitioning tables

Ingestion time

```
bq query --destination_table mydataset.mytable
--time_partitioning_type=DAY
...
```

Any column that is of type DATE or TIMESTAMP

```
bq mk --table --schema a:STRING,tm:TIMESTAMP
--time_partitioning_field tm
```

Integer-typed column

```
bq mk --table --schema "customer_id:integer,value:integer"
--range_partitioning=customer_id,0,100,10 my_dataset.my_table
```

Proprietary + Confidential

Partitioning can improve query cost and performance by cutting down on data being queried

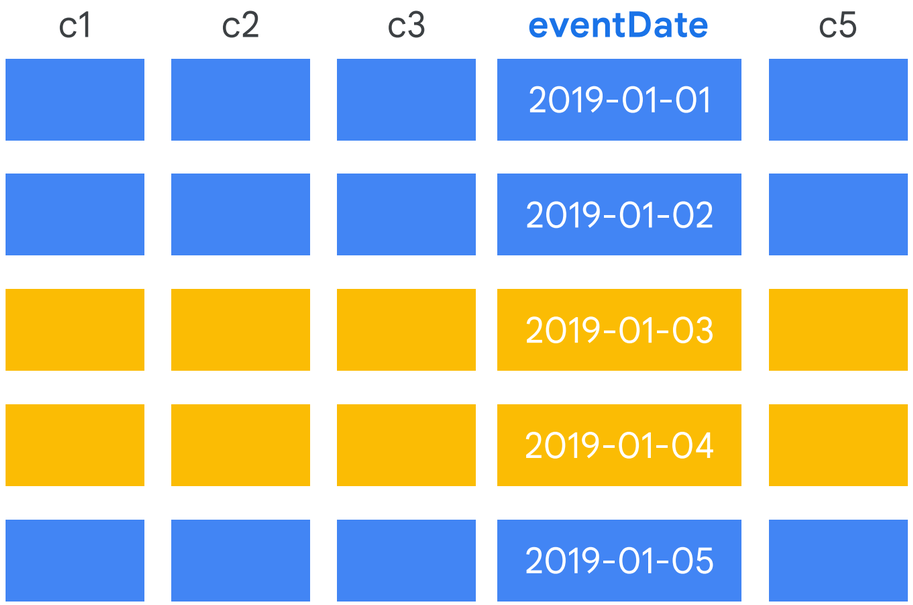
```
SELECT
  field1
FROM
  mydataset.table1
WHERE
  _PARTITIONTIME > TIMESTAMP_SUB(TIMESTAMP('2016-04-15'), INTERVAL 5 DAY)
```

Isolate the partition field in the left-hand side of the query expression!

```
bq query \
--destination_table mydataset.mytable
--time_partitioning_type=DAY
--require_partition_filter
...
```

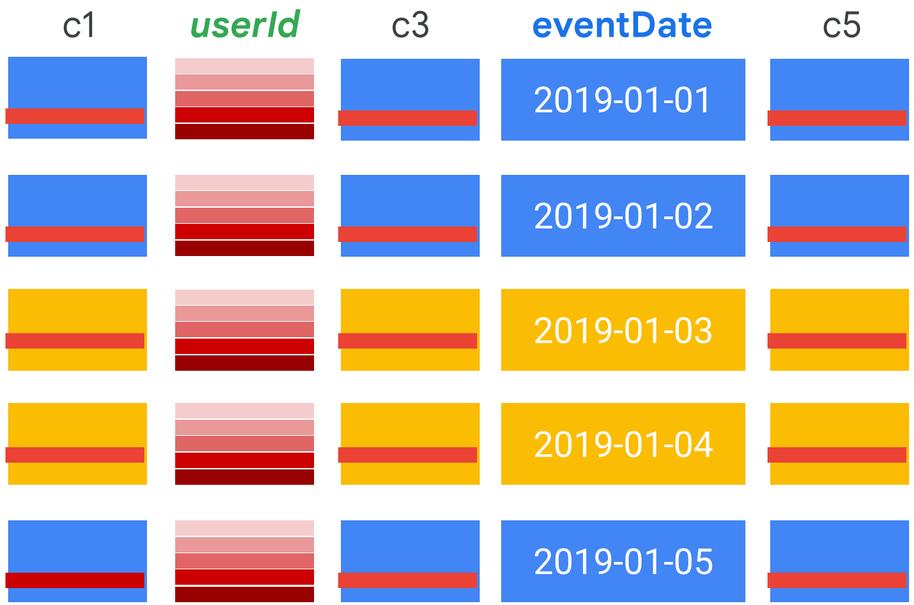
Proprietary + Confidential

BigQuery automatically sorts the data based on values in the clustering columns



SELECT c1, c3 FROM ...
WHERE **eventDate** BETWEEN “2019-01-03” AND “2019-01-04”

Partitioned tables



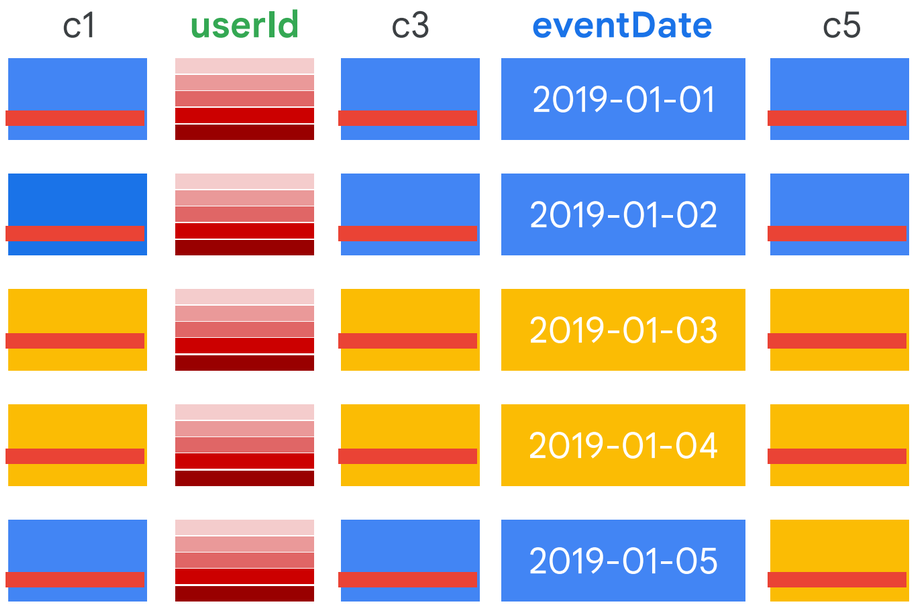
SELECT c1, c3 FROM ... WHERE ***userId*** BETWEEN 52 and 63 AND **eventDate** BETWEEN “2019-01-03” AND “2019-01-04”

