Brand Protection Monitor (PoC)
PostgreSQL Source of Truth — Full Database Design

Generated on 2026-02-01 18:58:36 (local build time). This PDF is a complete, implementation-ready database specification derived from the PRD and User Stories. It follows strict flow-based PDF layout rules: fixed 36pt margins, no absolute positioning, automatic page breaks, and long tables that wrap and repeat headers across pages.

Scope and Inputs

This document converts product requirements into a production-grade PostgreSQL schema. It is intended to be the database “Source of Truth” for a PoC that monitors a single public Certificate Transparency (CT) log, executes periodic batch cycles, extracts X.509 CN/SAN DNS names, performs case-insensitive substring matching against configured brand keywords, persists only matches, and supports a UI for keyword management, triage, and CSV export.

External CT log (PoC): Let’s Encrypt Oak CT log (2026h2) as specified in the PRD.

Core UI/API capabilities: Keyword CRUD; monitor status and run metrics; paginated matches with filters; CSV export with filters.

Step 1 — Context Gathering and Stress-Testing (Inferred Answers)

The DatabaseGPT process begins with clarifying questions. For this project, the answers are inferred from the PRD/User Stories and recorded here as explicit assumptions so engineering can implement without ambiguity.

Question	Inferred Answer
Naming convention (tables/columns)?	English, snake_case. (Requested: 100% English.)
Top 3 user actions that must work flawlessly?	1) Create/delete/list keywords with validation and dedupe. 2) View monitor status + latest run metrics. 3) Search/filter/export matches reliably.
Scale expectation?	PoC scale: small-to-medium (thousands to tens of thousands of matches). Design remains scalable to millions with indexing, pg_trgm, and optional partitioning.

Stress-test Questions (Resolved Assumptions)

• **Deletion chains**: Keywords are soft-deleted. Historical matches are never deleted automatically. Runs are append-only. Users may be soft-deleted without erasing audit trails.

• **Ownership transfer**: A single-tenant PoC is assumed. If a user leaves, created_by/updated_by become NULL via ON DELETE SET NULL, preserving evidence.

• **Audit depth**: Baseline immutable audit columns on all tables. For legal-grade field-level change history, an optional audit_events table is included.

• **Concurrency**: Optimistic locking via version_number on mutable tables (users, keywords, monitor_state, monitor_config).

• **Temporal logic**: Point-in-time is required for runs (monitor_run append-only) and for match visibility (first_seen_at/last_seen_at). Current state is stored in monitor_state.

Step 2 — Inferred Entity and Relationship Analysis

Entities are inferred directly from the PRD and User Stories. The schema targets 3NF where it increases integrity, but intentionally uses denormalization and JSONB where it reduces operational complexity (notably for export tracking and filter snapshots).

Primary Entities

Entity	Purpose
users	Authentication and system actors. Required by the agent standard even if the PoC UI runs without login.
keywords	Configurable brand keywords to match against CN/SAN (case-insensitive substring). Soft delete supported.
monitor_config	Singleton configuration (poll interval, batch size, CT log base URL). Stored for runtime reloadability.
monitor_state	Singleton status for dashboard: idle/running/error, last success/error timestamps, last error code/message.
monitor_run	Append-only run history: CT tree size, processed count, match count, parse errors, timing metrics, error fields.
matched_certificates	Deduplicated matches: one row per (cert_fingerprint_sha256, keyword, matched_domain). Tracks first_seen/last_seen and last_run_id.
exports	Operational trace for CSV exports: filter snapshot JSONB, counts, timing. Supports KPI export-usage.
audit_events (optional)	Field-level audit log for compliance-grade traceability (only if required).

Cardinalities and FK Behaviors

Relationship	Behavior
users (1) — (N) keywords	created_by/updated_by/deleted_by are nullable; ON DELETE SET NULL
monitor_run (1) — (N) matched_certificates	matched_certificates.last_run_id references monitor_run; ON DELETE SET NULL
monitor_state (singleton)	No FK; single-row table with id=1
monitor_config (singleton)	No FK; single-row table with id=1
exports (N) — (1) users	exports.created_by references users; ON DELETE SET NULL

Step 3 — PostgreSQL Schema Generation (Production-Ready DDL)

This section provides complete DDL: extensions, enums, tables, constraints, indexes, and triggers. All tables include the Immutable Audit Standard fields and optimistic locking (version_number). No plaintext secrets are stored; passwords must be bcrypt hashes (cost ≥ 10).

