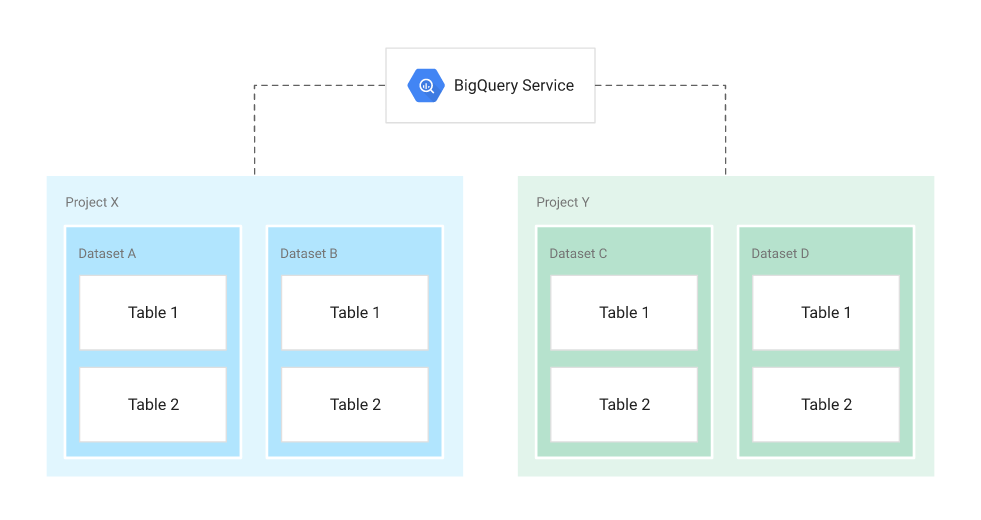
BigQuery Storage Overview

BigQuery Resource Model

BigQuery organizes data tables into units called datasets. These datasets are scoped to your GCP project. These multiple scopes — project, dataset, and table — helps you structure your information logically. You can use multiple datasets to separate tables pertaining to different analytical domains, and you can use project-level scoping to isolate datasets from each other according to your business needs.



**Fig: Structural overview of BigQuery Resources [**[**source**](https://cloud.google.com/solutions/bigquery-data-warehouse)**]**

Projects

* Root namespace for objects
* Contain multiple datasets, jobs, access control lists and IAM roles
* Control billing, users, and user privileges

Datasets

* Collections of “related” tables/views together with labels and description
* Allow storage access control at Dataset level
* Define location of data i.e. multi-regional (US, EU) or regional (asia-northeast1)

Tables

* Collections of columns and rows stored in managed storage
* Defined by a schema with strongly-typed columns of values
* Allow access control at [Table level](https://cloud.google.com/bigquery/docs/table-access-controls-intro) and [Column level](https://cloud.google.com/bigquery/docs/column-level-security-intro)

Views

* Virtual tables defined by a SQL query
* Allow access control at View level

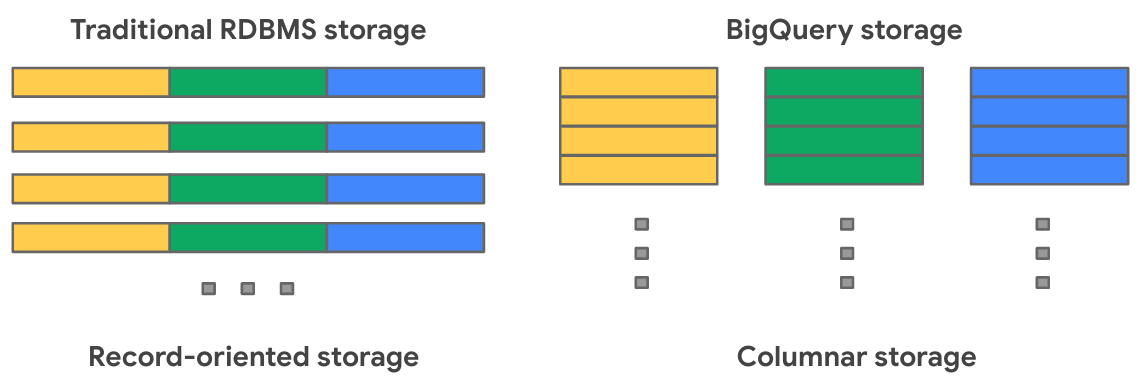
Jobs

* Actions run by BigQuery on your behalf — load data, export data, copy data, or query data
* Jobs are executed asynchronously

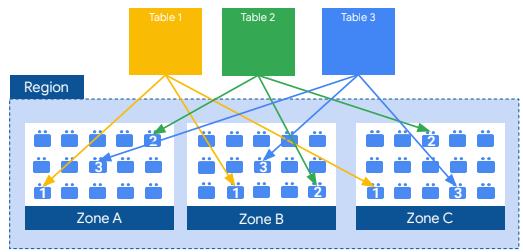
When you reference a table from the command line, in SQL queries, or in code, you refer to it by using the following construct: project.dataset.table.

