**Introduction to partitioned tables**

This page provides an overview of partitioned tables in BigQuery.

A partitioned table is a special table that is divided into segments, called partitions, that make it easier to manage and query your data. By dividing a large table into smaller partitions, you can improve query performance, and you can control costs by reducing the number of bytes read by a query.

You can partition BigQuery tables by:

* **Time-unit column**: Tables are partitioned based on a [TIMESTAMP](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-types#timestamp_type), [DATE](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-types#date_type), or [DATETIME](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-types#datetime_type) column in the table.
* **Ingestion time**: Tables are partitioned based on the timestamp when BigQuery ingests the data.
* **Integer range**: Tables are partitioned based on an integer column.

If a query filters on the value of the partitioning column, BigQuery can scan the partitions that match the filter and skip the remaining partitions. This process is called *pruning*.

**Time-unit column partitioning**

You can partition a table on a DATE,TIMESTAMP, or DATETIME column in the table. When you write data to the table, BigQuery automatically puts the data into the correct partition, based on the values in the column.

For TIMESTAMP and DATETIME columns, the partitions can have either hourly, daily, monthly, or yearly granularity. For DATE columns, the partitions can have daily, monthly, or yearly granularity. Partitions boundaries are based on UTC time.

For example, suppose that you partition a table on a DATETIME column with monthly partitioning. If you insert the following values into the table, the rows will be written to the following partitions:

| Column value | Partition (monthly) |
| --- | --- |
| DATETIME("2019-01-01") | 201901 |
| DATETIME("2019-01-15") | 201901 |
| DATETIME("2019-04-30") | 201904 |

In addition, two special partitions are created:

* \_\_NULL\_\_: Contains rows with NULL values in the partitioning column.
* \_\_UNPARTITIONED\_\_: Contains rows where the value of the partitioning column is earlier than 1960-01-01 or later than 2159-12-31.

**Ingestion time partitioning**

When you create a table partitioned by ingestion time, BigQuery automatically assigns rows to partitions based on the time when BigQuery ingests the data. You can choose hourly, daily, monthly, or yearly granularity for the partitions. Partitions boundaries are based on UTC time.

An ingestion-time partitioned table has a pseudocolumn named \_PARTITIONTIME. The value of this column is the ingestion time for each row, truncated to the partition boundary (such as hourly or daily). For example, suppose that you create an ingestion-time partitioned table with hourly partitioning and send data at the following times:

| Ingestion time | \_PARTITIONTIME | Partition (hourly) |
| --- | --- | --- |
| 2021-05-07 17:22:00 | 2021-05-07 17:00:00 | 2021050717 |
| 2021-05-07 17:40:00 | 2021-05-07 17:00:00 | 2021050717 |
| 2021-05-07 18:31:00 | 2021-05-07 18:00:00 | 2021050718 |

Because the table in this example uses hourly partitioning, the value of \_PARTITIONTIME is truncated to an hour boundary. BigQuery uses this value to determine the correct partition for the data.

You can also write data to a specific partition. For example, you might want to load historical data or adjust for time zones. You can use any valid date between 0001-01-01 and 9999-12-31. However, [DML statements](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-manipulation-language) cannot reference dates prior to 1970-01-01 or after 2159-12-31. For more information, see [Write data to a specific partition](https://cloud.google.com/bigquery/docs/managing-partitioned-table-data#write-to-partition).

Instead of using \_PARTITIONTIME, you can also use [\_PARTITIONDATE](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#the_partitiondate_pseudo_column). The \_PARTITIONDATE pseudocolumn contains the UTC date corresponding to the value in the \_PARTITIONTIME pseudocolumn.

**Integer range partitioning**

You can partition a table based on ranges of values in a specific INTEGER column. To create an integer-range partitioned table, you provide:

* The partitioning column.
* The starting value for range partitioning (inclusive).
* The ending value for range partitioning (exclusive).
* The interval of each range within the partition.

For example, suppose you create an integer range partition with the following specification:

| Argument | Value |
| --- | --- |
| column name | customer\_id |
| start | 0 |
| end | 100 |
| interval | 10 |

The table is partitioned on the customer\_id column into ranges of interval 10. The values 0 to 9 go into one partition, values 10 to 19 go into the next partition, and so on, up to 99. Values outside this range go into a partition named \_\_UNPARTITIONED\_\_ Any rows where customer\_id is NULL go into a partition named \_\_NULL\_\_.

**Choosing daily, hourly, monthly, or yearly partitioning.**

When you partition a table by time-unit column or ingestion time, you choose whether the partitions have daily, hourly, monthly, or yearly granularity.

* **Daily partitioning** is the default partitioning type. Daily partitioning is a good choice when your data is spread out over a wide range of dates, or if data is continuously added over time.
* Choose **hourly partitioning** if your tables have a high volume of data that spans a short date range — typically less than six months of timestamp values. If you choose hourly partitioning, make sure the partition count will stay within the [partition limits](https://cloud.google.com/bigquery/quotas#partitioned_tables).
* Choose **monthly or yearly partitioning** if your tables have a relatively small amount of data for each day, but span a wide date range. This option is also recommended if your workflow requires frequently updating or adding rows that span a wide date range (for example, more than 500 dates). In these scenarios, use monthly or yearly partitioning along with clustering on the partitioning column to achieve the best performance. For more information, see [Partitioning versus clustering](https://cloud.google.com/bigquery/docs/partitioned-tables#partitioning_versus_clustering) on this page.

**Partitioning versus clustering**

Both partitioning and clustering can improve performance and reduce query cost.

Use clustering under the following circumstances:

* You don't need strict cost guarantees before running the query.
* You need more granularity than partitioning alone allows. To get clustering benefits in addition to partitioning benefits, you can use the same column for both partitioning and clustering.
* Your queries commonly use filters or aggregation against multiple particular columns.
* The cardinality of the number of values in a column or group of columns is large.

Use partitioning under the following circumstances:

* You want to know query costs before a query runs. Partition pruning is done before the query runs, so you can get the query cost after partitioning pruning through a [dry run](https://cloud.google.com/bigquery/docs/dry-run-queries). Cluster pruning is done when the query runs, so the cost is known only after the query finishes.
* You need partition-level management. For example, you want to set a partition expiration time, load data to a specific partition, or delete partitions.
* You want to specify how the data will be partitioned and what data is in each partition. For example, you want to define time granularity or define the ranges used to partition the table for integer range partitioning.

Prefer clustering over partitioning under the following circumstances:

* Partitioning results in a small amount of data per partition (approximately less than 1 GB).
* Partitioning results in a large number of partitions beyond the [limits on partitioned tables](https://cloud.google.com/bigquery/quotas#partitioned_tables).
* Partitioning results in your mutation operations modifying the majority of partitions in the table frequently (for example, every few minutes).

You can also combine partitioning with clustering. Data is first partitioned and then data in each partition is clustered by the clustering columns.

When the table is queried, partitioning sets an upper bound of the query cost based on partition pruning. There might be other query cost savings when the query actually runs, because of cluster pruning.

**Partitioning versus sharding**

Table sharding is the practice of storing data in multiple tables, using a naming prefix such as [PREFIX]\_YYYYMMDD.

