

SQLodin Working Group

Request for Discussion and Implementation Record

SOD 0002

Architectural Specification

SQLodin Architecture: Multi-Master Replicated SQLite Protocol

STATE

COMMITTED

CREATED

2026-09-22

LAST UPDATED

2026-09-25

INTENDED STATUS

Committed

AUTHORS

Vikrant Rathore, with assistance from Ronak Rathore

DISCUSSION FORUM

Fixed-voter SQL ordering, host boundaries and client semantics

TOPICS & LABELS

architecture

sqlite

multi-master

consensus

sqlite-vec

fts5

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	2
5	Problem Statement	2
6	Goals and Non-Goals	2
6.1	Goals	2
6.2	Non-Goals	3
7	Design Overview	3
8	Detailed Design	3
8.1	Rotating ownership and recovery	3
8.2	Logical transactions and SQL Policy 9	4
8.3	Reads and interactive transactions	4
8.4	Search and service boundary	4
9	Security & Correctness Considerations	5
10	Operational Considerations	5
11	Validation and Acceptance Gates	5
12	Alternatives Considered	5
13	Open Questions	5
14	Discussion and Revision Notes	6

15 References	6
---------------------	---

1 Abstract

SQLodin orders bounded SQL transactions through a fixed set of voters. Any voter can admit a write. Each replica applies the same chosen prefix to SQLite. This record explains why the system separates consensus, durable storage, SQL execution and client service. SOD 0003 states the proof obligations; SOD 0004 records the durable host and qualification decisions.

SOD stands for **SQLODIN Discussions** — versioned engineering records for discussion on improvement, architecture, and enhancements of SQLodin.

2 Status and Implementation Boundary

Committed; reviewed 25 September 2026. The accepted architecture is implemented and qualified for the fixed three-voter scope in `docs/releases/2026-09-25.typ`. This is not a frozen Published specification. Qualification does not establish arbitrary SQL support, live membership changes, Byzantine tolerance or increasing write throughput with each added replica.

The service uses store format 5, SQL policy 9 and peer wire 3. Native CLI and Python package versions are separate. The complete `paxos-odin` dependency is pinned at `c3d197016c1f938db23fdf7f1fe87fbdbb86ac1c`; SQLodin does not maintain a partial protocol copy.

DECISION — Complete Upstream Paxos Library Pin

SQLodin pins the complete upstream `paxos-odin` library rather than a partial protocol fork. The consensus state machine is completely pure: zero heap allocation during state transitions, zero direct disk or network I/O, and strictly bounded memory.

3 Introduction

SQLite supplies a transactional application store. Consensus supplies an order across machines. Neither supplies the other's guarantees. A chosen command can still be waiting for local SQL application. A local commit can still be unsafe to acknowledge if the ordering evidence was not made durable. The host must connect these boundaries explicitly.

4 Terminology and Scope

A **voter** persists promises and accepted values. A **slot** names one position in the ordered log. A value is **chosen** after a durable majority accepts it. The **applied prefix** is the contiguous sequence executed locally. An **owner** has the initial proposal right for a slot; it is not the sole writer for the database. Membership is fixed for the qualified configuration.

5 Problem Statement

Concurrent entry points must agree on one SQL history without forwarding every proposal to a standing leader. An idle or failed owner must not leave an unfillable hole. Crash recovery must preserve both the voting evidence and the application prefix. Client retries must not repeat an already executed effect.

6 Goals and Non-Goals

6.1 Goals

- Admit writes at any configured voter and preserve one agreed execution order.
- Keep consensus transitions deterministic, bounded and separate from I/O.

- Acknowledge only after quorum durability and local durable application.
- Offer explicit read consistency and bounded, durable retry identities.

6.2 Non-Goals

- Live voter enrollment, sharding, Byzantine agreement and arbitrary extensions are outside this design.
- Replication provides availability and local read placement; every replica executes the ordered writes through one local SQLite engine.

7 Design Overview

The SQLodin system architecture cleanly separates external client services, admission gating, pure consensus transitions, durable journal storage, and local SQLite application.

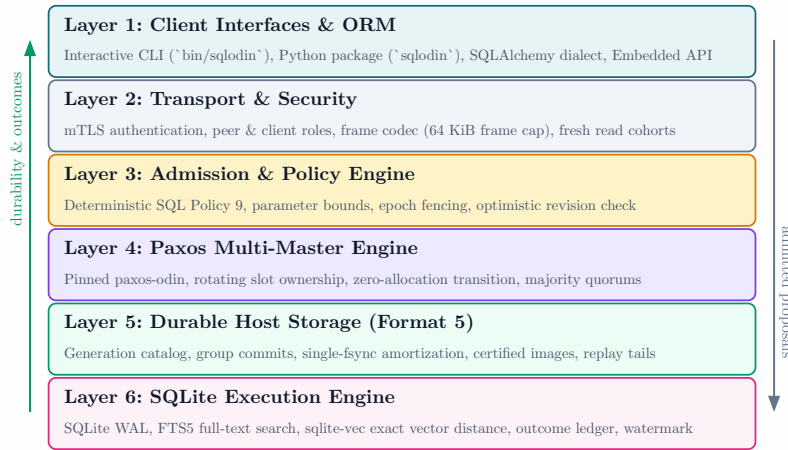


Figure 1: SQLodin layered architecture and subsystem boundaries.

