Engineer

# BigQuery

Google's warehouse for querying very large tables

## What it is

BigQuery is a managed SQL analytics service on Google Cloud: nobody provisions a server for it, tables are stored column by column, and each query is paid for either by the bytes it reads or out of compute capacity reserved in advance. It is also sold and searched for as Google BigQuery, and the SQL it speaks is GoogleSQL, a dialect that follows the ANSI standard and adds functions of its own.

The design rests on keeping storage and compute apart. Tables live in Google's storage layer, and a query borrows compute for as long as it runs. Because each column is stored separately, a query that names three columns of a fifty-column table reads those three and leaves the rest on disk. Tuning therefore looks different from tuning a PostgreSQL server. Instead of designing B-tree indexes and sizing memory, the work is the shape of the table: it can be partitioned by a date or timestamp column, by ingestion time or by an integer range, and clustered on up to four columns so that related rows share storage blocks.

BigQuery belongs to the family of data warehouses, and the products it is usually weighed against are Snowflake and Amazon Redshift, plus the option of keeping analytics on the transactional database a company already runs. It is built for queries that sweep millions or billions of rows and return a handful of totals. The single-row lookups and updates an application makes on every request belong in a database such as PostgreSQL or SQL Server, and a common arrangement runs both, with a pipeline carrying data from one to the other.

Use the concept in a decision

Put "What is BigQuery?" to work in a real decision. We can help relate the concept to the system you are building or buying. Sales, operations and finance figures brought together, so two departments stop bringing two answers to the same meeting.

**Explore our services** Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Made a geographical data pipeline 50x faster — 50x faster geographical data pipelines

**Made a geographical data pipeline 50x faster** Get in touch →

# Why it matters

The decision BigQuery puts in front of a team is how it wants to pay for analysis. The bill follows the queries people write, not the size of a machine somebody once chose, and that cuts both ways.

Under on-demand pricing, every query is charged for the bytes it processes. That suits irregular work: a team that runs a few heavy reports a week pays for those reports and for nothing in between, with no server idling overnight. It also gives a careless query a price. A `SELECT *` over a year of sensor readings costs the same whether the analyst wanted one row or all of them. Capacity pricing replaces this with slots, units of compute bought through reservations, which turns the bill into a figure known in advance and makes heavy, steady use cheaper to plan for. A project can move between the two models, so the first choice can be revisited once real usage is visible.

Storage has rules of its own. A table or partition left unmodified for 90 consecutive days moves to the long-term rate, 50 percent lower, without anyone moving it. Each dataset is billed either on the logical, uncompressed size of its data or on the physical, compressed size, and for data that compresses well the second model can be the cheaper one.

Three things deserve weight before signing up. The location of a dataset, a single region or the US or EU multi-region, is set when the dataset is created and stays fixed, so it is a compliance decision taken on the first day. The dialect, the `bq` command-line tool and the client libraries are Google’s own, so queries move to another warehouse with edits rather than unchanged; the data itself leaves easily, since tables export to CSV, Avro and Parquet. And the hiring question is gentler than it looks: analysts who know SQL are productive in days, while the rarer skill is reading a query in bytes as well as in rows.

So when is BigQuery the right call? When data arrives in large, mostly appended volumes such as events, logs or meter readings, when the questions asked of it change from week to week, and when the team would rather not operate database servers for analysis. For data that a single PostgreSQL server holds

comfortably, a warehouse adds a second system, a second bill and a pipeline between the two. The case for it grows with the volume and with the number of people asking questions.

## What it looks like in practice

An estate of more than 100 databases, split between PostgreSQL, Microsoft SQL Server and Google BigQuery, held mostly geographic or time-series data: coordinates, sensor readings, logs and real-time metrics. BigQuery handled the large-scale time-series data, its tables time-partitioned and clustered, with compression and columnar storage applied to keep storage costs down and scans fast. Partitioned, clustered scans in BigQuery, together with indexing, materialized views and caching across the estate, took 40–60% off the response times of GIS and time-series queries.