Clustered tables

Google Cloud

Proprietary + Confidential

Set up clustering at table creation time



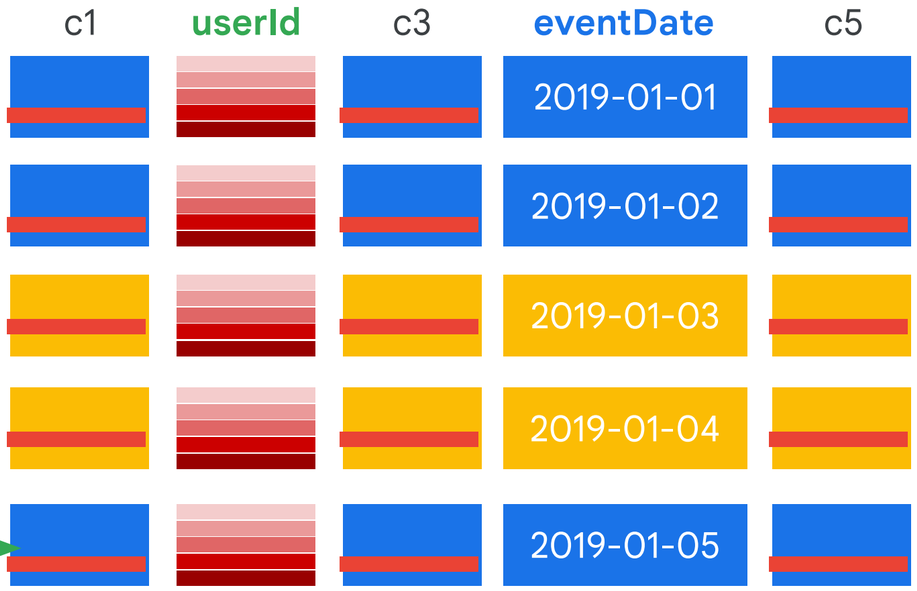

```
CREATE TABLE mydataset.myclusteredtable
 (
   c1 NUMERIC,
   userId STRING,
   c3 STRING,
   eventDate TIMESTAMP,
   C5 GEOGRAPHY
  )
 PARTITION BY DATE(eventDate)
 CLUSTER BY userId
 OPTIONS (
   partition_expiration_days=3,
   description="cluster")
 AS SELECT * FROM mydataset.myothertable
```