Partitioning is recommended over table sharding, because partitioned tables perform better. With sharded tables, BigQuery must maintain a copy of the schema and metadata for each table. BigQuery might also need to verify permissions for each queried table. This practice also adds to query overhead and affects query performance.

If you previously created date-sharded tables, you can convert them into an ingestion-time partitioned table. For more information, see [Convert date-sharded tables into ingestion-time partitioned tables](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#convert-date-sharded-tables).

**Limitations**

You cannot use legacy SQL to query partitioned tables or to write query results to partitioned tables.

Time-unit column-partitioned tables are subject to the following limitations:

* The partitioning column must be either a scalar DATE, TIMESTAMP, or DATETIME column. While the mode of the column can be REQUIRED or NULLABLE, it cannot be REPEATED (array-based).
* The partitioning column must be a top-level field. You cannot use a leaf field from a RECORD (STRUCT) as the partitioning column.

Integer-range partitioned tables are subject to the following limitations:

* The partitioning column must be an INTEGER column. While the mode of the column may be REQUIRED or NULLABLE, it cannot beREPEATED (array-based).
* The partitioning column must be a top-level field. You cannot use a leaf field from a RECORD (STRUCT) as the partitioning column.

# Creating partitioned tables

This page describes how to create partitioned tables in BigQuery. For an overview of partitioned tables, see [Introduction to partitioned tables](https://cloud.google.com/bigquery/docs/partitioned-tables).

## Before you begin

Grant Identity and Access Management (IAM) roles that give users the necessary permissions to perform each task in this document.

### Required permissions

To create a table, you need the following IAM permissions:

* bigquery.tables.create
* bigquery.tables.updateData
* bigquery.jobs.create

Additionally, you might require the bigquery.tables.getData permission to access the data that you write to the table.

Each of the following predefined IAM roles includes the permissions that you need in order to create a table:

* roles/bigquery.dataEditor
* roles/bigquery.dataOwner
* roles/bigquery.admin (includes the bigquery.jobs.create permission)
* roles/bigquery.user (includes the bigquery.jobs.create permission)
* roles/bigquery.jobUser (includes the bigquery.jobs.create permission)

Additionally, if you have the bigquery.datasets.create permission, you can create and update tables in the datasets that you create.

For more information on IAM roles and permissions in BigQuery, see [Predefined roles and permissions](https://cloud.google.com/bigquery/docs/access-control).

## Create an empty partitioned table

The steps to create a partitioned table in BigQuery are similar to creating a [standard table](https://cloud.google.com/bigquery/docs/tables), except that you specify the partitioning options, along with any other table options.

### Create a time-unit column-partitioned table

To create an empty time-unit column-partitioned table with a schema definition:

[Console](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#console)[SQL](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#sql)[bq](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#bq)[API](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#api)[Go](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#go)[Java](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#java)[Node.js](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#node.js)[Python](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#python)

To create a time-unit column-partitioned table, use the [CREATE TABLE statement](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#create_table_statement) with a [PARTITION BY clause](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#partition_expression).

The following example creates a table with daily partitions based on the transaction\_date column.

CREATE TABLE  
  mydataset.newtable (transaction\_id INT64, transaction\_date DATE)  
PARTITION BY  
  transaction\_date  
OPTIONS(  
  partition\_expiration\_days=3,  
  require\_partition\_filter=true  
)

Use the [OPTIONS clause](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#table_option_list) to set table options such as the [partition expiration](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#partition-expiration) and the [partition filter requirements](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#require-filter).

The default partitioning type for DATE columns is daily partitioning. To specify a different partitioning type, include the [DATE\_TRUNC](https://cloud.google.com/bigquery/docs/reference/standard-sql/date_functions#date_trunc) function in the PARTITION BY clause. For example, the following query creates a table with monthly partitions:

CREATE TABLE  
  mydataset.newtable (transaction\_id INT64, transaction\_date DATE)  
PARTITION BY  
  DATE\_TRUNC(transaction\_date, MONTH)  
OPTIONS(  
  partition\_expiration\_days=3,  
  require\_partition\_filter=true  
)

You can also specify a TIMESTAMP or DATETIME column as the partitioning column. In that case, include the TIMESTAMP\_TRUNC or DATETIME\_TRUNC function in the PARTITION BY clause to specify the partition type. For example, the following statement creates a table with daily partitions based on a TIMESTAMP column:

CREATE TABLE  
  mydataset.newtable (transaction\_id INT64, transaction\_ts TIMESTAMP)  
PARTITION BY  
  TIMESTAMP\_TRUNC(transaction\_ts, DAY)  
OPTIONS(  
  partition\_expiration\_days=3,  
  require\_partition\_filter=true  
)

For information about how to run queries, see [Running interactive queries](https://cloud.google.com/bigquery/docs/running-queries#queries).

### Create an ingestion-time partitioned table

To create an empty ingestion-time partitioned table with a schema definition:

[Console](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#console)[SQL](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#sql)[bq](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#bq)[API](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#api)

To create an ingestion-time partitioned table, use the [CREATE TABLE statement](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#create_table_statement) with a [PARTITION BY clause](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#partition_expression) that partitions on \_PARTITIONDATE.

The following example creates a table with daily partitions.

CREATE TABLE  
  mydataset.newtable (transaction\_id INT64)  
PARTITION BY  
  \_PARTITIONDATE  
OPTIONS(  
  partition\_expiration\_days=3,  
  require\_partition\_filter=true  
)

Use the [OPTIONS clause](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#table_option_list) to set table options such as the [partition expiration](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#partition-expiration) and the [partition filter requirements](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#require-filter).

The default partitioning type for ingestion-time partitioning is daily partitioning. To specify a different partitioning type, include the [DATE\_TRUNC](https://cloud.google.com/bigquery/docs/reference/standard-sql/date_functions#date_trunc) function in the PARTITION BY clause. For example, the following query creates a table with monthly partitions:

CREATE TABLE  
  mydataset.newtable (transaction\_id INT64)  
PARTITION BY  
  DATE\_TRUNC(\_PARTITIONTIME, MONTH)  
OPTIONS(  
  partition\_expiration\_days=3,  
  require\_partition\_filter=true  
)

For information about how to run queries, see [Running interactive queries](https://cloud.google.com/bigquery/docs/running-queries#queries).

### Create an integer-range partitioned table

To create an empty integer-range partitioned table with a schema definition:

[Console](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#console)[SQL](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#sql)[bq](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#bq)[API](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#api)[Java](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#java)[Node.js](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#node.js)[Python](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#python)

To create an integer-range partitioned table, use the [CREATE TABLE statement](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#create_table_statement) with a [PARTITION BY clause](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#partition_expression).

The following example creates a table that is partitioned on the customer\_id column with start 0, end 100, and interval 10.

CREATE TABLE mydataset.newtable (customer\_id INT64, date1 DATE)  
PARTITION BY  
  RANGE\_BUCKET(customer\_id, GENERATE\_ARRAY(0, 100, 10))  
OPTIONS(  
  require\_partition\_filter=true  
)

Use the [OPTIONS clause](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#table_option_list) to set table options such as the [partition filter requirements](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#require-filter).

For information about how to run queries, see [Running interactive queries](https://cloud.google.com/bigquery/docs/running-queries#queries).

