Cost Optimization

**Understanding the basics of pricing in BigQuery**

For any location, the [BigQuery pricing](https://cloud.google.com/products/calculator/) is broken down to:

* Storage
  + Active storage
  + Long-term storage
  + Streaming inserts
* Query processing
  + On-demand
  + Flat-rate

### Cost optimization techniques in BigQuery: storage

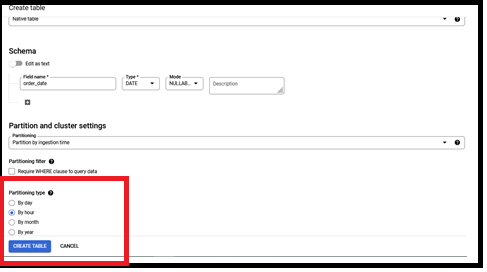
Once data is loaded into BigQuery, charges are based on the amount of data stored in your tables per second. Here are a few tips to optimize your BigQuery storage costs.

1. **Optimizing a query**

The most important point is to limit the bytes. So instead of selecting the whole table we need to select a particular column which we need. For example

|  |  |
| --- | --- |
| **Original code** | **Optimized** |
| Select \* from dataset.table | Select dim1,dim2 from dataset.table |

1. Auto-pruning with Partitioning and clustering

****

BigQuery automatically prunes our data which we don’t want to use in query.

|  |  |
| --- | --- |
| **Original code** | **Optimized** |
| Select ….from dataset.table | Select … from dataset.table where DATE(datehour) = ‘2020-06-09’ and name=”android”; |

1. **Aggregation**

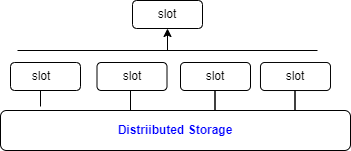
* An aggregation function summarizes the rows of a group into single value.
* First the data is read from storage system and GROUP BY is done on individual slots and then results from each slots is bucketed using hash function( data having same key ends on same slot). So all partial aggregations for the “title” ends up on the same slot. If there is need of more aggregations, it occurs and the final result is passed to the last slot which limits the result. Since this requires a few different aggregation steps GROUB BY query can be costly.

select title, sum(visitors) from dataset.table

where DATE(datehour) = “2020-09-08”

group by title

order by sum(views) desc limit 10;



* We can avoid GROUP BY is to nest repeated data. For example we have a transaction table that have information about the order.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Order id | Order date | Customer id | Product name | Product price |
| 1 | 6/11/21 | 1 | Dennis shorts | 21 |
| 1 | 6/11/21 | 1 | Blue shirt | 8 |

* If we use flat denormalization, we would have 1 row for each ordered item. Instead of this we could use the below format and use ARRAY\_LENGTH method to query the data.

|  |  |  |  |
| --- | --- | --- | --- |
| Order id | Order date | Customer id | Products |
| 1 | 6/11/21 | 1 | [  {“name”:Dennis shorts, “price”:21},  {“name”: Blue shirt, “price”:8}  ] |

1. **Joints**

If we join so large tables together, BigQuery uses a hash JOIN or shuffle JOIN similar to the aggregation function(hash function: BigQuery shuffles both the left and right tables so that the matching keys can be used to join data). So **how to Optimize JOIN pattern?**

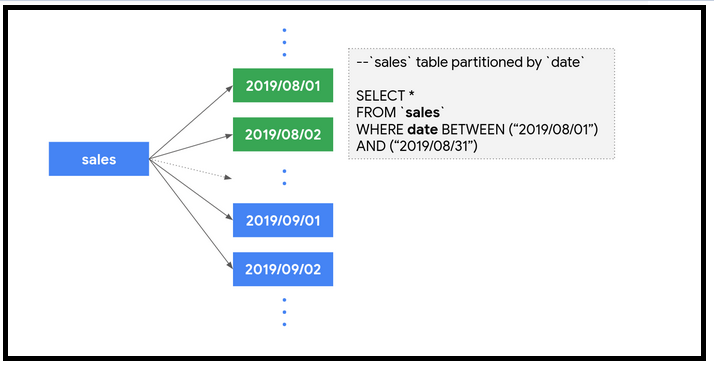
|  |  |
| --- | --- |
| Original code | Optimized code |
| Select t1.dim1, t2.dim2,sum(cost)  From  ‘dataset.small\_table’ t1  JOIN  ‘dataset.large\_table1’ t2  ON  t1.id = t2.id. | Select t1.dim1, t2.dim2,sum(cost)  From  ‘dataset.large\_table’ t2  JOIN  ‘dataset.small\_table’ t1  ON  t1.id = t2.id  *where t2.dim2=’abc’ and AND t2.dim2=’abc’*  GROUP BY 1; |

* First use large table and then the small table
* Can use filters on both the tables

1. Using Cluster and partitions

**Partition**

Suppose we have a dataset ‘Sales’ which is based on ingestion time, date, or any timestamp column. Let’s say you partition a sales table that contains data for the last 12 months.



Now, when you query to analyze sales data for the month of August, you only pay for data processed in those 31 partitions, not the entire table.

* Can use cluster and partition on tables based on common based on common features because less data will be scanned and the query will execute faster.
* Clustering improves the efficiency more than partitioning.

1. Using WHERE clause.

While using WHERE clause, BigQuery thinks that you have chosen the best order to filter the data. The use of WHERE clause is to eliminate the most data.

