# Console Storage for the Rust Port: A Typed Schema with Invariants in Rust

STATE
accepted
TYPE
Architectural Specification
CREATED
2026-10-15
STATUS
Accepted
AUTHORS
Vikrant Rathore
UPDATED
2026-10-15

SUMMARY
Specifies the console's storage for the Rust port as one typed baseline migration of 33 STRICT tables without triggers, in which every invariant that a Zig trigger enforced is enforced by exactly one transactional Rust function on SQLite and PostgreSQL alike, together with one search language, retention jobs and a per-route table of reads and writes.

DISCUSSION
No external discussion URL assigned

LABELS
console, storage, rust

**Record status**

The catalog defines the review process and the meaning of each state. This numbered record forms part of the catalog.

## Contents

1. Abstract ..... 1
2. Motivation and scope ..... 2
3. Specification ..... 3
4. Rationale and alternatives ..... 25
5. Security and operational considerations ..... 26
6. Backwards compatibility ..... 26
7. Acceptance criteria and implementation status ..... 27
8. Open questions ..... 27
9. Review history ..... 27
10. References ..... 28

## 1. Abstract

This record specifies the console's storage for the Rust port before any of it is written. The console keeps the behaviour of SID 0007 and stores its state through sqlx as SID 0011 specifies: an embedded SQLite file for one node and one PostgreSQL database for a cluster. The schema is one baseline migration of the component `console`. It has 33 typed tables; the Zig console had 44 tables and 49 triggers. No console table has a trigger. Each invariant that a Zig trigger enforced is now enforced by exactly one Rust function that runs its statements in one transaction, and the database's `NOT NULL`, `CHECK` and foreign-key constraints refuse

what the Rust types cannot represent. The record also fixes one search language for the Events page with the same matches on both backends, the retention jobs and their bounds, and the tables each HTTP route group reads and writes.

## 2. Motivation and scope

The Zig console grew its schema in 40 additive versions. Its invariants lived in SQLite triggers, because Zaxonlite replicated one SQL batch atomically and the storage owner could not hold an interactive transaction across replicated statements. A mutation was therefore a single `INSERT` into a one-row stage table whose trigger performed the real writes, the history rows and the audit record, and then deleted the stage row.

We redesign this storage instead of transcribing it, for three reasons.

1. **Two dialects.** SQLite and PostgreSQL write triggers differently. SQLite has row triggers with `NEW` and `OLD` and `RAISE(ABORT)`; PostgreSQL needs a PL/pgSQL function per trigger, distinguishes row and statement triggers, has no `NEW` in statement triggers, and fires them in name order. Forty-nine triggers written twice would be two programs that must agree, tested twice, and read by nobody who reads the Rust code.
2. **History no deployment needs.** Versions 1 to 40 rebuilt tables, renamed them, re-created triggers with one more column, and kept legacy columns readable. No Rust deployment has a console database, so none of that history applies. One baseline states the final design.
3. **Types.** SID 0011 states in SQL what the Rust types state in Rust. The Zig schema stored binary values as hex text, enumerations as integers, absent values as `0` or the empty string, and copied the client address into mutated rows only so that a trigger could read it. A typed schema refuses such rows.

This record covers every console table, the indexes the console adds to core tables, the console's search over incidents, retention, and the storage interface the console server calls. It changes four things in the core storage of SID 0011, listed in "Changes to the core storage". It does not change the console's behaviour, routes, wire formats or bounds that protect operators or the server.

### 2.1. Constraints that existed only because of Zig

The following constraints of the Zig console came from Zig, Zaxonlite or the Zig standard library, not from the problem. The Rust design drops each of them.

**Table 25** Zig-only constraints and their Rust replacement

| Zig constraint | Rust design |
|---|---|
| A 32-slot bounded mailbox of request and result unions, polled by the storage thread with a quota of 16 operations per tick, with owned bounded payload copies, explicit release functions and ticket cancellation, because the console threads could not run asynchronous I/O. | The console calls `Storage::with_database` (SID 0011) with an asynchronous job, which runs on the storage thread and returns its result. Storage functions take ordinary Rust arguments and return ordinary Rust values. The protocol's request and result unions are not a storage interface. The remaining resource bound is stated in "Calling convention and resource bounds". |
| Single-statement mutations through one-row stage tables (`console_policy_stage`, `console_inspection_stage`, `console_policy_order_stage`, `console_reputation_stage`, `console_country_commit`, `console_policy_import_commit`, `console_page_stage`, `console_password_rotation`) and triggers, because a replicated batch was the unit of atomicity. | A Rust function runs its statements in one database transaction. The stage tables are removed. |
| Authorization, the mutation and its audit in one SQL statement (`WITH a AS (...) INSERT ... SELECT ... FROM a`), with the | The function reads the principal, checks it, and writes in the same transaction; each step returns a typed result. |

| Zig constraint | Rust design |
|---|---|
| outcome inferred from a change count that also counted trigger writes. | |
| Binary values as lowercase hex text (digests, boot ids, minute and challenge payloads, ranking and GeoIP chunks, captured heads, recovery digests, TOTP envelopes), because Zaxonlite's cell transport passed text. | `BLOB` on SQLite and `BYTEA` on PostgreSQL, with exact lengths. |
| Columns that existed only to signal a trigger: `ended_at`, `mfa_step` and `recovery_slot` on sessions, `remove_requested` on tokens, `guard` on rule hits, and `client_ip` on 16 mutated tables. | Removed. The function passes the values directly to the statements that need them. |
| Token credentials mirrored as rows of `console_sessions` so that one registry served both credential kinds. | Tokens are authenticated from `console_tokens` alone. |
| Sentinels for absence: actor `0`, empty digests and sources, `password_expires = 0`, challenge cause `255`, response state `0` meaning "not recorded" for pre-version-40 rows. | `NULL` where the Rust field is an `Option`; states that only old Zig rows could carry are not representable. |
| Enumerations stored as integers (challenge outcomes, algorithms, response states). | The lowercase name of the Rust enum, with a `CHECK`, as SID 0011 requires. |
| Durable cursors as keys of the text key/value table `sibuna_meta` (`console_audit_cursor`, `policy_version`). | Typed single-row tables (`console_audit_cursor`; the core `policy_revision`). |
| Page sizes and lookahead rows sized to fit Zaxonlite's 100-row and 64 KiB statement envelope (for example 32 challenge-minute rows because a row was 1,792 hex bytes, or ten-row chunk reads). | A storage function takes its page size from the protocol request and imposes no envelope of its own. Page sizes remain response bounds of the protocol. |
| Rankings archives written in 2,048-byte chunks with a reservation and a separate publish step, and policy imports staged in chunks, to fit Zaxonlite's 64 KiB statement envelope and the mailbox's bounded payloads. | An archive is one row holding one `BLOB` of at most 48,536 bytes, stored by one function in one transaction. A policy import is one bounded request whose documents are validated and applied in one transaction. |
| Fixed SQL text buffers and fixed parameter arrays, and SQL text assembled per filter combination. | Static statements with bound parameters; optional filters are written as `($n IS NULL OR column = $n)`. |
| Saturating arithmetic inside trigger bodies, because a SQLite integer overflow in a trigger becomes a float. | Checked arithmetic in Rust; an overflow makes the value `NULL` (unavailable), as before. |

Every bound that protects an operator or the server is kept: sizes of fields and documents, counts of users, tokens, sessions, grants, destinations and queued events, the rankings quota, retention periods, lease lifetimes and query deadlines.

## 3. Specification

### 3.1. Principles

**One baseline migration.** The console's schema is the component `console` of `sibuna_schema` (SID 0011), version 1. It creates every table and index of this record at once. A node built without the console neither applies nor refuses it. The Zig table `console_schema` does not exist; `sibuna_schema` records the version.

**Typing.** The typing rules of SID 0011 apply without exception:

- Every SQLite table is `STRICT`. Tables with a small row and a composite or text key are also `WITHOUT ROWID`.
- Timestamps are integers holding Unix seconds; minutes are Unix minutes; `*_ms` columns are milliseconds.
- Raw bytes are `BLOB` (`BYTEA`). `TEXT` holds only textual values: names, labels, addresses, prefixes, URLs, HTML and JSON documents.
- A column is `NOT NULL` exactly when its Rust field is not an `Option`.
- An enumerated column has a `CHECK` listing exactly the names its Rust enum writes with `as_str` and reads with `parse`.
- An integer with a bounded meaning has a `CHECK` with that range. A node identifier is the storage node of SID 0011, 1 to 8,388,607; a single node is node 1. A boot identifier is 16 bytes. A digest is 32 bytes.
- References are foreign keys. SQLite enforces them because `sibuna-storage` opens every connection with `foreign_keys` on.

**Invariants in Rust.** Each invariant that a Zig trigger enforced is enforced by exactly one Rust function in the crate `sibuna-console-storage`. The function runs every statement of the invariant in the caller's transaction, so the invariant holds at every commit. No other function writes the rows the invariant governs. The database's constraints are the second line of defence: a row that a defective function or a direct operator write would leave inconsistent is refused where SQL can state the rule.

**Table 26** Storage invariants and the one function that enforces each

| Id | Invariant | Function |
|---|---|---|
| S1 | A new account is written with a `user.create` audit row naming the creator (or no actor at bootstrap); the bootstrap administrator is created only while no account exists. | `accounts::insert_user` |
| S2 | A change of an account's authority (role, disablement, password, factor or explicit revocation) increments its revision, deletes all of its sessions, and writes one audit row with redacted before and after summaries. Tokens of the account stop authenticating because their creator revision no longer matches. | `accounts::update_authority` |
| S3 | A sign-in inserts the session, raises the account's last sign-in time to the larger value, consumes the TOTP step or recovery slot it used exactly once, and writes `session.create`; a replayed or older step, a used slot, an expired bootstrap password or a full session table refuses it. | `accounts::insert_session` |
| S4 | A sign-out writes `session.logout` with the address of the sign-out and deletes the session. | `accounts::end_session` |
| S5 | Enabling TOTP requires an unexpired enrollment at the expected factor revision, stores the recovery digests, writes `totp.enable`, and revokes the account through S2, which writes `user.revoke`. | `accounts::enable_totp` |
| S6 | A password change replaces the hash, clears `must_change` and the bootstrap expiry, revokes through S2, and inserts the replacement session through S3, all or nothing. | `accounts::change_password` |
| S7 | A token is created only by an administrator cookie session, within the capacity of 1,024 tokens, and with a `token.create` audit row; its authority (digest, label, role, scopes, creator, creator revision, expiry) never changes afterwards. | `tokens::create_token` |
| S8 | The only change of a token is its disablement, which increments its revision and writes `token.revoke`. | `tokens::revoke_token` |
| S9 | Only an inactive token is removed, and its removal writes `token.remove`; the audit trail survives the row. | `tokens::remove_token` |
| S10 | At most 64 kiosk grants are outstanding, and each grant writes `kiosk.grant` without the code. | `kiosk::grant_kiosk` |

| Id | Invariant | Function |
|---|---|---|
| S11 | A kiosk grant is exchanged at most once, before its use-by time and while the granting account keeps its revision; the exchange inserts one kiosk session expiring with the grant, writes `kiosk.exchange` and `session.create`, and leaves the granting account's last sign-in time unchanged. | `kiosk::exchange_kiosk` |
| S12 | A GeoIP generation becomes active only when all of its ranges are stored and the active revision is the expected one; the switch writes `geoip.activate`. | `geoip::activate_generation` |
| S13 | A rule edit writes the rule, a baseline history row when the rule had none, the edit history row, and a redacted `policy.edit` row, and advances the policy revision to exactly the expected revision plus one. | `policy::edit_rule` |
| S14 | An inspection edit writes the four modes, their history row and `inspection.edit`, and advances the revision as in S13. | `policy::edit_inspection` |
| S15 | An ordering swaps the priorities of two adjacent rules, writes two history rows and `policy.order`, and advances the revision once. | `policy::order_rules` |
| S16 | An import replaces every rule with the complete set carried by one request (at most 128 documents), writes one history row per rule and `policy.import`, and advances the revision once. | `policy::import_rules` |
| S17 | A reputation edit or removal by the console writes or deletes the prefix, records the console as its owner (or removes the ownership), writes `reputation.edit` or `reputation.remove`, and advances the revision once. | `reputation::change_prefix` |
| S18 | A country replacement deletes the country's owned prefixes that the reviewed set no longer contains, upserts the reviewed set with the pinned generation, audits the counts before and after, deletes the staged chunks, and advances the revision once. | `reputation::apply_country` |
| S19 | A page edit or reset writes or deletes the template, writes `page.edit` or `page.reset` with digests and sizes only, and increments the policy revision so every node rebuilds. | `pages::change_page` |
| S20 | A setting change is within the catalogue's bounds, matches the expected revision, and writes `setting.change` with the old and new value. | `settings::change_setting` |
| S21 | At most eight notification destinations exist; a creation writes `notification.create` with a redacted summary. | `notifications::create_destination` |
| S22 | A destination change increments its revision, drops a secret whose target changed, writes `notification.change`, and marks every pending or sending delivery of the old revision as skipped. | `notifications::change_destination` |
| S23 | A destination removal marks its pending and sending deliveries as skipped, writes `notification.remove` naming the remover, and deletes the destination. | `notifications::remove_destination` |
| S24 | An event is enqueued at most once per node, boot and sequence, only while fewer than 256 events are pending and fewer than 4,096 are stored (after purging up to 16 completed events that are older than seven days or exceed the total), with one delivery per enabled subscribed destination; an event without a destination is complete when raised. | `notifications::enqueue_event` |
| S25 | A claim moves one due delivery of a current, enabled, cooled-down destination to `sending` under the notifier lease, and records the attempt time on the destination; deliveries whose claim expired return to `pending` (or `failed` after three attempts) through S26. | `notifications::claim_delivery` |
| S26 | A delivery that leaves `sending` records the outcome on its destination when the revision still matches, writes `notification.delivery`, and completes its event once no delivery of the event is pending or sending. | `notifications::finish_delivery` |