Extensions

```
-- Core extensions
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "pgcrypto";

-- For fast substring search (ILIKE/contains) on domain/issuer and general 'q' search
CREATE EXTENSION IF NOT EXISTS "pg_trgm";
```

Shared Types (Enums)

```
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'keyword_status') THEN
        CREATE TYPE keyword_status AS ENUM ('active', 'inactive');
    END IF;

    IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'monitor_state_enum') THEN
        CREATE TYPE monitor_state_enum AS ENUM ('idle', 'running', 'error');
    END IF;

    IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'monitor_run_state_enum') THEN
        CREATE TYPE monitor_run_state_enum AS ENUM ('running', 'success', 'error');
    END IF;

    IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'matched_field_enum') THEN
        CREATE TYPE matched_field_enum AS ENUM ('cn', 'san', 'both');
    END IF;
END$$;
```

Trigger Functions

```
-- Updated-at trigger (reused by all tables)
CREATE OR REPLACE FUNCTION trigger_set_updated_at()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Optimistic locking helper: bump version_number on update
CREATE OR REPLACE FUNCTION trigger_bump_version_number()
RETURNS TRIGGER AS $$
BEGIN
    NEW.version_number = OLD.version_number + 1;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Users (Authentication)

Even if the PoC UI runs without login, the database schema includes a secure users table to satisfy the authentication requirement and enable future hardening (e.g., admin actions, audit attribution).

```
CREATE TABLE IF NOT EXISTS users (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL, -- bcrypt hash, cost >= 10 (enforced by app)
    full_name VARCHAR(255) NOT NULL,
    is_active BOOLEAN DEFAULT TRUE NOT NULL,

    created_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    updated_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    last_login_at TIMESTAMPTZ,
```

```
    deleted_at TIMESTAMPTZ,
    deleted_by UUID REFERENCES users(id) ON DELETE SET NULL,
    is_deleted BOOLEAN DEFAULT FALSE NOT NULL,
    version_number INTEGER DEFAULT 1 NOT NULL,

    CONSTRAINT email_format CHECK (
        email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
    )
);

CREATE INDEX IF NOT EXISTS idx_users_email ON users(email) WHERE is_deleted = FALSE;
CREATE INDEX IF NOT EXISTS idx_users_active ON users(is_active) WHERE is_active = TRUE AND is_deleted = FALS
CREATE INDEX IF NOT EXISTS idx_users_deleted ON users(is_deleted);

DROP TRIGGER IF EXISTS set_updated_at_users ON users;
CREATE TRIGGER set_updated_at_users
    BEFORE UPDATE ON users
    FOR EACH ROW
    EXECUTE FUNCTION trigger_set_updated_at();

DROP TRIGGER IF EXISTS bump_version_users ON users;
CREATE TRIGGER bump_version_users
    BEFORE UPDATE ON users
    FOR EACH ROW
    EXECUTE FUNCTION trigger_bump_version_number();
```

keywords

```
CREATE TABLE IF NOT EXISTS keywords (
    keyword_id BIGSERIAL PRIMARY KEY,
    value TEXT NOT NULL,
    normalized_value TEXT NOT NULL,
    status keyword_status NOT NULL DEFAULT 'active',

    created_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    updated_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    deleted_at TIMESTAMPTZ,
    created_by UUID REFERENCES users(id) ON DELETE SET NULL,
    updated_by UUID REFERENCES users(id) ON DELETE SET NULL,
    deleted_by UUID REFERENCES users(id) ON DELETE SET NULL,
    is_deleted BOOLEAN DEFAULT FALSE NOT NULL,
    version_number INTEGER DEFAULT 1 NOT NULL,

