

BENCHMARK AND COST MODEL

BY THE NUMBERS

SQL and graph benchmark results for TETRA against PostgreSQL, DuckDB and the leading open-source graph database, with a three-year cost model.

TETRA is a data storage and retrieval engine. It stores data as a matrix and has two interfaces: SQL over the PostgreSQL wire protocol, and a labeled property graph over openCypher and Bolt. This report benchmarks both. The SQL benchmark is a manual of 251 scenarios, run at four table sizes on TETRA, PostgreSQL and DuckDB. The graph benchmark is the Linked Data Benchmark Council's Social Network Benchmark, run at five scale factors from 17 million to 1.78 billion edges and compared with the leading open-source graph database at the three smallest. The cost model estimates three years of infrastructure cost from these measurements.

4,000,000 rows · 1,775,513,741 edges · 251 SQL scenarios
21 LDBC queries · single node · 2026-10-12

SECTION 01

EXECUTIVE SUMMARY

TETRA was 82x faster than PostgreSQL and 9.5x faster than DuckDB on the SQL manual at four million rows, and 44x faster than the graph leader on LDBC at SF10. It used 8.8x less disk than PostgreSQL and 4.4x less than the graph leader.

The system

TETRA stores data as a matrix and runs a query as operations on that matrix. Tables and graphs are both stored this way, so one store serves a SQL interface and a graph interface. TETRA used no indexes, in either interface, in any measurement in this report.

SQL results

The SQL manual has 251 scenarios that all three engines answered at four million rows. They include casts, joins, grouping, window functions, set operators, correlated subqueries, DDL and writes. PostgreSQL ran with btree indexes and a fresh ANALYZE. At four million rows the sum of the median times was 373 ms for TETRA, 30,558 ms for PostgreSQL and 3,526 ms for DuckDB. TETRA answered every scenario at every table size.

Graph results

TETRA answered the 21 LDBC Social Network Benchmark Interactive queries at five scale factors, SF1 to SF100. The graph leader was measured at SF1, SF3 and SF10. On the sum of the 21 median times TETRA was 22.96x faster at SF1 and 44.21x faster at SF10. Both engines returned the same number of rows for all 21 queries at all three sizes. TETRA's peak memory at SF10 was 2.0x lower.

Scaling

Query time grew more slowly than the data in both benchmarks. The log-log gradient of total query time was 0.43 against table size and 0.73 against edge count. A gradient of 1.00 means time grows in proportion to the data. On disk, the four-million-row table was 8.8x smaller than in PostgreSQL, and the graph was 4.4x smaller per edge than in the graph leader.

Cost

The cost model takes a dataset that grows by 500 GB a month for three years and prices disk, backups, cache memory and compute. The three-year total is $7.4k for TETRA and $65k for PostgreSQL on the relational workload, and $14k for TETRA and $63k for the graph leader on the graph workload.

These results are from one run on one host, on a single node without replication. The reference engines were measured once and stored, and were not re-run alongside TETRA. The Limitations section lists every qualification.

SECTION 02

SUMMARY

Headline figures and the summary table.

82x

Faster than PostgreSQL

373 ms against 30,558 ms, 4M rows

44x

Faster than the graph leader

2.57 s against 113.7 s at SF10

8.8x

Smaller on disk

19.5 bytes per row against 171.1

$58k

Three-year saving

at 500 GB a month against PostgreSQL

	TETRA	POSTGRESQL	DUCKDB	GRAPH LEADER
SQL manual at 4M rows	**373.1 ms**	30,558.2 ms	3,526.5 ms	n/a
Scenarios TETRA answered no slower		201 of 251	226 of 251	n/a
Scenarios declined	**0 at every size**	3 at every size	22 at every size	n/a
Growth against table size	**0.43**	0.54	0.44	n/a
LDBC 21 queries at SF10	**2,570.8 ms**	n/a	n/a	113,666.2 ms
LDBC CPU per execution at SF10	**10,243 ms**	n/a	n/a	121,665 ms
LDBC queries answered	**21 of 21 at every size**	n/a	n/a	21 of 21 to SF10
Peak memory at SF10	**14,143 MB**	n/a	n/a	28,420 MB
Bytes on disk per row	**19.5**	171.1	none, in memory	n/a
Bytes on disk per edge	**23.6**	n/a	n/a	104.7
Indexes	**none**	4 btree, ANALYZE	none	17
Three years at 500 GB a month	**$7.4k SQL, $14k graph**	$65k	$1.5M	$63k

SQL totals are sums of medians over the scenarios all three SQL engines answered at that size. DuckDB ran in memory with no store on disk, so its cost is the cost of memory. The graph leader was not measured above SF10.

SECTION 03

WHAT WAS MEASURED

Two workloads and three reference engines, on one host.

The SQL manual

The manual is a catalogue of 251 SQL scenarios. Each scenario is a statement and the answer it must return. The scenarios include casts, comparisons, joins, grouping and folds, window functions, set operators, correlated subqueries, DDL and writes. They run against one generated table of N rows and two small companion tables, at four sizes from 50,000 rows to 4,000,000.

Each engine receives the same statement text. TETRA and PostgreSQL answer over a wire protocol. DuckDB runs the same text in-process, so its times do not include the protocol or the serialisation of results. Every figure is a median over the recorded executions, which follow a set of warm-up executions.

The graph benchmark

The graph workload is the LDBC Social Network Benchmark, Interactive workload v1, published by the Linked Data Benchmark Council at ldbcouncil.org. LDBC is an independent body, and its benchmarks are the standard reference for graph database performance. The 21 queries are the LDBC reference implementations, unmodified.

The LDBC generator produced the dataset at scale factors 1, 3, 10, 30 and 100. The smallest has 17,256,038 edges and the largest has 1,775,513,741. Query parameters are pinned from a stored parameter set. At SF1, SF3 and SF10 the parameters are the sets the graph leader answered, replayed against TETRA, so both engines receive the same queries.

The three references

All three reference engines were measured once on this host, and the results were stored. Each TETRA run is compared with the stored results. The reference figures are therefore the same in every report, and only the TETRA figures change.

- **PostgreSQL** with btree indexes on the four filtered columns and a fresh ANALYZE, over the v3 wire protocol.
- **DuckDB** running the same statements in-process, with no wire protocol and no store on disk.
- **The graph industry leader**, community release, with the constraints and indexes of the LDBC reference implementation, measured at SF1, SF3 and SF10.

SECTION 04

HANDICAPS AND CONTROLS

Differences in test conditions between the engines, and which engine each one favours.

Statements and parameters: the same for every engine

Every engine receives the same statement text and the same parameters. Row counts are compared per query. On the graph benchmark all 21 queries return the same number of rows as the leader at SF1, SF3 and SF10.

Indexes: favours the references

PostgreSQL has btree indexes on every column the manual filters on, built and analysed before the run. The graph leader has the nine constraints and eight indexes of the LDBC reference implementation, all ONLINE. TETRA has no indexes in either workload.

Wire protocol: favours DuckDB

TETRA and PostgreSQL answer over a wire protocol, so their times include encoding and sending the result. DuckDB is called in-process, and its times do not. Part of DuckDB's lead in the scenarios where it is faster comes from this.

