UNIVERSITÉ LIBRE DE BRUXELLES

Benchmarking Embedded Databases on IoT Sensor Data

INFO-H-415 Advanced Databases

ZIMANYI Esteban

DON Basem
CHEOUIRFA Anas
ROGGE Ethan

Outline

- Motivation of this benchmark
- Berkeley DB low-level, key-value embedded engine
- DuckDB analytical, in-process SQL DB
- PostgreSQL used as a baseline
- Methodology
- Key results (OLTP / OLAP)
- Comparative takeaways & recommandations

What is an Embedded Database ?

Library

As a library inside the application

Local

No server process, no network communication

Execution

Direct in-process execution

Storage

Data stored locally (files or memory)

Embedded Database

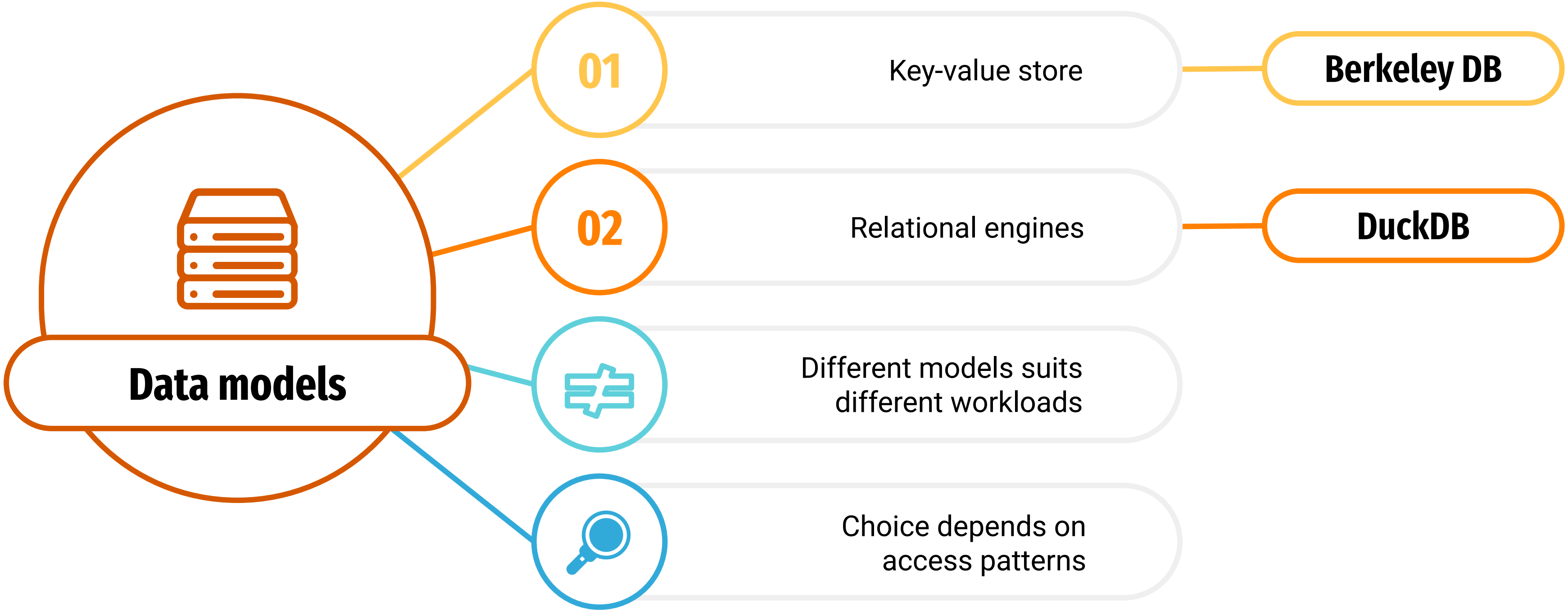
Advantages

1. Low latency
2. Simple deployment
3. Small footprint

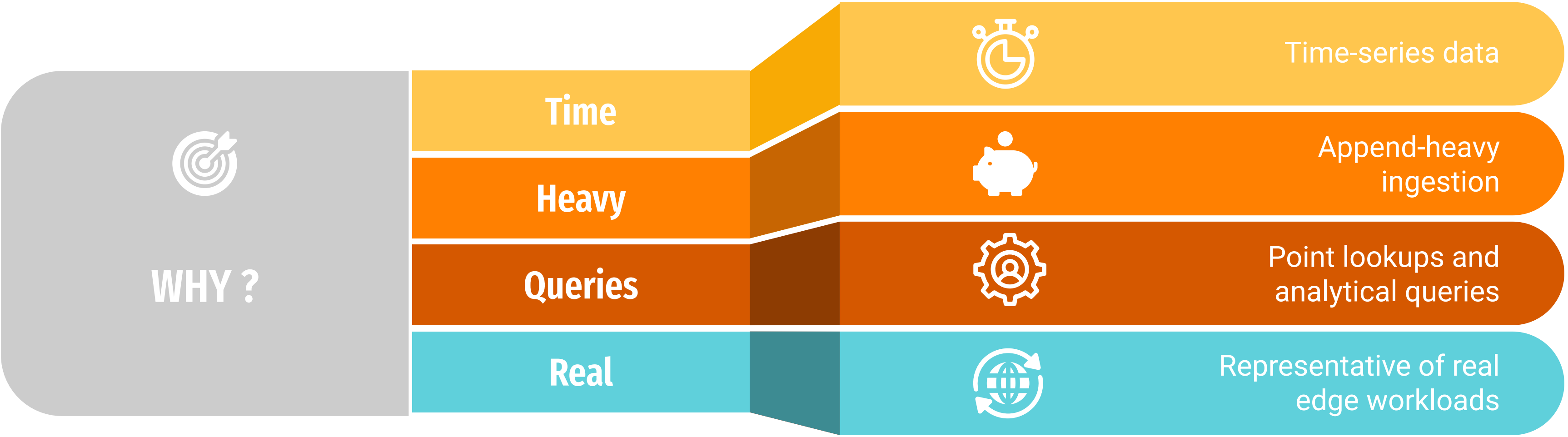
Trade-offs

1. Limited concurrency
2. Reduced expressiveness
3. Application-level logic

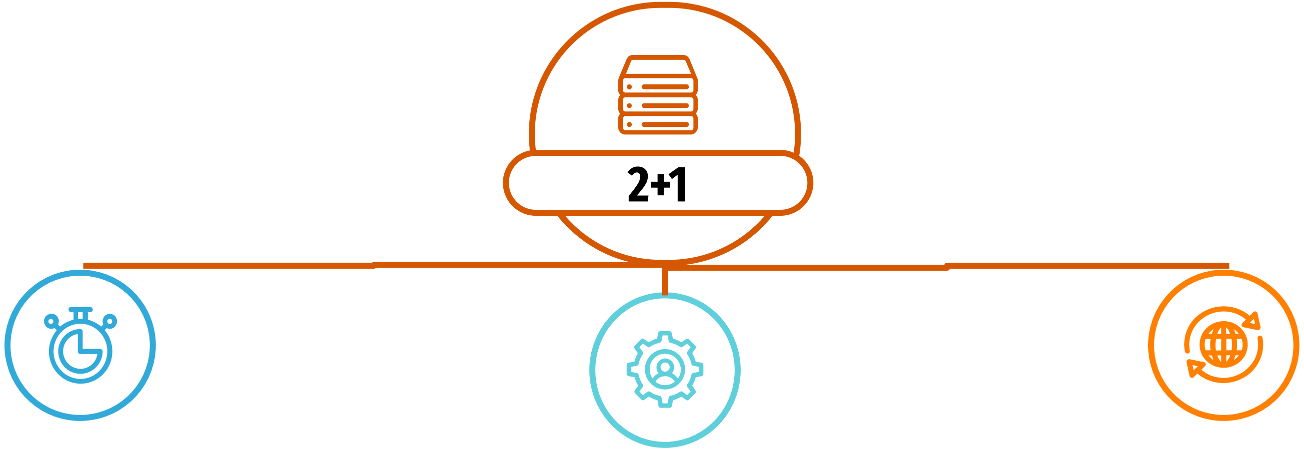
Data models


Data models
01
Key-value store
Berkeley DB
02
Relational engines
DuckDB
Different models suits
different workloads
Choice depends on
access patterns

Why Iot Sensor Data ?