## Create a partitioned table from a query result

You can create a partitioned table from a query result in the following ways:

* Use the bq command-line tool or the BigQuery API to set a destination table for a query. When the query runs, BigQuery writes the results to the destination table. You can use this approach for any partitioning type.
* In SQL, use a CREATE TABLE ... AS SELECT statement. You can use this approach to create a table that is partitioned by time-unit column or integer range, but not ingestion time.

[SQL](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#sql)[bq](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#bq)[API](https://cloud.google.com/bigquery/docs/creating-partitioned-tables#api)

Use the [CREATE TABLE](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#create_table_statement) statement with a [SELECT AS](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#query_statement) clause for the query. Include a [PARTITION BY](https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#partition_expression) clause to configure the partitioning.

The following example creates a table that is partitioned on the transaction\_date column.

CREATE TABLE  
  mydataset.newtable (transaction\_id INT64, transaction\_date DATE)  
PARTITION BY  
  transaction\_date  
AS SELECT transaction\_id, transaction\_date FROM mydataset.mytable

## Convert date-sharded tables into ingestion-time partitioned tables

If you previously created date-sharded tables, you can convert the entire set of related tables into a single ingestion-time partitioned table by using the [partition](https://cloud.google.com/bigquery/docs/reference/bq-cli-reference#bq_partition) command in the bq command-line tool.

bq --location=**LOCATION** partition \  
  --time\_partitioning\_type=**PARTION\_TYPE** \  
  --time\_partitioning\_expiration **INTEGER** \  
  **PROJECT\_ID**:**SOURCE\_DATASET**.**SOURCE\_TABLE** \  
  **PROJECT\_ID**:**DESTINATION\_DATASET**.**DESTINATION\_TABLE**

Replace the following:

* **LOCATION**: The name of your location. The --location flag is optional.
* **PARTITION\_TYPE**: The partition type. Possible values include DAY, HOUR, MONTH, or YEAR.
* **INTEGER**: The partition expiration time, in seconds. There is no minimum value. The expiration time evaluates to the partition's UTC date plus the integer value. The time\_partitioning\_expiration flag is optional.
* **PROJECT\_ID**: Your project ID.
* **SOURCE\_DATASET**: The dataset that contains the date-sharded tables.
* **SOURCE\_TABLE**: The prefix of your date-sharded tables.
* **DESTINATION\_DATASET**; The dataset for the new partitioned table.
* **DESTINATION\_TABLE**; The name of the partitioned table to create.

The partition command does not support the --label, --expiration, or --description flags. You can add labels, a table expiration, and a description to the table after it is created.

When you run the partition command, BigQuery creates a copy job is created that generates partitions from the sharded tables.

The following example creates an ingestion-time partitioned table named mytable\_partitioned from a set of date-sharded tables prefixed with sourcetable\_. The new table is partitioned daily, with a partition expiration of 259,200 seconds (3 days).

bq partition \  
--time\_partitioning\_type=DAY \  
--time\_partitioning\_expiration 259200 \  
mydataset.sourcetable\_ \  
mydataset.mytable\_partitioned

If the date-sharded tables were sourcetable\_20180126 and sourcetable\_20180127, this command would create the following partitions: mydataset.mytable\_partitioned$20180126 and mydataset.mytable\_partitioned$20180127.

# Managing partitioned tables

This document describes how to manage partitioned tables in BigQuery.

**Note:** The information in [Managing tables](https://cloud.google.com/bigquery/docs/managing-tables) also applies to partitioned tables.

## Get partition metadata

You can get information about partitioned tables in the following ways:

* Use the [INFORMATION\_SCHEMA.PARTITIONS](https://cloud.google.com/bigquery/docs/information-schema-tables#partitions_view) view ([Preview](https://cloud.google.com/products#product-launch-stages)).
* Use the \_\_PARTITIONS\_SUMMARY\_\_ meta-table (legacy SQL only).

### Getting partition metadata using **INFORMATION\_SCHEMA** views

When you query the INFORMATION\_SCHEMA.PARTITIONS view, the query results contain one row for each partition. For example, the following query lists all of the table partitions in the dataset named mydataset:

SELECT table\_name, partition\_id, total\_rows  
FROM `mydataset.INFORMATION\_SCHEMA.PARTITIONS`  
WHERE partition\_id IS NOT NULL

For more information, see [INFORMATION\_SCHEMA.PARTITIONS](https://cloud.google.com/bigquery/docs/information-schema-tables#partitions_view).

### Getting partition metadata using meta-tables

In legacy SQL, you can get metadata about table partitions by querying the \_\_PARTITIONS\_SUMMARY\_\_ meta-table. Meta-tables are read-only tables that contain metadata.

Query the \_\_PARTITIONS\_SUMMARY\_\_ meta-table as follows:

#legacySQL  
SELECT  
  **column**  
FROM  
  [**dataset.table**$\_\_PARTITIONS\_SUMMARY\_\_]

**Note:** Standard SQL does not support the partition decorator separator (**$**), so you cannot query **\_\_PARTITIONS\_SUMMARY\_\_** in standard SQL.

The \_\_PARTITIONS\_SUMMARY\_\_ meta-table has the following columns:

|  |  |
| --- | --- |
| **Value** | **Description** |
| project\_id | Name of the project. |
| dataset\_id | Name of the dataset. |
| table\_id | Name of the time-partitioned table. |
| partition\_id | Name (date) of the partition. |
| creation\_time | The time at which the partition was created, in milliseconds since January 1, 1970 UTC. |
| last\_modified\_time | The time at which the partition was last modified, in milliseconds since January 1, 1970 UTC. |

At a minimum, to run a query job that uses the \_\_PARTITIONS\_SUMMARY\_\_ meta- table, you must be granted bigquery.jobs.create permissions and bigquery.tables.getData permissions.

For more information on IAM roles in BigQuery, see [Access control](https://cloud.google.com/bigquery/access-control).

## Set the partition expiration

When you create a table partitioned by ingestion time or time-unit column, you can specify a partition expiration. This setting specifies how long BigQuery keeps the data in each partition. The setting applies to all partitions in the table, but is calculated independently for each partition based on the partition time.

A partition's expiration time is calculated from the partition boundary in UTC. For example, with daily partitioning, the partition boundary is at midnight (00:00:00 UTC). If the table's partition expiration is 6 hours, then each partition expires at 06:00:00 UTC the following day. When a partition expires, BigQuery deletes the data in that partition.

You can also specify a [default partition expiration](https://cloud.google.com/bigquery/docs/updating-datasets#partition-expiration) at the dataset level. If you set the partition expiration on a table, then the value overrides the default partition expiration. If you don't specify any partition expiration (on the table or dataset), then partitions never expire.

**Note:** Integer-range partitioned tables do not support partition expiration times.

If you set a table expiration, that value takes precedence over the partition expiration. For example, if the table expiration is set to 5 days, and the partition expiration is set to 7 days, then the table and all partitions in it are deleted after 5 days.

At any point after a table is created, you can update the table's partition expiration. The new setting applies to all partitions in that table, regardless of when they were created. Existing partitions expire immediately if they are older than the new expiration time.