Durability and encryption: favours the references

PostgreSQL ran with `synchronous_commit=off`. DuckDB ran in memory and wrote nothing to disk. TETRA wrote its store to disk and encrypted it. TETRA always encrypts its store.

Not controlled

The reference engines were measured once and stored. They did not run at the same time as TETRA or under the same background load. The host is also used for other work. Repeated runs of the SQL manual with no code change vary by up to about 30 ms in the four-million-row total.

> THE 30 SECOND LIMIT
>
> A scenario that runs longer than 30 seconds has no median and is excluded from the comparison at that size. PostgreSQL exceeded the limit on one scenario at one million and four million rows. Totals in this report cover only the scenarios that all three engines answered.

SECTION 05

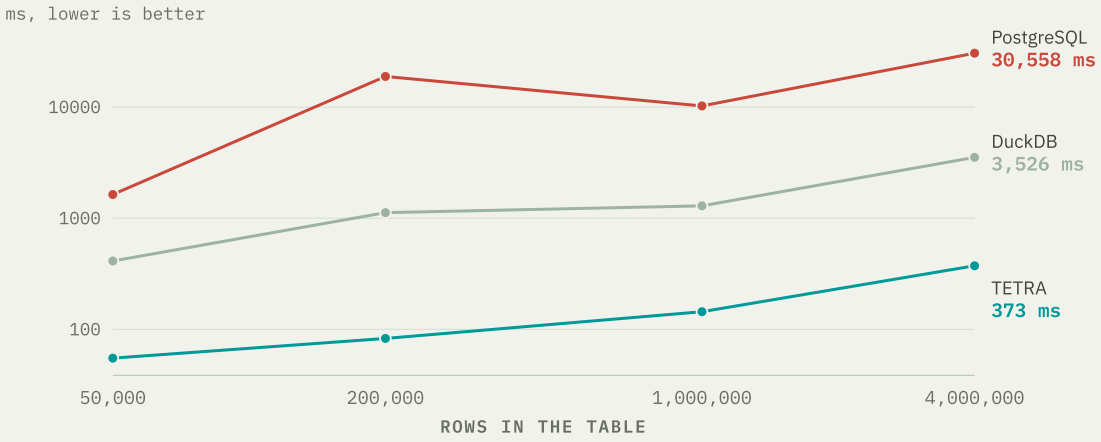
SQL RESULTS

Sum of median times for the 251-scenario manual at four table sizes, on three engines.

ROWS	COMPARED	TETRA	POSTGRESQL	DUCKDB	VS POSTGRESQL	VS DUCKDB
50,000	252	**55.1 ms**	1,633.5 ms	412.4 ms	**29.7x**	**7.5x**
200,000	252	**82.8 ms**	18,876.0 ms	1,123.3 ms	**227.9x**	**13.6x**
1,000,000	251	**144.3 ms**	10,244.9 ms	1,294.3 ms	**71.0x**	**9.0x**
4,000,000	251	**373.1 ms**	30,558.2 ms	3,526.5 ms	**81.9x**	**9.5x**

Each figure is the sum of medians over the scenarios all three engines answered at that size. The number compared falls from 252 to 251 at one million rows because one scenario, a cross-variable inequality, exceeds the 30 second limit on PostgreSQL from that size up. At 200,000 rows that scenario is 16,270 ms of PostgreSQL's 18,876 ms total.

SQL manual total by table size

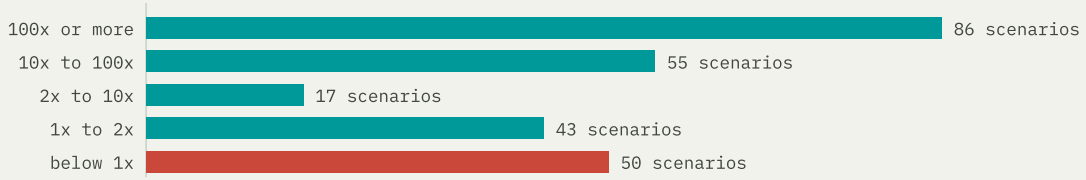


Both axes are logarithmic, and the horizontal axis is spaced by table size. TETRA has the lowest total at every size. PostgreSQL's total is higher at 200,000 rows than at 1,000,000 because the cross-variable inequality scenario is counted at 200,000 rows and excluded at 1,000,000, where it exceeded the limit.

Margin by scenario

Of the 251 scenarios all three engines answered at four million rows, TETRA was at least as fast as PostgreSQL on 201 and at least as fast as DuckDB on 226. The two charts count the scenarios in each band of margin.

Scenarios by margin over PostgreSQL



SCENARIOS AT FOUR MILLION ROWS, BY MARGIN BAND

The margin is PostgreSQL's median divided by TETRA's, per scenario, at four million rows. Each bar is the number of scenarios in that band. TETRA is at least as fast in the four teal bands and slower in the red band.

Scenarios by margin over DuckDB

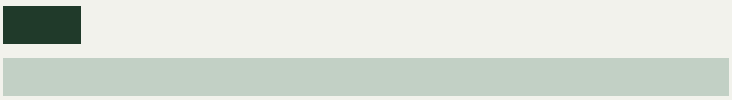


SCENARIOS AT FOUR MILLION ROWS, BY MARGIN BAND

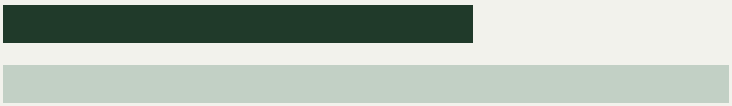
The same calculation against DuckDB, which runs in-process with no wire protocol. TETRA is at least as fast on 226 of 251.

SQL MANUAL TOTAL AT 4M ROWS, AGAINST POSTGRESQL — 82x faster

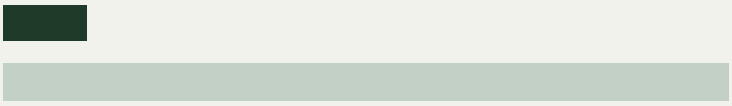
SQL MANUAL TOTAL AT 4M ROWS, AGAINST DUCKDB — 9.5x faster



BULK LOAD OF FOUR MILLION ROWS — 1.5x faster



THE TABLE ON DISK AT FOUR MILLION ROWS — 8.8x smaller



Growth with table size

The growth gradient is the slope of log total time against log row count. A gradient of 1.00 means the total grows in proportion to the rows. TETRA's gradient is 0.43, PostgreSQL's is 0.54 and DuckDB's is 0.44.

All three are below 1.00. PostgreSQL uses indexes and DuckDB uses a columnar layout. TETRA uses neither. TETRA's margin over DuckDB is between 7.5x and 13.6x at the four sizes.

TETRA's twenty slowest scenarios at four million rows

The scenarios are sorted by TETRA's median, slowest first.