| Id | Invariant | Function |
|---|---|---|
| S27 | Deliveries never outlive their event. | `notifications::prune_events` |
| S28 | A test of a destination writes `notification.test` and, on completion, `notification.test_result`, and records the result on the destination when its revision still matches. | `notifications::record_test` |
| S29 | Audit identifiers increase in commit order and are never reused, even after retention deletes every audit row. | `audit::append_audit` |
| S30 | A node command is recorded as an intent with `node.command.intent` before any local effect, within 4,096 commands per node. | `nodes::request_command` |
| S31 | A command leaves `intent` once, to `applied` or `rejected`, and writes `node.command.applied` or `node.command.rejected`; a repeated identical completion changes nothing. | `nodes::complete_command` |
| S32 | A ranking archive is stored only when its bytes have its digest as their SHA-256 and decode as an archive, and only within the 512 MiB charged quota; its charge is added to the usage counter by a conditional update and the archive row, with its node, boot and minute, is inserted in the same transaction. | `rankings::store_archive` |
| S33 | Merged into S32: an archive is stored whole, so there is no separate publication. | `rankings::store_archive` |
| S34 | Deleting an archive subtracts its charge, so the usage counter equals the sum of the charges of the stored archives. | `rankings::prune_rankings` |
| S35 | A rule-hit observation and its hour and day roll-ups commit together; replaying an identical observation changes nothing, a conflicting observation with the same identity aborts the whole batch, and an overflow makes a roll-up's hit count unavailable rather than wrong. | `rule_hits::record_rule_hits` |
| S36 | Deleting an incident deletes its full-text entry and its console sidecars in the same transaction. | `sibuna_storage::Database::delete_incidents` |

The Zig console also enforced invariants with conditional statements rather than triggers. They are kept, each in one function:

**Table 27** Invariants that Zig enforced with conditional statements

| Id | Invariant | Function |
|---|---|---|
| S37 | Authorization is rechecked inside the transaction that mutates, from the authoritative rows: the session or token, its account, revision, disablement, role, CSRF digest for cookies, required TOTP and token scope. | `access::authorize_in` |
| S38 | A policy workflow (S13 to S18) commits only when the policy revision equals the caller's expected revision, and then sets it to that revision plus one. | `revision::advance` |
| S39 | An export is audited (`events.export_prepared` or `audit.export`) before its bytes leave storage; an export whose credential no longer authorizes returns nothing. | `audit::record_export` |
| S40 | A minute journal row (traffic or challenge) only advances: an identical write changes nothing, an older write is ignored, and a write that conflicts with the stored row is refused. | `observations::write_minute` |

**Transactions and locks.** A storage function receives `&mut Transaction` from `sibuna-storage` and never commits; its caller begins and commits. On SQLite, one storage thread runs one job at a time, so a transaction never waits for another writer of the same node. On PostgreSQL several nodes write concurrently under `READ COMMITTED`, and the functions take locks in one fixed order, which prevents deadlocks:

1. A capacity-checked insert (users, tokens, sessions, kiosk grants, destinations, queued events, node commands, GeoIP generations, staged country sets) first takes `pg_advisory_xact_lock` with the key of its table, so two nodes cannot both pass the count.

2. A policy workflow then locks the revision with `SELECT revision FROM policy_revision WHERE id = 1 FOR UPDATE`, as the core's ban batches already do (SID 0011).
3. Conditional updates of the governed rows follow (`WHERE revision = $expected` and similar), which PostgreSQL re-evaluates after a row lock. When a change involves two accounts (an administrator changing another account), both rows are locked with `FOR UPDATE` in identifier order and the actor is rechecked, so two administrators cannot disable each other at once.
4. `console_rank_usage` is locked by its conditional update.
5. `audit::append_audit` runs last and locks `console_audit_cursor`. Audit appends are therefore serialised cluster-wide, and identifiers follow commit order.

**No triggers.** The console defines no trigger. The only triggers in the database are the core's policy-revision triggers on `policies`, `ip_reputation` and `policy_inspection` (SID 0011), which exist because operators may write those tables directly. Operators do not write console tables directly; what such a write can and cannot break is stated under "Security and operational considerations". The one console table that refers to an operator-writable core table, `console_reputation_sources`, follows direct deletions through `ON DELETE CASCADE`, a declared foreign-key action rather than a trigger. `console_pages` gets no revision trigger, because a template must be validated by the policy crate before it is stored, which SQL cannot do.

### 3.2. Calling convention and resource bounds

The console server calls storage through `Storage::with_database`. A route handler builds a job that begins a transaction, calls one storage function, and commits:

```
let revision = storage.with_database(move |db| async move {
    let mut tx = db.begin().await?;
    let revision = settings::change_setting(&mut tx, &credential, &change, now).await?;
    tx.commit().await?;
    Ok(revision)
})?;
```

A storage function has the shape below. Failures are values of `ConsoleStorageError` (`Unauthorized`, `Forbidden`, `Conflict`, `Capacity`, `InvalidInput`, `Unavailable` and the driver's error), which the server maps to the protocol's failures; a failure leaves the transaction uncommitted.

```
/// Changes one setting and audits it (S20). Borrows the credential and the change;
/// returns the new revision. Fails with `Forbidden` unless the credential is an
/// administrator cookie session, and with `Conflict` on a stale expected revision.
pub async fn change_setting(
    tx: &mut Transaction,
    credential: &Credential,
    change: &SettingChange,
    now: u64,
) -> Result<u64, ConsoleStorageError> {
    let principal = access::authorize_in(tx, credential, Need::Administrator, now).await?;
    let before = read_setting(tx, change.key).await?;
    if before.revision != change.expected_revision {
        return Err(ConsoleStorageError::Conflict);
    }
    let revision = write_setting(tx, change, principal.user, now).await?;
    let row = AuditRow::setting_change(&principal, change, &before, revision, now);
    audit::append_audit(tx, &row).await?;
    Ok(revision)
}
```

Time is a parameter, read once by the caller, so tests control it. The resource bounds are:

- **Outstanding jobs.** `with_database` blocks its caller until the job finishes, so a calling thread has at most one outstanding job. The bounded job queue of SID 0011 bounds the jobs of all callers together: when 32 jobs are queued, `with_database` refuses the next one at once with a busy error, and the console answers that refusal with its 503 "console busy" response. The console adds no queue of its own.
- **Fairness.** The storage thread of SID 0011 runs at most 16 jobs between two ticks and always runs a due tick first, so console work cannot hold back incident batches and policy rebuilds. The console relies on that bound; SID 0011 owns it.

- **Query deadlines.** A console read that aggregates over a time window (security trends, minute and challenge summaries, events with a search, rankings history, rule-hit history, replay) runs under a work bound. On SQLite the bound is `--console-query-steps` virtual-machine steps (100,000 to 50,000,000, default 4,000,000), enforced through sqlx's safe progress handler (`LockedSqliteHandle::set_progress_handler` of sqlx-sqlite 0.9). On PostgreSQL it is `SET LOCAL statement_timeout = 2000` (milliseconds) in the read's transaction. No new option is added. Exhaustion returns `Unavailable` with the hint to narrow the period, as in SID 0007. Point reads by key have no separate bound.
- **Write batches.** A background delete removes at most 256 rows per transaction (see "Retention and quotas"), so no job holds the storage thread or a lock for long.

### 3.3. Changes to the core storage

The console needs four changes in `sibuna-storage`. No Rust deployment exists, so they are folded into the core baseline (version 1) as earlier changes of SID 0011 were. The bounded job queue and the tick-first fairness of "Calling convention and resource bounds" are SID 0011's and are not changes made for the console.

1. **Search columns.** The full-text index holds the canonical tokens of "Search" instead of a lossy rendering. On SQLite the table becomes `CREATE VIRTUAL TABLE incidents_fts USING fts5(path, offending_payload, tokenize = 'ascii')`. On PostgreSQL the columns `search_text` and the generated `search` are replaced by `search tsvector NOT NULL`, written by Rust from the tokens, with the existing GIN index.
2. **Incident sidecars.** The incident batch writes the console's evidence, heads and country sidecars in the batch's transaction, through a hook the console registers with `Hooks`. The hook receives the batch's incident identifiers and records and inserts rows only when the console is enabled.
3. **Incident deletion.** `Database::delete_incidents(tx, before, limit)` deletes at most `limit` incidents recorded before `before`, oldest first, together with their `incidents_fts` rows on SQLite (S36). The sidecars follow by `ON DELETE CASCADE`. Receipts are never deleted, so identifiers stay monotonic.
4. **Test harness.** `testing::for_each_backend` is exported behind the `testing` feature, so the console's tests run on SQLite and, when `SIBUNA_TEST_DATABASE_URL` is set, on PostgreSQL.

The rebuild of the core tick also reads `console_pages` when the console component is applied, because the templates are part of the published engine (SID 0007).

### 3.4. Tables

The SQLite text below is normative. The PostgreSQL text differs only as "Dialect differences" states. The console also adds four indexes to core tables; it adds no columns to them.

```
CREATE INDEX console_policies_order ON policies (priority, name, id);
CREATE INDEX console_incidents_category
    ON security_incidents (violation_category, recorded_at, id);
CREATE INDEX console_incidents_campaign ON security_incidents (campaign_id, recorded_at, id);
CREATE INDEX console_incidents_client ON security_incidents (client_ip, recorded_at, id);
```

The first serves the rules table in evaluation order including disabled rules, which the core's partial index omits. The next three serve the Events filters by category, campaign and address; the address index is new, because the grouped view and the address filter otherwise scan the whole period.