Proprietary + Confidential

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)
```

Can force a recluster using DML on necessary partition



Proprietary + Confidential

BigQuery will automatically recluster your data

Automatic re-clustering

free

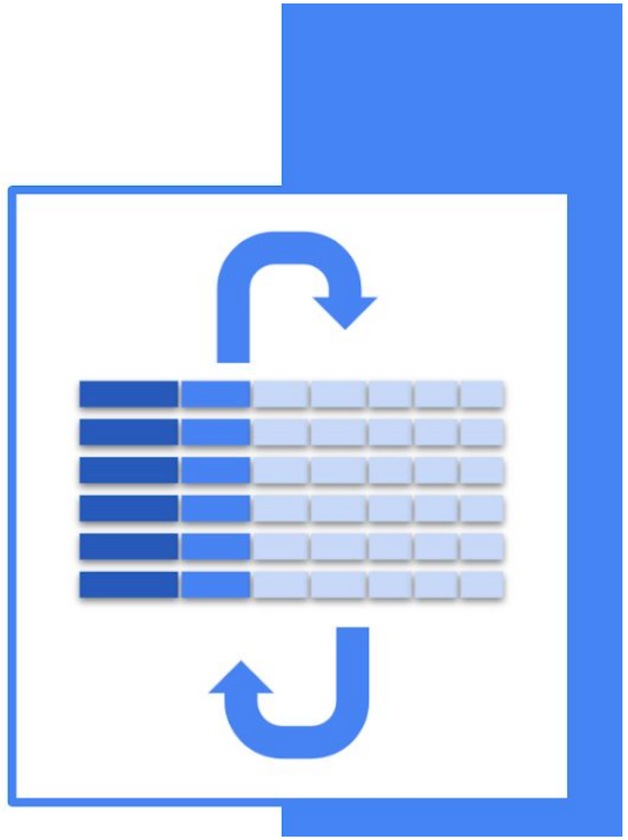
Doesn't consume your query resources

maintenance-free

Requires no setup or maintenance

autonomous

Automatically happens in the background



Proprietary + Confidential

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).

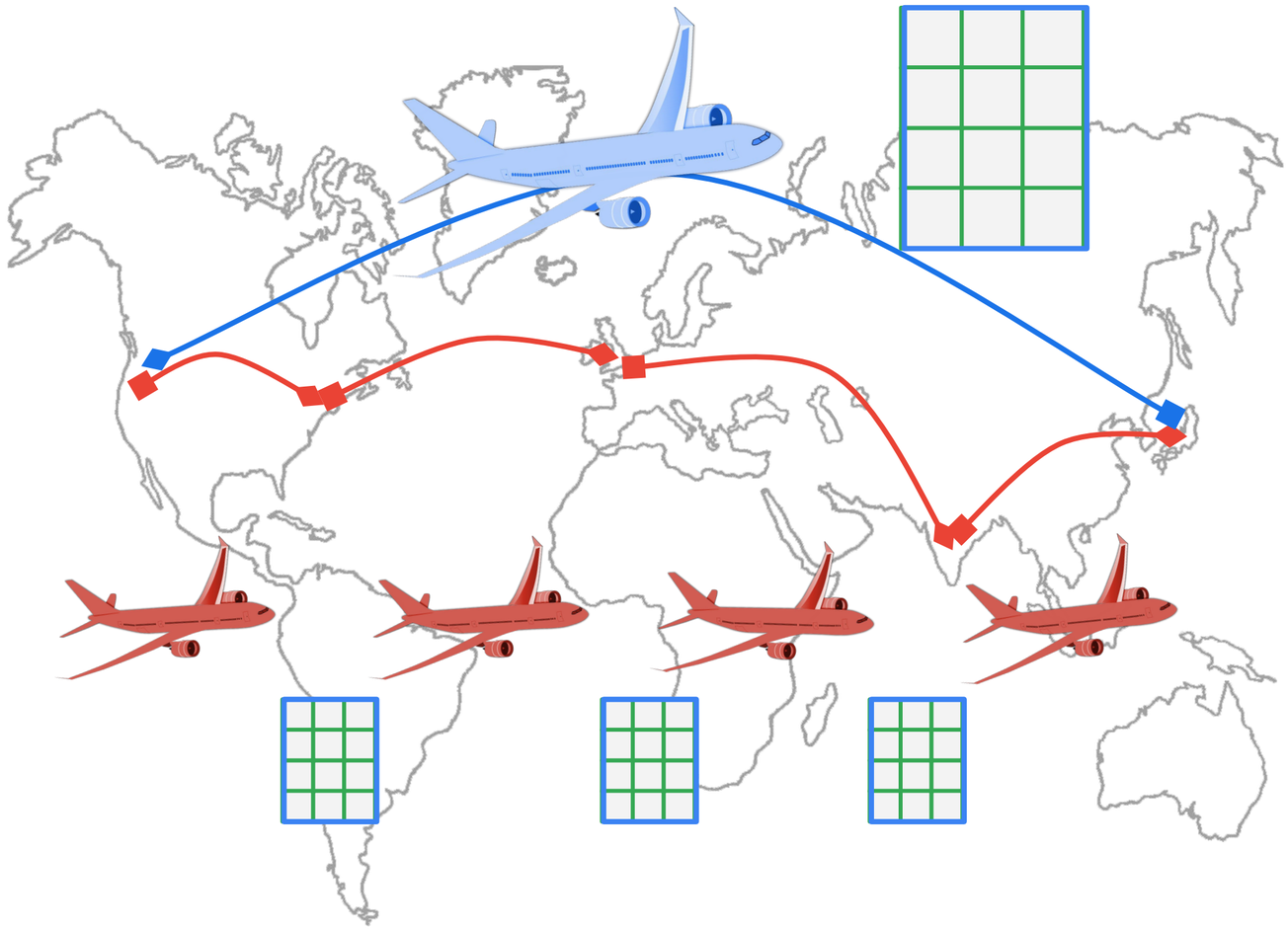
Proprietary + Confidential

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.

Proprietary + Confidential

Break queries into stages using intermediate tables

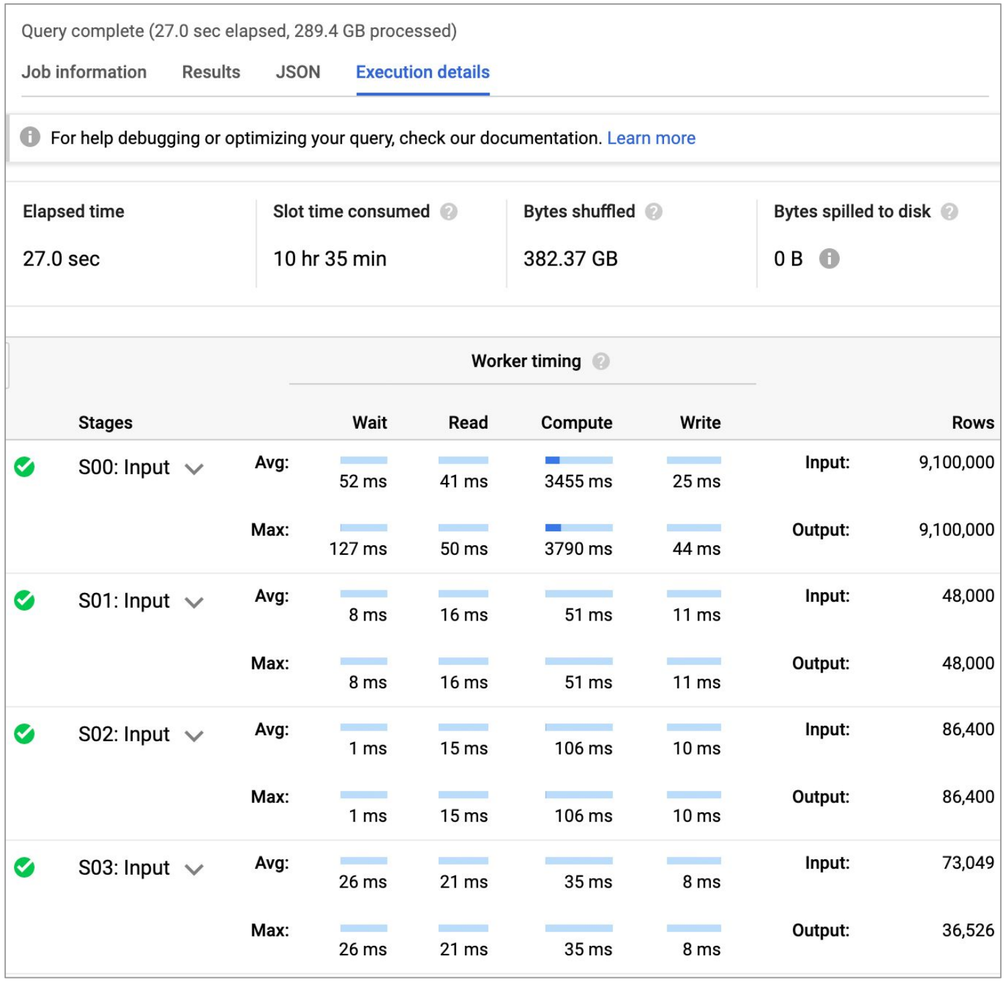


Read a huge dataset and process it.

Read only what is needed and store results at each step.

Proprietary + Confidential

Track input and output counts with Execution details tab



