Ephemerality of DuckDB, SQLite and traditional RDBMSes

https://chatgpt.com/share/6ae4ba98-c59c-83e9-aa83-d771f1dfb59d

> Is it weird to think of DuckDB databases as more ephemeral than even SQLIte databases? Both of them are more ephemeral than traditional RDBMS databases with a dedicated server and high structure vs just a file. But even comparing SQLite to DuckDB I think of DuckDB as more fleeting - probably because it is SO easy to ingest that I think nothing of tossing the DB aside because I can reload from JSON, CSV or parquet so trivially.

No, I think that's a perfectly reasonable distinction. In fact, you're identifying something important: the perceived permanence of a database depends not just on its architecture, but on the cost of reconstructing it.

I'd distinguish between technical ephemerality and practical ephemerality.

Three levels of perceived permanence

Characteristic	Traditional RDBMS (SQL Server/Postgres)	SQLite	DuckDB
Typical role	System of record	Embedded application database	Analytical processing engine
Storage	Managed database files	Single database file	Single database file or in-memory
Data ingestion	Often deliberate ETL/ELT	Usually application-driven inserts	Extremely easy bulk ingestion

Characteristic	Traditional RDBMS (SQL Server/Postgres)	SQLite	DuckDB
Data reconstruction	Potentially expensive or impossible	Depends on application	Often trivial from source files
Typical data lifespan	Years	Months to years	Minutes to years
Mental model	Persistent asset	Portable asset	Disposable analytical workspace

These are conventions rather than architectural limitations. All three can persist data indefinitely.

Why DuckDB feels more ephemeral

I think there are three reinforcing reasons.

1. DuckDB makes reconstruction exceptionally cheap.

Consider this:

```
CREATE TABLE sales AS
SELECT *
FROM read_parquet('sales/*.parquet');
```

If that table is merely a materialized copy of the source Parquet files, deleting it loses very little.

You haven't destroyed irreplaceable data. You've destroyed a reproducible representation of that data.

Contrast that with a SQLite database containing user-entered information. Losing the database might mean losing the only authoritative copy of those records.

2. DuckDB blurs the distinction between querying a file and querying a database.

You can query Parquet directly:

```
SELECT customer_id, SUM(amount)
FROM read_parquet('sales/*.parquet')
GROUP BY customer_id;
```

There's no requirement to create a persistent database at all.

This makes the database itself feel optional.

3. SQLite and DuckDB have different cultural identities.

SQLite is frequently the destination for data.

DuckDB is frequently the vehicle for processing data.

With SQLite, you often build an application around the database. With DuckDB, you often build a database around the analysis.

That's a subtle but meaningful inversion.

The deeper concept: data versus derived state

Your intuition connects to a fundamental systems-design distinction:

- Authoritative state: Information that cannot simply be reconstructed from another source.
- Derived state: Information that can be regenerated from authoritative sources.

A DuckDB database populated from CSV, JSON, or Parquet is frequently derived state. It's conceptually closer to a cache or materialized view than a traditional database of record.

This resembles the philosophy behind reproducible builds and infrastructure as code: if the inputs and transformation logic are preserved, the resulting artifact becomes disposable.

There is an important caveat: the reconstruction must actually be deterministic, and the source files must remain available. If you've cleaned data manually, modified tables, or built transformations that aren't saved elsewhere, your DuckDB database may now contain authoritative

information.

An interesting implication

I'd argue that ephemerality is more a property of the workflow than of the database engine.

You can have an ephemeral SQL Server database, a mission-critical DuckDB database, or a SQLite database that exists for only five seconds.

But DuckDB's design encourages disposable workflows to an unusual degree.

And I'd take your observation one step further:

The easier data becomes to reconstruct, the less psychologically valuable its current materialization becomes.

In your recent work with Azure Blob metadata, for example, the source of truth is Azure Storage. Your rclone JSON files and DuckDB transformations into Parquet are reproducible derivatives. You have little reason to become attached to an intermediate DuckDB database.

There is almost an inversion of the traditional DBA mindset here. Historically, databases were precious and required careful preservation. In this workflow, the pipeline and source data are precious; the database is just a convenient, replaceable execution artifact.

That's a meaningful shift from database administration toward data engineering.

Revision #3
Created 30 October 2026 14:55:38 by Brian Dill
Updated 30 October 2026 15:41:52 by Brian Dill