SCENARIO	TETRA	POSTGRESQL	DUCKDB	
EXCEPT ALL	**45.60**	353.07	202.25	**7.7x**
INTERSECT ALL	**24.54**	319.93	100.20	**13.0x**
CREATE INDEX	**21.23**	1,158.59	190.45	**54.6x**
LIKE prefix	**17.03**	129.88	23.58	**7.6x**
grouped DISTINCT sum	**13.00**	n/a	n/a	
grouped DISTINCT under HAVING	**8.23**	n/a	n/a	
grouped DISTINCT over a join	**8.19**	n/a	n/a	
JOIN USING	**7.50**	179.51	12.30	**23.9x**
scaled sum	**6.96**	125.68	2.19	**18.0x**
UNION nulls	**6.82**	394.54	10.04	**57.9x**
UPDATE FROM folding	**6.78**	1,047.37	5.78	**154.5x**
a cross-var inequality is not a join	**6.12**	bounded	14,720.25	
DELETE range	**5.81**	22.75	77.00	**3.9x**
three way	**5.17**	275.01	7.70	**53.2x**
HAVING over a mean	**4.69**	174.58	2.78	**37.2x**
constant key over rows	**4.69**	67.67	1.18	**14.4x**
INSERT … SELECT	**4.39**	1.48	1.89	**0.3x**
a text-keyed grouped fold over a join	**4.16**	180.85	3.90	**43.4x**
ordered by a fold the answer hides	**3.95**	172.06	2.62	**43.5x**
oid to name	**3.92**	0.12	n/a	**0.0x**

Medians in milliseconds at four million rows. `n/a` means the reference declined the scenario. `bounded` means it did not finish within the 30 second limit. Neither has a ratio, and neither is included in any total in this report.

EXCEPT ALL, INTERSECT ALL and LIKE prefix return between 333,000 and 666,000 rows. Engine CPU for each is about one millisecond. The rest of the time is spent encoding the rows for the wire and parsing them in the client. The in-process reference does not pay this cost.

SECTION 06

SCENARIOS WHERE DUCKDB IS FASTER

DuckDB is faster than TETRA on 25 of the 251 scenarios at four million rows.

SCENARIO	TETRA	DUCKDB	POSTGRESQL	TETRA GROWTH
scaled sum	6.96	**2.19**	125.68	0.63
constant key over rows	4.69	**1.18**	67.67	0.63
ALTER DROP COLUMN	3.19	**0.41**	0.37	0.13
outer JOIN a VALUES holding an expression	3.75	**1.02**	0.14	0.42
INSERT … SELECT	4.39	**1.89**	1.48	0.49
HAVING over a mean	4.69	**2.78**	174.58	0.43
outer JOIN a set-returning call	3.08	**1.21**	0.23	0.40
ALTER RENAME COLUMN	1.88	**0.24**	0.29	0.24
ordered by a fold the answer hides	3.95	**2.62**	172.06	0.51
a rounded mean over a join	3.89	**2.71**	174.07	0.50
UPDATE range	2.04	**0.89**	71.59	0.21
scalar correlated	3.42	**2.41**	282.74	0.33
UPDATE FROM folding	6.78	**5.78**	1,047.37	0.25
ordered by a fold with NULLS LAST	3.33	**2.36**	61.13	0.55
ordered by a fold over a join	3.37	**2.43**	61.52	0.52
folding JOIN cursor fold	3.82	**3.19**	550.48	0.53
decimal times integer	1.94	**1.31**	104.36	0.61
GROUP BY sum	2.36	**1.78**	174.82	0.49
sum(float)	0.97	**0.58**	50.99	0.59
a text-keyed grouped fold over a join	4.16	**3.90**	180.85	0.44
read after truncate	1.31	**1.14**	0.44	0.49
rollback took	0.48	**0.34**	0.12	0.30
read after rename	0.74	**0.67**	56.63	0.56
LEFT JOIN folding	3.16	**3.12**	82.05	0.59
UPDATE WHERE IN subquery	1.45	**1.43**	0.25	0.40

Medians in milliseconds, largest gap first. The last column is TETRA's growth gradient for that scenario.

The scenarios are of three kinds: scan-bound folds, where a columnar engine reads one column and TETRA reads the matrix; DDL, at 1.9 to 3.2 ms against 0.2 to 0.4 ms; and small-constant joins, where most of the measured time is overhead, including the wire protocol that DuckDB does not use.

SECTION 07

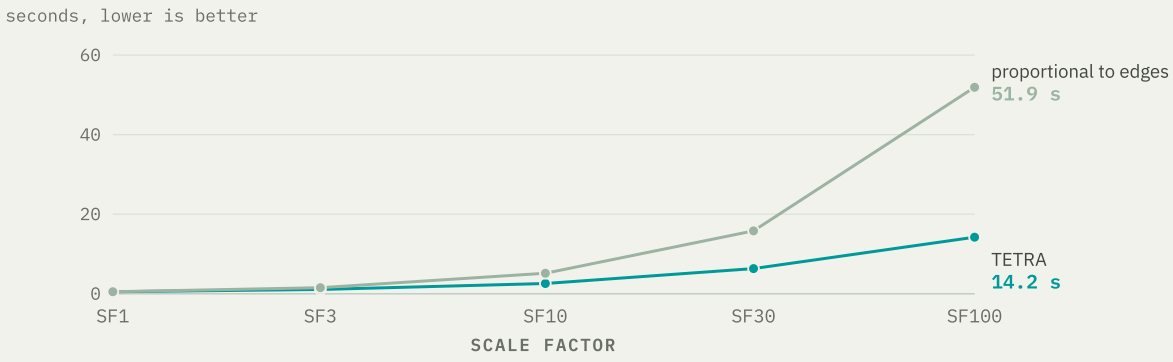
GRAPH RESULTS

The 21 LDBC Interactive queries at five scale factors, from 17 million to 1.78 billion edges.

	SF1	SF3	SF10	SF30	SF100
Nodes	3,181,724	9,281,922	29,987,835	88,789,833	282,637,871
Edges	17,256,038	52,695,734	176,623,433	540,915,383	1,775,513,741
On disk	0.34 GiB	1.05 GiB	3.62 GiB	11.58 GiB	38.95 GiB
Bytes per edge	20.9	21.4	22.0	23.0	23.6
Cold load	1.1 s	2.9 s	9.5 s	2.1 s	6.4 s
Resident at ready	750 MB	2,154 MB	7,052 MB	3,962 MB	12,559 MB
Resident at peak	1,664 MB	4,665 MB	14,143 MB	23,577 MB	66,743 MB
Sum of 21 medians	**504.5 ms**	**1,083.9 ms**	**2,570.8 ms**	**6,324.4 ms**	**14,209.7 ms**
CPU per execution	1,291 ms	3,505 ms	10,243 ms	33,557 ms	107,350 ms
Answered	**21 of 21**	**21 of 21**	**21 of 21**	**21 of 21**	**21 of 21**

Bytes per edge rises from 20.9 to 23.6 as the edge count grows 103-fold. The SF100 store holds 1,775,513,741 edges in 38.95 GiB, encrypted, with no index. Cold load at SF30 and SF100 is shorter than at SF10 because those two stores were already in the page cache. The SF10 figure is the cold reading.

