BigQuery Demonstration notebook

- Uses BigQuery Python API
- Demonstrates costs of various different approaches to optimizing BigQuery usage

In [1]:
```
from google.cloud import bigquery
```

In [2]:
```
project = 'redapt-aa-sandbox'
```

In [3]:
```
# Create client connection to GCP Project
client = bigquery.Client(project=project)
```

Create a reusable method to estimate query costs and compare to other queries:

- the job_config sets up:
  - dry-run=True to only preview the data that will be scanned by the query
  - use_query_cache=False, ignores query cache.
- estimated costs are based on $5/TB data scanned

In [10]:
```
def query_bytes(query, compare_query=None):
    # create job configuration to use dry-run and not use query cache
    job_config = bigquery.QueryJobConfig(dry_run=True, use_query_cache=False)

    # dry_run of query
    query_job = client.query(
        (query), job_config=job_config
    )

    # calculate gigabytes processed from bytes processed results of the query dr
    gigabytes_processed = round(query_job.total_bytes_processed/1024/1024/1024,3

    # calculate the cost per month as running query run once per day for 30 days
    cost_per_month = round((((gigabytes_processed)*30)/1024)* 5, 3)

    # display results
    print(f"This query will process {gigabytes_processed} gigabytes")
    print(f"${cost_per_month} per month")

    # if camparison query provided, do the same for the comparison query
    if compare_query:
        compare_job = client.query(compare_query, job_config=job_config)
        compare_gigabytes = round(compare_job.total_bytes_processed/1024/1024/10
        compare_per_month = round((((compare_gigabytes)*30)/1024)*5,3)
        print(f"Compared to {compare_gigabytes} gigabytes")
        print(f"and ${compare_per_month} per month")
```

First a look at our initial tables

bigquery-public-data.stackoverflow

In [4]:
```
# get the bigquery public stackoverflow dataset
so_dataset_ref = client.dataset('stackoverflow', project='bigquery-public-data')

# get the posts_questions table
post_questions_table = client.get_table(so_dataset_ref.table('posts_questions'))

# show the schema of this table
post_questions_table.schema
```

Out[4]:
```
[SchemaField('id', 'INTEGER', 'NULLABLE', None, (), None),
 SchemaField('title', 'STRING', 'NULLABLE', None, (), None),
 SchemaField('body', 'STRING', 'NULLABLE', None, (), None),
 SchemaField('accepted_answer_id', 'INTEGER', 'NULLABLE', None, (), None),
 SchemaField('answer_count', 'INTEGER', 'NULLABLE', None, (), None),
 SchemaField('comment_count', 'INTEGER', 'NULLABLE', None, (), None),
 SchemaField('community_owned_date', 'TIMESTAMP', 'NULLABLE', None, (), None),
 SchemaField('creation_date', 'TIMESTAMP', 'NULLABLE', None, (), None),
 SchemaField('favorite_count', 'INTEGER', 'NULLABLE', None, (), None),
 SchemaField('last_activity_date', 'TIMESTAMP', 'NULLABLE', None, (), None),
 SchemaField('last_edit_date', 'TIMESTAMP', 'NULLABLE', None, (), None),
 SchemaField('last_editor_display_name', 'STRING', 'NULLABLE', None, (), None),
 SchemaField('last_editor_user_id', 'INTEGER', 'NULLABLE', None, (), None),
 SchemaField('owner_display_name', 'STRING', 'NULLABLE', None, (), None),
 SchemaField('owner_user_id', 'INTEGER', 'NULLABLE', None, (), None),
 SchemaField('parent_id', 'STRING', 'NULLABLE', None, (), None),
 SchemaField('post_type_id', 'INTEGER', 'NULLABLE', None, (), None),
 SchemaField('score', 'INTEGER', 'NULLABLE', None, (), None),
 SchemaField('tags', 'STRING', 'NULLABLE', None, (), None),
 SchemaField('view_count', 'INTEGER', 'NULLABLE', None, (), None)]
```

In [5]:
```
count_posts = """
SELECT count(*) as tot_rows
FROM `bigquery-public-data.stackoverflow.posts_questions`
"""
count_config = bigquery.QueryJobConfig(use_query_cache=False)

count_job = client.query(
        (count_posts), job_config=count_config
)

for row in count_job:
    print(f"Rows: {row.tot_rows}")
```

```
Rows: 20890054
```

bigquery-public-data.wiki.pageviews_2020 -> redapt-aa-sandbox.demo_dataset.pageviews_2020