    CONSTRAINT kw_value_length CHECK (char_length(value) BETWEEN 1 AND 64),
    CONSTRAINT kw_normalized_length CHECK (char_length(normalized_value) BETWEEN 1 AND 64),
    CONSTRAINT kw_status_deleted_consistency CHECK (
        (is_deleted = FALSE AND deleted_at IS NULL) OR
        (is_deleted = TRUE  AND deleted_at IS NOT NULL)
    )
);

CREATE UNIQUE INDEX IF NOT EXISTS ux_keywords_normalized_active
    ON keywords (normalized_value)
    WHERE is_deleted = FALSE;

CREATE INDEX IF NOT EXISTS idx_keywords_status
    ON keywords(status)
    WHERE is_deleted = FALSE;

DROP TRIGGER IF EXISTS set_updated_at_keywords ON keywords;
CREATE TRIGGER set_updated_at_keywords
    BEFORE UPDATE ON keywords
    FOR EACH ROW
    EXECUTE FUNCTION trigger_set_updated_at();

DROP TRIGGER IF EXISTS bump_version_keywords ON keywords;
CREATE TRIGGER bump_version_keywords
    BEFORE UPDATE ON keywords
    FOR EACH ROW
    EXECUTE FUNCTION trigger_bump_version_number();
```

monitor_config (singleton)

```
CREATE TABLE IF NOT EXISTS monitor_config (
    id SMALLINT PRIMARY KEY DEFAULT 1,

    ct_log_base_url TEXT NOT NULL,
    poll_interval_seconds INT NOT NULL DEFAULT 60,
    batch_size INT NOT NULL DEFAULT 100,
    ct_connect_timeout_ms INT NOT NULL DEFAULT 2000,
    ct_read_timeout_ms INT NOT NULL DEFAULT 5000,

    created_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    updated_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    created_by UUID REFERENCES users(id) ON DELETE SET NULL,
    updated_by UUID REFERENCES users(id) ON DELETE SET NULL,
    deleted_at TIMESTAMPTZ,
    deleted_by UUID REFERENCES users(id) ON DELETE SET NULL,
    is_deleted BOOLEAN DEFAULT FALSE NOT NULL,
    version_number INTEGER DEFAULT 1 NOT NULL,

    CONSTRAINT monitor_config_singleton CHECK (id = 1),
    CONSTRAINT poll_interval_positive CHECK (poll_interval_seconds BETWEEN 5 AND 86400),
    CONSTRAINT batch_size_positive CHECK (batch_size BETWEEN 1 AND 10000),
    CONSTRAINT timeout_positive CHECK (ct_connect_timeout_ms > 0 AND ct_read_timeout_ms > 0),
    CONSTRAINT mc_deleted_consistency CHECK (
        (is_deleted = FALSE AND deleted_at IS NULL) OR
        (is_deleted = TRUE  AND deleted_at IS NOT NULL)
    )
);

DROP TRIGGER IF EXISTS set_updated_at_monitor_config ON monitor_config;
CREATE TRIGGER set_updated_at_monitor_config
    BEFORE UPDATE ON monitor_config
    FOR EACH ROW
    EXECUTE FUNCTION trigger_set_updated_at();

DROP TRIGGER IF EXISTS bump_version_monitor_config ON monitor_config;
CREATE TRIGGER bump_version_monitor_config
    BEFORE UPDATE ON monitor_config
    FOR EACH ROW
    EXECUTE FUNCTION trigger_bump_version_number();
```

monitor_state (singleton)

```
CREATE TABLE IF NOT EXISTS monitor_state (
    id SMALLINT PRIMARY KEY DEFAULT 1,
    state monitor_state_enum NOT NULL DEFAULT 'idle',

    last_run_at TIMESTAMPTZ,
    last_success_at TIMESTAMPTZ,
    last_error_at TIMESTAMPTZ,
    last_error_code TEXT,
    last_error_message TEXT,

    created_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    updated_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    created_by UUID REFERENCES users(id) ON DELETE SET NULL,
    updated_by UUID REFERENCES users(id) ON DELETE SET NULL,
    deleted_at TIMESTAMPTZ,
    deleted_by UUID REFERENCES users(id) ON DELETE SET NULL,
    is_deleted BOOLEAN DEFAULT FALSE NOT NULL,
    version_number INTEGER DEFAULT 1 NOT NULL,

