

SQLodin Working Group

Request for Discussion and Implementation Record

SOD 0004

Architecture and Implementation Plan

Production SQL and Durable Throughput

STATE

COMMITTED

CREATED

2026-09-22

LAST UPDATED

2026-09-25

INTENDED STATUS

Committed

AUTHORS

Vikrant Rathore, with assistance from Ronak Rathore

DISCUSSION FORUM

Durable host tradeoffs, mixed SQL and scoped qualification

TOPICS & LABELS

storageperformancemulti-mastercorrectness

Status of This Record

This document is a **SQLodin Discussion (SOD)** record authored in Typst and tracked in git. It follows an RFC-style specification structure so that architectural scope, rationale, trade-offs, proof obligations, and operational boundaries remain explicit. Documents using the placeholder number XXXXX are provisional drafts. Numbered SOD documents are permanent engineering records for discussion on improvement, architecture, and enhancements of SQLodin.

Contents

1	Abstract	2
2	Status and Implementation Boundary	2
3	Introduction	2
4	Terminology and Scope	3
5	Problem Statement	3
6	Goals and Non-Goals	3
6.1	Goals	3
6.2	Non-Goals	3
6.3	Performance objectives and acceptance decision	3
7	Design Overview	3
8	Detailed Design	4
8.1	P1: deterministic outcomes and retries	4
8.2	P2: grouping without changing transactions	4
8.3	P3: representation and CPU cost	5
8.4	P4: service and client semantics	5
8.5	P5: recovery and storage lifetime	5
9	Security & Correctness Considerations	5
10	Operational Considerations	6
11	Validation and Acceptance Gates	6
12	Alternatives Considered	6

13 Open Questions	6
14 Discussion and Revision Notes	7
14.1 22 September: correctness before speed	7
14.2 23 September: service and recovery boundary	7
14.3 24 September: progress and verification	7
14.4 25 September: closure and disclosed limits	7
14.5 25 September (later): follow-up in SOD 0005	7
14.6 Historical cost attribution supporting the batching decision	7
14.6.1 Controlled Linux Attribution	7
14.6.2 Mechanism and Cost Model	8
15 References	8

1 Abstract

Build the durable SQL host around the complete pinned paxos-odin library. Preserve agreement, retry identity and SQL semantics while reducing synchronization and copying costs. The chosen mechanisms are bounded group commit, concurrent replica I/O, deterministic transaction outcomes, fresh read markers and certified recovery. This record explains their tradeoffs and the accepted release scope. It does not claim that an efficient language makes durable replication inexpensive.

SOD stands for **SQLODIN Discussions** — versioned engineering records for discussion on improvement, architecture, durable storage design, and throughput optimization for SQLodin.

2 Status and Implementation Boundary

Committed; reviewed 25 September 2026. Promoted on 22 September as the accepted design. The fixed three-voter correctness qualification is complete under the owner's explicit acceptance decisions. The 23 closure criteria, tested source and binaries are bound in docs/releases/2026-09-25.typ and benchmarks/results/verification-20260924/release-decision.json. This record remains Committed, not frozen as Published. A qualified scope is not unrestricted production suitability, a performance pass or evidence of physical failure-domain independence.

Design area	Current disposition
P0: attribution	Durable cost attribution and mixed-workload measurements retained.
P1: SQL outcomes	Policy 9, durable outcomes, retry epochs and optimistic validation implemented.
P2: batching	Bounded journal/application groups and fair owner service implemented.
P3: representation	Lossless journal/wire packing and statement reuse implemented; further copy reduction remains an optimization.
P4: service	Native mTLS, fresh reads, CLI, Python and SQLAlchemy implemented. Membership remains fixed.
P5: recovery	Format-5 generations, certified images, retention, backup/restore and offline migration implemented.
P6: qualification	Correctness scope accepted; throughput and latency shortfalls disclosed.

3 Introduction

The early disk-backed path achieved roughly 22 writes/s in its measured environment. The useful question was how much durable work each acknowledged transaction required. Serial voter execution, repeated sync barriers and large copied values explained costs that instruction-level tuning alone could not remove. The historical attribution below records the basis for this design.

4 Terminology and Scope

A transaction is a bounded SQL request with typed parameters and durable retry identity. Journal durability, chosen progress, application progress and response delivery are distinct frontiers. A timeout after admission can mean an unknown outcome. It does not cancel a chosen request. The qualified scope uses three fixed authenticated voters with compatible Linux x86_64 builds.

