Pushing PostgreSQL to the Limits

Tackling OLAP workloads with Extensions

Shivji Kumar Jha (Shiv)
with substantial help from Mehboob Alam

Safe Harbour Statement

The views, opinions, and conclusions presented in these slides are solely my own and do not reflect the official stance, policies, or perspectives of my employer or any affiliated organization. Any statements made are based on my personal experiences and research and should not be interpreted as official guidance or endorsement.

Contents

Speaker => Shivji Jha(Shiv)

- Areas of Interest
  - Databases & Streaming
  - Distributed Storage Infrastructure
  - Application Architectures
- Passion for OSS DB and Community
  - Contributed to MySQL and Apache Pulsar
  - Mentor reports on OSS NATS, Druid, ClickHouse
  - 25+ talks at conferences & meetups
  - Co-organizer, Postgres Bangalore Meetups (pgblr.in), no 5 on April 26, 2025 (Saturday)

linkedin.com/in/shivjijha/

github.com/shiv4289/shiv-tech-talks/

1. Background

Postgres and its Neighborhood

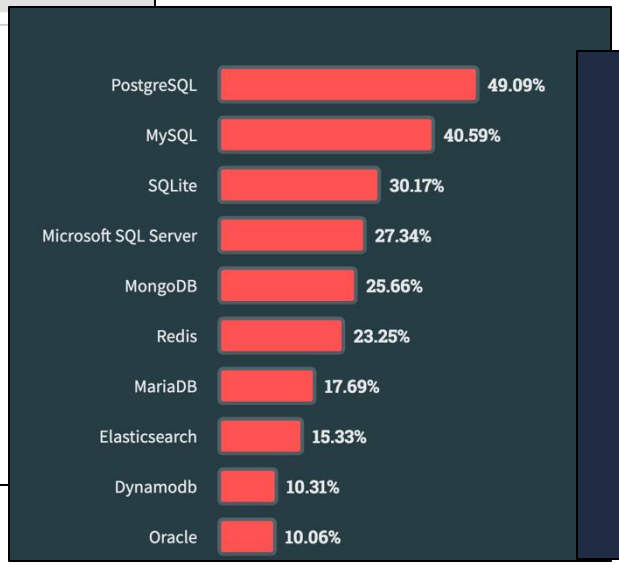
https://db-engines.com/en/ranking

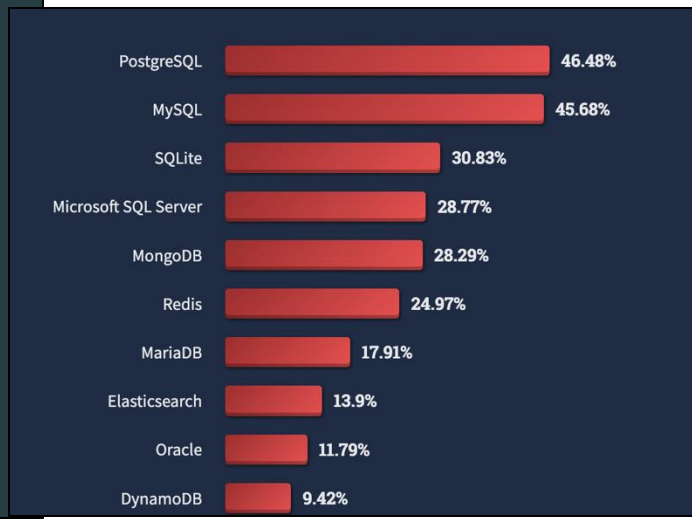
https://survey.stackoverflow.co/

(last 3 years)

en.wikipedia.org/wiki/DB-Engines_ranking

- 2017 - PostgreSQL
- 2018 - PostgreSQL
- 2019 - MySQL
- 2020 - PostgreSQL
- 2021 - Snowflake
- 2022 - Snowflake
- 2023 - PostgreSQL
- 2024 - Snowflake



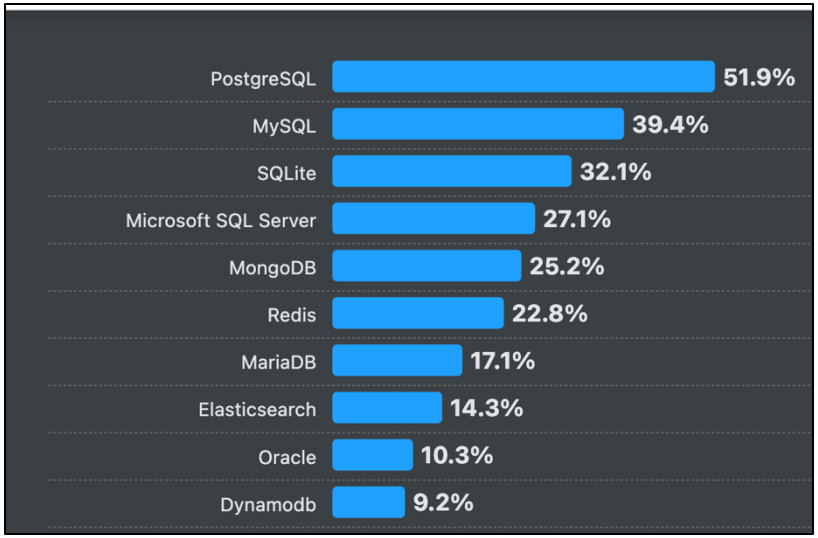


PostgreSQL's Rise

PostgreSQL is topping DB-Engines rankings and gaining popularity on Stack Overflow.

- Postgres 4/8
- Snowflake 3/4 in last 4 years!

Get in the snowflake territory?

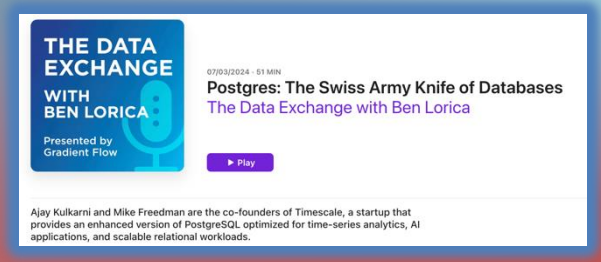


Postgres is for OLTP, why OLAP?

- Apps often have OLAP needs before OLAP stack
- For most, Postgres is already in the stack
  - An extra DB is a lot of “extra” work!
- The data is already there in Postgres. No ETL!
- Of course, Postgres can do some analytics
  - Can do even more with custom indexes
  - More storage & writes for lesser latency
- But can Postgres stretch more on OLAP?
  - Can we delay dedicated OLAP DB a bit more?

Unified Data Management with ClickHouse® and Postgres at OSA CON ’24
https://thenewstack.io/unleashing-postgres-for-analytics-with-duckdb-integration/

Postgres: The Swiss Army knife of DBs



Done it in past!

- Oracle used to be gold standard for RDBMS
  - Postgres adoption is taking off now!
- MongoDB popularized JSON
  - Postgres JSONB can take you quite far!
- Vector DBs are getting popular now..
  - pgvector has support for similarity search
  - Find nearest neighbours of a given vector with indexes
- Popular extensions & forks (citus, timescale, Greenplum etc)

https://tembo.io/blog/vector-indexes-in-pgvector

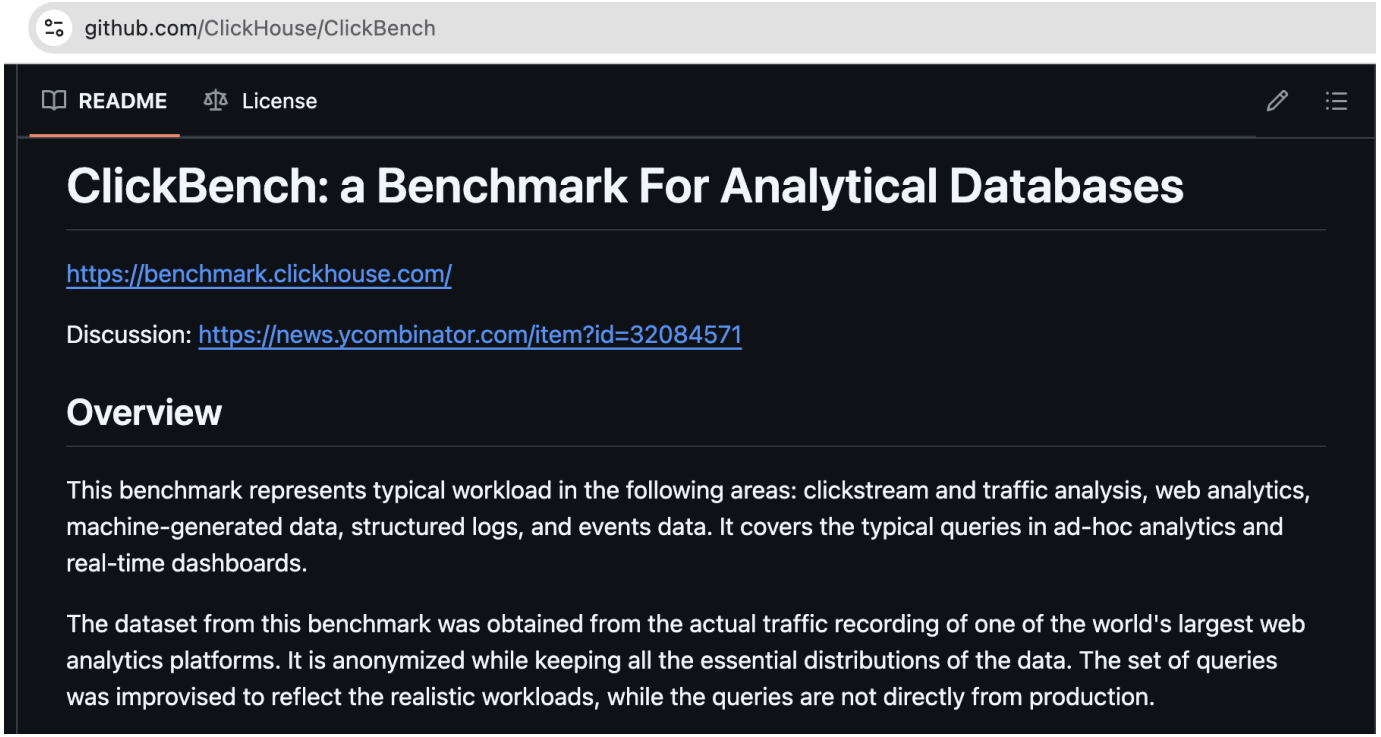
2. Benchmarks

"All benchmarks are **lies**."
Do your own perf....
In any case, perf is not enough.

Take it with a grain of salt ☺

Introducing ClickBench

- A benchmark for analytics databases.
- Originally built to show ClickHouse performance
- Evaluates databases on real-world analytics workloads:
  - High-volume table scans
  - Complex aggregations
- Historically, ClickHouse & analytics databases dominated
- A lot of PostgreSQL compatible databases in the list now!



~100 million records , 42 OLAP style queries & lots of limitations, of course!

Postgres vs Snowflake for OLAP

*** A comparison like this is wrong in so many ways…*

*** Timing is based on geometric mean of latencies of 42 queries. In real life, you need those specific queries only…*

Tuned PostgreSQL?

**Timing is based on geometric mean of latencies of 42 queries. In real life, you need some specific queries…*

Tuned PostgreSQL?

