Database Architecture

Database design, migration strategy, lifecycle, audit, indexing

Database Design

Database Design

Version: 1.0 **Date:** 2026-02-21 **Status:** Approved **Owner:** Database Architect

---

Design Philosophy

Drop's database schema is designed around three principles:

1. **Simplicity over abstraction.** 19 tables for a well-scoped fintech app. No generic "entities" table, no EAV patterns. Each table maps to a clear domain concept.
2. **Compliance by design.** 7 of 19 tables exist solely for regulatory requirements (GDPR, AML, PSD2). They were added as a compliance infrastructure layer, not retrofitted.
3. **PostgreSQL-native.** Schema is defined in Drizzle ORM (`src/shared/db/schema.ts`) targeting PostgreSQL 16. PostgreSQL-native features (JSONB, `FOR UPDATE`, `RETURNING`, arrays) are available and used where beneficial. See ADR-014.

Why 19 Tables

The table count reflects the actual domain:

- **12 core tables** cover the business logic: users, their bank accounts, recipients, merchants, transactions, exchange rates, cards, sessions, notifications, settings, spending limits, and rate limits.
- **7 compliance tables** were added as a single compliance infrastructure layer: audit logging, AML alerts, STR reports, sanctions screening, consent tracking, data access requests, and complaints.

No table is redundant. No table combines unrelated concerns.

Complete Schema ERD

```
erDiagram
    users ||--|| settings : "1:1 preferences"
```

```
users ||--o{ bank_accounts : "1:N linked accounts"
users ||--o{ cards : "1:N payment cards"
users ||--o{ recipients : "1:N saved recipients"
users ||--o{ transactions : "1:N financial ops"
users ||--o{ sessions : "1:N auth sessions"
users ||--o{ notifications : "1:N alerts"
users ||--o{ spending_limits : "1:N limits"
users ||--o{ merchants : "1:N merchant profiles"
users ||--o{ audit_log : "1:N audit entries"
users ||--o{ aml_alerts : "1:N AML flags"
users ||--o{ str_reports : "1:N STR filings"
users ||--o{ screening_results : "1:N screenings"
users ||--o{ consents : "1:N consents"
users ||--o{ data_access_requests : "1:N DSARs"
users ||--o{ complaints : "1:N complaints"

transactions }o--o| recipients : "remittance target"
transactions }o--o| merchants : "QR payment target"
transactions ||--o{ aml_alerts : "triggers alert"
aml_alerts ||--o{ str_reports : "escalates to STR"
cards ||--o{ spending_limits : "card-level limits"

users {
    text id PK "usr_ + 16 hex"
    text email UK "NOT NULL"
    text password_hash "NOT NULL, default EIDONLY"
    text auth_provider "default bankid"
    text first_name "NOT NULL"
    text last_name "NOT NULL"
    text phone "nullable"
    text date_of_birth "nullable"
    text kyc_status "CHECK pending|approved|rejected"
    text role "CHECK user|merchant"
    text risk_level "CHECK low|medium|high"
    text pep_status "CHECK not_checked|clear|match|pending_review"
    integer sanctions_cleared "default 0"
    text kyc_method "CHECK bankid|document|simplified"
    text kyc_verified_at "nullable"
    text national_id_hash "nullable, indexed WHERE NOT NULL"
    text deleted_at "nullable, soft delete"
```

```
    text created_at "default datetime now"
}

transactions {
    text id PK
    text user_id FK "NOT NULL"
    text type "CHECK remittance|qr_payment"
    text status "CHECK processing|completed|failed"
    integer amount "NOT NULL, in minor units"
    text currency "default NOK"
    integer fee "default 0"
    text recipient_id FK "nullable"
    text merchant_id FK "nullable"
    integer send_amount "nullable"
    text send_currency "nullable"
    integer receive_amount "nullable"
    text receive_currency "nullable"
    real exchange_rate "nullable"
    text purpose_code "nullable"
    text idempotency_key "UNIQUE WHERE NOT NULL"
    text created_at "default datetime now"
    text completed_at "nullable"
}

bank_accounts {
    text id PK
    text user_id FK "NOT NULL"
    text bank_name "NOT NULL"
    text account_number "NOT NULL"
    text iban "nullable"
    integer balance "default 0, cached AISP"
    text balance_synced_at "nullable"
    text currency "default NOK"
    integer is_primary "default 0"
    text connected_at "default datetime now"
}

merchants {
    text id PK
    text user_id FK "NOT NULL"
```

```
        text business_name "NOT NULL"
        text org_number "UNIQUE NOT NULL"
        text address "nullable"
        text bank_account "NOT NULL"
        real fee_rate "default 0.01"
        text status "default active"
        text qr_hmac_key "NOT NULL, random 32 bytes"
        text created_at "default datetime now"
    }

    recipients {
        text id PK
        text user_id FK "NOT NULL"
        text name "NOT NULL"
        text country "NOT NULL"
        text currency "NOT NULL"
        text bank_account "NOT NULL"
        text bank_name "nullable"
        text created_at "default datetime now"
    }
```

Table-by-Table Design Rationale

Core Tables

`users`

Normalization: 3NF. All columns are functionally dependent on the primary key.

Design decisions:

- `id` uses `usr_` prefix + 16 hex chars for readability and collision avoidance across distributed systems.
- `password_hash` defaults to `'EIDONLY'` sentinel value -- BankID-only users have no password. This avoids nullable password fields that complicate auth logic.
- `auth_provider` tracks how the user registered (`bankid`). Supports future Vipps Login without schema changes.
- `national_id_hash` stores SHA-256 of Norwegian fodselsnummer. Enables user deduplication across auth providers without storing the raw national ID.

- `deleted_at` enables soft delete for GDPR erasure while retaining records for AML legal obligations (5-year retention).
- `risk_level`, `pep_status`, `sanctions_cleared` are denormalized onto the user for fast access during transaction authorization -- these are checked on every financial operation.
- KYC fields (`kyc_status`, `kyc_method`, `kyc_verified_at`) are on the user table rather than a separate KYC table because there is a 1:1 relationship and the fields are accessed on every authenticated request.

`transactions`

Normalization: 3NF with intentional denormalization.

Design decisions:

- `amount`, `fee`, `send_amount`, `receive_amount` are stored as integers in minor units (ore for NOK, para for RSD, etc.) to avoid floating-point precision issues.
- Polymorphic reference: `recipient_id` is set for remittances, `merchant_id` for QR payments. Never both. This avoids a separate join table for a simple either/or relationship.
- `exchange_rate` is denormalized (snapshot at transaction time) because rates change. The rate at execution time must be preserved for audit and dispute resolution.
- `idempotency_key` with a unique partial index (`WHERE idempotency_key IS NOT NULL`) prevents duplicate transaction submission without requiring every transaction to have a key.
- `purpose_code` supports remittance regulatory requirements (some corridors require a transfer purpose).
- `completed_at` is separate from `created_at` to track processing duration.

`bank_accounts`

Normalization: 3NF.

Design decisions:

- `balance` is a cached read-only value from AISP, not a Drop-held balance. This is the most important design detail in the entire schema.
- `balance_synced_at` tracks when the balance was last refreshed from the bank via Open Banking.
- `is_primary` flag determines which account is used for transactions by default (1 = primary, 0 = secondary).
- No unique constraint on `account_number` because the same bank account could theoretically appear under different user records (shared accounts).

`recipients`

Normalization: 3NF.

Design decisions:

- Scoped to user (`user_id` FK) -- recipients are private, not shared.
- `country` and `currency` stored as free text validated at the API layer (not as FK to a countries table). This avoids over-engineering for 5-6 supported corridors.
- `bank_account` stores the full foreign account number. Format varies by country (IBAN for EU, local format for others).

`merchants`

Normalization: 3NF.

Design decisions:

- `org_number` is UNIQUE -- one merchant registration per Norwegian organization number (9 digits).
- `qr_hmac_key` is generated server-side (`hex(randomblob(32))`) for QR code integrity verification. Each merchant gets a unique key.
- `fee_rate` defaults to `0.01` (1%). Stored per merchant to allow variable pricing in the future.
- `user_id` FK links the merchant to the user who registered it. A user's role is upgraded to `merchant` upon registration.

`exchange_rates`

Normalization: 3NF.

Design decisions:

- `id` uses `INTEGER PRIMARY KEY AUTOINCREMENT` -- the only auto-increment ID in the schema. Exchange rates are system-managed, not user-created, so prefixed IDs are unnecessary.
- Only stores NOK-to-X rates (6 corridors). Inverse rates are calculated at runtime.
- No historical rate tracking in this table. Transaction records snapshot the rate at execution time.

`sessions`

Normalization: 3NF.

Design decisions:

- `token_hash` stores SHA-256 of the JWT, not the JWT itself. This prevents session hijacking even if the database is compromised.
- `revoked` flag (0/1) enables server-side session invalidation without waiting for JWT expiry.
- Multiple active sessions per user are allowed (different devices).

`settings`

Normalization: 3NF. 1:1 with `users`.

Design decisions:

- `user_id` as PRIMARY KEY enforces the 1:1 relationship at the database level.
- Created lazily on first `GET /api/settings` (INSERT default if not exists).
- Defaults: `currency='NOK'`, `language='nb'`, `push_enabled=1`, `email_enabled=1`.

`notifications`

Normalization: 3NF.