Sum of the 21 medians by scale factor

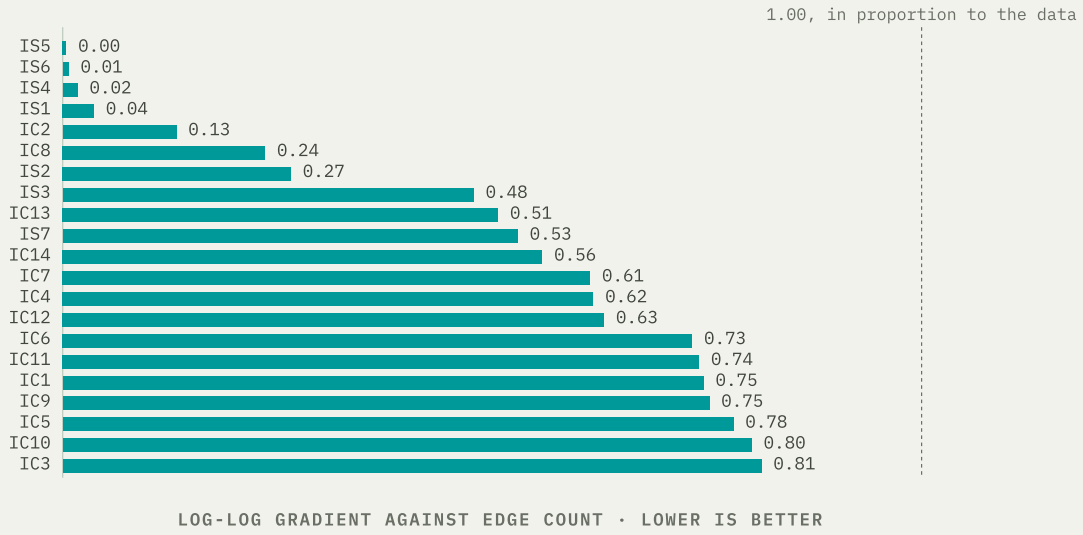


The vertical axis is linear, and the horizontal axis is spaced by edge count. The grey line is the SF1 total scaled in proportion to the edge count, which reaches 51.9 seconds at SF100. The measured total at SF100 is 14.2 seconds.

Growth by query

The chart shows the growth gradient of each of the 21 queries across the five scale factors, a 103-fold range in edge count. All 21 gradients are below 1.00, and 4 are at or below 0.10. A query with a gradient near zero takes about the same time at SF100 as at SF1.

Growth gradient by query



The gradient of log median against log edge count for each query, sorted.

At SF100 the seven Interactive Short queries take 31.4 ms in total, out of 14.21 seconds for all 21. Their medians at SF100 are between 0.14 ms and 19.12 ms. The Interactive Complex queries take the rest, and IC10 is 40 % of the total.

SECTION 08

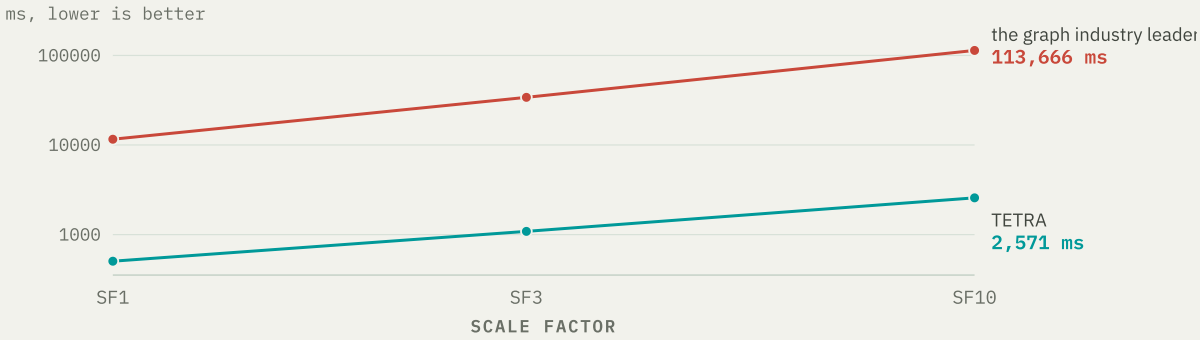
COMPARISON WITH THE GRAPH LEADER

The 21 queries with the same parameter sets at SF1, SF3 and SF10, the three sizes at which the leader was measured.

	SF1	SF3	SF10
Edges	17,256,038	52,695,734	176,623,433
TETRA, sum of 21 medians	**504.5 ms**	**1,083.9 ms**	**2,570.8 ms**
Leader, sum of 21 medians	11,582.3 ms	33,989.8 ms	113,666.2 ms
Latency margin	**22.96x**	**31.36x**	**44.21x**
TETRA CPU per execution	**1,291 ms**	**3,505 ms**	**10,243 ms**
Leader CPU per execution	12,326 ms	40,725 ms	121,665 ms
CPU margin	**9.5x**	**11.6x**	**11.9x**
TETRA peak memory	**1,664 MB**	**4,665 MB**	**14,143 MB**
Leader peak memory	3,783 MB	12,486 MB	28,420 MB
Queries TETRA answered no slower	**20 of 21**	**20 of 21**	**20 of 21**
Row parity	**21 of 21**	**21 of 21**	**21 of 21**

The parameter sets are the ones the leader answered, replayed against TETRA, so both engines receive the same queries. Row counts are compared per query. Both engines return the same number of rows for all 21 queries at all three sizes.

Sum of the 21 medians, both engines

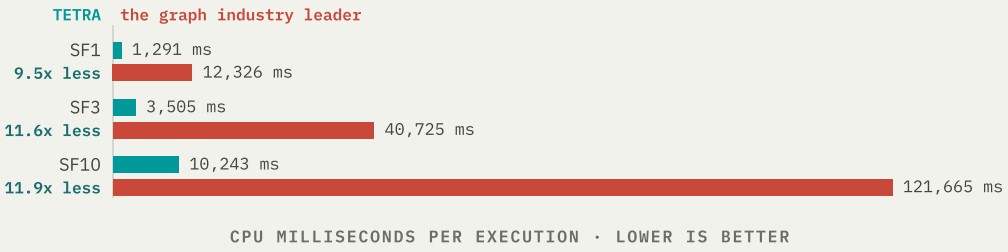


The vertical axis is logarithmic, and the horizontal axis is spaced by edge count. The margin is 22.96x at SF1, 31.36x at SF3 and 44.21x at SF10.

CPU and memory

At SF10 TETRA used 11.9x less CPU per execution than the leader and 2.0x less memory at its peak, and was 44.2x faster.

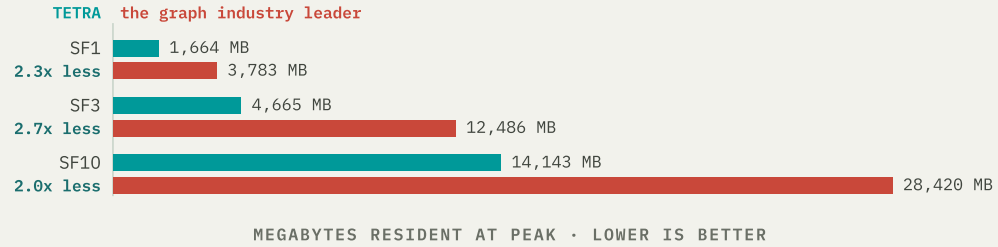
CPU per execution of the 21 queries



CPU MILLISECONDS PER EXECUTION · LOWER IS BETTER

Server-process CPU time for one pass of the 21 queries, as reported by the operating system.

Peak resident memory



MEGABYTES RESIDENT AT PEAK · LOWER IS BETTER

The largest resident set each process reached during the run. Both engines stay near this level for the whole run.