Configuration Parameters

This postgresql runs with the following values of postgresql parameters changed.

Parameter	Old Configuration	New Configuration
shared_buffers	128MB	8GB
max_parallel_workers	8	16
max_parallel_workers_per_gather	2	8
max_wal_size	1GB	32GB

Change configs and add indexes

ClickBench / postgresql-tuned / **index.sql**

patricklauer postgresql-tuned: Add indexes

Code | Blame | 24 lines (18 loc) · 967 Bytes

```
1   CREATE INDEX adveng on hits (advengineid);
2   CREATE INDEX regid  on hits (RegionID);
3   CREATE INDEX cid on hits (counterid);
4   CREATE INDEX eventtime on hits (eventtime);
5   CREATE INDEX eventdate on hits (eventdate);
6   CREATE INDEX mobile on hits (mobilephonemodel);
7   CREATE INDEX refresh on hits (isrefresh, dontcounthits);
8   CREATE INDEX resolutionwidth on hits (resolutionwidth);
9   CREATE INDEX search on hits (searchphrase);
10  CREATE INDEX userid on hits (userid);
11
12  CREATE INDEX useridsearch on hits (userid, searchphrase);
13  CREATE INDEX widcip on hits (watchid, clientip);
14  CREATE INDEX mobileuser on hits (MobilePhoneModel,UserID);
15  CREATE INDEX regionuser on hits (RegionID,UserID);
16
17  CREATE INDEX mobile2 on hits (mobilephonemodel) WHERE mobilephonemodel <> ''::text;
18  CREATE INDEX search2 on hits (searchphrase) WHERE searchphrase <> ''::text;
19
20
21  CREATE INDEX trgm_idx_title ON hits USING gin (title gin_trgm_ops);
22  CREATE INDEX trgm_idx_url ON hits USING gin (url gin_trgm_ops);
```

More insights at https://github.com/shiv4289/pg-olap-recipies/blob/main/benchmarking/postgresql_tuned/benchmark.md

Trade-off: Storage vs latency

```
test=# SELECT
    i.indexname AS index_name,
    i.indexdef AS index_def,
    pg_size_pretty(pg_relation_size(s.indexrelid)) AS index_size
FROM pg_indexes i
JOIN pg_stat_user_indexes s ON i.indexname = s.indexrelname
WHERE i.tablename = 'hits'
ORDER BY pg_relation_size(s.indexrelid) DESC;
   index_name    |                                                                                                  index_def                                                                                                  | index_size
-----------------+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+------------
 trgm_idx_url    | CREATE INDEX trgm_idx_url ON public.hits USING gin (url gin_trgm_ops)                                                                                                                                       | 12 GB
 trgm_idx_title  | CREATE INDEX trgm_idx_title ON public.hits USING gin (title gin_trgm_ops)                                                                                                                                   | 9125 MB
 widcip          | CREATE INDEX widcip ON public.hits USING btree (watchid, clientip)                                                                                                                                          | 3004 MB
 useridsearch    | CREATE INDEX useridsearch ON public.hits USING btree (userid, searchphrase)                                                                                                                                 | 1936 MB
 regionuser      | CREATE INDEX regionuser ON public.hits USING btree (regionid, userid)                                                                                                                                       | 1179 MB
 search          | CREATE INDEX search ON public.hits USING btree (searchphrase)                                                                                                                                               | 1174 MB
 mobileuser      | CREATE INDEX mobileuser ON public.hits USING btree (mobilephonemodel, userid)                                                                                                                               | 1169 MB
 userid          | CREATE INDEX userid ON public.hits USING btree (userid)                                                                                                                                                     | 1015 MB
 eventtime       | CREATE INDEX eventtime ON public.hits USING btree (eventtime)                                                                                                                                               | 680 MB
 cid             | CREATE INDEX cid ON public.hits USING btree (counterid)                                                                                                                                                     | 662 MB
 regid           | CREATE INDEX regid ON public.hits USING btree (regionid)                                                                                                                                                    | 662 MB
 resolutionwidth | CREATE INDEX resolutionwidth ON public.hits USING btree (resolutionwidth)                                                                                                                                   | 661 MB
 mobile          | CREATE INDEX mobile ON public.hits USING btree (mobilephonemodel)                                                                                                                                           | 661 MB
 eventdate       | CREATE INDEX eventdate ON public.hits USING btree (eventdate)                                                                                                                                               | 661 MB
 refresh         | CREATE INDEX refresh ON public.hits USING btree (isrefresh, dontcounthits)                                                                                                                                  | 661 MB
 adveng          | CREATE INDEX adveng ON public.hits USING btree (advengineid)                                                                                                                                                | 661 MB
 search2         | CREATE INDEX search2 ON public.hits USING btree (searchphrase) WHERE (searchphrase <> ''::text)                                                                                                             | 600 MB
 mobile2         | CREATE INDEX mobile2 ON public.hits USING btree (mobilephonemodel) WHERE (mobilephonemodel <> ''::text)                                                                                                     | 37 MB
(18 rows)
```

Choose indexes you need.

```
test=# SELECT pg_size_pretty(pg_indexes_size('hits')) AS indexes_size;
 indexes_size
--------------
 36 GB
(1 row)
```

More insights at https://github.com/shiv4289/pg-olap-recipies/blob/main/benchmarking/postgresql_tuned/benchmark.md

Improving Postgresql on ClickBench

- Insert time improved by almost **22%**
- Table size reduced by **5Gb** (~ 4%)
- Indexing Time improved by 2%
- Query time improved by ~10%

	Postgresql Tuned (16 vCpu, 32 Gb, 500Gb)	Postgresql Tuned Padding Aligned (16 vCpu, 32 Gb, 500Gb)
Load time:	502s (×1.22)	**410s (×1.00)**
Data size:	120 GiB (×1.17)	**115.79 GiB (×1.00)**
Indexing time:	**7478s (×1.00)**	7642s (×1.02)
Index size:	**36.00 GiB (×1.00)**	**36.00 GiB (×1.00)**

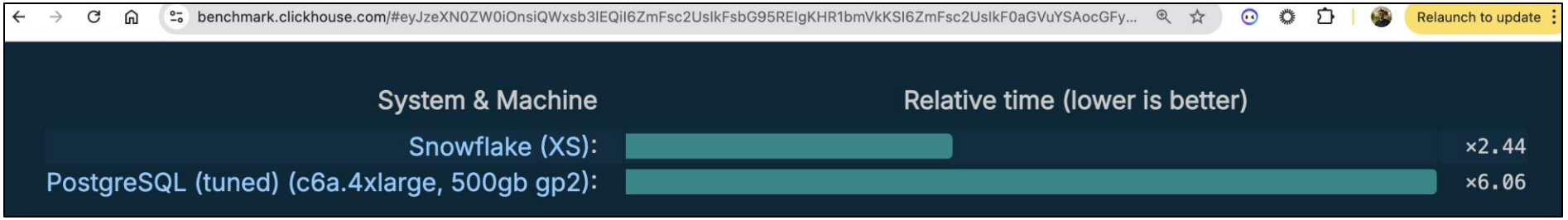
Add pg_duckdb benchmarking for existing postgres tables. 1 5
#311 by saurabhojha was merged 1 hour ago

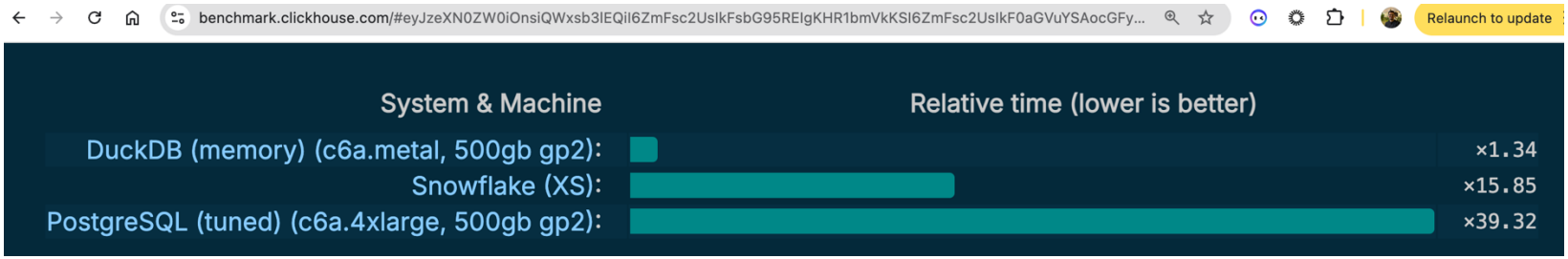
Update create.sql query in pg_duckdb to comply to r['colname'] syntax introduced in pg_duckdb 0.3.0 1 4
#307 by saurabhojha was merged last week

Refactor `postgresql-tuned` Benchmark: Optimize Data Insertion with Column Padding Alignment
#310 by somratdutta was merged last week

Refactor `postgresql/benchmark.sh` to Wrap COPY FREEZE in a Transaction Block
#309 by somratdutta was merged last week

Let’s check out DuckDB

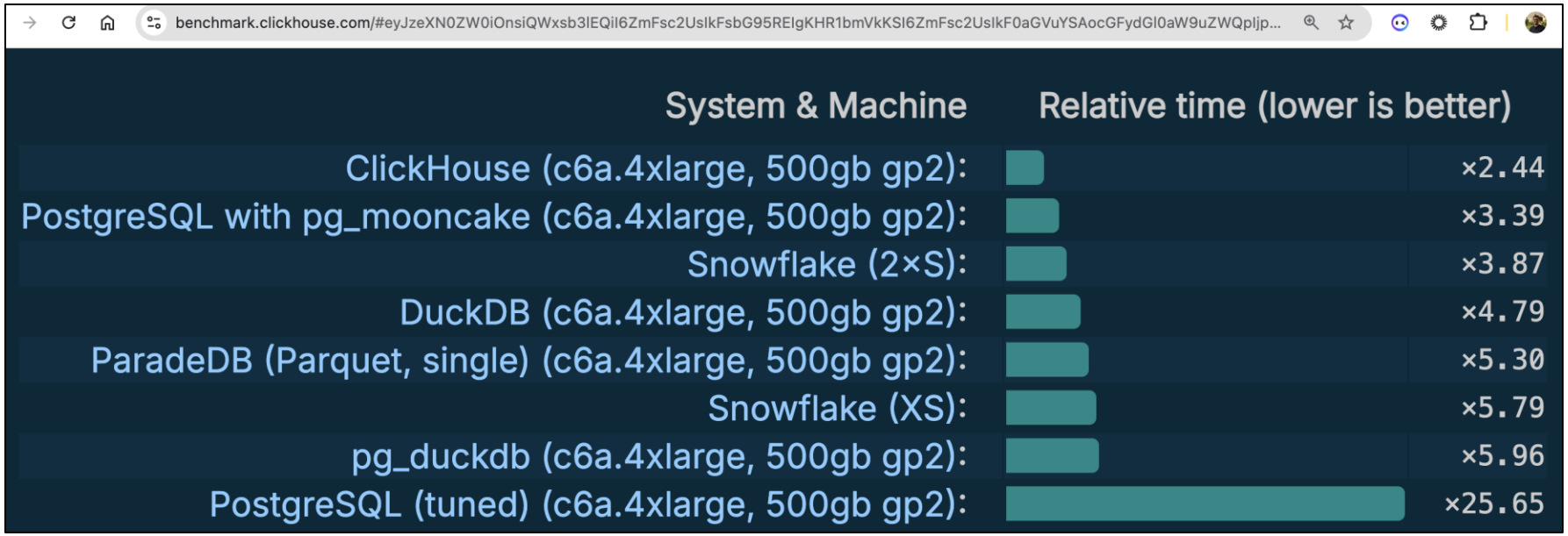




*** We’ll get back on DuckDB in just a while..*

Let’s play with clickBench just a little more??