Tools overview



Berkeley DB : transactional key-value store

DuckDB: analytical embedded SQL engine

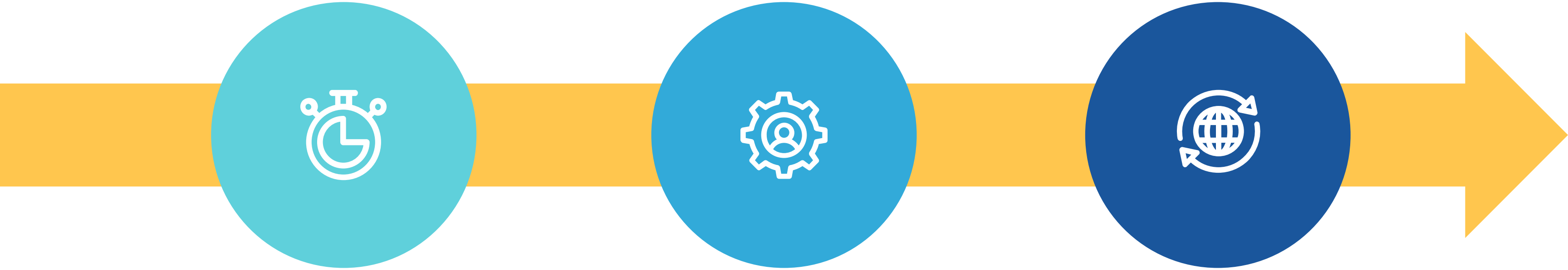
PSQL: to compare

Berkeley DB - Positioning

Mature

Designed

widely



Mature embedded key-value database

Designed for reliability and low overhead

Widely used in embedded and edge system

Berkeley DB - Architecture

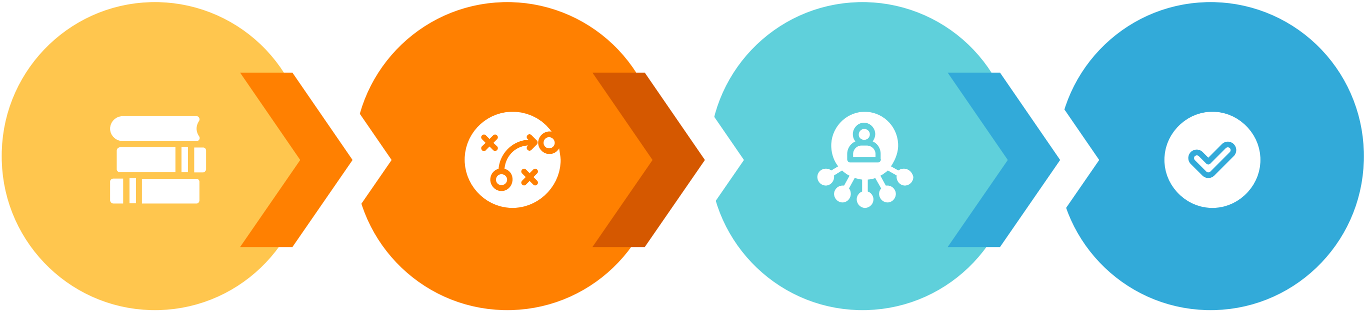
Key-value data model

B-tree, hash and record-based storage

Page-based engine with buffer cache

Write-ahead login (WAL)