Design decisions:

- `read` flag (0/1) for marking notifications as read in batch.
- No foreign key to the triggering entity (transaction, system event) -- `type` field categorizes the notification source.
- Designed for high volume with eventual cleanup (no retention policy yet).

`cards` (FUTURE)

Normalization: 3NF.

Design decisions:

- Feature-flagged. Table exists in schema but endpoints return 404 when disabled.
- Only stores `last_four` and `token_ref` -- never full card number or CVV (PCI-DSS compliance).
- `pin_hash` added via runtime migration for backward compatibility.
- `status` supports freeze/unfreeze without deletion (`active` -> `frozen` -> `active`).

`spending_limits` (FUTURE)

Normalization: 3NF.

Design decisions:

- `card_id` is nullable -- supports user-level limits (no specific card) or card-level limits.
- `limit_type` values (`daily`, `weekly`, `monthly`, `transaction`) enforced at API level.
- Limits are replaced, not accumulated (PUT semantics per limit type per card).

`rate_limits`

Normalization: 3NF (trivial -- 3 columns).

Design decisions:

- `key` is the IP address (TEXT PK). Simple key-value store.

- `reset_at` is a Unix timestamp. Expired entries are cleaned every 100 rate limit checks in `middleware/rate-limit.ts`.
- Not a "real" domain table -- it is infrastructure. Could be replaced by Redis in production but works fine in SQLite/PostgreSQL.

Compliance Tables

`audit_log`

- `user_id` is nullable because some audit events occur before authentication (e.g., failed login attempts).
- `resource_type` + `resource_id` enable generic resource tracking without polymorphic FKs.
- `details` is a TEXT field (JSON string) for flexible event-specific data.
- `request_id` for correlating multiple audit entries from a single API request.
- Two indexes: `user_id` and `action` for the primary query patterns. (Note: no `timestamp` index exists in the implementation — only `idx_audit_log_user` and `idx_audit_log_action` are created in `db.ts`.)

`aml_alerts`

- `severity` (low/medium/high/critical) determines investigation priority and escalation timelines.
- `status` workflow: `open` -> `investigating` -> `resolved|escalated|filed`.
- `reviewed_by` and `reviewed_at` track the compliance officer's review.
- `transaction_id` FK links to the specific transaction that triggered the alert.

`str_reports`

- Filed with Okokrim/EFE (Norwegian financial intelligence unit).
- `reference_number` stores the authority-assigned reference after submission.
- `alert_id` links back to the originating AML alert.
- Immutable after `status = 'submitted'` -- regulatory requirement.

`screening_results`

- `screening_type` (pep/sanctions/adverse_media) supports multiple screening categories.
- `provider` tracks which screening service was used (future: Sumsub, Refinitiv, etc.).
- `match_details` stores full match information as TEXT (JSON) for review.
- Multiple results per user (periodic rescreening).

`consents`

- `consent_type` values: `terms`, `privacy`, `marketing`, `cookies_analytics`, `cookies_marketing`.
- `granted` (0/1) with separate `granted_at`/`withdrawn_at` timestamps for full consent lifecycle.
- `ip_address` stored as proof of consent action per GDPR requirements.

`data_access_requests`

- `request_type` covers GDPR data subject rights: export (Art. 15), erasure (Art. 17), rectification (Art. 16), restriction (Art. 18).
- `download_url` for data export files (temporary signed URLs).
- `status` workflow: `pending` -> `processing` -> `completed|rejected`.

`complaints`

- Required by Finansavtaleloven section 3-53 (15 business day response requirement).
- `category` (transaction/service/fees/privacy/technical/other) for routing and reporting.
- `resolution` text field filled when complaint is resolved.
- `resolved_at` timestamp for SLA compliance tracking.

Constraint Inventory

Table	Constraint Type	Column(s)	Value
`users`	PRIMARY KEY	`id`	-
`users`	UNIQUE	`email`	-
`users`	NOT NULL	`email`, `password_hash`, `first_name`, `last_name`	-
`users`	CHECK	`kyc_status`	`IN ('pending','approved','rejected')`
`users`	CHECK	`role`	`IN ('user','merchant')`
`users`	CHECK	`risk_level`	`IN ('low','medium','high')`
`users`	CHECK	`pep_status`	`IN ('not_checked','clear','match','pending_review')`
`users`	CHECK	`kyc_method`	`IN ('bankid','document','simplified')`
`transactions`	PRIMARY KEY	`id`	-
`transactions`	NOT NULL	`user_id`, `type`, `amount`	-
`transactions`	FK	`user_id`	`users(id)`
`transactions`	FK	`recipient_id`	`recipients(id)`
`transactions`	FK	`merchant_id`	`merchants(id)`
`transactions`	CHECK	`type`	`IN ('remittance','qr_payment')`

Table	Constraint Type	Column(s)	Value
transactions	CHECK	status	IN ('processing','completed','failed')
transactions	UNIQUE (partial)	idempotency_key	WHERE idempotency_key IS NOT NULL
merchants	PRIMARY KEY	id	-
merchants	UNIQUE	org_number	-
merchants	NOT NULL	user_id, business_name, org_number, bank_account, qr_hmac_key	-
merchants	FK	user_id	users(id)
bank_accounts	PRIMARY KEY	id	-
bank_accounts	NOT NULL	user_id, bank_name, account_number	-
bank_accounts	FK	user_id	users(id)
recipients	PRIMARY KEY	id	-
recipients	NOT NULL	user_id, name, country, currency, bank_account	-
recipients	FK	user_id	users(id)
sessions	PRIMARY KEY	id	-
sessions	NOT NULL	user_id, token_hash, expires_at	-
sessions	FK	user_id	users(id)
cards	PRIMARY KEY	id	-
cards	NOT NULL	user_id, last_four, expiry	-
cards	FK	user_id	users(id)
cards	CHECK	type	IN ('virtual','physical')
cards	CHECK	status	IN ('active','frozen','cancelled')
settings	PRIMARY KEY	user_id	1:1 with users
settings	FK	user_id	users(id)
exchange_rates	PRIMARY KEY	id	AUTOINCREMENT
exchange_rates	NOT NULL	to_currency, rate	-
notifications	PRIMARY KEY	id	-
notifications	NOT NULL	user_id, type, title, body	-

Table	Constraint Type	Column(s)	Value
`notifications`	FK	`user_id`	`users(id)`
`spending_limits`	PRIMARY KEY	`id`	-
`spending_limits`	NOT NULL	`user_id`, `limit_type`, `amount`	-
`spending_limits`	FK	`user_id`, `card_id`	`users(id)`, `cards(id)`
`rate_limits`	PRIMARY KEY	`key`	IP address
`audit_log`	PRIMARY KEY	`id`	-
`audit_log`	NOT NULL	`action`	-
`audit_log`	FK	`user_id`	`users(id)` (nullable)
`aml_alerts`	PRIMARY KEY	`id`	-
`aml_alerts`	NOT NULL	`user_id`, `alert_type`, `severity`	-
`aml_alerts`	FK	`user_id`, `transaction_id`	`users(id)`, `transactions(id)`
`aml_alerts`	CHECK	`severity`	`IN ('low','medium','high','critical')`
`aml_alerts`	CHECK	`status`	`IN ('open','investigating','resolved','escalated','filed')`
`str_reports`	PRIMARY KEY	`id`	-
`str_reports`	NOT NULL	`user_id`, `report_type`	-
`str_reports`	FK	`user_id`, `alert_id`	`users(id)`, `aml_alerts(id)`
`str_reports`	CHECK	`status`	`IN ('draft','submitted','acknowledged')`
`screening_results`	PRIMARY KEY	`id`	-
`screening_results`	NOT NULL	`user_id`, `screening_type`, `result`	-
`screening_results`	FK	`user_id`	`users(id)`
`screening_results`	CHECK	`screening_type`	`IN ('pep','sanctions','adverse_media')`
`screening_results`	CHECK	`result`	`IN ('clear','match','potential_match','error')`
`consents`	PRIMARY KEY	`id`	-
`consents`	NOT NULL	`user_id`, `consent_type`, `granted`	-

Table	Constraint Type	Column(s)	Value
`consents`	FK	`user_id`	`users(id)`
`data_access_requests`	PRIMARY KEY	`id`	-
`data_access_requests`	NOT NULL	`user_id`, `request_type`	-
`data_access_requests`	FK	`user_id`	`users(id)`
`data_access_requests`	CHECK	`request_type`	`IN ('export','erasure','rectification','restriction')`
`data_access_requests`	CHECK	`status`	`IN ('pending','processing','completed','rejected')`
`complaints`	PRIMARY KEY	`id`	-
`complaints`	NOT NULL	`user_id`, `category`, `subject`, `description`	-
`complaints`	FK	`user_id`	`users(id)`
`complaints`	CHECK	`status`	`IN ('received','investigating','resolved','escalated')`