#### 3.4.1. Accounts and sessions

```
CREATE TABLE console_users (
    id INTEGER PRIMARY KEY,
    username TEXT NOT NULL UNIQUE CHECK (length(CAST(username AS BLOB)) BETWEEN 1 AND 64),
    password_hash TEXT NOT NULL CHECK (length(password_hash) BETWEEN 1 AND 255),
    role TEXT NOT NULL CHECK (role IN ('viewer', 'operator', 'admin')),
    revision INTEGER NOT NULL DEFAULT 1 CHECK (revision >= 1),
    disabled INTEGER NOT NULL DEFAULT 0 CHECK (disabled IN (0, 1)),
    must_change INTEGER NOT NULL DEFAULT 0 CHECK (must_change IN (0, 1)),
    password_expires_at INTEGER CHECK (password_expires_at >= 0),
    last_login_at INTEGER CHECK (last_login_at >= 0),
    modified_by INTEGER REFERENCES console_users (id),
```

```
    modified_at INTEGER NOT NULL CHECK (modified_at >= 0)) STRICT;

CREATE TABLE console_sessions (
    digest BLOB NOT NULL PRIMARY KEY CHECK (length(digest) = 32),
    user_id INTEGER NOT NULL REFERENCES console_users (id),
    user_revision INTEGER NOT NULL CHECK (user_revision >= 1),
    kind TEXT NOT NULL CHECK (kind IN ('browser', 'kiosk')),
    csrf_digest BLOB NOT NULL CHECK (length(csrf_digest) = 32),
    created_at INTEGER NOT NULL CHECK (created_at >= 0),
    expires_at INTEGER NOT NULL,
    idle_expires_at INTEGER NOT NULL,
    client_ip TEXT CHECK (length(client_ip) BETWEEN 1 AND 48),
    user_agent_digest BLOB CHECK (length(user_agent_digest) = 32),
    CHECK (expires_at > created_at AND expires_at - created_at <= 43200),
    CHECK (idle_expires_at > created_at AND idle_expires_at <= expires_at)) STRICT, WITHOUT ROWID;
CREATE INDEX console_sessions_user ON console_sessions (user_id);
CREATE INDEX console_sessions_deadline ON console_sessions (idle_expires_at);

CREATE TABLE console_totp (
    user_id INTEGER PRIMARY KEY REFERENCES console_users (id),
    envelope BLOB NOT NULL CHECK (length(envelope) = 60),
    key_id BLOB NOT NULL CHECK (length(key_id) = 32),
    revision INTEGER NOT NULL CHECK (revision >= 1),
    enabled INTEGER NOT NULL DEFAULT 0 CHECK (enabled IN (0, 1)),
    enroll_by INTEGER NOT NULL CHECK (enroll_by >= 0),
    last_step INTEGER CHECK (last_step >= 0),
    recovery_digests BLOB CHECK (length(recovery_digests) = 320),
    recovery_used INTEGER NOT NULL DEFAULT 0 CHECK (recovery_used BETWEEN 0 AND 1023),
    modified_at INTEGER NOT NULL CHECK (modified_at >= 0),
    CHECK (enabled = 0 OR recovery_digests IS NOT NULL)) STRICT;

CREATE TABLE console_kiosk_grants (
    digest BLOB NOT NULL PRIMARY KEY CHECK (length(digest) = 32),
    user_id INTEGER NOT NULL REFERENCES console_users (id),
    user_revision INTEGER NOT NULL CHECK (user_revision >= 1),
    label TEXT NOT NULL DEFAULT '' CHECK (length(CAST(label AS BLOB)) <= 64),
    created_at INTEGER NOT NULL CHECK (created_at >= 0),
    use_by INTEGER NOT NULL,
    expires_at INTEGER NOT NULL,
    consumed_at INTEGER,
    CHECK (use_by = created_at + 600 AND expires_at = created_at + 43200),
    CHECK (consumed_at IS NULL OR consumed_at BETWEEN created_at AND use_by)) STRICT, WITHOUT ROWID;
CREATE INDEX console_kiosk_outstanding ON console_kiosk_grants (use_by)
    WHERE consumed_at IS NULL;
```

- `console_user_activity` is merged into `console_users.last_login_at`. Zig kept it apart because every update of a user row fired the revoking trigger; without triggers, a sign-in updates one column and revokes nothing.
- The session's deadline is `idle_expires_at` alone, because the `CHECK` keeps it at most `expires_at`; the Zig expression index on `MIN(expires, idle_expires)` becomes a plain index.
- A session is valid while its `user_revision` equals the account's revision, the account is enabled, and `idle_expires_at` lies in the future. S2 deletes the sessions as well; the revision comparison is the second line.
- A browser session lives at most 12 hours and idles out after 30 minutes; only authenticated HTTP activity moves `idle_expires_at` forward, never past `expires_at`. A kiosk session has `idle_expires_at = expires_at`, the grant's expiry.
- The TOTP envelope is the 60-byte XChaCha20-Poly1305 envelope of SID 0007, the key identifier a 32-byte digest, and the recovery digests ten 32-byte digests in slot order; bit `i` of `recovery_used` marks slot `i` as consumed. A child table of recovery codes was considered; ten fixed slots consumed by one conditional update are simpler and as well typed.
- Capacities: 1,024 accounts and 4,096 live sessions (browser and kiosk together), checked by S1 and S3 under the capacity lock.

### 3.4.2. Tokens

```
CREATE TABLE console_tokens (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    digest BLOB NOT NULL UNIQUE CHECK (length(digest) = 32),
    label TEXT NOT NULL CHECK (length(CAST(label AS BLOB)) BETWEEN 1 AND 64),
    role TEXT NOT NULL CHECK (role IN ('viewer', 'operator', 'admin')),
    scopes INTEGER NOT NULL CHECK (scopes BETWEEN 1 AND 255),
    created_by INTEGER NOT NULL REFERENCES console_users (id),
    creator_revision INTEGER NOT NULL CHECK (creator_revision >= 1),
    created_at INTEGER NOT NULL CHECK (created_at >= 0),
    expires_at INTEGER CHECK (expires_at > created_at),
    disabled INTEGER NOT NULL DEFAULT 0 CHECK (disabled IN (0, 1)),
    revision INTEGER NOT NULL DEFAULT 1 CHECK (revision >= 1),
    modified_by INTEGER NOT NULL REFERENCES console_users (id),
    modified_at INTEGER NOT NULL CHECK (modified_at >= 0)) STRICT;
CREATE INDEX console_tokens_creator ON console_tokens (created_by, id);
```

- `scopes` is the set of the eight capabilities of SID 0007 (`stats_read`, `events_read`, `policy_read`, `policy_write`, `geoip_read`, `geoip_write`, `users_read`, `users_write`) as bits 0 to 7. A ninth capability is a new console migration.
- A token authenticates while it is enabled and unexpired and its creator is an enabled administrator without `must_change` whose revision equals `creator_revision`. Tokens are separate from sessions. Zig mirrored each token as a session row so that one registry and one revocation statement served both kinds; here S2 invalidates tokens through the revision, and one row holds a token's whole authority.
- `AUTOINCREMENT` keeps identifiers unique for ever, because an identifier is printed to operators and names the token in audit rows after its removal.

### 3.4.3. Audit

```
CREATE TABLE console_audit (
    id INTEGER PRIMARY KEY CHECK (id >= 1),
    recorded_at INTEGER NOT NULL CHECK (recorded_at >= 0),
    actor INTEGER CHECK (actor >= 1),
    actor_role TEXT CHECK (actor_role IN ('viewer', 'operator', 'admin')),
    action TEXT NOT NULL CHECK (action IN (
        'user.create', 'user.disable', 'user.enable', 'user.role', 'user.password_reset',
        'user.password_change', 'user.revoke', 'session.create', 'session.logout',
        'session.denied', 'totp.enable', 'token.create', 'token.revoke', 'token.remove',
        'kiosk.grant', 'kiosk.exchange', 'geoip.activate', 'policy.edit', 'policy.order',
        'policy.import', 'inspection.edit', 'reputation.edit', 'reputation.remove',
        'reputation.country', 'page.edit', 'page.reset', 'setting.change',
        'notification.create', 'notification.change', 'notification.remove',
        'notification.delivery', 'notification.test', 'notification.test_result',
        'node.command.intent', 'node.command.applied', 'node.command.rejected',
        'events.export_prepared', 'audit.export', 'peer.rejected')),
    subject INTEGER CHECK (subject >= 0),
    target TEXT CHECK (length(CAST(target AS BLOB)) BETWEEN 1 AND 128),
    client_ip TEXT CHECK (length(client_ip) BETWEEN 1 AND 48),
    before_summary TEXT CHECK (CASE WHEN json_valid(before_summary)
        THEN json_type(before_summary) = 'object' AND length(before_summary) <= 2048
        ELSE before_summary IS NULL END),
    after_summary TEXT CHECK (CASE WHEN json_valid(after_summary)
        THEN json_type(after_summary) = 'object' AND length(after_summary) <= 2048
        ELSE after_summary IS NULL END)) STRICT;
CREATE INDEX console_audit_time ON console_audit (recorded_at, id);
CREATE INDEX console_audit_action ON console_audit (action, id);
CREATE INDEX console_audit_actor ON console_audit (actor, id);

CREATE TABLE console_audit_cursor (
    id INTEGER PRIMARY KEY CHECK (id = 1),
    last_id INTEGER NOT NULL CHECK (last_id BETWEEN 0 AND 9223372036854775807)) STRICT;
INSERT INTO console_audit_cursor (id, last_id) VALUES (1, 0);
```

- `action` is the Rust enum `AuditAction` with 38 names; `user.update` of Zig's first version is not one of them.
- `actor` is `NULL` when no account acted: bootstrap, refused sign-ins, deliveries and command completions. `subject` is `NULL` when the action has no numeric subject. Neither has a foreign key, because audit rows must outlive anything they name, and an append-only log takes no part in referential actions.

- `audit::append_audit` increments `last_id` and inserts the row with that identifier; it refuses with `Capacity` when `last_id` is $2^{63} - 1$. This replaces Zig's cursor trigger, which renumbered the inserted row, and the `sibuna_meta` key. `AUTOINCREMENT` alone would keep identifiers unique on SQLite but, on PostgreSQL, an identity sequence hands out identifiers in start order, not commit order, and a reader following the audit stream by identifier would skip a row that commits late.
- Summaries are redacted JSON objects of at most 2,048 bytes. They never contain passwords, token values, TOTP seeds, recovery codes, cookies, webhook secrets, page markup or policy selectors, as SID 0007 requires. The two new indexes serve the Audit page's action and actor filters, which scanned by identifier in Zig.

### 3.4.4. Settings and retention

```
CREATE TABLE console_settings (
    key TEXT NOT NULL PRIMARY KEY CHECK (key IN ('notify.spike_factor', 'notify.spike_minimum',
        'retention.minutes', 'retention.rankings', 'retention.incidents', 'retention.audit')),
    value INTEGER NOT NULL CHECK (value >= 1 AND value <= CASE key
        WHEN 'notify.spike_factor' THEN 1000000 WHEN 'notify.spike_minimum' THEN 1000000
        WHEN 'retention.minutes' THEN 90 WHEN 'retention.rankings' THEN 7
        WHEN 'retention.incidents' THEN 30 WHEN 'retention.audit' THEN 365 END),
    revision INTEGER NOT NULL CHECK (revision >= 1),
    updated_at INTEGER NOT NULL CHECK (updated_at >= 0),
    updated_by INTEGER NOT NULL REFERENCES console_users (id)) STRICT, WITHOUT ROWID;

CREATE TABLE console_job_leases (
    job TEXT NOT NULL PRIMARY KEY CHECK (job IN ('retention', 'notifier')),
    node INTEGER NOT NULL CHECK (node BETWEEN 1 AND 8388607),
    boot BLOB NOT NULL CHECK (length(boot) = 16),
    fence INTEGER NOT NULL CHECK (fence >= 1),
    expires_at INTEGER NOT NULL CHECK (expires_at >= 0)) STRICT, WITHOUT ROWID;
```

- A setting is an integer with the per-key range of the catalogue in SID 0007. A missing row means the default (3, 100, 90, 7, 30 and 365 in key order). The Zig value was text of at most seven digits checked by two triggers; the `CHECK` now states the same bound.
- A lease is held for 30 seconds and renewed after 10. Taking an expired lease, or a lease held by another node or boot, increments `fence`; every write under the lease rechecks `job`, `fence`, `node`, `boot` and `expires_at` in its own transaction.

### 3.4.5. Policy workflow and revisions

```
CREATE TABLE console_policy_history (
    policy_id TEXT NOT NULL CHECK (length(CAST(policy_id AS BLOB)) BETWEEN 1 AND 128),
    revision INTEGER NOT NULL CHECK (revision >= 0),
    kind TEXT NOT NULL CHECK (kind IN ('baseline', 'edit')),
    actor INTEGER REFERENCES console_users (id),
    recorded_at INTEGER NOT NULL CHECK (recorded_at >= 0),
    document TEXT NOT NULL CHECK (CASE WHEN json_valid(document)
        THEN json_type(document) = 'object' AND length(CAST(document AS BLOB)) <= 4096
        ELSE 0 END),
    PRIMARY KEY (policy_id, revision),
    CHECK ((kind = 'baseline') = (actor IS NULL))) STRICT;
CREATE INDEX console_policy_history_time ON console_policy_history (recorded_at);

CREATE TABLE console_inspection_history (
    revision INTEGER PRIMARY KEY CHECK (revision >= 1),
    actor INTEGER NOT NULL REFERENCES console_users (id),
    recorded_at INTEGER NOT NULL CHECK (recorded_at >= 0),
    path_traversal TEXT NOT NULL CHECK (path_traversal IN ('disabled', 'audit', 'enforce')),
    sqli TEXT NOT NULL CHECK (sqli IN ('disabled', 'audit', 'enforce')),
    xss TEXT NOT NULL CHECK (xss IN ('disabled', 'audit', 'enforce')),
    rce TEXT NOT NULL CHECK (rce IN ('disabled', 'audit', 'enforce')),
    previous_path_traversal TEXT NOT NULL
        CHECK (previous_path_traversal IN ('disabled', 'audit', 'enforce')),
    previous_sqli TEXT NOT NULL CHECK (previous_sqli IN ('disabled', 'audit', 'enforce')),
    previous_xss TEXT NOT NULL CHECK (previous_xss IN ('disabled', 'audit', 'enforce')),
    previous_rce TEXT NOT NULL CHECK (previous_rce IN ('disabled', 'audit', 'enforce'))) STRICT;
```

- `revision` in both history tables is the policy revision that the edit committed. A baseline row records the rule as it stood at the expected revision before its first console edit; it has no actor.
- The inspection document of Zig was a canonical JSON object of exactly four mode names. Eight enumerated columns state that exactly, so the history table is typed instead of holding JSON.
- The policy history and inspection history stay separate tables: one is keyed by rule and revision and the other by revision alone, and merging them would need a nullable key.
- An import carries the whole set in one console request body: at most 128 documents (the engine's rule capacity) within the policy crate's file bound of 256 KiB. `policy::import_rules` validates the set by building the candidate engine and replaces the rules in one transaction, so nothing is staged. Zig staged the documents and their typed columns in chunks, because a replicated statement had a 64 KiB envelope and a trigger copied the columns.
- Both history tables follow the audit retention setting, except that a rule's newest history row and the newest inspection row are never deleted, so revert and comparison always have a baseline.
- The stage tables of Zig are removed (see "Motivation and scope").