Storage Management

Now let’s review how BigQuery manages the storage that holds your data. Traditional relational databases, like MySQL, store data row-by-row ([record-oriented storage](http://publications.csail.mit.edu/abstracts/abstracts06/dna/dna.html)). This makes them good at transactional updates and [OLTP](https://en.wikipedia.org/wiki/Online_transaction_processing) (Online Transaction Processing) use cases. BigQuery, on the other hand, uses columnar storage, where each column is stored in a separate file block. This makes BigQuery an ideal solution for [OLAP](https://en.wikipedia.org/wiki/Online_analytical_processing) (Online Analytical Processing) use cases. You can stream (append) data easily to BigQuery tables and update or delete existing values. BigQuery [supports mutations](https://cloud.google.com/blog/products/data-analytics/dml-without-limits-now-in-bigquery) (INSERT, UPDATE, MERGE, DELETE) without limits.



BigQuery uses variations and advancements on columnar storage. Internally, BigQuery stores data in a proprietary columnar format called [Capacitor](https://cloud.google.com/blog/products/gcp/inside-capacitor-bigquerys-next-generation-columnar-storage-format), which has a number of benefits for data warehouse workloads. BigQuery uses a proprietary format because the storage engine can evolve in tandem with the query engine, which takes advantage of deep knowledge of the data layout to optimize query execution. Each column in the table is stored in a separate file block and all the columns are stored in a single capacitor file, , which are compressed and encrypted on disk. BigQuery uses query access patterns to determine the optimal number of physical shards and how data is encoded.