On a geographical data pipeline that had grown slow, the SQL and the data models in PostgreSQL, MS SQL and Google Cloud BigQuery were analysed first, which surfaced weak indexing, inefficient queries and missing constraints. Each engine then got the tool it has: B-tree and GiST indexes in PostgreSQL, filtered indexes in MS SQL and clustering in BigQuery, alongside normalised models, partitioning and rewritten queries. The founder's achievement record reports that the pipeline as a whole ran 50x faster; it does not specify the benchmark dataset, hardware or comparison procedure.

And on smart-meter data from several regions, more than 500 analysis reports were written in pure SQL against PostgreSQL and BigQuery, with CTEs, window functions, subqueries and geospatial joins over meter IDs, timestamps, coordinates and environmental metadata. Each report shipped with its SQL annotated, so that a colleague could rerun and audit it, and with notes on known data-quality problems such as missing GPS coordinates and corrupted meter values. The reports became a reference across departments and contributed to a 10–15% improvement in operational planning efficiency.

In all three cases BigQuery worked next to PostgreSQL rather than in its place, and each engine was given the work it does best.

# Where it goes wrong

**A `LIMIT 10` query is billed for the whole table.** On a table without clustering, `LIMIT` trims the result, not the bytes read, and `SELECT *` reads every column there is. A quick look at the data becomes a full scan. Name the columns the question needs, and for a glance at sample rows use the console's table preview or `bq head`, which cost nothing.

**A partitioned table gets scanned end to end anyway.** Partitions are skipped only when the filter on the partitioning column is a constant the engine can evaluate before reading. Comparing the column with a subquery's result, wrapping it in a function or doing arithmetic on it sends the query through every partition. Two guards help. A table can require a partition filter, so a query without a usable one stops with an error instead of running. And a dry run reports the bytes a query would process without running it or charging for it:

```
bq query --use_legacy_sql=false --dry_run \
  'SELECT meter_id, kwh
   FROM energy.readings
   WHERE reading_date = DATE "2026-10-01"'
```

**Clustering is added and the bytes read barely move.** Clustering sorts storage blocks by up to four columns in order, and the order matters: a table clustered by order date, country and status is not organised for a filter on country and status alone. Tables under 64 MB gain little from it, and on a clustered table the bytes actually scanned are known only after the query finishes. Order the clustering columns the way queries filter. Where cost has to be capped before anything runs, rely on partitioning, whose pruning a dry run already accounts for, and set a maximum bytes billed so that a query over the limit stops before it is charged.

**A query that joins two datasets is rejected.** Every dataset lives in one location, and a query runs in one location, so every dataset it touches has to share it. Teams that let each project take its own default end up with tables that can only be joined after one of them is copied across. Pick the location once, for compliance and for where the data comes from, and create every dataset there on purpose.

# Further reading

- BigQuery overview — Google's own description of the service, its separate storage and compute layers and what is managed for you.
- Introduction to partitioned tables — the partitioning types, their granularity and their limits.
- Query partitioned tables — which filters prune partitions and which send a query through all of them.
- Introduction to clustered tables — how column order shapes block pruning, and what that means for cost estimates.
- Estimate and control costs — dry runs, maximum bytes billed and the free ways to preview a table.
- BigQuery locations — regions, multi-regions and the rules for queries that touch more than one dataset.

Glossary

Choose for your actual constraints

Bring a representative dataset, the questions it must answer and the current bottleneck. We can agree how to measure quality, throughput and recovery.

**See what it costs** Get in touch →

## Make the terminology practical

Ask how "What is BigQuery?" applies to your organisation. A concrete example from your current system is enough to start the discussion.

**Explore our services**

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

**Carefully Crafted With Love by D.**

**CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N**

**Copyright © 2025 – present · Engineer ApS · All rights reserved.**

**Printed from https://engineer.company/glossary/bigquery/**