

SQLodin Working Group

Request for Discussion and Implementation Record

SOD 0001

Process Memo

The SQLodin Discussion Process

STATE

COMMITTED

CREATED

2026-09-22

LAST UPDATED

2026-09-25

INTENDED STATUS

Committed

AUTHORS

Vikrant Rathore, with assistance from Ronak Rathore

DISCUSSION FORUM

The SOD process, the Zen of Odin for InsanAI, and its enforced structural constraints

TOPICS & LABELS

processdocumentationclistandards

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	1
2	Status and Implementation Boundary	2
3	Record Structure and Revision Rules	2
4	Introduction	2
5	The SOD Lifecycle	3
6	Numbering & Promotion Workflow	3
7	Registry and Compilation	3
8	The Zen of Odin for InsanAI	4
9	Structural Constraints	4
	9.1 Elm-style error handling and diagnostics	5
	9.2 Performance and longevity architecture	5
10	Documentation Policy and Editorial Guidance	5
11	Discussion and Revision Notes	5
12	References	5

1 Abstract

This document defines the **SQLodin Discussions (SOD)** RFC process, metadata schema, authoring lifecycle, and CLI tooling for sqlodin. Modeled directly after the Paxos Odin Discussions (POD) from paxos-odin and inspired by the ensodiscussions (EDS) architecture, SOD records serve as versioned architectural specifications, multi-master consensus derivations, and process memos for distributed SQLite

engineering. Their purpose is to explain decisions, alternatives, and unresolved questions, not to collect implementation logs.

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

2 Status and Implementation Boundary

Committed records may be corrected with a dated update; published records are frozen under the lifecycle defined below. Status describes a design decision, not a claim that every proposed feature ships. The registry and each record must agree.

SOD	State	Implementation or evidence boundary
0001	Committed	Active process, Typst editorial policy, and Zen structural limits.
0002	Committed	Implemented fixed-voter architecture, durable SQL, native service, fresh reads and bounded clients.
0003	Committed	Compositional protocol and host arguments, bounded models and selected inductive proofs; no whole-executable proof.
0004	Committed	Implemented fixed-voter host; correctness qualification accepted with disclosed performance shortfalls.

The registry is the index of accepted decisions, not an implementation checklist. Keep lifecycle state separate from implementation status. Committed means accepted; Published requires an explicit finalization decision and freezes the record. Passing tests alone does not publish a SOD.

3 Record Structure and Revision Rules

Use `docs/sod/template/rfc-template.typ` for engineering proposals. Keep these sections in order: Abstract; Status and Implementation Boundary; Introduction; Terminology and Scope; Problem Statement; Goals and Non-Goals; High-Level Architecture & Workflow; Detailed Design; Invariants & Correctness Guarantees; Security & Operational Considerations; Resource Budgets & Performance Targets; Validation and Acceptance Gates; Alternatives Considered; Open Questions; Discussion and Revision Notes; References. A process memo may use policy-specific sections, as this record does. Explain any inapplicable engineering section rather than silently losing its purpose.

The status section dates the review and distinguishes implemented behavior, proposed behavior, qualified scope, and known limits. Validation names the evidence and its revision boundary; it does not turn a hypothesis into an achieved guarantee. Alternatives explain the choice and its cost. Open questions identify unresolved decisions, with future scope separated from release blockers.

Discussion notes record the problem raised, the decision, its reason, and its consequence. Attribute owner decisions where known; do not invent reviewer consensus or meeting history. Fold historical development into the relevant SOD. Do not create separate progress diaries or historical document collections. Keep raw results in `benchmarks/results/` and protocol contracts and models in `specs/`. The existing release record binds a candidate's qualification; it is not another design proposal.

For a committed-record revision, update the current design in place, add a dated decision note and update the registry date and summary together. Preserve original measurement attribution and failed results. Do not append a new status that contradicts an obsolete "current" section above it. A materially different proposal should reference the decision it replaces. A superseded draft becomes Abandoned with a successor pointer; it does not need a number merely to close it. Published records are frozen: a replacement needs a new record and an explicit supersession reference.