### 3.4.6. Reputation and country actions

```
CREATE TABLE console_reputation_sources (
    ip_or_cidr TEXT NOT NULL PRIMARY KEY
        REFERENCES ip_reputation (ip_or_cidr) ON DELETE CASCADE,
    country TEXT CHECK (country GLOB '[A-Z][A-Z]'),
    geo_generation BLOB CHECK (length(geo_generation) = 32),
    note TEXT NOT NULL DEFAULT '' CHECK (length(CAST(note AS BLOB)) <= 128),
    CHECK ((country IS NULL) = (geo_generation IS NULL)),
    CHECK (country IS NULL OR note = '')) STRICT, WITHOUT ROWID;
CREATE INDEX console_reputation_country ON console_reputation_sources (country, ip_or_cidr)
    WHERE country IS NOT NULL;

CREATE TABLE console_country_chunks (
    digest BLOB NOT NULL CHECK (length(digest) = 32),
    ordinal INTEGER NOT NULL CHECK (ordinal BETWEEN 0 AND 1023),
    prefix TEXT NOT NULL CHECK (length(prefix) BETWEEN 1 AND 48),
    recorded_at INTEGER NOT NULL CHECK (recorded_at >= 0),
    PRIMARY KEY (digest, ordinal),
    UNIQUE (digest, prefix)) STRICT, WITHOUT ROWID;
```

- A row of `console_reputation_sources` says that the console owns a reputation prefix: a console edit (`country` is `NULL`) or a country action (`country` and the pinned generation are set). A prefix without a row belongs to the data plane or to an operator. Zig added the columns `source`, `note` and `geo_generation` to `ip_reputation` and encoded ownership as the text `console` or `console:country:XX`. A separate table keeps the core table and its row type exactly as SID 0011 specifies, gives a node without the console no unknown columns, and states ownership with typed columns. A direct deletion of a reputation row removes its ownership through the cascade.
- A country set holds at most 1,024 prefixes; the `UNIQUE` key serves the membership test of S18 and replaces Zig's index on `(digest, prefix)`.

### 3.4.7. Pages

```
CREATE TABLE console_pages (
    kind TEXT NOT NULL PRIMARY KEY
        CHECK (kind IN ('challenge', 'denied', 'rate_limited', 'banned', 'overloaded')),
    html TEXT NOT NULL CHECK (length(CAST(html AS BLOB)) <= 16384),
    sha256 BLOB NOT NULL CHECK (length(sha256) = 32),
    revision INTEGER NOT NULL CHECK (revision >= 1),
    updated_at INTEGER NOT NULL CHECK (updated_at >= 0),
    updated_by INTEGER NOT NULL REFERENCES console_users (id)) STRICT;
```

- A missing row means the built-in template at revision 0. A reset deletes the row. `pages::change_page` compiles the template with the policy crate before it writes, and increments `policy_revision` itself; on PostgreSQL it also issues `pg_notify('sibuna_policy', '')`, because no core trigger fires for this table.

### 3.4.8. Notifications and deliveries

```
CREATE TABLE console_notifications (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    kind TEXT NOT NULL CHECK (kind IN ('webhook', 'syslog')),
    transport TEXT NOT NULL CHECK (transport IN ('udp', 'tcp')),
    label TEXT NOT NULL CHECK (length(CAST(label AS BLOB)) BETWEEN 1 AND 64),
    target TEXT NOT NULL CHECK (length(CAST(target AS BLOB)) BETWEEN 1 AND 256),
    target_host TEXT NOT NULL CHECK (length(CAST(target_host AS BLOB)) <= 128),
    secret_envelope BLOB CHECK (length(secret_envelope) BETWEEN 1 AND 128),
    events INTEGER NOT NULL CHECK (events BETWEEN 1 AND 15),
    cooldown_seconds INTEGER NOT NULL CHECK (cooldown_seconds BETWEEN 0 AND 86400),
    enabled INTEGER NOT NULL DEFAULT 1 CHECK (enabled IN (0, 1)),
    revision INTEGER NOT NULL DEFAULT 1 CHECK (revision >= 1),
    created_by INTEGER NOT NULL REFERENCES console_users (id),
    created_at INTEGER NOT NULL CHECK (created_at >= 0),
    modified_by INTEGER NOT NULL REFERENCES console_users (id),
    modified_at INTEGER NOT NULL CHECK (modified_at >= 0),
    last_attempt_at INTEGER CHECK (last_attempt_at >= 0),
    last_outcome TEXT CHECK (last_outcome IN ('delivered', 'failed')),
    last_detail TEXT CHECK (length(CAST(last_detail AS BLOB)) <= 96)) STRICT;

CREATE TABLE console_notification_events (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    node INTEGER NOT NULL CHECK (node BETWEEN 1 AND 8388607),
    boot BLOB NOT NULL CHECK (length(boot) = 16),
    sequence INTEGER NOT NULL CHECK (sequence >= 0),
    event TEXT NOT NULL
        CHECK (event IN ('denial_spike', 'ban', 'node_unhealthy', 'leader_change')),
    raised_at INTEGER NOT NULL CHECK (raised_at >= 0),
    detail TEXT NOT NULL CHECK (length(CAST(detail AS BLOB)) <= 128),
    completed_at INTEGER CHECK (completed_at >= raised_at),
    UNIQUE (node, boot, sequence)) STRICT;
CREATE INDEX console_notification_completed
    ON console_notification_events (completed_at, id) WHERE completed_at IS NOT NULL;
CREATE INDEX console_notification_pending
    ON console_notification_events (id) WHERE completed_at IS NULL;

CREATE TABLE console_notification_deliveries (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    event_id INTEGER NOT NULL
        REFERENCES console_notification_events (id) ON DELETE CASCADE,
    destination_id INTEGER REFERENCES console_notifications (id) ON DELETE SET NULL,
    destination_revision INTEGER NOT NULL CHECK (destination_revision >= 1),
    state TEXT NOT NULL DEFAULT 'pending'
        CHECK (state IN ('pending', 'sending', 'delivered', 'failed', 'skipped')),
    attempts INTEGER NOT NULL DEFAULT 0 CHECK (attempts BETWEEN 0 AND 3),
    next_attempt_at INTEGER NOT NULL DEFAULT 0 CHECK (next_attempt_at >= 0),
    claim_fence INTEGER CHECK (claim_fence >= 1),
    claim_expires_at INTEGER,
    updated_at INTEGER NOT NULL CHECK (updated_at >= 0),
    detail TEXT NOT NULL DEFAULT '' CHECK (length(CAST(detail AS BLOB)) <= 96),
    UNIQUE (event_id, destination_id),
    CHECK (state <> 'sending' OR (claim_fence IS NOT NULL AND claim_expires_at IS NOT NULL)),
    CHECK (destination_id IS NOT NULL OR state IN ('delivered', 'failed', 'skipped'))) STRICT;
CREATE INDEX console_delivery_due
    ON console_notification_deliveries (state, next_attempt_at, id);
CREATE INDEX console_delivery_destination
    ON console_notification_deliveries (destination_id, destination_revision, state);
```

- `events` is the set of event kinds as bits (denial spike 1, ban 2, node unhealthy 4, leader change 8). The secret envelope is sealed under the console key and bound to the target; S22 drops it when the target changes without a new secret.
- `completed_at` replaces Zig's `delivered_at`, which also marked events that had no destination or whose deliveries all failed. The legacy event-wide `attempts` column is gone.
- The foreign keys replace two triggers: deliveries of a deleted event go with it (S27), and a removed destination leaves its finished deliveries as history with a `NULL` reference. The second `CHECK` refuses the removal of a destination that still has pending or sending deliveries, so S23 must skip them first.
- A delivery is retried after 2 seconds and then after 8, at most three attempts. A claim is refused when the lease has less than 12 seconds left.

### 3.4.9. Node membership and commands

```
CREATE TABLE console_nodes (
    node INTEGER PRIMARY KEY CHECK (node BETWEEN 1 AND 8388607),
    address TEXT NOT NULL CHECK (length(CAST(address AS BLOB)) BETWEEN 1 AND 64),
    console_url TEXT NOT NULL DEFAULT '' CHECK (length(CAST(console_url AS BLOB)) <= 128),
    version TEXT NOT NULL CHECK (length(CAST(version AS BLOB)) BETWEEN 1 AND 32),
    boot BLOB NOT NULL CHECK (length(boot) = 16),
    first_seen INTEGER NOT NULL CHECK (first_seen >= 0),
    last_seen INTEGER NOT NULL,
    applied_revision INTEGER NOT NULL DEFAULT 0 CHECK (applied_revision >= 0),
    control_revision INTEGER NOT NULL DEFAULT 0 CHECK (control_revision >= 0),
    draining INTEGER NOT NULL DEFAULT 0 CHECK (draining IN (0, 1)),
    CHECK (last_seen >= first_seen)) STRICT;

CREATE TABLE console_commands (
    id BLOB NOT NULL PRIMARY KEY CHECK (length(id) = 16),
    node INTEGER NOT NULL REFERENCES console_nodes (node),
    boot BLOB NOT NULL CHECK (length(boot) = 16),
    kind TEXT NOT NULL CHECK (kind IN ('drain', 'resume', 'clear_local_bans')),
    actor INTEGER NOT NULL REFERENCES console_users (id),
    actor_role TEXT NOT NULL CHECK (actor_role IN ('operator', 'admin')),
    expected_revision INTEGER NOT NULL CHECK (expected_revision >= 0),
    state TEXT NOT NULL DEFAULT 'intent' CHECK (state IN ('intent', 'applied', 'rejected')),
    requested_at INTEGER NOT NULL CHECK (requested_at >= 0),
    completed_at INTEGER,
    applied_revision INTEGER CHECK (applied_revision >= 0),
    cleared_entries INTEGER CHECK (cleared_entries >= 0),
    CHECK ((state = 'intent') = (completed_at IS NULL)),
    CHECK (completed_at IS NULL OR completed_at >= requested_at),
    CHECK (cleared_entries IS NULL OR kind = 'clear_local_bans')) STRICT, WITHOUT ROWID;
CREATE INDEX console_commands_node ON console_commands (node, requested_at, id);
```

- Each node writes only its own membership row: at start, every 60 seconds, and after every applied rebuild, coalesced to at most one write per 2 seconds and retried after 30 seconds on failure. `first_seen` is kept on update.
- `address` is the node's management address, or `local` for a single node. The Zaxonlite log frontiers `applied_slot` and `decided_slot` are dropped from the table, the console protocol and the interface, because the Rust port has no consensus log; a node row carries its applied policy revision instead.
- A command targets a node that has a membership row, hence the foreign key.

### 3.4.10. GeoIP generations

```
CREATE TABLE console_geo_generations (
    digest BLOB NOT NULL PRIMARY KEY CHECK (length(digest) = 32),
    provider TEXT NOT NULL CHECK (provider IN ('dbip', 'user-country')),
    source_version TEXT NOT NULL CHECK (length(source_version) BETWEEN 1 AND 10),
    source_digests BLOB CHECK (length(source_digests) IN (32, 64)),
    ranges INTEGER NOT NULL CHECK (ranges BETWEEN 1 AND 1048576),
    actor INTEGER NOT NULL REFERENCES console_users (id),
    created_at INTEGER NOT NULL CHECK (created_at >= 0)) STRICT;

CREATE TABLE console_geo_chunks (
    digest BLOB NOT NULL REFERENCES console_geo_generations (digest) ON DELETE CASCADE,
    ordinal INTEGER NOT NULL CHECK (ordinal BETWEEN 0 AND 10485),
    ranges BLOB NOT NULL
        CHECK (length(ranges) BETWEEN 34 AND 3400 AND length(ranges) % 34 = 0),
    PRIMARY KEY (digest, ordinal)) STRICT;

CREATE TABLE console_geo_active (
    id INTEGER PRIMARY KEY CHECK (id = 1),
    revision INTEGER NOT NULL CHECK (revision >= 0),
    digest BLOB REFERENCES console_geo_generations (digest),
    loaded_at INTEGER CHECK (loaded_at >= 0),
    actor INTEGER REFERENCES console_users (id),
    CHECK ((digest IS NULL) = (loaded_at IS NULL)),
    CHECK ((digest IS NULL) = (actor IS NULL)),
    CHECK ((digest IS NULL) = (revision = 0))) STRICT;
INSERT INTO console_geo_active (id, revision) VALUES (1, 0);
```

- A chunk holds up to 100 ranges of 34 bytes (first address, last address, both 16 bytes, and a 2-byte country code), in ascending order across chunks; a generation has at most 1,048,576 ranges. `source_digests` holds the publisher digests in provider file order when the provider publishes them.
- At most two generations exist: the active one and one being imported. The foreign key from `console_geo_active` makes the active generation impossible to delete, and the cascade removes a pruned generation's chunks.

