

SQLodin command-line client

Written by Vikrant Rathore with assistance from Ronak Rathore.

One self-contained executable provides a local SQLite shell, an authenticated cluster client, and the SQL service. Build it with `python3 tools/build_cli.py`.

```
sqlodin local notes.db                # standalone file, SQLite shell
sqlodin connect client.json           # interactive cluster SQL
sqlodin connect client.json --mode json -c 'SELECT id, name FROM customer LIMIT 20;'
sqlodin connect client.json --bail -f migration.sql
sqlodin serve node.json               # existing cluster voter
```

With no arguments, `sqlodin` opens the local in-memory shell. `sqlodin sql` is an alias for `sqlodin local`; it now prints actual results and returns SQL errors as nonzero exit codes. Toolchain commands remain available through `sqlodin help`.

Local files

`sqlodin local [DATABASE] [SQL] [OPTIONS]` embeds the **SQLite 3.51.3 shell** from the same pinned amalgamation as the engine. It does not invoke an installed `sqlite3` program. FTS5 and `sqlite-vec` are registered automatically.

Use `.help` to discover the commands compiled into this build, including `.open`, `.read`, `.import`, `.dump`, `.backup`, `.restore`, `.schema`, `.tables`, `.indexes`, `.parameter`, `.mode`, `.output`, and `.eqp`. Normal local SQLite transactions, savepoints and supported PRAGMAs apply. Dynamic extension loading is disabled by the self-contained build; optional SQLite features remain dependent on the documented compile flags. The local shell uses the upstream line-input implementation without an external readline library. Its SQL diagnostics retain SQLite's detail, with a correction hint when the shell exits unsuccessfully.

Local mode bypasses replication. Never open an active SQLodin voter database for modification through this interface. A standalone SQLite backup is not a backup of a cluster's consensus journal, durable sessions and application state. Use the cluster client for replicated writes.

Cluster connection

A client certificate must be authorized by the voters. Paths in this JSON are resolved from the process working directory; absolute paths are recommended. The identity is the expected DNS SAN of the server certificate, not a replacement for CA verification.

```
{
  "cluster": "accounts",
  "address": "10.175.52.19:7443",
  "identity": "node1.sqlodin.example",
  "certificate": "/secure/operator.pem",
  "key": "/secure/operator.key",
  "ca": "/secure/ca.pem"
}
```

The shell verifies mutual TLS and the response cluster/protocol. Reads default to a fresh quorum fence. Writes can enter through any voter. `.consistency local` explicitly permits stale standalone reads; transaction previews always use the service's revision-checked transaction path.

SQL ends with a semicolon. Statements can span lines, and quoted semicolons, comments and trigger bodies are recognized using the pinned SQLite completeness checker. Multiple complete statements on a line run in order. Without `BEGIN`, each write is a separate replicated transaction. A script is not automatically atomic: use `BEGIN/COMMIT` where the transaction fits the service limits.

On a capable terminal, the Odin editor supports left/right arrows, Home/End, Backspace/Delete, Ctrl-A/E, Ctrl-U/K, and up/down history. Tab completes a unique SQL keyword or dot-command prefix. History holds at most 100 lines in memory and is not written to disk. Ctrl-C clears the current input; Ctrl-D on an empty line exits. Ctrl-C during a synchronous request is handled when the request returns; `.timeout` bounds network waiting. Staged transaction work remains until commit, rollback or exit. `TERM=dumb` falls back to canonical line input.

Errors use a named diagnostic, source/script line and a concrete correction hint. Colors use Odin's `core:terminal/ansi`; `NO_COLOR` (including an empty value), `TERM=dumb`, and redirected streams disable styling. Error text goes to `stderr`; CSV/JSON data stays on `stdout` or the chosen output file. Terminal column/line modes escape control bytes in database values. CSV/JSON preserve the values.

Commands

Command	Behavior
<code>.tables [LIKE-pattern], .schema [LIKE-pattern], .indexes [LIKE-table]</code>	Query the user catalog; internal tables are filtered.
<code>.connection, .databases</code>	Show cluster, endpoint, server identity and client state path.
<code>.status, .cluster</code>	Show the contacted voter's local node ID and applied slot; not quorum health.
<code>.nodes</code>	Show configured voter IDs, addresses and certificate identities; not live reachability. Requires the new status response.
<code>.health</code>	Perform a fresh quorum-fenced read.
<code>.reconnect CLIENT.json</code>	Switch endpoint within the same cluster and client certificate; retain session and pending write. Finish an active transaction first.
<code>.consistency linearizable local</code>	Choose standalone read consistency.
<code>.timeout MS</code>	Set service deadline, 1–60,000 ms; transport allows 250 ms for reply delivery.
<code>.pending, .retry</code>	Inspect or resolve the exact saved write. <code>.pending</code> contains SQL; keep its output private.
<code>.mode column list csv tabs line json</code>	Select rendering. Interactive default: column; redirected default: list.
<code>.headers on off, .separator TEXT, .nullvalue TEXT</code>	Configure result formatting. CSV uses quoted fields; JSON retains numbers and nulls.
<code>.read FILE, .output [FILE], .once FILE</code>	Read a script, redirect results, or redirect the next query result. Paths may be quoted; no shell expansion.
<code>.timer, .changes, .echo, .bail on off</code>	Timing, acknowledged/staged change counts, SQL echo, and stop-on-error behavior.
<code>.parameter set ?N JSON_VALUE, .parameter list clear</code>	Positional string, signed 64-bit integer, finite real or null bindings.
<code>.show, .limits, .help, .print TEXT, .quit, .exit</code>	Settings, bounds, help, literal output and exit.