Berkeley DB - Transactions & Concurrency



ACID

Full ACID guaranties

Grained

Fine-grained locking

optimized

Optimized for single-application access

Predictable

Predictable performance

Berkeley DB - Capabilities & Limits

Excellent

1. Fast insert
2. Point lookups

Limitations

1. No SQL
2. No build-in analytics

DuckDB - positioning

Modern in-process analytical SQL engine
columnar storage & vectorised execution

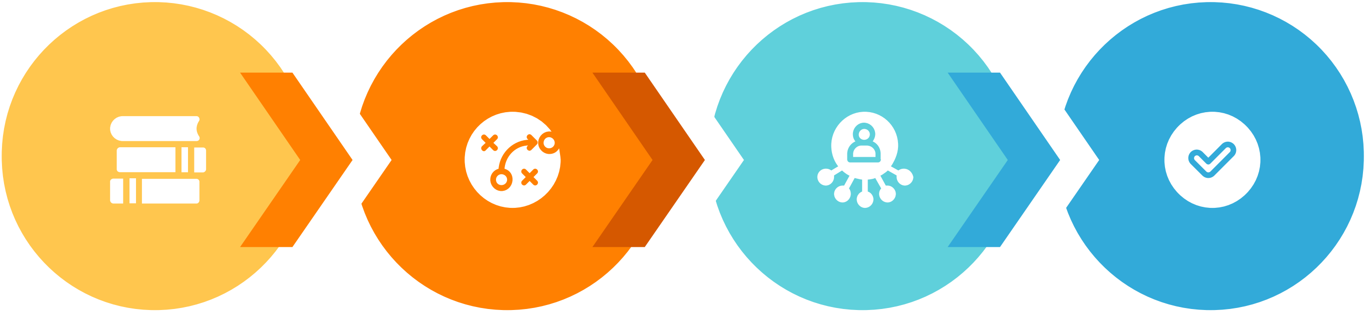
Optimised for OLAP
fast scans, joins, aggregations and native Parquet/CSV access

Analytical
Ideal for embedded analytics, data science and notebook workflows

Rich language bindings

Python, R, Java, Node.js, etc

DuckDB - Architecture



Columnar

Columnar on-disk/in-memory storage (values stored per column in compressed vectors)

Vectorised

Vectorised execution engine (batch processing, SIMD-friendly, pipelined operators)

Native

Native integration with Parquet/CSV and file-backed segments for zero-load analytics

MVCC

MVCC single-writer / multi-reader model with WAL for durability (optimised for read-heavy OLAP)

DuckDB - Capabilities & Limits

Excellent for

1. In-process analytical SQL: complex joins, aggregations, window functions
2. Large sequential scans and columnar analytics (fast compression + SIMD)

Limitations

1. Not designed for high-frequency point updates, deletes or many small random writes
2. Single-writer model (MVCC) limits concurrent writers
3. Higher memory/CPU during heavy analytical queries; less suitable for extremely constrained devices
4. Slower for repeated low-latency point lookups compared to key–value engines