Postgres – DuckDB Combo!

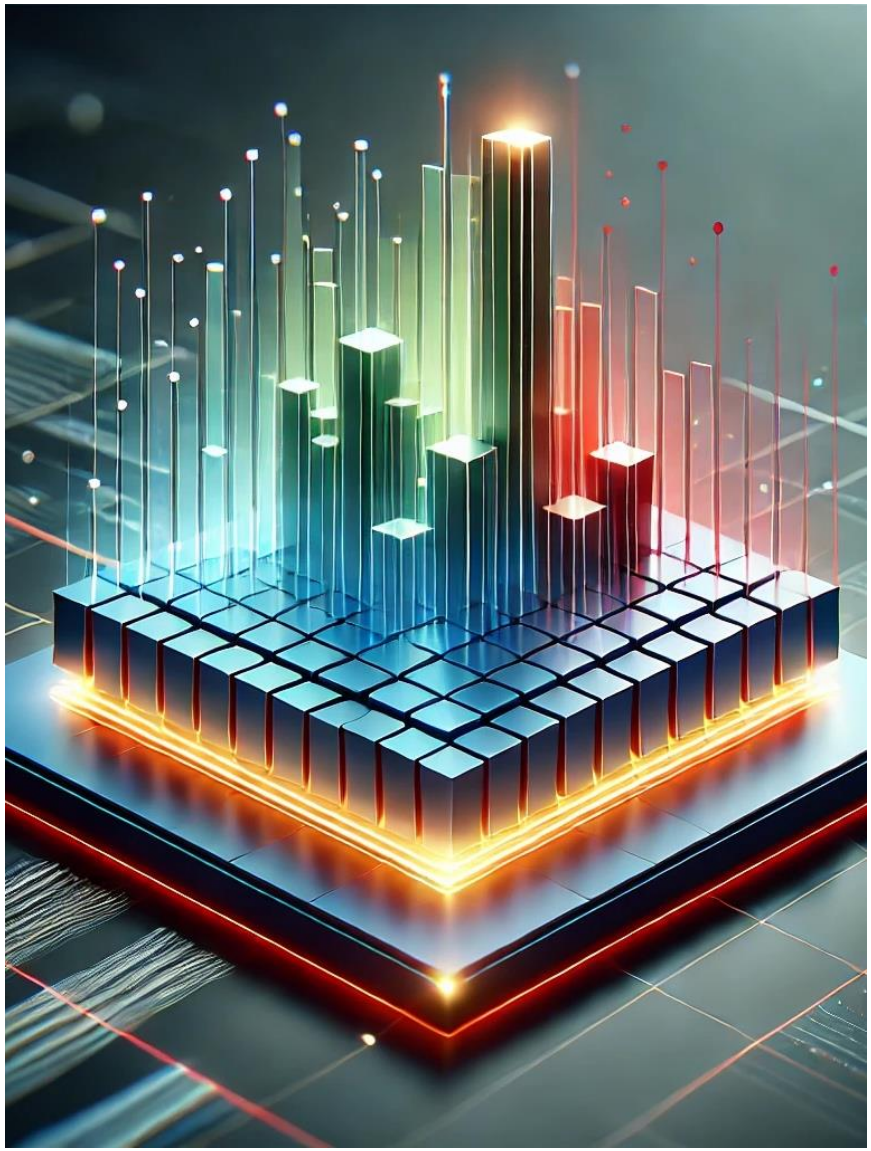


- *What are these other DBs?*
- *Why are they performing better than even tuned postgres?*
- *Are some really better than snowflake? Or is the benchmark lying (again)?*
- *So, clickhouse is the fastest OLAP DB?*

Objective of the Talk

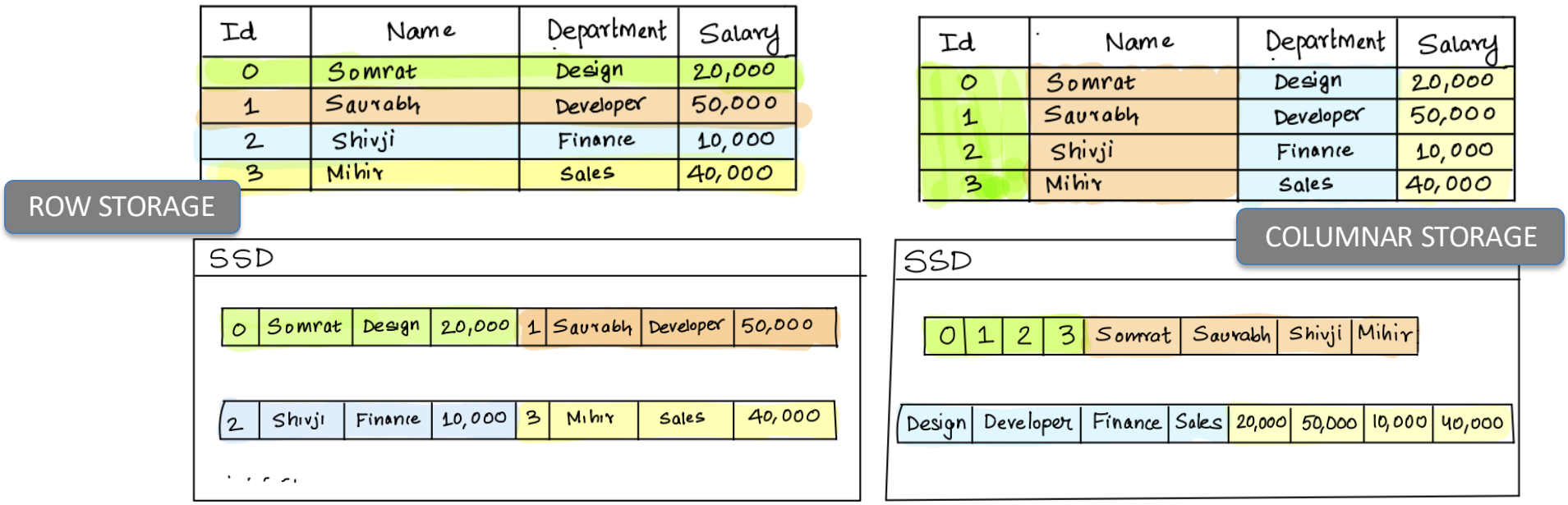
To explore how PostgreSQL extensions bridge the gap to OLAP workloads.

3. What Makes a Good OLAP DB in 2025 ?



Characteristics of OLAP Systems

- Key features include:
  1. **Columnar storage**



Characteristics of OLAP Systems

- Key features include:
  1. **Columnar storage**
     - Easier fetching by column

Id	Name	Department	Salary
0	Somrat	Design	20,000
1	Saurabh	Developer	50,000
2	Shivji	Finance	10,000
3	Mihir	Sales	40,000

ROW STORAGE

SSD

0	Somrat	Design	20,000	1	Saurabh	Developer	50,000
2	Shivji	Finance	10,000	3	Mihir	Sales	40,000

Id	Name	Department	Salary
0	Somrat	Design	20,000
1	Saurabh	Developer	50,000
2	Shivji	Finance	10,000
3	Mihir	Sales	40,000

COLUMNAR STORAGE

SSD

0	1	2	3	Somrat	Saurabh	Shivji	Mihir
Design	Developer	Finance	Sales	20,000	50,000	10,000	40,000

https://news.ycombinator.com/item?id=37247945

Characteristics of OLAP Systems

- Key features include:
  1. **Columnar storage**
     - Allows for better compression ratios

File Format/System	Relative Size	Multiplier
hits.parquet	13.76 GiB	1.0
hits.tsv	69.67 GiB	5.07
hits.csv	75.56 GiB	5.5
hits.json	216.75 GiB	15.77

https://news.ycombinator.com/item?id=37247945

Characteristics of OLAP Systems

- Key features include:
  1. **Columnar storage**
     - Allows for better compression ratios

Database	Relative Size	Size Multiplier
Snowflake (XS)	11.46 GiB	x1.00
PostgreSQL with pg_mooncake (c6a.4xlarge, 500gb gp2)	13.62 GiB	x1.19
Crunchy Bridge for Analytics (Parquet) (Analytics-256GB, 64 vCores)	13.76 GiB	x1.20
ParadeDB (Parquet, single) (c6a.4xlarge, 500gb gp2)	13.76 GiB	x1.20
pg_duckdb (c6a.4xlarge, 500gb gp2)	13.80 GiB	x1.20
PostgreSQL (c6a.4xlarge, 500gb gp2)	**72.45 GiB**	**x6.32**
DuckDB (memory) (c6a.metal, 500gb gp2)	95.29 GiB	x8.32
PostgreSQL (tuned) (c6a.4xlarge, 500gb gp2)	120.02 GiB	x10.48

Columnar Store: Apache Parquet

Spec : https://github.com/apache/parquet-format

- A free, open-source, widely adopted data storage format.
- **Efficiency**:
  - Traditional databases load data in row-oriented formats for analysis.
  - Parquet is optimized for direct querying.
  - Can read specific columns without the need to process entire datasets.
  - Zone Maps: Uses statistical metadata to skip over unnecessary data blocks, reducing latency and I/O.
- **Use Case**:
  - Ideal for quick reads of specific columns, making it highly effective for analytical querying and data-intensive applications.

[AWS Blog] Adapting to Change with Data Patterns on AWS

- Widespread Adoption of Parquet
  - Major data lake customers (Netflix, Nubank, Lyft, Pinterest) use Apache Parquet for storing business data.
  - Parquet stores data in a table format and is highly compressed.
- Scale and Growth of Parquet on AWS
  - One of the fastest-growing data types in Amazon S3.
  - Exabytes of Parquet data stored on AWS.
  - AWS handles 15M+ requests per second and serves hundreds of petabytes of Parquet daily.
- Case study: Standardization with Apache Iceberg at Pinterest
  - Pinterest standardizes storage using S3 (storage layer), Parquet (tabular data format), and Apache Iceberg (open table format - OTF).
  - Thousands of business-critical Iceberg tables.

Source: https://aws.amazon.com/blogs/storage/adapting-to-change-with-data-patterns-on-aws-the-aggregate-cloud-data-pattern/

Characteristics of OLAP Systems

- Key features include:
  1. Columnar storage
  2. **Vectorized Execution**

Characteristics of OLAP Systems

- Key features include:
  1. Columnar storage
  2. **Vectorized Execution**

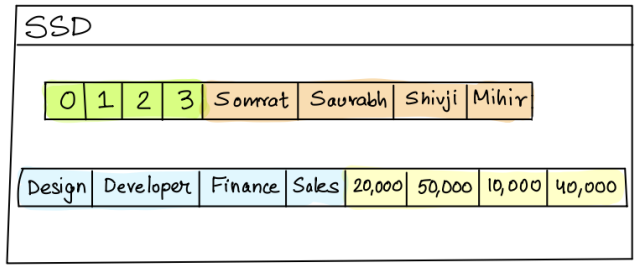
Takes advantages of modern CPUs, which can perform operations on multiple values simultaneously using SIMD (Single instruction, Multiple Data) instructions.