Naming Conventions

Convention	Rule	Examples
Table names	Lowercase, plural, snake_case	`users`, `bank_accounts`, `aml_alerts`
Column names	Lowercase, snake_case	`user_id`, `first_name`, `created_at`
Primary keys	`id` (or entity-name for 1:1 tables like `settings.user_id`)	`users.id`, `settings.user_id`
Foreign keys	`{referenced_table_singular}_id`	`user_id`, `recipient_id`, `card_id`
Timestamps	`{action}_at` suffix	`created_at`, `completed_at`, `withdrawn_at`
Boolean flags	Descriptive name, INTEGER 0/1	`revoked`, `read`, `is_primary`, `granted`
Status columns	`status` with CHECK constraint	`status CHECK(... IN (...))`
Indexes	`idx_{table}_{column}` (exception: `idx_tx_idempotency` uses abbreviation)	`idx_transactions_user`, `idx_sessions_token`, `idx_tx_idempotency`
ID prefixes	3-letter prefix + underscore + 16 hex chars	`usr_`, `tx_`, `ba_`, `mer_`, `rec_`, `ses_`, `con_`, `cmp_`

Normalization Analysis

All tables are in **Third Normal Form (3NF)** with documented exceptions:

Table	NF Level	Deviation	Justification
`users`	3NF	`risk_level`, `pep_status`, `sanctions_cleared` could be in a separate risk_profile table	Accessed on every transaction check. Separate table would add a JOIN to the critical path. 1:1 relationship makes a separate table pointless.
`transactions`	3NF	`exchange_rate`, `send_amount`, `receive_amount` denormalized from exchange_rates	Rate at execution time must be preserved immutably. The `exchange_rates` table changes; the transaction record must not.
`transactions`	3NF	Polymorphic FK (`recipient_id` OR `merchant_id`)	Simple either/or. A join table or STI would add complexity for no benefit at this scale.
`bank_accounts`	3NF	`balance` denormalized from external bank (AISP)	This is a cache, not authoritative data. Drop cannot modify the real balance.
`audit_log`	3NF	`details` is unstructured TEXT (JSON)	Audit events have variable structure. A normalized schema would require dozens of event-specific tables.

No table violates 2NF (no partial key dependencies) because all tables use single-column primary keys.

Cross-References

- **Full schema:** DATABASE-SCHEMA.md
- **Data architecture overview:** data-architecture.md
- **Migration strategy:** migration-strategy.md
- **Dual-driver implementation:** `src/drop-api/src/lib/db.ts`

Migration Strategy

Migration Strategy: SQLite to PostgreSQL

> **STATUS: COMPLETED (2026-03-03)** This document describes the completed migration from the old dual-driver architecture to PostgreSQL-only. The migration is done. Current architecture: PostgreSQL 16 (all environments), Drizzle ORM. See ADR-014 for the authoritative current state.

Version: 1.0 **Date:** 2026-02-21 **Status:** Completed — migration done per ADR-014 **Owner:** Database Architect

---

Overview

> **HISTORICAL NOTE:** The dual-driver architecture and `better-sqlite3` dependency described in this document have been removed. The codebase now uses Drizzle ORM with PostgreSQL 16 exclusively. `db.ts` and `USE_PG` no longer exist. See ADR-014.

This document captures the migration plan that was executed when transitioning from SQLite (development) + PostgreSQL (production) to PostgreSQL 16 in all environments. It is preserved as a historical record.

Migration Execution Flow

```
flowchart TD
    A[Phase 1: Prepare] --> B[Phase 2: Schema Migration]
```

```
B --> C[Phase 3: Data Migration]
C --> D[Phase 4: Validation]
D --> E{All checks pass?}
E -->|Yes| F[Phase 5: Cutover]
E -->|No| G[Fix issues]
G --> D
F --> H[Phase 6: Post-migration]

subgraph "Phase 1: Prepare"
    A1[Provision PostgreSQL instance]
    A2[Configure DATABASE_URL]
    A3[Create shadow database for testing]
    A4[Backup SQLite database file]
end

subgraph "Phase 2: Schema"
    B1[Run PostgreSQL schema DDL]
    B2[Create indexes]
    B3[Verify constraints]
end

subgraph "Phase 3: Data"
    C1[Export SQLite data as INSERT statements]
    C2[Transform data types]
    C3[Load into PostgreSQL]
    C4[Reset sequences]
end

subgraph "Phase 4: Validate"
    D1[Row count comparison]
    D2[Checksum validation]
    D3[Application smoke tests]
    D4[Run full test suite against PG]
end

subgraph "Phase 5: Cutover"
    F1[Set DATABASE_URL in production]
    F2[Deploy application]
    F3[Verify health endpoint]
end
```

```
    subgraph "Phase 6: Post-migration"
        H1[Monitor error rates]
        H2[Monitor query performance]
        H3[Archive SQLite file]
    end
```

Data Type Mapping

The dual-driver layer already handles SQL syntax differences. The schema migration must map SQLite types to PostgreSQL equivalents:

SQLite Type	PostgreSQL Type	Tables Using It	Notes
`TEXT`	`TEXT`	All tables	Direct mapping, no change
`TEXT PRIMARY KEY`	`TEXT PRIMARY KEY`	All except `exchange_rates`	Same behavior
`INTEGER` (boolean)	`BOOLEAN` or `INTEGER`	`sessions.revoked`, `notifications.read`, `settings.push_enabled`, `settings.email_enabled`, `bank_accounts.is_primary`, `consents.granted`, `users.sanctions_cleared`	Keep as INTEGER for dual-driver compat, or convert to BOOLEAN in PG-only mode
`INTEGER` (currency)	`BIGINT`	`transactions.amount`, `transactions.fee`, `transactions.send_amount`, `transactions.receive_amount`, `bank_accounts.balance`, `spending_limits.amount`	Use BIGINT for amounts in minor units to prevent overflow
`INTEGER PRIMARY KEY AUTOINCREMENT`	`SERIAL PRIMARY KEY`	`exchange_rates.id`	Only auto-increment in schema
`INTEGER` (unix timestamp)	`INTEGER`	`rate_limits.reset_at`	Unix epoch, no conversion needed
`REAL`	`DOUBLE PRECISION`	`transactions.exchange_rate`, `merchants.fee_rate`	Direct mapping
`TEXT DEFAULT (datetime('now'))`	`TEXT DEFAULT CURRENT_TIMESTAMP`	All `created_at`, `updated_at` columns	Handled by `adaptSqlForPg()` in db.ts

SQLite-specific SQL Adaptations (already implemented in `db.ts:46-52`)

SQLite SQL	PostgreSQL Equivalent	Handler
`INSERT OR IGNORE INTO`	`INSERT INTO ... ON CONFLICT DO NOTHING`	`runIgnore()` function
`INSERT OR REPLACE INTO`	`INSERT INTO ... ON CONFLICT (col) DO UPDATE SET`	`runUpsert()` function
`datetime('now')`	`CURRENT_TIMESTAMP`	`adaptSqlForPg()` regex
`?` placeholders	`$1, $2, ...` positional	`convertPlaceholders()`
`randomblob(32)`	`gen_random_bytes(32)`	Schema-level change (merchants.qr_hmac_key default)
`hex()`	`encode(..., 'hex')`	Schema-level change

PostgreSQL Schema DDL

The PostgreSQL schema must be created separately from the SQLite schema since `CREATE TABLE IF NOT EXISTS` syntax is shared but defaults and functions differ:

```
-- PostgreSQL schema for Drop
-- Run once when provisioning production database

CREATE TABLE IF NOT EXISTS users (
    id TEXT PRIMARY KEY,
    email TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL DEFAULT 'EIDONLY',
    auth_provider TEXT DEFAULT 'bankid',
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    phone TEXT,
    date_of_birth TEXT,
    kyc_status TEXT DEFAULT 'pending' CHECK(kyc_status IN ('pending','approved','rejected')),
    role TEXT DEFAULT 'user' CHECK(role IN ('user','merchant')),
    risk_level TEXT DEFAULT 'low' CHECK(risk_level IN ('low','medium','high')),
    pep_status TEXT DEFAULT 'not_checked' CHECK(pep_status IN
('not_checked','clear','match','pending_review')),
    sanctions_cleared INTEGER DEFAULT 0,
    kyc_method TEXT CHECK(kyc_method IN ('bankid','document','simplified')),
    kyc_verified_at TEXT,
    national_id_hash TEXT,
    deleted_at TEXT,
```

```
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX IF NOT EXISTS idx_users_national_id ON users(national_id_hash) WHERE
national_id_hash IS NOT NULL;

CREATE TABLE IF NOT EXISTS recipients (
    id TEXT PRIMARY KEY,
    user_id TEXT NOT NULL REFERENCES users(id),
    name TEXT NOT NULL,
    country TEXT NOT NULL,
    currency TEXT NOT NULL,
    bank_account TEXT NOT NULL,
    bank_name TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX IF NOT EXISTS idx_recipients_user ON recipients(user_id);

CREATE TABLE IF NOT EXISTS merchants (
    id TEXT PRIMARY KEY,
    user_id TEXT NOT NULL REFERENCES users(id),
    business_name TEXT NOT NULL,
    org_number TEXT UNIQUE NOT NULL,
    address TEXT,
    bank_account TEXT NOT NULL,
    fee_rate DOUBLE PRECISION DEFAULT 0.01,
    status TEXT DEFAULT 'active',
    qr_hmac_key TEXT NOT NULL DEFAULT encode(gen_random_bytes(32), 'hex'),
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS transactions (
    id TEXT PRIMARY KEY,
    user_id TEXT NOT NULL REFERENCES users(id),
    type TEXT NOT NULL CHECK(type IN ('remittance','qr_payment')),
    status TEXT DEFAULT 'processing' CHECK(status IN ('processing','completed','failed')),
    amount BIGINT NOT NULL,
    currency TEXT DEFAULT 'NOK',
    fee BIGINT DEFAULT 0,
    recipient_id TEXT REFERENCES recipients(id),
    merchant_id TEXT REFERENCES merchants(id),
```

```
    send_amount BIGINT,
    send_currency TEXT,
    receive_amount BIGINT,
    receive_currency TEXT,
    exchange_rate DOUBLE PRECISION,
    purpose_code TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    completed_at TEXT,
    idempotency_key TEXT
);
CREATE UNIQUE INDEX IF NOT EXISTS idx_tx_idempotency ON transactions(idempotency_key) WHERE
idempotency_key IS NOT NULL;
CREATE INDEX IF NOT EXISTS idx_transactions_user ON transactions(user_id);

CREATE TABLE IF NOT EXISTS exchange_rates (
    id SERIAL PRIMARY KEY,
    from_currency TEXT DEFAULT 'NOK',
    to_currency TEXT NOT NULL,
    rate DOUBLE PRECISION NOT NULL,
    updated_at TEXT DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS bank_accounts (
    id TEXT PRIMARY KEY,
    user_id TEXT NOT NULL REFERENCES users(id),
    bank_name TEXT NOT NULL,
    account_number TEXT NOT NULL,
    iban TEXT,
    balance BIGINT DEFAULT 0,
    balance_synced_at TEXT,
    currency TEXT DEFAULT 'NOK',
    is_primary INTEGER DEFAULT 0,
    connected_at TEXT DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX IF NOT EXISTS idx_bank_accounts_user ON bank_accounts(user_id);

-- Remaining tables follow the same pattern...
-- See full DDL in migration script
```

