

# Ex Falso SQL Quodlibet: Excluding the Middle

Chris Douglas

UC Berkeley

Berkeley, USA

## 1 Et Tu, Parquet?

As table formats have gained traction in “lakehouse” architectures, the next order of magnitude in transaction throughput has motivated development of *catalog services*. These services improve performance, but they mistake what is unique about table formats and undermine what makes them compelling. Table formats are not a means of locating a set of Parquet files in an object store, but rather a design that forces *transaction eversion*<sup>1</sup>: internal details like table schemata, views, access control, and concurrency control are externalized in storage, concurrently modified, and jointly owned by multiple transaction engines. Lifting these functions back into a service is just a data warehouse with more steps.

The only future where table formats are not an interchange format for data warehouses is one that compels transactions to digest transactions externally. We sketch an alternative to catalog services that addresses the performance, compatibility, and access control requirements by defining extensible formats for table catalogs.

## 2 In Repositorio Veritas

Transaction eversion entails:

**Truth in Storage** Direct access to the object store inherits high durability, availability, and scalability of the underlying store, but loses the ability to interpose between the transaction and the data to restrict visibility.

**ABI Compatibility** When readers and writers self-certify invariants in storage, backward and forward compatibility cannot be patched up by an active process.

**Invariants in Storage.** To support multiple transaction systems (everted federation), any invariant that precludes reads or writes must be encoded in the format.

## 3 Bernstein Aere Perennius

For a table format that endures, we make two changes. First, we define a format for lakehouse updates designed for high-throughput, multi-writer workloads. Second, we encode invariants in the format to ensure the everted execution of a transaction is verifiable.

The update format takes inspiration from the optimistic transaction layer in Hyder [1]. Writers execute transactions by appending *intention records* to a shared log. Entries ordered by the log are sequentially and conditionally applied to the shared database: if versioned read/write dependencies encoded in the record match the current state at that position, then their effects are applied. An intention record need only encode dependency versions for entities read (matched,

not updated) and written (matched, incremented when applied). The format of an intention record is fixed; how it is propagated and applied is a deployment question.

The catalog format encodes not only metadata, but also the rules governing its evolution: a set of versioned entities, derived rules, and invariants that determine whether an intention record applies. Policy rules determine the obligations of an intention record. Satisfying obligations may require the client to create a *witness*— a verifier-checkable function of catalog state and the proposed update— serialized with the intention record sufficient for validation. The policy layer can use existing, well-understood trust management languages [2, 3] for decidable, terminating evaluation with controlled expressiveness.

Validation of the witness depends on the operability requirements of the deployment. In settings with trusted engines, validation may only require that the engine affirm it understands and has satisfied the requirements in the catalog. If the engine is untrusted, intention records may be submitted to a service for explicit validation (e.g., OneHouse). Critically, policy predicates and invariants are also updated as intention records.

Consider two examples. First, access control is configured independently in each engine, using engine-specific mechanisms. Encoding these policies in the catalog— or in bootstrap configuration for an authorization service— allows engines to enforce them consistently and independently. Second, support for format features is uneven across engines. Avoiding unsupported features while requiring optional ones is manual and error-prone; encoding these requirements in the catalog allows intention records to affirm them before validation and integration with the catalog.

When catalog services centralize *enforcement* of policy, engines integrate with the catalog, not the format. The engine must also be trusted by the catalog service provider, which is a non-starter in managed deployments [4, 5]. In contrast, this approach makes invariants explicit in the catalog itself so admissibility is determined by replaying storage state rather than integrating with a particular service implementation. This does not remove complexity, but it makes it uniform, verifiable, and jointly owned. The policy layer need not be novel: it can be instantiated using existing trust management frameworks with restricted, decidable semantics. The alternative is not simplicity, but unverified duplication.

Single-writer acceleration of table formats is an inefficient, path-dependent route from table formats to a data warehouse. To support the multi-writer case, table formats must not only encode metadata useful for locating files on read, but also the rules that validate updates for multi-writers.

<sup>1</sup>Starfish and ribbon worms digest prey externally; generally *gastric eversion*.