5 Problem Statement

Independent durable requests cannot simply be merged into one SQL transaction: deferred constraints and rollback conflict actions can change their outcomes. Nor can a sync be removed because another stage has completed. The host must amortize work while preserving the reference execution and must keep recovery, peer service and client admission from starving one another.

6 Goals and Non-Goals

6.1 Goals

Preserve durable acknowledgement and serializable bounded transactions under retries and one-voter loss. Bound queues and retained consensus history. Make recovery publication crash-safe. Measure mixed transactional SQL against a matched durable SQLite baseline with verified outcomes.

6.2 Non-Goals

No unrestricted SQL, dynamic voter changes, rolling mixed-version upgrade, cross-shard transaction, Byzantine guarantee or arbitrary-query latency bound. Five-voter qualification and independent physical failure domains are not inferred from the three-instance evidence.

6.3 Performance objectives and acceptance decision

The original provisional objectives remain unchanged: 3,000 transactions/s for a 70/30 mix (including 900 successful writes/s), 1,000 pure writes/s, p99 reads/writes of 20/50 ms and at least 25% of matched SQLite throughput. The short-query RSS objective is 512 MiB per voter, separately from OS page cache. The recovery aspiration was 60 s with a local certified image and at most 1 GiB replay tail on qualified hardware; it is not a database-size-independent guarantee.

DECISION — Owner Acceptance Decision (25 September 2026)

The owner accepted release after correctness qualification with performance shortfalls disclosed. Throughput and p99 objectives are future improvement goals, not first-release blockers. Slow retained-store startup observations of 114/137 s remain disclosed. The owner also removed large-capacity growth campaigns as requirements. On 24 September mandatory soak durations were replaced by formal arguments and targeted fault tests.

7 Design Overview

The durable SQL host replaces the historical serialized pipeline with bounded group commit, concurrent replica I/O, and decoupled SQLite application.

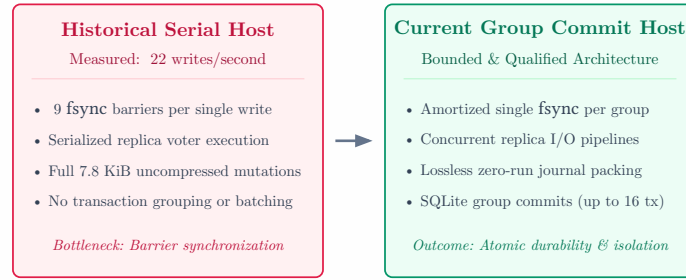


Figure 1: Architectural evolution: historical serialized path vs bounded group commit pipeline.

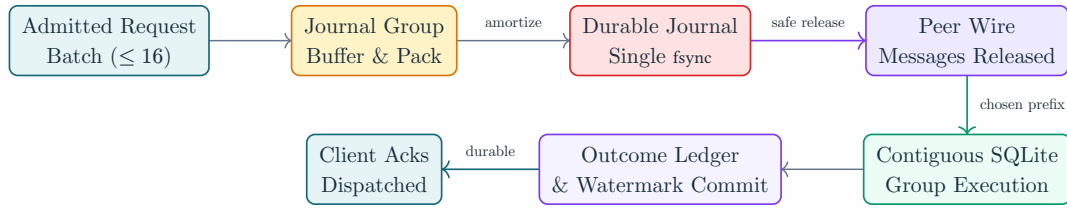


Figure 2: Bounded journal group and application pipeline.

One owner serializes protocol state. Workers receive owned work and return checked completions; they do not mutate consensus state concurrently. Replica processes can perform I/O independently. A completion releases only the effects covered by its durable sequence. Backpressure applies before admission where possible; accepted work retains its identity until its outcome is known.

8 Detailed Design

8.1 P1: deterministic outcomes and retries

SQL policy 9 constrains functions, schema behavior, row identifiers and extension use. Expected constraint failures become durable request outcomes. Unknown storage or execution failures stop application; inventing a rejection would allow replicas to diverge. Data, outcome, retry state and applied watermark commit together. A session/sequence identifies immutable request content. Explicit epoch retirement permits bounded outcome storage without permitting old effects to recur. The default session capacity is 65,536. Full semantics live in `specs/sql-policy.typ` and `specs/session-retirement.typ`.