Dataset

Intel Berkeley Research Lab dataset

1. Real-world sensor data
2. Temperature, humidity, light, voltage
3. Scales: 1K, 10k, 100K, 1M rows

Benchmark Methodology

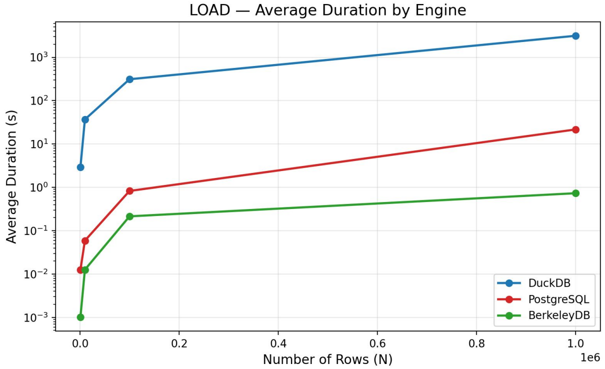
Same machine for all systems

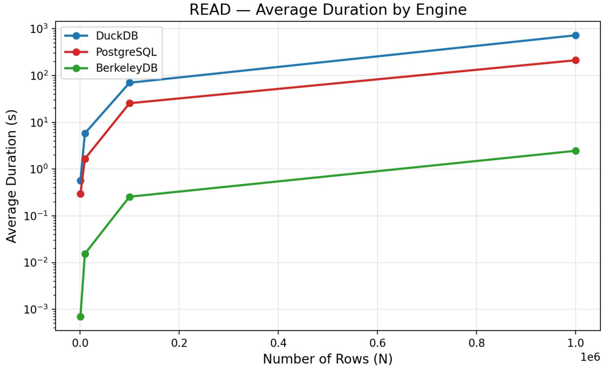
Same dataset and queries

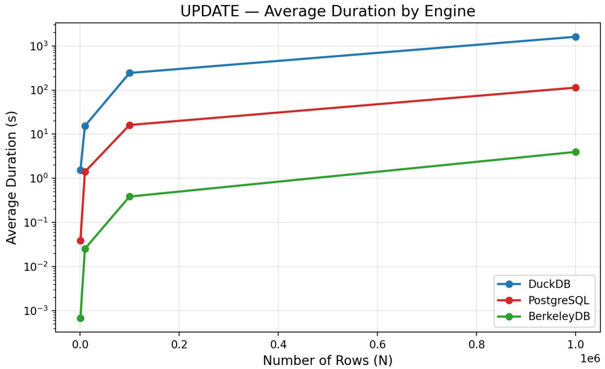
Each query executed 6 times

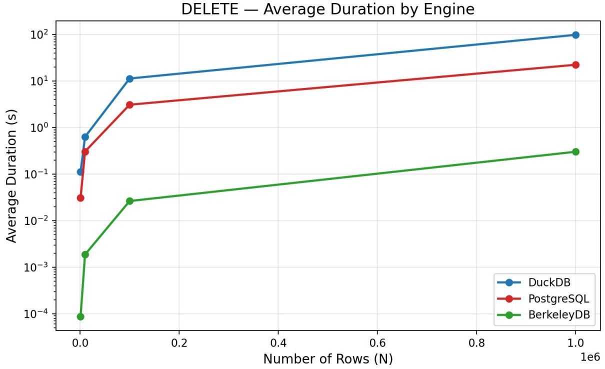
First run discarded, average reported