When a partition expires, the data in that partition is no longer available for querying and you are not charged for storage of that partition. BigQuery eventually deletes the expired partition. Until then, the partition counts for purposes of [table quotas](https://cloud.google.com/bigquery/quotas#partitioned_tables). To delete a partition immediately, you can [manually delete the partition](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#delete_a_partition).

For projects that were created before December 13, 2016, partition expiration is based on the last date that the partition was modified. This behavior applies to both existing and new tables created in the project. To migrate your project to the newer behavior, open a request in the [BigQuery issue tracker](https://issuetracker.google.com/issues/new?component=187149).

### Update the partition expiration

To update a partitioned table's partition expiration:

[Console](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#console)[SQL](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#sql)[bq](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#bq)[API](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#api)

Issue the bq update command with the --time\_partitioning\_expiration flag. If you are updating a partitioned table in a project other than your default project, add the project ID to the dataset name in the following format: **project\_id:dataset**.

bq update \  
--time\_partitioning\_expiration **integer** \  
--time\_partitioning\_type **unit\_time** \  
**project\_id:dataset.table**

Where:

* **integer** is the default lifetime (in seconds) for the table's partitions. There is no minimum value. The expiration time evaluates to the partition's date plus the integer value. If you specify 0, the partition expiration is removed, and the partition never expires. Partitions with no expiration must be manually deleted.
* **unit\_time** is either DAY, HOUR, MONTH, or YEAR, based on the table's partitioning granularity. This value must match the granularity that you set when you created the table.
* **project\_id** is your project ID.
* **dataset** is the name of the dataset that contains the table you're updating.
* **table** is the name of the table you're updating.

Examples:

Enter the following command to update the expiration time of partitions in mydataset.mytable to 5 days (432000 seconds). mydataset is in your default project.

bq update --time\_partitioning\_expiration 432000 mydataset.mytable

Enter the following command to update the expiration time of partitions in mydataset.mytable to 5 days (432000 seconds). mydataset is in myotherproject, not your default project.

bq update \  
--time\_partitioning\_expiration 432000 \  
myotherproject:mydataset.mytable

## Set partition filter requirements

When you create a partitioned table, you can require that all queries on the table must include a predicate filter (a WHERE clause) that filters on the partitioning column. This setting can improve performance and reduce costs, because BigQuery can use the filter to prune partitions that don't match the predicate.

For information on adding the **Require partition filter** option when you create a partitioned table, see [Creating partitioned tables](https://cloud.google.com/bigquery/docs/creating-partitioned-tables).

If a partitioned table has the **Require partition filter** setting, then every query on that table must include at least one predicate that only references the partitioning column. Queries without such a predicate return the following error:

Cannot query over table '**project\_id.dataset.table**' without a filter that can be used for partition elimination.

For more information, see [Querying partitioned tables](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#querying_partitioned_tables_2).

### Update the partition filter requirement

If you don't enable the **Require partition filter** option when you create the partitioned table, you can update the table to add the option.

[Console](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#console)[SQL](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#sql)[bq](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#bq)[API](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#api)[Java](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#java)

To update a partitioned table to require partition filters by using the bq command-line tool, enter the bq update command and supply the --require\_partition\_filter flag.

To update a partitioned table in a project other than your default project, add the project ID to the dataset in the following format: **project\_id:dataset**.

For example:

To update mypartitionedtable in mydataset in your default project, enter:

bq update --require\_partition\_filter mydataset.mytable

To update mypartitionedtable in mydataset in myotherproject, enter:

bq update --require\_partition\_filter myotherproject:mydataset.mytable

## Copy a partitioned table

The process for copying a partitioned table is the same as the process for copying a standard table. For more information, see [Copying a table](https://cloud.google.com/bigquery/docs/managing-tables#copy-table).

When you copy a partitioned table, note the following:

* **Copying a partitioned table to a new destination table**

All of the partitioning information is copied with the table. The new table and the old table will have identical partitions.

* **Copying a non-partitioned table into an existing partitioned table**

This operation is only supported for ingestion-time partitioning. BigQuery copies the source data into the partition that represents the current date. This operation is not supported for time-unit column-partitioned or integer-range partitioned tables.

* **Copying a partitioned table into another partitioned table**

The partition specifications for the source and destination tables must match.

* **Copying a partitioned table into a non-partitioned table**

The destination table remains unpartitioned.

* **Copying multiple partitioned tables**

If you copy multiple source tables into a partitioned table in the same job, the source tables can't contain a mixture of partitioned and non-partitioned tables.

If all of the source tables are partitioned tables, the partition specifications for all source tables must match the destination table's partition specification.

When you copy to an existing table, you can specify whether to append or overwrite the destination table.

## Copy individual partitions

You can copy the data from one or more partitions to another table.

**Note:** The required permissions are the same as for [copying a table](https://cloud.google.com/bigquery/docs/managing-tables#required_permissions_2).

[Console](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#console)[bq](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#bq)[API](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#api)

To copy a partition, use the bq command-line tool's bq cp (copy) command with a partition decorator ($**date**) such as $20160201.

Optional flags can be used to control the write disposition of the destination partition:

* -a or --append\_table appends the data from the source partition to an existing table or partition in the destination dataset.
* -f or --force overwrites an existing table or partition in the destination dataset and doesn't prompt you for confirmation.
* -n or --no\_clobber returns the following error message if the table or partition exists in the destination dataset: Table '<var>project\_id:dataset.table</var> or <var>table$date</var>' already exists, skipping. If -n is not specified, the default behavior is to prompt you to choose whether to replace the destination table or partition.
* --destination\_kms\_key is the customer-managed Cloud KMS key used to encrypt the destination table or partition.

The cp command does not support the --time\_partitioning\_field or --time\_partitioning\_type flags. You cannot use a copy job to convert an ingestion-time partitioned table into a partitioned table.

--destination\_kms\_key is not demonstrated here. See [Protecting data with Cloud KMS keys](https://cloud.google.com/bigquery/docs/customer-managed-encryption) for more information.

If the source or destination dataset is in a project other than your default project, add the project ID to the dataset names in the following format: **project\_id:dataset**.

(Optional) Supply the --location flag and set the value to your [location](https://cloud.google.com/bigquery/docs/dataset-locations).

bq --location=**location** cp \  
-a -f -n \  
**project\_id:dataset.source\_table$source\_partition** \  
**project\_id:dataset.destination\_table$destination\_partition**

Where:

* **location** is the name of your location. The --location flag is optional. For example, if you are using BigQuery in the Tokyo region, you can set the flag's value to asia-northeast1. You can set a default value for the location using the [.bigqueryrc file](https://cloud.google.com/bigquery/docs/bq-command-line-tool#setting_default_values_for_command-line_flags).
* **project\_id** is your project ID.
* **dataset** is the name of the source or destination dataset.
* **source\_table** is the table you're copying.
* **source\_partition** is the partition decorator of the source partition.
* **destination\_table** is the name of the table in the destination dataset.
* **destination\_partition** is the partition decorator of the destination partition.

Examples:

**Note:** The partition decorator separator ($) is a special variable in the unix shell. You might have to escape the decorator when you use the command- line tool. The following examples escape the partition decorator: **mydataset.table\$20160519**, **'mydataset.table$20160519'**.