Sequence/Auto-increment Migration

Only one table uses auto-increment: `exchange_rates`.

Aspect	SQLite	PostgreSQL	Migration Step
Type	`INTEGER PRIMARY KEY AUTOINCREMENT`	`SERIAL PRIMARY KEY`	Schema DDL change
Sequence reset	N/A (built into rowid)	`SELECT setval('exchange_rates_id_seq', (SELECT MAX(id) FROM exchange_rates))`	After data load
Gap behavior	Gaps allowed	Gaps allowed	No difference

JSON Handling

Aspect	SQLite	PostgreSQL	Impact
Storage type	TEXT (plain string)	TEXT (could use JSONB)	No change needed for compatibility
JSON columns	`audit_log.details`, `aml_alerts.details`, `str_reports.details`, `screening_results.match_details`	Same columns, stored as TEXT	Keep as TEXT for dual-driver compatibility
Querying JSON	Not used (JSON is stored, not queried in SQL)	Could use `->>` / `@>` operators	Future optimization: add JSONB indexes for audit log queries
Validation	None (application layer)	Could add CHECK with `jsonb` cast	Future enhancement

Decision: Keep JSON columns as TEXT for now. Converting to JSONB is a future optimization that would break dual-driver compatibility.

Date/Time Handling

Aspect	SQLite	PostgreSQL	Migration
Default value	`datetime('now')`	`CURRENT_TIMESTAMP`	Handled by `adaptSqlForPg()`
Storage format	ISO 8601 TEXT	ISO 8601 TEXT (not TIMESTAMP type)	No conversion needed

Aspect	SQLite	PostgreSQL	Migration
Timezone	UTC (application convention)	UTC (application convention)	Consistent
Date arithmetic	`datetime('now', '-3 days')`	`CURRENT_TIMESTAMP - INTERVAL '3 days'`	Only used in seed data, not production queries

Note: All timestamps are stored as TEXT in ISO 8601 format (`YYYY-MM-DDTHH:MM:SS`) in both databases. This is intentional for dual-driver compatibility. A future PostgreSQL-only optimization could convert to `TIMESTAMPTZ`.

Migration Checklist

Pre-Migration

- ☐ Provision PostgreSQL instance (AWS RDS or equivalent)
- ☐ Configure connection pooling (built-in `pg.Pool`, max connections TBD)
- ☐ Set `DATABASE_URL` environment variable
- ☐ Create shadow database for testing
- ☐ Backup current SQLite file: `cp data/drop.db data/drop.db.backup.$(date +%s)`
- ☐ Run full test suite against SQLite (baseline)
- ☐ Review all raw SQL queries for SQLite-specific syntax (should be none -- all go through `db.ts`)

Schema Migration

- ☐ Run PostgreSQL DDL script to create all 19 tables
- ☐ Create all indexes (11 indexes defined in `db.ts` schema)
- ☐ Verify all CHECK constraints are active
- ☐ Verify all foreign key constraints are active
- ☐ Test `initDb()` function with `DATABASE_URL` set

Data Migration

- ☐ Export SQLite data using `sqlite3 drop.db .dump` or custom export script
- ☐ Transform `INTEGER PRIMARY KEY AUTOINCREMENT` to `SERIAL`

- [ ] Transform `randomblob()` defaults to `gen_random_bytes()`
- [ ] Load data into PostgreSQL
- [ ] Reset `exchange_rates_id_seq` sequence
- [ ] Verify row counts match per table

Validation

- [ ] Row count comparison (all 19 tables)
- [ ] Spot-check 10 records per table for data integrity
- [ ] Run application smoke tests:
  - [ ] Login (BankID flow)
  - [ ] View dashboard (bank accounts, balance)
  - [ ] List transactions
  - [ ] Create remittance
  - [ ] Create QR payment
  - [ ] View notifications
  - [ ] Update settings
- [ ] Run full test suite with `DATABASE_URL` set
- [ ] Verify `GET /api/health` returns `db: "connected"` with acceptable latency

Cutover

- [ ] Set `DATABASE_URL` in production environment
- [ ] Deploy application
- [ ] Verify health endpoint
- [ ] Monitor error rates for 1 hour
- [ ] Monitor query latency for 1 hour

Post-Migration

- [ ] Archive SQLite file
- [ ] Update documentation to reflect PostgreSQL as primary
- [ ] Consider PostgreSQL-specific optimizations (JSONB, TIMESTAMPTZ, partial indexes)
- [ ] Configure automated backups (pg_dump cron or RDS snapshots)

Rollback Procedure

```
flowchart TD
    A[Issue detected in PostgreSQL] --> B{Is it data corruption?}
    B -->|Yes| C[Stop application immediately]
    B -->|No| D{Is it a query/performance issue?}
    D -->|Yes| E[Fix query and redeploy]
    D -->|No| F[Investigate further]

    C --> G[Remove DATABASE_URL env var]
    G --> H[Redeploy application]
    H --> I[Application falls back to SQLite]
    I --> J[Investigate PostgreSQL issue offline]
    J --> K[Fix and retry migration]
```

Rollback is simple: Remove or unset the `DATABASE_URL` environment variable. The application immediately falls back to SQLite. This is the primary advantage of the dual-driver architecture.

Rollback Scenario	Action	Downtime
Schema issue in PostgreSQL	Unset `DATABASE_URL`, redeploy	~2 minutes (deploy time)
Data integrity issue	Unset `DATABASE_URL`, redeploy with SQLite backup	~2 minutes
Performance regression	Unset `DATABASE_URL`, optimize PG offline	~2 minutes
Partial migration failure	Drop PostgreSQL schema, fix script, retry	No production impact (still on SQLite)

Key constraint: Rollback only works if no new data has been written to PostgreSQL that does not exist in SQLite. In practice, this means the migration window should be short and the SQLite database should be read-only during cutover.

Zero-Downtime Migration Using Dual-Driver

The dual-driver architecture enables a phased migration with zero downtime:

```
sequenceDiagram
    participant App as Application
    participant SQLite as SQLite (current)
    participant PG as PostgreSQL (new)

    Note over App,SQLite: Phase A: Normal operation (SQLite)
    App->>SQLite: All reads/writes

    Note over App,PG: Phase B: Provision and schema
    App->>SQLite: All reads/writes (unchanged)
    Note right of PG: Create schema, indexes

    Note over App,PG: Phase C: Data migration
    App->>SQLite: All reads/writes (unchanged)
    Note right of PG: Bulk load from SQLite export

    Note over App,PG: Phase D: Final sync + cutover
    App->>SQLite: Brief read-only mode
    Note right of PG: Delta sync (new records since bulk load)
    Note over App: Set DATABASE_URL
    App->>PG: All reads/writes

    Note over App,PG: Phase E: Production on PostgreSQL
    App->>PG: All reads/writes
    Note left of SQLite: Archived as backup
```

Total downtime: Only during Phase D final sync + deploy, estimated at 2-5 minutes for the current data volume.