    CONSTRAINT monitor_state_singleton CHECK (id = 1),
    CONSTRAINT ms_deleted_consistency CHECK (
        (is_deleted = FALSE AND deleted_at IS NULL) OR
        (is_deleted = TRUE  AND deleted_at IS NOT NULL)
    )
);

DROP TRIGGER IF EXISTS set_updated_at_monitor_state ON monitor_state;
CREATE TRIGGER set_updated_at_monitor_state
    BEFORE UPDATE ON monitor_state
    FOR EACH ROW
    EXECUTE FUNCTION trigger_set_updated_at();
```

```
DROP TRIGGER IF EXISTS bump_version_monitor_state ON monitor_state;
CREATE TRIGGER bump_version_monitor_state
    BEFORE UPDATE ON monitor_state
    FOR EACH ROW
    EXECUTE FUNCTION trigger_bump_version_number();
```

monitor_run (append-only)

```
CREATE TABLE IF NOT EXISTS monitor_run (
    run_id BIGSERIAL PRIMARY KEY,

    started_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    finished_at TIMESTAMPTZ,

    ct_tree_size BIGINT,
    range_start BIGINT,
    range_end BIGINT,

    processed_count INT NOT NULL DEFAULT 0,
    match_count INT NOT NULL DEFAULT 0,
    parse_error_count INT NOT NULL DEFAULT 0,

    duration_ms INT,
    ct_latency_ms INT,
    db_latency_ms INT,

    state monitor_run_state_enum NOT NULL DEFAULT 'running',
    error_code TEXT,
    error_message TEXT,

    created_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    updated_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    created_by UUID REFERENCES users(id) ON DELETE SET NULL,
    updated_by UUID REFERENCES users(id) ON DELETE SET NULL,
    deleted_at TIMESTAMPTZ,
    deleted_by UUID REFERENCES users(id) ON DELETE SET NULL,
    is_deleted BOOLEAN DEFAULT FALSE NOT NULL,
    version_number INTEGER DEFAULT 1 NOT NULL,

    CONSTRAINT counts_non_negative CHECK (
        processed_count >= 0 AND match_count >= 0 AND parse_error_count >= 0
    ),
    CONSTRAINT timing_non_negative CHECK (
        (duration_ms IS NULL OR duration_ms >= 0) AND
        (ct_latency_ms IS NULL OR ct_latency_ms >= 0) AND
        (db_latency_ms IS NULL OR db_latency_ms >= 0)
    ),
    CONSTRAINT finished_after_started CHECK (
        finished_at IS NULL OR finished_at >= started_at
    ),
    CONSTRAINT mr_deleted_consistency CHECK (
        (is_deleted = FALSE AND deleted_at IS NULL) OR
        (is_deleted = TRUE  AND deleted_at IS NOT NULL)
    )
);

CREATE INDEX IF NOT EXISTS idx_monitor_run_started_at_desc
    ON monitor_run (started_at DESC)
    WHERE is_deleted = FALSE;

CREATE INDEX IF NOT EXISTS idx_monitor_run_state
    ON monitor_run (state)
    WHERE is_deleted = FALSE;

DROP TRIGGER IF EXISTS set_updated_at_monitor_run ON monitor_run;
CREATE TRIGGER set_updated_at_monitor_run
    BEFORE UPDATE ON monitor_run
    FOR EACH ROW
    EXECUTE FUNCTION trigger_set_updated_at();

DROP TRIGGER IF EXISTS bump_version_monitor_run ON monitor_run;
CREATE TRIGGER bump_version_monitor_run
    BEFORE UPDATE ON monitor_run
```

```
    FOR EACH ROW
    EXECUTE FUNCTION trigger_bump_version_number();
```

matched_certificates (deduplicated matches)

