Module 13

Optimizing for Performance

In this module we will:

- **Avoid BigQuery Performance Pitfalls**
- Prevent Hotspots in your Data
- Diagnose Performance Issues with the Query Explanation map

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

This is one of the modules that everyone loves - how to make your queries run faster and save on data processing costs. In the performance optimization module we will cover the key types of work that BigQuery does on the back-end and how you can avoid falling into performance traps.

Then we’ll cover one of the most common data traps which is having too much data skewed to a few values.

Last up is your best tool for analyzing and debugging performance issues -- the query explanation colored map.

Let's get started

Four Key Elements of Work

1. **Input / Output** — How many bytes did you read? Write to storage?
2. **Communication between slots (shuffle)** — How many bytes did you pass to the next stage?
3. **CPU work** — User-defined functions (UDFs), functions
4. **SQL Syntax** — Is there a more efficient way to write your query?

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

Image (race) cc0: https://unsplash.com/search/race?photo=HHunRG19kF8

Avoid Input / Output Wastefulness

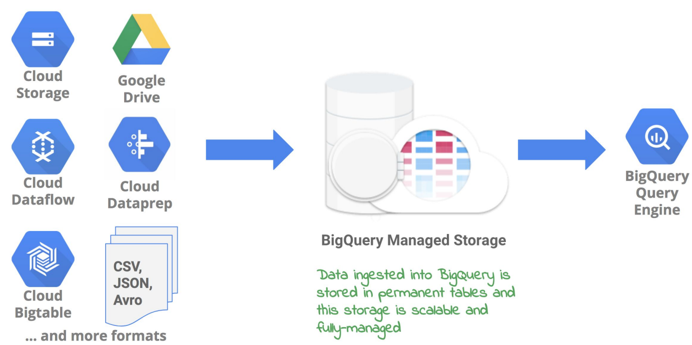
- Do not SELECT *, use only the columns you need
- Denormalize your schemas and take advantage of nested and repeated fields
- Use granular suffixes in your table wildcards for more specificity

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

Image (filter) cc0: https://unsplash.com/search/photos/filter?photo=1pZbNwIGzNY

Use BigQuery native storage for the best performance

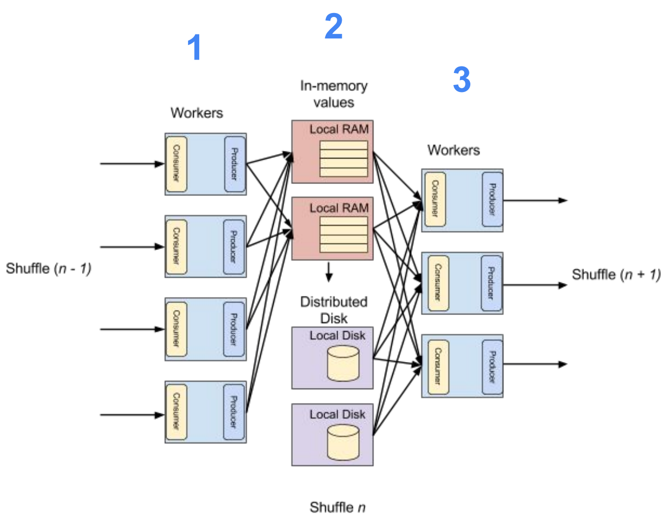


- External direct data connections can never be cached
- Live edits to underlying external sources (e.g. spreadsheets) could create race conditions
- Native BigQuery tables have intelligence built-in like automatic predicate pushdown

Google Cloud

Predicate pushdown = BigQuery will re-write the SQL statement you passed to it and apply your WHERE clause earlier in your query to save on bytes processed

Optimize communication between slots (via shuffle)



- Pre-filter your data before doing JOINs
- Many shuffle stages can indicate data partition issues (skew)

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

Do not use WITH clauses in place of materializing results

```
#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
    )
```

- Commonly filtering and transforming the same results? Store them into a permanent table
- WITH clause queries are not materialized and are re-queried if referenced more than once

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

Materialize your transformed results instead of re-querying

- Commonly filtering and transforming the same results? Store them into a permanent table

```
#standardSQL
SELECT
    totrevenue AS revenue,
    ein,
    operateschools170cd AS is_school
FROM
`bigquery-public-data.irs_990.irs_990_2015`
WHERE operateschools170cd = 'Y'
```

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Be careful using GROUP BY across many distinct values

- Best when the number of distinct groups is small (fewer shuffles of data).
- Grouping by a high-cardinality unique ID is a bad idea.