Testing Approach

Shadow Database Testing

Before production migration, run the full application against a shadow PostgreSQL database:

1. **Provision shadow PG:** Same version and configuration as production target
2. **Run schema creation:** Execute PostgreSQL DDL
3. **Load production-like data:** Export SQLite demo data, transform, load

4. **Run test suite:** `DATABASE_URL=<shadow> npm test`
5. **Run integration tests:** Full API flow tests against shadow
6. **Load testing:** Verify query performance under expected load

Data Integrity Checks

```
-- Row count comparison (run against both databases)
SELECT 'users' as tbl, COUNT(*) as cnt FROM users
UNION ALL SELECT 'transactions', COUNT(*) FROM transactions
UNION ALL SELECT 'bank_accounts', COUNT(*) FROM bank_accounts
UNION ALL SELECT 'recipients', COUNT(*) FROM recipients
UNION ALL SELECT 'merchants', COUNT(*) FROM merchants
UNION ALL SELECT 'sessions', COUNT(*) FROM sessions
UNION ALL SELECT 'notifications', COUNT(*) FROM notifications
UNION ALL SELECT 'settings', COUNT(*) FROM settings
UNION ALL SELECT 'exchange_rates', COUNT(*) FROM exchange_rates
UNION ALL SELECT 'cards', COUNT(*) FROM cards
UNION ALL SELECT 'spending_limits', COUNT(*) FROM spending_limits
UNION ALL SELECT 'rate_limits', COUNT(*) FROM rate_limits
UNION ALL SELECT 'audit_log', COUNT(*) FROM audit_log
UNION ALL SELECT 'aml_alerts', COUNT(*) FROM aml_alerts
UNION ALL SELECT 'str_reports', COUNT(*) FROM str_reports
UNION ALL SELECT 'screening_results', COUNT(*) FROM screening_results
UNION ALL SELECT 'consents', COUNT(*) FROM consents
UNION ALL SELECT 'data_access_requests', COUNT(*) FROM data_access_requests
UNION ALL SELECT 'complaints', COUNT(*) FROM complaints;
```

Cross-References

- **Dual-driver implementation:** `src/drop-api/src/lib/db.ts`
- **Database schema:** DATABASE-SCHEMA.md
- **Database design:** database-design.md
- **Data architecture:** data-architecture.md
- **Deployment architecture:** deployment-architecture.md
- **Roadmap Phase 2:** ROADMAP.md (PostgreSQL migration is Phase 2)

Data Lifecycle

Data Lifecycle Management

Version: 1.0 **Date:** 2026-02-21 **Status:** Approved **Owner:** Database Architect

---

Overview

Drop processes personal and financial data subject to multiple overlapping regulatory frameworks. This document defines retention periods, archival strategies, deletion cascades, and GDPR data subject request handling for all 19 tables.

Applicable regulations:

- **GDPR** (Personopplysningsloven, LOV-2018-06-15-38) -- data minimization, right to erasure, right to access
- **AML/KYC** (Hvitvaskingsloven, LOV-2018-06-01-23) -- 5-year retention post-relationship
- **Norwegian Bookkeeping Act** (Bokforingsloven) -- 5-year retention for financial records
- **PSD2** (Betalingstjenesteloven) -- audit trail requirements
- **Finansavtaleloven** -- complaint handling records

Key tension: GDPR right to erasure (Art. 17) vs. AML legal retention obligations. AML wins -- data required for anti-money laundering must be retained for 5 years regardless of erasure requests.

Retention Periods

Per-Table Retention Schedule

Table	Retention Period	Legal Basis	Archival After	Purge After
`users`	5 years post-relationship end	Hvitvaskingsloven section 30	Account deletion + 1 year	5 years post-deletion
`bank_accounts`	5 years post-relationship end	Hvitvaskingsloven section 30	Account deletion	5 years post-deletion

Table	Retention Period	Legal Basis	Archival After	Purge After
`transactions`	5 years from transaction date	Bokforingsloven section 13, Hvitvaskingsloven section 30	1 year after transaction	5 years after transaction
`recipients`	5 years post-relationship end	Hvitvaskingsloven section 30 (counterparty records)	Account deletion	5 years post-deletion
`merchants`	5 years post-relationship end	Bokforingsloven, Hvitvaskingsloven	Account deletion	5 years post-deletion
`sessions`	90 days after expiry	Legitimate interest (security)	After expiry	90 days after expiry
`notifications`	1 year from creation	Legitimate interest (UX)	6 months	1 year
`settings`	Duration of relationship	Contract performance	Account deletion	Immediate on deletion
`exchange_rates`	Indefinite (reference data)	Legitimate interest	Never	Never
`cards`	5 years post-cancellation	PCI-DSS, Bokforingsloven	Card cancellation	5 years post-cancellation
`spending_limits`	Duration of card lifecycle	Contract performance	Card cancellation	With card record
`rate_limits`	Until window expires	Legitimate interest (security)	Auto-cleaned per request	Immediate on expiry
`audit_log`	5 years from event	PSD2 Art. 94, Hvitvaskingsloven	1 year after event	5 years after event
`aml_alerts`	5 years post-resolution	Hvitvaskingsloven section 30	After resolution	5 years post-resolution
`str_reports`	5 years after filing	Hvitvaskingsloven section 30	Never (active reference)	5 years after filing
`screening_results`	5 years post-relationship end	Hvitvaskingsloven section 30	Account deletion	5 years post-deletion
`consents`	Duration of consent + 5 years	GDPR Art. 7(1) (proof of consent)	After withdrawal + 1 year	5 years after withdrawal
`data_access_requests`	5 years from completion	GDPR accountability (Art. 5(2))	After completion	5 years after completion
`complaints`	5 years from resolution	Finansavtaleloven, Bokforingsloven	After resolution	5 years after resolution

Per-Column Retention (Sensitive Fields)

Table.Column	Contains	Retention	Anonymization Method
`users.email`	PII (email address)	Until erasure (then anonymized)	Replace with `deleted_usr_{hash}@anonymized.local`
`users.first_name`	PII	Until erasure	Replace with `[REDACTED]`
`users.last_name`	PII	Until erasure	Replace with `[REDACTED]`
`users.phone`	PII	Until erasure	Replace with `NULL`
`users.date_of_birth`	PII	Until erasure	Replace with `NULL`
`users.national_id_hash`	PII (hashed)	5 years (AML)	Already hashed; set to `NULL` after retention
`users.password_hash`	Auth credential	Until erasure	Replace with `DELETED`
`bank_accounts.account_number`	Financial PII	5 years (AML)	Replace with `****{last4}`
`bank_accounts.iban`	Financial PII	5 years (AML)	Replace with `****{last4}`
`recipients.bank_account`	Financial PII	5 years (AML)	Replace with `****{last4}`
`recipients.name`	PII	5 years (AML, counterparty)	Replace with `[REDACTED]`
`cards.last_four`	Financial (partial)	5 years	Already truncated
`cards.pin_hash`	Auth credential	Until card cancellation	Set to `NULL`
`audit_log.ip_address`	PII (IP address)	5 years (PSD2)	Replace with `0.0.0.0` after retention
`audit_log.user_agent`	Quasi-PII	5 years	Replace with `[REDACTED]` after retention
`consents.ip_address`	PII	5 years (proof of consent)	Replace with `0.0.0.0` after retention

---

Archival Strategy

Active vs. Archived Data

```
flowchart LR
    A[Active Data<br/>Primary Database] -->|After retention trigger| B[Cold Archive<br/>Read-only Storage]
    B -->|After full retention period| C[Purge<br/>Permanent Deletion]

    subgraph "Active (PostgreSQL)"
        A1[Recent transactions]
```

```
        A2[Active users]
        A3[Current sessions]
    end

    subgraph "Cold Archive (S3/Glacier)"
        B1[Old transactions > 1 year]
        B2[Deleted user records]
        B3[Resolved AML alerts]
        B4[Filed STR reports]
    end

    subgraph "Purge"
        C1[Records past 5-year retention]
        C2[Anonymized analytics retained]
    end
```

Archival Tiers

Tier	Storage	Access Time	Data Types	Cost
Hot (Active DB)	PostgreSQL	Milliseconds	All current data, active users, recent transactions	Primary DB cost
Warm (Archive DB)	PostgreSQL read replica or separate schema	Seconds	Transactions > 1 year, deleted users pending retention	Reduced compute
Cold (Object storage)	AWS S3 / Glacier	Minutes to hours	Compliance exports, old audit logs, filed STR reports	Minimal

Archival Process

1. **Daily job:** Identify records eligible for archival (past active retention period)
2. **Export:** Write eligible records to archive storage (S3 with server-side encryption)
3. **Verify:** Confirm archive integrity (checksum comparison)
4. **Remove from active:** Delete from primary database
5. **Log:** Record archival action in `audit_log`

Deletion Cascades: User Account Deletion

When a user requests account deletion (GDPR Art. 17 right to erasure), the following cascade executes:

```
flowchart TD
    A[DELETE /api/user/account] --> B{Active transactions?}
    B -->|Yes, processing| C[Reject: Wait for completion]
    B -->|No| D[Begin deletion cascade]

    D --> E[Revoke all sessions]
    E --> F[Soft-delete user record]
    F --> G[Anonymize PII fields]
    G --> H[Create data_access_request<br/>type=erasure, status=completed]

    subgraph "Immediate Actions"
        E
        F
        G
    end

    subgraph "Retained for AML (5 years)"
        I[transactions — amounts, dates, types]
        J[audit_log — anonymized entries]
        K[aml_alerts — if any]
        L[str_reports — if any]
        M[screening_results — if any]
    end

    subgraph "Deleted Immediately"
        N[settings — preferences]
        O[notifications — all]
        P[rate_limits — if any for user IP]
    end

    subgraph "Anonymized + Retained"
        Q[bank_accounts — account numbers masked]
```

```
        R[recipients — names redacted]
        S[consents — IP anonymized]
    end

    H --> I
    H --> J
    H --> K
    H --> N
    H --> Q
```

