

**White Paper**

Jan 01, 2024

# **DB<sup>3</sup> Technical Overview**

**DB<sup>3</sup>**

## 1. Overview

DB<sup>3</sup> focuses on a modern database kernel. It started on March 2023, initially with one and later joined by two founding engineers. Each of them is a contributor or committer to significant OSS projects.

A modern database kernel shall prioritize resilience, be friendly to its owner developers, born with performance and scalability, and maintain an open-minded approach to possibilities.

DB<sup>3</sup> is based on the relational<sup>+</sup> model, ensuring full ACID compliance by integrating cloud-level resilience. It accommodates complex batch, streaming, and PL/SQL queries and remains adaptable to resource limitations, operating seamlessly from a standalone machine to a clustered environment. Our demonstration of its capabilities includes benchmarks like TPCC, TPCDS, TPCCH and its streaming variations.

Minimizing complexity is a promise for the DB<sup>3</sup> engineers. Ultimately, happy developers result in happy customers. Through streamlined design with the advances in database, programming languages, open-sources and environments, DB<sup>3</sup> is standing on giants' shoulder while shedding numerous traditional redundancies. DB<sup>3</sup> achieves functionalities and exceptional performance with significantly reduced codebase size compared to traditional implementations.

The majority of the code is written in C++23 and Rust. Differentiator components, such the storage engine, runtime and optimizer, are entirely new implementations from the ground up. Development progress is monitored through CI process. It comes with complete git logs, detailed commit messages and design documents to facilitate further developments.

## 2. Scenarios

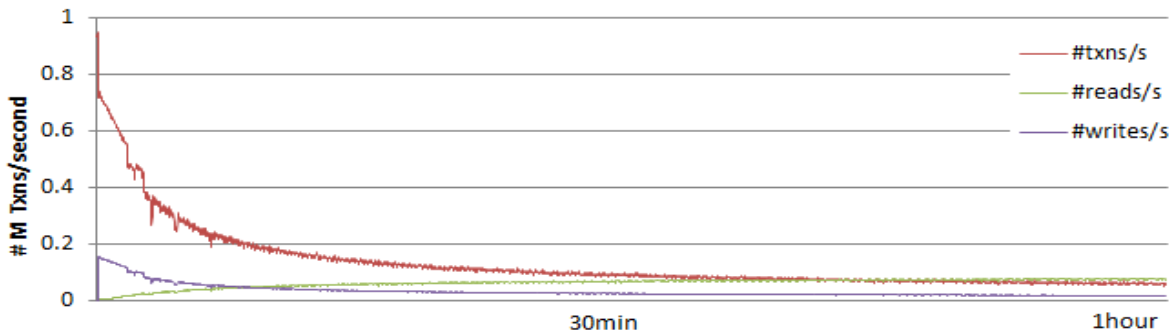
In this section, we assess DB<sup>3</sup>'s capabilities across various scenarios using corresponding benchmarks. Except for distributed benchmarks, the machine we are using is the same desktop with an Intel i9-13900k processor (8 3G performance-cores + 16 efficient-cores), 64G memory, and one P7000Z-4TB NVMe SSD drive (Official parameters: 7.45G/860K IOPS, write: 6.75G/690K IOPS).

### 2.1 OLTP

We evaluate DB<sup>3</sup>'s OLTP capacity by standard TPCC benchmark.

The TPCC five transactions and consistency checks are written in PL/SQL. The benchmark is configured with 8 warehouses. It runs using 8 hardware threads for 1 hour. The maximal data cached in memory is set to a fixed 4G.

So during the first 4 seconds, DB<sup>3</sup> is running at full speed when data does not exceed the 4G limit. Later when data grow out of the limit, DB<sup>3</sup> does data replacement by reading from / writing to the SSD. The benchmark is running with isolation level default to SI, logs are generated but not write out. At the end of the test, tpcc consistency conditions are verified for transaction compliance.



Configuration: 8C/4G fixed data cache/1-SSD, 8 Warehouses

The experiment demonstrates DB<sup>3</sup>'s transaction and PL/SQL JIT capacity. Even when compared to cutting-edge memory-only engines, which often evaluated via C/C++ APIs without a SQL layer, DB3 stands strong against them, achieving a rate of 0.94M/s.

Its performance declines smoothly when memory is insufficient. With only 1 SSD drive, it remains superior (0.06M/s at 1-hour point) to traditional disk-based engines. There are some development work going on and we expect to push performance to 0.08M ~ 0.1M/s at 1 hour point.

## 2.2 OLAP/Batch

We evaluate DB<sup>3</sup>'s OLTP capacity by standard TPCB and TPCDS benchmark.

