

BigQuery Cost Report

Project: **demo-warehouse** · Generated 2026-10-15T22:42:06.429955+00:00

Pricing: configured value (buyer's rate) (\$6.25 per TiB scanned, on-demand). Read-only; no data leaves this machine.
About this report: it was produced in demo mode from built-in synthetic data. No BigQuery project was contacted, no credentials were used and nothing was billed.
Note: Demo data is synthetic and fixed: the same command always produces the same findings and the same numbers, so this report can be regenerated and checked against the published sample. Only the generation timestamp differs between runs.

Total jobs	Total bytes billed	Estimated query cost
27076	43,290.61 GB	\$264.22

Estimated query cost is before the 1 TiB/month free tier and excludes storage.
Result cache: 9,865 of 27,076 jobs (36%) were served from BigQuery's result cache and billed nothing.
Errored jobs: 4 job(s) ended in an error but still billed \$0.03 (4.57 GB). Failed queries are charged for what they scanned.
Capacity comparison (indicative): this workload used 11.2 slot-hours; at an Editions pay-as-you-go rate of \$0.04/slot-hour that computes to ~\$0.45 vs \$264.22 on-demand. Real capacity bills depend on edition, commitments and autoscaling - treat this as a pointer, not a quote.

Cost attribution

Top query spenders (people)

Name	Jobs	Bytes	Cost
analyst.one@example.com	4779	1,017.66 GB	\$6.21
analyst.three@example.com	4875	994.52 GB	\$6.07
analyst.two@example.com	4780	902.16 GB	\$5.51

Top service accounts

Name	Jobs	Bytes	Cost
dbt-runner@demo-warehouse.iam.gserviceaccount.com	7976	38,431.22 GB	\$234.57
bi-reader@demo-warehouse.iam.gserviceaccount.com	4666	1,945.05 GB	\$11.87

Top dbt models

Name	Jobs	Bytes	Cost
model.dbt_dw.fact_sales	699	1,904.72 GB	\$11.63

Top datasets/tables

Name	Jobs	Bytes	Cost	Top consumer
dw_prod.fact_sales	11566	5,769.54 GB	\$35.21	dbt-runner@demo-warehouse.iam.gserviceaccount.com (52%)
dw_prod.fact_events_raw	210	2,621.97 GB	\$16.00	dbt-runner@demo-warehouse.iam.gserviceaccount.com (55%)
dw_prod_stg.stg_sales	699	1,904.72 GB	\$11.63	dbt-runner@demo-warehouse.iam.gserviceaccount.com (100%)
dw_archive.fact_sales_2024	2796	1,406.32 GB	\$8.58	dbt-runner@demo-warehouse.iam.gserviceaccount.com (100%)
dw_prod_stg.stg_customers	5918	603.82 GB	\$3.69	analyst.three@example.com (26%)
dw_prod.fact_orders	126	437.04 GB	\$2.67	dbt-runner@demo-warehouse.iam.gserviceaccount.com (100%)
dw_prod.dim_customer	5919	297.52 GB	\$1.82	analyst.three@example.com (27%)

Top queries (by cost)

Query	Jobs	Bytes	Cost
with table_shards_stage as (select info_tables.table_catalog as table_catalog, info_tables.table_sc	2638	31,656.00 GB	\$193.21
select event_type, count(*) as n from `demo-warehouse.dw_prod.fact_events_raw` group by event_type	210	2,621.97 GB	\$16.00
select * from `demo-warehouse.dw_prod.fact_sales` where erp_id is not null order by erp_id	633	2,379.50 GB	\$14.52
dbt model model.dbt_dw.fact_sales	699	1,904.72 GB	\$11.63
select erp_id, legal_unit, sum(amount) as total from `demo-warehouse.dw_prod.fact_sales` group by er	2214	1,678.47 GB	\$10.24
select s.erp_id, a.amount from `demo-warehouse.dw_prod.fact_sales` as s join `demo-warehouse.dw_arch	2796	1,406.32 GB	\$8.58
select count(*) as rows_loaded from `demo-warehouse.dw_prod_stg.stg_customers`	5918	603.82 GB	\$3.69
select sum(amount) as revenue from `demo-warehouse.dw_prod.fact_orders`	126	437.04 GB	\$2.67
select day, sum(amount) as total from `demo-warehouse.dw_prod.fact_sales` where day >= " group by d	5919	300.68 GB	\$1.84
select c.legal_unit, count(*) as customers from `demo-warehouse.dw_prod.dim_customer` as c group by	5919	297.52 GB	\$1.82

Identical statements are grouped: dbt jobs by their model name, other queries by their SQL with comments and literal values removed (shortened to fit).