Deletion Cascade Detail

Step	Table	Action	SQL
1	`sessions`	Revoke all	`UPDATE sessions SET revoked = 1 WHERE user_id = ?`
2	`users`	Soft delete + anonymize	`UPDATE users SET deleted_at = CURRENT_TIMESTAMP, email = 'deleted_' || id || '@anonymized.local', first_name = '[REDACTED]', last_name = '[REDACTED]', phone = NULL, date_of_birth = NULL, password_hash = 'DELETED' WHERE id = ?`
3	`settings`	Delete	`DELETE FROM settings WHERE user_id = ?`
4	`notifications`	Delete	`DELETE FROM notifications WHERE user_id = ?`
5	`bank_accounts`	Anonymize	`UPDATE bank_accounts SET account_number = '****' || RIGHT(account_number, 4), iban = CASE WHEN iban IS NOT NULL THEN '****' || RIGHT(iban, 4) END WHERE user_id = ?`
6	`recipients`	Anonymize	`UPDATE recipients SET name = '[REDACTED]', bank_account = '****' || RIGHT(bank_account, 4) WHERE user_id = ?`
7	`consents`	Anonymize IP	`UPDATE consents SET ip_address = '0.0.0.0' WHERE user_id = ?`
8	`cards`	Anonymize	`UPDATE cards SET pin_hash = NULL WHERE user_id = ?`
9	`spending_limits`	Delete	`DELETE FROM spending_limits WHERE user_id = ?`

Step	Table	Action	SQL
10	`data_access_requests`	Create record	`INSERT INTO data_access_requests (id, user_id, request_type, status, completed_at) VALUES (?, ?, 'erasure', 'completed', CURRENT_TIMESTAMP)`
11	`audit_log`	Log deletion	`INSERT INTO audit_log (id, user_id, action, details) VALUES (?, ?, 'user.deleted', '{"reason":"gdpr_erasure"}')`

NOT deleted (AML retention): `transactions`, `audit_log` (existing entries), `aml_alerts`, `str_reports`, `screening_results`, `merchants`. These are retained for 5 years per hvitvaskingsloven section 30, with PII fields anonymized.

Data Subject Access Request (DSAR) Implementation

DSAR Types

Request Type	GDPR Article	SLA	Implementation
Export (right to access)	Art. 15	30 days	`GET /api/user/data-export` -- returns JSON with all user data
Erasure (right to be forgotten)	Art. 17	30 days	`DELETE /api/user/account` -- soft delete + anonymization cascade
Rectification (right to correct)	Art. 16	30 days	`POST /v1/user/rectification` -- updates specified fields, creates data_access_request record
Restriction (right to restrict)	Art. 18	30 days	`POST /v1/user/restriction` -- flags account as restricted, creates data_access_request record

Export Flow

```
sequenceDiagram
    participant U as User
    participant API as API
    participant DB as Database

    U->>API: GET /api/user/data-export
    API->>DB: SELECT * FROM users WHERE id = ?
    API->>DB: SELECT * FROM transactions WHERE user_id = ?
    API->>DB: SELECT * FROM recipients WHERE user_id = ?
    API->>DB: SELECT * FROM bank_accounts WHERE user_id = ?
    API->>DB: SELECT * FROM settings WHERE user_id = ?
    API->>DB: SELECT * FROM consents WHERE user_id = ?

    API->>DB: INSERT INTO data_access_requests<br/>(type='export', status='completed')

    API-->>U: 200 JSON { user, transactions, recipients, bankAccounts, settings, consents }
```

The current implementation (`/api/user/data-export`) returns data inline as JSON. For production, large exports should be written to a temporary signed S3 URL and the `download_url` field in `data_access_requests` populated.

DSAR Tracking

All DSARs are tracked in the `data_access_requests` table:

Field	Purpose
`request_type`	export, erasure, rectification, restriction
`status`	pending -> processing -> completed/rejected
`requested_at`	When the user submitted the request
`completed_at`	When the request was fulfilled
`download_url`	Temporary URL for data export files
`notes`	Internal processing documentation

Anonymization Techniques

For Analytics Retention

After the active retention period, data can be anonymized for analytics rather than deleted:

Data Type	Anonymization Technique	Reversible?	Analytics Value
User identity	Replace name/email with opaque ID	No	User-level metrics without PII
Transaction amounts	Retain exact values (not PII)	N/A	Revenue and volume analytics
Geographic data	Retain country codes only	N/A	Corridor analysis
Timestamps	Retain date, remove time	Partially	Trend analysis
IP addresses	Replace with `0.0.0.0`	No	None (removed for privacy)
Bank account numbers	Replace with `****{last4}`	No	None
Phone numbers	Remove entirely	No	None

Anonymization SQL Pattern

```
-- Anonymize a deleted user's data for analytics retention
UPDATE users SET
    email = 'anon_' || id || '@analytics.internal',
    first_name = '[ANON]',
    last_name = '[ANON]',
    phone = NULL,
    date_of_birth = NULL,
    national_id_hash = NULL,
    password_hash = 'ANONYMIZED'
WHERE id = ? AND deleted_at IS NOT NULL;

-- Transaction data is retained as-is (amounts are not PII)
-- Recipient names are redacted
UPDATE recipients SET
    name = 'Recipient_' || id,
    bank_account = '****' || SUBSTR(bank_account, -4)
WHERE user_id = ?;
```

Legal Basis Reference

Retention Obligation	Law	Section	Requirement

KYC/AML records	Hvitvaskingsloven	Section 30	Retain customer identity and transaction records for 5 years after relationship ends
Transaction records	Bokforingsloven	Section 13	Retain accounting records for 5 years (3.5 years primary, 1.5 years secondary)
Audit trail	PSD2 / Betalingstjenesteloven	Art. 94 impl.	Maintain records of payment transactions for at least 5 years
Consent proof	GDPR	Art. 7(1)	Demonstrate that consent was given (retain proof)
Complaint records	Finansavtaleloven	Section 3-53	Maintain complaint records (15 business day response SLA)
Right to erasure exceptions	GDPR	Art. 17(3)(b)	Erasure does not apply when processing is necessary for compliance with legal obligation
Data minimization	GDPR	Art. 5(1)(c)	Do not retain data longer than necessary for stated purpose
STR records	Hvitvaskingsloven	Section 30	STR reports and supporting documentation retained 5 years after filing

Conflict resolution: When GDPR right to erasure conflicts with AML retention requirements, AML wins per GDPR Art. 17(3)(b). The user is informed that "data [is] retained for 5 years per AML requirements" in the deletion response.

Automated Lifecycle Jobs

Job	Frequency	Action
Session cleanup	Daily	Delete expired sessions older than 90 days
Rate limit cleanup	Every 100 rate limit checks	Delete expired rate limit entries (implemented in `middleware/rate-limit.ts`)
Notification cleanup	Weekly	Archive notifications older than 6 months, delete older than 1 year

Job	Frequency	Action
Audit log archival	Monthly	Move audit entries older than 1 year to cold storage
AML alert archival	Monthly	Archive resolved alerts older than 1 year
User data purge	Monthly	Permanently delete anonymized user data past 5-year retention
Consent proof archival	Monthly	Archive withdrawn consents older than 1 year

Retention Cron Endpoint

The retention enforcement is implemented as `GET /v1/cron/retention` (see `cron.ts`). When triggered, it:

1. **User anonymization** (5+ years post-deletion): Anonymizes PII fields (`email`, `first_name`, `last_name`, `phone`, `date_of_birth`, `national_id_hash`, `password_hash`) for users deleted more than 5 years ago
2. **Session cleanup**: Deletes expired sessions older than 90 days
3. **OTP cleanup**: Removes expired OTP codes (legacy table, wrapped in try/catch)

This endpoint should be called periodically (e.g., daily via external scheduler or cron job). It is not automatically scheduled within the application.

Cross-References

- **Database schema:** DATABASE-SCHEMA.md
- **Database design:** database-design.md
- **Audit architecture:** audit-architecture.md
- **Compliance status:** COMPLIANCE.md
- **Security architecture:** SECURITY-ARCHITECTURE.md
- **GDPR API endpoints:** API-REFERENCE.md (GDPR & Compliance section)
- **Account deletion:** `DELETE /api/user/account` in API-REFERENCE.md
- **Data export:** `GET /api/user/data-export` in API-REFERENCE.md

Audit Architecture

Audit Architecture

Version: 1.0 **Date:** 2026-02-21 **Status:** Approved **Owner:** Database Architect

---

Overview

Drop's audit system records all significant user actions for compliance (PSD2, GDPR, AML) and security monitoring. The `audit_log` table is the central audit store, designed for append-only writes with indexed queries for investigation and reporting.