Query complete (27.0 sec elapsed, 289.4 GB processed)

Job information | Results | JSON | Execution details

For help debugging or optimizing your query, check our documentation. Learn more

Elapsed time	Slot time consumed	Bytes shuffled	Bytes spilled to disk
27.0 sec	10 hr 35 min	382.37 GB	0 B

Stages		Worker timing: Wait	Read	Compute	Write		Rows
S00: Input	Avg:	52 ms	41 ms	3455 ms	25 ms	Input:	9,100,000
	Max:	127 ms	50 ms	3790 ms	44 ms	Output:	9,100,000
S01: Input	Avg:	8 ms	16 ms	51 ms	11 ms	Input:	48,000
	Max:	8 ms	16 ms	51 ms	11 ms	Output:	48,000
S02: Input	Avg:	1 ms	15 ms	106 ms	10 ms	Input:	86,400
	Max:	1 ms	15 ms	106 ms	10 ms	Output:	86,400
S03: Input	Avg:	26 ms	21 ms	35 ms	8 ms	Input:	73,049
	Max:	26 ms	21 ms	35 ms	8 ms	Output:	36,526

Google Cloud

Proprietary + Confidential

Using BigQuery plans to optimize

Significant difference between avg and max time?

Probably data skew 

- Check with APPROX_TOP_COUNT
- Filter early as a workaround