**Copying a partition to a new table**

Enter the following command to copy the January 30, 2018 partition from mydataset.mytable to a new table — mydataset.mytable2. mydataset is in your default project.

bq cp -a 'mydataset.mytable$20180130' mydataset.mytable2

**Copying a partition to a non-partitioned table**

Enter the following command to copy the January 30, 2018 partition from mydataset.mytable to a non-partitioned table — mydataset2.mytable2. The -a shortcut is used to append the partition's data to the non-partitioned destination table. Both datasets are in your default project.

bq cp -a 'mydataset.mytable$20180130' mydataset2.mytable2

Enter the following command to copy the January 30, 2018 partition from mydataset.mytable to a non-partitioned table — mydataset2.mytable2. The -f shortcut is used to overwrite the non-partitioned destination table without prompting.

bq --location=US cp -f 'mydataset.mytable$20180130' mydataset2.mytable2

**Copying a partition to another partitioned table**

Enter the following command to copy the January 30, 2018 partition from mydataset.mytable to another partitioned table — mydataset2.mytable2. The -a shortcut is used to append the partition's data to the destination table. Since no partition decorator is specified on the destination table, the source partition key is preserved and the data is copied to the January 30, 2018 partition in the destination table. You can also specify a partition decorator on the destination table to copy data to a specific partition. mydataset is in your default project. mydataset2 is in myotherproject, not your default project.

bq --location=US cp \  
-a \  
'mydataset.mytable$20180130' \  
myotherproject:mydataset2.mytable2

Enter the following command to copy the January 30, 2018 partition from mydataset.mytable to the February 20, 2018 partition of another partitioned table — mydataset2.mytable2. The -f shortcut is used to overwrite the February 20, 2018 partition in the destination table without prompting. If no partition decorator is used, all data in the destination table is overwritten. mydataset is in your default project. mydataset2 is in myotherproject, not your default project.

bq cp \  
-f \  
'mydataset.mytable$20180130' \  
'myotherproject:mydataset2.mytable2$20180220'

Enter the following command to copy the January 30, 2018 partition from mydataset.mytable to another partitioned table — mydataset2.mytable2. mydataset is in your default project. mydataset2 is in myotherproject, not your default project. If there is data in the destination table, the default behavior is to prompt you to overwrite.

bq cp \  
'mydataset.mytable$20180130' \  
myotherproject:mydataset2.mytable2

**Note:** The **bq cp** command with a partition decorator works on column-based partitions in which the source partition and destination partition are identical. The **bq cp** command also works on ingestion-time based partitions where the partition represents either the same time unit or a coarser time unit that contains the source partition. For example, if **$20180130** is the source partition decorator, valid destination partition decorators include **$20180130**, **$201801**, and **$2018**. To copy a column-based partition to a completely different partition decorator or to a time-unit partition with finer granularity, use an [**INSERT SELECT** statement](https://cloud.google.com/bigquery/docs/reference/standard-sql/dml-syntax#insert_statement).

To copy multiple partitions, specify them as a comma-separated list:

bq cp \  
'mydataset.mytable$20180130,mydataset.mytable$20180131' \  
myotherproject:mydataset.mytable2

## Delete a partition

You can delete an individual partition from a partitioned table. However, you can't delete the special \_\_NULL\_\_ or \_\_UNPARTITIONED\_\_ partitions.

You can only delete one partition at a time.

**Note:** The required permissions are the same as for [deleting a table](https://cloud.google.com/bigquery/docs/managing-tables#required_permissions_3).

You can delete a partition by specifying the partition's decorator unless it is one of the two [special partitions](https://cloud.google.com/bigquery/docs/partitioned-tables#date_timestamp_partitioned_tables).

To delete a partition in a partitioned table:

[Console](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#console)[bq](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#bq)[API](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#api)

Use the bq rm command with the --table flag (or -t shortcut) and specify the partition decorator to delete a specific partition.

bq rm --table **project\_id**:**dataset**.**table**$**partition**

Where:

* **project\_id** is your project ID. If omitted, your default project is used.
* **dataset** is the name of the dataset that contains the table.
* **table** is the name of the table.
* **partition** is the partition decorator of the partition you're deleting.

Partition decorators have the following format, depending on the type of partitioning:

* Hourly partition: yyyymmddhh. Example: $2016030100.
* Daily partition: yyyymmdd. Example: $20160301.
* Monthly partition: yyyymm. Example: $201603.
* Yearly partition: yyyy. Example: $2016.
* Integer range partition: Start of the partition range. Example: $20.

The bq command-line tool prompts you to confirm the action. To skip the confirmation, use the --force flag (or -f shortcut).

**Note:** The partition decorator separator ($) is a special variable in the unix shell. You might have to escape the decorator when you use the command- line tool. The following examples escape the partition decorator: **mydataset.table\$20160519**, **'mydataset.table$20160519'**.

Examples:

Delete the partition for March 1, 2016 in a daily partitioned table named mydataset.mytable in your default project:

bq rm --table 'mydataset.mytable$20160301'

Delete the partition for March, 2016 in a monthly partitioned table:

bq rm --table 'mydataset.mytable$201603'

Delete the integer range starting at 20 in an integer range partitioned table named mydataset.mytable:

bq rm --table 'mydataset.mytable$20'

# Managing partitioned table data

This document describes how to manage partitioned table data in BigQuery.

**Note:** The information in [Managing table data](https://cloud.google.com/bigquery/docs/managing-table-data) also applies to partitioned tables.

## Partition decorators

Partition decorators enable you to reference a partition in a table. For example, you can use them to [write data](https://cloud.google.com/bigquery/docs/managing-partitioned-table-data#write-to-partition) to a specific partition.

A partition decorator has the form table\_name$partition\_id where the format of the partition\_id segment depends on the type of partitioning:

| Partitioning type | Format | Example |
| --- | --- | --- |
| Hourly | yyyymmddhh | my\_table$2021071205 |
| Daily | yyyymmdd | my\_table$20210712 |
| Monthly | yyyymm | my\_table$202107 |
| Yearly | yyyy | my\_table$2021 |
| Integer range | range\_start | my\_table$40 |

## Browse the data in a partition

To browse the data in a specified partition, use the [bq head](https://cloud.google.com/bigquery/docs/reference/bq-cli-reference#bq_head) command with a partition decorator.

For example, the following command lists all fields in the first 10 rows of the my\_dataset.my\_table in the 2018-02-24 partition:

    bq head --max\_rows=10 'my\_dataset.my\_tablee$20180224'

## Write data to a specific partition

You can load data to a specific partition by using the [bq load](https://cloud.google.com/bigquery/docs/reference/bq-cli-reference#bq_load) command with a partition decorator. The following example writes data into the 20160501 (May 1, 2016) partition of an existing table, assuming the table is already partitioned by date:

bq load --source\_format=CSV 'my\_dataset.my\_table$20160501' data.csv

You can also write the results of a query to a specific partition:

bq query \  
  --use\_legacy\_sql=false  \  
  --destination\_table='my\_table$20160501' \  
  --append\_table=true \  
  'SELECT \* FROM my\_dataset.another\_table'

With ingestion-time partitioning, you can use this technique to load older data into the partition that corresponds to the time when the data was originally created.