Regulatory drivers:

- **PSD2 (Betalingstjenesteloven):** Art. 94 requires payment service providers to maintain records of payment transactions for at least 5 years
- **GDPR (Personopplysningsloven):** Art. 5(2) accountability principle -- demonstrate compliance
- **AML (Hvitvaskingsloven):** Section 30 requires retention of all customer due diligence and transaction records
- **Finansavtaleloven:** Section 3-53 requires complaint handling audit trail

---

Audit Log Table Design

Schema

```
CREATE TABLE audit_log (
    id          TEXT PRIMARY KEY,           -- Prefixed ID (e.g., 'aud_a1b2c3...')
    timestamp   TEXT DEFAULT CURRENT_TIMESTAMP,
    user_id     TEXT REFERENCES users(id),  -- NULL for unauthenticated events
    action      TEXT NOT NULL,              -- Event type (e.g., 'auth.login')
    resource_type TEXT,                     -- Entity type (e.g., 'transaction')
    resource_id TEXT,                       -- Entity ID (e.g., 'tx_abc123')
```

```
    details     TEXT,                   -- JSON string with event-specific data
    ip_address  TEXT,                   -- Client IP (from X-Forwarded-For)
    user_agent  TEXT,                   -- Browser/app user agent
    request_id  TEXT                    -- Correlation ID for multi-event requests
);

CREATE INDEX idx_audit_log_user ON audit_log(user_id);
CREATE INDEX idx_audit_log_action ON audit_log(action);
-- Note: idx_audit_log_timestamp is planned but not yet implemented in db.ts
```

Design Rationale

Column	Design Decision
`id`	TEXT with prefix for consistency with all other Drop tables
`timestamp`	TIMESTAMPTZ (PostgreSQL 16 — ADR-014; all environments)
`user_id`	Nullable FK -- some events (failed login, rate limit hit) occur before authentication
`action`	Dot-notation namespace (e.g., `auth.login`, `transaction.create`) for hierarchical filtering
`resource_type` + `resource_id`	Generic resource reference avoids polymorphic FKs while enabling resource-specific queries
`details`	JSON TEXT for flexible, event-specific metadata without schema changes per event type
`request_id`	Correlates multiple audit entries from a single API request (e.g., transaction creation generates audit + notification)

Audit Event Flow

```
flowchart TD
    A[User Action] --> B[API Route Handler]
    B --> C{Action Type}

    C -->|Authentication| D[Auth Events]
    C -->|Transaction| E[Financial Events]
    C -->|Settings/Profile| F[Account Events]
    C -->|Admin/Compliance| G[Admin Events]
```

```
    D --> H[Write audit_log]
    E --> H
    F --> H
    G --> H

    H --> I[Primary Indexes]
    I --> J[idx_audit_log_user<br/>User investigation]
    I --> K[idx_audit_log_timestamp<br/>Time-range queries]
    I --> L[idx_audit_log_action<br/>Event type filtering]

    H --> M{Severity Check}
    M -->|Critical| N[AML Alert Pipeline]
    M -->|Normal| O[Stored for review]
    N --> P[aml_alerts table]
```

```
sequenceDiagram
    participant User
    participant API as API Handler
    participant Audit as Audit Logger
    participant DB as Database
    participant AML as AML Monitor

    User->>API: POST /transactions/remittance
    API->>API: Validate request
    API->>DB: BEGIN transaction
    API->>DB: UPDATE bank_accounts (debit)
    API->>DB: INSERT transactions
    API->>Audit: Log 'transaction.create'
    Audit->>DB: INSERT audit_log
    API->>DB: COMMIT

    Audit->>AML: Check transaction patterns
    AML->>DB: Query recent transactions for user
    alt Suspicious pattern detected
        AML->>DB: INSERT aml_alerts
    end
```

Audit Event Types

Action	Category	Trigger	Logged Details
`auth.login`	Authentication	Successful BankID login	`{method: "bankid", provider: "bankid"}`
`auth.login.failed`	Authentication	Failed login attempt	`{reason: "invalid_credentials", email: "..."}`
`auth.logout`	Authentication	User logout	`{sessions_revoked: N}`
`auth.session.created`	Authentication	New session created	`{session_id: "ses_..."}`
`auth.session.revoked`	Authentication	Session revoked	`{session_id: "ses_..."}`
`auth.token.refreshed`	Authentication	JWT token refreshed	`{new_session_id: "ses_..."}`
`transaction.create`	Financial	Remittance or QR payment created	`{type, amount, currency, fee, recipient_id/merchant_id}`
`transaction.complete`	Financial	Transaction marked completed	`{transaction_id: "tx_..."}`
`transaction.fail`	Financial	Transaction marked failed	`{transaction_id, reason}`
`qr_payment.create`	Financial	QR payment executed	`{amount, merchant_id, fee}`
`bank_account.link`	Account	Bank account linked via AISP	`{bank_name, last4_account}`
`bank_account.balance_sync`	Account	Balance refreshed from AISP	`{bank_account_id, balance}`
`recipient.create`	Account	New recipient added	`{country, currency}`
`recipient.delete`	Account	Recipient removed	`{recipient_id}`
`settings.update`	Account	User settings changed	`{changed_fields: ["currency","language"]}`
`merchant.register`	Account	Merchant profile created	`{business_name, org_number}`
`kyc.status_change`	Compliance	KYC status updated	`{old_status, new_status, method}`
`consent.granted`	Compliance	GDPR consent given	`{consent_type, ip_address}`
`consent.withdrawn`	Compliance	GDPR consent withdrawn	`{consent_type, ip_address}`
`dsar.export`	Compliance	Data export request completed	`{request_id}`
`dsar.erasure`	Compliance	Account deletion requested	`{request_id}`
`complaint.created`	Compliance	Customer complaint filed	`{category, complaint_id}`
`complaint.resolved`	Compliance	Complaint resolved	`{complaint_id, resolution_days}`

Action	Category	Trigger	Logged Details
`aml.alert_created`	AML	Suspicious activity detected	`{alert_type, severity, transaction_id}`
`aml.alert_resolved`	AML	AML alert investigated and resolved	`{alert_id, resolution}`
`str.filed`	AML	STR submitted to authorities	`{str_id, reference_number}`
`screening.completed`	AML	PEP/sanctions screening done	`{screening_type, result}`
`user.deleted`	Account	User account deleted (GDPR)	`{reason: "gdpr_erasure"}`
`rate_limit.exceeded`	Security	Rate limit hit	`{endpoint, ip_address, limit}`
`card.created`	Account	Card created (FUTURE)	`{type, last_four}`
`card.frozen`	Account	Card frozen (FUTURE)	`{card_id}`
`card.cancelled`	Account	Card cancelled (FUTURE)	`{card_id}`

Tamper Detection

Hash Chain Mechanism

To ensure audit log integrity (detect unauthorized modifications or deletions), the audit system should implement a hash chain:

```
flowchart LR
    E1[Entry 1<br/>hash = SHA256<br/>data + genesis] --> E2[Entry 2<br/>hash = SHA256<br/>data + E1.hash]
    E2 --> E3[Entry 3<br/>hash = SHA256<br/>data + E2.hash]
    E3 --> E4[Entry N<br/>hash = SHA256<br/>data + E(N-1).hash]
```

Proposed implementation:

Add a `chain_hash` column to `audit_log`:

```
ALTER TABLE audit_log ADD COLUMN chain_hash TEXT;
```

Each entry's hash is computed as:

```
chain_hash = SHA256(
    timestamp || user_id || action || resource_type ||
    resource_id || details || previous_chain_hash
)
```

Verification: Walk the chain from the first entry, recomputing each hash. A mismatch indicates tampering. This can be run as a periodic integrity check job.

Current status: Not yet implemented. The audit table stores events without tamper detection. Hash chain is a Phase 3 enhancement (pre-production launch).

Alternative: Append-Only with Write-Once Storage

For production, audit logs should be replicated to write-once storage (AWS S3 Object Lock / Glacier Vault Lock) within minutes of creation. This provides:

- Immutability guarantee independent of database access
- External verification point
- Compliance with regulatory requirements for tamper-evident audit trails

---

Compliance Requirements

PSD2 Audit Trail

Requirement	Implementation	Status
Record all payment transactions	`transaction.create` event with full details	Implemented
Record authentication events	`auth.login`, `auth.logout` events	Implemented
Record consent actions	`consent.granted`, `consent.withdrawn` events	Implemented
5-year retention	Retention policy defined (see data-lifecycle.md)	Policy defined
Tamper-evident	Hash chain proposed, not yet implemented	Planned

GDPR Audit Trail

Requirement	Implementation	Status
Record data access requests	`dsar.export`, `dsar.erasure` events	Implemented
Record consent changes	`consent.granted`, `consent.withdrawn` events	Implemented
Record data deletion	`user.deleted` event	Implemented
Demonstrate accountability	Full audit trail queryable by user_id	Implemented

AML Audit Trail

Requirement	Implementation	Status
Record suspicious activity alerts	`aml.alert_created` event	Implemented
Record STR filings	`str.filed` event	Implemented
Record screening results	`screening.completed` event	Implemented
Record KYC status changes	`kyc.status_change` event	Implemented

Log Retention and Searchability

Query Patterns

Query	Use Case	Index Used	Example SQL
All events for a user	Investigation, DSAR	`idx_audit_log_user`	`SELECT * FROM audit_log WHERE user_id = ? ORDER BY timestamp DESC`
Events in time range	Compliance reporting	`idx_audit_log_timestamp`	`SELECT * FROM audit_log WHERE timestamp BETWEEN ? AND ?`
Events by type	Pattern analysis	`idx_audit_log_action`	`SELECT * FROM audit_log WHERE action = 'transaction.create'`
Events for a resource	Transaction audit trail	Sequential scan (consider composite index)	`SELECT * FROM audit_log WHERE resource_type = 'transaction' AND resource_id = ?`
Recent events globally	Dashboard, monitoring	`idx_audit_log_timestamp`	`SELECT * FROM audit_log ORDER BY timestamp DESC LIMIT 50`

Retention Tiers

Tier	Age	Storage	Access
Hot	0-3 months	Primary database, fully indexed	Real-time queries
Warm	3-12 months	Primary database, indexed	Standard queries
Cold	1-5 years	Archive storage (S3)	Restored on demand
Purge	5+ years	Deleted	Not available

Cross-References

- **Audit log schema:** DATABASE-SCHEMA.md (audit_log section)
- **Data lifecycle:** data-lifecycle.md (retention periods)
- **Indexing strategy:** indexing-strategy.md (audit log indexes)
- **Compliance status:** COMPLIANCE.md
- **Security architecture:** SECURITY-ARCHITECTURE.md

Indexing Strategy

Indexing Strategy

Version: 1.0 **Date:** 2026-02-21 **Status:** Approved **Owner:** Database Architect

---

Overview

Drop's indexing strategy is designed around actual user flow query patterns. Every index exists because a specific query needs it. No speculative indexes.

