LAST CLASS

Snowflake Data Warehouse

What matters to you when choosing a data platform? i.e. Snowflake, Databricks, BigQuery, Redshift? (self.dataengineering)

57 points submitted 1 day ago by Single_Anything_2980

29 comments share save hide report crosspost

sorted by: best

[+] discord-lan 67 points 1 day ago

You are way overthinking it. Here is the flow chart of how this decision is made. Are you on Google? If yes BigQuery. If you are on AWS or Azure, then ask are you a spark shop. If yes, then Databricks. If no, do you have money and like a positive experience? If yes, then Snowflake. Otherwise, choose Redshift.
CWI researchers recognized that data scientists do not use the full query capabilities of DBMSs due to the overhead of setting up and accessing data.

In 2017 they created an embedded version of MonetDB called MonetDBLite to run inside of R applications.
→ Running in-process reduces the cost of transferring data back and forth between the DBMS and the application.

But MonetDB had too much legacy baggage…
Multi-threaded embedded (in-process, serverless) DBMS that executes SQL over disparate data files. → PostgreSQL-like dialect with quality-of-life enhancements. → "SQLite for Analytics"

Provides zero-copy access to query results via Arrow to client code running in same process.

The core DBMS is nearly all custom C++ code with little to no third-party dependencies. → Relies on extensions ecosystem to expand capabilities.
Shared-Everything

**Push-based Vectorized Query Processing**

Precompiled Primitives

Multi-Version Concurrency Control

Morsel Parallelism + Scheduling

PAX Columnar Storage

Sort-Merge + Hash Joins

Stratified Query Optimizer

→ Supports unnesting of arbitrary subqueries
DUCKDB: PUSH-BASED PROCESSING

System originally used pull-based vectorized query processing but found it unwieldy to expand to support more complex parallelism.
→ Cannot invoke multiple pipelines simultaneously.

Switched to a push-based query processing model in 2021. Each operator determines whether it will execute in parallel on its own instead of a centralized executor.
System originally used pull-based vectorized query processing but found it unwieldy to expand to support more complex parallelism.

→ Cannot invoke multiple pipelines simultaneously.

Switched to a push-based query processing model in 2021. Each operator determines whether it will execute in parallel on its own instead of a centralized executor.
**DUCKDB: FINE-GRAINED CONTROL**

**Vector Cache:**
→ Buffer results between operators until it fills vector.

**Scan Sharing:**
→ Push results from one child operator to multiple parent operators (DAG plan).

**Backpressure / Async IO**
→ Pause operator execution when buffers are full or when waiting for remote I/O.

Source: Mark Raasveldt
**DUCKDB: VECTORS**

Custom internal vector layout for intermediate results that is compatible with Velox.

Supports multiple vector types:

- Flat Uncompressed array
  - 1 2 3 4 5
  - Physical & Logical

- Constant
  - All rows have the same value
  - 1
  - Physical

- Dictionary
  - Map of indexes to dictionary
  - 0 1 0 1
  - Physical

- Sequence
  - Base and increment
  - 1
  - Physical

Source: Mark Raasveldt
DuckDB uses a unified format to process all vector types without needing to decompress them first. → Reduce # of specialized primitives per vector type

**DUCKDB: VECTORS**

**Flat**
- Uncompressed array
  - Physical & Logical
  - 1 2 3 4 5

**Constant**
- All rows have the same value
  - Physical
    - 1
  - Logical
    - 1 1 1 1

**Dictionary**
- Map of indexes to dictionary
  - Physical
    - 0 1 0 1
  - Logical
    - a b
  - Dot
    - SelectionVector
    - 0 0
  - a b a b

**Unified Vector Format**

Source: Mark Raasveldt
DuckDB supports DataFrame libraries to query databases without using SQL.

→ **dplyr** (R-lang)
→ **Ibis** (Python)

Integration libraries generate DuckDB logical plans the DBMS converts into optimized physical plans.
→ Bypasses the SQL parser

Zero-copy result passing via Apache Arrow.
DUCKDB: STORAGE FORMAT

DBMS's built-in storage format maintains a single PAX-oriented file per database.
→ Splits tables into row groups with 120k tuples.
→ On-disk encoding is different than in-memory representation.