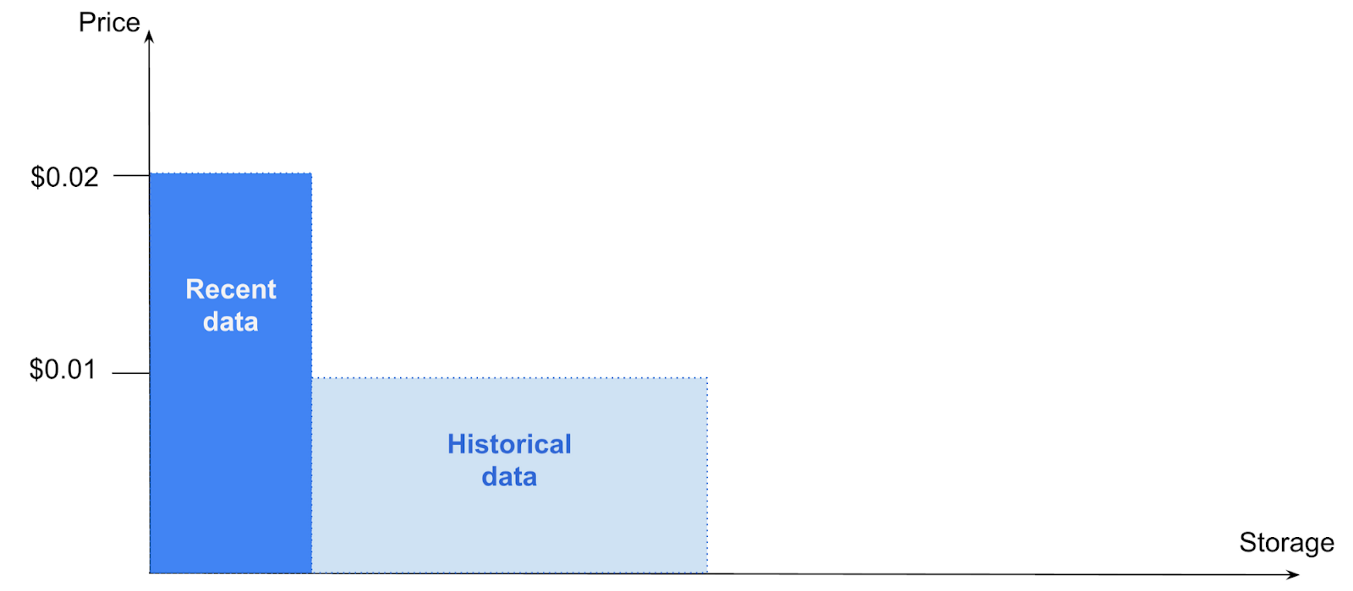


The actual persistence layer is provided by Google’s distributed file system, [Colossus](https://cloud.google.com/files/storage_architecture_and_challenges.pdf), where data is automatically compressed, encrypted, replicated, and distributed. Colossus ensures durability using erasure encoding to store redundant chunks of data on multiple physical disks. This is all accomplished without impacting the computing power available for your queries. Separating storage from compute allows you to scale to petabytes in storage seamlessly, without requiring additional expensive compute resources. There are a number of [other benefits](https://cloud.google.com/blog/products/gcp/separation-of-compute-and-state-in-google-bigquery-and-cloud-dataflow-and-why-it-matters) of decoupling compute and storage.

Take Advantage of Long-Term Storage

You can load data into BigQuery at [no cost](https://cloud.google.com/bigquery/pricing#free) (for batch loads) because [BigQuery storage costs](https://cloud.google.com/bigquery/pricing" \l "storage" \t "_blank) are based on the amount of data stored (first 10 GB is free each month) and whether storage is considered active or long-term.

* If you have a table or partition modified in the last 90 days, it is considered as active storage and incurs a monthly charge for data stored at BigQuery storage rates.
* If you have a table or partition that is not modified for 90 consecutive days, it is considered long term storage and the price of storage for that table [automatically drops by 50%](https://cloud.google.com/bigquery/pricing#long-term-storage) to the same cost as [Cloud Storage Nearline](https://cloud.google.com/storage/pricing#storage-pricing). Discount is applied on a per-table, per-partition basis. If you modify the data in the table, the 90-day counter resets.



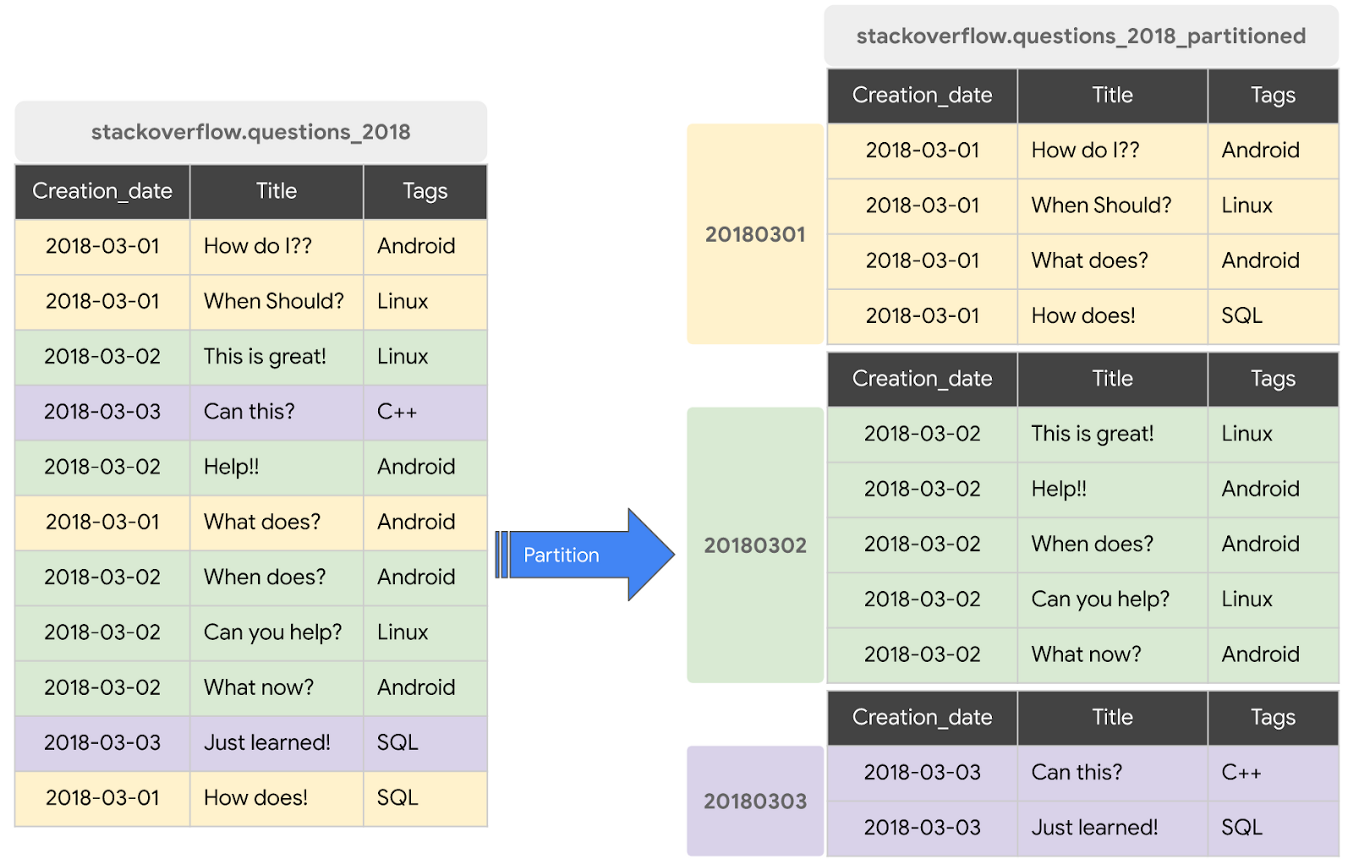
A best practice when optimizing costs is to keep your data in BigQuery. Rather than exporting your older data to another storage option (such as Cloud Storage), take advantage of BigQuery’s long-term storage pricing. This means not having to delete old data or architect a data archival process. Since the data remains in BigQuery, you can also query older data using the same interface, at the same cost levels, with the same performance characteristics.