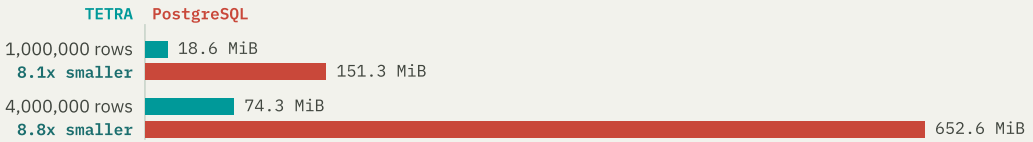
SECTION 09

STORAGE

The on-disk size of the same data in each engine. The cost model uses these figures.

Disk footprint sets the volume cost, the size of each snapshot, the time a restore takes and the share of the dataset a cache of a given size can hold. TETRA wrote less to disk than the reference engine on both workloads, with no index and with the store encrypted.

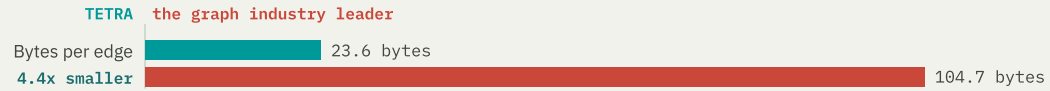
Table size on disk



MEBIBYTES ON DISK · LOWER IS BETTER

Both sizes are in mebibytes on one scale. PostgreSQL's figure is `pg_total_relation_size` over the three relations, which includes heap, indexes and TOAST. TETRA's figure is the size of its store file. The ratio is 8.1x at one million rows and 8.8x at four million.

Graph size on disk, per edge



BYTES ON DISK PER EDGE · LOWER IS BETTER

TETRA's figure is from this run's SF100 store, which has the highest bytes per edge of the five. The leader's figure is from the 2026-09-03 head to head, same benchmark, same class of host. This run did not measure the leader's store. That measurement was `1.683 GiB (+ 808 MiB transaction logs)`. The transaction logs are excluded here and from the cost model. Including them would raise the leader's figure by about half.

SECTION 10

THE COST MODEL

Estimated infrastructure cost over three years for a dataset that grows by 500 GB a month.

The model estimates the monthly bill for 36 months as the dataset grows. Its inputs are the measurements in this report and four list prices.

The dataset grows by 500 GB a month, measured as the incumbent engine's on-disk footprint. Each other engine stores the same data at its own measured footprint. There are two models because the two workloads have different incumbents. The relational model is anchored on PostgreSQL's footprint and the graph model on the graph leader's.

WHAT THE SCENARIO IS	RELATIONAL	GRAPH
Growth a month, incumbent's disk	500 GB	500 GB
Which is, per month	2.92 billion rows	4.78 billion edges
After 36 months	105 billion rows	172 billion edges
Incumbent on disk at 36 months	18.0 TB	18.0 TB
TETRA on disk at 36 months	**2.05 TB**	**4.05 TB**
Measured footprint ratio	**0.1138**	**0.2250**

Rows and edges per month are 500 GB divided by the incumbent's measured bytes per row and bytes per edge, from the Storage section.

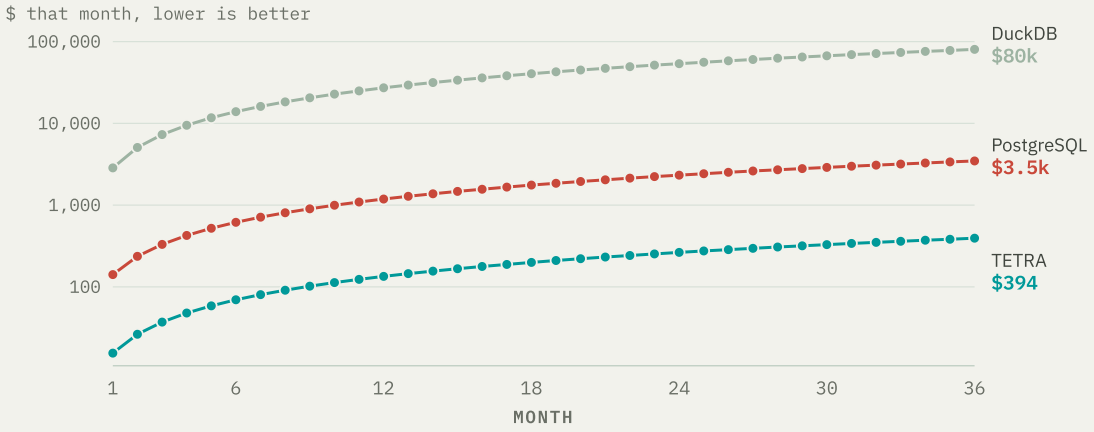
Prices and assumptions

ITEM	VALUE	BASIS
Provisioned SSD	$0.08 per GB-month	block storage list rate
Backup	$0.05 per GB-month	snapshots of the same volume
Memory	$3.00 per GB-month	taken out of instance pricing
Compute	$25 per core-month	one vCPU
Cache RAM	2 % of the engine's own data	swept from 0 % to 5 % in the sensitivity chart
Workload rate	the measured workload once a minute	the same rate on every engine
Copies	one	a replica multiplies both sides equally

The prices set the scale of the charts. They have little effect on the ratio between two engines, because both engines are priced for the same four items.

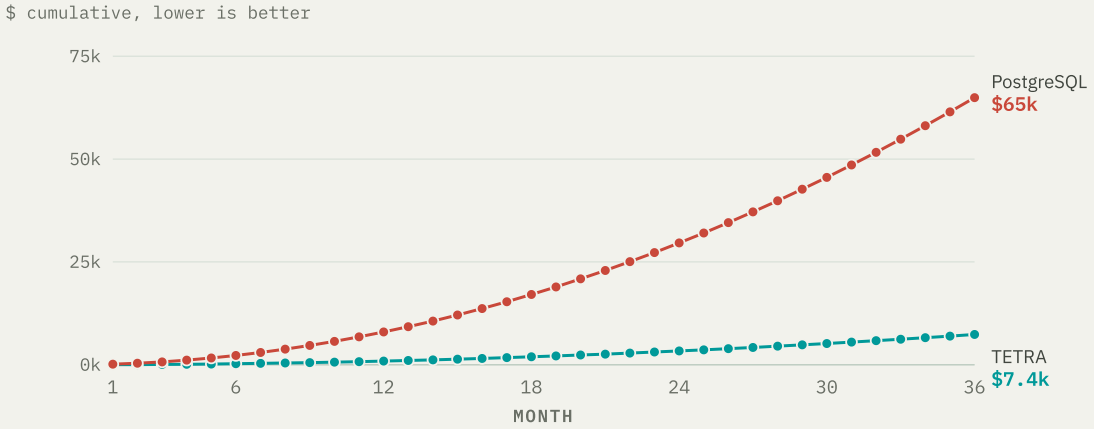
Three-year cost, relational workload

Monthly cost



The vertical axis is logarithmic because the three lines span three orders of magnitude. Stored data grows by the same amount every month, so each bill rises linearly: by about $95 a month for PostgreSQL and $11 a month for TETRA.