In our testing of benchmarks at various scales, DB<sup>3</sup> utilizes external Parquet files generated by DuckDB for storage, while DuckDB loads data into its native format. TPCB tests are conducted with DOP=8. Note that due to limitations in its operator support for spilling, DuckDB is unable to run at the 1000 scale (1T data) on this machine. Regarding the TPCDS benchmark, currently operates only with DOP=1, as efforts are underway to parallelize and optimize the necessary physical operators.

Following the practice of TPCB/TPCDS benchmark, we present both SUM and GEOMEAN of run time, where SUM indicates the overall performance and GEOMEAN measures the central tendency.

| DuckDB 0.81 (s)  |     |      |     |      |     | DB <sup>3</sup> |      |     |      |
|------------------|-----|------|-----|------|-----|-----------------|------|-----|------|
| TPCH-30          | 100 | 1000 | SUM | 18   | 68  | -               | 12   | 46  | 1110 |
| GEOMEAN          |     |      |     | .58  | 2.1 | -               | .42  | 1.6 | 37.2 |
| TPCDS-30 (DOP=1) |     |      | SUM | 241  |     |                 | 246  |     |      |
| GEOMEAN          |     |      |     | 0.89 |     |                 | 0.86 |     |      |

Configuration: 8C/64G/1-SSD

The following distributed TPCB-1000 test is conducted on 4 aliyun ECS.R8I.2XLarge/PL2 instances (8vCPU/64G/350M/Network unknown). Each instance contains ¼ data in mounted drive. Notes: (1) GreenPlum is configured to the best of our knowledge with GUC adjustments<sup>1</sup>. (2) Both are running with DOP=1, as DB3 is working on parallelize query with distributed deployment.

| GreenPlum 7.0 (s)           |      | DB <sup>3</sup> |
|-----------------------------|------|-----------------|
| TPCH-1000 4xcluster (DOP=1) | 9449 | 3267            |
| GEOMEAN                     | 281  | 122             |

Configuration: 4 \* ECS.R8I.2XLarge/PL2

Above experiments demonstrate DB<sup>3</sup>'s analytical processing capacity, supporting complex queries with details like NULL optimization, math overflow and type coerce all handled. It also demonstrates DB<sup>3</sup> is flexible with resource constraints where it can run on single machine or cluster, can run with or without sufficient memory.

## 2.3 OLAP/Streaming

We evaluate DB<sup>3</sup>'s OLAP/Streaming capacity by a variant of TPCB benchmark for streaming data sources.

The TPCB-streaming benchmark is using original TPCB queries. It emulates the largest two tables LINEITEM (180M rows) and ORDERS (45M rows) as streaming sources, while keep others are original static tables. "Stream/<N>" means each streaming barrier consumes around 1/N data, where we exercise an incremental materialized view maintenance (IMV). We can run 15 out of 22 queries, while the rest queries need more development work.

|                       | Batch(s) | Stream/1000 |
|-----------------------|----------|-------------|
| TPCH 30-Streaming SUM | 8        | 29          |
| GEOMEAN               | .4       | 1.2         |

Configuration: 8C/64G/1-SSD, 15/22 queries

This benchmark demonstrates DB<sup>3</sup> can handle complex streaming queries from optimizer to runtime, exhibits excellent low incremental maintenance overhead. Recall in TPCC benchmark, we showed exceptional transactional throughput, so DB<sup>3</sup> itself can acts as a frontend to ingest events or state management.

## 2.4 Possibilities with DB<sup>3</sup>

DB<sup>3</sup> is building on relational<sup>+</sup> model, which is a foundation to embrace old new data processing scenarios present in future. It encourages more features, less products.

- Graph/Stream/Vector/... processor: SQL2023/Property Graph (PG) standard essentially says "just do it". Streaming is already in ANSI's formal meeting. Will Vector be next? These extensions keep pushing more features into relational<sup>+</sup> model's atlas.
- With the open mindsets and its extensible architecture, DB<sup>3</sup> hopefully enables new scenarios without compromise and with reuse. For example, supporting both OLTP and OLAP actually mutual beneficial: TP fallbacks to AP to handle complex queries, while TP sets a solid foundation for AP's data ingestion or meta-data management.
- DB<sup>3</sup> is built with a set of core libraries and multiple servers. With this structure, we may package it as a new database, or integrate into existing kernel like PostgreSQL extension, or as plug-in query accelerator following substrait protocols, or introduce DataFrame APIs for data scientists, and more.

### 3. Formulas

DB<sup>3</sup> follows the classic three major components design: optimizer, runtime, storage engine. For each one, we do a ground up new implementation with significant innovation and modernization. In this section, we will briefly discuss how each component works and some highlights.