In [7]:
```
# get the bigquery public stackoverflow dataset
wiki_dataset_ref = client.dataset('demo_dataset', project='redapt-aa-sandbox')

# get the posts_questions table
pageviews_table = client.get_table(wiki_dataset_ref.table('pageviews_2020'))

# show the schema of this table
pageviews_table.schema
```

Out[7]:
```
[SchemaField('datehour', 'TIMESTAMP', 'NULLABLE', None, (), None),
 SchemaField('wiki', 'STRING', 'NULLABLE', None, (), None),
 SchemaField('title', 'STRING', 'NULLABLE', None, (), None),
 SchemaField('views', 'INTEGER', 'NULLABLE', None, (), None)]
```

In [8]:
```
count_posts = """
SELECT count(*) as tot_rows
FROM `redapt-aa-sandbox.demo_dataset.pageviews_2020`
"""
count_config = bigquery.QueryJobConfig(use_query_cache=False)

count_job = client.query(
        (count_posts), job_config=count_config
)

for row in count_job:
    print(f"Rows: {row.tot_rows}")
```

```
Rows: 55917037333
```

Baseline query: SELECT *

This will be our baseline query to compare other queries against.

- First will query a small table (@33GB)
- This table is not optimized in any way.

In [11]:
```
select_star_posts = """
SELECT *
FROM `bigquery-public-data.stackoverflow.posts_questions`
LIMIT 100
"""

query_bytes(select_star_posts)
```

```
This query will process 33.141 gigabytes
$4.855 per month
```

- and then a larger dataset (@2TB)
- this table is not optimized in any way, either

In [12]:
```
select_star_pageviews = """
SELECT *
FROM `redapt-aa-sandbox.demo_dataset.pageviews_2020`
LIMIT 100
"""

query_bytes(select_star_pageviews)
```

```
This query will process 2308.184 gigabytes
$338.113 per month
```

SELECT * with WHERE clause

If we add a WHERE statement to our select, we can see we do not have gains in reducing out costs on non-optimized BigQuery table if we do a SELECT *

In [13]:
```
select_star_posts_where = """
SELECT *
FROM `bigquery-public-data.stackoverflow.posts_questions`
WHERE comment_count = 25
LIMIT 100
"""

query_bytes(select_star_posts_where, select_star_posts)
```

```
This query will process 33.141 gigabytes
$4.855 per month
Compared to 33.141 gigabytes
and $4.855 per month
```

In [14]:
```
select_star_pageviews_where = """
SELECT *
FROM `redapt-aa-sandbox.demo_dataset.pageviews_2020`
WHERE views = 100
LIMIT 100
"""

query_bytes(select_star_pageviews_where, select_star_pageviews)
```

```
This query will process 2308.184 gigabytes
$338.113 per month
Compared to 2308.184 gigabytes
and $338.113 per month
```

SELECT * is a BigQuery anti-pattern

Reduce columns queried

- if we take the same queries as we ran above and only select the columns we are interested in, look at the difference it makes

First without a WHERE clause:

In [15]:
```
select_cols_post = """
SELECT id, view_count
FROM `bigquery-public-data.stackoverflow.posts_questions`
LIMIT 100
"""

query_bytes(select_cols_post, select_star_posts)
```

```
This query will process 0.311 gigabytes
$0.046 per month
Compared to 33.141 gigabytes
and $4.855 per month
```

In [16]:
```
select_pageviews_cols_limits = """
```

```
SELECT wiki, views
FROM `redapt-aa-sandbox.demo_dataset.pageviews_2020`
LIMIT 100
"""

query_bytes(select_pageviews_cols_limits, select_star_pageviews)
```

```
This query will process 690.683 gigabytes
$101.174 per month
Compared to 2308.184 gigabytes
and $338.113 per month
```

- and then with a WHERE clause and compare it to a SELECT * with a WHERE clause

In [17]:

```
select_cols_posts = """
SELECT
CONCAT('https://stackoverflow.com/questions/',
       CAST(id as STRING)) as url,
view_count
FROM `bigquery-public-data.stackoverflow.posts_questions`
WHERE tags like '%google-bigquery%'
ORDER BY view_count DESC """

query_bytes(select_cols_posts, select_star_posts)
```

```
This query will process 0.843 gigabytes
$0.123 per month
Compared to 33.141 gigabytes
and $4.855 per month
```