Cumulative cost over three years

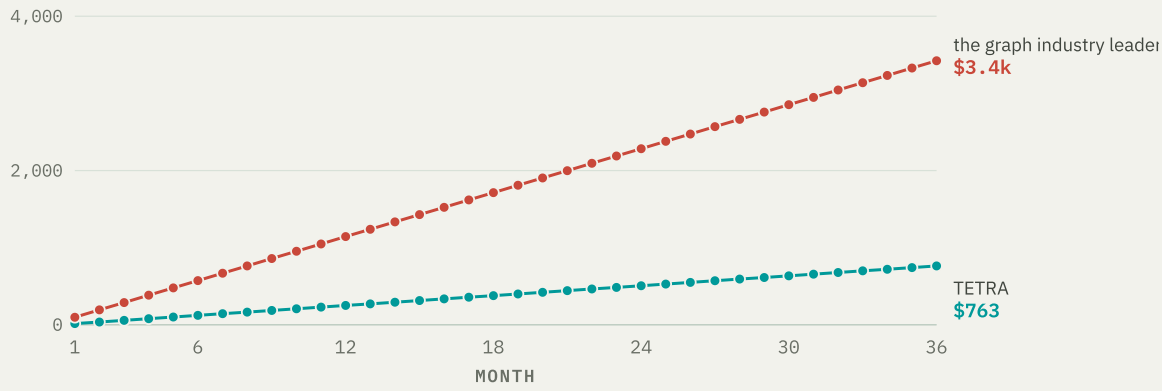


The vertical axis is linear. The three-year difference between TETRA and PostgreSQL is $58k, of which $42k accrues in months 19 to 36. DuckDB is not shown because its total of $1.5M would compress the other two lines against the axis.

Three-year cost, graph workload

Monthly cost

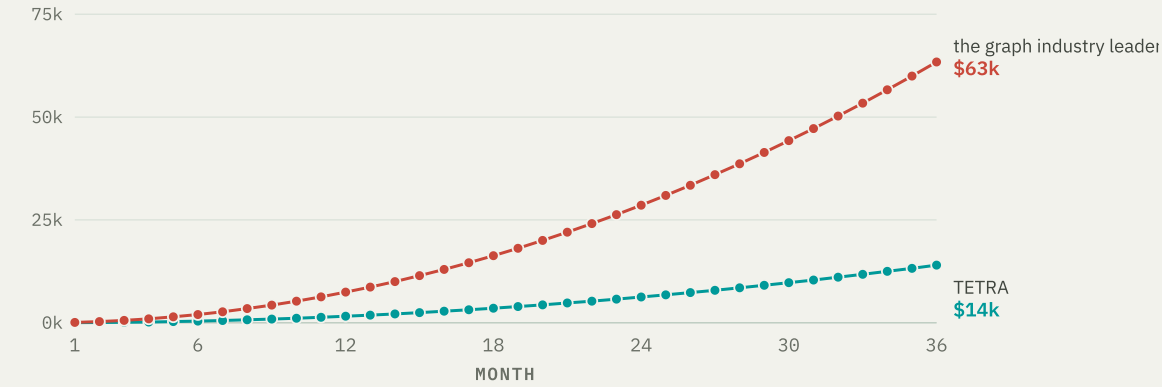
$ that month, lower is better



The vertical axis is linear. In month 36 the leader's bill is $3,423 and TETRA's is $763 for the same data, which TETRA stores in 4.4x less space.

Cumulative cost over three years

$ cumulative, lower is better



The vertical axis is linear. The three-year difference is $49k, a factor of 4.5. The factor is smaller than on the relational workload because the footprint ratio is 4.4x, against 8.8x for the table. The leader's transaction logs are excluded.

Cost by line item

THREE-YEAR TOTALS, RELATIONAL	TETRA	POSTGRESQL
Disk	**$3.0k**	$26k
Backup	**$1.8k**	$16k
Memory	**$2.2k**	$19k
Compute	**$364**	$3.4k
Three-year total	**$7.4k**	$65k

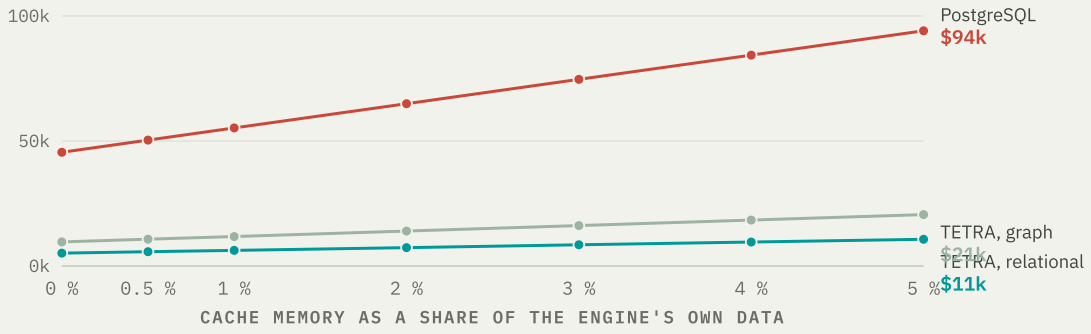
THREE-YEAR TOTALS, GRAPH	TETRA	GRAPH LEADER
Disk	**$5.8k**	$26k
Backup	**$3.6k**	$16k
Memory	**$4.4k**	$19k
Compute	**$154**	$1.8k
Three-year total	**$14k**	$63k

Disk and backup are 65 % of PostgreSQL's total and 66 % of the graph leader's. Compute is under 6 % of each of the four totals.

Sensitivity to cache size

Three-year cost by cache size

$ over three years, lower is better



Cache memory is the least certain input, so the chart varies it from 0 % to 5 % of each engine's own stored data. TETRA costs less than its incumbent at every setting in both models. The graph leader's curve is not shown because it is within 4 % of PostgreSQL's at every setting. At 5 % the leader's total is $93k and PostgreSQL's is $94k.

SECTION 11

DUCKDB IN THE COST MODEL

DuckDB's three-year total of $1.5M is the cost of holding the dataset in memory, which is how DuckDB was run in this benchmark.

The harness calls DuckDB in-process with `duckdb.connect(":memory:")`. DuckDB has no store on disk at any size in this benchmark, so the model has no disk to price for it. Its footprint is resident memory. In the price list memory costs about 38 times as much per gigabyte as block storage.

DuckDB's measured resident set grows by 252 bytes per row, above a fixed 449 MiB. PostgreSQL uses 171.1 bytes per row on disk. At the model's 18 TB of PostgreSQL-equivalent data, DuckDB needs about 26.5 TB of memory.

AT FOUR MILLION ROWS	TETRA	POSTGRESQL	DUCKDB
On disk	**74.3 MiB**	652.6 MiB	none
Peak resident	3,026 MiB	4,094 MiB	1,398 MiB
Resident growth per row	n/a	n/a	252 bytes
Suite CPU	**24.3 s**	225.4 s	4,203 s

> **NOT MEASURED**
>
> DuckDB also has a persistent file format. A DuckDB that writes to disk would have a storage figure and a different cost line. This benchmark did not run DuckDB that way, so this report does not model it. The cost line shown is for the in-memory configuration that was measured.

SECTION 12

WHERE TETRA DOES NOT WIN

Results where a reference engine is faster, and known weaknesses.