Cost by day (oldest first)

Date	Jobs	Bytes	Cost	Relative cost	Top user
2026-09-15	70	123.82 GB	\$0.76	..	dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-16	896	1,546.41 GB	\$9.44		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-17	934	1,550.67 GB	\$9.46		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-18	894	1,572.66 GB	\$9.60		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-19	894	1,529.68 GB	\$9.34		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-20	889	1,482.52 GB	\$9.05		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-21	903	1,657.67 GB	\$10.12		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-22	894	1,505.85 GB	\$9.19		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-23	928	1,532.94 GB	\$9.36		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-24	867	1,501.65 GB	\$9.17		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-25	944	1,576.88 GB	\$9.62		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-26	879	1,540.43 GB	\$9.40		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-27	849	1,515.85 GB	\$9.25		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-28	901	1,489.70 GB	\$9.09		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-09-29	951	1,533.22 GB	\$9.36		dbt-runner@demo-warehouse.iam.gserviceaccount.com

Date	Jobs	Bytes	Cost	Relative cost	Top user
2026-09-30	923	1,490.16 GB	\$9.10		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-01	954	1,584.06 GB	\$9.67		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-02	926	1,533.41 GB	\$9.36		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-03	870	1,536.44 GB	\$9.38		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-04	940	1,585.84 GB	\$9.68		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-05	921	1,551.85 GB	\$9.47		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-06	920	1,472.95 GB	\$8.99		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-07	878	1,469.56 GB	\$8.97		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-08	942	1,554.97 GB	\$9.49		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-09	915	1,571.16 GB	\$9.59		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-10	902	1,565.51 GB	\$9.56		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-11	932	1,549.35 GB	\$9.46		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-12	959	1,602.41 GB	\$9.78		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-13	843	843.43 GB	\$5.15		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-14	817	374.59 GB	\$2.29		dbt-runner@demo-warehouse.iam.gserviceaccount.com
2026-10-15	741	344.97 GB	\$2.11		dbt-runner@demo-warehouse.iam.gserviceaccount.com

Storage

10 table(s): 307.52 GB active + 0.00 GB long-term (logical), ~ \$6.15/month at the configured rates.

Compressed (physical) footprint: 86.11 GB active + 0.00 GB long-term + 6.15 GB time-travel. Billed on the physical model this would be ~ \$3.69/month.

Worth a look: physical storage billing looks ~ \$2.46/month cheaper for this data. The switch is made per dataset and also bills time-travel/fail-safe bytes, so verify per dataset before changing.

Largest tables by monthly storage cost

Name	Jobs	Bytes	Cost
dw_prod.fact_sales		217.00 GB	\$4.34
dw_prod.fact_events_raw		42.00 GB	\$0.84
dw_prod.fact_orders		28.00 GB	\$0.56
dw_archive.fact_sales_2024		9.00 GB	\$0.18
dw_prod.fact_legacy_sales		4.20 GB	\$0.08
dw_prod.fact_legacy_orders		2.80 GB	\$0.06
dw_prod.dim_customer		1.60 GB	\$0.03
dw_prod.dim_product		1.20 GB	\$0.02

Name	Jobs	Bytes	Cost
dw_prod_stg.stg_sales		1.10 GB	\$0.02
dw_prod_stg.stg_customers		0.62 GB	\$0.01

Partitioning and clustering

Which tables are set up to let BigQuery skip data, and which are not. Partitioning is what prunes a scan; clustering only reduces the work within what is already scanned. This section is layout, not activity - it changes when models are rebuilt rather than day to day.

Layout	Value
Tables with a layout recorded	10
Partitioned	5
Clustered	3
Bytes in partitioned tables	261.00 GB
Bytes in clustered tables	246.60 GB

Largest tables, with their layout

Table	Size	Partitioned by	Clustered by
dw_prod.fact_sales	217.00 GB	event_date	erp_id, legal_unit
dw_prod.fact_events_raw	42.00 GB	-	-
dw_prod.fact_orders	28.00 GB	event_date	erp_id
dw_archive.fact_sales_2024	9.00 GB	event_date	-
dw_prod.fact_legacy_sales	4.20 GB	event_date	-
dw_prod.fact_legacy_orders	2.80 GB	event_date	-
dw_prod.dim_customer	1.60 GB	-	legal_unit
dw_prod.dim_product	1.20 GB	-	-
dw_prod_stg.stg_sales	1.10 GB	-	-
dw_prod_stg.stg_customers	0.62 GB	-	-

Surprise-bill risk findings

9 finding(s): 1 HIGH · 5 MEDIUM · 3 LOW

[HIGH] Partitioned table queried without its partition filter

Top offender: dbt-runner@demo-warehouse.iam.gserviceaccount.com

Largest scan: 17.98 GB · 803 matching job(s)