```
CREATE TABLE IF NOT EXISTS matched_certificates (
    match_id BIGSERIAL PRIMARY KEY,

    cert_fingerprint_sha256 TEXT NOT NULL,
    matched_domain TEXT NOT NULL,
    matched_keyword TEXT NOT NULL,
    matched_field matched_field_enum NOT NULL,

    issuer TEXT,
    not_before TIMESTAMPTZ,
    not_after TIMESTAMPTZ,

    source_ct_log TEXT NOT NULL,

    first_seen_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    last_seen_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

    last_run_id BIGINT REFERENCES monitor_run(run_id) ON DELETE SET NULL,

    created_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    updated_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    created_by UUID REFERENCES users(id) ON DELETE SET NULL,
    updated_by UUID REFERENCES users(id) ON DELETE SET NULL,
    deleted_at TIMESTAMPTZ,
    deleted_by UUID REFERENCES users(id) ON DELETE SET NULL,
    is_deleted BOOLEAN DEFAULT FALSE NOT NULL,
    version_number INTEGER DEFAULT 1 NOT NULL,

    CONSTRAINT fingerprint_non_empty CHECK (char_length(cert_fingerprint_sha256) BETWEEN 16 AND 128),
    CONSTRAINT domain_non_empty CHECK (char_length(matched_domain) BETWEEN 1 AND 253),
    CONSTRAINT keyword_non_empty CHECK (char_length(matched_keyword) BETWEEN 1 AND 64),
    CONSTRAINT validity_window CHECK (
        not_before IS NULL OR not_after IS NULL OR not_after >= not_before
    ),
    CONSTRAINT last_seen_after_first CHECK (
        last_seen_at >= first_seen_at
    ),
    CONSTRAINT mc_deleted_consistency CHECK (
        (is_deleted = FALSE AND deleted_at IS NULL) OR
        (is_deleted = TRUE  AND deleted_at IS NOT NULL)
    )
);

CREATE UNIQUE INDEX IF NOT EXISTS ux_match_key_active
    ON matched_certificates (cert_fingerprint_sha256, matched_keyword, matched_domain)
    WHERE is_deleted = FALSE;

CREATE INDEX IF NOT EXISTS idx_matches_first_seen_desc
    ON matched_certificates (first_seen_at DESC)
    WHERE is_deleted = FALSE;

CREATE INDEX IF NOT EXISTS idx_matches_last_seen_desc
    ON matched_certificates (last_seen_at DESC)
    WHERE is_deleted = FALSE;

CREATE INDEX IF NOT EXISTS idx_matches_keyword
    ON matched_certificates (matched_keyword)
    WHERE is_deleted = FALSE;

CREATE INDEX IF NOT EXISTS gin_matches_domain_trgm
    ON matched_certificates USING GIN (matched_domain gin_trgm_ops)
    WHERE is_deleted = FALSE;

CREATE INDEX IF NOT EXISTS gin_matches_issuer_trgm
    ON matched_certificates USING GIN (issuer gin_trgm_ops)
    WHERE is_deleted = FALSE AND issuer IS NOT NULL;

DROP TRIGGER IF EXISTS set_updated_at_matched_certificates ON matched_certificates;
```

```
CREATE TRIGGER set_updated_at_matched_certificates
    BEFORE UPDATE ON matched_certificates
    FOR EACH ROW
    EXECUTE FUNCTION trigger_set_updated_at();

DROP TRIGGER IF EXISTS bump_version_matched_certificates ON matched_certificates;
CREATE TRIGGER bump_version_matched_certificates
    BEFORE UPDATE ON matched_certificates
    FOR EACH ROW
    EXECUTE FUNCTION trigger_bump_version_number();
```

exports (CSV export tracking)

```
CREATE TABLE IF NOT EXISTS exports (
    export_id BIGSERIAL PRIMARY KEY,

    requested_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    completed_at TIMESTAMPTZ,

    filters JSONB NOT NULL DEFAULT '{}'::jsonb,
    rows_exported INT,
    duration_ms INT,

    created_by UUID REFERENCES users(id) ON DELETE SET NULL,

    created_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    updated_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    updated_by UUID REFERENCES users(id) ON DELETE SET NULL,
    deleted_at TIMESTAMPTZ,
    deleted_by UUID REFERENCES users(id) ON DELETE SET NULL,
    is_deleted BOOLEAN DEFAULT FALSE NOT NULL,
    version_number INTEGER DEFAULT 1 NOT NULL,

