

Database benchmarks:

Lessons learned from running a benchmark standard organization

Gábor Szárnyas

FOSDEM | Software Performance devroom | 2026-02-01

About me

2014–2023
academia



2018–
benchmark non-profit



2023–
database vendor



DuckDB Labs

**database
benchmarks**



The need for
DB benchmarks



The TPC



The LDBC



Popular
benchmarks



Takeaways

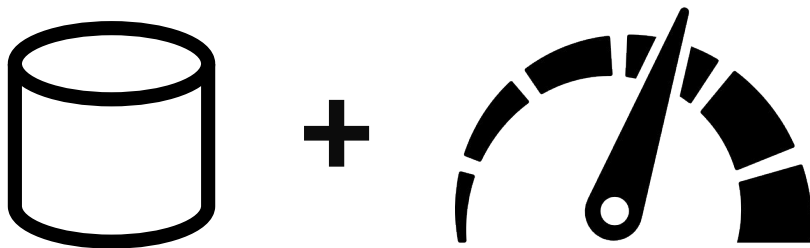
1. The need for DB benchmarks



Database benchmarks

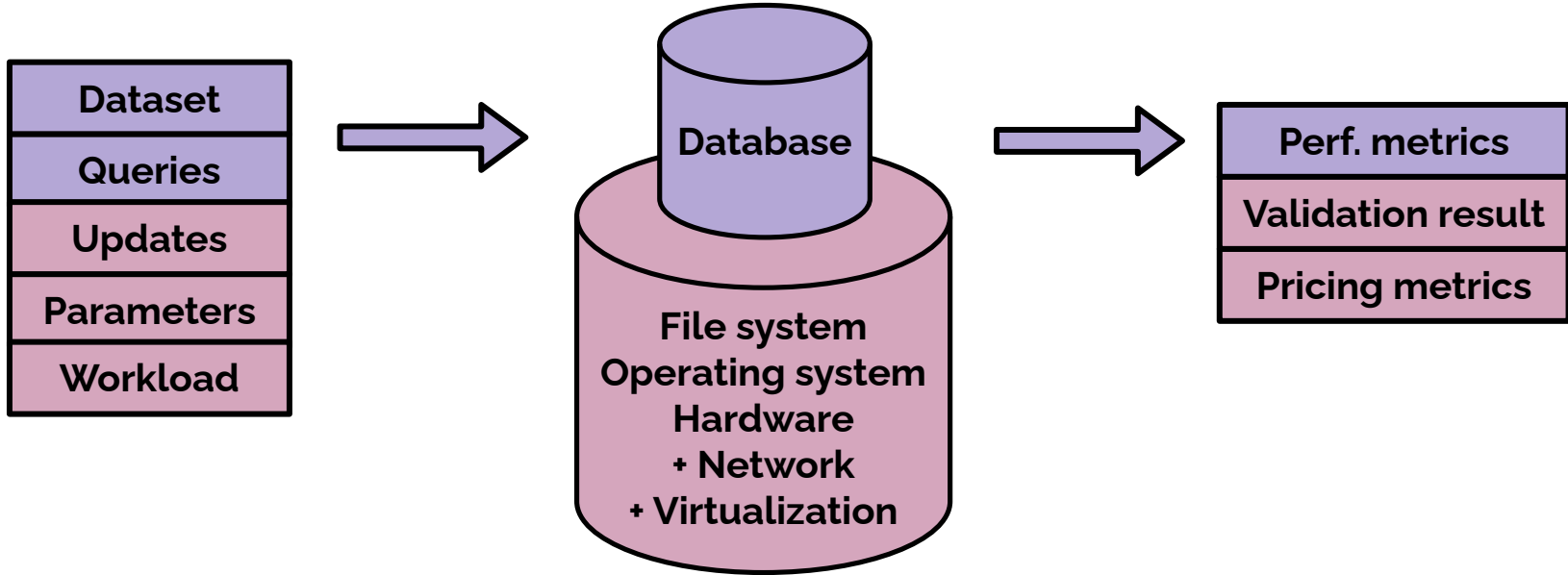
Databases and benchmarks: *a great match*

- users care (to some extent)
- vendors are incentivized
- (mostly) deterministic execution and results



Expectation Reality

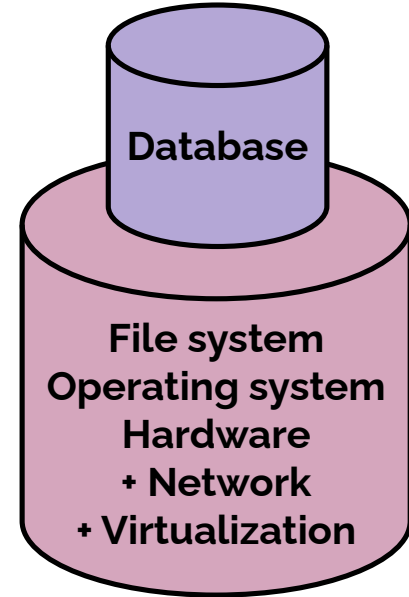
p6



Running the same code path (?)

p7

- Reproducibility is difficult
- Cold ... hot runs
- Lukewarm ... warm runs



Reporting summary statistics

 T. Hoefler, R. Belli, [Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results](#) (2012) [[talk](#), [recording](#)]

Rule 3:

Use the **arithmetic mean** only for summarizing **costs**.

Use the **harmonic mean** for summarizing **rates**.

$$\text{HM}(x_1, \dots, x_n) = \frac{n}{\frac{1}{x_1} + \dots + \frac{1}{x_n}}$$

Rule 4:

Avoid summarizing **ratios**; summarize the costs or rates that the ratios base on instead.

Only if these are not available use the **geometric mean** for summarizing ratios.

$$\text{GM}(x_1, \dots, x_n) = \sqrt[n]{|x_1 \times \dots \times x_n|}$$


What scope to measure?

p9

Microbenchmark

Macrobenchmark

Application-level
benchmark



*more complex
more realistic*

1980s: Benchmark wars

p10

Relational databases are immature and have performance problems

“Benchmarking”: vendors implement their own benchmarks and boast their results

1982: the [DeWitt Clause](#) prohibits publishing unsanctioned benchmark results

Need an independent authority and a standard



2. TPC overview





Transaction Processing Performance Council

Non-profit founded in 1988

Registered in the US

Mission: to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry.

TPC member organizations

p13



approx.: 11  8  1  1 

TPC-H

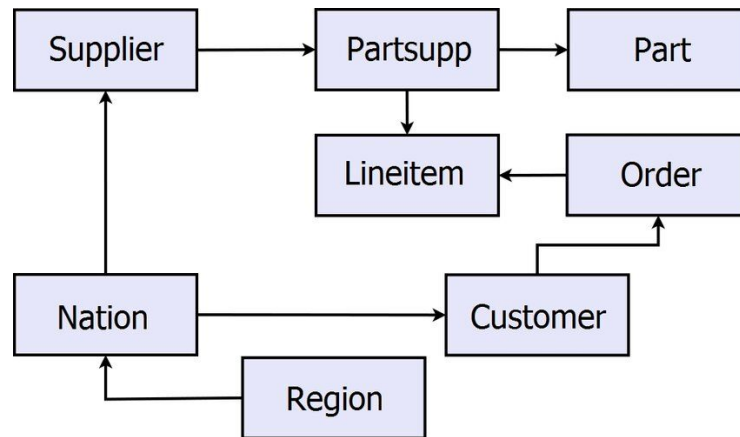
—

TPC-H

p15

Ad-hoc analytics for a wholesale supplier

- 8 tables
- 22 queries
- Simple refresh operations (insert / delete)



Released in 1999

TPC-H data generation

Scale factor (SF) is the datasets size in GiB when serialized in CSV format

SF100 = 100 GiB CSV files

In 2025, the TPC-H dbgen was rewritten in Rust: [clflushopt/tpchgen-rs](https://github.com/clflushopt/tpchgen-rs)

```
$ pip install tpchgen-cli
$ time tpchgen-cli --scale-factor 100
... 58.723 total
```

TPC-H Q1

p17

```
SELECT l_returnflag, l_linestatus,
```

```
  sum (l_quantity) AS sum_qty,
```

```
  sum (l_extendedprice) AS sum_base_price,
```

```
  sum (l_extendedprice * (1 - l_discount)) AS sum_disc_price,
```

```
  sum (l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
```

```
  avg (l_quantity) AS avg_qty,
```

```
  avg (l_extendedprice) AS avg_price,
```

```
  avg (l_discount) AS avg_disc,
```

```
  count (*) AS count_order
```

```
FROM lineitem
```

```
WHERE l_shipdate <= DATE '1998-12-01' - INTERVAL '$1' DAY
```

```
GROUP BY l_returnflag, l_linestatus
```

```
ORDER BY l_returnflag, l_linestatus;
```

TPC-H benchmark workflow

p18



Power test: sequential execution → geometric mean runtime

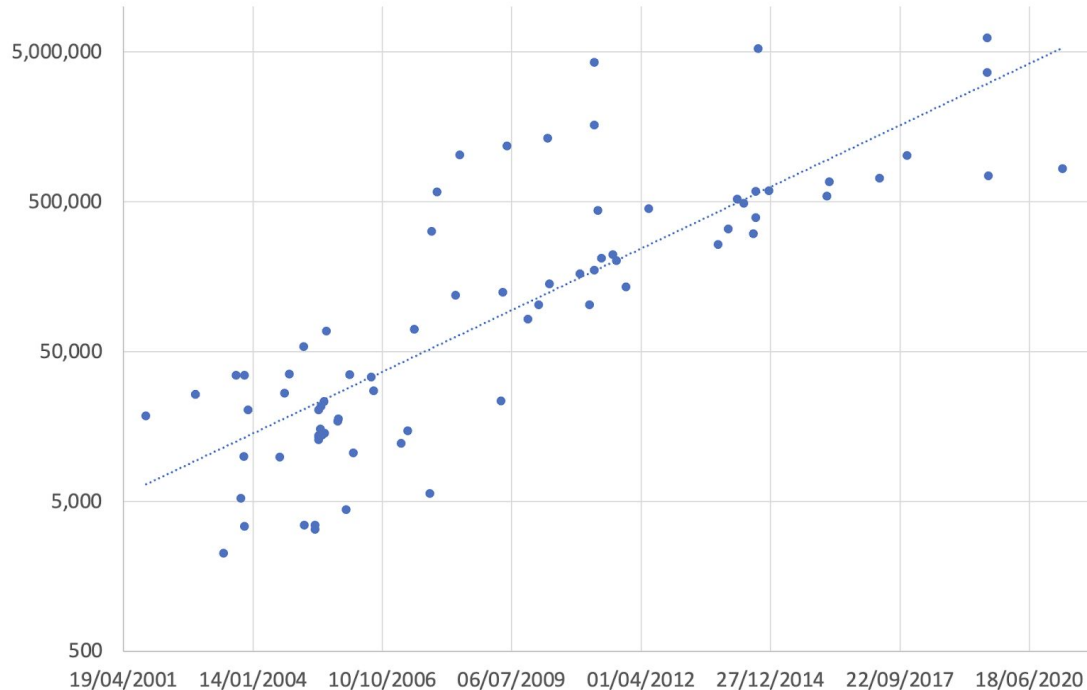
Throughput test: concurrent execution → throughput [queries / s]

Composite metric: $Q_{phH} = \sqrt{\text{power} \cdot \text{throughput}}$

Analytical workloads (2001-2020)

p19

TPC-H v2 Performance (QphH) on the SF1,000 data set



**~1000× performance
increase over 20 years
(41% YoY)**

TPC-H breakdown by choke points

p20

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22
CP1 Aggregation Performance. Performance of aggregate calculations.																					
■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CP1.1 QEXE: Ordered Aggregation.																					
CP1.2 QOPT: Interesting Orders.																					
CP1.3 QOPT: Small Group-by Keys (array lookup).																					
CP1.4 QEXE: Dependent Group-By Keys (removal of).																					
CP2 Join Performance. Voluminous joins, with or without selections.																					
■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CP2.1 QEXE: Large Joins (out-of-core).																					
CP2.2 QEXE: Sparse Foreign Key Joins (bloom filters).																					
CP2.3 QOPT: Rich Join Order Optimization.																					
CP2.4 QOPT: Late Projection (column stores).																					
CP3 Data Access Locality. Non-full-scan access to (correlated) table data.																					
■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CP3.1 STORAGE: Columnar Locality (favors column storage).																					

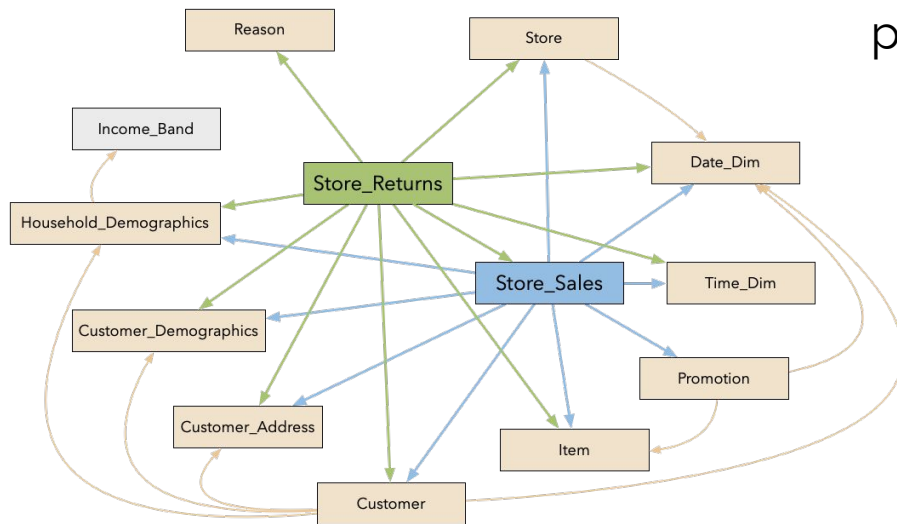
TPC-DS

—

TPC-DS

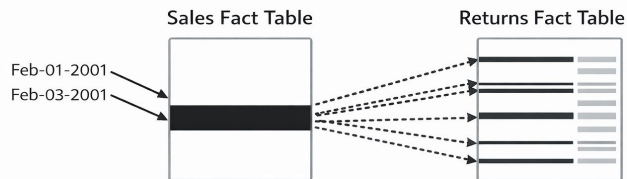
Models a retail product supplier with multiple sales channels

- 24 tables
- 99 queries (~1800 chars/query)
- more complex deletes



p22

Delete Operation




J. Menzler, [A Summary of TPC-DS](#) (2019)

M. Poess et al., [Why you should run TPC-DS: A workload analysis](#) (2007)

TPC-DS

Timeline:

- 2011: Workload released
- 2018: First audit by Cisco, SF10,000
- 2021: Record-breaking SF100,000 audit by Databricks
and subsequent benchmark war with Snowflake

 [Cisco UCS publishes the first ever audited result of the TPC-DS benchmark with Hadoop](#) (2018)

 [Alibaba Cloud E-MapReduce sets world record again on TPC-DS benchmark](#) (2020)

 [Databricks sets official data warehousing performance record](#) (2021)

 [Industry benchmarks and competing with integrity](#) (2021)

 [Snowflake claims similar price/performance to Databricks, but not so fast!](#) (2021)

TPC's auditing process

—

Auditing process

The **test sponsor** commissions the audit

The **test sponsor** runs the experiment and writes the full disclosure report

The **auditor** validates the results

The **auditor** can perform re-runs, additional checks, durability test, etc.