### 3.4.11. Observations: minutes, challenges, rankings and rule hits

```
CREATE TABLE console_minutes (
    node INTEGER NOT NULL CHECK (node BETWEEN 1 AND 8388607),
    boot BLOB NOT NULL CHECK (length(boot) = 16),
    epoch INTEGER NOT NULL CHECK (epoch BETWEEN 1 AND 4294967295),
    minute INTEGER NOT NULL CHECK (minute >= 0),
    start_ms INTEGER NOT NULL CHECK (start_ms >= 0),
    end_ms INTEGER NOT NULL,
    sealed INTEGER NOT NULL CHECK (sealed IN (0, 1)),
    payload BLOB NOT NULL
        CHECK (length(payload) = 176 AND substr(payload, 1, 4) = X'53424D32'),
    PRIMARY KEY (node, boot, epoch, minute),
    CHECK (end_ms > start_ms)) STRICT, WITHOUT ROWID;
CREATE INDEX console_minutes_time ON console_minutes (minute, node, boot, epoch);
CREATE INDEX console_minutes_node_time ON console_minutes (node, minute, boot, epoch);

CREATE TABLE console_challenge_minutes (
    node INTEGER NOT NULL CHECK (node BETWEEN 1 AND 8388607),
    boot BLOB NOT NULL CHECK (length(boot) = 16),
    epoch INTEGER NOT NULL CHECK (epoch BETWEEN 1 AND 4294967295),
    minute INTEGER NOT NULL CHECK (minute >= 0),
    start_ms INTEGER NOT NULL CHECK (start_ms >= 0),
    end_ms INTEGER NOT NULL,
    sealed INTEGER NOT NULL CHECK (sealed IN (0, 1)),
    payload BLOB NOT NULL CHECK (length(payload) BETWEEN 128 AND 896
        AND (length(payload) - 128) % 96 = 0 AND substr(payload, 1, 4) = X'53424331'),
    PRIMARY KEY (node, boot, epoch, minute),
    CHECK (end_ms > start_ms)) STRICT, WITHOUT ROWID;
CREATE INDEX console_challenge_minutes_time
    ON console_challenge_minutes (minute, node, boot, epoch);
CREATE INDEX console_challenge_minutes_node_time
    ON console_challenge_minutes (node, minute, boot, epoch);

CREATE TABLE console_challenge_records (
    id INTEGER PRIMARY KEY,
    node INTEGER NOT NULL CHECK (node BETWEEN 1 AND 8388607),
    boot BLOB NOT NULL CHECK (length(boot) = 16),
    second INTEGER NOT NULL CHECK (second >= 0),
    ip TEXT NOT NULL CHECK (length(ip) BETWEEN 1 AND 48),
    outcome TEXT NOT NULL CHECK (outcome IN ('issued', 'accepted', 'rejected')),
    cause INTEGER CHECK (cause BETWEEN 0 AND 12),
    algorithm TEXT NOT NULL CHECK (algorithm IN ('hashcash', 'posw')),
    parameter INTEGER NOT NULL CHECK (parameter BETWEEN 0 AND 255),
    openings INTEGER NOT NULL CHECK (openings BETWEEN 0 AND 255),
    duration_ms INTEGER CHECK (duration_ms BETWEEN 0 AND 4294967295),
    CHECK ((outcome = 'rejected') = (cause IS NOT NULL))) STRICT;
CREATE INDEX console_challenge_records_time ON console_challenge_records (second, id);
CREATE INDEX console_challenge_records_cause
    ON console_challenge_records (cause, second, id);
CREATE INDEX console_challenge_records_ip ON console_challenge_records (ip, second, id);

CREATE TABLE console_challenge_difficulty (
    node INTEGER NOT NULL CHECK (node BETWEEN 1 AND 8388607),
    boot BLOB NOT NULL CHECK (length(boot) = 16),
    second INTEGER NOT NULL CHECK (second >= 0),
    previous_bits INTEGER NOT NULL CHECK (previous_bits BETWEEN 0 AND 255),
    bits INTEGER NOT NULL CHECK (bits BETWEEN 0 AND 255),
    rate_256 INTEGER NOT NULL CHECK (rate_256 >= 0),
    PRIMARY KEY (node, boot, second)) STRICT, WITHOUT ROWID;
CREATE INDEX console_challenge_difficulty_time
    ON console_challenge_difficulty (second, node, boot);

CREATE TABLE console_rank_usage (
```

```
    id INTEGER PRIMARY KEY CHECK (id = 1),
    charged_bytes INTEGER NOT NULL CHECK (charged_bytes BETWEEN 0 AND 536870912)) STRICT;
INSERT INTO console_rank_usage (id, charged_bytes) VALUES (1, 0);

CREATE TABLE console_rank_archives (
    digest BLOB NOT NULL PRIMARY KEY CHECK (length(digest) = 32),
    node INTEGER NOT NULL CHECK (node BETWEEN 1 AND 8388607),
    boot BLOB NOT NULL CHECK (length(boot) = 16),
    minute INTEGER NOT NULL CHECK (minute >= 0),
    bytes BLOB NOT NULL CHECK (length(bytes) BETWEEN 92 AND 48536),
    charge INTEGER NOT NULL CHECK (charge = (length(bytes) + 2047) / 2048 * 8192 + 1024),
    stored_at INTEGER NOT NULL CHECK (stored_at >= 0),
    UNIQUE (node, boot, minute)) STRICT;
CREATE INDEX console_rank_minute ON console_rank_archives (minute, digest);
CREATE INDEX console_rank_node ON console_rank_archives (node, minute, digest);

CREATE TABLE console_rule_hits (
    node INTEGER NOT NULL CHECK (node BETWEEN 1 AND 8388607),
    boot BLOB NOT NULL CHECK (length(boot) = 16),
    sequence INTEGER NOT NULL CHECK (sequence >= 0),
    rule_key TEXT NOT NULL CHECK (length(CAST(rule_key AS BLOB)) BETWEEN 1 AND 160),
    generation INTEGER NOT NULL CHECK (generation >= 0),
    revision INTEGER NOT NULL CHECK (revision >= 0),
    name TEXT NOT NULL CHECK (length(CAST(name AS BLOB)) <= 128),
    minute INTEGER NOT NULL CHECK (minute >= 0),
    utc_start INTEGER NOT NULL CHECK (utc_start >= 0),
    utc_end INTEGER NOT NULL,
    start_ms INTEGER NOT NULL CHECK (start_ms >= 0),
    end_ms INTEGER NOT NULL,
    observed_ms INTEGER NOT NULL CHECK (observed_ms >= 0),
    observations INTEGER NOT NULL CHECK (observations >= 0),
    complete INTEGER NOT NULL CHECK (complete IN (0, 1)),
    gap INTEGER NOT NULL CHECK (gap IN (0, 1)),
    unconfirmed INTEGER NOT NULL CHECK (unconfirmed >= 0),
    hits INTEGER CHECK (hits >= 0),
    PRIMARY KEY (node, boot, sequence, rule_key),
    CHECK (utc_end >= utc_start AND end_ms >= start_ms)) STRICT;
CREATE INDEX console_rule_hits_history
    ON console_rule_hits (rule_key, node, minute, boot, sequence);
CREATE INDEX console_rule_hits_revision
    ON console_rule_hits (rule_key, node, revision, minute, boot, sequence);
CREATE INDEX console_rule_hits_retention ON console_rule_hits (minute);

CREATE TABLE console_rule_hit_rollups (
    grain INTEGER NOT NULL CHECK (grain IN (60, 1440)),
    node INTEGER NOT NULL CHECK (node BETWEEN 1 AND 8388607),
    boot BLOB NOT NULL CHECK (length(boot) = 16),
    generation INTEGER NOT NULL CHECK (generation >= 0),
    rule_key TEXT NOT NULL CHECK (length(CAST(rule_key AS BLOB)) BETWEEN 1 AND 160),
    bucket INTEGER NOT NULL CHECK (bucket >= 0 AND bucket % grain = 0),
    revision INTEGER NOT NULL CHECK (revision >= 0),
    name TEXT NOT NULL CHECK (length(CAST(name AS BLOB)) <= 128),
    hits INTEGER CHECK (hits >= 0),
    observed_ms INTEGER NOT NULL CHECK (observed_ms >= 0),
    intervals INTEGER NOT NULL CHECK (intervals >= 1),
    complete INTEGER NOT NULL CHECK (complete >= 0),
    gaps INTEGER NOT NULL CHECK (gaps >= 0),
    unconfirmed INTEGER NOT NULL CHECK (unconfirmed >= 0),
    through_minute INTEGER NOT NULL
        CHECK (through_minute >= bucket AND through_minute < bucket + grain),
    PRIMARY KEY (grain, node, boot, generation, rule_key, bucket)) STRICT;
CREATE INDEX console_rule_rollup_history
    ON console_rule_hit_rollups (rule_key, node, grain, bucket, boot, generation);
CREATE INDEX console_rule_rollup_retention ON console_rule_hit_rollups (bucket, grain);
```

- Minute and challenge-minute payloads stay versioned binary records (`SBM2` of 176 bytes; `SBC1` of 128 bytes plus 96 bytes for each of at most eight partitions), because their counters are unsigned 64-bit values that exceed SQL's signed integers. The `CHECK` on the magic prefix refuses another format. The Rust port writes only `SBM2`; `SBM1` rows existed only in Zig databases.
- The two minute tables stay separate: their payloads, readers and summaries differ, and a `kind` column would make every query and index longer without a shared reader.

- A challenge record's `cause` is the rejection cause ordinal (13 causes) and is `NULL` unless the outcome is a rejection; Zig stored `255` for "no cause". The Rust record type maps its `NO_CAUSE` constant to `NULL` at the storage boundary. `algorithm` uses the names of SID 0011's `ChallengeAlgorithm`.
- An archive is one row holding its whole `SBR2` record as one `BLOB` of 92 to 48,536 bytes (SID 0007 as corrected for the Rust port). Zig reserved an archive, wrote it in 2,048-byte chunks and published it in a separate step, because a replicated statement had a 64 KiB envelope and the mailbox carried bounded payloads; neither exists here, so the chunk table, the reservation state and the bound on pending reservations are gone. `rankings::store_archive` checks the digest and the format, takes the quota and inserts the row in one transaction (S32). Zig also refused chunk ordinals from 19, a bound left from the 37,468-byte `SBR1` format, so an `SBR2` archive above 38,912 bytes could not be stored; the Rust bound follows the format.
- The `CHECK` on `charge` keeps the charging formula of SID 0007 unchanged: whole 2,048-byte units at 8,192 bytes each plus 1,024 bytes. Retention arithmetic and the 512 MiB quota therefore keep their meaning.
- `console_rank_usage` stays a counter row instead of a sum over archives, because on PostgreSQL its conditional update (`SET charged_bytes = charged_bytes + $charge WHERE charged_bytes + $charge <= 536870912`) is the atomic quota check across nodes. S34 keeps it equal to the sum, and a test checks the equality.
- Rule hits keep the `(node, boot, sequence, rule_key)` identity. Zig refused a conflicting replay through a `guard` column whose `CHECK` failed on update; S35 inserts with `ON CONFLICT DO NOTHING`, compares the stored row of every conflict, and aborts the transaction on a difference. A hit count is `NULL` when its counter was exhausted or a roll-up sum would overflow.