    CONSTRAINT export_finished_after_started CHECK (
        completed_at IS NULL OR completed_at >= requested_at
    ),
    CONSTRAINT rows_non_negative CHECK (rows_exported IS NULL OR rows_exported >= 0),
    CONSTRAINT export_timing_non_negative CHECK (duration_ms IS NULL OR duration_ms >= 0),
    CONSTRAINT exports_deleted_consistency CHECK (
        (is_deleted = FALSE AND deleted_at IS NULL) OR
        (is_deleted = TRUE  AND deleted_at IS NOT NULL)
    )
);

CREATE INDEX IF NOT EXISTS gin_exports_filters
    ON exports USING GIN (filters jsonb_path_ops)
    WHERE is_deleted = FALSE;

CREATE INDEX IF NOT EXISTS idx_exports_requested_at_desc
    ON exports (requested_at DESC)
    WHERE is_deleted = FALSE;

DROP TRIGGER IF EXISTS set_updated_at_exports ON exports;
CREATE TRIGGER set_updated_at_exports
    BEFORE UPDATE ON exports
    FOR EACH ROW
    EXECUTE FUNCTION trigger_set_updated_at();

DROP TRIGGER IF EXISTS bump_version_exports ON exports;
CREATE TRIGGER bump_version_exports
    BEFORE UPDATE ON exports
    FOR EACH ROW
    EXECUTE FUNCTION trigger_bump_version_number();
```

Optional: refresh_tokens

```
CREATE TABLE IF NOT EXISTS refresh_tokens (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    token_hash VARCHAR(255) NOT NULL,
    expires_at TIMESTAMPTZ NOT NULL,
    created_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    revoked_at TIMESTAMPTZ,
```

```
    CONSTRAINT valid_expiration CHECK (expires_at > created_at)
);

CREATE INDEX IF NOT EXISTS idx_refresh_tokens_user ON refresh_tokens(user_id);
CREATE INDEX IF NOT EXISTS idx_refresh_tokens_expiry ON refresh_tokens(expires_at) WHERE revoked_at IS NULL
```

Optional: audit_events

```
CREATE TABLE IF NOT EXISTS audit_events (
    event_id BIGSERIAL PRIMARY KEY,
    entity_table TEXT NOT NULL,
    entity_pk TEXT NOT NULL,
    action TEXT NOT NULL,
    changed_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    actor_user_id UUID REFERENCES users(id) ON DELETE SET NULL,
    before JSONB,
    after JSONB,
    created_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    updated_at TIMESTAMPTZ DEFAULT NOW() NOT NULL,
    is_deleted BOOLEAN DEFAULT FALSE NOT NULL,
    version_number INTEGER DEFAULT 1 NOT NULL
);

CREATE INDEX IF NOT EXISTS idx_audit_events_entity
    ON audit_events(entity_table, entity_pk, changed_at DESC)
    WHERE is_deleted = FALSE;

CREATE INDEX IF NOT EXISTS gin_audit_events_after
    ON audit_events USING GIN (after jsonb_path_ops)
    WHERE is_deleted = FALSE;
```

Data Integrity Report

Area	Implementation	Why it matters
Keyword validation	CHECK length 1..64; normalized_value unique where is_deleted=false	Meets HU-KW-01 ACs (no empty, normalized, no duplicates).
Soft delete	is_deleted + deleted_at consistency checks; status=inactive for keywords	Meets HU-KW-02 ACs (keep history; avoid hard deletes).
Dedup matches	Unique index on (fingerprint, keyword, domain) where is_deleted=false; UPSERT updates last_seen_at	Meets HU-MON-03 ACs (no repeated matches between cycles).
Temporal correctness	first_seen_at <= last_seen_at; not_after >= not_before	Prevents inconsistent evidence windows.
Monitor state model	monitor_state singleton + monitor_run history	Meets HU-DASH-01 and observability NFRs.
Ownership/audit	created_by/updated_by/deleted_by -> users with ON DELETE SET NULL	Preserves traceability without blocking user deletion.

Reference UPSERT Pattern for Matches