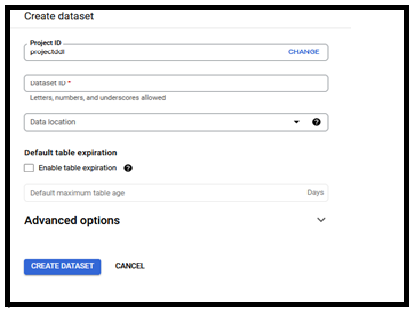
1. Using ORDER BY with LIMIT

|  |
| --- |
| Original code |
| select  t1.dim1,  t1.dim2  from  ‘dataset.table’ t1  ORDER BY t1.customer\_name desc; |

While using ORDER BY we may get into resources which exceeded error. So to avoid this we can use LIMIT so that the result set is easier to manage.

1. Keep your data only as long as you need it.

* By default, BigQuery’s data stores in [Capacitor columnar data format](https://cloud.google.com/blog/products/gcp/inside-capacitor-bigquerys-next-generation-columnar-storage-format) which is in encrypted and compressed format. We can set the default table expiration time as



* BigQuery also offers the flexibility to provide different table expiration dates within the same dataset.

1. Avoid duplicate copies of data.

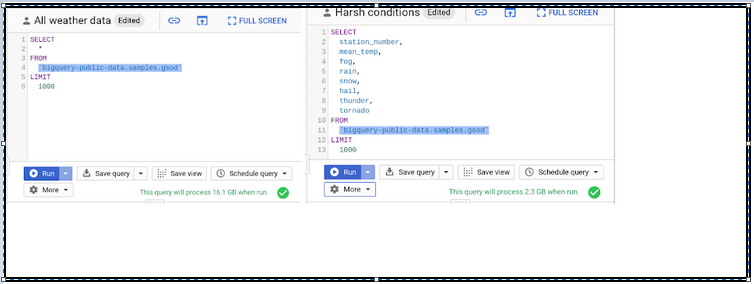
BigQuery uses a [**federated data access model**](https://cloud.google.com/bigquery/external-data-sources) (*A federated query is a way to send a query statement to an external database and get the result back as a temporary table. Federated queries use the BigQuery Connection API to establish a connection with the external database.* ) that allows to query data directly from external data sources like Cloud Bigtable, Cloud Storage, Google Drive and [Cloud SQL](https://cloud.google.com/bigquery/docs/cloud-sql-federated-queries). This is useful *for avoiding duplicate copies* of data, thus reducing storage costs.

**Note**

Typically, [queries that run on external sources](https://cloud.google.com/bigquery/external-data-sources#external_data_source_limitations) don’t perform as well compared to queries executed on same data stored on BigQuery, since data stored on BigQuery is in a columnar format that yields much better performance.

1. Understanding BigQuery’s backup and disaster recovery scenarios for data backup.
2. Only query the data you need.

BigQuery performance is incredible because it stores data as a columnar format. SELECT \* is the most expensive way to query data. This is because it will perform a full query scan across every column present in the table(s), including the ones you might not need.



From above we can check that by selecting the necessary columns, we can reduce the bytes processed by about eight-fold, which is a quick way to optimize for cost.

**Note**

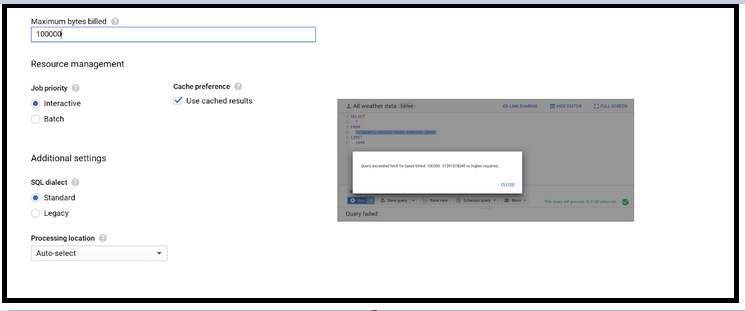
It is necessary to explore the data and understand its semantics, you can always use the no-charge [**data preview option**](https://cloud.google.com/bigquery/docs/best-practices-costs#preview-data).

 In the Cloud Console, on the table details page, click the **Preview** tab to sample the data.

* In the bq command-line tool, use the [bq head](https://cloud.google.com/bigquery/docs/managing-table-data#browse-table) command and specify the number of rows to preview.

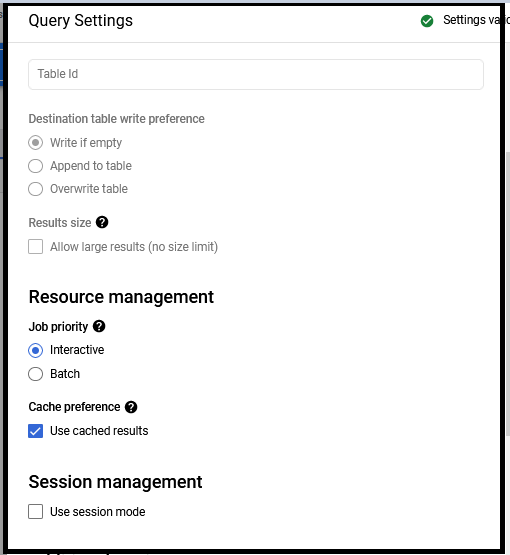
1. Set up controls for accidental human errors.

When we have dataset tables that are in the magnitude of TBs or PBs and are accessed by multiple individuals, unknowingly querying all columns could result in a substantial query cost. In this case, use the *maximum bytes* billed setting to limit query cost.



1. Use caching

Caching can actually boost your query performance, and you won’t be charged for the results retrieved from the cached tables. By default, cache preference is turned on. Check them in your GCP console by clicking More -> Query settings on your query editor, as shown here.



In Real world example, caching helps to store queries across multiple users.