

Replicated SQL contract under R1.1

This records the enforced policy, its execution argument and qualification limits. The qualified distributed deployment uses the same release build on three Linux x86-64 voters. Local macOS checks exercise portability; they do not qualify a heterogeneous floating-point cluster. It adds no release gate.

Transactions and outcomes

A write is one ordered transaction body: at most 4,096 UTF-8 bytes, eight prepared statements and 16 bound parameters. Statements share the parameter tuple; Python transaction construction assigns distinct parameter positions. The service owns BEGIN, COMMIT, savepoints, the application watermark and retry metadata. User SQL cannot replace those boundaries. SQLAlchemy stages work and submits one optimistic transaction body with the quorum-backed revision; savepoint release never acknowledges a durable write.

The write result reports completion, changes, node, applied slot and request sequence. It does not stream query rows. INSERT/UPDATE/DELETE RETURNING is rejected before stepping under policy 8. Use the read API for result rows and the supported ORM generated-key path for inserts. Statement errors roll back the whole request; SQL conflict actions cannot commit or discard another request in a grouped commit. The outer application commit, after the FULL journal barrier, gates acknowledgement. Read-only SELECT statements inside a write body may validate expressions but do not return a client result stream.

Functions, schema and extensions

The writer allowlist is implemented in `engine_policy_function`. It admits `abs`, `coalesce`, `ifnull`, `nullif`, `length`, `lower`, `upper`, `substr/substring`, `trim/ltrim/rtrim`, `replace`, `instr`, `hex/unhex`, `zeroblob`, `unicode`, `char`, `typeof`, `quote`, `like/glob`, `printf/format`, `min/max/count`, and the enumerated JSON constructors, inspection and extraction functions. All other registered writer functions are replaced by rejecting callbacks, including uses hidden in defaults. Wall-clock, random, connection-local state, last-insert-rowid and version functions cannot affect replicated execution. Local read connections retain aggregate/window and vector functions; their outputs become replicated input only if a client explicitly submits those concrete values in a later request.

Ordinary main-schema tables, supported indexes, constraints and triggers execute under the same restrictions. Temporary/attached state, user transaction control, ANALYZE, pragma table-valued interfaces, metadata mutation and arbitrary extension loading are excluded. The pragma namespace check is case-insensitive: SQL identifier capitalization cannot admit connection-local `data_version` into a write. SQLite's internal read-only `data_version` use by FTS5 remains permitted. Application catalog reads cannot bypass policy through rowid, although SQLite needs catalog row reads internally while compiling ordinary DDL.

Policy 9 requires a rowid table to leave at least one of `rowid`, `_rowid_` or `oid` unshadowed, so certified snapshots can bind its hidden row identity. WITHOUT ROWID tables need no hidden alias. After CREATE/ALTER, the final schema is checked inside the request's transaction/savepoint. An incompatible schema rejects the whole request, including accompanying DML; unknown validation errors fail closed. Startup and migration also validate this condition. An unsupported legacy schema requires explicit offline repair before upgrade, with the original files and backup preserved; it is never silently rewritten or partially activated.

DROP and ALTER require SQLite's own catalog reads. These are permitted only within the corresponding prepared maintenance statement and outside trigger/view expressions. The

permission resets before every subsequent statement. It cannot be borrowed by CREATE TABLE AS SELECT or later application catalog reads.

FTS5 is the admitted writable virtual module. Its shadow tables are protected by SQLite defensive mode. The vector API stores bounded finite float vectors in ordinary BLOB columns and uses the pinned sqlite-vec read functions. Writable vec0 virtual tables are outside the current replicated SQL policy. The configured 384-component total vector limit, 256-byte text parameters and existing SQL value, row, expression, trigger-depth and result limits remain explicit bounds.

The random-rowid fallback at signed INT64_MAX is prevented from committing, including writes reached through triggers. User edits to sqlite_sequence are rejected. The host owns engine connections; registering local functions, collations, modifying the schema directly or changing planner state outside the ordered service are unsupported privileged operations.

Ordering and failure handling

SQL result order is unspecified without ORDER BY. Applications requiring a stable sequence use an explicit total order, including a unique tie-breaker. Unordered result presentation is not a license for replicas to diverge in stored values or outcomes. Replicated execution starts from the same ordered schema and logical rows, with identical SQL bytes, parameter types/values, writer configuration, function policy and pinned SQLite build. Persistent planner statistics cannot be introduced through the public write interface. The pinned build does not enable STAT4; table/index estimates without ANALYZE come from SQLite's schema-derived defaults (sqlite3DefaultRowEst in the verified amalgamation). Connection-local planner controls are inaccessible to application SQL. Replaying the same schema therefore preserves index enumeration and planning inputs. Parameter-dependent replanning sees the same submitted parameters at every voter.