4 Introduction

Distributed database libraries require uncompromising precision. Subtle design choices — such as multi-master slot partitioning, write-ahead durability order, logical mutation replication versus physical WAL

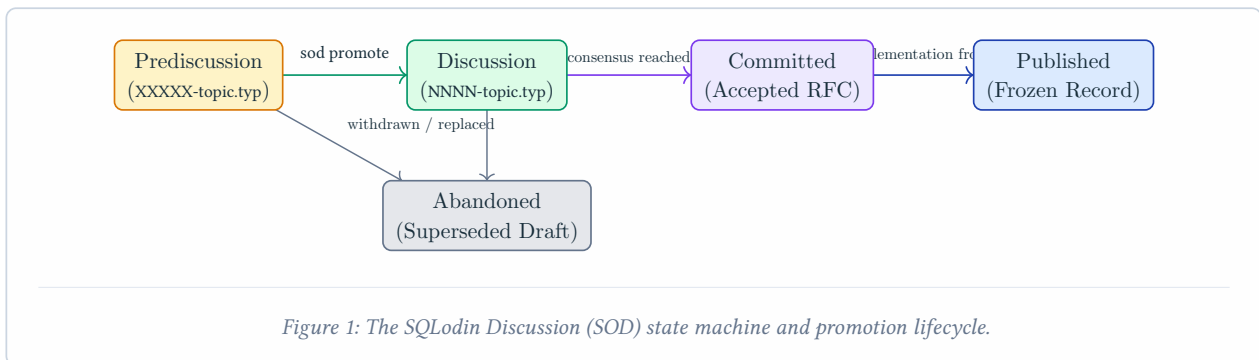
page capture, conflict-free primary keys, and vector search indexing — cannot be captured solely in inline source comments or transient issue tracker threads.

SOD provides a structured, versioned, Typst-rendered specification pipeline embedded in the repository.

5 The SOD Lifecycle

A SOD document progresses through five standardized states:

1. **Prediscussion (Draft):** A placeholder file named XXXXX-<slug>.typ created from docs/sod/template/rfc-template.typ. In this state, authors outline ideas and gather initial informal feedback.
2. **Discussion:** A numbered document (NNNN-<slug>.typ) actively reviewed by maintainers and contributors.
3. **Committed:** Consensus has been reached, the design is accepted, and implementation is scheduled or completed.
4. **Published:** The specification is finalized and frozen as a permanent reference.
5. **Abandoned:** The proposal was withdrawn or superseded by another record.



6 Numbering & Promotion Workflow

To prevent Git merge conflicts on sequence numbers across branches, new proposals begin with the placeholder XXXXX.

The sqlodin automation manages the entire lifecycle:

```

# 1. Create a new draft
./bin/sqlodin sod new multimaster-read-fences

# 2. List all active records and draft placeholders
./bin/sqlodin sod list

# 3. Promote the draft to the next permanent 4-digit number
./bin/sqlodin sod promote multimaster-read-fences

# 4. Compile PDFs via Typst
./bin/sqlodin docs sod
  
```

7 Registry and Compilation

Every promoted record is registered in docs/sod/registry.typ. The document suite is compiled to PDF using the installed typst binary:

- Individual records: docs/build/sod-NNNN-<slug>.pdf
- Master Index: docs/build/sod-index.pdf
- Complete Book: docs/build/sqlodin-book.pdf

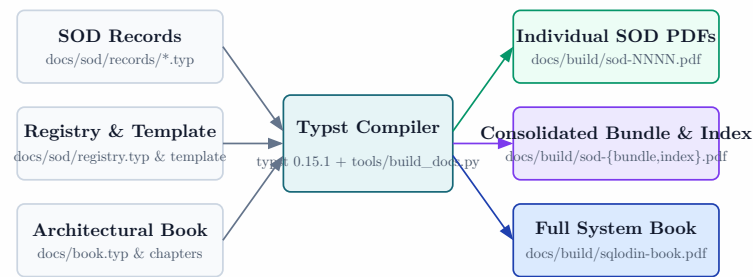


Figure 2: The SQLodin documentation build and artifact compilation pipeline.