In [18]:

```
select_pageviews_cols_where = """
SELECT wiki, views
FROM `redapt-aa-sandbox.demo_dataset.pageviews_2020`
WHERE views = 100
LIMIT 100
"""

query_bytes(select_pageviews_cols_where, select_star_pageviews)
```

```
This query will process 690.683 gigabytes
$101.174 per month
Compared to 2308.184 gigabytes
and $338.113 per month
```

Notice that adding a WHERE clause when selecting all columns doesn't reduce the size of the data request

Partitioning tables

A partitioned table allows BigQuery to organize the data into 'buckets' and we start to see how BigQuery can perform. What if we run a SELECT * on a partitioned table?

- this first query is on a table partitioned by comment counts comparing it to the same query on a non-partitioned table

In [19]:

```
select_star_posts_partitioned = """
SELECT * FROM `redapt-aa-sandbox.demo_dataset.posts_questions_partitioned`
LIMIT 100
"""

query_bytes(select_star_posts_partitioned, select_star_posts)
```

```
This query will process 33.141 gigabytes
$4.855 per month
Compared to 33.141 gigabytes
and $4.855 per month
```

In [20]:

```
select_star_pageviews_partitioned = """
SELECT *
FROM `redapt-aa-sandbox.demo_dataset.page_views_2020_partitioned`
LIMIT 100
"""

query_bytes(select_star_pageviews_partitioned, select_star_pageviews)
```

```
This query will process 2308.184 gigabytes
$338.113 per month
Compared to 2308.184 gigabytes
and $338.113 per month
```

But look what happens now when we add a WHERE clause on our partition and compare it to a query with a WHERE clause on a non-partitioned table

SELECT * with WHERE clause on partitioned table

In [21]:

```
select_star_posts_where_partitioned = """
SELECT * FROM `redapt-aa-sandbox.demo_dataset.posts_questions_partitioned`
WHERE comment_count = 25
LIMIT 100
"""

query_bytes(select_star_posts_where_partitioned, select_star_posts_where)
```

```
This query will process 0.003 gigabytes
$0.0 per month
Compared to 33.141 gigabytes
and $4.855 per month
```

- Do we see the same performance gains when running a query against the large dataset partitioned by views?

In [22]:

```
select_star_pageviews_where_partitioned = """
SELECT *
FROM `redapt-aa-sandbox.demo_dataset.page_views_2020_partitioned`
WHERE views = 100
"""

query_bytes(select_star_pageviews_where_partitioned, select_star_pageviews_where
```

```
This query will process 1.313 gigabytes
$0.192 per month
```

```
Compared to 2308.184 gigabytes
and $338.113 per month
```

and what happens when we select specific columns on a partitioned table?

SELECT cols with WHERE clause on partitioned table

In [23]:
```
select_cols_posts_where_partitioned = """
SELECT
CONCAT('https://stackoverflow.com/questions/',
       CAST(id as STRING)) as url,
view_count
FROM `redapt-aa-sandbox.demo_dataset.posts_questions_partitioned`
WHERE comment_count = 25
LIMIT 100
"""

query_bytes(select_cols_posts_where_partitioned, select_star_posts_where)
```

```
This query will process 0.0 gigabytes
$0.0 per month
Compared to 33.141 gigabytes
and $4.855 per month
```

In [24]:
```
select_cols_pageviews_where_partitioned = """
SELECT wiki, sum(views) as tot_views
FROM `redapt-aa-sandbox.demo_dataset.page_views_2020_partitioned`
WHERE views = 100
AND wiki LIKE 'advisory%'
GROUP BY wiki
ORDER BY tot_views
"""

query_bytes(select_cols_pageviews_where_partitioned, select_star_pageviews_where
```

```
This query will process 0.431 gigabytes
$0.063 per month
Compared to 2308.184 gigabytes
and $338.113 per month
```

Queries without limits

A common misconception is that adding a limit to a BigQuery request will reduce the amount of data scanned. Let's look at some of our sample queries without a limit.