Partitioning and Clustering

Keeping data in BigQuery is a best practice if you’re looking to optimize both cost and performance. Another best practice is using BigQuery’s table partitioning and clustering features to structure your data to match common data access patterns.

Partitioning

A partitioned table is a special table that is divided into segments, called partitions, that make it easier to manage and query your data. You can typically split large tables into many smaller partitions using data ingestion time or TIMESTAMP/DATE column or an INTEGER column. BigQuery’s decoupled storage and compute architecture leverages column-based partitioning simply to minimize the amount of data that slot workers read from disk. Once slot workers read their data from disk, BigQuery can automatically determine more optimal data sharding and quickly repartition data using BigQuery’s in-memory shuffle service.



Data written to a column-based time partitioned table is automatically delivered to the appropriate partition based on the value of the data. Similarly, queries that express filters on the partitioning column can reduce the overall data scanned, which can yield improved performance and reduced query cost for on-demand queries.

BigQuery supports following ways to create partitioned tables

*Ingestion time partitioned tables*

* Partitioned on the data’s ingestion time or arrival time.
* BigQuery automatically loads data into daily, date based partitions reflecting the data’s ingestion or arrival time.
* BigQuery adds two pseudo columns to ingestion-time partitioned tables — a \_PARTITIONTIME pseudo column containing a date-based timestamp for data and a \_PARTITIONDATE pseudo column contains a date representation.

*DATE/TIMESTAMP column partitioned tables*

* Partitioned based on a TIMESTAMP or DATE column.
* BigQuery routes data to the appropriate partition based on the date value (expressed in UTC) in the partitioning column.
* BigQuery creates two special partitions: the \_\_NULL\_\_ partition for capturing rows NULL values in the partitioning column and the \_\_UNPARTITIONED\_\_ partition for data outside the allowed range of dates.
* You can create partitions with granularity starting from hourly partitioning.

*INTEGER range partitioned tables*

* Partitioned based on an integer column with start, end, and interval values.
* BigQuery creates two special partitions: the \_\_NULL\_\_ partition for capturing rows with NULL values in the partitioning column and the \_\_UNPARTITIONED\_\_ partition for data outside the allowed range of integers.

Let’s look at partitioning in action. To see the difference in performance between a non-partitioned and a partitioned table, we will create non-partitioned and partitioned tables with the same dataset and check the query performance.

