

ColdFront: Transparent PostgreSQL Data Tiering to Apache Iceberg

Jimmy Angelakos
Staff Software Engineer, pgEdge

PostgreSQL Edinburgh User Group
2026-09-24



What is this about?

- Why build ColdFront
- How it works
- A cold tier you can write to (yes, really)
- New: adopt the Iceberg tables you already have
- The hard part: multi-writer correctness
- New: vector search on cold data
- Live demo

About Jimmy Angelakos

- Based in Edinburgh, Scotland
- Involved in the Open Source Community for 25+ years
- PostgreSQL Significant Contributor
- Organizer, PostgreSQL Edinburgh User Group
- Member, PostgreSQL Europe Diversity Committee
- Author, [PostgreSQL Mistakes and How to Avoid Them](#)
- Co-author, [PostgreSQL 16 Administration Cookbook](#)



Why build this?

- Cold data sits on expensive SSDs
 - Nobody has queried it in months... but it can't leave
- Open format: Apache Iceberg
 - Any engine can read and write it, no lock-in
- Existing options:
 - Proprietary services
 - Or many components to keep alive
- The pieces existed but they weren't assembled
 - So we wrote the assembly, and kept it slim

Design philosophy: KISS / UNIX

- Few moving parts, each doing what it does best:
 - Postgres: your tables, your SQL
 - pg_duckdb: Iceberg I/O, in-process
 - Lakekeeper: Iceberg REST catalog
 - Any object store: S3, Azure, GCS, on-prem
- ColdFront is the orchestrator between them
- Both tiers are queryable and inspectable
 - No opaque storage engine

How does it work?

- Your app queries one relation, same as before
- A view performs the UNION of hot + cold
- The extension directs each read/write to the right tier
 - Hot: Native PG range partitions
 - Cold: Iceberg (Parquet) on object storage, via pg_duckdb
- The archiver moves old partitions hot → cold on a schedule
- Stock PostgreSQL 16+

Two modes, one SQL interface

Tiered

- Recent rows in PG
- Older rows in Iceberg

Decoupled

- Iceberg from row one
- PG only holds a thin view
- Every write goes to Iceberg
- Scale out: just add PG nodes

The cold tier is writable

- Same SQL, no application changes
- Data modification/deletion request?
- Reaches Parquet files that are months/years old
- No dump/restore song-and-dance
- DELETE/UPDATE work on cold data too

Inside the extension

- One C hook sees every statement
 - Not a ColdFront registered table? Fast path out
- First cold query: The catalog attaches itself
 - Once per session, zero setup boilerplate
- DML on the view → one `duckdb.raw_query()` → one Iceberg snapshot
 - No INSTEAD OF triggers
 - No HTTP calls of our own
- INSERT...SELECT streams rows straight into the Iceberg writer
- ROLLBACK works: DuckDB's transaction is tied to Postgres's

Distributed correctness

The multi-writer problem

- Many PG nodes but:
 - One Iceberg catalog
 - One storage bucket
- An Iceberg commit is a conditional update of a metadata pointer
- Two writers commit at once
 - One gets a 409/Conflict HTTP error
- "Just retry in the application"
 - Err... no thanks
- Goal: never collide in the first place

The bakery protocol

- Lamport's bakery + Ricart-Agrawala deferred replies
- Every cold write takes a *claim* on the table first
 - Take a ticket (Snowflake ID), wait your turn
- Claims and ACKs are plain rows, replicated by Spock
- Commit when your ticket is the lowest *and* every live peer has ACKed
 - Peer with a lower ticket? It ACKs after its own commit
 - Dead peer? Liveness check makes it count as ACKed
 - Crashed writer? Its claim is reaped: no timeouts, no daemon
- You hold the claim → nobody else commits → no conflict, no 409

The DuckDB patch: Async Parquet upload

- Parquet upload is slow but commit is a short POST
- Stock: upload + commit inside the ticket
 - Writers queue on each other's uploads. Ouch.
- One duckdb-iceberg patch: upload first, ticket only for the commit
 - Commit refreshes table metadata inside the ticket → no stale parent
- ~2.6x contended write throughput
- Patch missing? Falls back to claim-first. Never a silent 409.

Trust nothing: verify with TLA+

- Distributed correctness is hard to intuit
 - Even for people who do this for a living
 - So we modeled the protocol in PlusCal/TLA+
- TLC explores *every* interleaving of N writers
 - Including crashes the CI can't easily reproduce
- The model caught problems code inspection missed
- Sanity check: 5 deliberately broken configs, all caught
- In the repo: docs/formal/

Okay, but small files pile up

- Every cold write → one Parquet file + one snapshot
 - Busy table → tens of thousands of tiny files
 - Ouch
- Lakekeeper is a catalog: it does no maintenance
- Compactor: one Go binary, run from `cron`
 - Merges small files into ~128 MiB ones
 - Expires old snapshots and frees their files
 - Deletes orphan files
 - Takes the same bakery ticket as a cold write → no 409s

Already use Iceberg?

- Spark, Trino, whatever
- `adopt_iceberg_table()`: view + registry row
 - Schema comes from the catalog, nothing provisioned
- Read-only by default
 - `p_writable` → `true` enables INSERT/UPDATE/DELETE
- `release_iceberg_table()` unregisters it from ColdFront
- Active-Active: one node adopts, replicated to peers with Spock
- Other engines still writing to it? Outside the bakery
- Merged today (yes, really)

Vector search on cold data

IVF without building an index

How?

- `pgvector` columns tier like any other column
- Centroids in a tiny PG table
- Each Parquet row has cluster ID
- Files sorted by cluster
 - Row-group stats act as inverted lists
- DuckDB pushdown skips the rest

And...

- A probe is just a WHERE clause
 - ~93% recall
 - ~54x faster (10M rows, 1024 dims)
- Probe everything: provably exact
- No index to build & maintain

What's next?

- Release candidate in the next couple of days
 - Then 1.0.0
- Read and write performance
- Smarter tiering, partitioning, pruning
- What we want to hear:
 - Does this fit your use case?
 - Can you spot a missing feature?

Where to find it

- github.com/pgEdge/ColdFront
- Open source, PostgreSQL Licence
- Beta maturity level, almost GA
- Docs, guided walkthrough, and TLA+ model are all in the repo