Characteristics of OLAP Systems

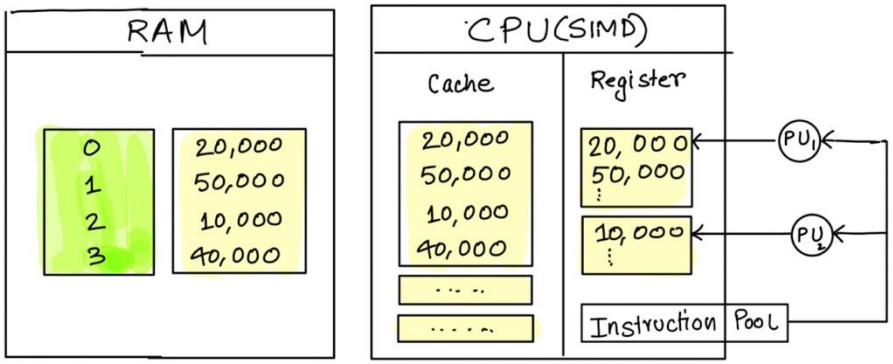
- Key features include:
  1. Columnar storage
  2. **Vectorized Execution**

Id	Name	Department	Salary
0	Somrat	Design	20,000
1	Saurabh	Developer	50,000
2	Shivji	Finance	10,000
3	Mihir	Sales	40,000

COLUMNAR STORAGE



Select id, salary where Salary ?15,000;



Columnar storage & SIMD is a match made in heaven

Characteristics of OLAP Systems

- Key features include:
  1. Columnar storage
  2. Vectorized Execution
  3. **Custom OLAP Indexes**
     a. **We will look at sparse indexing in clickhouse**

ClickHouse: Storage Layout

```
CREATE TABLE hits_UserID_URL
(
    `UserID` UInt32,
    `URL` String,
    `EventTime` DateTime
)
ENGINE = MergeTree
// highlight-next-line
PRIMARY KEY (UserID, URL)
ORDER BY (UserID, URL, EventTime)
```

```
INSERT INTO hits_UserID_URL SELECT
   intHash32(UserID) AS UserID,
   URL,
   EventTime
FROM url('https://datasets.clickhouse.com/hits/tsv/hits_v1.tsv.xz',
WHERE URL != '';
```

```
~/dev/clickhouse/store/f75/f753d709-e220-4ca1-90b5-26e9a0b4292f/all_1_7_1 (0.044s)
du -h * | sort -hr

161M    URL.bin
 27M    EventTime.bin
704K    UserID.bin
 76K    primary.idx
4.0K    serialization.json
4.0K    metadata_version.txt
4.0K    default_compression_codec.txt
4.0K    count.txt
4.0K    columns.txt
4.0K    checksums.txt
4.0K    UserID.cmrk
4.0K    URL.cmrk
4.0K    EventTime.cmrk
```

Column files

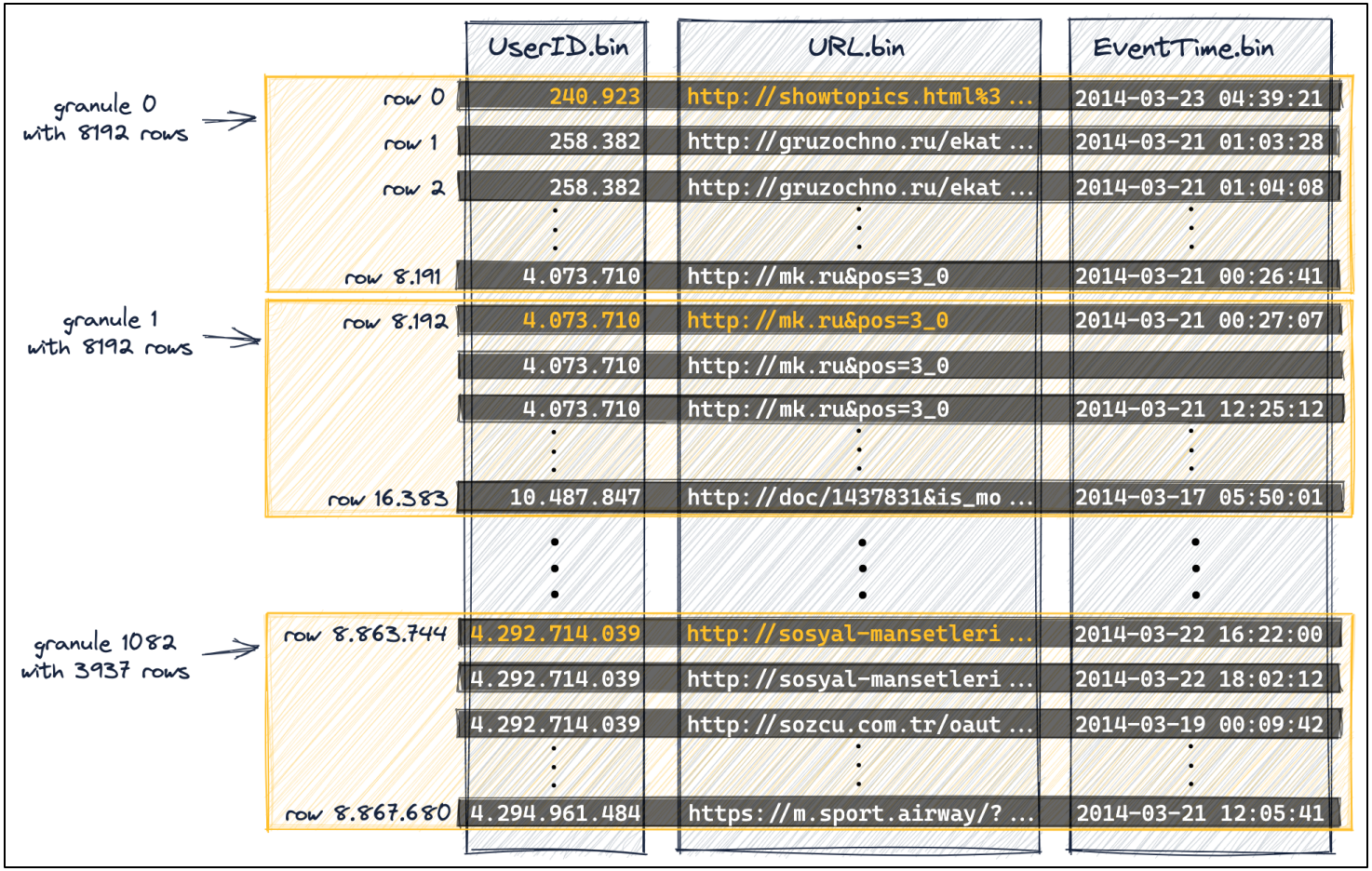
Primary index

cmrk files

offset for primary index granules

https://clickhouse.com/docs/en/guides/best-practices/sparse-primary-indexes

ClickHouse: Column Granules



https://clickhouse.com/docs/en/guides/best-practices/sparse-primary-indexes

ClickHouse: Sparse (PK) Index

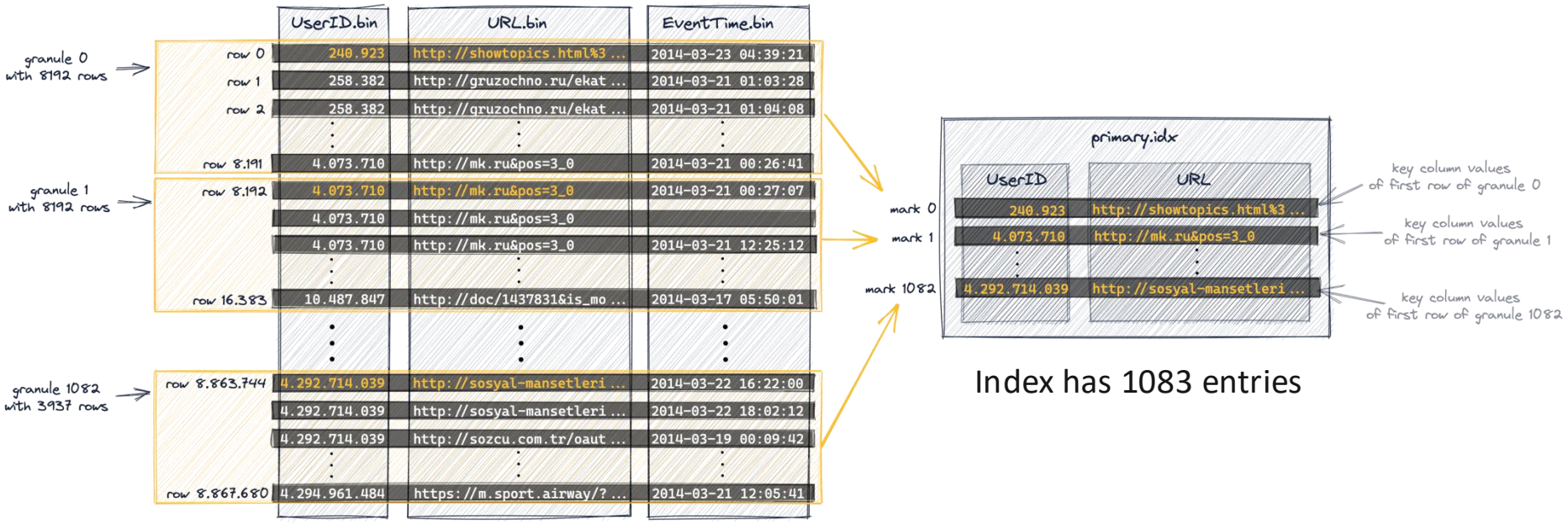


Table has 8.87 million rows (1083 granules)

Index has 1083 entries

https://clickhouse.com/docs/en/guides/best-practices/sparse-primary-indexes

ClickHouse: Query Path

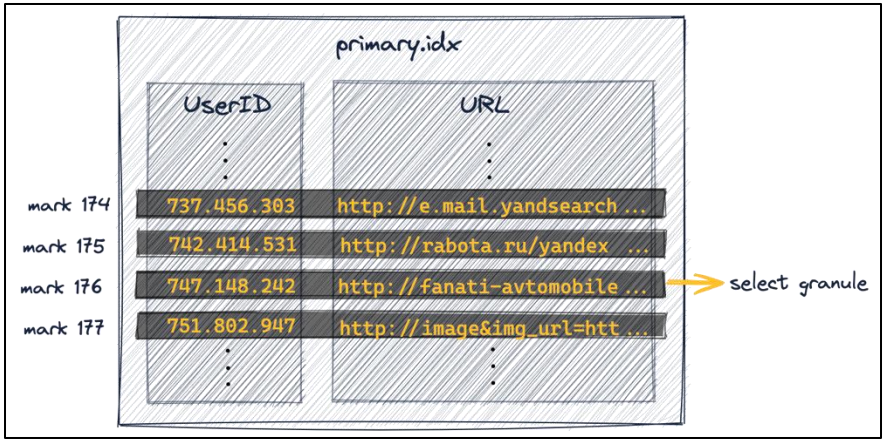
The following calculates the top 10 most clicked urls for the UserID 749927693.

```
SELECT URL, count(URL) AS Count
FROM hits_UserID_URL
WHERE UserID = 749927693
GROUP BY URL
ORDER BY Count DESC
LIMIT 10;
```