### 3.4.12. Incident sidecars: evidence, heads and country

```
CREATE TABLE console_incident_evidence (
    incident_id INTEGER PRIMARY KEY REFERENCES security_incidents (id) ON DELETE CASCADE,
    selected_status INTEGER NOT NULL CHECK (selected_status BETWEEN 100 AND 599),
    query_bytes INTEGER NOT NULL CHECK (query_bytes BETWEEN 0 AND 4294967295),
    body_bytes INTEGER NOT NULL CHECK (body_bytes BETWEEN 0 AND 4294967295),
    declared_body_bytes INTEGER NOT NULL
        CHECK (declared_body_bytes BETWEEN 0 AND 4294967295),
    truncated INTEGER NOT NULL CHECK (truncated BETWEEN 0 AND 255)) STRICT;

CREATE TABLE console_incident_heads (
    incident_id INTEGER PRIMARY KEY REFERENCES security_incidents (id) ON DELETE CASCADE,
    request_head BLOB NOT NULL CHECK (length(request_head) <= 2048),
    response_head BLOB NOT NULL CHECK (length(response_head) <= 1024),
    request_truncated INTEGER NOT NULL CHECK (request_truncated IN (0, 1)),
    response_truncated INTEGER NOT NULL CHECK (response_truncated IN (0, 1)),
    response_state TEXT NOT NULL
        CHECK (response_state IN ('captured', 'local', 'unobserved', 'unavailable')),
    CHECK (response_state = 'captured'
        OR (length(response_head) = 0 AND response_truncated = 0))) STRICT;

CREATE TABLE console_incident_country (
    incident_id INTEGER PRIMARY KEY REFERENCES security_incidents (id) ON DELETE CASCADE,
    country TEXT CHECK (country GLOB '[A-Z][A-Z]'),
    generation BLOB NOT NULL CHECK (length(generation) = 32)) STRICT;
CREATE INDEX console_incident_country_code ON console_incident_country (country, incident_id);
```

- The sidecars stay three tables, because they are written under different conditions (evidence whenever the console captures, heads only with `--console-capture-heads`, country only with a loaded GeoIP generation). One table would need groups of nullable columns, each all set or all absent, which is weaker typing than three tables of `NOT NULL` columns.
- The evidence `version` column of Zig held the constant 1 and is removed; a new evidence format is a new console migration.
- Heads are the redacted bytes themselves, no longer hex text. The state `unknown` labelled Zig rows written before version 40 and cannot occur.
- A missing sidecar row means "not recorded". A country row with `country NULL` means the address mapped to no country in that generation (Unknown).

- The cascades replace two Zig triggers on `security_incidents`; the full-text row is deleted by the core function of S36.

### 3.4.13. Dialect differences

The PostgreSQL text of the console migration differs from the SQLite text only as follows.

**Table 28** Console dialect differences

| Concern | SQLite | PostgreSQL |
|---|---|---|
| column typing | `STRICT` tables | typed columns |
| integers | `INTEGER` | `BIGINT` for every integer column, with the same `CHECK`; the decoder reads `i64` on both backends |
| keys the database fills in (`console_users`, `console_tokens`, `console_notifications`, `console_notification_events`, `console_notification_deliveries`, `console_challenge_records`) | `INTEGER PRIMARY KEY`, with `AUTOINCREMENT` where identifiers must never be reused | `BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY` |
| single-row tables and the audit key | `INTEGER PRIMARY KEY` | `BIGINT PRIMARY KEY` |
| flags (`disabled`, `enabled`, `sealed`, `complete` and the like) | `INTEGER` holding 0 or 1 | `BOOLEAN`; roll-up `complete` counts stay `BIGINT` |
| byte strings | `BLOB`, bounded with `length`; a magic prefix with `substr(payload, 1, 4) = X'...'` | `BYTEA`, bounded with `octet_length`; `substring(payload FROM 1 FOR 4) = '\x...'::bytea` |
| text measured in bytes | `length(CAST(x AS BLOB))` | `octet_length(x)` |
| text keys ordered or ranged by bytes (`policy_id`, `rule_key`, `prefix`, `username`) | `TEXT` (`BINARY` collation) | `TEXT COLLATE "C"` |
| country codes | `country GLOB '[A-Z][A-Z]'` | `country ~ '^[A-Z]{2}$'` |
| JSON documents and summaries | `TEXT` checked with `json_valid` and `json_type` | `JSONB` checked with `jsonb_typeof`; the size bound with `octet_length(CAST(x AS TEXT))`, which measures the rendering (a space after every colon and comma), so a policy history document is bounded at 8,192 bytes there and at 4,096 bytes of compact text in Rust before the insert |
| table options | `WITHOUT ROWID` where stated | none |
| smaller or larger of two values | `MIN(a, b)`, `MAX(a, b)` | `LEAST(a, b)`, `GREATEST(a, b)` |
| bounded deletes | `DELETE ... WHERE key IN (SELECT key ... ORDER BY ... LIMIT n)` | the same, with `FOR UPDATE SKIP LOCKED` in the subquery |
| capacity checks | the single writer thread | `pg_advisory_xact_lock` with a per-table key |
| singleton counters (`policy_revision`, `console_audit_cursor`, `console_rank_usage`) | the single writer thread | row lock by `SELECT ... FOR UPDATE` or the conditional `UPDATE` |
| full-text search | `incidents_fts` with `tokenize = 'ascii'` and `MATCH` | `search tsvector` with a GIN index and `search @@ CAST($n AS tsquery)` |
| page change notification | polling | `pg_notify('sibuna_policy', '')` in S19 |

| Concern | SQLite | PostgreSQL |
|---|---|---|
| read work bound | progress handler counting virtual-machine steps | `SET LOCAL statement_timeout` |

### 3.5. Invariant catalogue

The table maps each of the 49 Zig triggers (schema version 40) to its replacement and to the test that proves it. Every test lives in `crates/console-storage/tests/invariants.rs`, runs through `for_each_backend` on SQLite and on PostgreSQL, and checks both the function's effect and, where a constraint is the second line, that the direct write breaking it is refused.

**Table 29** Zig triggers, their Rust replacements and their tests

| Zig trigger | What it enforced | Replacement | Test |
|---|---|---|---|
| `console_user_created` | audit `user.create` with the creator's role and a summary | S1 | `user_creation_is_audited`; `bootstrap_creates_only_the_first_account` |
| `console_user_updated` | audit of the authority change with before and after; delete the account's sessions | S2 | `authority_change_revokes_sessions_and_tokens` |
| `console_session_created` | audit `session.create` for cookie sessions with role and address | S3 | `sign_in_is_audited_and_records_last_login` |
| `console_login_recorded` | last sign-in time, never moving backwards | S3 | `sign_in_is_audited_and_records_last_login` |
| `console_totp_consumed` | a TOTP step or recovery slot used once | S3 | `totp_step_and_recovery_slot_are_single_use` |
| `console_session_ended` | audit `session.logout` and delete the session | S4 | `sign_out_is_audited_and_deletes_the_session` |
| `console_totp_enabled` | bump the revision (revoking through `console_user_updated`) and audit `totp.enable` | S5, S2 | `enabling_totp_revokes_and_audits_once` |
| `console_password_rotated` | replace the password, revoke, insert the replacement session, atomically | S6 | `password_change_rotates_atomically` |
| `console_token_created` | mirror session row and audit `token.create` | S7 | `token_creation_is_audited_and_bounded` |
| `console_token_immutable` | a token's authority never changes; the only update is disablement or removal of a disabled token | S7, S8 | `token_authority_is_immutable` |
| `console_token_changed` | delete the mirror session; audit `token.revoke` or `token.remove`; delete a removed token | S8, S9 | `token_revocation_is_audited`; `only_inactive_tokens_are_removed` |
| `console_token_deleted` | delete the mirror session | S9 (no mirror exists) | `only_inactive_tokens_are_removed` |
| `console_kiosk_capacity` | at most 64 outstanding grants | S10 | `kiosk_grants_are_bounded_and_audited` |
| `console_kiosk_granted` | audit `kiosk.grant` without the code | S10 | `kiosk_grants_are_bounded_and_audited` |
| `console_kiosk_exchanged` | audit `kiosk.exchange` (and, through the cookie-session triggers, `session.create`) | S11 | `kiosk_exchange_is_single_use` |
| `console_geo_activated` | audit `geoip.activate` | S12 | `geoip_activation_requires_complete_generation` |

| Zig trigger | What it enforced | Replacement | Test |
|---|---|---|---|
| `console_policy_commit` | baseline and edit history, rule upsert, redacted `policy.edit` | S13, S38 | `rule_edit_writes_history_audit_and_one_revision` |
| `console_inspection_commit` | inspection modes, history, `inspection.edit` | S14, S38 | `inspection_edit_writes_history_and_audit` |
| `console_policy_order_commit` | two priorities, two history rows, `policy.order`, revision set to expected plus one | S15, S38 | `ordering_swaps_two_rules_in_one_revision` |
| `console_reputation_commit` | prefix upsert or deletion with console ownership, audit, revision | S17, S38 | `reputation_change_records_ownership_and_audit` |
| `console_country_apply` | replace the country's owned set, audit counts before and after, delete the stage, revision | S18, S38 | `country_replacement_removes_unreviewed_prefixes` |
| `console_policy_import_apply` | replace every rule, history per rule, `policy.import`, delete the stage, revision | S16 (nothing is staged), S38 | `import_replaces_all_rules_in_one_revision` |
| `console_page_commit` | page upsert or deletion, `page.edit` or `page.reset`, revision bump | S19 | `page_change_audits_digests_and_bumps_revision` |
| `console_setting_changed`, `console_setting_updated` | audit `setting.change` with the old and new value | S20 | `setting_change_is_audited` |
| `console_retention_insert`, `console_retention_update` | retention values within the catalogue bounds | S20 and the `CHECK` on `console_settings.value` | `settings_outside_the_catalogue_are_refused` |
| `console_notification_capacity` | at most eight destinations | S21 | `destinations_are_bounded_and_audited` |
| `console_notification_created` | audit `notification.create` without the secret | S21 | `destinations_are_bounded_and_audited` |
| `console_notification_changed` | audit `notification.change` without the secret | S22 | `destination_change_skips_stale_deliveries` |
| `console_delivery_retargeted` | skip pending and sending deliveries of an old revision | S22 | `destination_change_skips_stale_deliveries` |
| `console_notification_removed` | audit `notification.remove` | S23 | `destination_removal_skips_and_audits` |
| `console_delivery_removed` | skip the removed destination's pending and sending deliveries | S23 and the `CHECK` on `destination_id` | `destination_removal_skips_and_audits` |
| `console_notification_events_bound` | purge old completed events; at most 256 pending and 4,096 stored | S24 | `event_queue_is_bounded` |
| `console_delivery_enqueued` | one delivery per enabled subscribed destination; complete an event without one | S24 | `events_fan_out_to_subscribed_destinations` |
| `console_delivery_changed` | destination attempt time on claim; outcome, `notification.delivery` and event completion on leaving `sending` | S25, S26 | `delivery_outcome_is_audited_and_completes_event` |
| `console_delivery_pruned` | delete an event's deliveries with it | S27 (foreign-key cascade) | `deliveries_go_with_their_event` |

| Zig trigger | What it enforced | Replacement | Test |
|---|---|---|---|
| `console_notification_tested` | record a test result on the destination at a matching revision | S28 | `test_result_updates_current_destination_only` |
| `console_audit_cursor` | audit identifiers monotonic across retention; exhaustion refused | S29 | `audit_ids_increase_after_retention` |
| `console_command_intent` | audit `node.command.intent` | S30 | `command_intent_is_audited_and_bounded` |
| `console_command_completed` | audit the completion once | S31 | `command_completes_once` |
| `console_rank_reserved` | add the charge to usage | S32 | `rankings_quota_is_enforced` |
| `console_rank_published` | remove the pending row | S32 (an archive is stored whole; S33 is merged into S32) | `archive_stores_only_verified_bytes` |
| `console_rank_abandoned`, `console_rank_expired` | delete chunks and release the charge | S34 (no chunks exist) | `usage_equals_charges_after_pruning` |
| `console_rule_hit_hour`, `console_rule_hit_day` | roll-ups with saturation and `NULL` on overflow | S35 | `rule_hits_roll_up_idempotently`; `conflicting_replay_aborts_batch` |
| `console_incident_deleted` | delete the full-text row, the vector row and the evidence sidecar | S36 and the sidecar cascades | `incident_deletion_removes_index_and_sidecars` |
| `console_incident_country_delete` | delete the country sidecar | S36 and the cascade | `incident_deletion_removes_index_and_sidecars` |

The audit rows stay as Sibuna writes them, with one side effect of Zig's trigger chains removed. Enabling TOTP writes both `totp.enable` and `user.revoke`: the account's sessions are revoked, and the trail says so (S5 through S2). A kiosk exchange writes `kiosk.exchange` and `session.create`, because a session was created (S11). It does not move the granting account's last sign-in time: in Zig that followed from the cookie-session trigger chain, but the account's owner did not sign in.

### 3.6. Search

The Events page has one search language over the path and the payload of incidents. The same language serves the core's `search_incidents` (SID 0011). The query string is parsed in Rust and translated, so both backends receive an expression whose meaning is fixed.