#### 3.1 Optimizer: DAG $\mapsto$ IRs

The query optimizer is responsible for translate query into optimized plan or code.

- After parser translates the PL/SQL into a parse tree, the optimizer applies a DAG oriented optimization. A DAG is considered superior to a tree for sub-plan sharing purpose.
- Following the Cascades framework, DB<sup>3</sup> implements a top-down rule based CBO. It further accelerates planning by early avoiding or pruning without compromising plan quality. The framework supports rule based transformations with logical/physic property enforcement and stats propagation. Optimization rules include completed subquery decorrelation; cost based remote exchange marking, aggregation placement, CTE pooling/inlining, and numerous others.
- The optimized PL/SQL queries and control statements are either run in interpreter mode, or compiled into IRs and further lowering down layer by layer to instructions and runtime primitives for execution. This method comes with much better debug-ability than traditional LLVM based method without sacrificing performance.
- Optimizer extensively uses compile time code generation help to reduced 12K+ lines of boilerplate code for serde, clone, hash/equal etc. It is built on DB<sup>3</sup>'s managed memory infrastructure as easy as coding in GC supported language.

#### 3.2 Runtime: (R + B + ...) $\times$ JIT $\neq$ Optimizer

Runtime is responsible for types and primitives, execution and optimizer feedback.

- A query can run in row, batch or mixed mode. Allowing mixed run is benefits customers and internally best leveraging each other's power. JIT is an accelerator for both modes.
- Realizing batch (bound) and streaming (unbound) query processing's duality, runtime implements a unified  $V^{UB}$  vector engine which shares native parallelization, spill support for both batch and streaming query. This largely reduces development efforts.
- Batch mode runtime is for performance: native parallel operators with light memory footprint; parallel scheduling effectively addresses skew and scales; cost based remote exchange placement maximally reduces data shuffling in cluster deployment. Besides that, it supports controls like pause/resume/cancel and other query resilience features.
- Runtime is flexible with resource constraints and is optionally stateless. It can complete OLTP queries in micro-seconds, and it can also run large OLAP queries in small machine or cluster. In distributed mode, a remote runtime depends on a self-contained plan without catalog dependencies.

Here is a table summarizing row vs. batch mode:

| Row | Batch (bounded + unbounded) |
|-----|-----------------------------|
|     |                             |

| Target                  | Mediocre                                      | Analytic, Streaming/IMV                                |
|-------------------------|---|--|
| Functional completeness | Partial + fallback to batch                   | Partial + fallback to row                              |
| Resource constraints    | Query is in memory but data cache can do swap | Minimal requirement on memory for both query and cache |
| JIT                     | Full stack                                    | Partial  |
| Scale up/out            |   | Full   |
| Query resilience        |   | Yes  |
| Error handling          | Abort   | Abort or partial restart                               |
| Pause/resume/cancel     | Partial                                       | Yes  |
| Always-on profiling     |   | Yes  |

### 3.3 Storage Engine: Local 𐀀 + Remotes 𐀁

The storage engine is responsible for transactions, data accesses and resiliency.

- DB<sup>3</sup> is designed to differentiate while seamlessly coordinate local and remotes: local handles transactions while remote storage is the final guarantee of capacity and resilience; Local processes read/write while remotes offload read-only; Local runs user centric logics while shifting maintenance to remote spot stances.
- Built on tiered low latency local + high throughput and highly reliable remote shared storage, it achieves performance and resilience while simplifies implementation. It designs to leverage shared storage and idle computational power in cloud-like environment.
- The local/remote protocol is TLA+ proved. It is an ACID implementation with maximal SI isolation level. It is supported by fully parallel SI maintenance, GC and write-behind-logging, recovery. Memory and IO optimized indexing with memory size limit management. The coroutine based backbone hides I/O and synchronization latency without developer's awareness. This allows us to handle many connections easily.
- The catalog prioritizes reads over writes and diverges from the traditional schema design. The approach results in a much simpler implementation, including support for online DDLs.

## 4. Conclusion

DB<sup>3</sup> is treating resilience with priority, friendly to its owner developers, born with performance and scalability, and maintains an open-minded architecture. The DB<sup>3</sup> team is looking for anything to take it to next.

---

<sup>i</sup> GreenPlum is configured with the following GUC modifications:

```
shared_buffers = 131072
max_statement_mem = 32GB
gp_vmem_protect_limit = 32GB
work_mem = 32GB
max_parallel_workers_per_gather = 0
```

Greenplum schema is without PK/FK. We have experimented to add PK/FK constraints which actually largely degrade its benchmark performance.