OLTP Queries Results







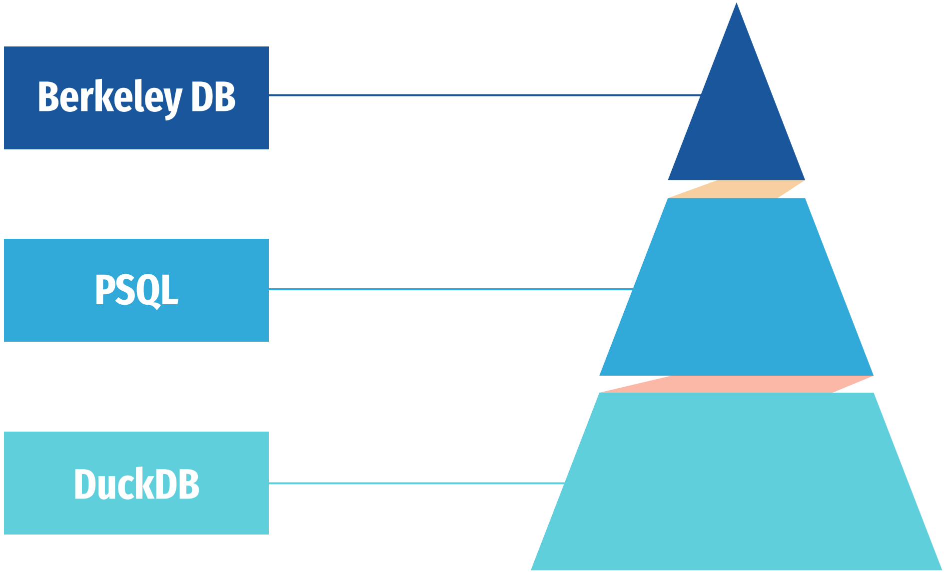


OLTP Results - Explanation

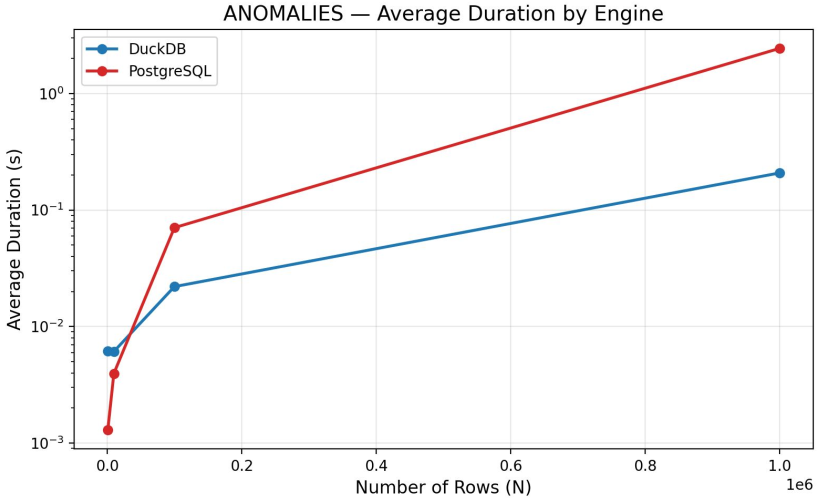
Direct key-value access, minimal execution path

Index-based access but SQL + MVCC overhead

Columnar layout -> write amplification & poor random access

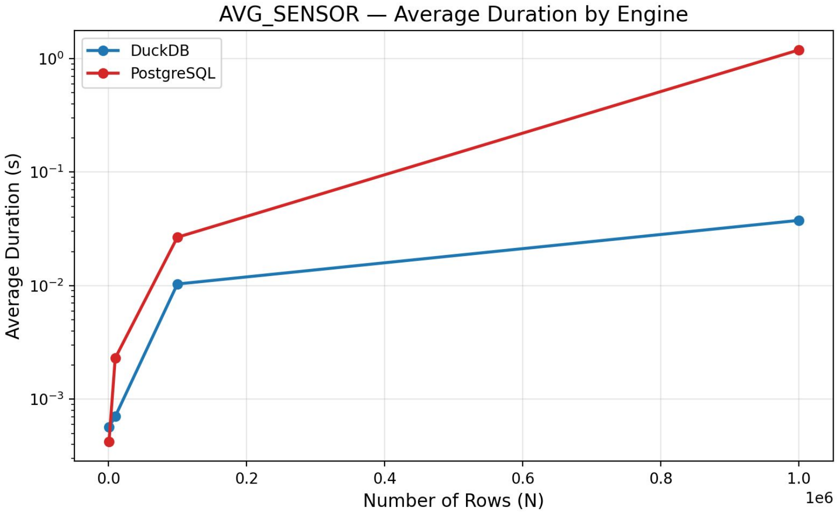


ANOMALIES



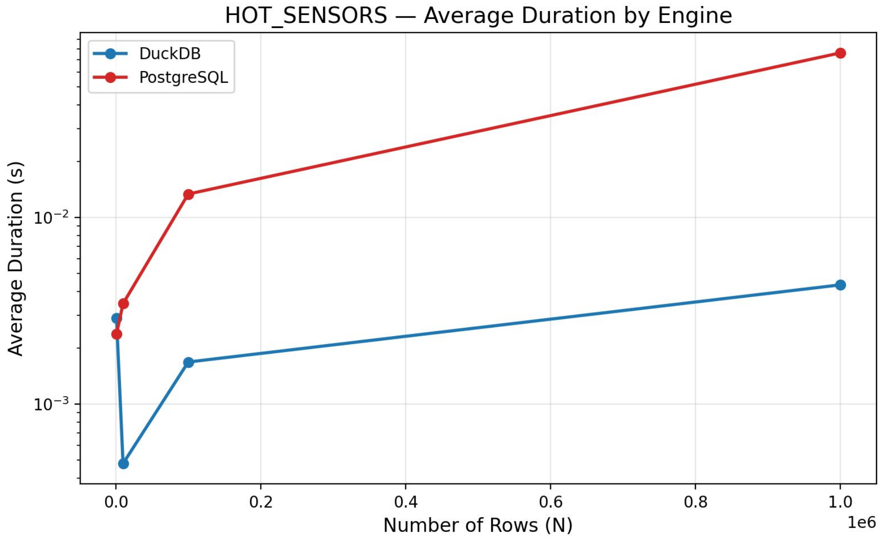
- Combinaison of aggregations and joins or window functions