The response is:

```
┌─URL────────────────────────────────┬─Count─┐
│ http://auto.ru/chatay-barana..     │   170 │
│ http://auto.ru/chatay-id=371...    │    52 │
│ http://public_search               │    45 │
│ http://kovrik-medvedevushku-...    │    36 │
│ http://forumal                     │    33 │
│ http://korablitz.ru/L_1OFFER...    │    14 │
│ http://auto.ru/chatay-id=371...    │    14 │
│ http://auto.ru/chatay-john-D...    │    13 │
│ http://auto.ru/chatay-john-D...    │    10 │
│ http://wot/html?page/23600_m...    │     9 │
└────────────────────────────────────┴───────┘

10 rows in set. Elapsed: 0.005 sec.
Processed 8.19 thousand rows,
740.18 KB (1.53 million rows/s., 138.59 MB/s.)
```



1. Select granule (binary search)
2. Find disk offset in <column>.cmrk file
3. Decompress disk block
4. Stream granule 176 to clickhouse

Table has 8.87 million rows (1083 granules)
Each granule has 8192 rows

https://clickhouse.com/docs/en/guides/best-practices/sparse-primary-indexes

Characteristics of OLAP Systems

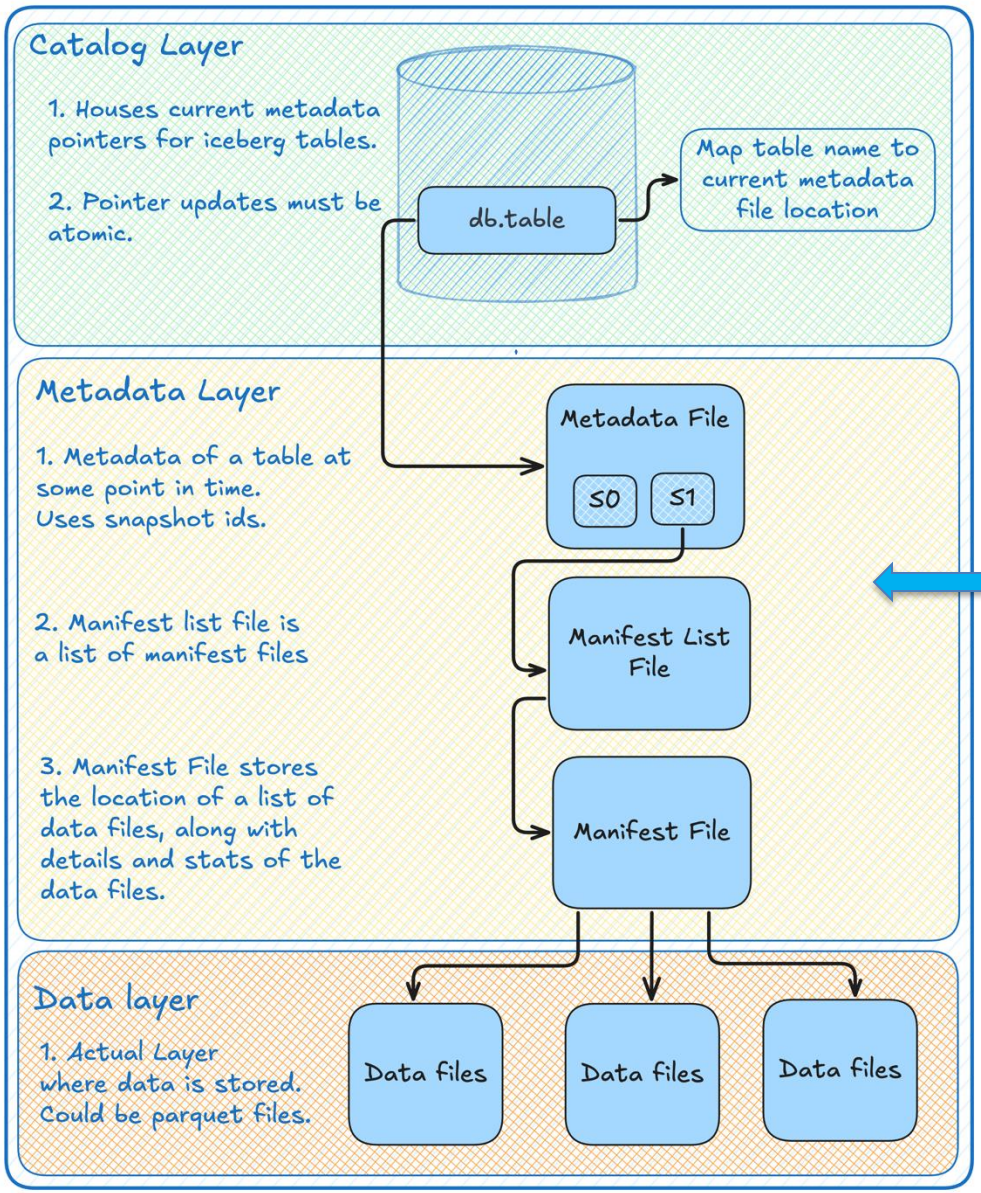
- Key features include:
  1. Columnar storage
  2. Vectorized Execution
  3. Custom OLAP Indexes
     a) We will look at sparse indexing in clickhouse
  4. **Data Lake Integrations**
     a) **Apache Iceberg, Delta Lake and Apache Hudi**
     b) **Querying parquet files directly**

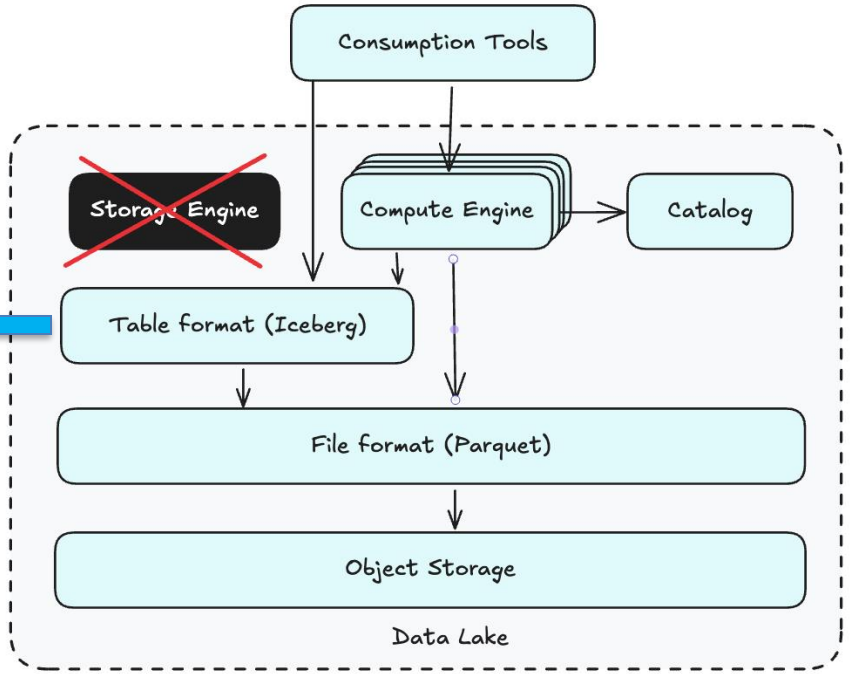
Iceberg: Open Table Formats

✅ What Iceberg is	❌ What Iceberg is not
- A table format specification - A set of APIs and libraries for engines to interact with tables following that specification	- A storage engine - An execution engine - A service

https://www.dremio.com/resources/guides/apache-iceberg-an-architectural-look-under-the-covers/