Most time spent reading during intermediate stages?

Order of operations? 

- Consider filtering earlier in the query

Most time spent on CPU tasks?

Probably slow code 

- Look carefully at UDFs
- Use approximate functions
- Filter earlier in the query

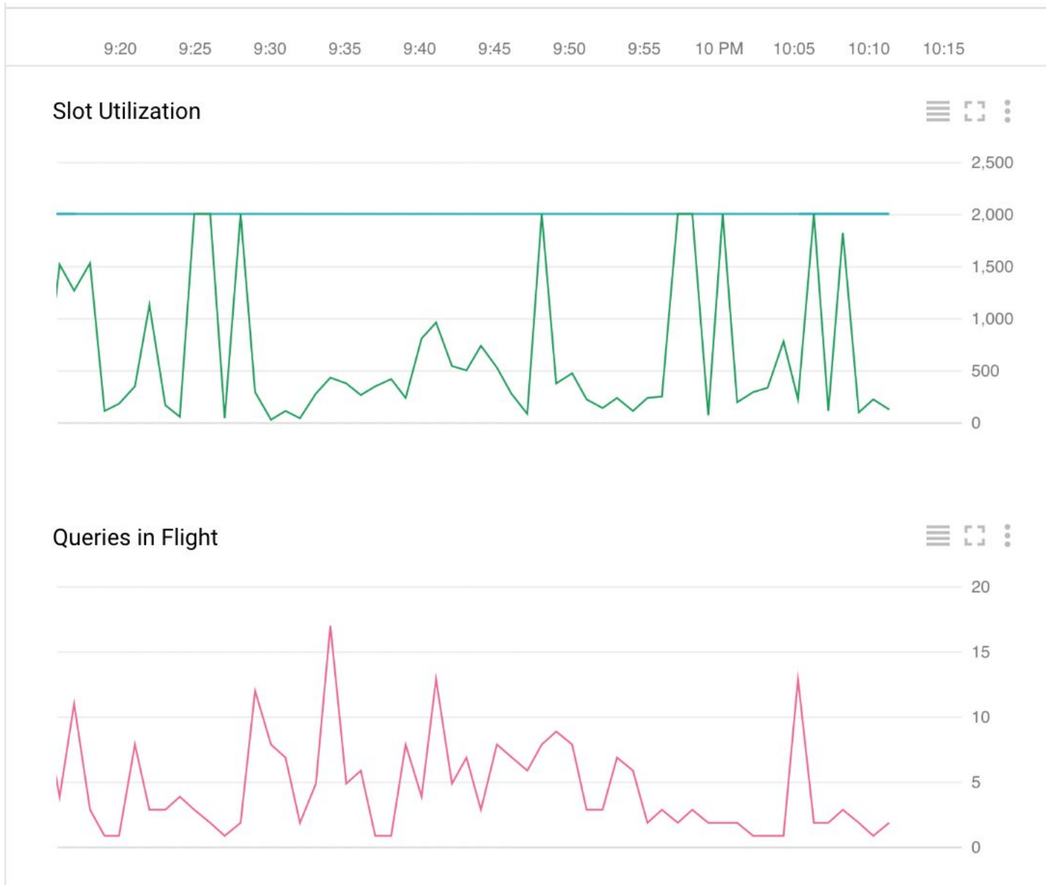
Proprietary + Confidential

Analyze BigQuery performance in Cloud Monitoring

Custom Dashboards

Metrics include:

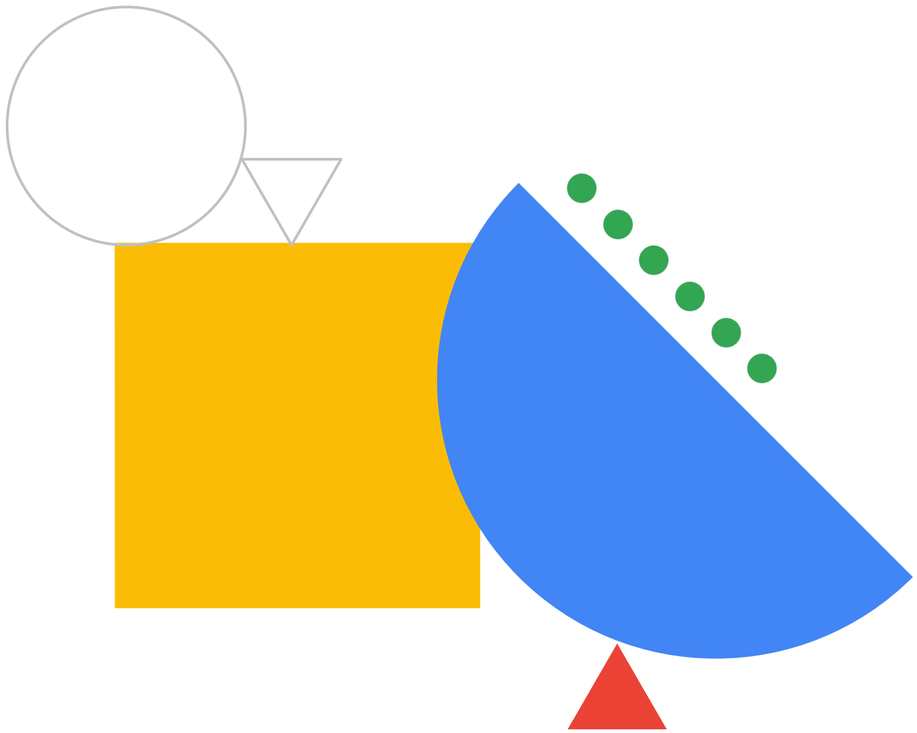
- slots utilization
- queries in flight
- upload bytes
- stored bytes



Proprietary + Confidential

Lab Intro

Optimizing your BigQuery Queries for Performance



Proprietary + Confidential

Lab objectives

01 Minimizing I/O

02 Caching results of previous queries

03 Performing efficient joins

04 Avoid overwhelming single workers

05 Using approximate aggregation functions

Proprietary + Confidential

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:

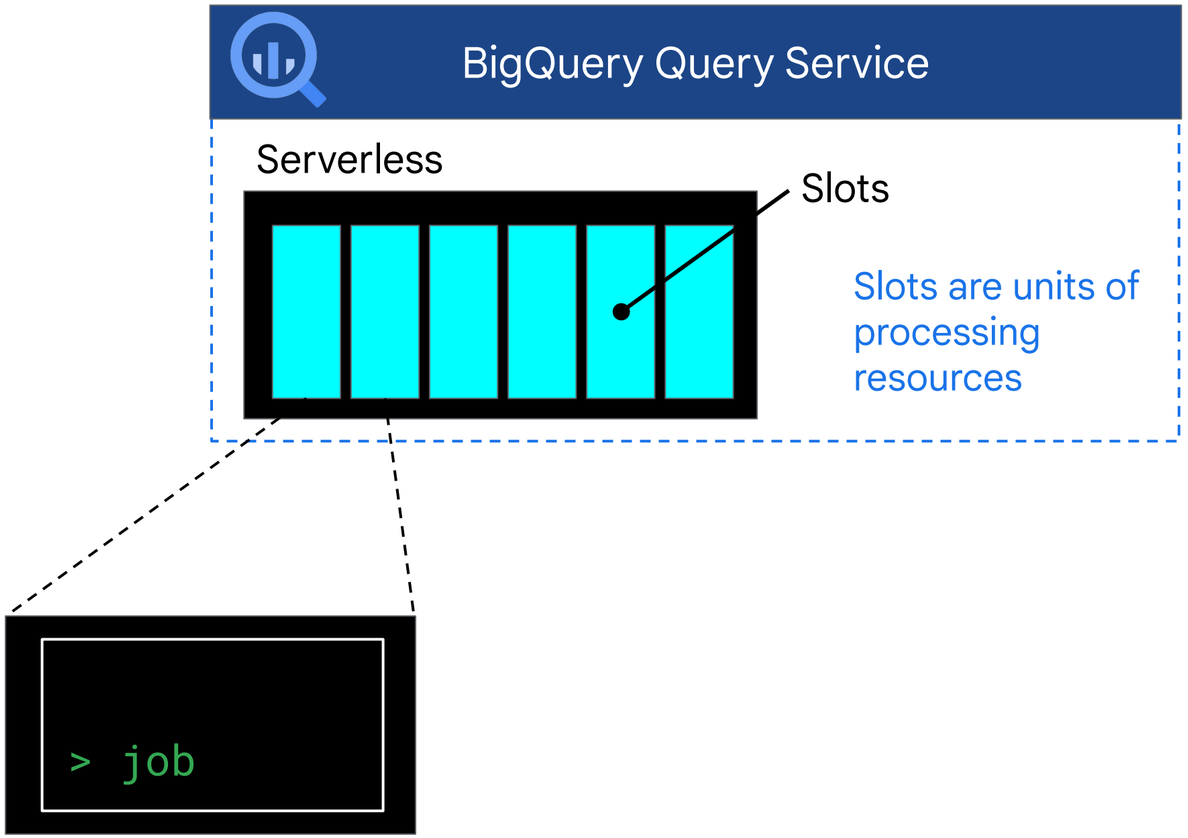