8.2 P2: grouping without changing transactions

Journal groups hold owned effect copies and release dependent work only after a checked durable completion. Application groups contain at most 16 ordered requests. Savepoints isolate requests; deferred foreign-key checks run at each logical boundary. A whole-transaction ROLLBACK takes the individual-commit fallback before any response.

INVARIANT – Equivalence to Individual-Commit Reference ($G_i = R_i$)

For every admitted transaction i in an application group of size ≤ 16 , the observable database state D_i and returned outcome O_i must be identical to executing request i in an isolated individual commit:

$$G_i = R_i \quad \text{for all } i \in [1, |\text{group}|]$$

If any savepoint fails or constraint violation occurs, the group rolls back to the individual-commit fallback before acknowledging the client.

The owner services bounded batches and peer work fairly. The protocol window is 64 slots and peer bursts are bounded to eight. Private generation catch-up applies at most one chosen SQL transaction per owner turn, with journal-copy limits of 128 records and 1 MiB. These limits bound scheduled work; they do not preempt a SQLite statement or blocked kernel I/O.

8.3 P3: representation and CPU cost

Lossless zero-run packing reduces journal and peer payload bytes while preserving the complete Paxos value, including unused fields and floating-point bits used by equality. Shared bounded vector storage avoids reserving a full vector for every column. Statement reuse reduces preparation work. Fixed-capacity storage still has a copy and cache cost. A new representation must preserve value identity and pass codec/recovery tests before performance measurements can justify it.

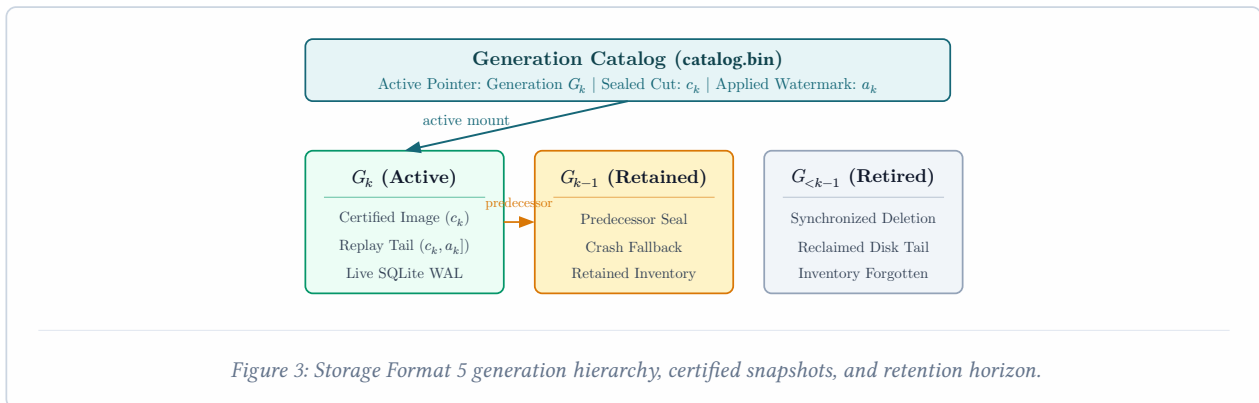
8.4 P4: service and client semantics

Native mTLS authenticates fixed peer and client roles. The service rejects incompatible identities and versions. Admission reserves peer capacity: at most 32 connections, 24 clients and four pending handshakes. Queues hold at most 64 frames or 2 MiB. Client frames are bounded to 64 KiB; responses to 1 MiB. Internal queries have separate row, byte and execution budgets.

Fresh reads use closed cohorts and ordered markers. Interactive transactions preview privately, then validate a database-wide revision at ordered commit. This covers predicate conflicts but can reject disjoint transactions and repeats some SQL work. CLI and Python expose conflict and unknown outcome recovery. Retry with the same request identity after an uncertain response; a new identity can execute a second transaction.

8.5 P5: recovery and storage lifetime

Format 5 separates application state from voter-local consensus state. A certified image names an agreed prefix; its seal and retained suffix connect it to later progress. A private generation becomes durable before catalog publication. Retirement protects the active generation and its exact predecessor, checks ownership, and synchronizes deletion before forgetting inventory.



Automatic maintenance starts at a 256 MiB tail or 15 minutes of dirty history. Transfer uses 1 MiB chunks and a 32 MiB buffer budget. An 8 GiB retained-history cap and free-space checks exert backpressure. These are resource policies, not a maximum database size. Snapshot and generation work must keep up with admitted writes. The single maintenance worker has a 3,600 s budget.