Strict rule: no “benchmark specials”

TPC-H Q1

p26

```
SELECT l_returnflag, l_linestatus,  
       sum (l_quantity) AS sum_qty,  
       sum (l_extendedprice) AS sum_base_price,  
       sum (l_extendedprice * (1 - l_discount)) AS sum_disc_price,  
       sum (l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,  
       avg (l_quantity) AS avg_qty,  
       avg (l_extendedprice) AS avg_price,  
       avg (l_discount) AS avg_disc,  
count (*) AS count_order  
FROM lineitem  
WHERE l_shipdate <= DATE '1998-12-01' - INTERVAL '$1' DAY  
GROUP BY l_returnflag, l_linestatus  
ORDER BY l_returnflag, l_linestatus;
```

TPC-H Q1 (garbled)

p27

```
SELECT t_yZXR1cm5mbGFn, t_saW5lc3RhdHVz,  
       sum (t_xdWFudGl0eQ) AS sum_qty,  
       sum (t_leHRlbnRlZHB5) AS sum_base_price,  
       sum (t_leHRlbnRlZHB5 * (1 - t_kaXNjb3V)) AS sum_disc_price,  
       sum (t_leHRlbnRlZHB5 * (1 - t_kaXNjb3V) * (1 + t_0YX)) AS sum_charge,  
       avg (t_xdWFudGl0eQ) AS avg_qty,  
       avg (t_leHRlbnRlZHB5) AS avg_price,  
       avg (t_kaXNjb3V) AS avg_disc,  
count (*) AS count_order  
FROM db_bGluzWl0ZW0K  
WHERE t_zaglwZGF0ZQ <= DATE '1998-12-01' - INTERVAL '$1' DAY  
GROUP BY t_yZXR1cm5mbGFn, t_saW5lc3RhdHVz  
ORDER BY t_yZXR1cm5mbGFn, t_saW5lc3RhdHVz;
```

Prevents systems from using
hard-coded query plans!

TPC-Pricing: Total Cost of Ownership

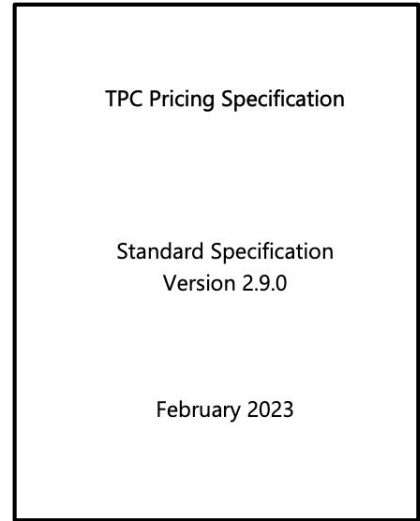
p28

[TPC-Pricing](#) is mandatory for audited results and defines the TCO as:

- 3-year software license
- 3-year hardware / cloud service
- 3-year maintenance (enterprise-grade support)

Problems:

- enterprise support has strict requirements (24/7, 4h response)
- doesn't work well with cloud/serverless setups



68 pages


TPC-Pricing: Total Cost of Ownership

p29

[TPC-Pricing](#) is mandatory for audited results and defines the TCO as:

- 3-year software license
- 3-year hardware / cloud service

TPC Pricing Specification

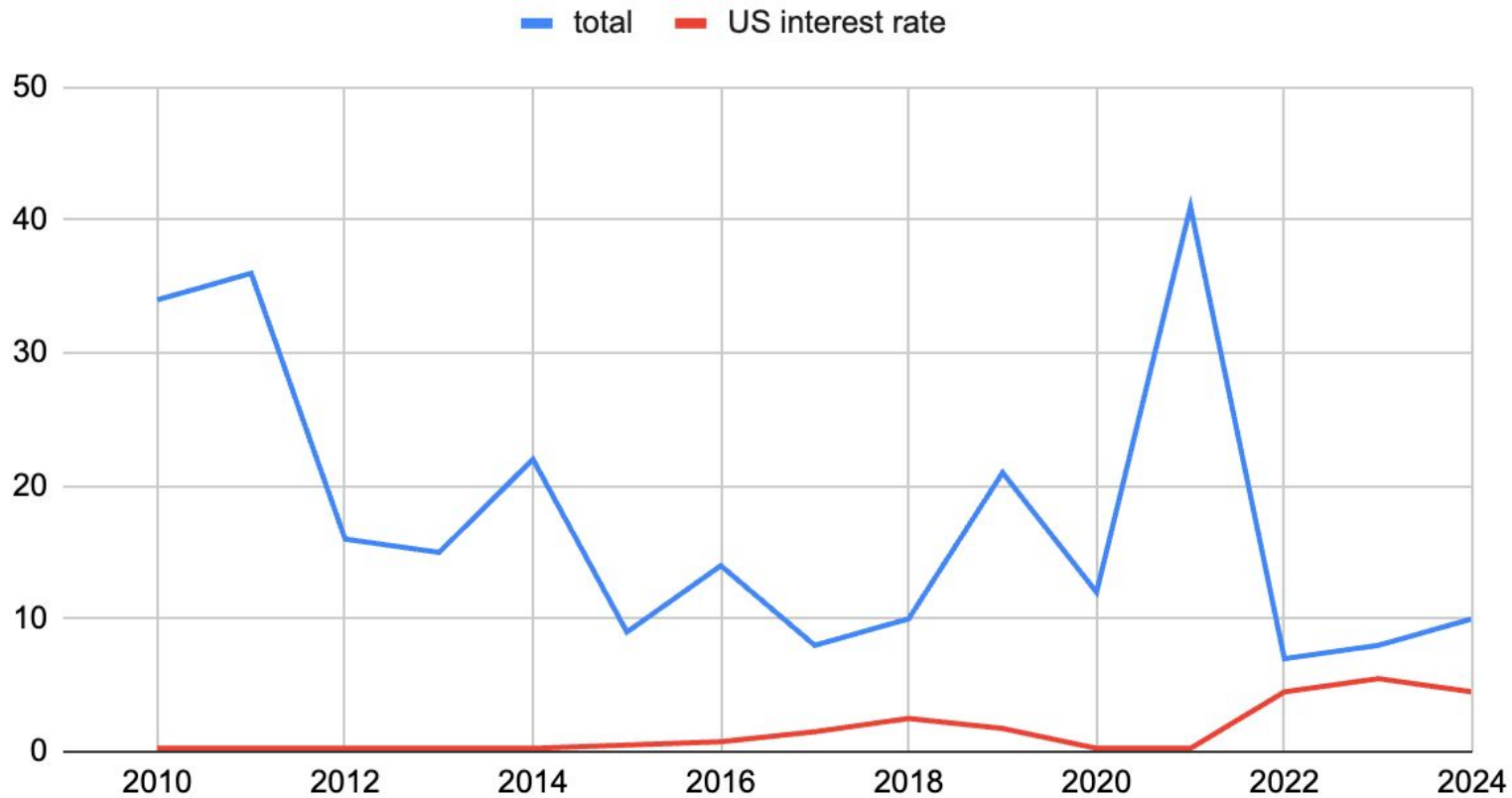
 databricks	Databricks SQL 8.3		TPC-DS: 3.2.0 TPC-Pricing: 2.7.0 Report Date: 2021-11-02	
Total System Cost	TPC-DS Throughput	Price/Performance	System Availability Date	
\$5,190,345 USD	32,941,245 QphDS@100000GB	\$157.57 USD/kQphDS@100000GB	As of Publication	
Dataset Size	Database Manager	Operating System	Other Software	Cluster
100,000 GB	Databricks PhotonEngine 8.3	Linux	N/A	Yes

Specification
2.9.0

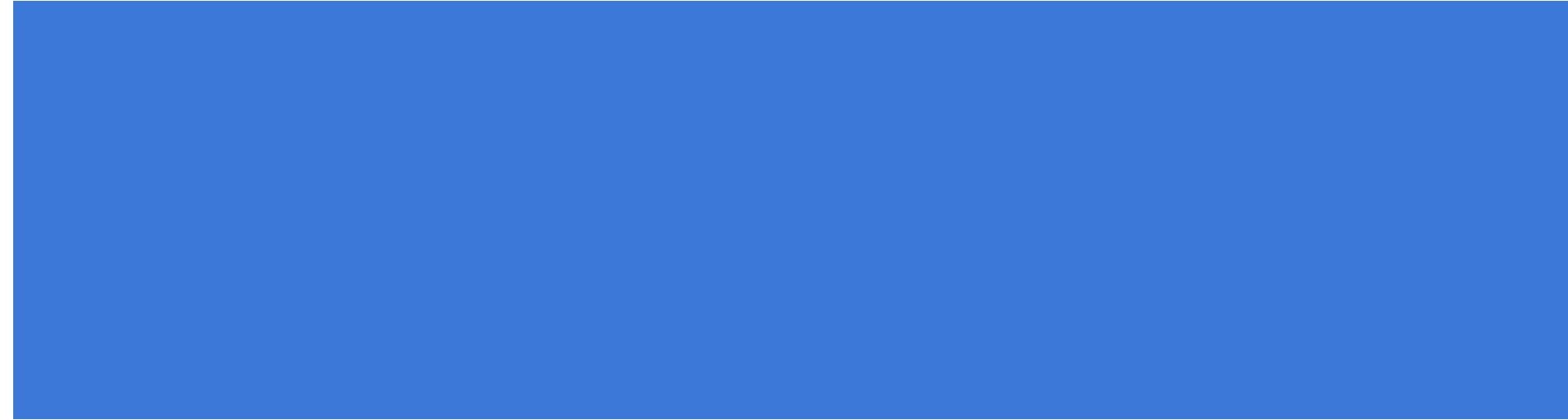
2023

Pages

number of audits <> Fed funds interest rate



3. LDBC overview





Linked Data Benchmark Council

Non-profit founded in 2013

Registered in the UK

Mission: Accelerate progress in graph data management by facilitating procompetitive work

The need for (graph) DB benchmarks

p33

Graph databases were very immature in 2011:

- performance issues
- no standard query language
- no common understanding of what they should be able to do



8



6



2

2



Predictable Labs, Inc.

LDBC's benchmark design

p35

LDBC takes inspiration from TPC:

- complex database benchmarks
- datasets of different scale factors
- benchmarks are designed using choke points (TPC-H choke points++)
- stringent auditing process
- TPC-Pricing for reporting total cost of ownership