**Before Partitioning**

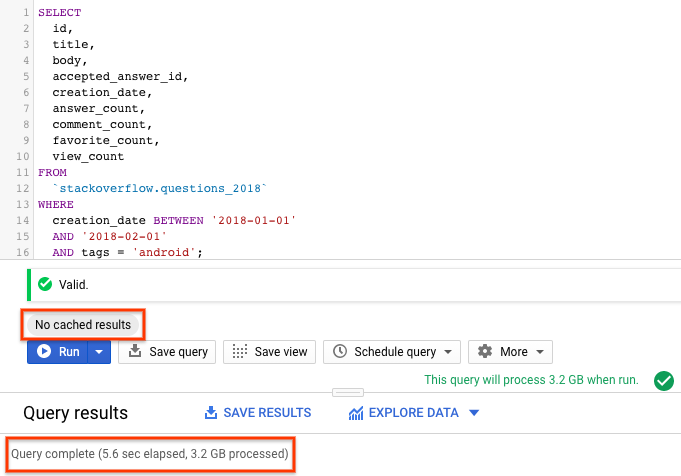
By running the following SQL query, we will create a non-partitioned table with data loaded from a [public data set](https://console.cloud.google.com/marketplace/browse?filter=solution-type:dataset&_ga=2.99891288.755387171.1599359331-1764086539.1598401521) based on StackOverflow posts by [creating a new table from an existing table](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#creating_a_new_table_from_an_existing_table). This table will contain the StackOverflow posts created in 2018.

CREATE OR REPLACE TABLE `stackoverflow.questions\_2018` AS  
SELECT \*  
FROM `bigquery-public-data.stackoverflow.posts\_questions`  
WHERE creation\_date BETWEEN '2018-01-01' AND '2018-07-01'

Let’s **query the non-partitioned table** to fetch all StackOverflow questions tagged with ‘android’ in the month of January 2018.

SELECT  
 id,  
 title,  
 body,  
 accepted\_answer\_id,  
 creation\_date,  
 answer\_count,  
 comment\_count,  
 favorite\_count,  
 view\_count  
FROM  
 `stackoverflow.questions\_2018`  
WHERE  
 creation\_date BETWEEN '2018-01-01' AND '2018-02-01'  
 AND tags = 'android';

Before the query runs, [caching is disabled](https://cloud.google.com/bigquery/docs/cached-results#console) to be fair when comparing performance with partitioned and clustered tables.



From the query results, you can see that the query on a ***non-partitioned table took 5.6s to scan the entire 3.2GB of data*** with StackOverflow posts created in 2018.

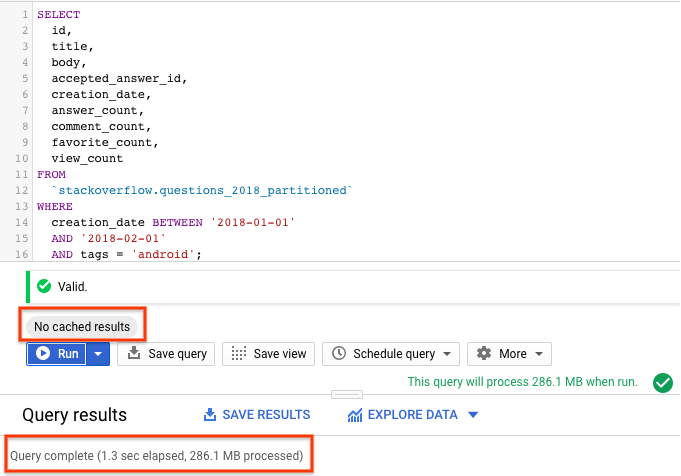
**Partition the Table**

Now let’s see whether partitioned tables can do better. You can create partitioned tables in [multiple ways](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#creating_ingestion-time_partitioned_tables). We will create a DATE/TIMESTAMP partitioned table using [BigQuery DDL statements](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language" \t "_blank). We chose the partitioning column as creation\_date based on the query access pattern.

CREATE OR REPLACE TABLE `stackoverflow.questions\_2018\_partitioned`  
PARTITION BY  
 DATE(creation\_date) AS  
SELECT  
 \*  
FROM  
 `bigquery-public-data.stackoverflow.posts\_questions`  
WHERE  
 creation\_date BETWEEN '2018-01-01' AND '2018-07-01';

Now let’s run the previous **query on the partitioned table,**[with cache disabled](https://cloud.google.com/bigquery/docs/cached-results#console), to fetch all StackOverflow questions tagged with ‘android’ in the month of January 2018.