Verified backups support fenced restore into a globally unused namespace. Offline migration accepts the reviewed format-4 policies 6/7/8; sources are preserved. Coordinated certificate renewal is supported. Hot reload, rolling upgrade and live voter enrollment are outside this qualification.

9 Security & Correctness Considerations

SOD 0003 states the composition assumptions. Authentication is not Byzantine tolerance. Storage must honor synchronization, identities must not be reused, and admitted SQL must execute under

compatible builds. Recovery must not silently create an empty voter when required state is absent. Backups, migration and restore must preserve or deliberately fence retry namespaces.

10 Operational Considerations

Keep peer service available under client overload and snapshot catch-up. Treat admission rejection as distinct from an uncertain accepted request. Observe lag, retained bytes, free space, queue depth, RSS and storage latency. A memory bound on one queue is not a bound on SQLite or the page cache. Deployment procedures belong in the existing guides; precise resource and recovery contracts remain in `specs/resource-contract.typ` and `specs/recovery-bootstrap.typ`.

11 Validation and Acceptance Gates

Correctness qualification combines the 59 model configurations and 48 induction obligations with 163 tests in each of the Linux debug, optimized and individual-commit configurations. It includes 43 current-source crash boundaries, three live generation cycles, 38 native mTLS checks across three Linux instances, local CLI checks and isolated Python wheel checks. Evidence and candidate hashes are centralized in the release record rather than copied into each design section.

One-voter loss probes restored first-write service in 2.37, 1.24 and 2.48 s in the recorded run. These samples are not an unconditional deadline. Storage-fault tests cover ENOSPC, short writes and sync EIO. Process kills do not certify device power-loss behavior. Formal assumptions and implementation fault boundaries define the scope; there is no new duration or capacity gate.

The Linux mixed-workload matrix completed 32 cases and 91,840 verified operations. At 32 clients, 256-byte values and a 70/30 mix, it measured 152.3 transactions/s and 46.4 writes/s against 2,971.8 SQLite transactions/s (5.13%). Read/write p99 was 397.1/471.8 ms. The pure-write case measured 73.6 writes/s against 2,845 for SQLite, with p99 1,230.8 ms. This matrix predates the final maintenance-budget and replay-turn correction; it is not a final-binary performance measurement. It does not meet the objectives. A future performance claim requires a new matched measurement.

12 Alternatives Considered

Alternative	Pros	Primary Reason for Rejection
Disabling fsync barriers	Dramatically higher raw IOPS	Violates durability guarantee; data lost on node power cut
EPaxos Dynamic Proposals	Eliminates slot gaps	Predicate and SQL join dependencies trigger state space explosion
Independent Raft Groups	Parallel multi-shard writers	Requires 2PC cross-group coordinator; out of fixed-voter scope
Row-Only Optimistic Validation	Reduces conflict aborts	Fails to isolate SQL table predicates, foreign keys, and triggers

13 Open Questions

Further performance work should identify the current saturation point before choosing a mechanism: SQL execution, sync barriers, payload copies, contention or maintenance interference. Finer conflict validation and smaller in-memory values remain candidates. No alternate consensus algorithm is approved by this record. Dynamic membership and rolling upgrades require separate designs. These are future scope, not reopened conditions for the accepted fixed-voter correctness release.

14 Discussion and Revision Notes

14.1 22 September: correctness before speed

The first review reproduced a chosen uniqueness error blocking later application and raw random() producing divergent replica values. The response was durable per-request outcomes, rollback isolation and an enforced SQL policy. The dependency decision retained the complete upstream library. Early application and journal reviews supplied the durability-before-release and group/reference obligations now stated above.

14.2 23 September: service and recovery boundary

The initial durable embedding host became a native authenticated SQL service with fresh reads, bounded results, Python and optimistic ORM transactions. TLS used the pinned OpenSSL integration; an installed Odin core TLS package was not assumed. Early snapshots were primitives, not permission to trim. Certified seals, generation publication and fenced recovery completed that boundary later.

14.3 24 September: progress and verification

The failed native campaign had completed five surviving-quorum probes before a read timed out; the returning voter was still behind. That observation did not alone prove loss of majority service. The accepted response was to model bounded ownership progress and recovery, retain counterexamples, and add targeted regressions. Formal reasoning and implementation tests replaced mandatory soak durations as approval requirements.