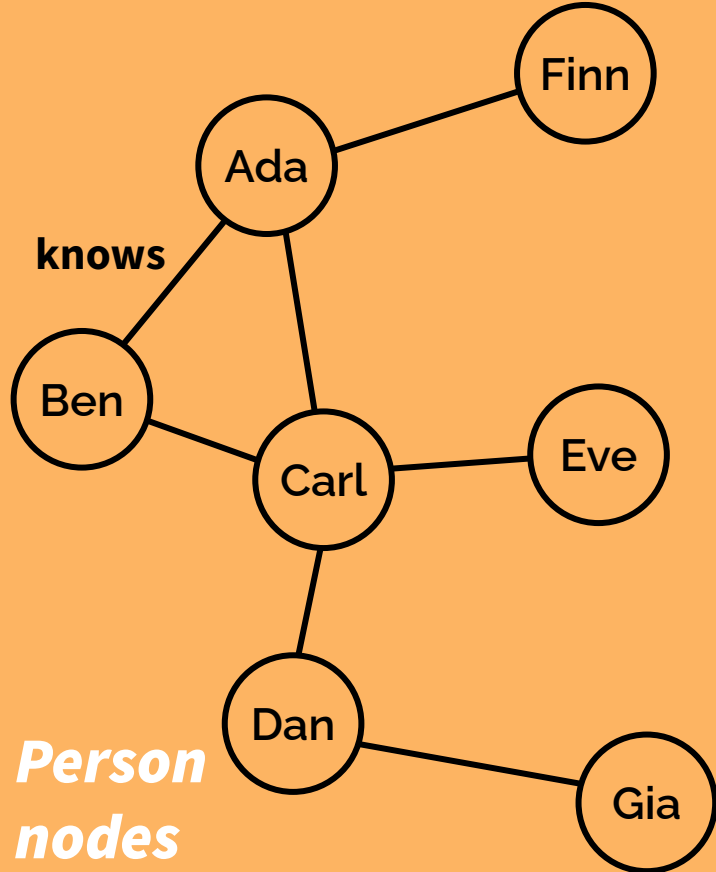
LDDB Social Network Benchmark

- **Dataset, queries, updates**
- Workload mix
- Auditing

Dataset

Queries

Updates



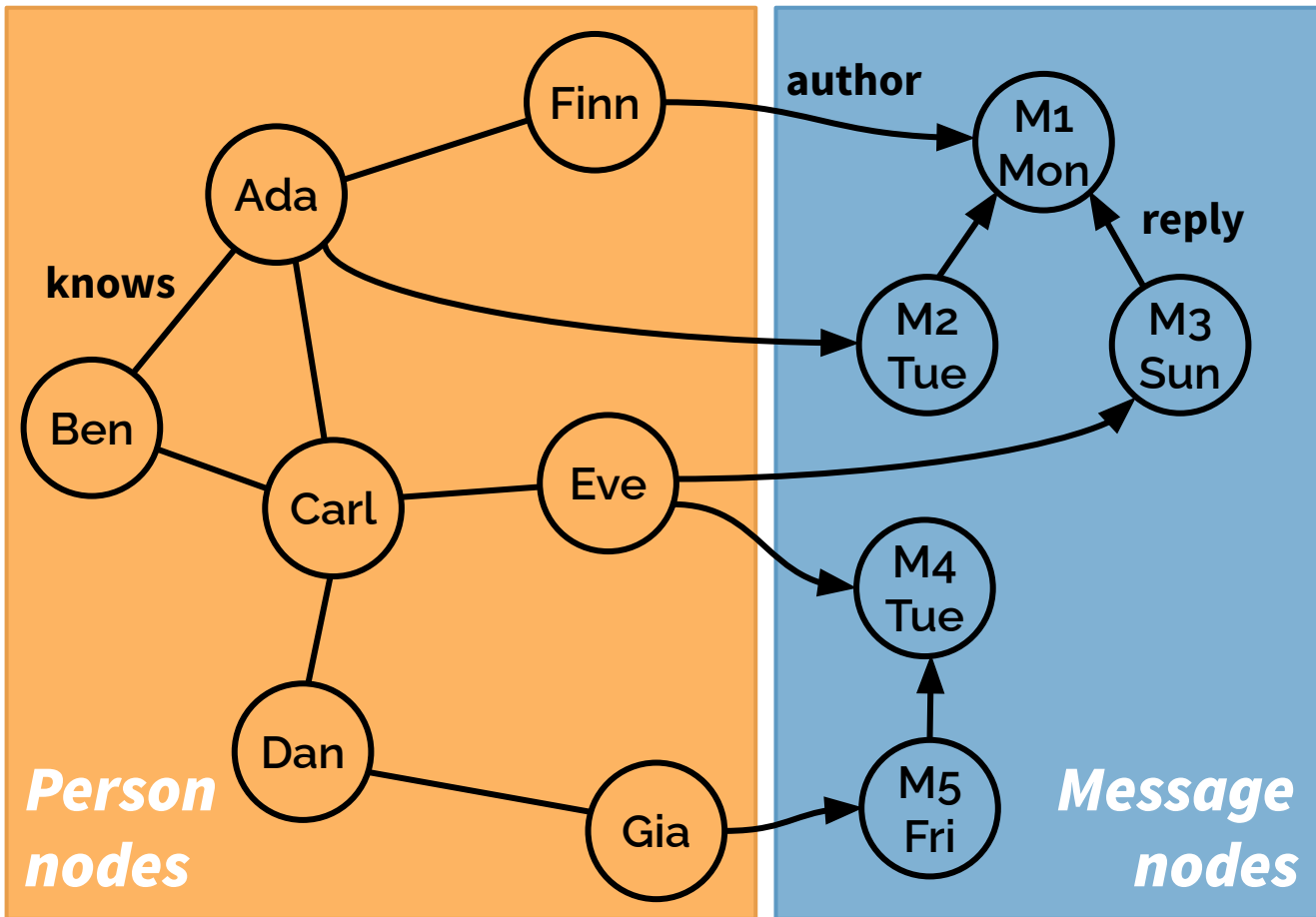
The Person-knows-Person graph follows the degree distribution described in paper Ugander et al., [The anatomy of the Facebook social graph](#) (2011)