- 100 MB for half a month, you pay $0.001 (a tenth of a cent)
- 500 GB for half a month, you pay $5 ($0.02/GB per month)
- 1 TB for a full month, you pay $20

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

Proprietary + Confidential

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



Proprietary + Confidential

Which BigQuery pricing model to pick?

On-Demand 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

Good for **exploratory work** and discovery

Flat-Rate Pricing

- 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 **multiple large workloads** and **consistent monthly billing**

Proprietary + Confidential

Flex Slots provide flexibility and control

- Deploy BigQuery Slots in minutes
- Only pay for what you consume
- Cancel any time after 60 seconds
- Combine with longer-term commitments

Proprietary + Confidential

Flex Slots enable you to scale

- Planning for major calendar events, such as the tax season, Black Friday, popular media events, and video game launches.
- Meeting cyclical periods of high demand for analytics, like Monday mornings.
- Completing data warehouse evaluations and dialing in the optimal number of slots to use.

Proprietary + Confidential

Considerations when enrolling for flat-rate pricing

- Flex slots are a special commitment type and a subject to capacity availability.
- Monthly commitments cannot be canceled or downgraded for 30 calendar days from the purchase confirmation date.
- Annual commitments cannot be canceled or downgraded for one calendar year.
- To purchase additional BigQuery slots, you must enter into a new commitment.
- Flat-rate pricing is purchased for a specific BigQuery location.
- Flat-rate and on-demand pricing can be used together.
- To discontinue a flat-rate pricing plan, you must cancel or downgrade your commitment.

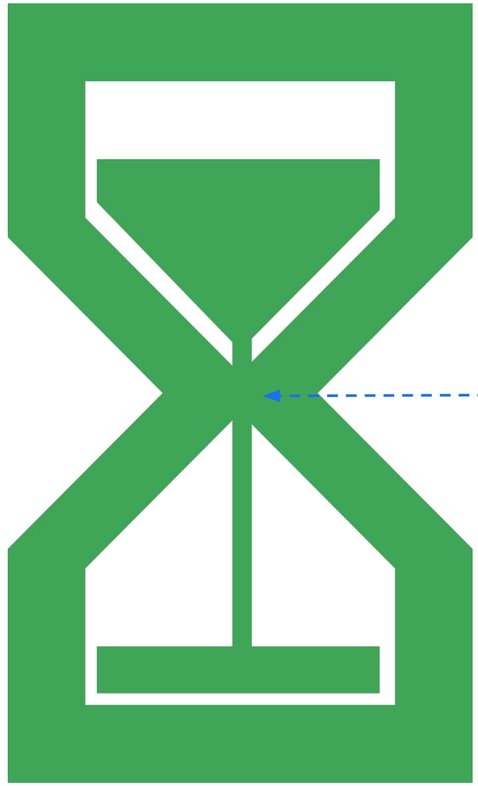
Proprietary + Confidential