8 The Zen of Odin for InsanAI

Every SOD, and every line of Odin in this repository, is written under one short creed. It is quoted in full so that a reviewer can point at the line a change violates.

*Data is real; code is just the stream.
 Explicit is better than a hidden scheme.
 Simple blocks beat abstractions built too high,
 A mere mortal should see how the segments tie.
 Keep close to the metal, let allocations show,
 Pass your contexts cleanly so the lifetimes flow.
 Errors are values, never cast aside,
 Handle them explicitly; let nothing hide.
 Fail with grace, let diagnostics guide:
 Show the break, the hint, the fix inside.
 Design for speed, let safety lead the pace,
 Waste no cycle, leave no leaking trace.
 Keep it simple to use, explain, and maintain,
 So years from now, the logic remains plain.
 Coherence beats purity when real problems strike,
 But structure your memory as hardware would like.*

9 Structural Constraints

The creed is enforced by `tools/check_style.py`, which make vet, make check, and the `sqlodin check` command run before anything else. A hard limit fails the build.

INVARIANT — Source Boundaries Enforced by CI

- **Maximum file length:** A single source file must not exceed 1,408 physical lines, including comments and blank lines. Modular separation is mandatory.
- **Line width bounds:** Soft limit at 99 columns; hard limit at 108 columns (tabs count as 4 columns). A line exceeding 108 columns fails the build.
- **Procedure code density:** The execution body of a procedure must not exceed 70 lines (excluding blank lines, comments, and divider lines).

9.1 Elm-style error handling and diagnostics

- **Actionable reporting:** An error does not merely state what failed; it explains why and provides an actionable path to resolution. In code this is the Error enum plus `explain_error`, a static table with one entry per value, and a test that fails when a value has no entry.
- **Diagnostic structure:** Every error block or runtime diagnostic carries three parts: the **context** (the failing input or state), the **hint** (the assumption or constraint that was breached), and the **remediation** (how to fix it).

9.2 Performance and longevity architecture

- **Resource-optimum design:** Memory layouts favour mechanical sympathy: contiguous arrays (the Ledger columns and bitmaps, inline small array effect buffers), predictable transformations, and no redundant heap allocations during consensus transitions.
- **Safety through visibility:** Performance never justifies unvetted cleverness. The library relies on Odin's type checking, explicit compile-time bounds (`#assert`), and runtime durability gates rather than implicit trust.
- **Direct owner admission:** The adapter enables upstream rotating ownership. Healthy slot choice can take one quorum round; durable SQL completion also waits for persistence and the contiguous prefix.

10 Documentation Policy and Editorial Guidance

Typst is the canonical format for project documentation. Markdown is reserved for GitHub-facing entry pages (`README.md`, `CONTRIBUTING.md`).

Keep teaching material in `docs/book/`, design and process records in `docs/sod/records/`. Benchmark drivers belong under `bench/`. Historical design discussion belongs in the relevant SOD; raw evidence remains separate. Temporary notes do not belong in the repository.

11 Discussion and Revision Notes

DECISION — 22 September 2026: Foundation

Established the numbered Typst discussion process (SOD), metadata schema, and structural code limits.

DECISION — 25 September 2026: Alignment with RFC Standards

Clarified the template, revision rules, and separation of lifecycle state from implementation qualification. Updated the shared SOD design system inspired by ensodiscussions (EDS), integrating Fletcher workflows and CeTZ architectural diagrams.

12 References

- POD 0001: Defines the upstream Paxos Odin discussion process.
- POD 0002: Records the pure state-machine architecture.
- The SQLodin engineering template is `docs/sod/template/rfc-template.typ`.
- ensodiscussions (EDS): Design and discussion records (`docs/ensodiscussions/`).