Dataset

Queries

Updates

p39



Person nodes

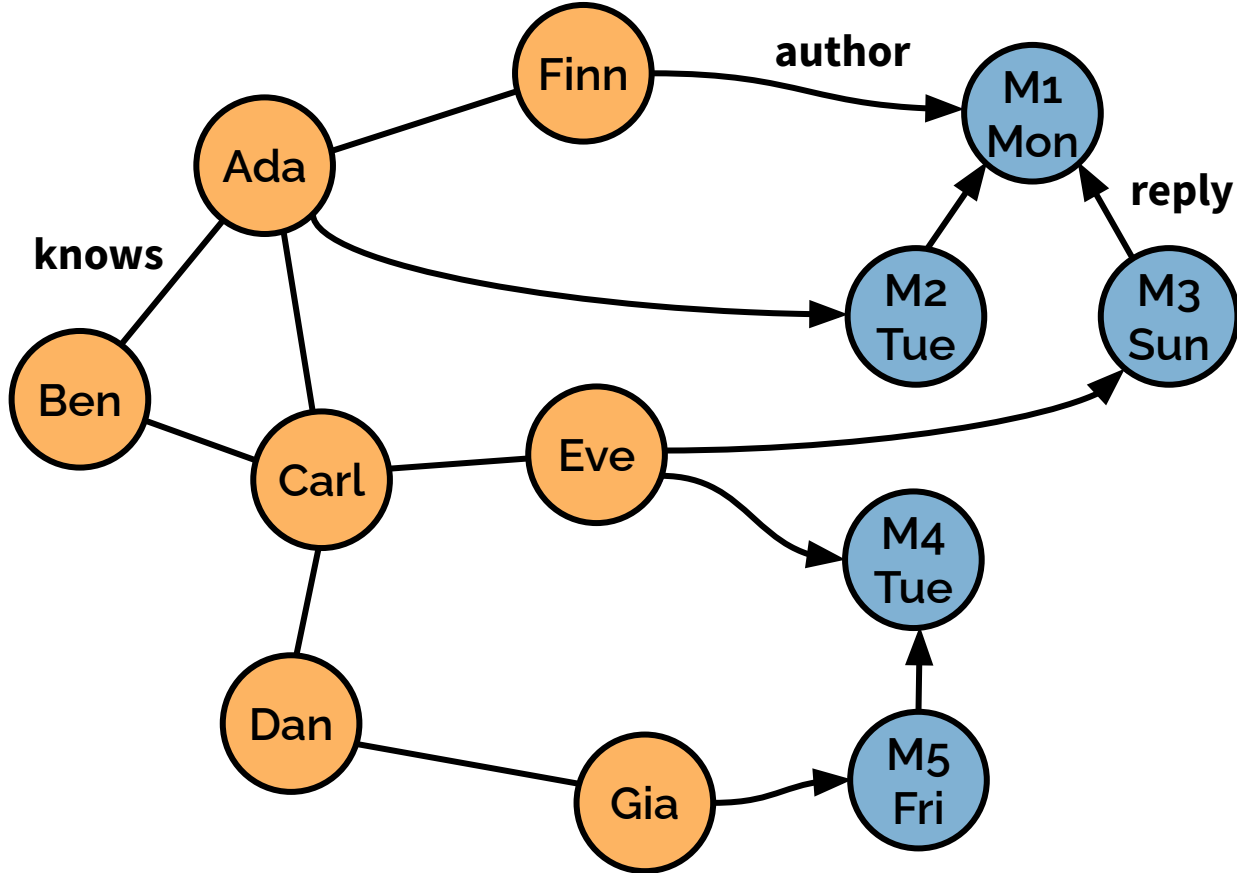
Message nodes

Dataset

Queries

Updates

p40

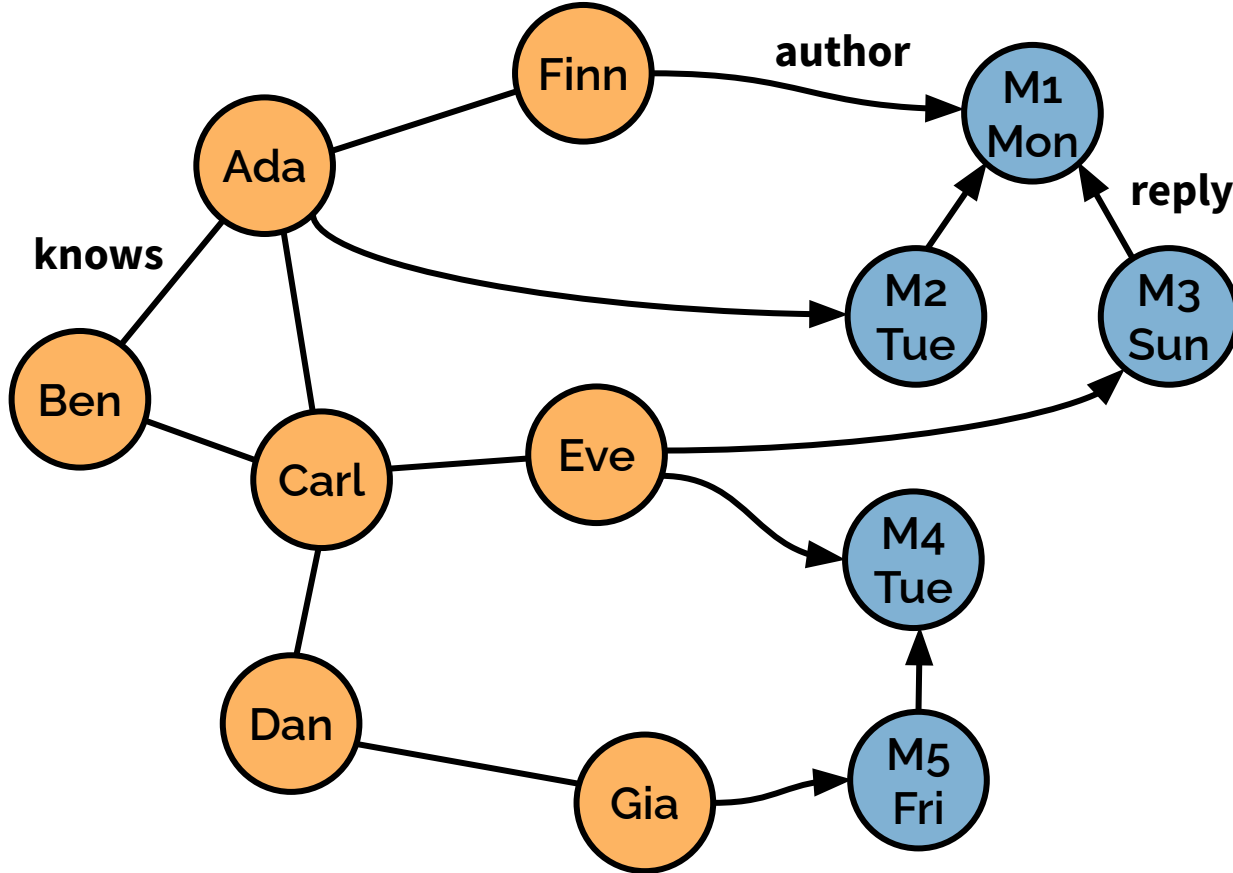


Dataset

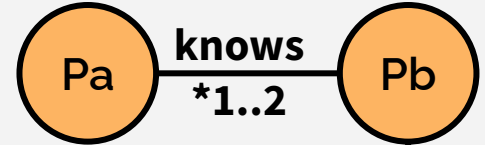
Queries

Updates

p41



$Q(\$name, \$day)$



$name = \$name$

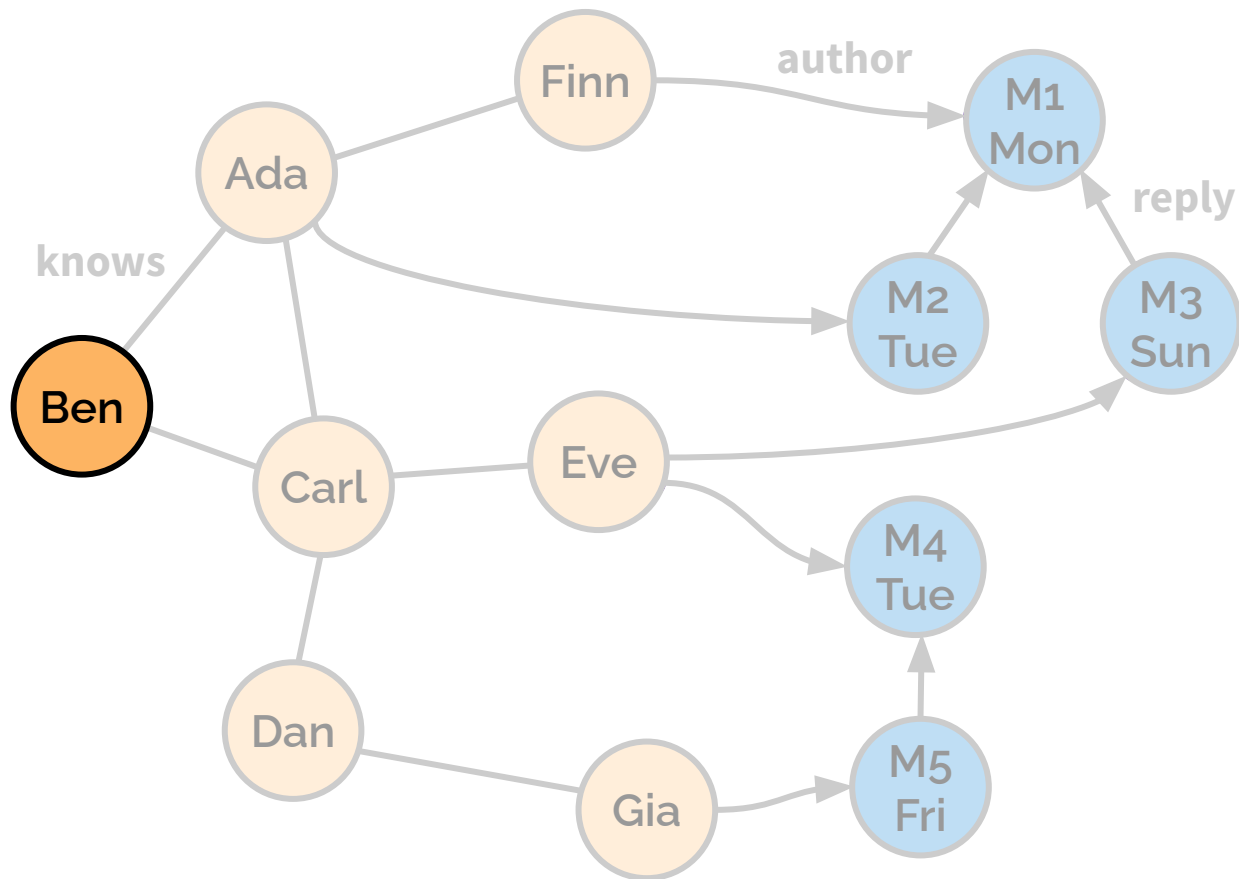
$creation\ date < \$day$

Dataset

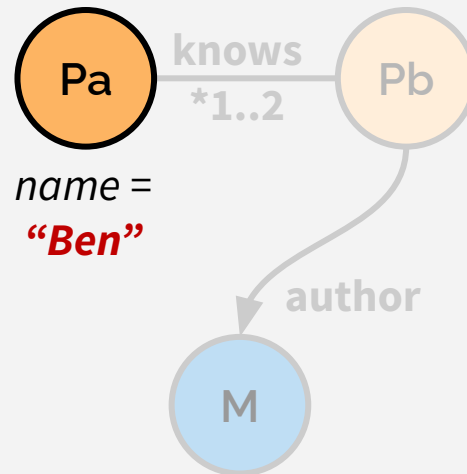
Queries

Updates

p42



$Q(\text{"Ben"}, \text{"Sat"})$



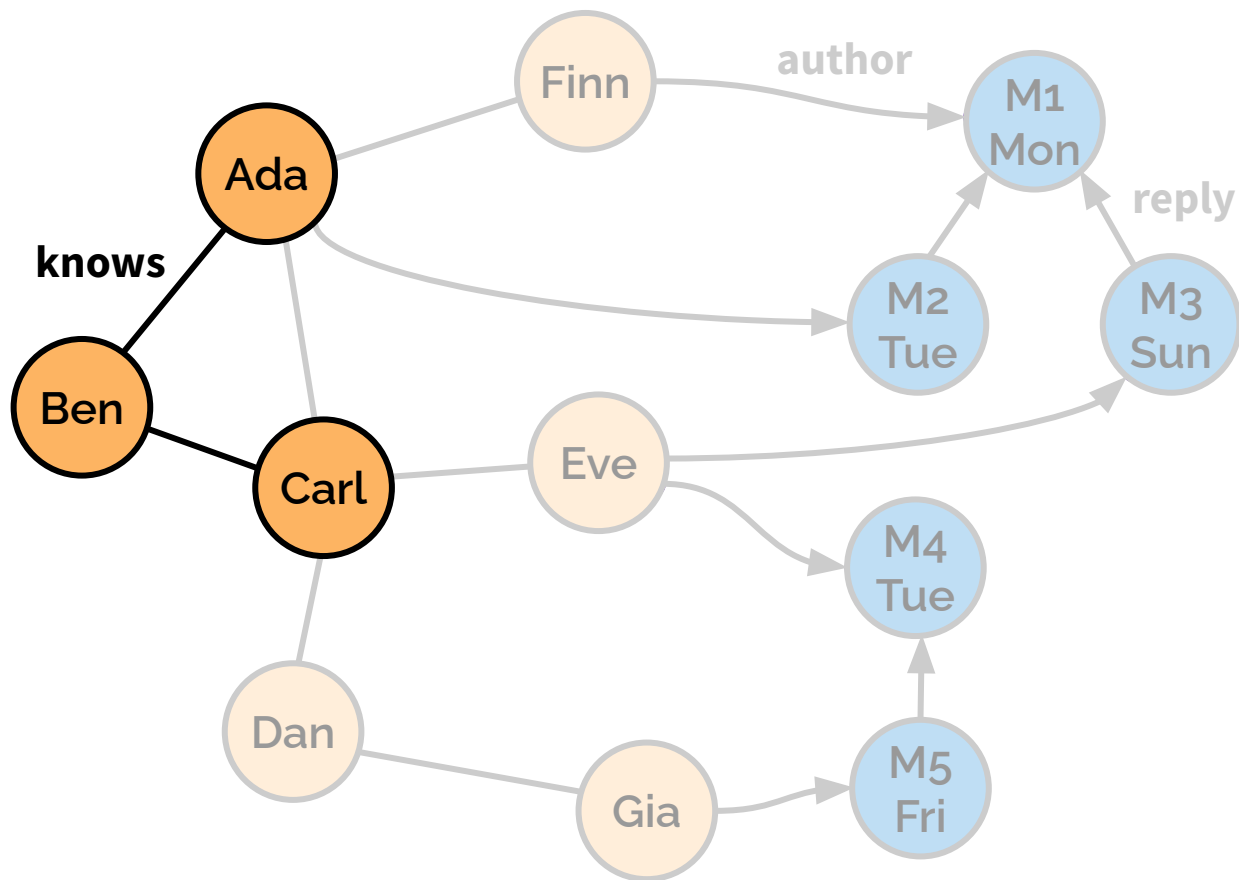
$creation\ date < \text{"Sat"}$

Dataset

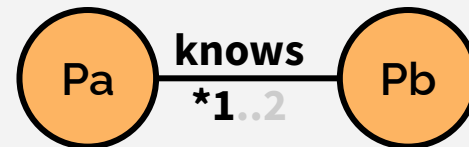
Queries

Updates

p43



$Q(\text{"Ben"}, \text{"Sat"})$



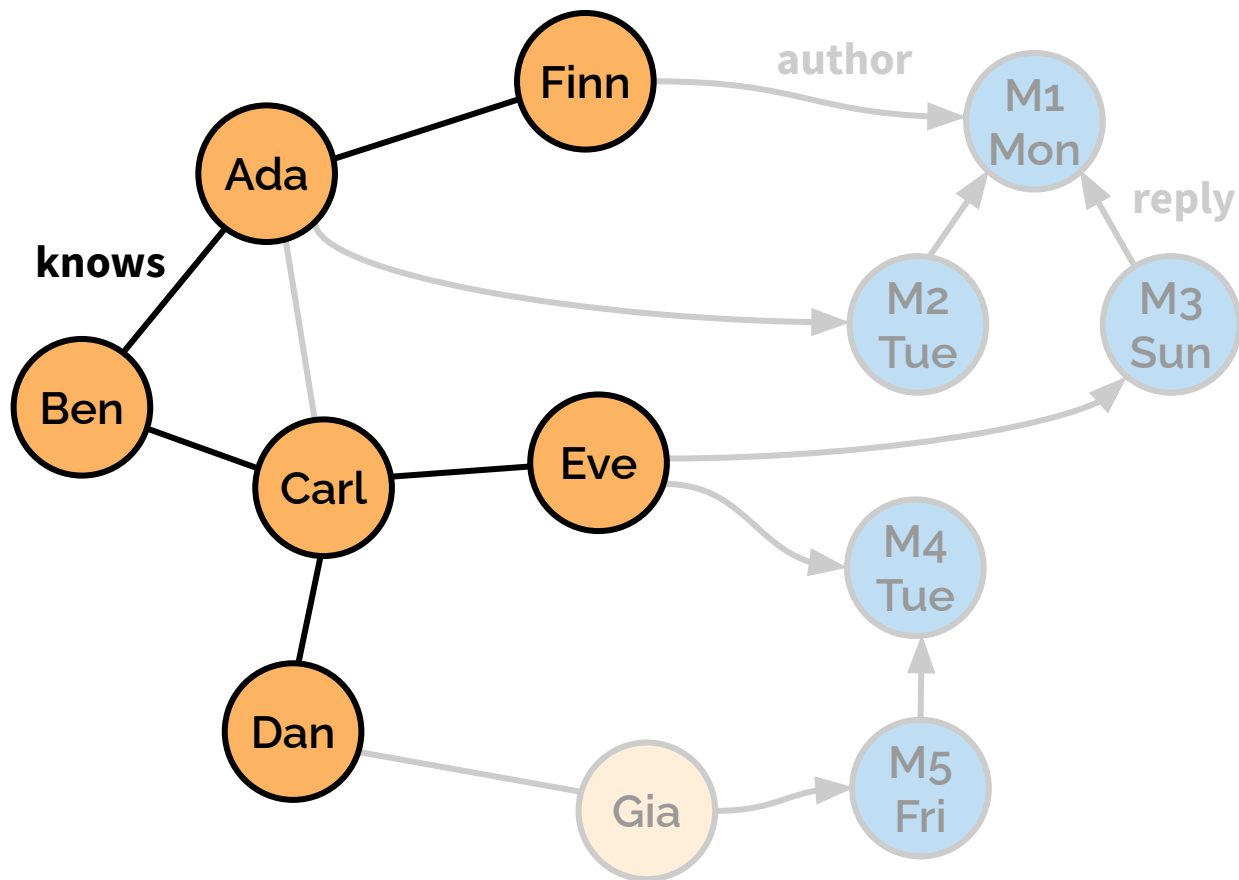
creation date < "Sat"

Dataset

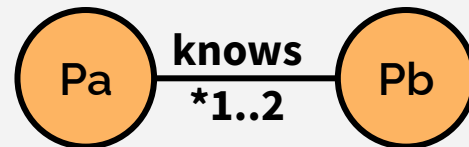
Queries

Updates

p44



Q("Ben", "Sat")



name =
"Ben"

author

M

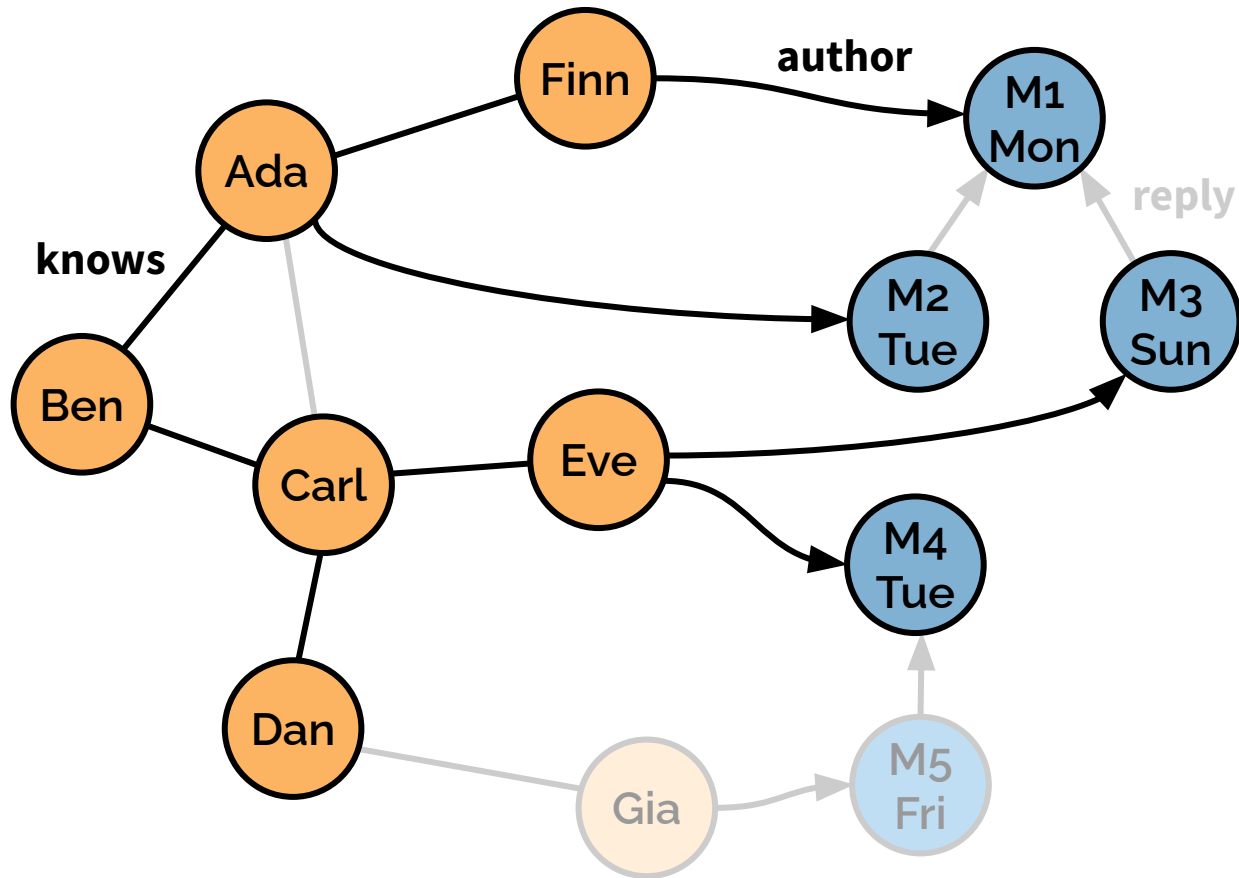
creation date < "Sat"

Dataset

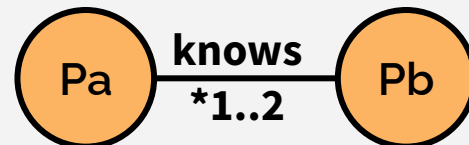
Queries

Updates

p45



Q("Ben", "Sat")

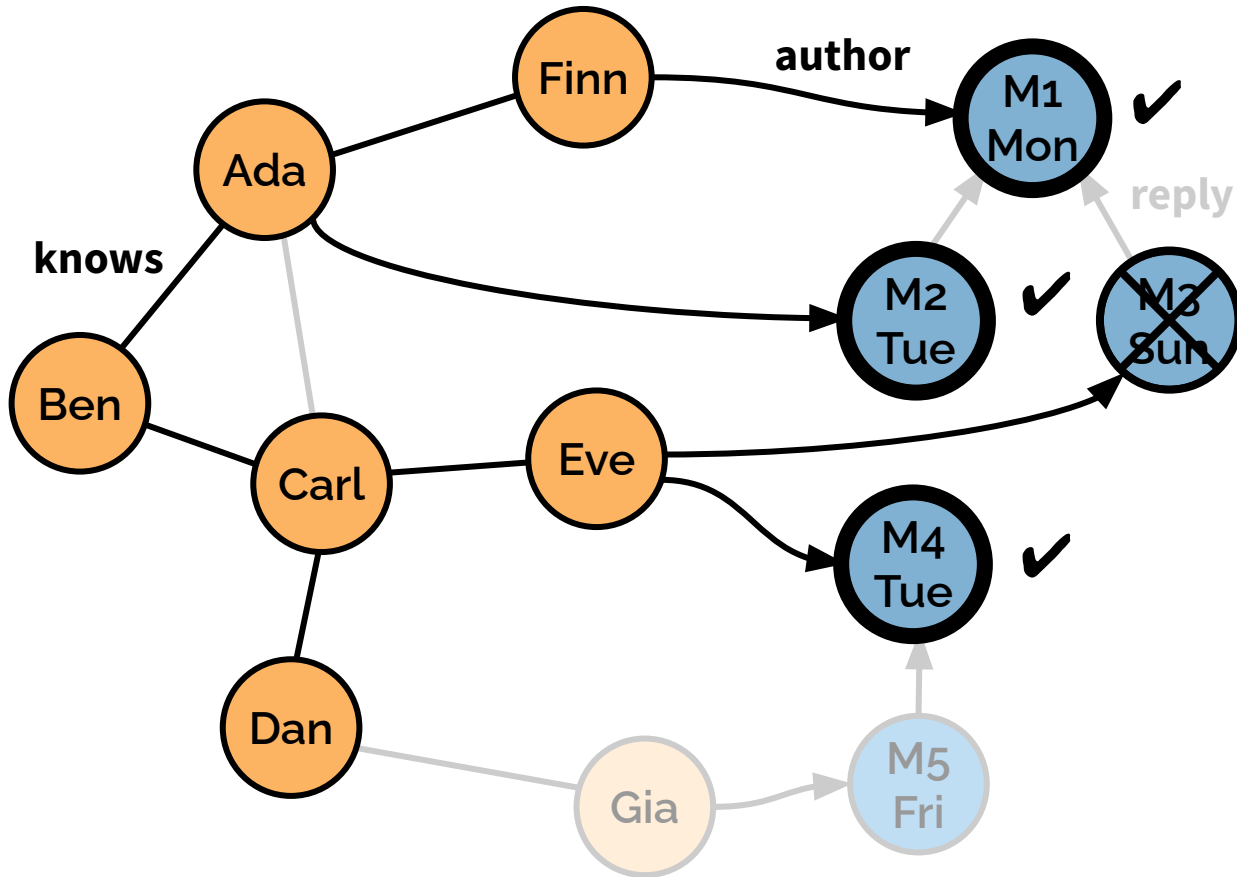


Dataset

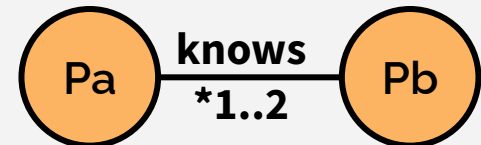
Queries

Updates

p46

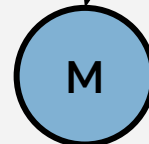


Q("Ben", "Sat")