**Canonical tokens.** One Rust function, `search::tokens`, turns a field's bytes into tokens. Indexing and queries use it alike:

1. Percent-decode once: `%` followed by two hexadecimal digits becomes that byte; any other `%` stays.
2. Decode as UTF-8, replacing invalid sequences with U+FFFD.
3. A token character is an ASCII letter or digit, or a non-ASCII character for which `char::is_alphanumeric` holds. Every other character separates tokens.
4. Lowercase each token with `char::to_lowercase`.
5. Split a token longer than 64 bytes into pieces of at most 64 bytes at character boundaries.
6. When indexing, keep the first 128 tokens of each field.

The core writes the path's tokens and the payload's tokens, joined by single spaces, into the two columns of `incidents_fts`, whose `ascii` tokenizer then splits exactly at the spaces: it treats ASCII letters and digits and every non-ASCII character as token characters, and the canonical text contains no other characters besides

the separating spaces. On PostgreSQL the core writes the `tsvector` literal itself: path tokens at positions 1 to $n$ with weight A, payload tokens at the following positions with weight B. PostgreSQL never parses the text, so its default parser and its locale play no part. The 128-token bound keeps every lexeme under PostgreSQL's limit of 256 positions, so no position is dropped.

**Grammar.**

```
query  = group *( sp "OR" sp group )
group  = term *( sp term )
term   = [ field ":" ] ( word / phrase )
field  = "path" / "payload"
word   = 1*( any character except space and DQUOTE )   ; never the word OR alone
phrase = DQUOTE 1*( any character except DQUOTE ) DQUOTE
sp     = 1*" "
```

Juxtaposition is AND and binds tighter than `OR`; there are no parentheses, no negation and no prefix search. `OR` is an operator only in upper case; the token "or" is searched as `or` or `"OR"`. A word or phrase is reduced to its canonical tokens: one token matches that token, several tokens match them as a phrase (so `/etc/passwd` means the phrase `etc passwd`), and none is an error. A field prefix restricts a term to the path or to the payload; an unprefixed term matches either field, and a phrase always lies within one field.

**Bounds.** A query has at most 256 bytes, 8 groups, 16 terms, and 8 tokens per term. An empty query applies no search. A violation is an `InvalidInput` with an Elm-style diagnostic naming the byte offset, for example "This term has no searchable characters" with the hint to add letters or digits.

**Translation.** A token `t` is safe to quote on both backends, because it contains no quote, backslash or other ASCII punctuation.

**Table 30** Translation of the query language

| Construct | SQLite FTS5 `MATCH` | PostgreSQL `tsquery` |
|---|---|---|
| token `t` | `"t"` | `'t'` |
| phrase `t1 t2` | `"t1 t2"` | `('t1':A <-> 't2':A) \| ('t1':B <-> 't2':B)` |
| `path:` term | `path : "t"` or `path : "t1 t2"` | `'t':A` or `'t1':A <-> 't2':A` |
| `payload:` term | `offending_payload : "t"` | `'t':B` or `'t1':B <-> 't2':B` |
| group `a b` | `(a AND b)` | `(a & b)` |
| query `g1 OR g2` | `g1 OR g2` | `g1 \| g2` |

On PostgreSQL the expression is bound as `CAST($n AS tsquery)`, which takes the lexemes as written. A phrase in FTS5 matches consecutive tokens of one column; in PostgreSQL every lexeme of the translated phrase carries the same weight, so it matches consecutive positions within one field and never across the boundary between path and payload. Both backends therefore index the same tokens at the same relative positions and evaluate the same Boolean expression over them, and they return the same set of incidents for the same query.

**Ordering.** The Events page orders matches by `(recorded_at, id)` descending and pages by that key, as without a search, so both backends return the same rows on every page. The core's `search_incidents` orders by relevance: FTS5's `bm25` on SQLite and `ts_rank` on PostgreSQL, ties broken by identifier. The two functions weigh terms differently, so that order may differ between backends even though the set of matches is the same.

**Statement.** The search is one more Events filter, evaluated under the read work bound:

```
-- SQLite
AND s.id IN (SELECT rowid FROM incidents_fts WHERE incidents_fts MATCH $q)
-- PostgreSQL
AND s.search @@ CAST($q AS tsquery)
```

## 3.7. Retention and quotas

One Rust job, `retention::RetentionJob`, deletes expired rows on behalf of the whole cluster. It runs on every node and acts only while it holds the `retention` lease. Every 5 seconds it makes one pass over all kinds below. For each kind it deletes in batches of at most 256 rows, oldest first, each batch a separate storage

job and transaction that rechecks the lease and reads the retention setting; it stops a kind when a batch deletes fewer than 256 rows or after 8 batches (2,048 rows, about 400 rows per second per kind). Zig rotated one kind of 16 rows every 5 seconds, which removed incidents at under one row per second against an ingestion of up to 64 per second per node; the Rust bounds keep pace with ingestion while each transaction stays short. Zig also pruned minutes, rankings, GeoIP and rule hits on every node without a lease; under one lease the work is done once.

**Table 31** Retention kinds and their bounds

| Kind | Rows deleted | Bound |
|---|---|---|
| incidents | `security_incidents` recorded before now minus `retention.incidents` days, through `Database::delete_incidents` (S36) | 1 to 30 days, default 30 |
| audit | `console_audit` rows recorded before now minus `retention.audit` days | 1 to 365 days, default 365 |
| minutes | `console_minutes` and `console_challenge_minutes` with `minute` before the cutoff minute | `retention.minutes`, 1 to 90 days, default 90 |
| rule hits | `console_rule_hits` with `minute` before the cutoff; roll-ups whose `bucket + grain` is at most the cutoff | `retention.minutes` |
| rankings | archives with `minute` before now minus `retention.rankings` days; then the oldest archives while `charged_bytes` exceeds 536,673,280 (the quota less one largest charge, 197,632) | 1 to 7 days, default 7; 512 MiB charged |
| challenge records | `console_challenge_records` and `console_challenge_difficulty` older than 7 days; each batch write also deletes up to 64 expired records first | 7 days, fixed |
| sessions | sessions with `idle_expires_at` at or before now | 12 hours absolute, 30 minutes idle |
| kiosk grants | grants consumed or past `use_by` | 10 minutes to use |
| notification history | completed events older than 7 days, with their deliveries (S27) | 7 days; at most 4,096 stored |
| policy history | `console_policy_history` and `console_inspection_history` rows recorded before now minus `retention.audit` days, except the newest row of each rule and the newest inspection row | `retention.audit` |
| staged chunks | `console_country_chunks` older than 600 seconds | 10 minutes |
| GeoIP | chunks, then the generation row, of every generation that is not active and was created before the active one was loaded or more than an hour ago | 2 generations |
| node commands | commands requested more than 30 days ago, except an intent of the target node's current boot | 30 days; 4,096 per node |

The quotas checked at insertion are: 1,024 accounts; 1,024 tokens; 4,096 live sessions; 64 outstanding kiosk grants; 8 notification destinations; 256 pending and 4,096 stored notification events; 4,096 commands per node; 2 GeoIP generations of at most 1,048,576 ranges; 512 MiB of charged ranking bytes; 128 documents per import, 1,024 prefixes per country set and 16 country sets staged within ten minutes. The charged quota bounds the stored archives, not the database file; journal and index costs remain separate, as SID 0007 states. Policy and inspection history follow the audit retention setting. Zig kept them for ever; they grow with human edits, but an automated client with a token could grow them quickly. A rule's newest history row is never deleted, so revert and comparison always have a baseline.

SID 0011 named "a retention delete cannot remove an incident under investigation" as an example invariant. It has no counterpart here: the Zig console had no such rule, and a hold on incidents would be a new feature.

### 3.8. Reads and writes per route group

Every authenticated route first reads its credential (`console_sessions` or `console_tokens`, with `console_users` and `console_totp`) through S37; the table omits these reads. Every mutation and export writes `console_audit` and `console_audit_cursor` through S29. Paths are under `/console/api/`.

**Table 32** Tables per route group

| Group | Routes | Reads | Writes |
|---|---|---|---|
| setup and sign-in | `setup`, `login`, `logout`, `session`, `password` | `console_users`, `console_sessions`, `console_totp` | `console_users`, `console_sessions`, `console_totp` (S1, S3, S4, S6); refused sign-ins audit only |
| two-factor | `totp`, `totp/enroll`, `totp/confirm` | `console_totp` | `console_totp`, `console_users`, `console_sessions` (S5) |
| users | `users/query`, `users/create`, `users/change` | `console_users`, `console_totp` | `console_users`, `console_sessions` (S1, S2) |
| tokens | `tokens/query`, `tokens/create`, `tokens/revoke` | `console_tokens`, `console_users` | `console_tokens` (S7, S8, S9) |
| kiosk | `kiosk/token`, `kiosk/exchange` | `console_kiosk_grants`, `console_users` | `console_kiosk_grants`, `console_sessions` (S10, S11) |
| audit | `audit/query`, `audit/read`, `audit/export` | `console_audit` | audit only for exports (S39) |
| events | `events/query`, `events/export`, `events/heads`, `events/similar` | `security_incidents`, `incidents_fts` or `search`, `console_incident_evidence`, `console_incident_country`, `console_incident_heads` | audit only for exports (S39) |
| security statistics | `security/query`, `security/trends` | `security_incidents`, `console_incident_country` | none |
| traffic statistics | `stats`, `timeline`, `minutes`, `minutes/summary` | `console_minutes`, `console_settings` | none |
| challenges | `challenges`, `challenges/summary`, `challenges/records`, `challenges/difficulty` | `console_challenge_minutes`, `console_challenge_records`, `console_challenge_difficulty`, `console_settings` | none |
| rankings | `rankings`, `rankings/query`, `rankings/history` | `console_rank_archives`, `console_rank_usage`, `console_settings` | none |
| policy | `policies/query`, `policies/read`, `policies/test`, `policies/replay`, `policies/hits` | `policies`, `policy_inspection`, `ip_reputation`, `policy_revision`, `console_pages`, `console_policy_history`, `console_rule_hits`, `console_rule_hit_rollups`, `security_incidents`, `console_incident_evidence` | none |
| policy edits | `policies/edit`, `policies/order`, `policies/import`, `inspection/edit` | the policy reads above | `policies`, `policy_inspection`, `policy_revision`, `console_policy_history`, `console_inspection_history` (S13 to S16) |
| reputation and country actions | `reputation/query`, `reputation/edit`, `reputation/remove`, | `ip_reputation`, `console_reputation_sources`, `console_country_chunks`, | `ip_reputation`, `console_reputation_sources`, |

| Group | Routes | Reads | Writes |
|---|---|---|---|
| | `geoip/country/preview`, `geoip/country/apply` | `policies`, `policy_revision`, `console_geo_active` | `console_country_chunks`, `policy_revision` (S17, S18) |
| GeoIP | `geoip` (status and update) | `console_geo_active`, `console_geo_generations`, `console_geo_chunks` | `console_geo_generations`, `console_geo_chunks`, `console_geo_active` (S12) |
| pages | `pages/read`, `pages/preview/<kind>`, `pages/edit` | `console_pages` | `console_pages`, `policy_revision` (S19) |
| settings | `settings/query`, `settings/change` | `console_settings` | `console_settings` (S20) |
| notifications | `notifications/query`, `notifications/save`, `notifications/remove`, `notifications/test` | `console_notifications` | `console_notifications`, `console_notification_deliveries` (S21 to S23, S28) |
| nodes | `nodes`, `nodes/local`, `nodes/command`, `nodes/command/read` | `console_nodes`, `console_commands`, `policy_revision` | `console_commands` (S30) |
| about | `about` | none | none |
| live stream | `/console/ws` topics `stats`, `events`, `nodes`, `policy`, `challenges`, `audit` | `security_incidents`, `incident_receipts`, `console_incident_country`, `console_audit`, `console_nodes`, `policy_revision` | none |
| background jobs (no route) | retention, notifier, membership, minute, challenge, ranking and rule-hit journals, command completion, incident sidecars | the rows they maintain, `console_job_leases`, `console_settings` | the tables of “Retention and quotas”; `console_notification_events` and deliveries (S24 to S27); `console_nodes`; the observation tables (S32, S34, S35, S40); `console_commands` (S31); the sidecars |

## 4. Rationale and alternatives

**Why not port the triggers.** A trigger port would be two programs, one per dialect, with different firing rules: SQLite fires row triggers per row, PostgreSQL statement triggers once per statement and row triggers in name order. Zig’s triggers also chained (a TOTP change fired a user update, which fired a session deletion and an audit), and one chain moved the granting account’s last sign-in time on a kiosk exchange, a side effect noted in the catalogue. A Rust function states the whole invariant in one place, in the language its readers already read, with typed results instead of change counts that include trigger writes.

**Why one function per invariant.** It makes the rule checkable: a review finds every write of a governed table in one function, and the test of that function is the test of the invariant. Functions that need the same effect call the enforcing function (S6 calls S2 and S3) rather than repeating its statements.