SELECT  
 id,  
 title,  
 body,  
 accepted\_answer\_id,  
 creation\_date,  
 answer\_count,  
 comment\_count,  
 favorite\_count,  
 view\_count  
FROM  
 `stackoverflow.questions\_2018\_partitioned`  
WHERE  
 creation\_date BETWEEN '2018-01-01' AND '2018-02-01'  
 AND tags = 'android';



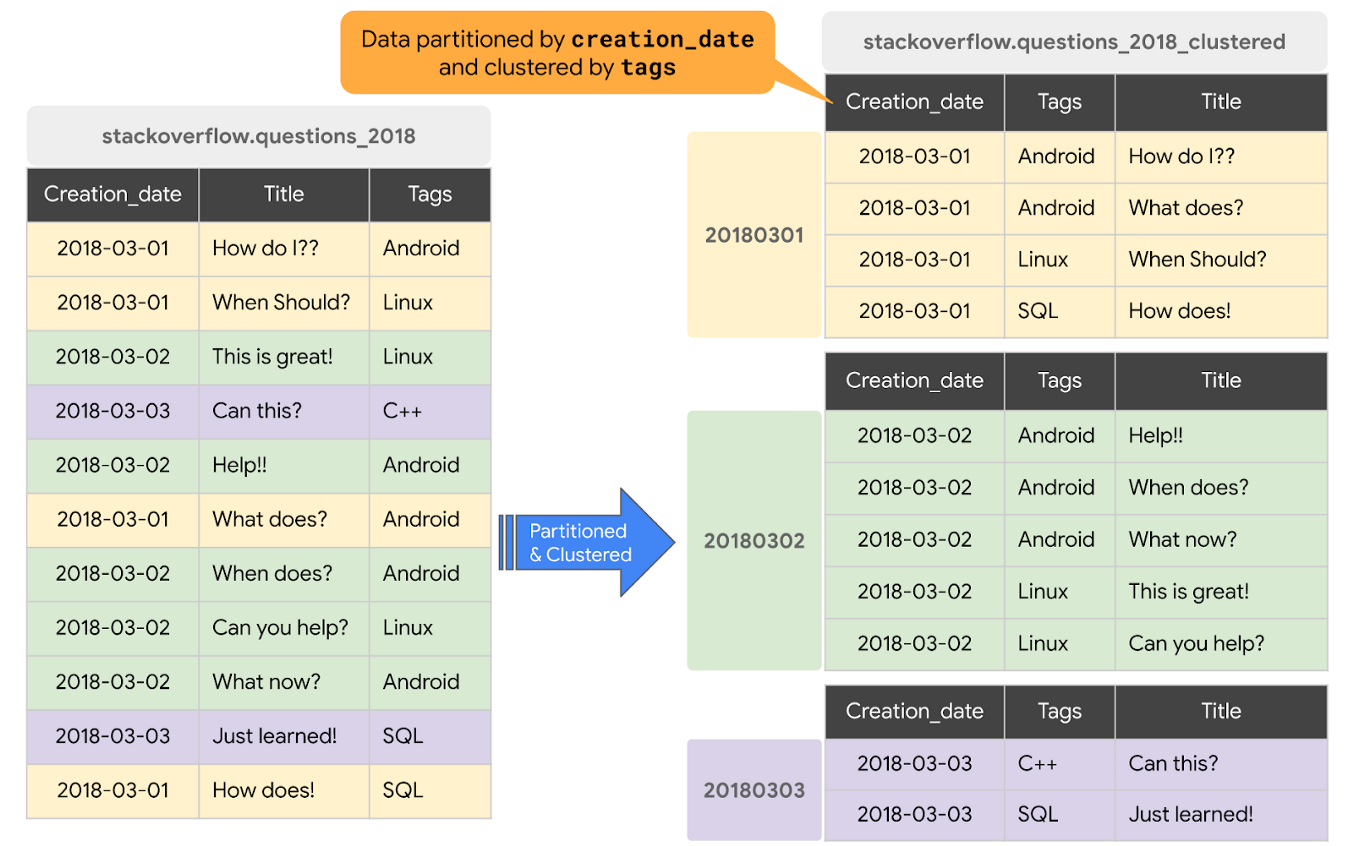
***With partitioned table query scanned only the required partitions in <2s processing ~290MB data*** compared to query running with non-partitioned table processing 3.2GB.