You can also use this technique to adjust for time zones. By default, ingestion- time partitions are based on UTC time. If you want the partition time to match a particular time zone, you can use partition decorators to offset the UTC ingestion time. For example, if you are on Pacific Standard Time (PST), you can load data that was generated on May 1, 2016 PST into the partition for that date by using the corresponding partition decorator, $20160501.

For time-unit column and integer-range partitioned tables, the partition ID specified in the decorator must match the data being written. For example, if the table is partitioned on a DATE column, the decorator must match the value in that column. Otherwise, an error occurs. However, if you know beforehand that your data is in a single partition, specifying the partition decorator can improve write performance.

For more information on loading data, see [Introduction to loading data into BigQuery](https://cloud.google.com/bigquery/docs/loading-data).

## Stream data into partitioned tables

For information about streaming data into a partitioned table by using the [tabledata.insertAll](https://cloud.google.com/bigquery/docs/reference/rest/v2/tabledata/insertAll) method, see [Streaming into partitioned tables](https://cloud.google.com/bigquery/streaming-data-into-bigquery#streaming_into_partitioned_tables).

## Export table data

Exporting all data from a partitioned table is the same process as exporting data from a non-partitioned table. For more information, see [Exporting table data](https://cloud.google.com/bigquery/docs/exporting-data).

To export data from an individual partition, append the partition decorator to the table name. For example, my\_table$20160201.

You can also export data from the [\_\_NULL\_\_ and \_\_UNPARTITIONED\_\_](https://cloud.google.com/bigquery/docs/partitioned-tables#date_timestamp_partitioned_tables) partitions by appending the partition names to the table name. For example, my\_table$\_\_NULL\_\_ or my\_table$\_\_UNPARTITIONED\_\_.

# Querying partitioned tables

You can query partitioned tables by:

* Using the Cloud Console
* Using the bq command-line tool's bq query command
* Calling the [jobs.insert](https://cloud.google.com/bigquery/docs/reference/rest/v2/jobs/insert) API method and configuring a [query job](https://cloud.google.com/bigquery/docs/reference/rest/v2/jobs#configuration.query)
* Using the client libraries

For more information on running queries, see [Running interactive and batch queries](https://cloud.google.com/bigquery/docs/running-queries).

## Before you begin

Grant Identity and Access Management (IAM) roles that give users the necessary permissions to perform each task in this document.

### Required permissions

To query partitioned tables, you need IAM permissions to query tables and run query jobs.

#### Permissions to query a table

To query a table, you need the bigquery.tables.getData IAM permission.

Each of the following predefined IAM roles includes the permissions that you need in order to create a view:

* roles/bigquery.dataViewer
* roles/bigquery.dataOwner
* roles/bigquery.dataEditor
* roles/bigquery.admin

Additionally, if you have the bigquery.datasets.create permission, you can query tables and views in the datasets that you create.

#### Permissions to run a query job

To run a query job, you need the bigquery.jobs.create IAM permission.

Each of the following predefined IAM roles includes the permissions that you need in order to run a query job:

* roles/bigquery.user
* roles/bigquery.jobUser
* roles/bigquery.admin

For more information on IAM roles and permissions in BigQuery, see [Predefined roles and permissions](https://cloud.google.com/bigquery/access-control).

## Ingestion-time partitioned table pseudo columns

When you create an [ingestion-time partitioned table](https://cloud.google.com/bigquery/docs/creating-partitioned-tables), two pseudo columns are added to the table: a \_PARTITIONTIME pseudo column and a \_PARTITIONDATE pseudo column. The \_PARTITIONTIME pseudo column contains a date-based timestamp for data that is loaded into the table. The \_PARTITIONDATE pseudo column contains a date representation. Both pseudo column names are reserved, which means that you cannot create a column with either name in any of your tables.

\_PARTITIONTIME and \_PARTITIONDATE are available only in ingestion-time partitioned tables. Partitioned tables do not have pseudo columns. For information on querying partitioned tables, see [Querying partitioned tables](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#querying_partitioned_tables_2).

### The \_PARTITIONTIME pseudo column

The \_PARTITIONTIME pseudo column contains a timestamp that is based on UTC time and represents the number of microseconds since the unix epoch. For example, if data is appended to a table on April 15, 2016, 08:15:00 UTC, all of the rows of data that are appended on that day have the \_PARTITIONTIME column that contains one of the following values: + TIMESTAMP("2016-04-15 08:00:00") for hourly partitioned tables. + TIMESTAMP("2016-04-15") for daily partitioned tables. + TIMESTAMP("2016-04-01") for monthly partitioned tables. + TIMESTAMP("2016-01-01") for yearly partitioned tables.

To query the \_PARTITIONTIME pseudo column, you must use an alias. For example, the following query selects \_PARTITIONTIME by assigning the alias pt to the pseudo column:

SELECT  
  \_PARTITIONTIME AS pt,  
  **column**  
FROM  
  **dataset.table**

Where:

* **column** is the name of a column to query. You can specify multiple columns as a comma-separated list.
* **dataset** is the dataset containing the partitioned table.
* **table** is the partitioned table.

Data in the [streaming buffer](https://cloud.google.com/bigquery/streaming-data-into-bigquery) has NULL values in the \_PARTITIONTIME column.

### The \_PARTITIONDATE pseudo column

The \_PARTITIONDATE pseudo column contains the UTC date corresponding to the value in the \_PARTITIONTIME pseudo column. This column is not supported in hourly, monthly, or yearly partitioned tables.

To query the \_PARTITIONDATE pseudo column, you must use an alias. For example, the following query selects \_PARTITIONDATE by assigning the alias pd to the pseudo column:

SELECT  
  \_PARTITIONDATE AS pd,  
  **column**  
FROM  
  **dataset.table**

Where:

* **column** is the name of a column to query. You can specify multiple columns as a comma-separated list.
* **dataset** is the dataset containing the partitioned table.
* **table** is the partitioned table.

Data in the [streaming buffer](https://cloud.google.com/bigquery/streaming-data-into-bigquery) has NULL values in the \_PARTITIONDATE column.

## Querying ingestion-time partitioned tables using pseudo columns

When you query data in ingestion-time partitioned tables, you reference specific partitions by specifying the values in the \_PARTITIONTIME or \_PARTITIONDATE pseudo columns. For example:

* \_PARTITIONTIME >= "2018-01-29 00:00:00" AND \_PARTITIONTIME < "2018-01-30 00:00:00"
* \_PARTITIONTIME BETWEEN TIMESTAMP('2016-01-01') AND TIMESTAMP('2016-01-02')

or

* \_PARTITIONDATE >= "2018-01-29" AND \_PARTITIONDATE < "2018-01-30"
* \_PARTITIONDATE BETWEEN '2016-01-01' AND '2016-01-02'

### Limiting partitions queried using pseudo columns

You can use the \_PARTITIONTIME and \_PARTITIONDATE pseudo columns to limit the number of partitions scanned during a query. This is also referred to as pruning partitions. Partition pruning is the mechanism BigQuery uses to eliminate unnecessary partitions from the input scan. The pruned partitions are not included when calculating the bytes scanned by the query, reducing the on-demand analysis cost. In general, partition pruning helps reduce query cost when the filters consist of constant expressions, which can be evaluated at the outset of the query.