```
INSERT INTO matched_certificates (
    cert_fingerprint_sha256,
    matched_domain,
    matched_keyword,
    matched_field,
    issuer,
    not_before,
    not_after,
    source_ct_log,
    first_seen_at,
    last_seen_at,
    last_run_id,
    created_by,
    updated_by
)
VALUES (
    $1, $2, $3, $4::matched_field_enum, $5, $6, $7, $8,
    NOW(), NOW(), $9,
    $10, $10
)
ON CONFLICT (cert_fingerprint_sha256, matched_keyword, matched_domain)
WHERE is_deleted = FALSE
DO UPDATE SET
    last_seen_at = NOW(),
    last_run_id = EXCLUDED.last_run_id,
    updated_by = EXCLUDED.updated_by
RETURNING match_id, first_seen_at, last_seen_at;
```

User Story Traceability (DB Support Matrix)

User Story	DB Objects	Key Constraints/Indexes	Notes
HU-KW-01 Create keyword	keywords	kw_value_length; ux_keywords_normalized_active	Trim + lower in API; store both value and normalized_value.
HU-KW-02 Delete keyword	keywords	soft delete + consistency CHECK	Set status=inactive, is_deleted=true, deleted_at=NOW().
HU-KW-03 List/search keywords	keywords	idx_keywords_status; ux_keywords_normalized_active	Search by normalized_value ILIKE '%q%'.
HU-MON-01 Periodic cycle	monitor_config, monitor_run, monitor_state	idx_monitor_run_started_at_desc	Run writes metrics; updates monitor_state.
HU-MON-02 CT range	monitor_run	range_start/range_end persisted	Ensures reproducibility per run.
HU-MON-03 Dedup matches	matched_certificates	ux_match_key_active	UPSERT updates last_seen_at.
HU-MAT-01 List matches	matched_certificates	idx_matches_first_seen_desc	Server-side pagination.
HU-MAT-02 Filter/sort matches	matched_certificates	keyword index + trigram GIN + time indexes	Supports keyword, q substring, date range, new_only.
HU-EXP-01 Export CSV	matched_certificates, exports	ordering indexes; gin_exports_filters	Export respects filters and records export event.
HU-DASH-01 Monitor status	monitor_state, monitor_run	singleton + latest run query	Dashboard reads state and last run metrics.

Step 4 — Performance and Scaling Roadmap

Topic	Recommendation
Indexes	B-Tree for time sorting and exact filters; pg_trgm GIN for substring search; JSONB GIN for export filter snapshots.
Vacuum/Analyze	Autovacuum should handle PoC volumes; for higher write rates tune autovacuum on matched_certificates and monitor_run.
Partitioning	If monitor_run or matched_certificates exceed millions of rows, add RANGE partitions by month on started_at/first_seen_at.
Read replicas	Route dashboards and exports to a read replica to isolate write-path jitter.
Materialized views	Optional daily aggregates by keyword/issuer to accelerate reporting at scale.

Recommended Views (Soft Delete Convenience)

```
CREATE OR REPLACE VIEW v_keywords AS
SELECT * FROM keywords WHERE is_deleted = FALSE;

CREATE OR REPLACE VIEW v_matches AS
SELECT * FROM matched_certificates WHERE is_deleted = FALSE;

CREATE OR REPLACE VIEW v_monitor_runs AS
SELECT * FROM monitor_run WHERE is_deleted = FALSE;
```

Acceptance Checklist (PDF + Schema)

- Flow-based layout only (Platypus flowables), no absolute positioning.
- Fixed 36pt margins on all pages; no content outside printable area.
- LongTable used for multi-page tables with repeatRows=1; wrapped cell content; valign TOP; no truncation.
- Complete DDL included: extensions, enums, tables, constraints, indexes, triggers.
- All tables include immutable audit columns and version_number.
- Secure users table included and ready for future authentication.
- Deduplication implemented with unique indexes + UPSERT pattern.
- Indexes support required filters/sorts for matches and export performance.