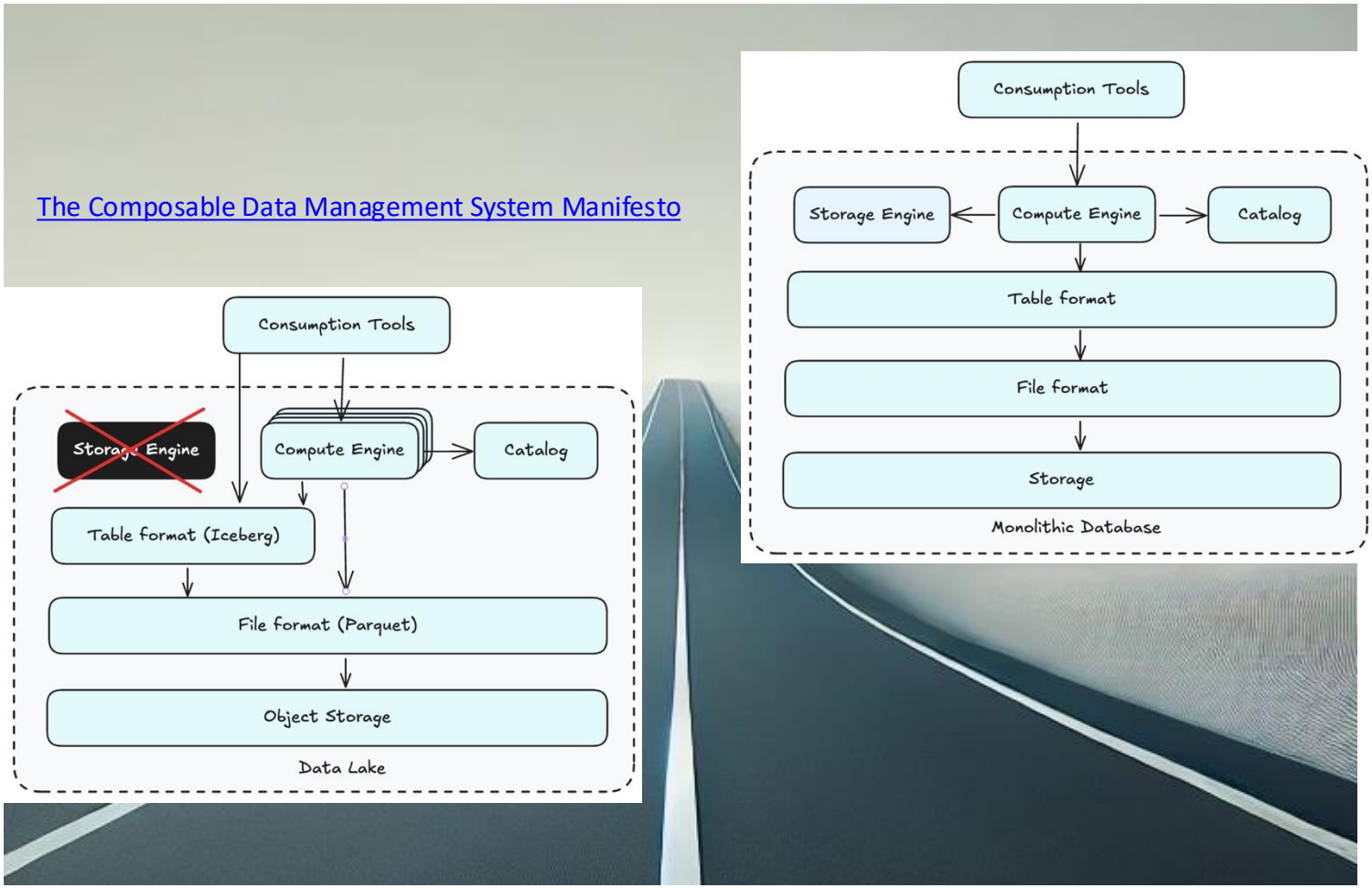
Iceberg: Open Table Formats





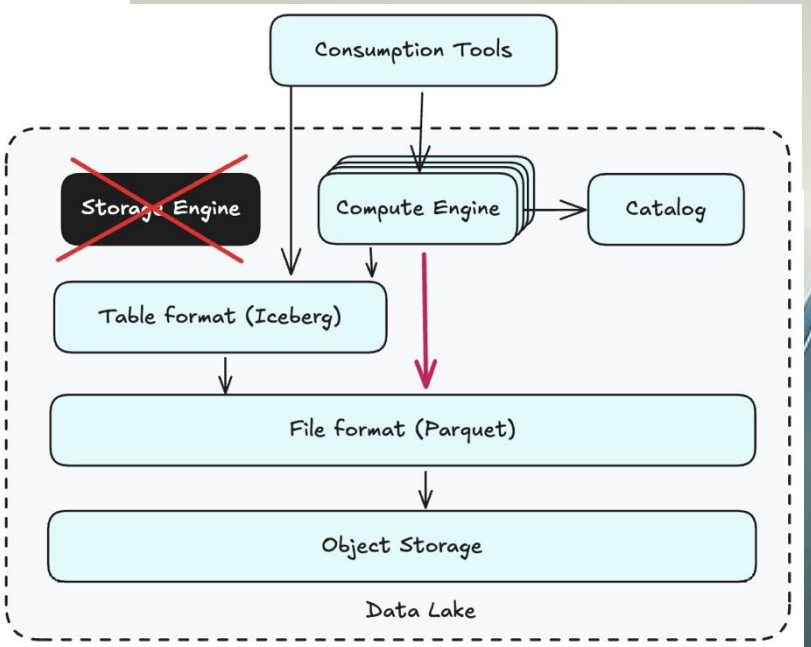
https://www.dremio.com/resources/guides/apache-iceberg-an-architectural-look-under-the-covers/

Iceberg Table Format



https://www.oreilly.com/library/view/apache-iceberg-the/9781098148614

Postgres : read_parquet()?

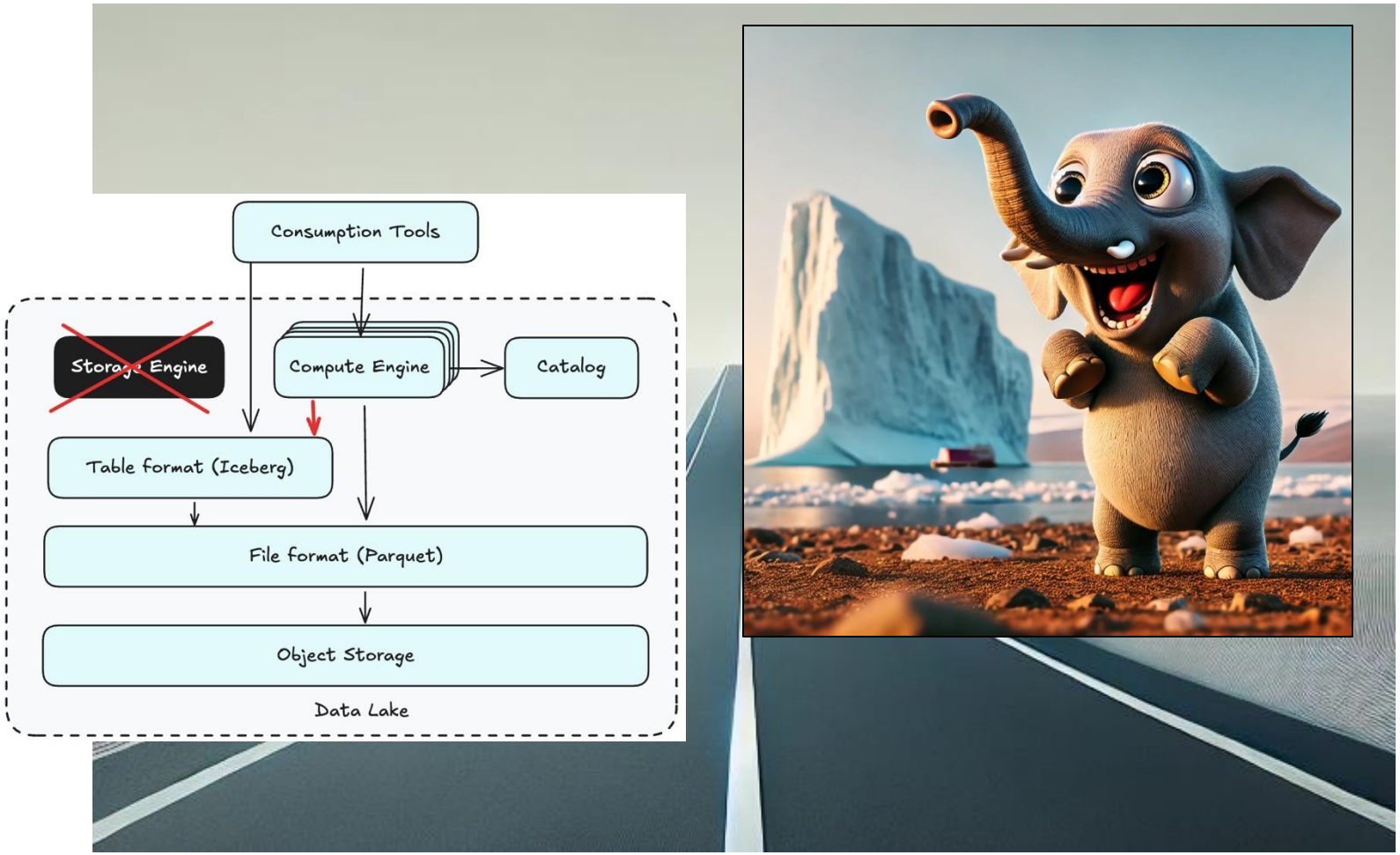


Parquet files

S3 Table

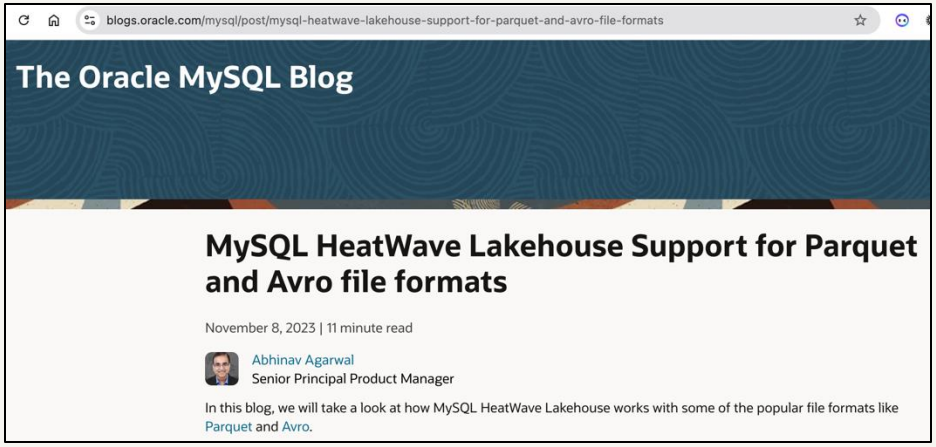
https://www.oreilly.com/library/view/apache-iceberg-the/9781098148614

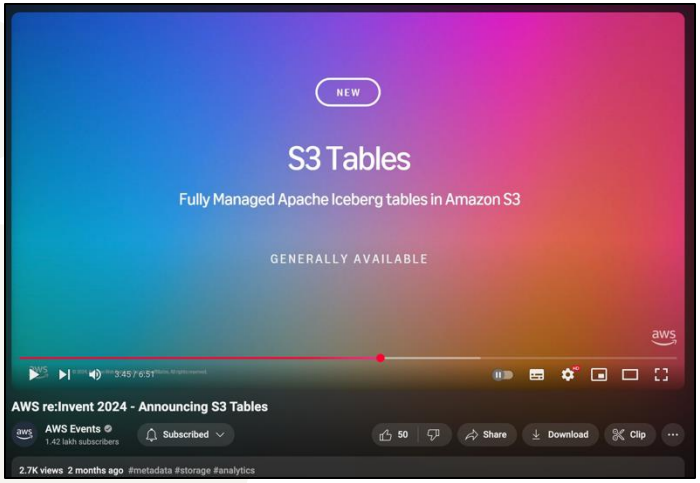
Postgres : read_iceberg()?

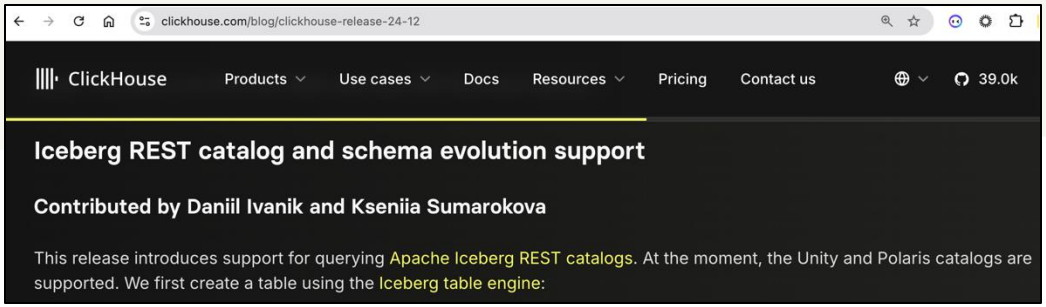


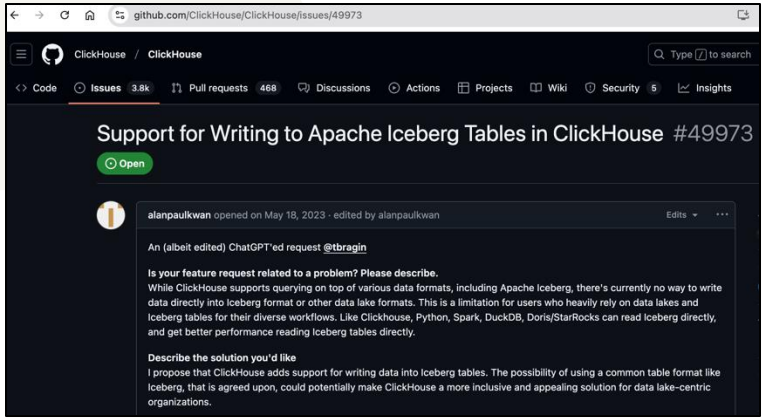
https://www.oreilly.com/library/view/apache-iceberg-the/9781098148614

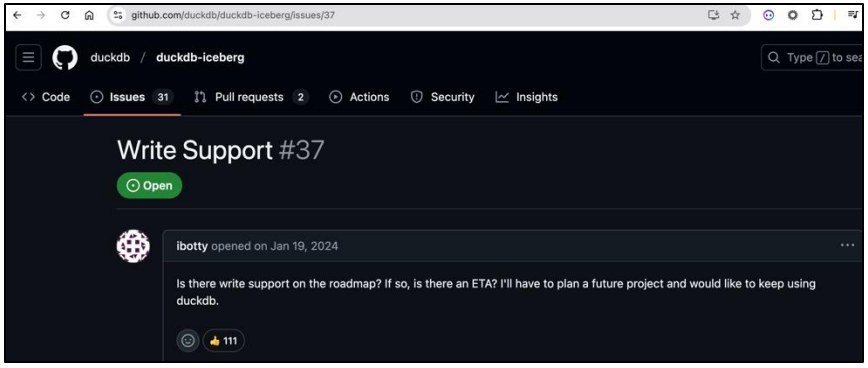
How about the neighborhood again?