**Caution:** Some uses of the **\_PARTITIONTIME** and **\_PARTITIONDATE** pseudo columns do not limit the number of partitions scanned. For more information, see [Pruning (limiting) partitions](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#pruning_limiting_partitions). For examples of queries that include the pseudo columns, but do not limit the number of partitions scanned, see [Pseudo column queries that scan all partitions](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#pseudo_column_queries_that_scan_all_partitions).

For example, the following query scans only the partitions between the dates January 1, 2016 and January 2, 2016 from the partitioned table:

[\_PARTITIONTIME](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#_partitiontime)[\_PARTITIONDATE](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#_partitiondate)

SELECT  
  **column**  
FROM  
  **dataset.table**  
WHERE  
  \_PARTITIONTIME BETWEEN TIMESTAMP('2016-01-01')  
  AND TIMESTAMP('2016-01-02')

#### Partition pruning examples

This example demonstrates limiting the number of scanned partitions using a pseudo column filter in a subquery:

[\_PARTITIONTIME](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#_partitiontime)[\_PARTITIONDATE](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#_partitiondate)

SELECT  
  **column1**,  
  **column2**  
FROM (  
  SELECT  
    **column1**,  
    **column2**  
  FROM  
    **dataset.table**  
  WHERE  
    \_PARTITIONTIME = TIMESTAMP('2016-03-28')) t1  
CROSS JOIN  
  **dataset.table** t2  
WHERE  
  t1.**column2** = "one"

The following query limits the partitions that are queried, based on the first filter condition in the WHERE clause. However, the second filter condition in the WHERE clause does not limit the partitions that are queried because it uses table values, which are dynamic.

[\_PARTITIONTIME](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#_partitiontime)[\_PARTITIONDATE](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#_partitiondate)

SELECT  
  **column**  
FROM  
  **dataset.table2**  
WHERE  
  \_PARTITIONTIME BETWEEN TIMESTAMP('2017-01-01') AND TIMESTAMP('2017-03-01')  
  AND \_PARTITIONTIME = (SELECT MAX(timestamp) from **dataset.table1**)

### Pseudo column queries that scan all partitions

The following examples use pseudo columns but scan all the partitions in a time-unit partitioned table.

In legacy SQL, the \_PARTITIONTIME filter works only when the filter is specified as closely as possible to the table name. For example, the following query scans all partitions in table1 despite the presence of the \_PARTITIONTIME filter:

#legacySQL  
# Scans all partitions on t1  
SELECT  
  t1.**field1**,  
  t2.**field1**  
FROM  
  **dataset.table1** t1  
CROSS JOIN  
  **dataset.table2** t2  
WHERE  
  **table1**.\_PARTITIONTIME = TIMESTAMP('2016-03-28')  
  AND t1.**field1** = "one"

Do not include any other columns in a \_PARTITIONTIME filter. For example, the following query does not limit the partitions that are scanned because field1 is a column in the table and BigQuery can't determine in advance which partitions to select. For more information, see [Pruning (limiting) partitions](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#pruning_limiting_partitions).

# Scans all partitions of table2  
SELECT  
  **field1**  
FROM  
  **dataset.table2**  
WHERE  
  \_PARTITIONTIME + **field1** = TIMESTAMP('2016-03-28');

### Better performance with pseudo columns

To improve query performance, use the \_PARTITIONTIME pseudo column by itself on the left side of a comparison. For example, the following queries process the same amount of data, but the second example can provide better performance.

**Example 1:** The following query can be slower because it combines the pseudo column value with other operations in the WHERE filter.

[Standard SQL](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#standard-sql)[Legacy SQL](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#legacy-sql)

#standardSQL  
/\* Can be slower \*/  
SELECT  
  **field1**  
FROM  
  **dataset.table1**  
WHERE  
  TIMESTAMP\_ADD(\_PARTITIONTIME, INTERVAL 5 DAY) > TIMESTAMP("2016-04-15")

**Example 2:** The following query can perform better because it places the pseudo column by itself on the left side of the filter comparison.

[Standard SQL](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#standard-sql)[Legacy SQL](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#legacy-sql)

#standardSQL  
/\* Often performs better \*/  
SELECT  
  **field1**  
FROM  
  **dataset.table1**  
WHERE  
  \_PARTITIONTIME > TIMESTAMP\_SUB(TIMESTAMP('2016-04-15'), INTERVAL 5 DAY)

Depending on the table size, the second query, which places \_PARTITIONTIME by itself on the left side of the > comparison operator, can provide better performance than the first query. Because the queries process the same amount of data, the number of bytes billed is the same in both cases.

## Querying ingestion-time partitioned tables using a wildcard table

In addition to using the pseudo columns to limit the number of partitions scanned during a query, you can also use the pseudo columns to query a range of partitioned tables using a wildcard table. For information on using a wildcard table with partitioned tables, see [Scanning a range of partitioned tables using \_PARTITIONTIME](https://cloud.google.com/bigquery/docs/querying-wildcard-tables#scanning_a_range_of_ingestion-time_partitioned_tables_using_partitiontime).

## Querying ingestion-time partitioned tables using time zones

The value of \_PARTITIONTIME is based on the UTC date when the field is populated, which means that partitions are divided based on 12:00 AM UTC. If you want to query data based on a time zone other than UTC, you should choose one of the following options before you start loading data into your table.

There are two ways to query data in a partitioned table using a custom, non-UTC, time zone. You can either create a separate timestamp column or you can use partition decorators to load data into a specific partition.

If you use a [timestamp column](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#tracking_timezones_using_a_timestamp), you can use the default UTC-based partitioning and account for time zone differences in your SQL queries. Alternately, if you prefer to have partitions that are grouped by a time zone other than UTC, use [partition decorators to load data](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#loading_data_using_partition_decorators) into partitions based on a different time zone.

### Querying time zones using a timestamp column

To adjust for timezones using a timestamp, create a separate column to store a timestamp that enables you to address rows by the hour or minute.

To query for data based on a timezone other than UTC, use both the \_PARTITIONTIME pseudo column and your custom timestamp column. Using \_PARTITIONTIME limits the table scan to the relevant partitions, and your custom timestamp further limits the results to your timezone. For example, to query data from a partitioned table (mydataset.partitioned\_table) with a timestamp field **MY\_TIMESTAMP\_FIELD** for data added to the table between 2016-05-01 12:00:00 PST and 2016-05-05 14:00:00 PST:

[Standard SQL](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#standard-sql)[Legacy SQL](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#legacy-sql)

#standardSQL  
SELECT  
  **field1**  
FROM  
  **dataset.partitioned\_table**  
WHERE  
  \_PARTITIONTIME BETWEEN TIMESTAMP("2016-05-01")  
  AND TIMESTAMP("2016-05-06")  
  AND TIMESTAMP\_ADD(**MY\_TIMESTAMP\_FIELD**, INTERVAL 8 HOUR) BETWEEN TIMESTAMP("2016-05-01 12:00:00")  
  AND TIMESTAMP("2016-05-05 14:00:00");

## Querying an integer-range partitioned table

Integer-range partitioned tables can only be queried through Standard SQL. When an integer-range partitioned table is queried, if there are filters on the integer partitioning column, the filters will be used to prune the partitions and reduce the query cost.