Row	contributor_id	LogEdits
1	2221364	4
2	104574	4
3	73576	4
4	311307	4
5	291919	4
6	140178	4
7	181636	4
8	3661553	4
9	3600820	4
10	4737290	4
11	938404	4
12	295955	4
13	183812	4
14	1811786	4
15	8918196	4
16	561624	4
17	5338406	4

← Do not Group on an ID

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

Large group by = many forced shuffle steps

Reduce Javascript UDFs to Reduce Computational Load

- Javascript UDFs require BigQuery to launch a Java subprocess to run
- Use native SQL functions whenever possible
- Concurrent rate limits:
  - for non-UDF queries: 50
  - for UDF-queries: **6**

```
CREATE TEMP FUNCTION SumFieldsNamedFoo(json_row STRING)
  RETURNS FLOAT64
  LANGUAGE js AS """
function SumFoo(obj) {
  var sum = 0;
  for (var field in obj) {
    if (obj.hasOwnProperty(field) && obj[field] != null) {
      if (typeof obj[field] == "object") {
        sum += SumFoo(obj[field]);
      } else if (field == "foo") {
        sum += obj[field];
      }
    }
  }
  return sum;
}
var row = JSON.parse(json_row);
return SumFoo(row);
""";
```

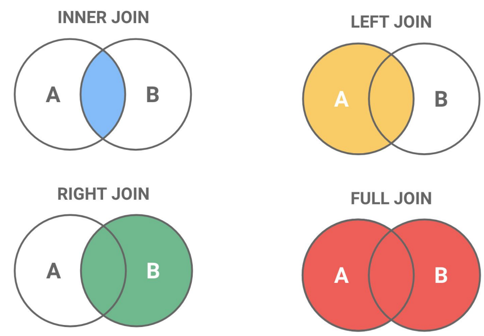
© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

https://cloud.google.com/bigquery/docs/reference/standard-sql/user-defined
-functions

Understand your data model before applying Joins and Unions

- Know your join conditions and if they're unique -- no accidental cross joins
- Filter wildcard UNIONS with _TABLE_SUFFIX filter
- Do not use self-joins (consider window functions instead)



© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Push intensive operations to the end of the query

```sql
#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)
  ORDER BY name DESC
    ),

  # 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
    )
```

- Large sorts should be the last operation in your query
- If you need to sort early, filter or limit your data before sorting

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Module 13

Optimizing for Performance

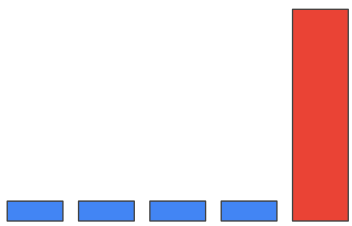
In this module we will:

- Avoid BigQuery Performance Pitfalls
- **Prevent Hotspots in your Data**
- Diagnose Performance Issues with the Query Explanation map

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

Shuffle Wisely: **Be Aware of Data Skew** in your Dataset



Skewed Data creates an imbalance between BigQuery worker slots (uneven data partition sizes)

- **Filter your dataset** as early as possible (this avoids overloading workers on JOINs)
- Hint: Use the Query Explanation map and compare the Max vs the Avg times to highlight skew
- BigQuery will automatically attempt to reshuffle workers that are overloaded with data

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

If you have heavy skew to a few values, filter early. This is because one worker (slot) has to hold all the values in memory (2TB compressed) and excess will spill onto disk.

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

Walk through example using the Query Explanation map: https://cloud.google.com/bigquery/query-plan-explanation#data_skew_in_stage_1

Module 13

Optimizing for Performance

In this module we will:

- Avoid BigQuery Performance Pitfalls
- Prevent Hotspots in your Data
- **Diagnose Performance Issues with the Query Explanation map**

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

Diagnose Performance Issues with the **Query Explanation** Map

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

Walkthrough example using the Query Explanation map:
https://cloud.google.com/bigquery/query-plan-explanation#data_skew_in_stage_1

Diagnose Performance Issues with the **Query Explanation** Map

The following ratios are also available for each stage in the query plan.

API JSON Name	Web UI*	Ratio Numerator **
`waitRatioAvg`	AVG	Time the average worker spent waiting to be scheduled.
`waitRatioMax`	MAX	Time the slowest worker spent waiting to be scheduled.
`readRatioAvg`	AVG	Time the average worker spent reading input data.
`readRatioMax`	MAX	Time the slowest worker spent reading input data.
`computeRatioAvg`	AVG	Time the average worker spent CPU-bound.
`computeRatioMax`	MAX	Time the slowest worker spent CPU-bound.
`writeRatioAvg`	AVG	Time the average worker spent writing output data.
`writeRatioMax`	MAX	Time the slowest worker spent writing output data.