- **25 of 251 SQL scenarios are faster on DuckDB** at four million rows, and 50 are faster on PostgreSQL. The DuckDB scenarios are listed in the section Scenarios where DuckDB is faster.
- **The graph leader is faster on IC7 at every size it ran.** TETRA takes 37.75 ms against the leader's 16.49 at SF1, and 59.16 against 22.48 at SF10. IC7 is the only one of the 21 queries on which the leader is faster, at any of the three sizes.
- **DDL takes TETRA 1.9 to 3.2 ms**, against 0.2 to 0.4 ms for DuckDB. A workload that changes schema in a loop would be affected.
- **EXCEPT ALL, INTERSECT ALL and LIKE prefix are dominated by the wire protocol.** Each returns hundreds of thousands of rows and uses about one millisecond of engine CPU. The rest of the time is encoding and client parsing. They are three of TETRA's four slowest scenarios at four million rows.
- **The graph engine's snapshot label lanes are full width.** Fixing this needs a windowed lane type across the graph engine. The cost is about 0.1 ms per schema change, and it was left in place deliberately.
- **IC2 does not grow smoothly.** Its gradient across the five sizes is 0.13, and its median at SF30 is lower than at SF10. This is not yet understood, so the IC2 figures should not be relied on.

> RUN-TO-RUN NOISE
>
> Repeated runs of the SQL manual with no code change vary by up to about 30 ms in the four-million-row total. A difference of that size between two totals is not a result. The margins this report draws conclusions from are one to three orders of magnitude larger.

SECTION 13

LIMITATIONS

Known qualifications to the figures in this report.

- **The references were not re-run.** PostgreSQL, DuckDB and the graph leader were measured once on this host and stored. They did not run at the same time as TETRA or under the same background load.
- **One host, one node, no replication.** All engines ran on a single machine in their community editions, with no cluster and no failover. Clients connected over the loopback interface.
- **The graph benchmark is read-only.** LDBC Interactive queries only read. Writes through the graph interface are not measured here.
- **The graph leader was measured only up to SF10.** SF30 and SF100 are TETRA alone and have no comparison.
- **The leader's on-disk footprint comes from another run.** This benchmark did not measure it. The figure is from the 2026-09-03 head to head, same benchmark, same class of host. TETRA's footprint in that run was 12.9 bytes per edge, against 23.6 in this one. The cost model uses the figure from this run, which is higher and less favourable to TETRA.
- **DuckDB ran in memory.** It has no store on disk in this benchmark, so it is not in the storage comparison, and its cost line prices memory.
- **Cold load at SF30 and SF100 is not cold.** Those two stores were already in the page cache when they were opened. The cold figure is SF10's 9.5 seconds.
- **The cost model is an estimate.** Its four prices are list rates. The cache size is an assumption and was not measured. Compute is held at the top rung's measured figure. The model is not a quotation.
- **Latencies are medians.** Every latency in this report is a p50 over the recorded executions. Tail latency under concurrency is not measured here. An earlier report covers it.

SECTION 14

RIG AND REPRODUCTION

Host, software versions and harness configuration.

```
commit      d9189c84
run         2026-10-12T17:05:37Z
host        rescue, AMD EPYC 7502P, 32c/64t, Zen 2
harness     ops/epyc/bench/bench.sh
binaries    ldbc 186efef481d2d7a7
            sqlversus 8da9cd481a54bb5b
            tetra aac9b50f9bf7f163
TETRA       no index and no constraint in either interface, in every arm
            store encrypted at rest, which is its only mode
SQL manual  251 scenarios compared, at 50,000, 200,000, 1,000,000, 4,000,000 rows
            median over the recorded executions behind warm ones
            30 s bound: past it a scenario has no median and leaves the set
postgres    PostgreSQL 15.18, btree on age, score, name and dept, ANALYZE, synchronous_commit=off, over
the v3 wire
duckdb      DuckDB 1.5.5, the identical statement text replayed in-process, no wire, no store on disk
LDBC        SNB Interactive v1 reference queries, unmodified
            SF1 3 10 30 100: up to 282,637,871 nodes
            and 1,775,513,741 edges, 38.95 GiB
            parameters pinned from a stored set; SF1 3 10 use the sets
            the leader answered, replayed against TETRA
graph ref   current community release 5.26, OpenJDK 17, page cache and heap sized per scale factor as
the rig section lists
            LDBC reference implementation's 9 constraints + 8 indexes
references  measured once on this host and stored as
            ops/epyc/bench/reference/, not re-run
raw         results/bench/d9189c84/{results.json,report.md}
            schema tetra-bench/1
```

Reproduction

The raw readings, the stored reference results, the parameter sets and the harness are available on request. The generator for this document reads its figures from `results.json` and the two reference files, so the text and the appendices come from the same data.

Generated from results/bench/d9189c84/results.json, the two stored reference ladders under ops/epyc/bench/reference/, and the graph leader's footprint from the 2026-09-03 head to head, same benchmark, same class of host.

APPENDIX A

SQL MEASUREMENTS AT EACH TABLE SIZE

Every measurement the harness recorded at each table size, for all three engines.

	50,000	200,000	1,000,000	4,000,000
Scenarios compared	252	252	251	251
TETRA, sum of medians	**55.1 ms**	**82.8 ms**	**144.3 ms**	**373.1 ms**
PostgreSQL, sum of medians	1,633.5 ms	18,876.0 ms	10,244.9 ms	30,558.2 ms
DuckDB, sum of medians	412.4 ms	1,123.3 ms	1,294.3 ms	3,526.5 ms
Margin over PostgreSQL	**29.7x**	**227.9x**	**71.0x**	**81.9x**
Margin over DuckDB	**7.5x**	**13.6x**	**9.0x**	**9.5x**
TETRA no slower than PostgreSQL	208 of 252	216 of 252	210 of 251	201 of 251
TETRA no slower than DuckDB	249 of 252	248 of 252	243 of 251	226 of 251
TETRA on disk	**1.0 MiB**	**3.5 MiB**	**18.6 MiB**	**74.3 MiB**
PostgreSQL on disk	7.8 MiB	30.4 MiB	151.3 MiB	652.6 MiB
DuckDB on disk	none	none	none	none
TETRA bytes per row	**22.0**	**18.4**	**19.5**	**19.5**
PostgreSQL bytes per row	162.9	159.3	158.7	171.1
TETRA peak resident	130 MiB	280 MiB	759 MiB	3,026 MiB
PostgreSQL peak resident	3,658 MiB	3,694 MiB	3,782 MiB	4,094 MiB
DuckDB peak resident	449 MiB	495 MiB	727 MiB	1,398 MiB
TETRA suite CPU	**1.0 s**	**1.7 s**	**6.2 s**	**24.3 s**
PostgreSQL suite CPU	16.6 s	189.2 s	71.2 s	225.4 s
DuckDB suite CPU	10.8 s	24.7 s	259.5 s	4,203.3 s
TETRA bulk load	**0.3 s**	**1.0 s**	**4.7 s**	**19.4 s**
PostgreSQL bulk load	0.4 s	1.6 s	7.8 s	30.1 s
DuckDB bulk load	1.9 s	7.6 s	38.1 s	152.2 s
PostgreSQL index build	0.3 s	0.8 s	2.7 s	10.2 s

Suite CPU is the server process's CPU time for the whole manual, as reported by the operating system. Bulk load excludes the index build, which is listed separately for PostgreSQL. TETRA builds no index.

APPENDIX B

GRAPH RESULTS BY QUERY

All 21 LDBC Interactive queries at all five scale factors.