Two phase compression scheme:
→ **Analyze**: Sample a small portion of a column to determine the best encoding scheme
→ **Compress**: Encode the values and write it to disk.
DuckDB: Storage Format

DBMS's built-in storage format maintains a single PAX-oriented file:

→ Splits tables into row groups with 120k tuples.
→ On-disk encoding is different than in memory representation.

Two phase compression:

→ Analyze: Sample a small portion of a column to determine the best encoding scheme.
→ Compress: Encode the values and write it to disk.

<table>
<thead>
<tr>
<th>Version</th>
<th>Taxi</th>
<th>On-Time</th>
<th>Lineitem</th>
<th>Notes</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>DuckDB v0.2.8</td>
<td>15.3GB</td>
<td>1.73GB</td>
<td>0.85GB</td>
<td>Uncompressed</td>
<td>July 2021</td>
</tr>
<tr>
<td>DuckDB v0.2.9</td>
<td>11.2GB</td>
<td>1.25GB</td>
<td>0.79GB</td>
<td>RLE + Constant</td>
<td>September 2021</td>
</tr>
<tr>
<td>DuckDB v0.3.2</td>
<td>10.8GB</td>
<td>0.98GB</td>
<td>0.56GB</td>
<td>Bitpacking</td>
<td>February 2022</td>
</tr>
<tr>
<td>DuckDB v0.3.3</td>
<td>6.9GB</td>
<td>0.23GB</td>
<td>0.32GB</td>
<td>Dictionary</td>
<td>April 2022</td>
</tr>
<tr>
<td>DuckDB v0.5.0</td>
<td>6.6GB</td>
<td>0.21GB</td>
<td>0.29GB</td>
<td>FOR</td>
<td>September 2022</td>
</tr>
<tr>
<td>DuckDB v0.6.0</td>
<td>4.8GB</td>
<td>0.21GB</td>
<td>0.17GB</td>
<td>FSST + Chimp</td>
<td>October 2022</td>
</tr>
<tr>
<td>CSV</td>
<td>17.0GB</td>
<td>1.11GB</td>
<td>0.72GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parquet (Uncompressed)</td>
<td>4.5GB</td>
<td>0.12GB</td>
<td>0.31GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parquet (Snappy)</td>
<td>3.2GB</td>
<td>0.11GB</td>
<td>0.18GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parquet (ZSTD)</td>
<td>2.6GB</td>
<td>0.08GB</td>
<td>0.15GB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The DBMS can also access external data files via extensions.

→ Parquet, Arrow, SQLite, JSON,

Can also install extensions to retrieve files from remote filesystems (HTTP, S3)
Cloud-based service that provides automatic execution of DuckDB queries on serverless compute nodes.

→ Remote nodes are DuckDB instances running inside of containers and connected to object stores.

→ Exposes remote catalog to local instance.

The latest versions of DuckDB already include extension to connect to MotherDuck.
Cloud-based service that provides automatic execution of DuckDB queries on serverless compute nodes.

Remote nodes are DuckDB instances running inside of containers and connected to object stores.

Exposes remote catalog to local instance.

The latest versions of DuckDB already include an extension to connect to MotherDuck.

Source: MotherDuck
Introduces a new "bridge" operators that passes tuple streams between local and remote DuckDB instances.
→ Leverages operator pausing feature that DuckDB added from switching to push-based execution.

Query optimization occurs on the local instance as normal and then uses cost-based rules to decide what to run locally vs. remote.
DuckDB is brilliant and its adoption is enviable. → Right place. Right time. Right problem.

Andy bet his earlier research agenda wrongly on in-memory DBMSs.

This is what HyPer/Umbra could have become if they were open-source...
DuckDB is brilliant and its adoption is enviable. 
→ Right place. Right time. Right problem.

Andy bet his earlier research agenda wrongly on in-memory DBMSs.

This is what HyPer/Umbra could have become if they were open-source...
NEXT CLASS

Yellowbrick