803 queries read a partitioned table without filtering on its partition column, so BigQuery scanned (and billed) every partition instead of just the needed ones. This is the most common cause of surprise bills.

demo_job_005142 · 2026-09-18 22:47:00 · dbt-runner@demo-warehouse.iam.gserviceaccount.com · 17.98 GB

Table: dw_prod.fact_sales · partition column: **event_date**

```
select erp_id, legal_unit, sum(amount) as total from `demo-warehouse.dw_prod.fact_sales` group by erp_id, legal_unit
```

demo_job_005197 · 2026-09-21 01:32:00 · dbt-runner@demo-warehouse.iam.gserviceaccount.com · 17.73 GB

Table: dw_prod.fact_sales · partition column: **event_date**

```
select erp_id, legal_unit, sum(amount) as total from `demo-warehouse.dw_prod.fact_sales` group by erp_id, legal_unit
```

demo_job_004592 · 2026-10-04 11:21:00 · analyst.one@example.com · 17.53 GB

Table: dw_prod.fact_sales · partition column: **event_date**

```
select erp_id, legal_unit, sum(amount) as total from `demo-warehouse.dw_prod.fact_sales` group by erp_id, legal_unit
```

Recommended: Add a WHERE filter on the partition column (named per example below) so BigQuery prunes partitions. For critical tables, also set `require_partition_filter`.

[MEDIUM] SELECT * on a large scan

Top offender: dbt-runner@demo-warehouse.iam.gserviceaccount.com

Largest scan: 6.50 GB · 633 matching job(s)

633 SELECT * queries scanned a large amount of data. BigQuery bills on bytes referenced, so this is expensive even if the result is small. The largest offenders are listed below.

demo_job_002719 · 2026-10-04 20:17:00 · dbt-runner@demo-warehouse.iam.gserviceaccount.com · 6.50 GB

Tables: dw_prod.fact_sales

```
select * from `demo-warehouse.dw_prod.fact_sales` where erp_id is not null order by erp_id
```

demo_job_002746 · 2026-10-04 21:56:00 · analyst.two@example.com · 6.48 GB

Tables: dw_prod.fact_sales

```
select * from `demo-warehouse.dw_prod.fact_sales` where erp_id is not null order by erp_id
```

demo_job_003251 · 2026-10-14 11:21:00 · analyst.one@example.com · 6.48 GB

Tables: dw_prod.fact_sales

```
select * from `demo-warehouse.dw_prod.fact_sales` where erp_id is not null order by erp_id
```

Recommended: Select only the columns you need and add a filter on the partition column.

[MEDIUM] Heavy query without LIMIT

Top offender: dbt-runner@demo-warehouse.iam.gserviceaccount.com

Largest scan: 17.98 GB · 2889 matching job(s)

2889 queries that processed a large volume of data have no LIMIT, so there is nothing capping the result or the scan. The largest offenders are listed below.

demo_job_005142 · 2026-09-18 22:47:00 · dbt-runner@demo-warehouse.iam.gserviceaccount.com · 17.98 GB

Tables: dw_prod.fact_sales

```
select erp_id, legal_unit, sum(amount) as total from `demo-warehouse.dw_prod.fact_sales` group by erp_id, legal_unit
```

demo_job_005197 · 2026-09-21 01:32:00 · dbt-runner@demo-warehouse.iam.gserviceaccount.com · 17.73 GB

Tables: dw_prod.fact_sales

```
select erp_id, legal_unit, sum(amount) as total from `demo-warehouse.dw_prod.fact_sales` group by erp_id, legal_unit
```

demo_job_004592 · 2026-10-04 11:21:00 · analyst.one@example.com · 17.53 GB

Tables: dw_prod.fact_sales

```
select erp_id, legal_unit, sum(amount) as total from `demo-warehouse.dw_prod.fact_sales` group by erp_id, legal_unit
```

Recommended: Add LIMIT to exploratory queries.