14.4 25 September: closure and disclosed limits

Certified generations, image retirement, aggregate retention and final replay scheduling were qualified. The old multi-transaction replay loop fails its retained negative regression; the bounded loop passes. A 64 MiB current fixture restarted in 1.32 s. The owner accepted the performance shortfalls and closed the agreed scope without asserting arbitrary capacity or sustained throughput.

14.5 25 September (later): follow-up in SOD 0005

The open question above has an answer. Traces show up to seven **sequential** sync barriers per write and per fresh read, and eight-frame peer bursts limiting packets per turn. SOD 0005 adopts one barrier per service turn, Mencius-style no-op learning, voter-plus-owner value learning for three voters, a journal-backed WAL NORMAL application database and quorum-frontier reads. The queue bound quoted in P4 becomes 256 frames, with the 2 MiB byte bound unchanged. The 64-frame text and the measurements here describe the qualified release candidate.

14.6 Historical cost attribution supporting the batching decision

The following diagnostic describes the early serial host, not the current service. It is retained because it explains why the design prioritized barriers and amplification over language changes.

14.6.1 Controlled Linux Attribution

tools/profile_durability_cost.py runs the same pinned SQLite build on the same ZFS-backed Linux host. Each of three shuffled repetitions inserts 480 timed rows after 96 warmup rows, with a 256-byte payload. All resulting row counts and payloads are checked. Setup, warmup, verification and shutdown are outside the measured region. A single-threaded C interposer times real fsync/fdatasync and pwrite calls while forwarding every call unchanged. FULL durability is never disabled. Results come directly from benchmarks/results/linux-durability-cost.json.

Measured path	Rows/s	Syncs/row	Time in sync	pwrite KiB/row
SQLite FULL, 1 row/tx	346.21	1	94.53%	4.56
SQLite FULL, 32 rows/tx	7300.19	0.031	85.75%	0.64
SQLodin, 1 voter	117.63	2.038	86.61%	63.22

Measured path	Rows/s	Syncs/row	Time in sync	pwrite KiB/row
SQLodin, 3 serial voters	21.5	9.131	89.68%	223.9

These are instrumented small-write diagnostics, not mixed-SQL capacity results. The 32-row case changes transaction granularity; it illustrates amortization and is not a claim that independent client transactions already share commits. Aggregate pwrite bytes are bytes submitted to file writes across the measured process, not physical-device bytes, ZFS allocation or user payload size. Minor excess syncs include checkpoint/WAL lifecycle work. Short repetitions do not establish long-run distributions.

14.6.2 Mechanism and Cost Model

The historical `src/durable/host.odin:finish` path synchronously persisted each transition, applied its contiguous SQL prefix in another transaction, then released packets. In a healthy, three-voter, one-row path, each replica commonly flushes its vote, its chosen record and its application watermark separately: roughly nine process-wide barriers. The harness executes all replicas serially and drains them before the next request. This is more work than one local SQLite transaction, and it also hides the opportunity for independent disks and protocol stages to overlap.

The measured snapshot's fixed 7,800-byte mutation was copied and journaled in full, even for small requests. Later format 2 adds request identity; format 3 packs journal zero runs. This historical profile also exposes large file-write amplification.

For the measured serial path, a useful attribution is:

$$T \approx T_{\text{sync}} + T_{\text{file writes}} + T_{\text{other}}.$$

If sync duration and frequency remain unchanged, eliminating **all** non-sync work gives an optimistic speedup bound of $\frac{1}{p_{\text{sync}}}$, about 1.12 times here. Reducing barriers, write amplification or serialized waiting changes the bound. The language is valuable for predictable memory and CPU cost; architecture determines how much durable work must be performed.

15 References

- SOD 0002: architecture; SOD 0003: agreement and composition assumptions.
- `specs/grouped-sql.typ`, `specs/service-batching.typ`, `specs/resource-contract.typ`.
- `specs/generation-catalog.typ`, `specs/image-retirement.typ`, `specs/recovery-bootstrap.typ`.
- `docs/releases/2026-09-25.typ`: accepted scope and source-bound evidence.
- `benchmarks/results/verification-20260924/workload-matrix-linux.json` and `benchmarks/results/verification-20260924/workload-matrix-analysis-linux.json`.
- `benchmarks/results/linux-durability-cost.json`, `bench/durability_cost/`, `tools/profile_durability_cost.py`: historical sync attribution.