* The labels 'AVG' and 'MAX' are for illustration only and do not appear in the web UI.

** All of the ratios share a common denominator that represents the longest time spent by any worker in any segment.

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

Example: Large GROUP BY

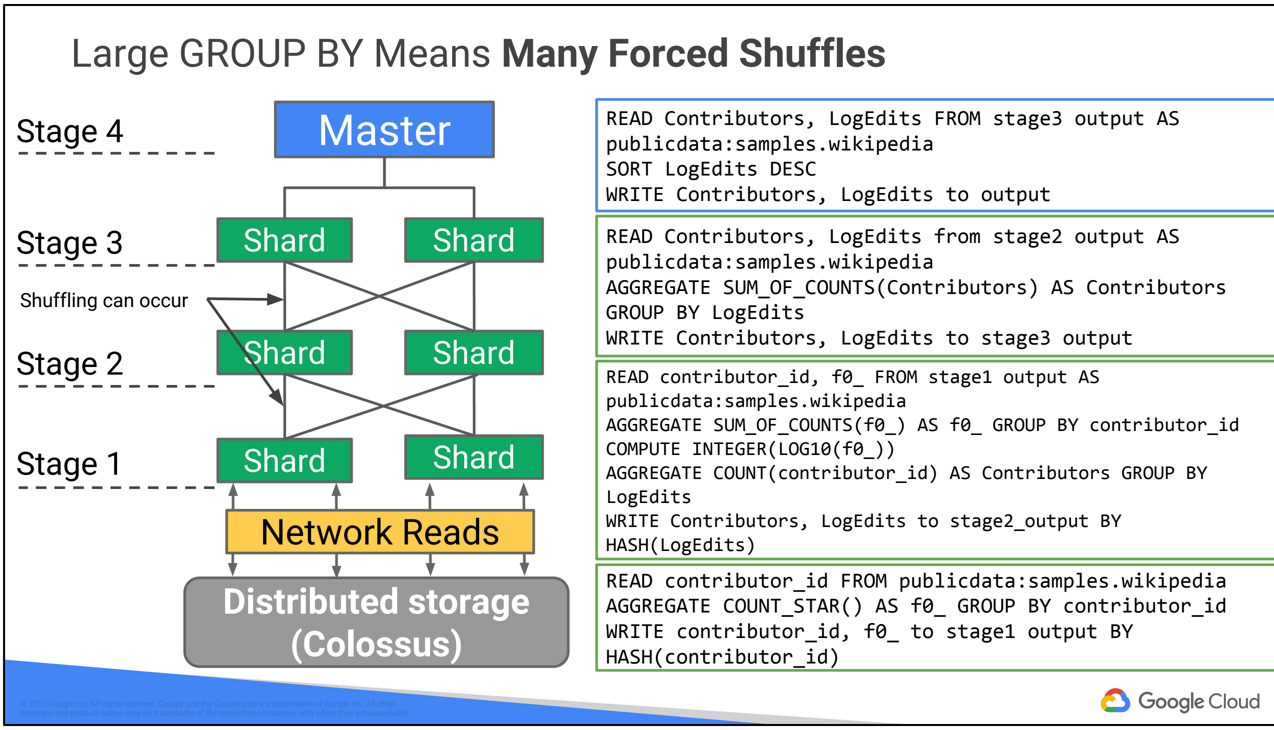
Large GROUP BY query

```
SELECT
  LogEdits, COUNT(contributor_id) AS Contributors
FROM (
  SELECT
    contributor_id,
    CAST(LOG10(COUNT(*)) AS INT64) LogEdits # Buckets user edits
  FROM `publicdata.samples.wikipedia`
  GROUP BY contributor_id)
GROUP BY LogEdits
ORDER BY LogEdits DESC
```

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

This is a more complicated query to find out how many authors contribute to wikipedia by the number of edits that he/she made. The LOG10(COUNT(*)) is a way to broadly categorize the number of edits into buckets. For example, if a user contributed 120 edits, then INTEGER(LOG10(120)) will yield 2. Similarly, if a user contributed 185 edits, he will be grouped into the same bucket. If a user contributed 1500 edits, then INTEGER(LOG10(1500) will yield 3. So, the inner SELECT statement groups each contributor into buckets and the final SELECT statement counts the number of contributors for each bucket.