**Why constraints as well.** A function can have a defect, and an operator can write SQL. `NOT NULL`, `CHECK` and foreign keys refuse rows that no Rust value could produce, on both dialects, at no cost to the functions.

**Why not ``SERIALIZABLE`` on PostgreSQL.** Serializable isolation would make the capacity checks and counters correct without explicit locks, but every function would need a retry loop for serialization failures, including those that call the network afterwards. Three kinds of explicit lock, taken in one order, are fewer moving parts.

**Why an audit cursor row.** Identity columns and `AUTOINCREMENT` give unique identifiers but, on PostgreSQL, not in commit order. The Audit stream reads by identifier, so the cursor row orders appends cluster-wide. Audit appends are rare management events, so the serialisation costs nothing measurable.

**Why canonical tokens.** FTS5's tokenizers and PostgreSQL's parser disagree on URLs, paths, numbers, hyphens and non-ASCII letters, and PostgreSQL's parser depends on the database locale. Tokenizing once in Rust and giving both backends the finished tokens is the only way to state "the same matches" as a property we can test. Percent-decoding once lets an operator find `union select` in `UNION%20SELECT`.
**Why keep binary payloads for observations.** Decomposing minute payloads into columns would lose unsigned 64-bit counters at $2^{63}$ and add hundreds of columns whose only reader is the codec that already validates the binary record.
**Alternatives for reputation ownership.** Adding columns to `ip_reputation` from the console migration, as Zig did, is shorter but makes a core table's shape depend on whether the console is built, and keeps ownership as parsed text. The separate table costs one join on the reputation page.

## 5. Security and operational considerations

**Audit integrity without triggers.** On one node a single storage thread runs every storage job, one at a time, and every mutation is a storage function that appends its audit row in the same transaction as its change. There is no other path to the console tables in the daemon: route handlers hold no database handle, and the protocol carries no SQL. A mutation without its audit row would need a function that writes a governed table without calling `append_audit`, which the one-function rule and its tests exclude. In a cluster every node follows the same rule against PostgreSQL, and the transaction makes each change and its audit row commit or roll back together. The triggers of Zig gave no stronger guarantee: they protected against the console's own code, and the console's own code is now the function under test.
**What an operator writing SQL directly can break.** An operator with write access to the database can delete or rewrite audit rows, insert rows that claim actions nobody took, change a setting or a rule without an audit row, and delete console rows. The audit is append-only to the console, not tamper-proof, as SID 0007 states; a deployment that needs more can grant the operators' PostgreSQL role `INSERT` and `SELECT` only on `console_audit` and keep a separate role for the daemon. Direct writes to `policies`, `ip_reputation` and `policy_inspection` remain supported and reach every node through the core's triggers, without console history or audit, as in Zig.
**What such an operator cannot break.** The constraints refuse every row outside the Rust types: an unknown role, action, state or enumerated name; a digest, boot or payload of the wrong length; a setting beyond its catalogue bound; a session longer than 12 hours or idling past its expiry; a kiosk grant with other lifetimes; a sending delivery without a claim; the removal of a destination with unfinished deliveries; an archive without its node, boot and minute or with a wrong charge; a GeoIP activation of a missing generation or the deletion of the active one; a reference to a missing account, node, event, incident or reputation prefix. A direct deletion of an incident or reputation row takes its console sidecars or ownership with it.
**Secrets.** TOTP seeds and webhook secrets are stored only as sealed envelopes; sessions, CSRF tokens, kiosk codes, tokens and recovery codes only as SHA-256 digests. A read of the database therefore yields no usable credential, as in SID 0007.
**Failure.** If the database is unreachable, storage jobs fail with the driver's error, the console answers `Unavailable`, and the data plane keeps serving from its last engine (SID 0011). A job abandoned by a timeout of its caller still runs to completion or rollback on the storage thread; it never commits half an invariant.

## 6. Backwards compatibility

A Zig console database is not migrated, consistent with SID 0011: the Rust daemon does not read a Zaxonlite data directory. Before moving, operators export policies with `sibuna console policies export` and, if they need them, the audit log through `audit/export`. On the Rust daemon they create the administrator with `sibuna console init-admin`, recreate users, tokens, destinations, settings and page templates, and import the GeoIP dataset again. Minute history, rankings, rule hits, challenge records and incident sidecars start empty. Routes, wire formats and console behaviour are unchanged except in three points. A kiosk exchange no longer moves the granting account's last sign-in time (see the catalogue). A policy import is one request, `policies/import`, instead of chunks and a commit. The protocol and the interface drop the node fields `applied_slot` and `decided_slot`, because the Rust port has no consensus log.

## 7. Acceptance criteria and implementation status

1. The console migration, component `console` version 1, creates exactly the 33 tables and the indexes of this record on SQLite and on PostgreSQL, and a node built without the console starts against the same database.
2. `sqlite_master` lists no trigger except the core's nine revision triggers, and `pg_trigger` lists no user trigger except the core's three.
3. Every test of the invariant catalogue passes on SQLite, and on PostgreSQL when `SIBUNA_TEST_DATABASE_URL` names a test database.
4. For every `CHECK`, `NOT NULL` and foreign key of the console schema, a test shows that a direct write breaking it is refused on both backends.
5. A search conformance test indexes a corpus of recorded attack paths and payloads (including percent-encoding, non-ASCII letters, long tokens and repeated tokens) and shows, for a corpus of queries covering every construct of the grammar, identical sets of matching incidents on both backends; it also compares `fts5vocab` and `tsvector` contents with `search::tokens`.
6. On PostgreSQL, the end-to-end test of each route group passes under a database role granted only the reads and writes of its row in "Reads and writes per route group" (plus the credential tables and the audit tables).
7. Retention tests show each kind's cutoff, batch size and per-pass bound, and that `console_rank_usage` equals the sum of the stored charges after every operation.
8. Two nodes against one PostgreSQL database race every capacity-checked insert and every policy workflow without exceeding a capacity, losing an update or deadlocking, and their audit identifiers follow commit order.
9. `cargo build`, `cargo test`, `cargo clippy --all-targets` and the limit checker pass.

Status: accepted. `sibuna-console-storage` is implemented and released in 1.0.0-beta.1, but acceptance criterion 6 is still open: no test yet runs each route group under a PostgreSQL role granted only that group's reads and writes. The record moves to committed when that test passes.

## 8. Open questions

None remain. The review of 2026-10-15 decided the draft's questions as follows.

- Policy and inspection history follow the audit retention setting (1 to 365 days), except that a rule's newest history row is never deleted, so revert and comparison always have a baseline.
- The PostgreSQL work bound of console reads is `SET LOCAL statement_timeout = 2000` (milliseconds) for aggregate reads; SQLite uses `--console-query-steps` through sqlx's safe progress handler. No new option is added.
- `applied_slot` and `decided_slot` are dropped from the console protocol and the interface; a node row carries its applied policy revision instead.
- A rankings archive is one bounded `BLOB` of at most 48,536 bytes; the chunk table is gone and the charged-quota formula of SID 0007 is unchanged.
- No incident hold is added; SID 0011's example of an "incident under investigation" invariant has no counterpart.

## 9. Review history

- 2026-10-15: drafted before the implementation, from SID 0007, SID 0011 and the Zig schema at version 40 (44 tables, 49 triggers).
- 2026-10-15 (review): accepted with these changes. (1) The console relies on SID 0011's bounded job queue (a busy refusal at 32 queued jobs, answered with the console's 503 "console busy" response) and on its fairness (at most 16 jobs between ticks, a due tick first); the record no longer claims that no mailbox capacity is needed, and the storage-loop change is SID 0011's, not a console change. (2) A rankings archive is one row with one `BLOB` of at most 48,536 bytes: `console_rank_chunks`, the reservation state and the bound on pending reservations are dropped, `rankings::store_archive` checks the digest and the format, takes the quota and inserts in one transaction (S33 is merged into S32), S34 deletes archives and subtracts their charge, and the charge formula of SID 0007 is unchanged. (3) A

policy import is one bounded request of at most 128 documents within the policy crate's 256 KiB file bound; `console_policy_import_chunks` is dropped, while country actions keep their staged review. The schema has 33 tables. (4) Enabling TOTP writes `totp.enable` and `user.revoke`; a kiosk exchange writes `kiosk.exchange` and `session.create` but does not move the granting account's last sign-in time. (5) The open questions are decided: history follows the audit retention except each rule's newest row, the PostgreSQL read bound is a 2,000 ms `statement_timeout` and SQLite's is the progress handler, `applied_slot` and `decided_slot` are dropped from the protocol and the interface, and no incident hold is added.

- 2026-10-15 (Rust port, consolidation): The 4,096-byte bound of a policy history document is enforced in Rust on the compact text before the insert, because SQLite measures that text while PostgreSQL measures the JSONB rendering, which adds a space after every colon and comma; the PostgreSQL `CHECK` on `console_policy_history.document` is therefore 8,192 bytes of the rendering (SQLite keeps 4,096), so a document SQLite accepts is never refused only on PostgreSQL. A country preview's `nodes_before` and `nodes_after` count the whole reputation trie of the composed engine (`Engine::reputation_node_count` of `sibuna-policy`), the policy file's entries, the root and the IPv4 path included, instead of a trie rebuilt from the stored rows alone. At most 16 distinct country sets are staged in `console_country_chunks` within the 600-second staging window; a chunk that would start a seventeenth set is refused with `Capacity` under its own capacity lock, and chunks of a staged set still pass. The audit view shows the `value` member of `setting.change` summaries (a catalogue number, never a secret), so the Audit page shows a setting before and after its change. The console protocol now matches the accepted review: `storage::Request::Import` (`workflows::Import` with the expected revision and 1 to 128 non-empty documents of at most 4 KiB, at most 256 KiB in all) replaces `ImportChunk` and `ImportCommit`; the node `Member` row no longer has `applied_slot` and `decided_slot`, and `events::query_events` takes the search text from the `search` member of `events::Query` (at most 256 bytes), which the protocol gained, instead of a separate argument.
- 2026-10-15 (Rust port, console server mapping): refusals map Unauthorized to 401, Forbidden to 403, InvalidInput to 400, Conflict to 409 and others to 503 with the route's code (`CONSOLEPOLICY`, `CONSOLEEVENTS`, `CONSOLESECURITY`, `CONSOLENODE`, `CONSOLESETTINGS` or `CONSOLEPAGE`, `CONSOLECHALLENGE`, `RANKHISTORY`); Capacity is 422 `CONSOLECAPACITY` on workflows and 503 `CONSOLENOTIFYFULL` on destinations; a database failure is 503 `CONSOLEQUORUM`; minute and rule-hit reads storage cannot serve answer `MINUTES001` and `RULEHITS001`. Country actions stage their chunks and preview or apply in one transaction. The events form accepts `search`.
- 2026-10-15 (Rust port, feed reads): `feed::read_page` serves the `events` and `audit` topics. It reads pages of at most eight rows below a watermark captured per issuer, using the receipts `incident_receipts` and `console_audit_cursor` instead of the `sibuna_meta` keys. It counts ids without a row as `missing_ids` and refuses a cursor beyond the head as a conflict. `feed::committed_revision` serves the `policy` topic. Neither takes a credential, because each stream rechecks its own session. Invariant tests are in `tests/invariants/feed.rs`.
- 2026-10-15 (Rust port, peers): the `console_audit.action` CHECK gains `peer.rejected` (39 actions), so SID 0012's audit of refused peers has a row to write. Development databases created before this change refuse the row until they are recreated.
- 2026-10-15 (Rust port, incident country): an incident keeps the country it was first mapped with. When an incident batch rolls back and the storage thread retries it, the sidecar hook reuses the mappings of the failed attempt, so a GeoIP generation activated in between does not change them; Zig fixed them when it prepared the batch. Only the last attempt's mappings are kept.

## 10. References

- SID 0007: the console, its behaviour, bounds and data model.
- SID 0010: the Rust port and its crates.
- SID 0011: storage through sqlx, its typing rules, the storage thread and `Storage::with_database`.
- The Zig console schema, `libs/console/src/schema*.zig`, and its stores, `apps/sibuna/src/console_store*.zig` and related files, in the Zig repository.
- SQLite: `STRICT` tables, `WITHOUT ROWID` tables, foreign keys, FTS5 and its `ascii` tokenizer, `sqlite3_progress_handler`.

- PostgreSQL: text search types (`tsvector` and `tsquery` input syntax, phrase search and weights), explicit locking and advisory locks, identity columns, `statement_timeout`.

### *Lifecycle event*

```
2026-10-15: prediscussion -> discussion. Specification of the console
storage, written before the implementation
```

### *Lifecycle event*

```
2026-10-15: discussion -> accepted. Reviewed and accepted with the changes in
its review history
```