AVG_SENSOR



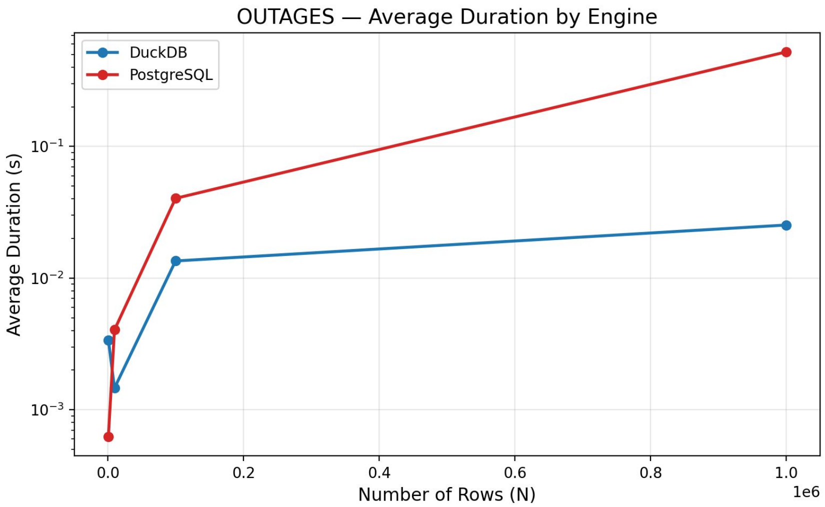
- GROUP BY query: for each sensor id, compute AVG(temperature)

HOT_SENSORS



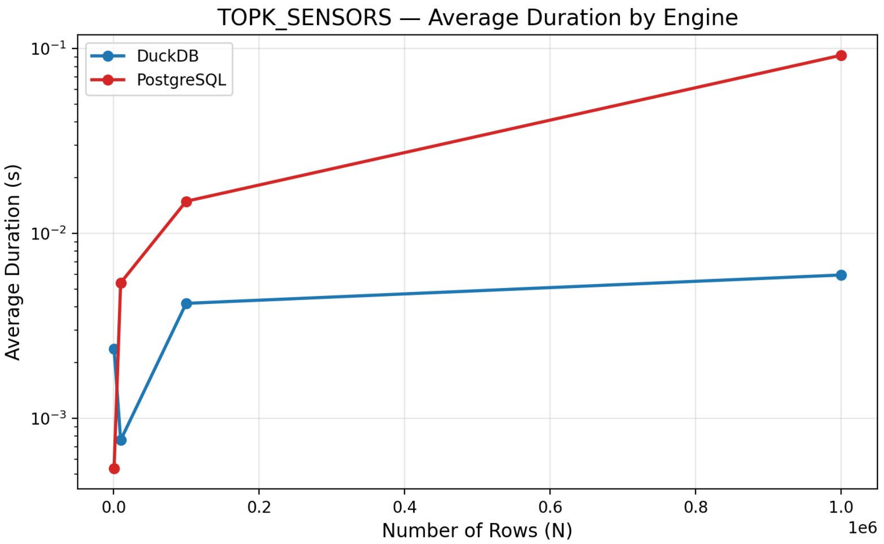
- Combine aggregation (AVG) with a HAVING filter and optionally a LIMIT k

OUTAGES



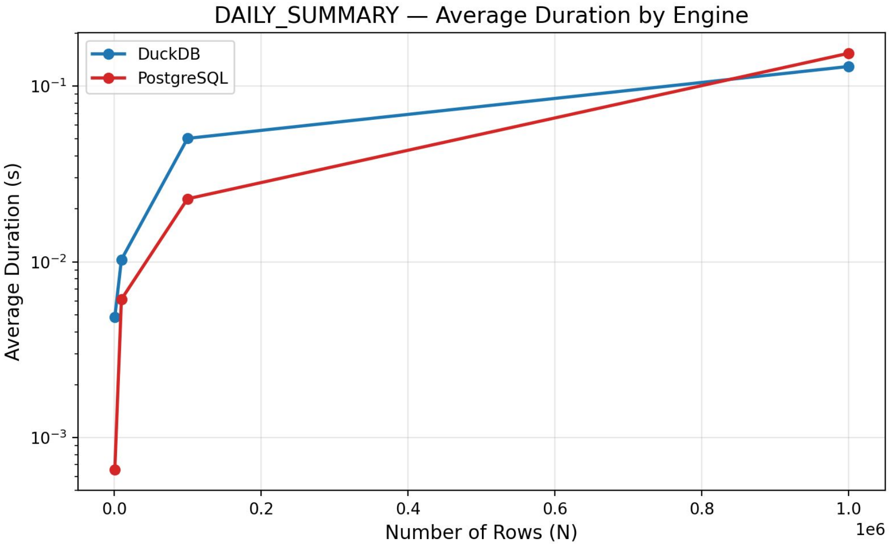
- Window function such as LAG(timestamp) partitioned by sensor id and ordered by time