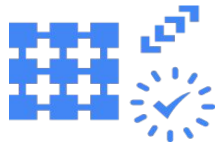
Because the number of contributors to Wikipedia is huge, the INNER SELECT used 'GROUP BY'. The large GROUP BY triggered shuffling between stages. Shuffling can be viewed as 'partitioning'. Shuffling guarantees that all records which have the same value will be stored in the same shard. This allows those operations to be performed efficiently. In the example, each contributor will be shuffled to the same shard. Assuming the distribution is quite even, the implication is that each shard will have fewer contributors to process, hence less memory consumption.

Table Sharding - Then and Now

Traditional Partitioning

Traditional databases get performance boost by partitioning very large tables.Usually requires an administrator to pre-allocate space, define partitions, and maintain them

Sharding Tables Manually (Better)



Manual **Table sharding** divides big table into smaller tables with new suffix of YYYYMMDD

Queries use Table Wildcard functions

Date-Partitioned Tables (Best)

order_id	order_date (partitioned column)
123	2017-01-01
456	2017-03-01
789	2016-02-01
101	1999-06-01
121	1999-11-01

Date Partition a single table based on specified day or a Date Column

Creating Partitioned Tables

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

Dividing a dataset into daily tables helped to reduce the amount of data scanned when querying a specific date range. For example, if you have a year's worth of data in a single table, a query that involves the last seven days of data still requires a full scan of the entire table to determine which data to return. However, if your table is divided into daily tables, you can restrict the query to the seven most recent daily tables.

Daily tables, however, have several disadvantages. You must manually, or programmatically, create the daily tables. SQL queries are often more complex because your data can be spread across hundreds of tables. Performance degrades as the number of referenced tables increases. There is also a limit of 1,000 tables that can be referenced in a single query.

How to create partitioned tables:
https://cloud.google.com/bigquery/docs/creating-partitioned-tables

Filtering on date-partitioned tables in SQL

```
SELECT
  order_id
FROM
  `mydataset.orders`
WHERE
  _PARTITIONTIME BETWEEN
  TIMESTAMP('2016-01-01')
  AND
  TIMESTAMP('2018-01-01');
```

order_id	order_date (partitioned column)
123	2017-01-01
456	2017-03-01
789	2016-02-01
101	1999-06-01
121	1999-11-01

Date Partition a single table based on specified day or a Date Column

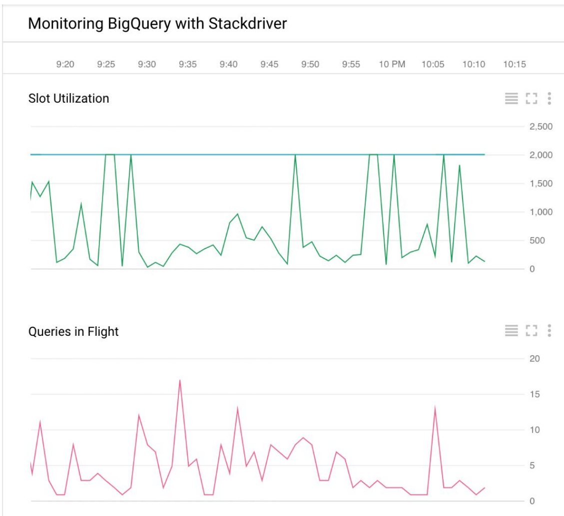
Creating Partitioned Tables

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

How to create partitioned tables:
https://cloud.google.com/bigquery/docs/creating-partitioned-tables

Monitor Performance with Stackdriver



- Available for all BigQuery customers
- Fully interactive GUI. Customers can create custom dashboards displaying up to 13 BigQuery metrics, including:
    - Slots Utilization
    - Queries in Flight
    - Uploaded Bytes (not shown)
    - Stored Bytes (not shown)

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

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

The Stackdriver charting tools offer

- Graphical User Interface to create custom dashboards for multiple GCP Products
- virtually real time data on many parameters (the lag on slot utilization for example is less than 5 minutes)
- Interactive graphical controls (zooming, creating new charts, selecting display modes, etc)

Known Issues:

- There is a known issue when Stackdriver reports slots available for customers that have subreservations. Please direct any questions to me.