- First, let's try a SELECT * query without a WHERE clause, our baseline query:

In [25]:
```
select_star_posts_nolimit = """
SELECT *
FROM `bigquery-public-data.stackoverflow.posts_questions`
"""

query_bytes(select_star_posts_nolimit, select_star_posts)
```

```
This query will process 33.141 gigabytes
```

```
$4.855 per month
Compared to 33.141 gigabytes
and $4.855 per month
```

- How about on a partitioned table with a WHERE clause. We know we have performance gains on a partitioned table:

In [26]:
```
select_star_posts_where_partitioned_nolimit = """
SELECT * FROM `redapt-aa-sandbox.demo_dataset.posts_questions_partitioned`
WHERE comment_count = 25
"""

query_bytes(select_star_posts_where_partitioned_nolimit, select_star_posts_where
```

```
This query will process 0.003 gigabytes
$0.0 per month
Compared to 0.003 gigabytes
and $0.0 per month
```

- or a query with selected cols?

In [27]:
```
select_posts_cols_nolimits = """
SELECT
CONCAT('https://stackoverflow.com/questions/',
       CAST(id as STRING)) as url,
view_count
FROM `bigquery-public-data.stackoverflow.posts_questions`
WHERE tags like '%google-bigquery%'
ORDER BY view_count DESC """

query_bytes(select_posts_cols_nolimits, select_cols_posts)
```

```
This query will process 0.843 gigabytes
$0.123 per month
Compared to 0.843 gigabytes
and $0.123 per month
```

No performance gains. Why is this?

Another way to see performance gains is to add a clustered index to a table. This is different than a partitioned table, as the clustered index exists within a partition as BigQuery assigns the data. Sometimes we can see gains, sometimes we won't.

Reduced columns query on clustered table

In [28]:
```
select_posts_cols_clustered = """
SELECT
CONCAT('https://stackoverflow.com/questions/',
       CAST(id as STRING)) as url,
view_count
FROM `redapt-aa-sandbox.demo_dataset.post_questions_clustered_tags`
WHERE tags like '%google-bigquery%'
ORDER BY view_count DESC """

query_bytes(select_posts_cols_clustered, select_posts_cols_nolimits)
```

```
This query will process 0.353 gigabytes
$0.052 per month
Compared to 0.843 gigabytes
and $0.123 per month
```

But what if we add a cluster to our partitioned table?

Query on partitioned/clustered table compared to unpartitioned/unclustered table

In [30]:

```
select_pageviews_cols_partitioned_clustered = """
SELECT wiki, sum(views) as tot_views
FROM `redapt-aa-sandbox.demo_dataset.pageviews_part_clust_tags`
WHERE views = 100
AND wiki LIKE 'advisory%'
GROUP BY wiki
ORDER BY tot_views
"""

query_bytes(select_pageviews_cols_partitioned_clustered, select_pageviews_cols_c
```

```
This query will process 0.443 gigabytes
$0.065 per month
Compared to 690.683 gigabytes
and $101.174 per month
```

Query on partitioned/clustered table compared to a unpartitioned/clustered table

In [31]:

```
select_pageviews_cols_partitioned_clustered = """
SELECT wiki, sum(views) as tot_views
FROM `redapt-aa-sandbox.demo_dataset.pageviews_part_clust_tags`
WHERE views = 100
AND wiki LIKE 'advisory%'
GROUP BY wiki
ORDER BY tot_views
"""

query_bytes(select_pageviews_cols_partitioned_clustered, select_pageviews_cols_c
```

```
This query will process 0.443 gigabytes
$0.065 per month
Compared to 690.683 gigabytes
and $101.174 per month
```

Query on partitioned/clustered table comparted to partitioned/unclustered table

In [32]:

```
select_pageviews_cols_partitioned_clustered = """
SELECT wiki, sum(views) as tot_views
FROM `redapt-aa-sandbox.demo_dataset.pageviews_part_clust_tags`
WHERE views = 100
AND wiki LIKE 'advisory%'
GROUP BY wiki
ORDER BY tot_views
```

```
"""

query_bytes(select_pageviews_cols_partitioned_clustered, select_cols_pageviews_w
```

```
This query will process 0.443 gigabytes
$0.065 per month
Compared to 0.431 gigabytes
and $0.063 per month
```