Ordinary B-tree scans compare logical keys, including rowid or the WITHOUT ROWID primary key; page numbers, free-list layout and cache residency are not SQL ordering inputs. A chosen plan's scans, tie handling, joins and trigger execution thus consume the same sequence. Generated integer keys use the same prior keys and high-water marks; the random fallback is blocked. Built-in collations and the admitted functions come from the same build, with no local extension registration. FTS updates are derived from the same ordered text changes and pinned tokenizer. This is a refinement argument under the pinned engine's implementation semantics, not a claim that arbitrary SQLite versions or independently altered schemas will choose the same plan. Upgrades require the documented coordinated procedure.

test_sql_ordering_survives_physical_layout_and_cache_changes applies ten query-sensitive writes to equivalent stores after page churn. One store is vacuumed, uses an eight-page cache and reopens before every write. Indexed and unordered LIMIT selection, explicit total orders, scalar subqueries, grouping, joins, UNION, trigger insertion order, generated rowids and deletion selection produce identical outcomes and full logical image digests. Both local and Linux reopen reports pass. R1.2 supplies individual/grouped, FTS/vector and restart equivalence; R1.4 supplies transaction-history rather than presentation-order checks.

Fixed parser/value/depth limits and audited SQL expression/constraint failures produce durable rejections. Unknown SQLite/storage/allocation failures halt the host without acknowledging a fabricated SQL outcome. Read work budgets apply to local read execution and may return Query_Limit. They are not reused as replicated write rejection decisions: planner-dependent VM counts cannot silently choose different outcomes on different voters. An expensive chosen write must finish or the voter must stop acknowledgement; a local timeout cannot skip it or certify a different rejection. After a resource fault, repair and replay are required before that voter serves fresh state. The surviving quorum can continue while one voter is unavailable. This preserves safety

without promising successful execution or bounded latency when all voters lack resources for the chosen SQL. R5 measures admission, bounded queues, copying/allocation and representative expensive work; R6 separately qualifies the declared mixed workload and capacity goals.

Versioning and evidence

REPLICATION_POLICY is centralized in `engine_limits.odin`. Store identity and peer compatibility include policy 9; wire version 3 carries request epochs. Ordinary startup refuses a different stored policy. The existing explicit private-destination migration recognizes reviewed policy-6/7/8 source formats; coordinated restore uses one verified backup and a fresh namespace. No rolling-policy compatibility is claimed. SQLite source/build options contribute to the stored engine fingerprint. The extension contract introduced in policy 8 retains its exact `sqlite-vec` source and compiler-flag SHA-256 in policy 9. The static builder compiles its identity stamp in the same translation unit as the hash-verified extension. Every registration checks that stamp against the policy's constant; engine startup fails if registration or identity validation fails. Thus a local extension replacement cannot retain policy-8 compatibility merely by keeping the SQLite library unchanged. Changing the extension contract requires a reviewed policy upgrade. This is a build-consistency check, not protection against a malicious compiler or a privileged actor replacing both the policy code and binary.

The mixed-case pragma reproducer committed two forbidden rows under the previous check. `pragma-case-local.json` retains that failure; the corrected cases pass in `policy8-local.json`. A blanket catalog-read denial broke DDL and was removed; `catalog-policy-local.json` instead verifies the existing rejection boundary with different catalog row allocation. The FTS comparison in `search-group-state-local.json` checks all per-request outcomes and the complete logical snapshot digest across 32 individual/grouped FTS/vector/trigger inserts, updates, deletes and row reuse. `all-tests-policy8-final-linux.json` passes 154 tests, `network-policy8-linux.json` passes 15 native checks, and `migration-policy8-linux.json` passes retained-binary rollback/remigration. These reports are supplemented by `orm-policy8-linux.json` (27 checks) and `restore-policy8-linux.json` (legacy backup/epoch/restart). The real native linkage negative control `extension-identity-local-v2.json` opens the accepted extension and rejects a mismatched extension at engine startup; the initial probe encountered an unused-import vet diagnostic in uninstantiated upstream generic code. Full-project strict/vet checks remain separate. Evidence reports are under `benchmarks/results/verification-20260924/`, with earlier failures preserved. They are criterion-level evidence, not final production qualification.