CLOUD & INFRASTRUCTURE

BigQuery for Data Engineers

Warehouse design, optimization and cost control on Google Cloud

Houssam Kodad

PDF · DATAFORGE BOOKS

Questions about this sample or the full edition: **support@dataforgebooks.com**

CONTENTS

Table of Contents

CHAPTER 01

How BigQuery Works

BigQuery makes it trivially easy to run a query — and just as easy to run a very expensive one. Its serverless model removes the operational burden of a warehouse, with no clusters to size or patch, but it replaces that burden with a new and unfamiliar skill: writing schemas and queries that work with the engine's architecture instead of against it. This book teaches that skill, so BigQuery stays both fast and predictable as your usage grows.

This chapter opens the hood. We look at how BigQuery separates storage from compute, what the units of compute called slots really are, and how columnar storage shapes everything about cost and performance. These are not academic details; they are the mental model that explains every optimisation and every surprising bill in the rest of the book.

The recurring lesson is that on BigQuery, cost is a design choice. The same question can be answered by a query that scans a terabyte or one that scans a gigabyte, depending on how the table is partitioned and how the query is written. Understanding the engine is what lets you consistently choose the cheap path, and that understanding starts here.

Separation of Storage and Compute

BigQuery's defining architectural choice is that storage and compute are completely decoupled. Your data sits in a distributed columnar store, and queries are executed by a separate, elastic pool of compute provisioned on demand. You pay for the two independently, and neither constrains the other — you can store a petabyte and query it with massive parallelism, or store a megabyte and query it the same way.

This separation is why BigQuery scales so smoothly and why a tiny table and a huge one feel identical to query. It also explains the two-part cost model that confuses newcomers: storage is billed by the byte stored over time, while compute is billed by the work each query does. Optimising each is a distinct discipline, and conflating them is the source of much avoidable expense and confusion.

Slots and the Execution Model

A slot is BigQuery's unit of compute — a virtual worker that executes a piece of a query. When you run a query, BigQuery decomposes it into stages and distributes the work across whatever slots are available, so more slots mean more parallelism and faster results. Contention for a finite pool of slots is exactly what makes queries queue and slow down under heavy concurrent load.

Understanding slots is the key to both performance and cost at scale. On-demand pricing hides slots behind the bytes-scanned metric, while the capacity-based editions expose them directly as a resource you reserve and share. Either way, knowing that your query is competing for a limited number of workers explains its behaviour, and choosing the right pricing model for your workload is a decision we return to repeatedly.

Columnar Storage Internals

BigQuery stores each column separately and compressed, and this single fact drives a huge fraction of everything in the book. When you query, BigQuery reads only the columns you actually reference, which is why SELECT star is the cardinal sin of BigQuery cost. A query touching three columns of a hundred-column table reads roughly three percent of the data and is billed accordingly.

Once the columnar model is in your bones, much of BigQuery optimisation becomes obvious rather than mysterious. Selecting only needed columns, avoiding unnecessary scans, and understanding why some operations are cheap while others are dear all follow from how the data is physically laid out. Partitioning and clustering, which we cover next, extend this by letting BigQuery skip whole blocks of rows as well as unneeded columns.

```
-- Reads two columns of one day's partition: a few cents.
SELECT user_id, event_type
FROM   `project.dataset.events`
WHERE  event_date = '2026-06-01';

-- Reads every column of every row, every time: avoid in production.
SELECT * FROM `project.dataset.events`;
```

Partitioning and Clustering

Partitioning and clustering are the two most powerful cost levers BigQuery gives you, and using them well is the difference between a cheap warehouse and an expensive one. Partitioning splits a table by a column — almost always a date — so that a query filtering on that column reads only the relevant partitions and ignores the rest of history entirely.

Clustering goes a level finer, physically sorting data within each partition by columns you frequently filter or aggregate on, so BigQuery can skip blocks that cannot contain matching rows. Together they can cut the data a query scans by orders of magnitude. Designing tables with the right partition and cluster keys, based on how they will actually be queried, is among the highest-value skills this book teaches.

Getting Data In

Data reaches BigQuery in several ways, each suited to a different need. Batch loads move large volumes efficiently and cheaply and are the workhorse for most pipelines. The streaming API inserts rows in near real time for low-latency use cases. External tables let you query data sitting in cloud storage without loading it at all, useful for occasional access to large, rarely-touched datasets.

The decision among them is mostly about latency versus cost, and idempotency is the recurring concern regardless of method. Pipelines fail and retry, and an ingestion design that double-counts rows on retry will quietly corrupt your numbers. We build ingestion patterns that are safe to re-run, because in a system that will inevitably retry, idempotency is not a nicety but a requirement.

Writing Queries That Scan Less

Because cost is driven by data scanned, query writing on BigQuery is partly an exercise in reading as little as possible. The habits are simple but consequential: select only the columns you need, filter on the partition column so pruning kicks in, and avoid patterns that defeat partition elimination. The query plan and the dry-run byte estimate tell you exactly how much a query will read before you run it.

Beyond pruning, understanding how BigQuery executes joins, window functions and aggregations at scale lets you write queries that finish quickly without exhausting slots. Approximate aggregation functions trade a sliver of accuracy for large savings on huge datasets. We cover these techniques not as micro-optimisations but as the everyday craft of using BigQuery economically.

Cost Control That Sticks

Individual optimisations help, but on a team you need guardrails that prevent expensive mistakes systemically. BigQuery provides custom quotas that cap how much data a user or project can scan, billing alerts that warn before spend runs away, and labels that make cost attributable to teams and pipelines. Configuring these is how you turn good intentions into enforced limits.

The information schema and audit logs let you see exactly which queries and users drive cost, turning cost management from guesswork into evidence-based action. The chapters ahead build on the engine fundamentals from this one — storage, slots, partitioning, ingestion — toward a BigQuery practice that is fast, governed, and predictable in its spend, which is exactly what a data engineer responsible for the bill needs.

This is a free sample

You've reached the end of the sample chapter.

Get the complete book — every chapter, fully worked — at **dataforgebooks.com**.

FULL EDITION · 208 PAGES · PDF

Read the full title at **dataforgebooks.com**

Questions? support@dataforgebooks.com