name =
"Ben"

author



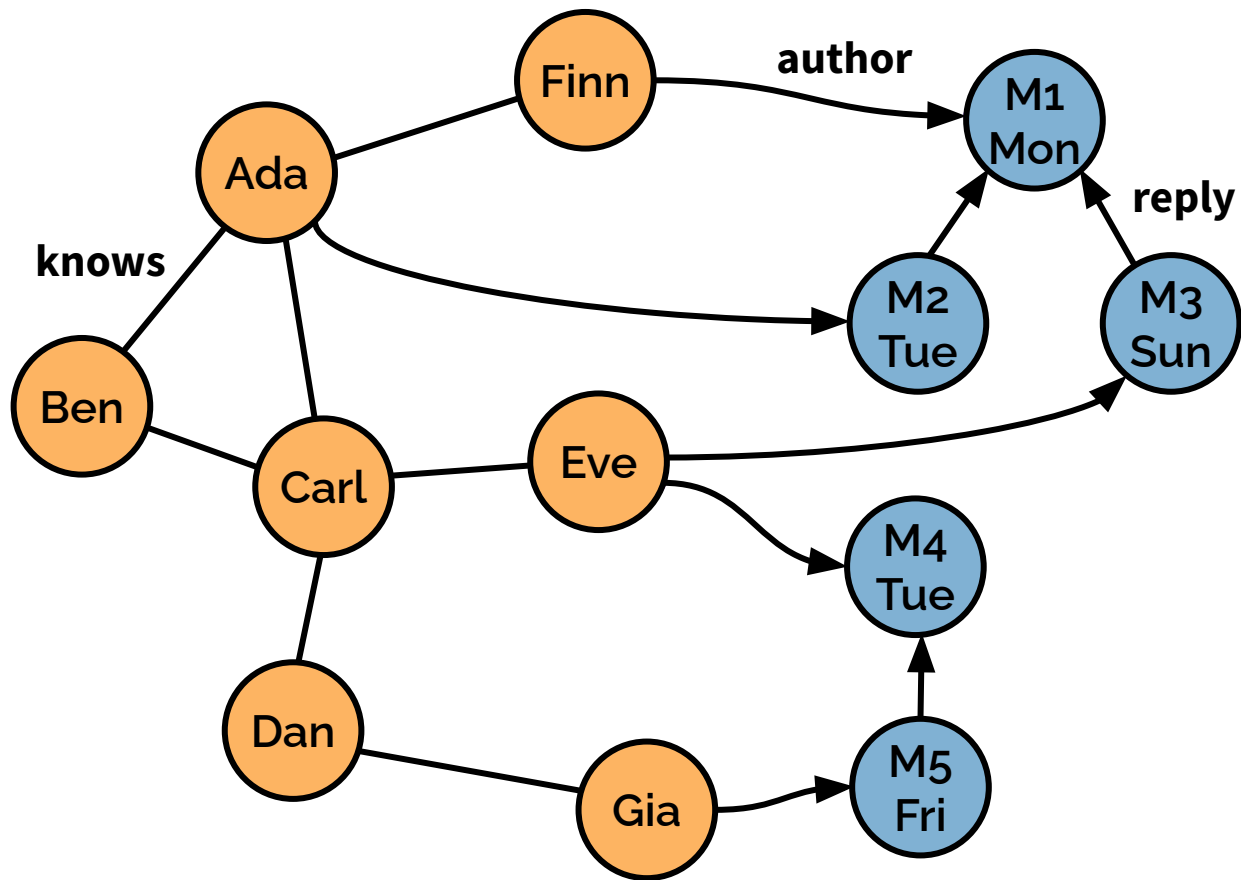
creation date < "Sat"

Dataset

Queries

Updates

p47

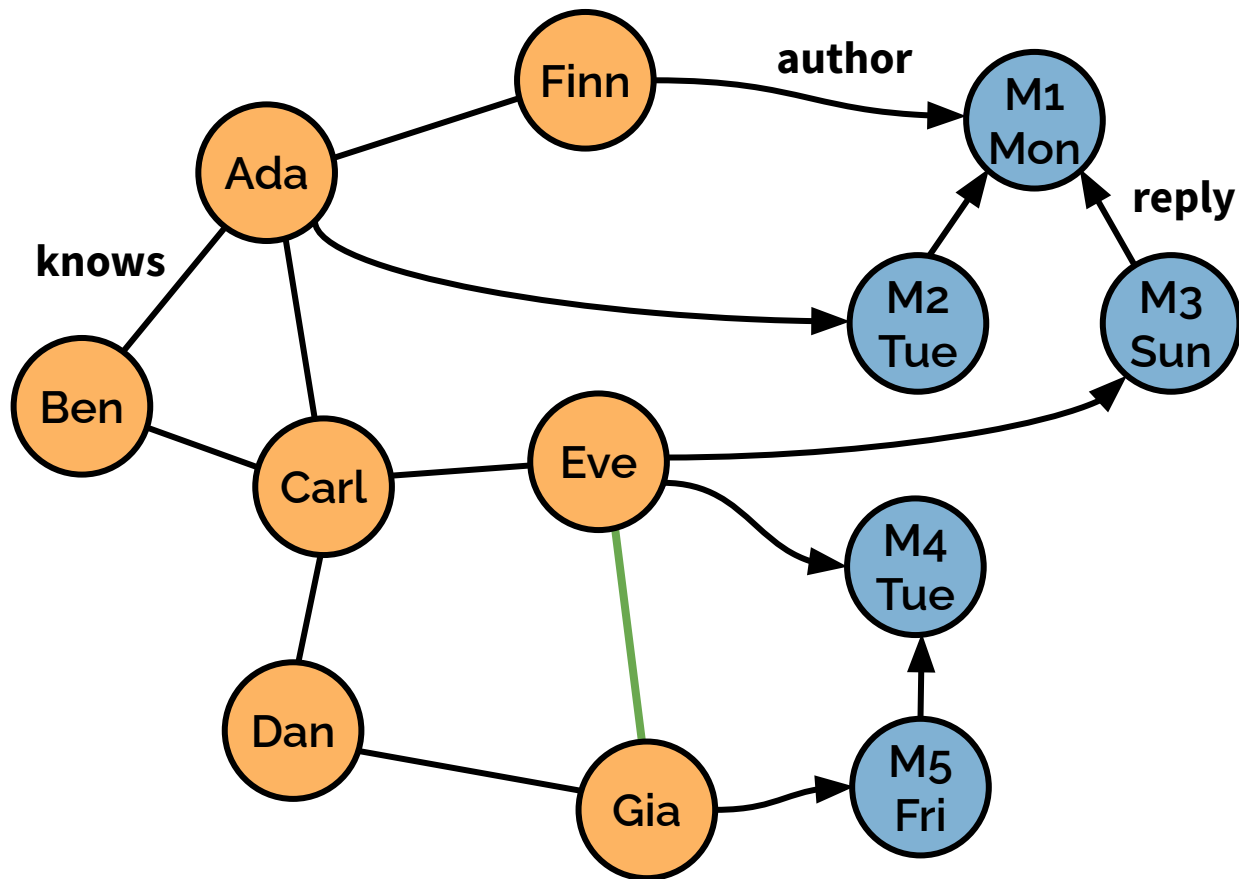


Dataset

Queries

Updates

p48



Updates

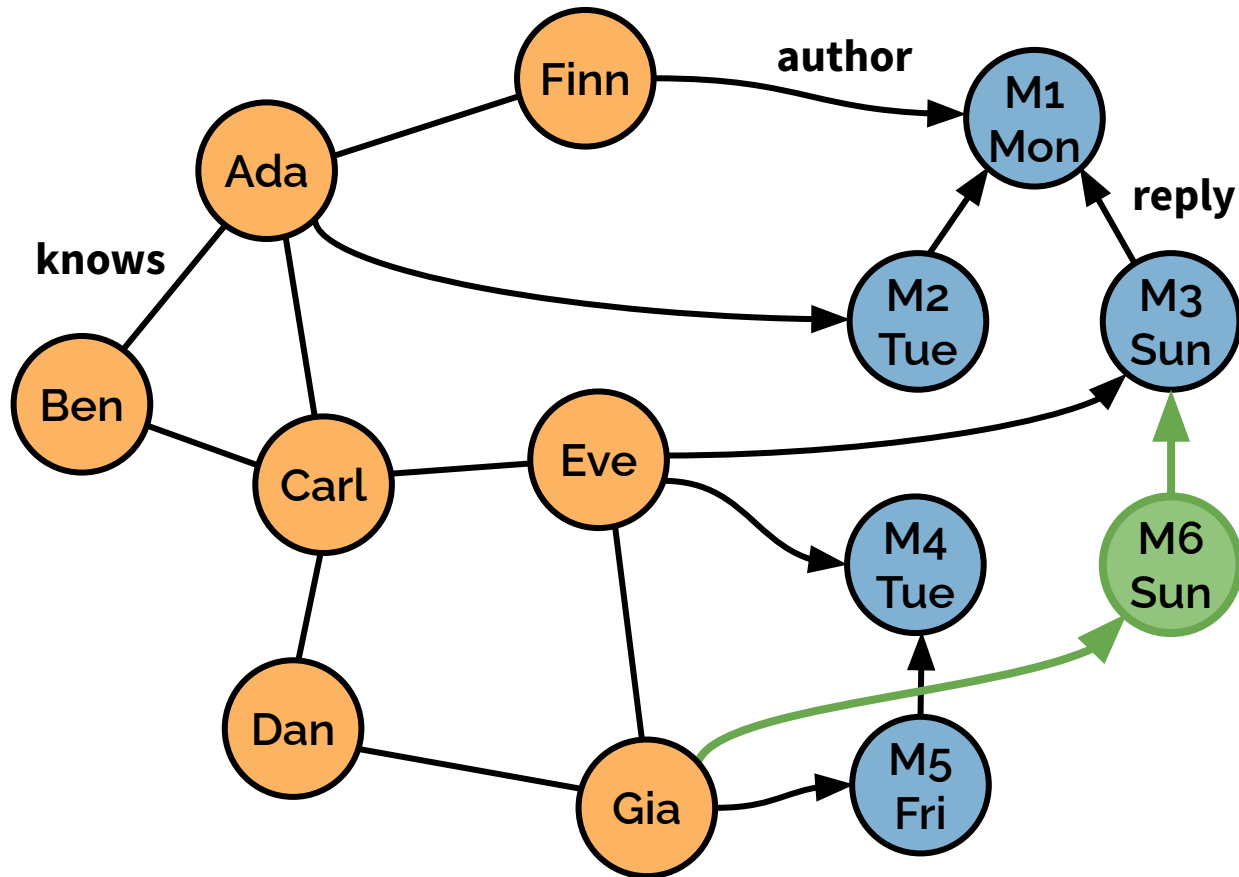
+ knows("Eve", "Gia")

Dataset

Queries

Updates

p49



Updates

+ knows("Eve", "Gia")

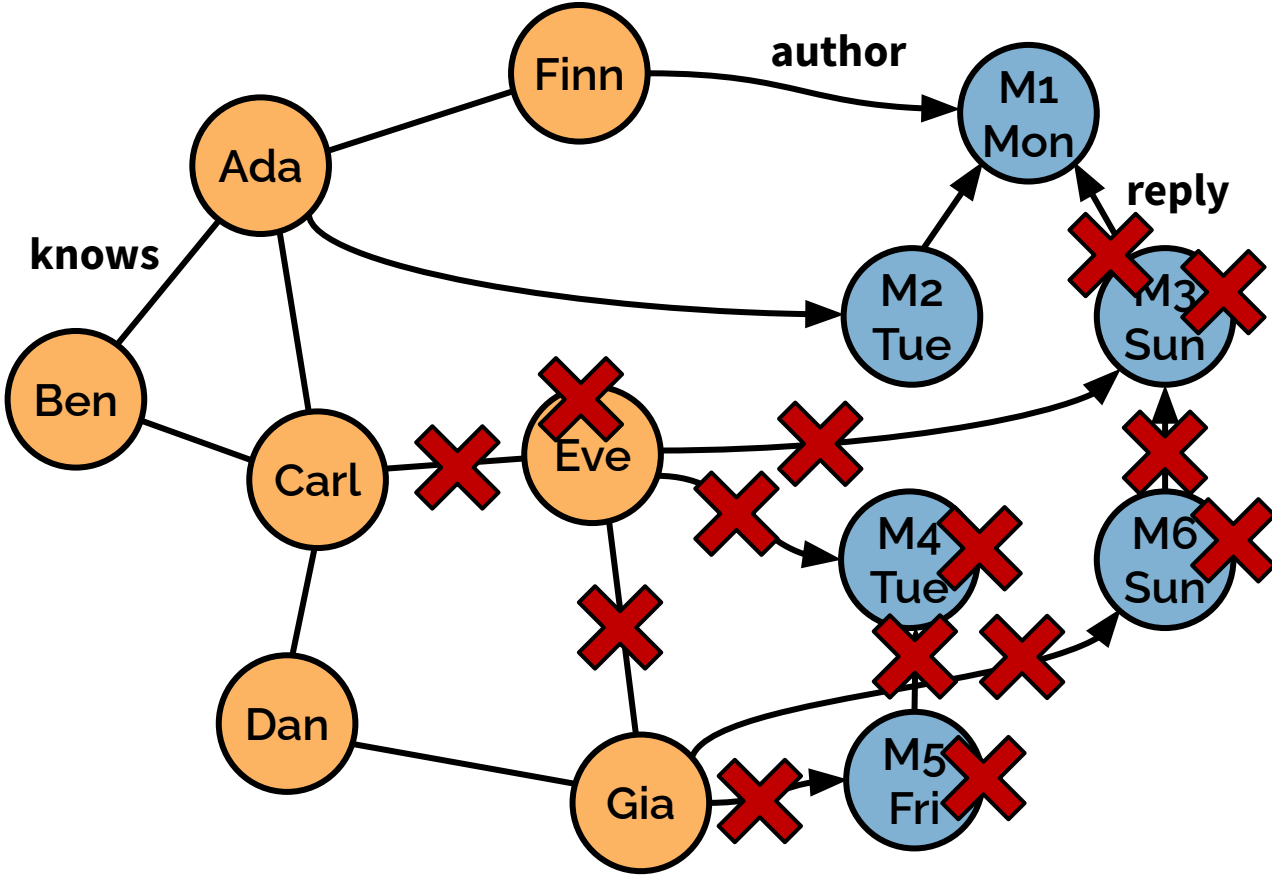
+ Message("Gia", "M3")