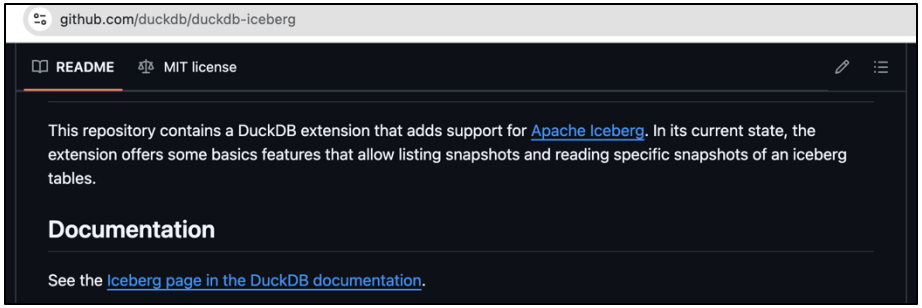








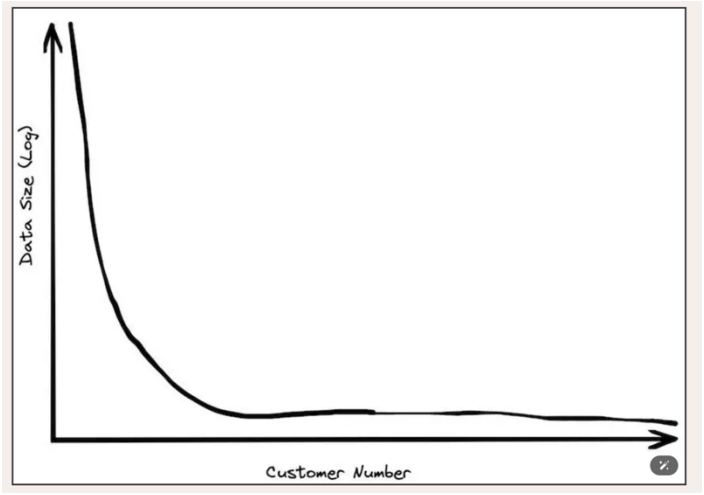




Characteristics of OLAP Systems

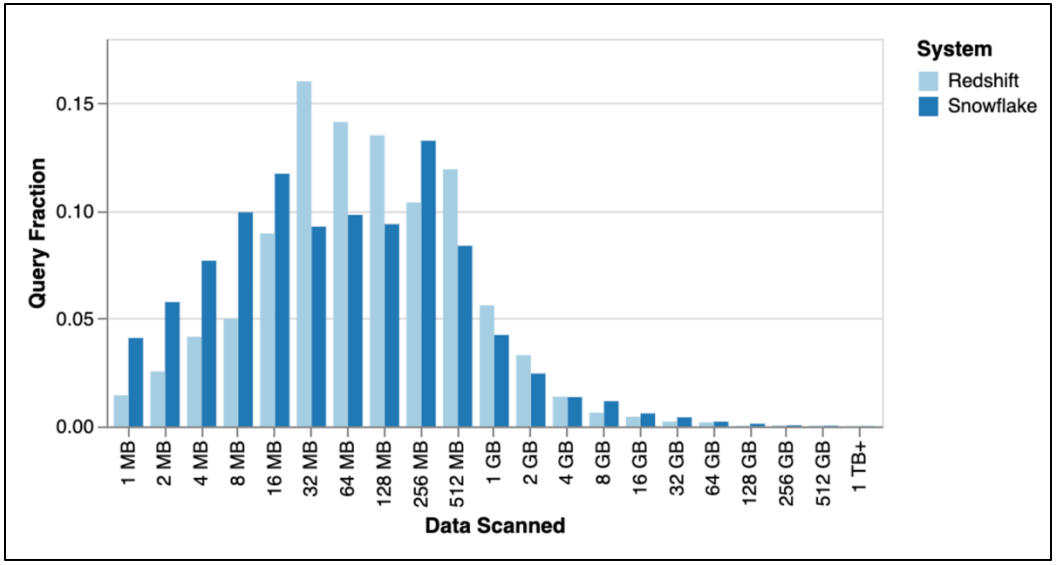
- Key features include:
  1. Columnar storage
  2. Vectorized Execution
  3. Custom OLAP Indexes
     a) We will look at sparse indexing in clickhouse
  4. Data Lake Integrations
     a) Apache Iceberg, Delta Lake and Apache Hudi
     b) Querying parquet files directly
  5. **Distribution (Sharding?)**
     a) **Single node capacity these days..**

Size of OLAP workloads



MOST PEOPLE DON'T HAVE THAT MUCH DATA

https://motherduck.com/blog/big-data-is-dead/

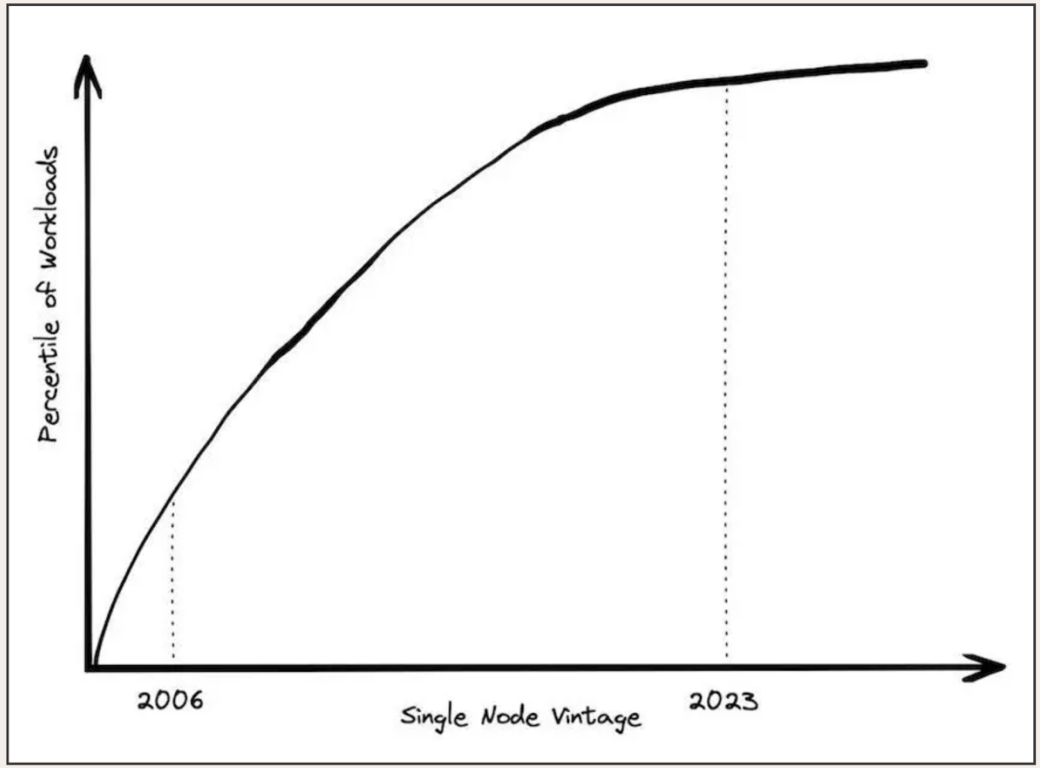


Of queries that scan at least 1 MB, the median query scans about 100 MB.

The 99.9th percentile query scans about 300 GB.

https://www.fivetran.com/blog/how-do-people-use-snowflake-and-redshift

Compute Evolution



THE BIG DATA FRONTIER KEEPS RECEDING

One definition of “Big Data” is “whatever doesn’t fit on a single machine.. By that definition, the number of workloads that qualify has been decreasing every year.

https://motherduck.com/blog/big-data-is-dead/

4. Extending PostgreSQL

[**We will go a bit faster now!]

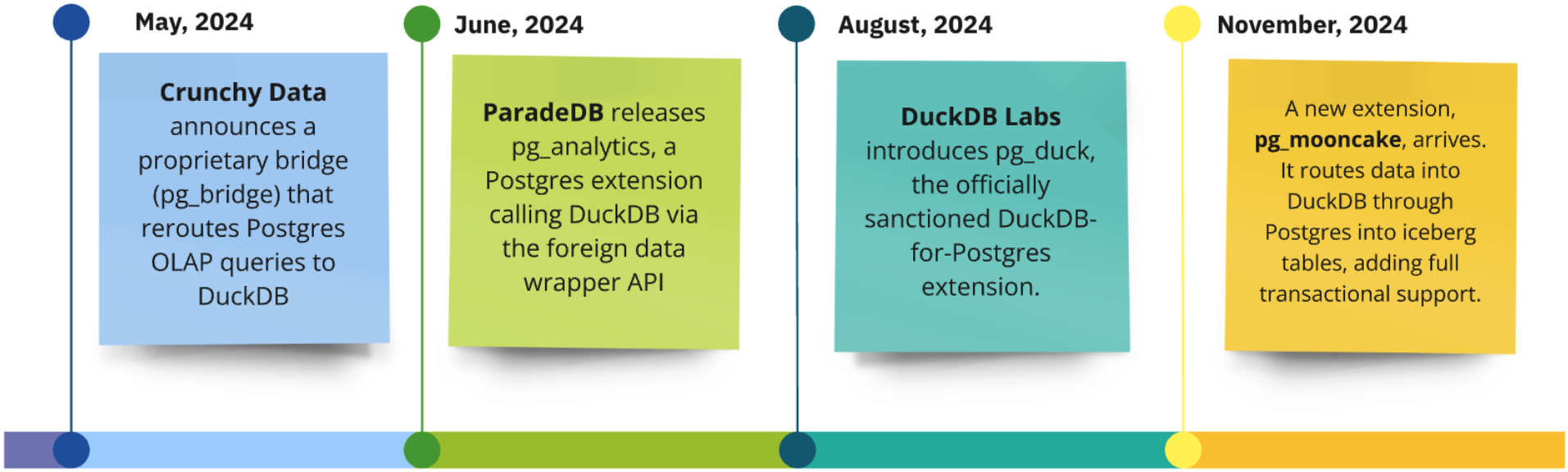
Rich Developer Ecosystem

- Extensions
  - TimescaleDB (time-series),
  - Citus (distributed SQL),
  - PostGIS (geospatial data)
  - **pg_duckdb**
- Foreign Data Wrappers (FDW)
  - (parquet_fdw, clickhouse_fdw…)
  - **paradeDB's pg_analytics**
- Table Access Methods (TAM)
  - **pg_mooncake (columnstore TAM)**
  - **Hydra (columnar TAM)**