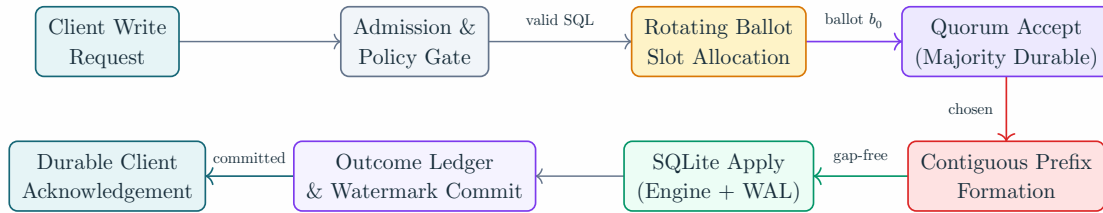


Figure 2: End-to-end transaction lifecycle from client submission to durable acknowledgement.

The upstream core emits effects into caller-owned storage. It does not perform disk or network I/O. The SQLodin host owns persistence, transport, application and responses. Explicit ownership keeps borrowed buffers from outliving their input and prevents workers from mutating protocol state.

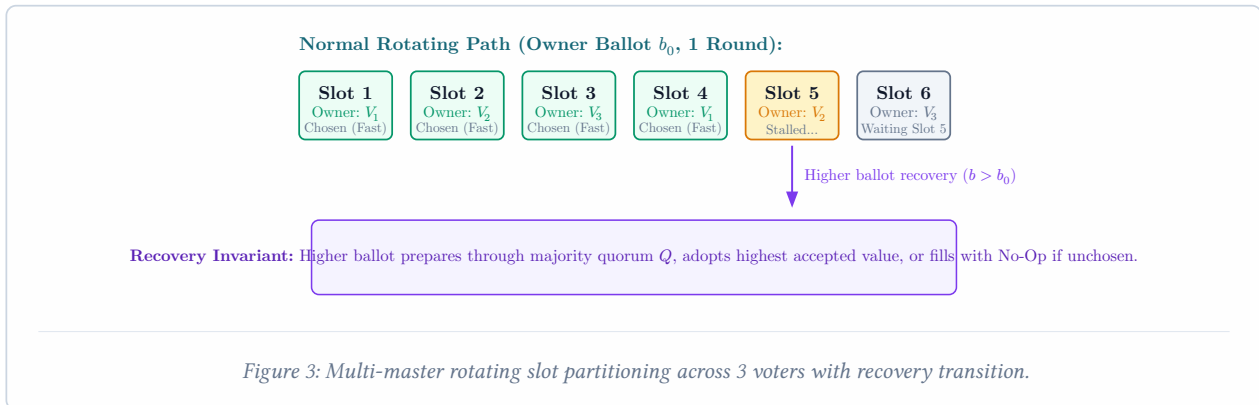
8 Detailed Design

8.1 Rotating ownership and recovery

For voters numbered 1 through N , slot ownership is defined by:

$$\text{owner}(s) = ((s - 1) \bmod N) + 1.$$

The reserved owner ballot may begin with Accept. A healthy, uncontested slot can be chosen in one quorum exchange, including its durable voting barriers. This is not end-to-end SQL latency.



Idle positions are closed with quorum-chosen no-ops. A stalled owner can be recovered through a higher-ballot prepare. Recovery must adopt the highest accepted value reported by that quorum; it may use a no-op only when the protocol permits it. Bounded demand frontiers and fair recovery service prevent idle slot selection from manufacturing an endless stream of new work.

8.2 Logical transactions and SQL Policy 9

Replicas agree on SQL and typed parameters, not independently modified SQLite pages. The durable SQL policy admits a constrained deterministic subset. A transaction carries a session, sequence, epoch and canonical request identity. The outcome and applied watermark commit with the data. Reusing an identity for different content is rejected. Retirement advances a durable epoch fence; old requests cannot silently become new requests after outcome rows are reclaimed.

INVARIANT — Deterministic Execution and Rollback Isolation

All replicas execute identical logical mutations through SQLite with Policy 9 constraints (disallowing non-deterministic functions such as `random()`, system clocks, or unversioned triggers). Every transaction commits its outcome atomically with the applied watermark.