Dataset

Queries

Updates

p50



Updates

+ knows("Eve", "Gia")

+ Message("Gia", "M3")

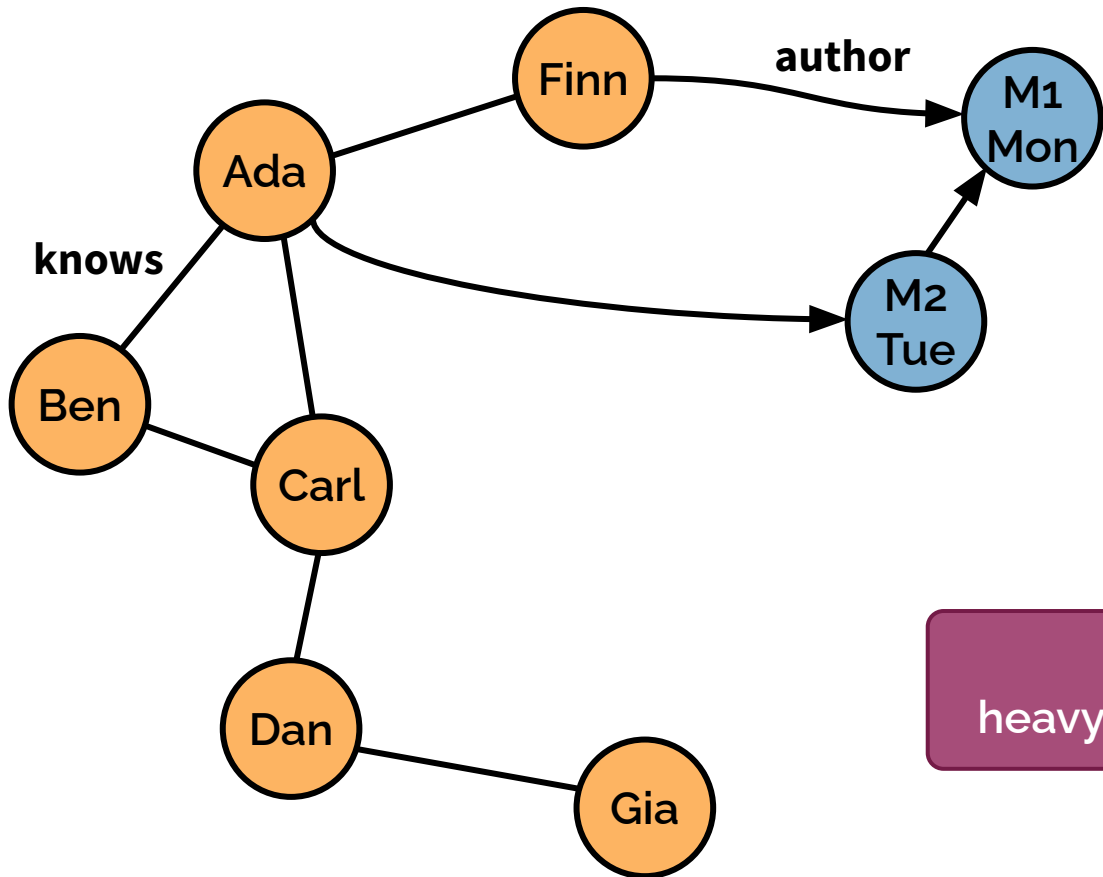
- Person("Eve")

Dataset

Queries

Updates

p51



Updates

+ knows("Eve", "Gia")

+ Message("Gia", "M3")

- Person("Eve")

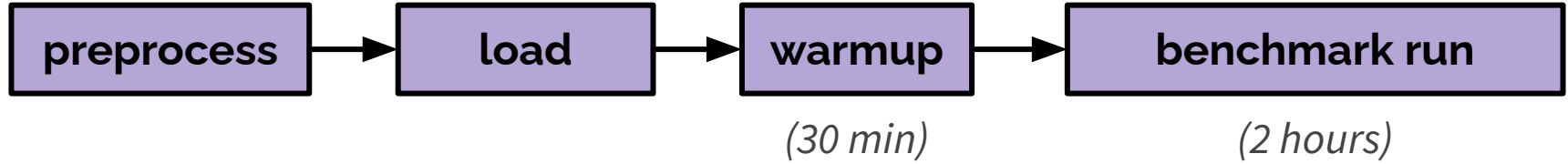
Deletes are heavy-hitting operations!

LDDB Social Network Benchmark

- Dataset, queries, updates
- **Workload mix**
- Auditing

Transactional benchmark workload

p53



Workload mix

p54

CR
complex read
1–500 ms

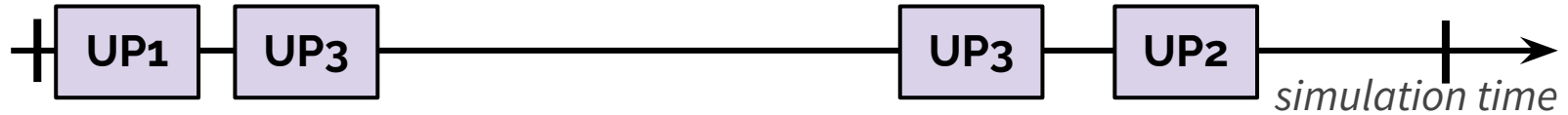
SR
short read
0.1–75 ms

UP
update
0.1–100 ms

Scheduling operations: Example

p55

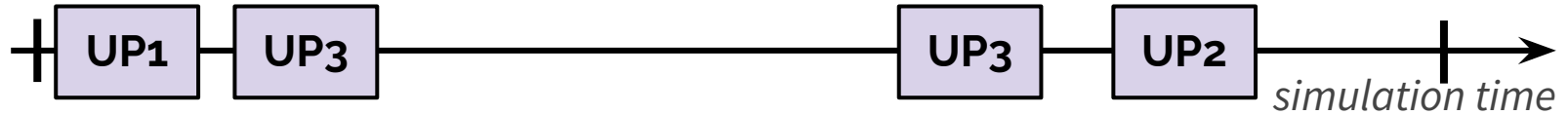
Replay is determined by the TCR (total compression ratio) and data dependencies



Scheduling operations: Example

p56

Replay is determined by the TCR (total compression ratio) and data dependencies



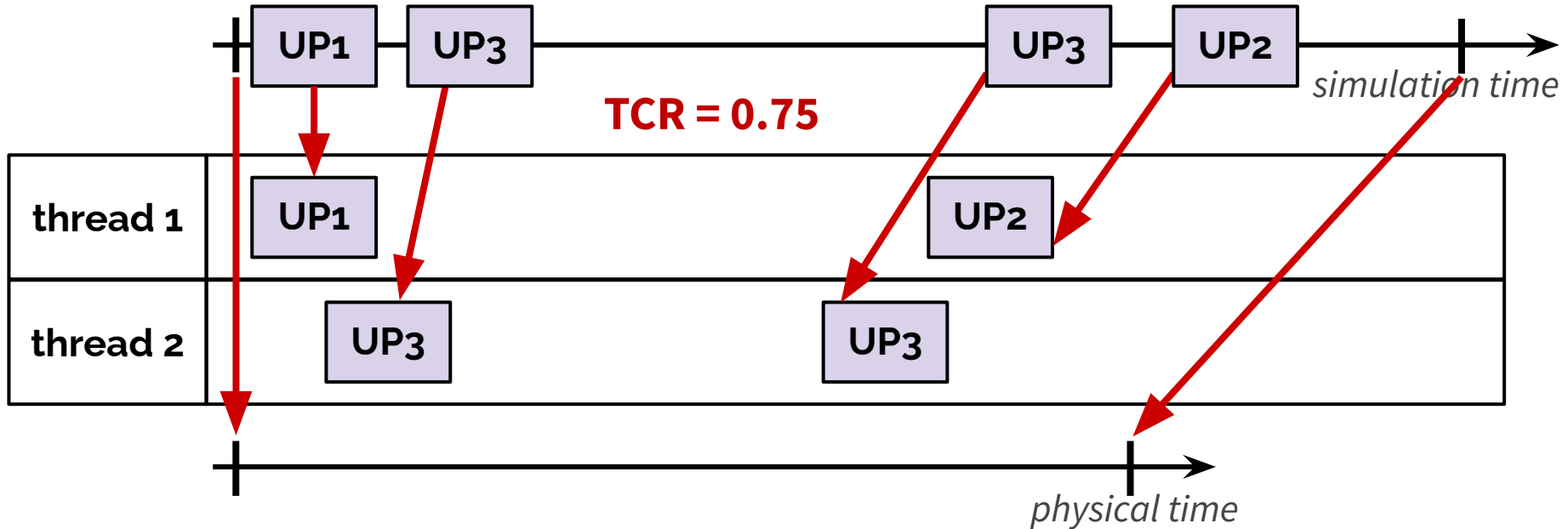
thread 1	
thread 2	



Scheduling operations: Example

p57

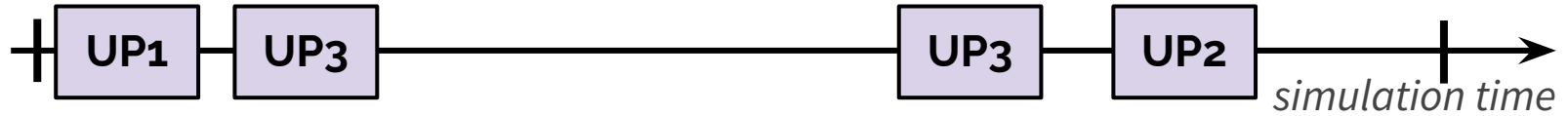
Replay is determined by the TCR (total compression ratio) and data dependencies



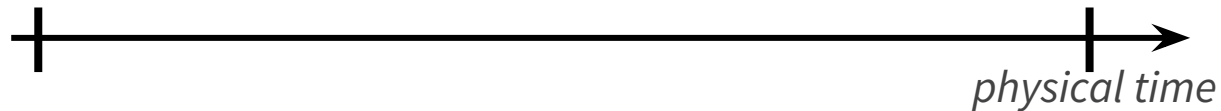
Scheduling operations: Example

p58

Replay is determined by the TCR (total compression ratio) and data dependencies

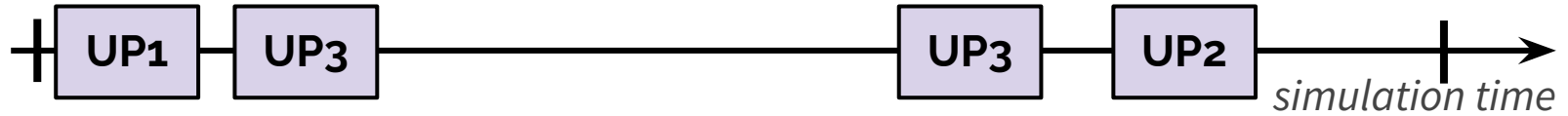


thread 1	UP1	CR		UP2	CR	
thread 2		UP3	CR		UP3	CR

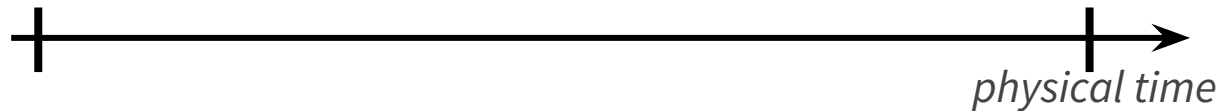


Scheduling operations: Example

Replay is determined by the TCR (total compression ratio) and data dependencies



thread 1	UP1	CR → SR → SR	UP2	CR → SR
thread 2	UP3	CR → SR	UP3	CR → SR → SR

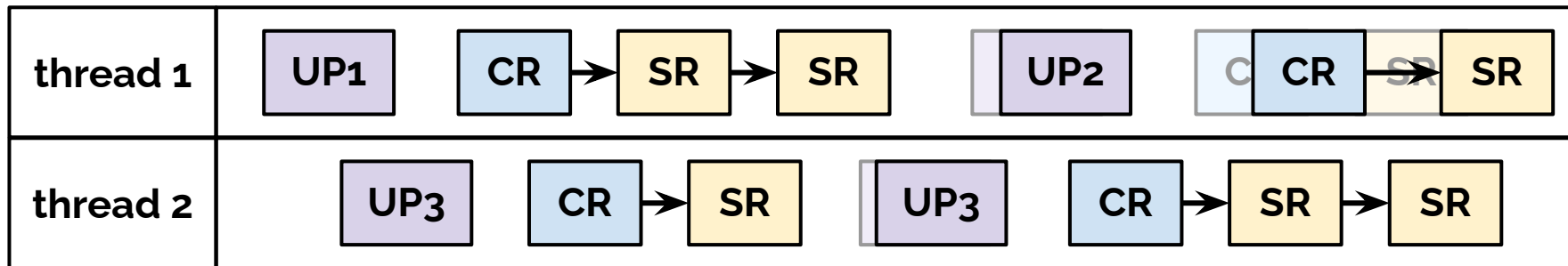


95% on-time requirement

p60

In order to pass an audit, 95% of the executed queries must meet the following condition:

actual start time – scheduled start time < 1 second



LDBC Social Network Benchmark

- Dataset, queries, updates
- Workload mix
- **Auditing**

Benchmark audits

Strict auditing guidelines

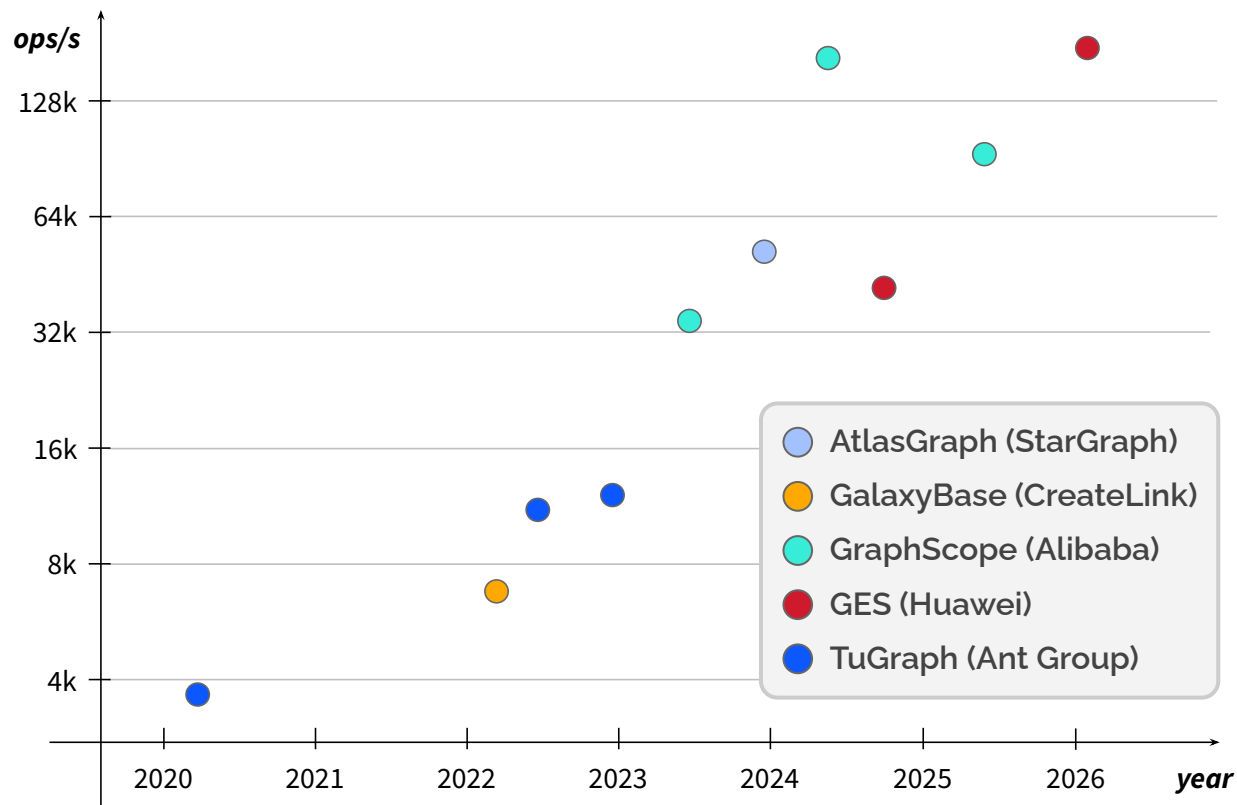
Performed by certified auditors

Audited results are used in RFPs (Request for Proposals)

DATE	SYSTEM	SF	TEST SPONSOR	PERFORMANCE	PERF/\$
2025-12-01	Graph Engine Service	1000	Huawei Cloud	125,872.54	286.60
2024-05-14	GraphScope Flex	1000	Alibaba Cloud	127,784.51	1,251.22

SNB Transactional workload, SF300

p63



~30× speedup in 5y
almost 2×/year

Issues encountered during auditing

Performance variability:

- Limited space on SSDs (<20%) causes significant performance reduction
- The performance – especially IO – of cloud VMs can be a hit and miss
- There was a major performance drop on certain VMs in the second benchmark run

Operationally:

- Misaligned expectations by the test sponsor about the work and responsibilities of the auditor. Some vendors come without having properly read the specification [...]. Sometimes they've missed implementing parts of the benchmark [...].
- **Audits always take MUCH longer than expected, usually 3-4 months.** Time differences do not help.

Questionnaire for test sponsors

p65

Questionnaire for LDBC SNB test sponsors

We created the following questionnaire to make the SNB auditing process more streamlined. If you need any clarification, please reach out to the SNB task force.

Checklist

- The SUT has a complete implementation of the benchmark which complies with the LDBC specification.
- The implementation uses a stable version of an LDBC SNB driver.
- The implementation passes cross-validation against one of the existing reference implementations on at least the SF10 data set.
- If you are not the vendor of the DBMS used in the SUT: do you have written permission of the vendor of the DBMS?

System

Overview

- Vendor name
- System name
- System version

High-level technical information

- System type (e.g. relational DBMS, graph DBMS)
- Storage type (in-memory/disk-based)
- Main implementation language of the system (e.g. C++)
- Query language(s) supported by the system (e.g. Cypher, Gremlin)
- Query language(s) used for the audited implementation (e.g. Cypher)
- Query execution strategy (interpreted/vectorized/compiled/etc.)
- Is a distributed version available (regardless of whether it is used in the audit)?
- If a distributed version is available, what sharding strategy does it use?

Licensing

- Product license (e.g. proprietary license, Apache Software License v2)
- License of the Java/Python client libraries (used in the LDBC SNB Interactive/BI drivers, respectively)

Database features

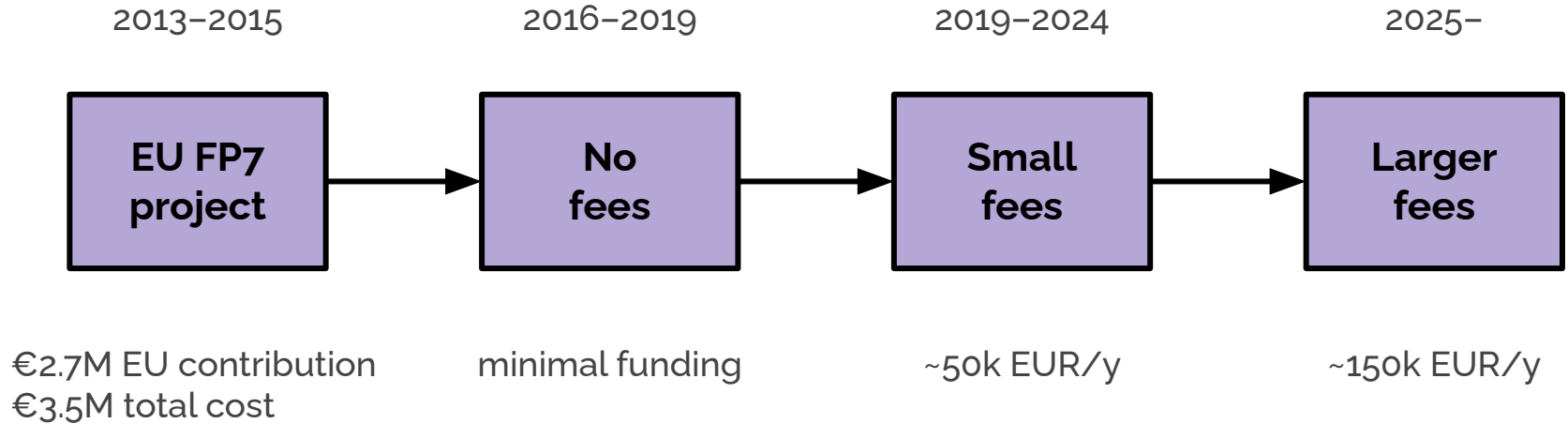
- Link to documentation
- Are stored procedures supported?
- If stored procedures are supported, what language(s) can they be implemented in?
- What is the maximum isolation level for transactions?

LDBC's organizational evolution

- Financing
 - Banking
 - Value proposition
 - Rebranding
-

Financing

p67



Banking



kicked out



urgent and extensive KYC check
triggered by unusual structure



Banking



kicked out

urgent and extensive KYC check
triggered by unusual structure

application denied

we passed the check & restructured

cannot accept money from X



Value proposition

Why would an organization join LDBC?

- **To have a seat at the table** when benchmarks and query languages are discussed
- **Networking** with other members
- **Defensive reasons:** to prevent the modification to the benchmark specifications that could put them at a competitive disadvantage

Multiple members indicated that LDBC should “**deemphasize the benchmarks and focus on datasets and synthetic data generators instead**”



Henry Gabb, [2025 GDC Member Survey Report](#) (2025)

LDBC has rebranded as GDC

p71



BENCHMARKS ▾

QUERY LANGUAGES & DATA MODELS ▾

RESOURCES ▾

ORGANIZATION ▾

BLOG

EVENTS

Graph Data Council

The Graph Data Council (GDC), formerly known as the Linked Data Benchmark Council (LDBC), is a non-profit organization that defines standard graph benchmarks and fosters a community for graph processing technologies.

[READ MORE →](#)



4. Popular benchmarks



ClickBench

—

ClickBench

p74

A macrobenchmark for analytical DBMSs

Designed and maintained by ClickHouse

Dataset:

- Single wide table, ~100M rows, ~75 GB in CSV
- Anonymized Yandex Metrika dataset (web analytics)

```
CREATE TABLE hits
(
    WatchID BIGINT,
    JavaEnable SMALLINT,
    Title TEXT,
    GoodEvent SMALLINT,
    EventTime TIMESTAMP,
    EventDate DATE,
    ...
    URL TEXT,
    Referer TEXT,
    ...
    -- 105 attributes in total
);
```

ClickBench queries

p75

43 queries focusing on:

- scan
- aggregation
- lookup

```
-- Q01  
SELECT COUNT(*) FROM hits;
```

```
-- Q02  
SELECT COUNT(*)  
FROM hits  
WHERE AdvEngineID <> 0;
```

```
-- Q11  
SELECT MobilePhoneModel, COUNT(DISTINCT UserID) AS u  
FROM hits  
WHERE MobilePhoneModel <> ''  
GROUP BY MobilePhoneModel  
ORDER BY u DESC  
LIMIT 10;
```

ClickBench's philosophy

Strives for simplicity:

- No scale factors, no query parameters
- Simple types (integers, strings, date, timestamp)
- Few restrictions on the implementation

Fast-paced:

- `git clone ... && cd ... && ./benchmark.sh`
- fast systems finish in <20 min end-to-end
- open a PR with the results
- results are for “latest” (see [GitHub issue](#))

ClickBench entries

p77

Systems:

- 75+ implementations: dataframe libraries, databases
- 125+ configurations: native/Parquet/partitioned, single node/distributed

Hardware:

- somewhat standardized
- common setups: c6a.4xlarge, c6a.metal, c8g.4xlarge, c7a.metal
- but you can bring your own

ClickBench — a Benchmark For Analytical DBMS



Methodology | [Reproduce and Validate the Results](#) | [Add a System](#) | [Hardware Benchmark](#) | [Versions Benchmark](#) | [See also: JSONBench](#)

System: [All](#) [AlloyDB](#) [AlloyDB \(tuned\)](#) [Apache Doris](#) [Apache Doris \(Parquet, partitioned\)](#) [Arc](#) [Athena \(partitioned\)](#) [Athena \(single\)](#) [Aurora for MySQL](#) [Aurora for PostgreSQL](#) [Bigquery](#) [ByConity](#) [ByteHouse](#) [CedarDB](#) [chDB](#) [chDB \(DataFrame\)](#) [chDB \(Parquet, partitioned\)](#) [CHYT](#) [Citrus](#) [ClickHouse](#) [ClickHouse \(data lake, partitioned\)](#) [ClickHouse \(data lake, single\)](#) [ClickHouse \(Parquet, partitioned\)](#) [ClickHouse \(Parquet, single\)](#) [ClickHouse \(TCHouse-C\)](#) [ClickHouse \(tuned, memory\)](#) [ClickHouse \(web\)](#) [ClickHouse \(aws\)](#) [ClickHouse \(azure\)](#) [ClickHouse \(gcp\)](#) [Cloudberry](#) [CockroachDB](#) [CrateDB](#) [CrateDB \(tuned\)](#) [Crunchy Bridge \(Parquet\)](#) [Daft \(Parquet, partitioned\)](#) [Daft \(Parquet, single\)](#) [Databend](#) [Databricks](#) [DataFusion \(Parquet, partitioned\)](#) [DataFusion \(Parquet, single\)](#) [DataFusion \(Vortex, partitioned\)](#) [DataFusion \(Vortex, single\)](#) [Drill](#) [Druid](#) [DuckDB](#) [DuckDB \(data lake, partitioned\)](#) [DuckDB \(data lake, single\)](#) [DuckDB \(DataFrame\)](#) [DuckDB \(memory\)](#) [DuckDB \(Parquet, partitioned\)](#) [DuckDB \(Parquet, single\)](#) [DuckDB \(Vortex, partitioned\)](#) [DuckDB \(Vortex, single\)](#) [Elasticsearch](#) [Elasticsearch \(tuned\)](#) [Firebolt](#) [Firebolt \(scan cache\)](#) [GlareDB \(Parquet, partitioned\)](#) [GlareDB \(Parquet, single\)](#) [Greenplum](#) [HeavyAI](#) [Hologres](#) [Hydra](#) [Infobright](#) [Kinetica](#) [MariaDB](#) [MariaDB ColumnStore](#) [MonetDB](#) [MongoDB](#) [MotherDuck](#) [MySQL](#) [MySQL \(MyISAM\)](#) [OctoSQL](#) [Opteryx](#) [Oxla](#) [Pandas \(DataFrame\)](#) [ParadeDB \(Parquet, partitioned\)](#) [ParadeDB \(Parquet, single\)](#) [Parseable \(Parquet, partitioned\)](#) [pg_clickhouse](#) [pg_duckdb](#) [pg_duckdb \(MotherDuck enabled\)](#) [pg_duckdb \(Parquet\)](#) [pg_duckdb \(with indexes\)](#) [pg_mooncake](#) [pgpro_tam](#) [pgpro_tam \(feather, local + cache\)](#) [pgpro_tam \(parquet, local + cache\)](#) [pgpro_tam \(parquet, local storage\)](#) [pgpro_tam \(parquet, local, parallel\)](#) [Pinot](#) [Polars \(DataFrame\)](#) [Polars \(Parquet\)](#) [PostgreSQL \(OrioleDB\)](#) [PostgreSQL \(with indexes\)](#) [QuestDB](#) [Redshift](#) [Sail \(Parquet, partitioned\)](#) [Sail \(Parquet\)](#) [Salesforce Hyper](#) [Salesforce Hyper \(Parquet\)](#) [SelectDB](#) [SigLens](#) [SingleStore](#) [Sirius](#) [Snowflake](#) [Spark](#) [Spark \(Auron\)](#) [Spark \(Comet\)](#) [Spark \(Gluten-on-Velox\)](#) [SQLite](#) [StarRocks](#) [Supabase](#) [Tablespace](#) [Tembo OLAP \(columnar\)](#) [TIDB \(TiFlash only\)](#) [TIDB \(TiKV only\)](#) [Timescale](#) [TimescaleDB](#) [TimescaleDB \(no columnstore\)](#) [Tinybird \(Free Trial\)](#) [Umbr](#) [Ursa](#) [VictoriaLogs](#) [YDB](#) [Yugabyte](#) [undefined](#)