[MEDIUM] Query joins tables across regions

Top offender: dbt-runner@demo-warehouse.iam.gserviceaccount.com

Largest scan: 0.80 GB · 2796 matching job(s)

2796 queries referenced tables stored in different regions. Cross-region queries move data between locations, bill at the pricier side, and are a recurring source of confusing invoices.

demo_job_006668 · 2026-10-09 02:33:00 · dbt-runner@demo-warehouse.iam.gserviceaccount.com · 0.80 GB

Tables: dw_prod.fact_sales, dw_archive.fact_sales_2024

Regions: eu, us-central1

```
select s.erp_id, a.amount from `demo-warehouse.dw_prod.fact_sales` as s join  
`demo-warehouse.dw_archive.fact_sales_2024` as a on s.erp_id = a.erp_id
```

demo_job_006493 · 2026-10-03 09:13:00 · dbt-runner@demo-warehouse.iam.gserviceaccount.com · 0.80 GB

Tables: dw_prod.fact_sales, dw_archive.fact_sales_2024

Regions: eu, us-central1

```
select s.erp_id, a.amount from `demo-warehouse.dw_prod.fact_sales` as s join  
`demo-warehouse.dw_archive.fact_sales_2024` as a on s.erp_id = a.erp_id
```

demo_job_006238 · 2026-10-14 19:12:00 · dbt-runner@demo-warehouse.iam.gserviceaccount.com · 0.80 GB

Tables: dw_prod.fact_sales, dw_archive.fact_sales_2024

Regions: eu, us-central1

```
select s.erp_id, a.amount from `demo-warehouse.dw_prod.fact_sales` as s join  
`demo-warehouse.dw_archive.fact_sales_2024` as a on s.erp_id = a.erp_id
```

Recommended: Co-locate the joined tables in one region, or materialise a copy of the smaller table in the other region.

[MEDIUM] Same query billed over and over

9 queries ran many times and billed 41381.32 GB in total. BigQuery's result cache only helps when the underlying data has not changed; these re-runs paid full price.

2638 runs · 0 from cache · 31,656.00 GB billed · dbt-runner@demo-warehouse.iam.gserviceaccount.com

```
with table_shards_stage as ( select info_tables.table_catalog as table_catalog, info_tables.table_schema as  
table_schema, coalesce(regexp_extract(info_tables.table_name, r'^'), info_tables.table_name)...
```

210 runs · 0 from cache · 2,621.97 GB billed · dbt-runner@demo-warehouse.iam.gserviceaccount.com

```
select event_type, count(*) as n from `demo-warehouse.dw_prod.fact_events_raw` group by event_type
```

633 runs · 0 from cache · 2,379.50 GB billed · analyst.one@example.com

```
select * from `demo-warehouse.dw_prod.fact_sales` where erp_id is not null order by erp_id
```

Recommended: Materialise the result into a small table on a schedule (or a materialised view) and point readers at that, instead of re-scanning the sources each run.

[MEDIUM] Large unpartitioned tables being queried

Largest table: 42.00 GB · 1 table(s)

1 table over 10 GB has no partitioning and was read in this window. Without a partition column there is nothing to prune, so every query pays for the entire table.

dw_prod.fact_events_raw · 42.00 GB stored · read by 210 queries

Recommended: Partition on the column queries filter by - usually a date. In dbt that is partition_by in the model config; the table rebuilds on the next full run. Clustering helps too, but only partitioning prunes the scan.

[LOW] Zombie table (storing money, not read)

dw_prod.fact_legacy_sales

This table stores data and costs money every month but was not referenced in the lookback window.

Recommended: Confirm it is still needed; if not, archive or delete it to stop the monthly storage cost.

[LOW] Zombie table (storing money, not read)

dw_prod.fact_legacy_orders

This table stores data and costs money every month but was not referenced in the lookback window.

Recommended: Confirm it is still needed; if not, archive or delete it to stop the monthly storage cost.

[LOW] Spend on jobs that failed anyway

Largest scan: 4.57 GB · 4 matching job(s)

4 jobs failed after BigQuery had already scanned 4.6 GB. A job that errors part-way still bills for what it read. Top reasons: invalidQuery (2), resourcesExceeded (1), stopped (1).

demo_job_009320 · 2026-10-04 17:28:00 · dbt-runner@demo-warehouse.iam.gserviceaccount.com · 1.68 GB

Tables: dw_prod.fact_sales

```
select cast(erp_id as int64) as erp_id, count(*) as n from `demo-warehouse.dw_prod.fact_sales` group by 1
```

demo_job_009317 · 2026-10-05 03:26:00 · dbt-runner@demo-warehouse.iam.gserviceaccount.com · 1.07 GB

Tables: dw_prod.fact_sales

```
select cast(erp_id as int64) as erp_id, count(*) as n from `demo-warehouse.dw_prod.fact_sales` group by 1
```

demo_job_009318 · 2026-09-25 18:23:00 · dbt-runner@demo-warehouse.iam.gserviceaccount.com · 1.03 GB

Tables: dw_prod.fact_sales

```
select cast(erp_id as int64) as erp_id, count(*) as n from `demo-warehouse.dw_prod.fact_sales` group by 1
```

Recommended: Fix the recurring failures first - they are pure waste. A maximum_bytes_billed ceiling on the offending workloads caps the damage while you do.