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)



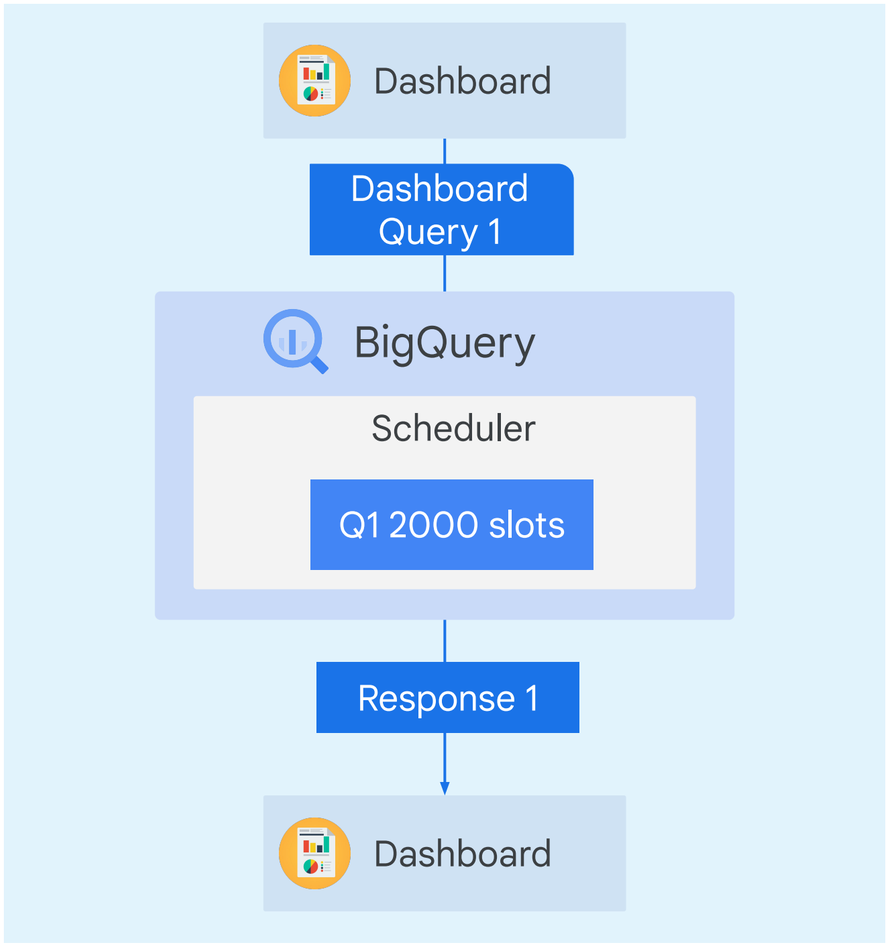
Slots available

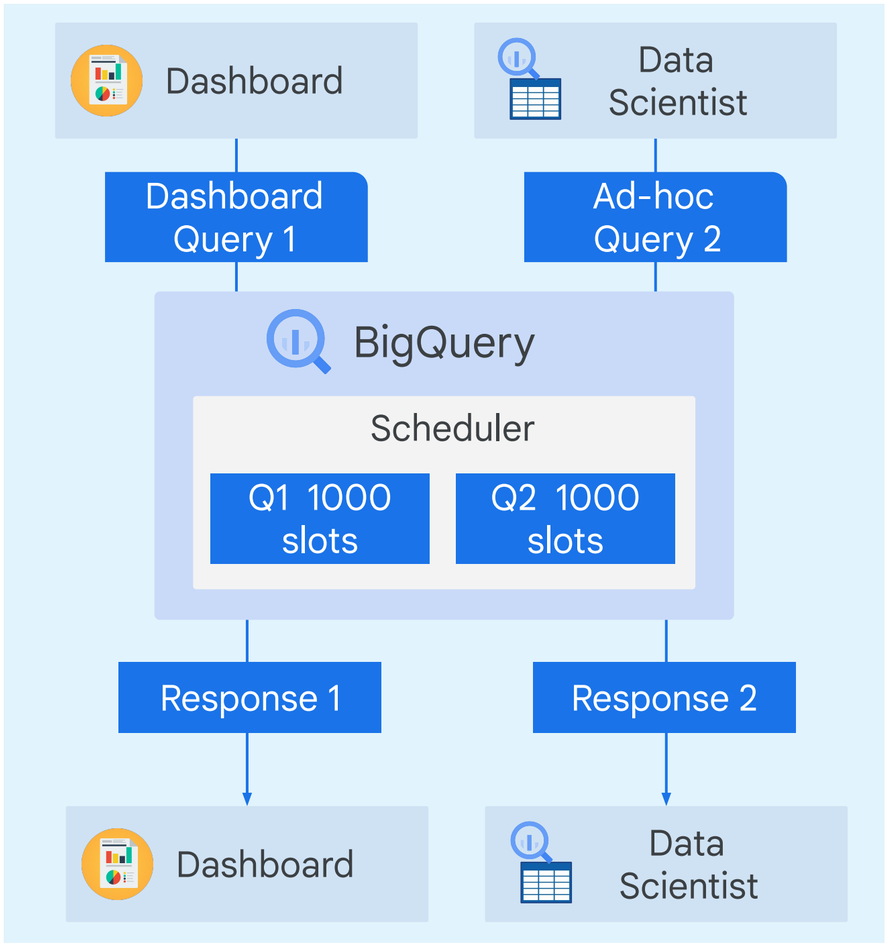
More slots = Faster depletion rate

More concurrent queries = Slower depletion rate

Proprietary + Confidential

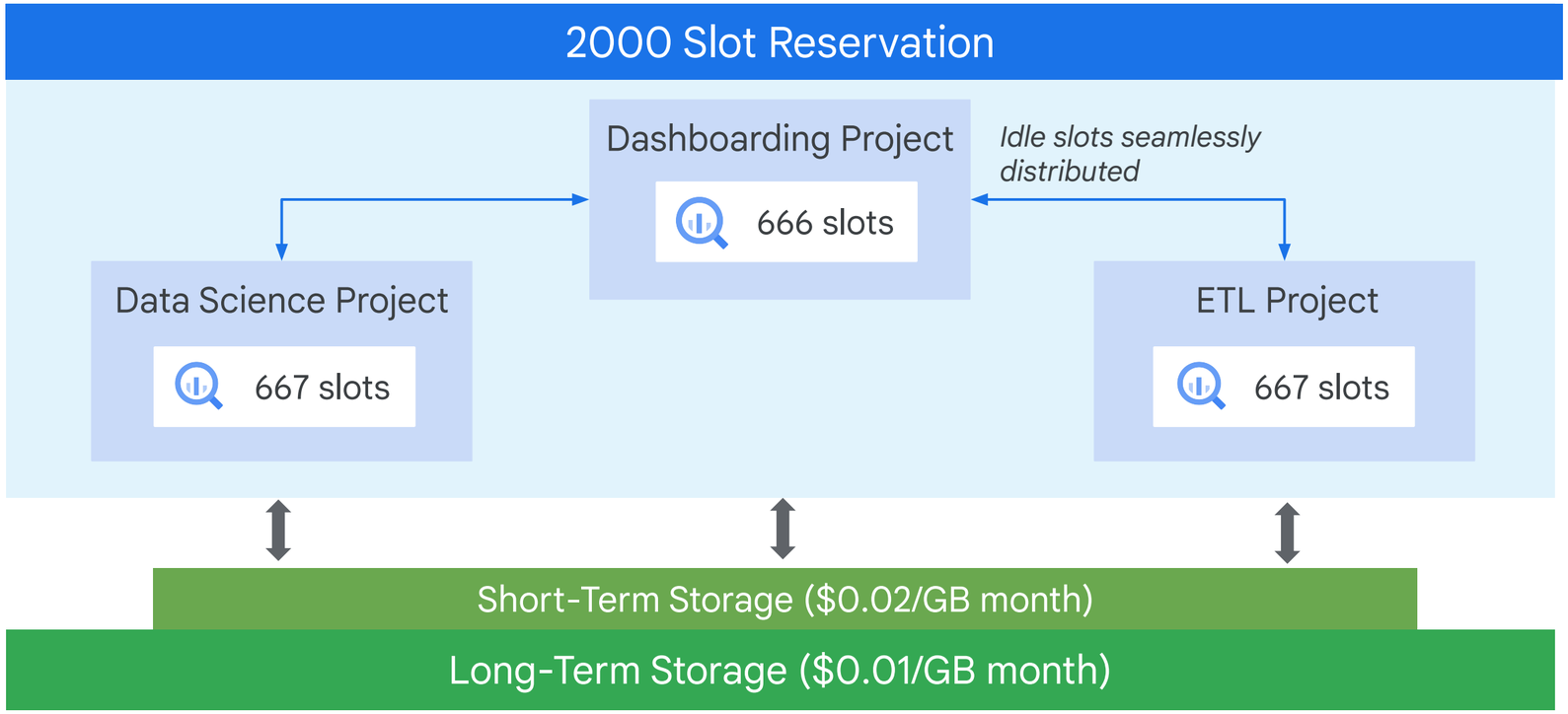
BigQuery has a fair scheduler





Proprietary + Confidential

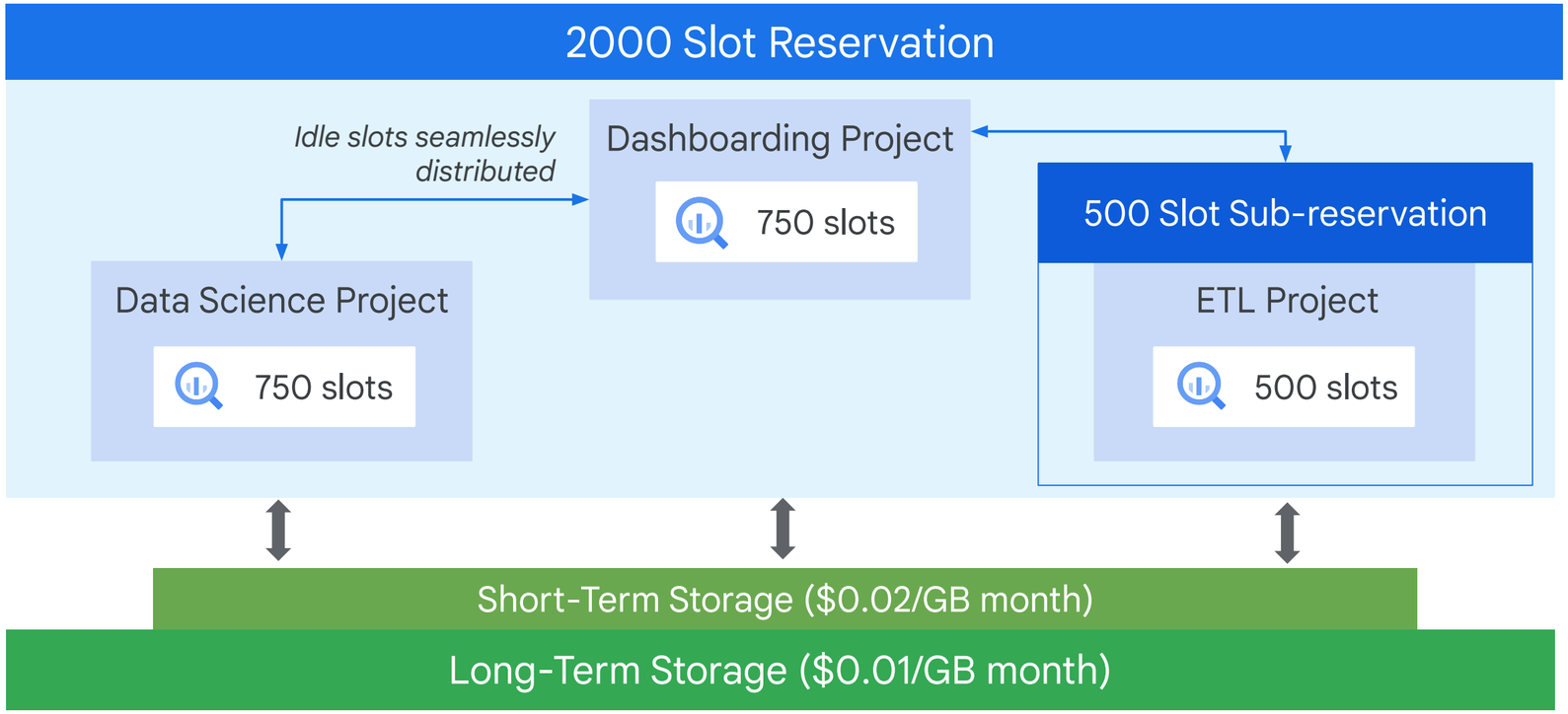
Concurrency is fair across projects, users, and queries, with unused capacity fairly divided among existing tasks



No duplication of storage

Proprietary + Confidential

To prioritize projects, set up a hierarchical reservation

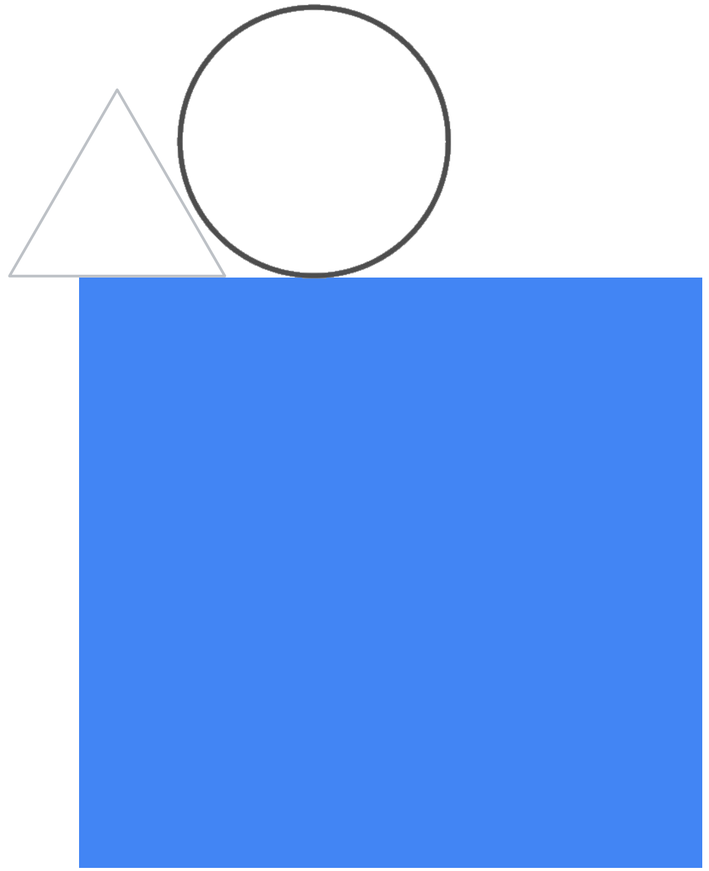


No duplication of storage

Proprietary + Confidential

Lab Intro

Partitioned Tables in BigQuery



Proprietary + Confidential

Lab objectives

01 Query partitioned datasets

02 Create your own dataset partitions to improve query performance and reduce cost

Proprietary + Confidential