The following query scans the 3 partitions that start with 30, 40, and 50, for an integer-range partitioned tables with the partitioning specification of customer\_id:0:100:10.

bq query --nouse\_legacy\_sql \  
'SELECT \* FROM mydataset.mytable WHERE customer\_id BETWEEN 30 AND 50'

Here's example output from the bq query command:

+-------------+-------+  
| customer\_id | value |  
+-------------+-------+  
|          40 |    41 |  
|          45 |    46 |  
|          30 |    31 |  
|          35 |    36 |  
|          50 |    51 |  
+-------------+-------+

For this example, each partition has 2 rows, and each row has 2 integer columns, so the query scans 3 \* 2 \* 2 \* 8 = 96 bytes. You can examine the job information:

bq show -j bqjob\_r4fce65fa3381528e\_000001670994aeb6\_1

Here's example output from the bq show command:

  Job Type    State      Start Time      Duration       User Email        Bytes Processed   Bytes Billed   Billing Tier   Labels  
 ---------- --------- ----------------- ---------- --------------------- ----------------- -------------- -------------- --------  
  query      SUCCESS   24 Sep 12:19:58   0:00:01    joe@example.com      96                10485760       1

DML statements are supported. For example:

bq query --nouse\_legacy\_sql \  
'DELETE FROM mydataset.mytable WHERE customer\_id = 30'

Currently, partition pruning is not supported for functions over an integer range partitioned column. As an example, the following query scans the entire table.

bq query --nouse\_legacy\_sql \  
'SELECT \* FROM mydataset.mytable WHERE customer\_id+1 BETWEEN 30 AND 50'

### Table decorators on integer-range partitioned tables

Legacy SQL supports using table decorators to address a partition in an integer range partitioned table. The key to address a range partition is the start of the range.

The following example queries the range partition that starts with 0. The partitioning column is customer\_id and the partition has two rows.

bq query --use\_legacy\_sql=true 'SELECT \* FROM mydataset.mytable$0'  
  
+-------------+-------+  
| customer\_id | value |  
+-------------+-------+  
|           0 |     1 |  
|           5 |     6 |  
+-------------+-------+

## Creating a view using an ingestion-time partitioned table's pseudo columns

To limit the amount of data read by a query to a set of partitions, create a view that contains a filter on the \_PARTITIONTIME or \_PARTITIONDATE pseudo column. For example, the following query can be used to create a view that includes only the most recent seven days of data from a table named dataset.partitioned\_table:

[Standard SQL](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#standard-sql)[Legacy SQL](https://cloud.google.com/bigquery/docs/querying-partitioned-tables#legacy-sql)

#standardSQL  
SELECT  
  \*  
FROM  
  **dataset.partitioned\_table**  
WHERE  
  \_PARTITIONTIME BETWEEN TIMESTAMP\_TRUNC(TIMESTAMP\_SUB(CURRENT\_TIMESTAMP(), INTERVAL 7 \* 24 HOUR),DAY)  
  AND TIMESTAMP\_TRUNC(CURRENT\_TIMESTAMP(), DAY);

For information about creating views, see [Creating views](https://cloud.google.com/bigquery/docs/views).

## Ingestion-time partitioned tables' \_UNPARTITIONED\_ partition

The \_\_UNPARTITIONED\_\_ partition temporarily holds data that is streamed to a partitioned table while it is in the streaming buffer. Data that is streamed directly to a specific partition of a partitioned table does not use the \_\_UNPARTITIONED\_\_ partition. Instead, the data is streamed directly to the partition. See [streaming into partitioned tables](https://cloud.google.com/bigquery/streaming-data-into-bigquery#streaming_into_partitioned_tables) for more information.

To query data in the \_\_UNPARTITIONED\_\_ partition, use the \_PARTITIONTIME pseudo column with the NULL value. For example:

SELECT  
  **column**  
FROM  
  **dataset.table**  
WHERE  
  \_PARTITIONTIME IS NULL

Where:

* **column** is the name of a column to query. You can specify multiple columns as a comma-separated list.
* **dataset** is the dataset containing the partitioned table.
* **table** is the partitioned table.

## Querying partitioned tables

Tables partitioned based on a TIMESTAMP, DATE, DATETIME, or INTEGER column do not have pseudo columns. To limit the number of partitions scanned when querying partitioned tables, use a predicate filter (a WHERE clause). Filters on the partitioning column will be used to prune the partitions and reduce the query cost.

Hourly, monthly, and yearly partitioned tables can only be queried through Standard SQL.

When you create a partitioned table, you can require the use of predicate filters by enabling the **Require partition filter** option. When this option is applied, attempts to query the partitioned table without specifying a WHERE clause produce the following error: Cannot query over table '**project\_id.dataset.table**' without a filter that can be used for partition elimination.

**Note:** There must be at least one predicate that only references a partition column for the filter to be considered eligible for partition elimination. For ingestion-time partitioned tables, this will be either the **\_PARTITIONTIME** or **\_PARTITIONDATE** pseudocolumn. As an example, for a table partitioned on column **partition\_id** with an additional column **f** in its schema, both of the following **WHERE** clauses satisfy the requirement:  
    **WHERE partition\_id = "foo"**  
    **WHERE partition\_id = "foo" AND f = "bar"**

However, **WHERE (partition\_id = "foo" OR f = "bar")** is not sufficient.

For more information about adding the **Require partition filter** option when you create a partitioned table, see [Creating partitioned tables](https://cloud.google.com/bigquery/docs/creating-partitioned-tables).

If you do not enable the **Require partition filter** option when you create a partitioned table, you can [update](https://cloud.google.com/bigquery/docs/managing-partitioned-tables#require-filter) the table to add the option.

### Pruning (limiting) partitions

To limit the partitions that are scanned in a query, use a constant expression in your filter. If you use dynamic expressions in your query filter, BigQuery must scan all of the partitions.

For example, the following query prunes partitions because the filter, WHERE t1.ts=CURRENT\_TIMESTAMP(), contains a constant expression:

#standardSQL  
SELECT  
  t1.**name**,  
  t2.**category**  
FROM  
  **table1** t1  
INNER JOIN  
  **table2** t2  
ON t1.**id\_field** = t2.**field2**  
WHERE  
  t1.**ts** = CURRENT\_TIMESTAMP()

However, the following query doesn't prune partitions, because the filter, WHERE t1.ts = (SELECT timestamp from table where key = 2), is not a constant expression; it depends on the dynamic values of the timestamp and key fields:

#standardSQL  
SELECT  
  t1.**name**,  
  t2.**category**  
FROM  
  **table1** t1  
INNER JOIN  
  **table2** t2  
ON  
  t1.**id\_field** = t2.**field2**  
WHERE  
  t1.**ts** = (SELECT timestamp from **table3** where key = 2)

### Isolate the partition column in your filter

Isolate the partition column when expressing a filter. Filters that require data from multiple fields to compute will not prune partitions. For example, a query with a date comparison using the partitioning column and a second field, or queries containing some field concatenations will not prune partitions.

For example, the following filter does not prune partitions because it requires a computation based on the partitioning ts field and a second field ts2:

WHERE TIMESTAMP\_ADD(ts, INTERVAL 6 HOUR) > ts2