Partition management is key to fully maximizing BigQuery performance and cost when querying over a specific range — it results in scanning less data per query, and pruning is determined before query start time. While partitioning reduces cost and improves performance, it also prevents cost explosion due to user accidentally querying really large tables in entirety.

Clustering

When a table is clustered in BigQuery, the table data is automatically organized based on the contents of one or more columns in the table’s schema. The columns you specify are used to collocate related data. Usually high cardinality and non-temporal columns are preferred for clustering.

When data is written to a clustered table, 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. The order of clustered columns determines the sort order of the data. When new data is added to a table or a specific partition, BigQuery performs [automatic re-clustering](https://cloud.google.com/blog/products/data-analytics/skip-the-maintenance-speed-up-queries-with-bigquerys-clustering) in the background to restore the sort property of the table or partition. *Auto re-clustering is completely free and autonomous for the users.*



Clustering can improve the performance of certain types of queries, such as those using filter clauses and queries aggregating data.

* When a query containing a filter clause filters data based on the clustering columns, BigQuery uses the sorted blocks to eliminate scans of unnecessary data.
* When a query aggregates data based on the values in the clustering columns, performance is improved because the sorted blocks collocate rows with similar values.

BigQuery supports clustering over both partitioned and non-partitioned tables. When you use clustering and partitioning together, your data can be partitioned by a DATE or TIMESTAMP column and then clustered on a different set of columns (up to four columns).

Coming back to the previous query, let’s find out how the query with clustered table performs.

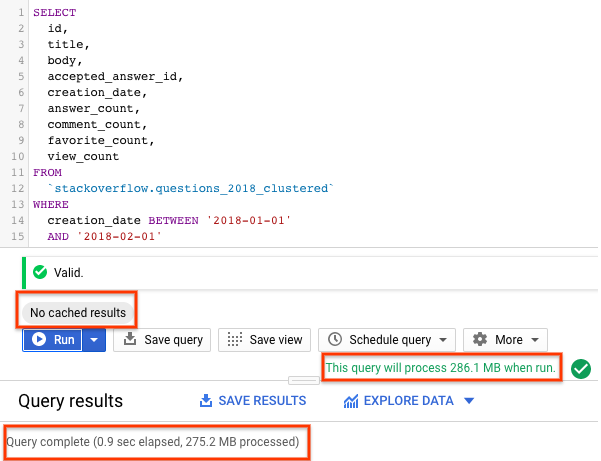
**Cluster the Table**

You can create clustered tables in [multiple ways](https://cloud.google.com/bigquery/docs/creating-clustered-tables#creating_clustered_tables). We will create a new DATE/TIMESTAMP partitioned and clustered table using [BigQuery DDL statements](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language" \l "creating_a_clustered_table" \t "_blank). We chose the partitioning column as creation\_date and cluster key as tag based on the query access pattern.

CREATE OR REPLACE TABLE `stackoverflow.questions\_2018\_clustered`  
PARTITION BY  
 DATE(creation\_date)  
CLUSTER BY  
 tags AS  
SELECT  
 \*  
FROM  
 `bigquery-public-data.stackoverflow.posts\_questions`  
WHERE  
 creation\_date BETWEEN '2018-01-01' AND '2018-07-01';

Now let’s run the **query on the partitioned and clustered table,**[with cache disabled](https://cloud.google.com/bigquery/docs/cached-results#console), to fetch all StackOverflow questions tagged with ‘android’ in the month of January 2018.

SELECT  
 id,  
 title,  
 body,  
 accepted\_answer\_id,  
 creation\_date,  
 answer\_count,  
 comment\_count,  
 favorite\_count,  
 view\_count  
FROM  
 `stackoverflow.questions\_2018\_clustered`  
WHERE  
 creation\_date BETWEEN '2018-01-01'  
 AND '2018-02-01'  
 AND tags = 'android';



***With a partitioned and clustered table, the query scanned ~275MB of data in less than 1s,*** which is better than a partitioned table. The way data is organized by partitioning and clustering minimizes the amount of data scanned by slot workers thereby improving query performance and optimizing costs.

Few things to note when using clustering:

* Clustering does not provide strict cost guarantees before running the query. Notice in the results above with clustering, query validation reported processing of 286.1MB but actually query processed only 275.2MB of data.
* Use clustering only when you need more granularity than partitioning alone allows.