TOPK_SENSORS



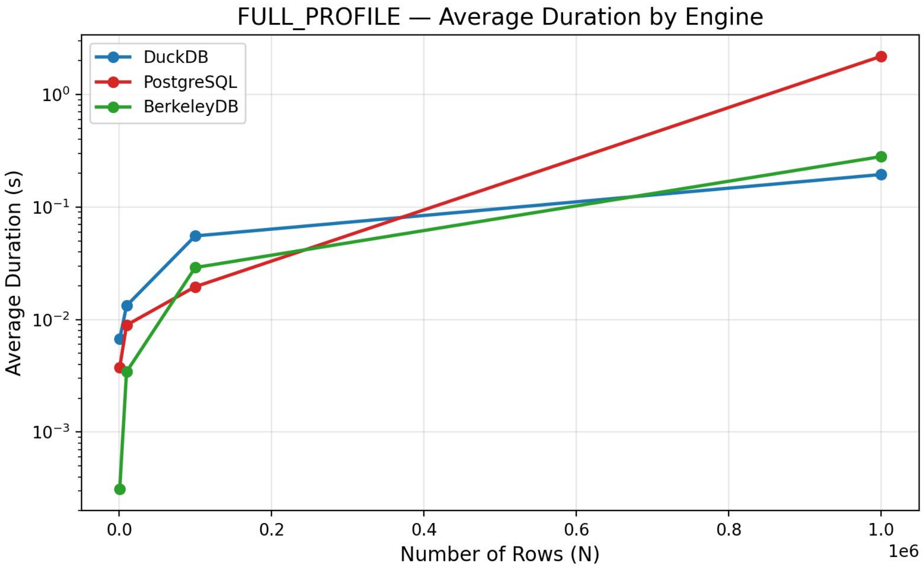
- GROUP BY followed by an ORDER BY and LIMIT

DAILY_SUMMARY



- GROUP BY sensor id and date and applying MIN, MAX and AVG aggregations

FULL_PROFILE



- DuckDB & PSQL: sum of averages, daily summary, outages and top-k sensors
- Berkeley DB: simple local statistical summary over the most recent cached values

Analytical SQL Operations Results - Explanation

Columnar storage + vectorized execution (SIMD, cache-friendly)

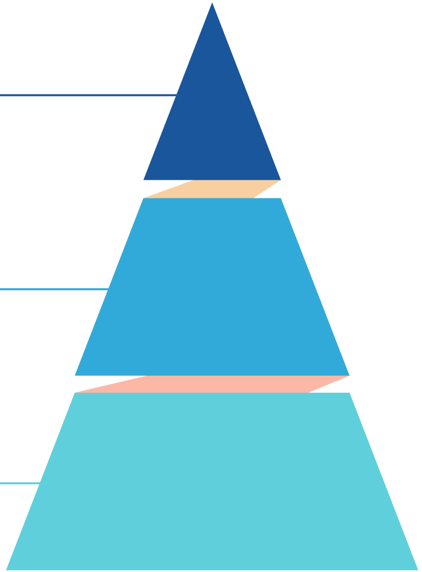
DuckDB

Row-oriented, tuple-at-a-time execution

PSQL

No native analytical operators

Berkeley DB



Comparative Assessment

Berkeley DB — Best for edge ingestion

DuckDB — Best for embedded analytics

PSQL — General-purpose baselinet

Conclusion

- No single best database
- Architecture determines performance
- Hybrid architectures are often optimal