`-c/--command` and `-f/--file` may be repeated and execute in argument order. `--mode`, `--headers`, `--bail`, and `--state PATH` configure the invocation. Exit status is **0** for success, **1** for a SQL/command/script failure, and **2** for configuration/state errors or a still-unresolved write. Scripts continue past errors unless `--bail/.bail on` is selected; they still exit nonzero.

Cluster JSON mode emits an array of objects per query. A BLOB is represented as `{"base64": "..."} so binary data is not silently decoded as text. Use distinct column aliases for consumers that reject duplicate JSON object keys. Parameters are parsed as typed JSON values, then rendered into escaped SQL literals before submission so each staged statement retains its values. Arbitrary SQL expressions are never evaluated as parameter values. NUL text parameters and typed vector parameters use the Python API; SQL vector functions remain available in the shell.`

Transactions and safe retries

```
BEGIN;  
UPDATE account SET balance=balance-10 WHERE id=1;  
SAVEPOINT credit;  
UPDATE account SET balance=balance+10 WHERE id=2;  
SELECT id, balance FROM account WHERE id IN (1,2);  
RELEASE credit;  
COMMIT;
```

Cluster BEGIN uses optimistic serializable transactions, not a long-held SQLite writer lock. Staged writes run in rolled-back previews; reads see those staged writes. COMMIT publishes the complete body through Paxos if the database revision has not changed. A revision conflict requires replaying the whole transaction. After a failed preview, use full ROLLBACK; savepoints cannot repair an invalid revision. ROLLBACK TO restores a saved staged body, and RELEASE forgets that savepoint and nested ones. Savepoints require an explicit BEGIN and are limited to 16. Exiting discards uncommitted previews. Read-only commits validate the revision.

The current service bounds still apply: 4,096 SQL bytes per request/transaction, eight write statements, 4,096 result rows, and 256 KiB of result storage. RETURNING, remote PRAGMA administration and arbitrary SQLite session features are outside the replicated SQL contract. The CLI does not weaken those checks. It first attempts a read-only query; an Invalid_SQL response is retried through the durable write path. Consequently CLI writes may incur an extra query round trip; benchmark clients continue to call execute/query explicitly.

Before sending a write, the shell durably records its session, sequence and exact SQL in a private SQLite recovery database. By default it is `CLIENT.json.shell.db`; `--state PATH` selects another. A separate advisory lock prevents simultaneous shells from sharing that state. New state and output files are owner-only. State commits use SQLite `synchronous=FULL` and rollback journaling.

If a reply is lost, another write is blocked. Keep the state database and its SQLite journal if present. State is bound to the loaded certificate fingerprint, and a cached TLS context prevents changing identity mid-session by replacing a PEM file. Reopen with the same client certificate and state path, optionally `.reconnect` to another voter, then run `.retry`. The service's durable session outcome prevents double execution. Do not delete state or resubmit the SQL under a fresh identity to work around an uncertain result. A terminal error outcome is saved before the next sequence is used. `.retry` never automatically repeats an entire script or changes the transaction's revision guard.

Distributed operation boundaries

The authenticated membership/status commands are observations. They do not change the fixed voter configuration. Safe online voter additions/removals, automatic enrollment, rolling upgrades,

and unbounded remote dump/import are outside the supported scope. Offline backup, verify-backup, restore, migrate and compact commands implement bounded maintenance; they are not local-shell commands. Follow the recovery procedures, including fencing the old namespace before restore. Certificate renewal uses coordinated shutdown and new trust material, not hot reload.

The CLI tests use disposable three-voter clusters, exercise local file operations, transactions, formatting, terminal styling/editing, state locking, whole-cluster restart, and a TLS proxy that deliberately loses an already-committed write reply. Run `python3 tools/check_cli.py --binary bin/sqlodin --server bin/sqlodin`. These checks do not turn an ongoing soak into a completed qualification run.

Recorded validation

The CLI validation manifest links **28 CLI checks and 14 native-service regression checks on each platform**: macOS arm64 and Linux x86_64 on insan@10.175.52.18. Each report identifies its tested optimized binary; client, server and linkage manifest hashes match. Static linkage checks passed for SQLite, sqlite-vec and OpenSSL. Linux tests used new on-disk project directories and stopped all their voter processes afterward. Four native source/cache rejection tests and Odin's strict vet/style checks also passed locally.

The Linux build used a task-local GCC link-driver wrapper because that instance had no Clang executable; it did not install system packages. The evidence records that setup and the optimized build flags. The ongoing three-host soak continues with its original binary and is separate from this validation.