QUERY	SF1	SF3	SF10	SF30	SF100	GROWTH	ROWS
IS1	0.21	0.21	0.20	0.24	0.24	0.04	1
IS2	1.90	2.17	2.83	4.12	6.44	0.27	10
IS3	2.08	3.65	4.00	11.17	19.12	0.48	2,409
IS4	0.14	0.14	0.11	0.17	0.14	0.02	1
IS5	0.17	0.19	0.18	0.17	0.18	0.00	1
IS6	0.23	0.24	0.22	0.23	0.24	0.01	1
IS7	0.50	0.52	0.80	2.38	5.00	0.53	10
IC1	1.94	3.96	10.23	25.14	58.20	0.75	20
IC2	53.52	105.76	206.77	70.37	140.37	0.13	20
IC3	70.40	151.03	439.80	1,395.05	2,591.96	**0.81**	20
IC4	26.33	44.22	75.62	249.58	399.24	0.62	10
IC5	34.66	74.35	190.62	459.38	1,289.13	**0.78**	20
IC6	54.46	107.58	228.48	560.75	1,668.02	0.73	9
IC7	37.75	49.02	59.16	223.22	622.49	0.61	20
IC8	5.22	6.74	5.22	10.80	16.23	0.24	20
IC9	16.54	49.49	131.95	200.33	645.42	**0.75**	20
IC10	140.07	386.47	1,056.71	2,586.91	5,672.66	**0.80**	10
IC11	8.27	18.39	43.15	100.90	258.90	0.74	10
IC12	43.69	71.51	108.17	402.28	714.33	0.63	18
IC13	0.43	0.59	0.89	2.88	3.73	0.51	1
IC14	5.94	7.65	5.69	18.36	97.63	0.56	90
Total	**504.5**	**1,083.9**	**2,570.8**	**6,324.4**	**14,209.7**	**0.73**	

Medians in milliseconds. Growth is the gradient of log median against log edge count across the five sizes. A gradient of 1.00 means time grows in proportion to the data. Rows is the number of rows returned at SF100.

APPENDIX C

COMPARISON WITH THE GRAPH LEADER BY QUERY

Medians in milliseconds at SF1, SF3 and SF10.

QUERY	SF1 TETRA	SF1 LEADER	SF3 TETRA	SF3 LEADER	SF10 TETRA	SF10 LEADER	SF10 MARGIN
IS1	**0.21**	0.95	**0.21**	0.90	**0.20**	0.90	4.5x
IS2	**1.90**	7.60	**2.17**	7.45	**2.83**	10.34	3.6x
IS3	**2.08**	9.60	**3.65**	14.67	**4.00**	18.09	4.5x
IS4	**0.14**	0.87	**0.14**	0.88	**0.11**	0.82	7.7x
IS5	**0.17**	0.86	**0.19**	0.84	**0.18**	0.82	4.6x
IS6	**0.23**	0.91	**0.24**	0.86	**0.22**	0.89	4.0x
IS7	**0.50**	1.22	**0.52**	1.16	**0.80**	1.21	1.5x
IC1	**1.94**	4.90	**3.96**	22.50	**10.23**	28.45	2.8x
IC2	**53.52**	388.92	**105.76**	703.81	**206.77**	1,231.36	6.0x
IC3	**70.40**	2,955.69	**151.03**	9,212.05	**439.80**	31,415.47	71.4x
IC4	**26.33**	255.00	**44.22**	471.62	**75.62**	832.01	11.0x
IC5	**34.66**	2,199.58	**74.35**	6,793.62	**190.62**	26,440.72	138.7x
IC6	**54.46**	1,612.66	**107.58**	4,875.91	**228.48**	15,553.50	68.1x
IC7	37.75	**16.49**	49.02	**20.65**	59.16	**22.48**	0.4x
IC8	**5.22**	16.41	**6.74**	15.95	**5.22**	18.39	3.5x
IC9	**16.54**	3,008.47	**49.49**	9,018.39	**131.95**	28,638.62	217.0x
IC10	**140.07**	792.92	**386.47**	2,176.62	**1,056.71**	5,970.12	5.6x
IC11	**8.27**	47.68	**18.39**	112.43	**43.15**	233.46	5.4x
IC12	**43.69**	138.70	**71.51**	141.55	**108.17**	3,177.62	29.4x
IC13	**0.43**	1.08	**0.59**	1.06	**0.89**	1.30	1.5x
IC14	**5.94**	121.78	**7.65**	396.85	**5.69**	69.61	12.2x
Total	**504.5**	11,582.3	**1,083.9**	33,989.8	**2,570.8**	113,666.2	44.2x

Bold marks the faster engine in each pair. The margin column is the leader's SF10 median divided by TETRA's. Both engines return the same number of rows for all 21 queries at all three sizes.

APPENDIX D

COST MODEL BY MONTH

Both models at every sixth month, in dollars.

Relational, anchored on PostgreSQL's footprint

MONTH	STORED	TETRA	POSTGRESQL	DUCKDB	TETRA CUM.	POSTGRESQL CUM.	DUCKDB CUM.
1	250.0 GB	**$16**	$141	$2.9k	**$16**	$141	$2.9k
6	2.75 TB	**$70**	$616	$14k	**$255**	$2.3k	$50k
12	5.75 TB	**$134**	$1.2k	$27k	**$900**	$8.0k	$180k
18	8.75 TB	**$199**	$1.8k	$40k	**$1.9k**	$17k	$389k
24	11.75 TB	**$264**	$2.3k	$54k	**$3.4k**	$30k	$678k
30	14.75 TB	**$329**	$2.9k	$67k	**$5.2k**	$46k	$1.0M
36	17.75 TB	**$394**	$3.5k	$80k	**$7.4k**	$65k	$1.5M

Graph, anchored on the leader's footprint

MONTH	STORED	TETRA	GRAPH LEADER	TETRA CUM.	GRAPH LEADER CUM.
1	250.0 GB	**$15**	$98	**$15**	$98
6	2.75 TB	**$122**	$573	**$410**	$2.0k
12	5.75 TB	**$250**	$1.1k	**$1.6k**	$7.4k
18	8.75 TB	**$378**	$1.7k	**$3.5k**	$16k
24	11.75 TB	**$506**	$2.3k	**$6.3k**	$29k
30	14.75 TB	**$635**	$2.9k	**$9.7k**	$44k
36	17.75 TB	**$763**	$3.4k	**$14k**	$63k

Stored is the average incumbent footprint in that month. Every line item is priced against it. Monthly figures rise linearly and cumulative figures rise with the square of the month.

Calculation

For month k from 1 to 36, A is (k minus 0.5) times 500 GB, the average incumbent footprint stored in that month. The incumbent's disk, backup and cache memory are priced against A. Each other engine's disk and memory are priced against A times its measured footprint ratio. Compute is held at the top rung's measured CPU time, with the workload run once a minute. The sum of A over n months is n squared over two, in units of 500 GB-months, so cumulative cost grows with the square of time.

Early access

Test TETRA on your own data and queries.

The early-access program is a small group of teams who run TETRA on their own workloads and report what they find. Contact us to take part.

CONTACT

tetra-ca.com

inquiries@tetra-ca.com

The raw readings, the stored reference results, the parameter sets and the harness are available on request.

TETRA CA PBC