Type: [All](#) [aws](#) [azure](#) [C](#) [C++](#) [ClickHouse derivative](#) [column-oriented](#) [dataframe](#) [document](#) [embedded](#) [gcp](#) [Go](#) [Java](#) [lukewarm-cold-run](#) [managed](#) [MySQL compatible](#) [PostgreSQL compatible](#) [Python](#) [row-oriented](#) [Rust](#) [search](#) [serverless](#) [Spark derivative](#) [stateless](#) [time-series](#)

Machine: [All](#) [c6a.4xlarge](#) [c6a.2xlarge](#) [c6a.metal](#) [c8g.4xlarge](#) [c6a.xlarge](#) [c7a.metal-48xl](#) [c6a.large](#) [c8g.metal-48xl](#) [t3a.small](#) [ClickHouse \(aws\): 12GiB](#) [ClickHouse \(aws\): 8GiB](#) [ClickHouse \(aws\): 16GiB](#) [ClickHouse \(aws\): 32GiB](#) [ClickHouse \(aws\): 64GiB](#) [ClickHouse \(aws\): 120GiB](#) [ClickHouse \(aws\): 236GiB](#) [ClickHouse \(aws\): 356GiB](#) [serverless](#) [c7i.metal-48xl](#) [CHYT: 12 vCPU 48GB](#) [Hologres: 16 CU](#) [pgpro_tam: 16 vCPU 32GB](#) [AlloyDB: 8 vCPU 64 GB](#) [Aurora: 16acu](#) [64 vCPU 256GB](#) [AlloyDB: 16 vCPU 128 GB](#) [ByteHouse: L](#) [ByteHouse: M](#) [ByteHouse: S](#) [ByteHouse: XS](#) [CHYT: 10 vCPU 40GB](#) [CrunchyBridge: Analytics-256GB](#) [Databricks: 2X-Large](#) [Databricks: 2X-Small](#) [Databricks: 4X-Large](#) [Databricks: Large](#) [Databricks: Medium](#) [Databricks: Small](#) [Databricks: X-Large](#) [Databricks: X-Small](#) [Hydra: XL](#) [lambda-GH200](#) [Motherduck: jumbo](#) [Motherduck: Jumbo](#) [Motherduck: mega](#) [Motherduck: pulse](#) [Motherduck: standard](#) [p5.4xlarge](#) [Redshift: dc2.8xlarge](#) [Redshift: ra3.4xlarge](#) [Redshift: ra3.16xlarge](#) [Redshift: ra3.xlplus](#) [SingleStore: S2](#) [SingleStore: S24](#) [Snowflake: 2XL](#) [Snowflake: 3XL](#) [Snowflake: 4XL](#) [Snowflake: L](#) [Snowflake: M](#) [Snowflake: S](#) [Snowflake: XL](#) [Snowflake: XS](#) [Supabase: 4XL](#) [Tablespace: L1 - 16CPU 32GB](#) [Timescale \(aws\): 4 vCPU 16GB](#) [Timescale \(aws\): 8 vCPU 32GB](#) [Timescale \(aws\): 16 vCPU 64GB](#) [undefined](#)

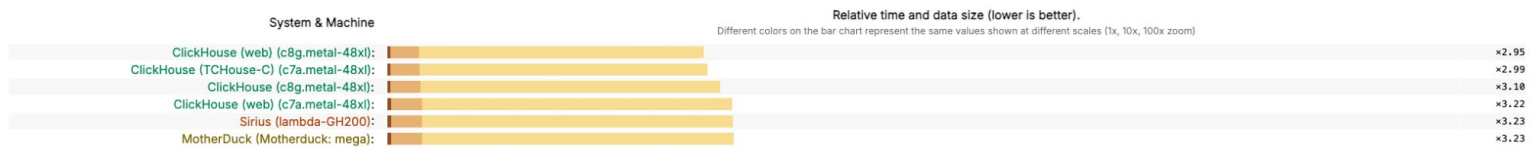
Cluster size: [All](#) [1](#) [2](#) [3](#) [4](#) [8](#) [9](#) [16](#) [32](#) [64](#) [128](#) [256](#) [serverless](#) [undefined](#)

Open source: [Yes](#) [No](#)

Hardware: [CPU](#) [GPU](#)

Tuned: [No](#) [Yes](#)

Metric: [Combined](#) [Cold Run](#) [Hot Run](#) [Load Time](#) [Storage Size](#)



Relative time and data size (lower is better).

System & Machine

Different colors on the bar chart represent the same values shown at different scales (1x, 10x, 100x zoom)

PostgreSQL (c6a.4xlarge):		×1.87
MySQL (c6a.4xlarge):		×1.96

Relative size (lower is better).

System & Machine

Different colors on the bar chart represent the same values shown at different scales (1x, 10x, 100x zoom)

hits.parquet:		13.76 GiB (×1.00)
hits.tsv.gz:		15.18 GiB (×1.10)
hits.csv.gz:		15.47 GiB (×1.12)
hits.json.gz:		22.10 GiB (×1.61)
hits.tsv:		69.67 GiB (×5.06)
hits.csv:		75.56 GiB (×5.49)
MySQL (c6a.4xlarge):		87.95 GiB (×6.39)
PostgreSQL (c6a.4xlarge):		99.18 GiB (×7.20)
hits.json:		216.75 GiB (×15.75)

Detailed Comparison

	PostgreSQL (c6a.4xlarge)	MySQL (c6a.4xlarge)
<input checked="" type="checkbox"/> Load time:	937s (×1.00)	10004s (×10.68)
<input checked="" type="checkbox"/> Data size:	99.18 GiB (×1.13)	87.95 GiB (×1.00)
<input checked="" type="checkbox"/> Q0.	261.381s (×1.00)	330.410s (×1.26)
<input checked="" type="checkbox"/> Q1.	261.279s (×1.00)	373.222s (×1.43)
<input checked="" type="checkbox"/> Q2.	261.309s (×1.00)	377.680s (×1.45)
<input checked="" type="checkbox"/> Q3.	261.231s (×1.00)	375.447s (×1.44)
<input checked="" type="checkbox"/> Q4.	275.527s (×1.00)	392.147s (×1.42)
<input checked="" type="checkbox"/> Q5.	281.111s (×1.00)	579.645s (×2.06)
<input checked="" type="checkbox"/> Q6.	261.142s (×1.00)	383.492s (×1.47)
<input checked="" type="checkbox"/> Q7.	261.146s (×1.00)	381.470s (×1.46)
<input checked="" type="checkbox"/> Q8.	287.942s (×1.00)	438.660s (×1.52)
<input checked="" type="checkbox"/> Q9.	289.754s (×1.00)	458.595s (×1.58)
<input checked="" type="checkbox"/> Q10.	261.894s (×1.00)	411.090s (×1.57)
<input checked="" type="checkbox"/> Q11.	262.253s (×1.00)	414.540s (×1.58)

H2O.ai DB benchmark



H2O.ai DB benchmark

p81

Originally created by Jan Gorecki funded by H2O.ai, [fork](#) maintained by DuckDB Labs

Task

groupby join

0.5 GB 5 GB 50 GB

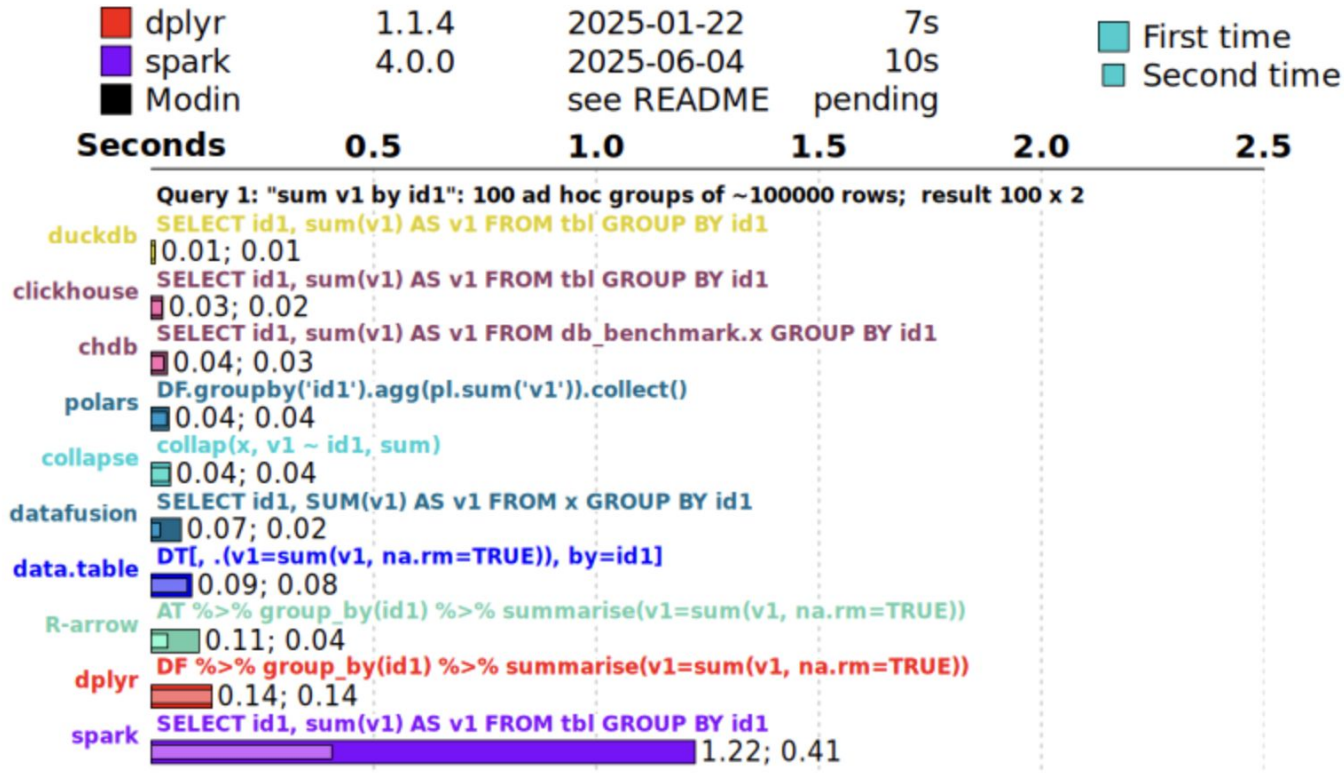
basic questions

Input table: 10000000 rows x 9 columns (0.5 GB)

Polars	1.34.0	2025-10-10	0s
Datafusion	50.0.0	2025-10-13	1s
DuckDB	1.4.1	2025-10-23	1s
collapse	2.1.3	2025-10-10	1s
ClickHouse	25.9.3.48	2025-10-10	1s

H2O.ai DB benchmark

p82



H2O.ai DB benchmark

p83

Data: synthetic

Aggregation workload:

- 5 basic grouping tests
- 5 advanced grouping tests
- Low/high cardinality, grouping on integer/string types

Join workload:

- 5 join queries

H2O.ai DB benchmark

Proposal: extend it with window functions

Dilemma: what do we do with it?

- Keep as is – becoming stale
- Extend – changes can be considered biased

So far: as is, maybe add larger scale factors and different machines

5. Takeaways



Benchmarks

- Carry tremendous value
- Capture a common understanding
- Drive innovation

They are:

- used extensively vendors
- difficult to finance

An "ideal benchmark"

- Aim for simplicity
- Single-node data generator that can scale up
- Simple driver/validation framework in Python

The benchmark should

- be implementable in a few days
- have a leaderboard
- have some approximate pricing specification
- and maybe have an auditing service too!

LDBC / GDC is open

If you would like to...

- learn more
- collaborate
- contribute
- audit your system

Join us by reaching out to

info@ldbouncil.org



Recommended reading for pt. 1

p89

D. H. Bailey, [*Twelve ways to fool the masses when giving performance results on parallel computers*](#) (1991)

J. Gray, [*The Benchmark Handbook*](#) (1993)

T. Hoefler, R. Belli, [*Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results*](#) (2012) [[talk](#), [recording](#)]

T. Hoefler, [*Benchmarking data science: Twelve ways to lie with statistics and performance on parallel computers*](#) (2021)

J. v. Kistowski et al., [*How to build a benchmark*](#) (2016) [[related talk](#)]

M. Raasveldt et al., [*Fair benchmarking considered difficult: Common pitfalls in DB performance testing*](#) (2018)

Recommended reading for pt. 2

M. Poess et al., [Why you should run TPC-DS: A workload analysis](#) (2007)

P. Boncz et al., [TPC-H analyzed: Hidden messages and lessons learned from an influential benchmark](#) (2013) [[talk](#)]

M. Dreseler et al., [Quantifying TPC-H choke points and their optimizations](#) (2020)

M. Poess, [New initiatives in the TPC](#) (2022)

M. Poess, [TPC, where art thou?](#) (2023)

Recommended reading for pt. 3

p91

P. Boncz et al., [*The Linked Data Benchmark Council project*](#) (2013)

O. Erling et al., [*The LDBC Social Network Benchmark: Interactive workload*](#) (2015)

G. Szárnyas et al., [*The LDBC Social Network Benchmark: Business Intelligence workload*](#) (2022)

G. Szárnyas et al., [*The LDBC benchmark suite*](#) (FOSDEM 2023)

G. Szárnyas et al., [*The Linked Data Benchmark Council \(LDBC\): Driving competition and collaboration in the graph data management space*](#) (2023)

Ω