System &Machine	Relative time (lower is better)
ParadeDB Parquet FDW Postgresql (16 vCpu, 32 Gb, 500Gb):	×2.04
pg_duckdb-Parquet Data (16 vCpu, 32 Gb, 500Gb):	×2.37
PG MoonCake Columnar TAM (16 vCpu, 32 Gb, 500Gb):	×3.12
Postgresql Tuned (16 vCpu, 32 Gb, 500Gb):	×8.02

Duck & Elephant Last year

1. github.com/duckdb/pg_duckdb
2. github.com/hydradatabase/columnar
3. github.com/paradedb/pg_analytics
4. github.com/Mooncake-Labs/pg_mooncake



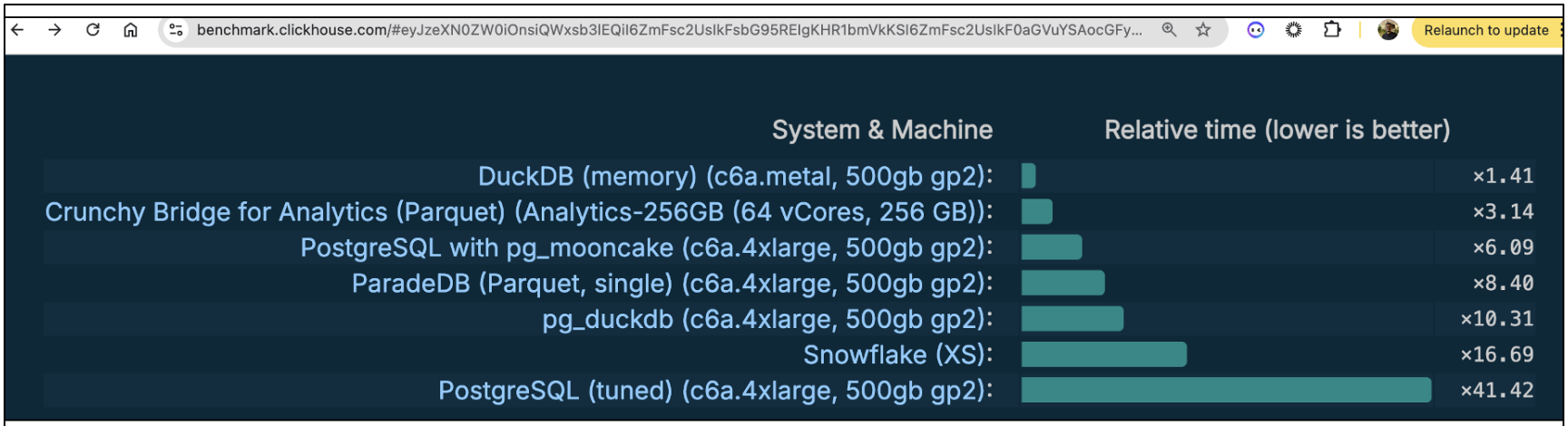
Embed OLAP in Postgres

"Overall, we are **very happy about choosing DuckDB** as the query engine…

We find:

- DuckDB is generally faster than DataFusion and
- more comprehensive than chdb,

https://www.crunchydata.com/blog/how-we-fused-duckdb-into-postgres-with-crunchy-bridge-for-analytics



- *What are these other DBs?*
- *Why are they performing better than even tuned postgres?*
- *Are these really better than snowflake? Or is the benchmark lying (again)?*

We Promised to be back ☺

Tell me about DuckDB

- Embeddable, in-memory analytics engine
- Optimized for single-node execution
- Columnar format for efficient storage
- Vectorized processing
- Flexible
  - In-memory
  - Persistent (single node)
  - Can read (parquet / Iceberg) on S3
  - MotherDuck for DBaaS

Postgres & DuckDB

☑	OLAP Features	Postgres + DuckDB
1	Columnar Storage	✓
2	Vectorized Execution	✓
3	Custom OLAP Indexes	✓
4	Querying Parquet Files Directly	✓
5	Apache Iceberg Integration	✓
6	Compute Capacity & Distribution	✓

pg_duckdb (by folks at duckdb & hydra)

- DuckDB's in-memory columnar, vectorized execution inside PostgreSQL extension. Efficient full-table scans & aggregations!
- No need to migrate data; runs queries in PostgreSQL memory
- Can query external data sources (Parquet, S3, MotherDuck).
- Fast ad-hoc analytics on PostgreSQL data*
- Inherits data lake integrations of DuckDB

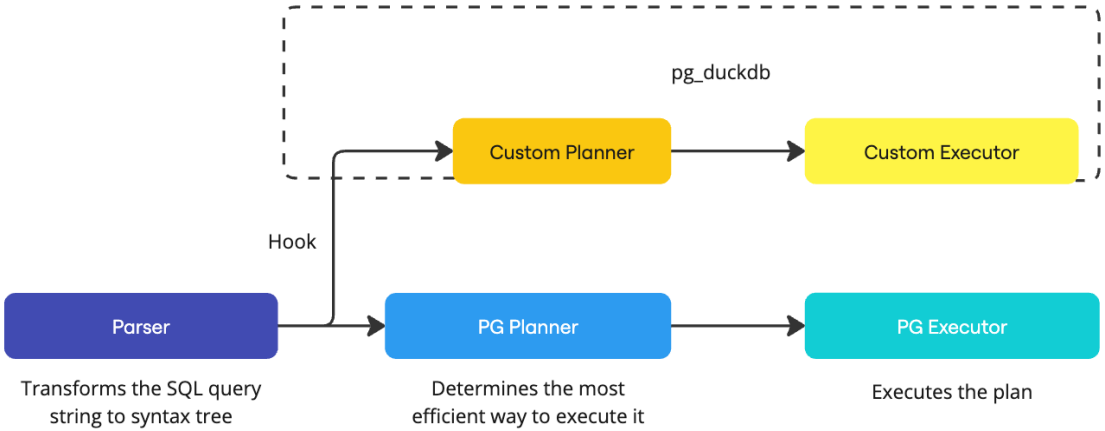
Data Lake Functions

Name	Description
`read_parquet`	Read a parquet file
`read_csv`	Read a CSV file
`read_json`	Read a JSON file
`iceberg_scan`	Read an Iceberg dataset
`iceberg_metadata`	Read Iceberg metadata
`iceberg_snapshots`	Read Iceberg snapshot information
`delta_scan`	Read a Delta dataset

https://github.com/duckdb/pg_duckdb/blob/main/docs/functions.md#data-lake-functions

Pg_duckdb Query Path

- pg_duckdb "steals" the query
  - if it involves a MotherDuck table
  - if it involves parquet/csv/json scanning
  - if duckdb.force_execution is set
- Then DuckDB fully executes the query
- DuckDB is also able to read Postgres' tables



Hydra (HTAP DB)

https://columnar.docs.hydra.so/

- Uses Table Access Methods for columnar engine
  - Append only like LSM tree.
- Choose (USING heap or columnar)
- Partitioned tables can combine row & columnar partitions.
  - Archive data from previous months in columnar, new in heap table
- Columnar storage and vectorization for OLAP
- Good for
  - aggregates (COUNT, SUM, AVG),
  - bulk INSERTs, UPDATE, DELETE…
  - Large numbers of columns where few are accessed
- Not good for frequent large updates.

```
CREATE EXTENSION IF NOT EXISTS columnar;

CREATE TABLE heap_table (id INT) USING heap;

CREATE TABLE columnar_table (id INT) USING columnar;
```

```
postgres=# \dt+
                          List of relations
 Schema |       Name       | Type  |  Owner   | Persistence | Access method | Size  | Description
--------+------------------+-------+----------+-------------+---------------+-------+-------------
 public | columnar_table   | table | postgres | permanent   | columnar      | 1 kB  |
 public | heap_table       | table | postgres | permanent   | heap          | 0 bytes |
(2 rows)
```

pg_analytics (paradeDB)

https://www.paradedb.com/

- Uses the foreign data wrapper (FDW) API to connect to S3 API.
- Uses executor hook API to push queries to DuckDB.
- Queries are pushed down to DuckDB query engine.
- Query object stores (S3) and table formats like Iceberg or Delta Lake.

To begin, enable the ParadeDB integrations with:

```
CREATE EXTENSION IF NOT EXISTS pg_analytics;
```

Now, let's create a **Postgres foreign data wrapper**, which is how ParadeDB connects to S3.

```
CREATE FOREIGN DATA WRAPPER parquet_wrapper
HANDLER parquet_fdw_handler VALIDATOR parquet_fdw_validator;

CREATE SERVER parquet_server FOREIGN DATA WRAPPER parquet_wrapper;

CREATE FOREIGN TABLE trips ()
SERVER parquet_server
OPTIONS (files 's3://paradedb-benchmarks/yellow_tripdata_2024-01.parquet');
```

Next, let's query the foreign table `trips`. You'll notice that the column names and types of this table are automatically inferred from the Parquet file.

```
SELECT vendorid, passenger_count, trip_distance FROM trips LIMIT 1;
```

pg_mooncake

https://github.com/Mooncake-Labs/pg_mooncake

- Table access method for columnstore table interface within postgres.
- pg_mooncake supports loading data from: Postgres heap tables, (Parquet, CSV, JSON files), (Iceberg, Delta Lake tables)
- Data stored in iceberg/delta lake format. External tools (Spark, Pandas, etc.) can directly read the same Parquet files.
- **Table metadata**, including addition and deletion of Parquet files, is stored **inside a Postgres table** for transactional consistency.

1. Enable the extension

```
CREATE EXTENSION pg_mooncake;
```

2. Create a columnstore table:

```
CREATE TABLE user_activity(
  user_id BIGINT,
  activity_type TEXT,
  activity_timestamp TIMESTAMP,
  duration INT
) USING columnstore;
```

3. Insert data:

```
INSERT INTO user_activity VALUES
  (1, 'login', '2024-01-01 08:00:00', 120),
  (2, 'page_view', '2024-01-01 08:05:00', 30),
  (3, 'logout', '2024-01-01 08:30:00', 60),
  (4, 'error', '2024-01-01 08:13:00', 60);

SELECT * from user_activity;
```

Query Execution:

1. Postgres parses SQL queries and generates execution plans.
2. Queries involving columnstore tables are treated as **analytics queries**.
3. These queries execute entirely in **DuckDB**, with results streamed back to **Postgres**.
4. Minor **query rewrites** bridge SQL syntax differences.

DuckDB Storage Extension:

1. Implements **custom storage format** similar to DuckDB's native storage.
2. Supports **physical operators** like TableScan, Insert, Update, Delete.
3. **pg_duckdb** enables reading **Postgres regular tables** from DuckDB.
4. Allows **joining columnstore tables** with Postgres heap tables.

https://www.mooncake.dev/blog/how-we-built-pgmooncake

Join Us in Advancing PostgreSQL for OLAP!

- We added some PRs to clickbench to optimize Postgres & Improve Ranking.
- Run Postgres on diverse OLAP datasets & queries
- Contribute best practices & patterns to olap-recipes
- Contribute to OLAP-focused Postgres extensions

🚀 **Let's make Postgres a top-tier OLAP database!**

github.com/shiv4289/olap-recipes

Q&A

github.com/shiv4289/shiv-tech-talks/

linkedin.com/in/shivjijha