Enable billing logs exports to BigQuery for more monitoring

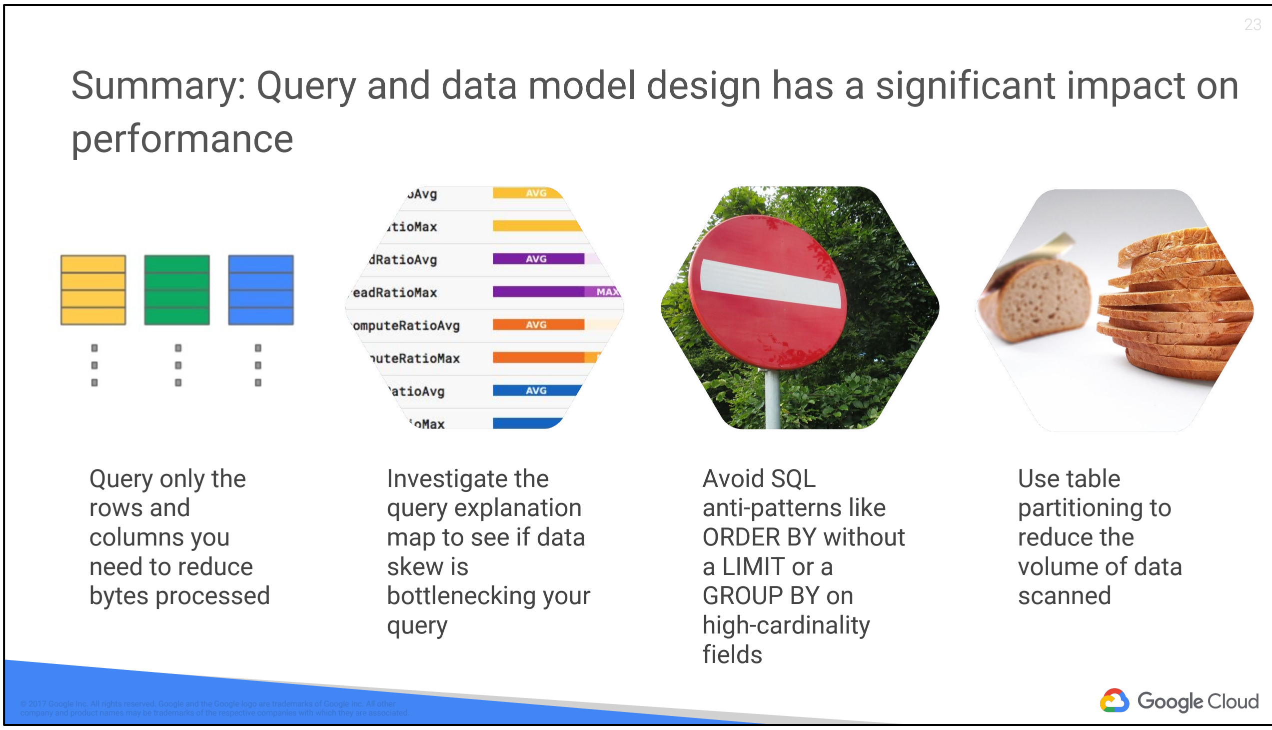
After this course, try exporting BigQuery logs using this *tutorial* *to recreate the above Data Studio billing* *dashboard*

- Monitor and track individual query logs visually
- Requires billing logs export to be enabled first (do this as early as possible to get the most data)

BigQuery Monitoring Tutorial

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud



Writing fast queries may not come naturally at first. Since you're paying for the total bytes processed, you want to first limit the columns of data you want returned and second consider using WHERE filters wherever possible. This doesn't mean you can't write queries that process your entire dataset, BigQuery was built for to be petabyte-scale, but just be mindful with your resources.

Next up we covered the Query Explanation map where you get visual cues of what types of work is most demanded by your queries. Note that a large difference between the average (dark bar graph color) and the max (lighter color) could indicate heavy skew inside of your dataset which can partially be solved by hashing or shuffling your identifying fields.

Then we covered SQL bad behavior like SELECTing every record and ordering it without a limit clause.

Finally, if you're only accessing recent data like the most recent events consider using table partitioning to reduce the bytes scanned by your query.

Let's troubleshoot some under-performing queries in our next lab.

Image (wrong way) cc0:
https://pixabay.com/en/traffic-sign-no-entry-do-not-enter-992648/
Image (sliced bread) cc0:
https://pixabay.com/en/bread-slice-of-bread-knife-cut-534574/

Lab 11
Optimizing and Troubleshooting Query Performance

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

Lab 11 in Qwiklabs

Optimizing and Troubleshooting Query Performance

In this lab, you will fix and troubleshoot SQL queries for performance improvements.

© 2017 Google Inc. All rights reserved. Google and the Google logo are trademarks of Google Inc. All other company and product names may be trademarks of the respective companies with which they are associated.

Google Cloud

Image (car repair) cc0:
https://pixabay.com/en/car-repair-car-workshop-repair-shop-362150/