Structured mutation helpers remain available. Generated Snowflake-style IDs are explicitly reserved and supplied by the caller; ordinary SQL inserts do not automatically use them. Unique node identities and durable reservation frontiers are required to prevent reuse after restart. Total ordering does not make raw SQL retries idempotent.

8.3 Reads and interactive transactions

Local reads use the replica's current snapshot. Watermark reads wait for a confirmed applied prefix. Fresh reads join a closed cohort before its ordered marker is allocated, then execute after that marker applies. Reusing an earlier marker would not establish freshness.

Interactive and ORM transactions use a private preview and an ordered commit that validates the database revision. Any intervening revision can cause a conflict. This conservative rule covers predicate reads, at the cost of conflicts even when two writes touch different rows. Rollback and savepoints operate within the bounded transaction contract.

8.4 Search and service boundary

FTS5 and scalar `sqlite-vec` distance functions support full-text and exact vector queries. The durable SQL policy excludes `vec0` virtual tables; extension availability is not permission to replicate every extension operation. Hybrid ranking combines query results under the chosen read contract. Exact distance scans do not promise indexed nearest-neighbor performance.

The native service authenticates configured peers and clients with mTLS and rejects incompatible identities and protocol versions. The CLI supplies interactive SQL and cluster operations. Python supplies

typed requests, search helpers and SQLAlchemy integration. The embedded API places transport and durable-effect obligations on its host.

9 Security & Correctness Considerations

BOUNDARY — Non-Byzantine and Failure Domain Assumptions

Authentication does not make a malicious voter safe. The model assumes non-Byzantine members, fixed membership, deterministic admitted SQL and storage that honors successful synchronization. SOD 0003 explains the composition. Unknown execution or storage failures stop progress; the host must not invent a deterministic SQL rejection or open an empty replacement store to continue.

10 Operational Considerations

Memory is bounded at admission, protocol windows, effect queues and result construction. SQLite, TLS and storage still consume memory outside the allocation-free consensus transition. The qualified build pins SQLite, sqlite-vec and OpenSSL; operating-system runtime libraries remain. Backup, restore, offline migration and coordinated certificate renewal have explicit procedures. Dynamic resizing and rolling upgrades require separate accepted designs.

11 Validation and Acceptance Gates

The release record binds the tested source, binaries and JSON evidence. Targeted tests exercise all-owner admission, one-voter loss, restart, retries, SQL conflicts, read freshness, malformed input and resource backpressure. Model checks and induction proofs address the abstract seams; implementation tests connect them to code. Neither test duration nor a throughput target proves agreement. Performance limitations are disclosed under SOD 0004.

12 Alternatives Considered

Alternative	Pros	Reason for Rejection
Standing Leader (Raft)	Simple admission without gaps	Bottlenecks all writes at one node; does not fulfill multi-master design
Independent Page Replication	Bypasses SQL parser	Concurrent SQLite page writes diverge without a mergeable history
Local Paxos Protocol Fork	Tailored local types	Duplicates upstream reasoning, proofs, and maintenance burden
Row-Level Optimistic Locks	Higher write concurrency	Fails to detect predicate, constraint, and table-wide trigger conflicts

13 Open Questions

No unresolved architecture question blocks the declared fixed-voter scope. Finer conflict validation, dynamic membership and sharding remain future design subjects, not hidden current features or new release conditions.

14 Discussion and Revision Notes

DECISION — 22 September 2026: Upstream Library and Atomic Application

Review found the partial upstream copy, unchecked application boundaries and ambiguous raw-SQL retry semantics. The integration proposal selected the complete library, atomic application watermarks, checked bindings and bounded shared vector storage.

DECISION — 23–25 September 2026: Durable Host & Native Service

The durable host, native service, fresh read markers and optimistic ORM transactions replaced the earlier volatile and watermark-only boundary. Replaced text ASCII sketches with formal CeTZ layer diagrams and Fletcher execution lifecycles.

DECISION — 25 September 2026: Revision under SOD 0005 (in discussion)

The service's fresh reads now cross a quorum frontier instead of a proposed marker, and one journal barrier covers each service turn. The separated application database is a WAL NORMAL cache of the FULL journal. Owners' round-zero no-ops, and with three voters values this voter has also voted for, are learned without waiting for the owner's Commit. The read-order contract above is unchanged; see SOD 0005 for proofs, models and measurements.

15 References

- SOD 0001: The SQLodin Discussion Process; SOD 0003: Proof obligations; SOD 0004: Durable host decisions.
- specs/multimaster-refinement.typ: Protocol-to-host composition and code map.
- specs/sql-policy.typ, specs/transaction-order.typ, specs/session-retirement.typ.
- docs/guides/network-service.typ, docs/guides/orm-transactions.typ.
- src/paxos.odin, src/durable/, deps/paxos-odin/: Adapter, host and complete protocol source.