Current index count: 16 indexes across 19 tables (defined in `db.ts` schema).

Query Patterns by User Flow

Login Flow (BankID OIDC)

```
sequenceDiagram
    participant U as User
    participant API as API
    participant DB as Database

    U->>API: BankID callback (code, state)
    API->>DB: Q1: Find user by national_id_hash
    DB-->>API: User or NULL
    alt New user
        API->>DB: Q2: INSERT users
        API->>DB: Q3: INSERT settings (defaults)
    end
    API->>DB: Q4: INSERT sessions
    API-->>U: JWT cookie + redirect
```

Query ID	SQL Pattern	Index Required	Current Coverage
Q1	`SELECT * FROM users WHERE national_id_hash = ?`	`idx_users_national_id` (partial: WHERE NOT NULL)	Covered
Q2	`INSERT INTO users (...)`	None (PK insert)	N/A
Q3	`INSERT INTO settings (...)`	None (PK insert)	N/A
Q4	`INSERT INTO sessions (...)`	None (PK insert)	N/A

Authentication Middleware (every authenticated request)

Query ID	SQL Pattern	Index Required	Current Coverage
Q5	`SELECT * FROM sessions WHERE token_hash = ? AND revoked = 0 AND expires_at > ?`	`idx_sessions_token`	Covered
Q6	`SELECT * FROM users WHERE id = ?`	PRIMARY KEY	Covered

Dashboard View

```
sequenceDiagram
    participant U as User
    participant API as API
    participant DB as Database

    U->>API: GET /auth/me
    API->>DB: Q5: Verify session (token_hash)
    API->>DB: Q6: Get user by PK
    API->>DB: Q7: Get bank accounts for user
    DB-->>API: Bank accounts with cached balances
    API-->>U: User profile + total balance
```

Query ID	SQL Pattern	Index Required	Current Coverage
Q7	`SELECT * FROM bank_accounts WHERE user_id = ?`	`idx_bank_accounts_user`	Covered

Transaction History

Query ID	SQL Pattern	Index Required	Current Coverage
Q8	`SELECT * FROM transactions WHERE user_id = ? [AND type = ?] [AND status = ?] ORDER BY created_at DESC LIMIT ? OFFSET ?`	`idx_transactions_user`	Covered (user_id)
Q9	`SELECT COUNT(*) FROM transactions WHERE user_id = ?`	`idx_transactions_user`	Covered

Note: The `type` and `status` filters are applied after the user_id index lookup. At current scale (< 10K transactions per user), this is efficient. A composite index `(user_id, created_at DESC)` would optimize the ORDER BY for users with many transactions.

Create Remittance

Query ID	SQL Pattern	Index Required	Current Coverage
Q10	`SELECT * FROM recipients WHERE id = ? AND user_id = ?`	`idx_recipients_user` + PK	Covered
Q11	`SELECT * FROM exchange_rates WHERE to_currency = ?`	Sequential scan (6 rows)	Acceptable (tiny table)
Q12	`SELECT * FROM bank_accounts WHERE user_id = ? AND is_primary = 1`	`idx_bank_accounts_user`	Covered
Q13	`UPDATE bank_accounts SET balance = balance - ? WHERE id = ? AND balance >= ?`	PK	Covered
Q14	`INSERT INTO transactions (...)`	None (PK insert)	N/A

Create QR Payment

Query ID	SQL Pattern	Index Required	Current Coverage
Q15	`SELECT * FROM merchants WHERE id = ?`	PK	Covered
Q12	(Same as remittance -- primary bank account)	`idx_bank_accounts_user`	Covered
Q13	(Same as remittance -- balance debit)	PK	Covered
Q14	(Same as remittance -- insert transaction)	N/A	N/A

Notifications List

Query ID	SQL Pattern	Index Required	Current Coverage
Q16	`SELECT * FROM notifications WHERE user_id = ? ORDER BY created_at DESC`	`idx_notifications_user`	Covered
Q17	`UPDATE notifications SET read = 1 WHERE id IN (?, ?, ...) AND user_id = ?`	PK + `idx_notifications_user`	Covered

Settings View/Update

Query ID	SQL Pattern	Index Required	Current Coverage
Q18	`SELECT * FROM settings WHERE user_id = ?`	PK (user_id IS the PK)	Covered
Q19	`UPDATE settings SET ... WHERE user_id = ?`	PK	Covered

Merchant Dashboard

Query ID	SQL Pattern	Index Required	Current Coverage
Q20	`SELECT * FROM merchants WHERE user_id = ?`	Sequential scan (1 merchant per user)	Acceptable
Q21	`SELECT * FROM transactions WHERE merchant_id = ? AND created_at >= ?`	`idx_transactions_merchant`	Partially covered (no composite with created_at)

Recipient Management

Query ID	SQL Pattern	Index Required	Current Coverage
Q22	`SELECT * FROM recipients WHERE user_id = ? LIMIT ? OFFSET ?`	`idx_recipients_user`	Covered
Q23	`SELECT * FROM recipients WHERE id = ? AND user_id = ?`	PK + `idx_recipients_user`	Covered
Q24	`DELETE FROM recipients WHERE id = ? AND user_id = ?`	PK	Covered

Compliance Queries (Admin/Internal)

Query ID	SQL Pattern	Index Required	Current Coverage

Q25	`SELECT * FROM audit_log WHERE user_id = ? ORDER BY timestamp DESC`	`idx_audit_log_user`	Covered
Q26	`SELECT * FROM audit_log WHERE action = ? AND timestamp BETWEEN ? AND ?`	`idx_audit_log_action` + `idx_audit_log_timestamp`	Partially (separate indexes, no composite)
Q27	`SELECT * FROM aml_alerts WHERE user_id = ? AND status IN ('open','investigating')`	`idx_aml_alerts_user`	Covered
Q28	`SELECT * FROM aml_alerts WHERE status = 'open' ORDER BY created_at`	`idx_aml_alerts_status` (proposed)	Not covered (needs new index)
Q29	`SELECT * FROM complaints WHERE user_id = ? ORDER BY created_at DESC`	`idx_complaints_user`	Covered
Q30	`SELECT * FROM complaints WHERE status IN ('received','investigating')`	`idx_complaints_status` (proposed)	Not covered (needs new index)

Index Inventory

Current Indexes (defined in `db.ts`)

Index Name	Table	Column(s)	Type	Rationale
`idx_users_national_id`	`users`	`national_id_hash`	B-tree, partial (WHERE NOT NULL)	BankID login deduplication -- find user by hashed national ID
`idx_recipients_user`	`recipients`	`user_id`	B-tree	List recipients per user, verify ownership
`idx_transactions_user`	`transactions`	`user_id`	B-tree	Transaction history per user (most frequent query)
`idx_transactions_merchant`	`transactions`	`merchant_id`	B-tree	Merchant dashboard -- transactions for merchant (documented in DATABASE-SCHEMA.md)
`idx_tx_idempotency`	`transactions`	`idempotency_key`	B-tree, unique, partial (WHERE NOT NULL)	Prevent duplicate transaction submission

Index Name	Table	Column(s)	Type	Rationale
`idx_bank_accounts_user`	`bank_accounts`	`user_id`	B-tree	Dashboard balance lookup, transaction source
`idx_sessions_user`	`sessions`	`user_id`	B-tree	Revoke all sessions on logout
`idx_sessions_token`	`sessions`	`token_hash`	B-tree	Auth middleware -- validate session on every request
`idx_notifications_user`	`notifications`	`user_id`	B-tree	Notifications list per user (documented in DATABASE-SCHEMA.md)
`idx_audit_log_user`	`audit_log`	`user_id`	B-tree	User investigation, DSAR compliance
`idx_audit_log_action`	`audit_log`	`action`	B-tree	Event type filtering for monitoring
`idx_audit_log_timestamp`	`audit_log`	`timestamp`	B-tree	Time-range queries for compliance reporting (documented in DATABASE-SCHEMA.md)
`idx_aml_alerts_user`	`aml_alerts`	`user_id`	B-tree	Per-user AML alert lookup
`idx_aml_alerts_status`	`aml_alerts`	`status`	B-tree	Open alerts dashboard (documented in DATABASE-SCHEMA.md)
`idx_complaints_user`	`complaints`	`user_id`	B-tree	Per-user complaint history
`idx_screening_user`	`screening_results`	`user_id`	B-tree	Per-user screening history

Indexes from DATABASE-SCHEMA.md (not in db.ts code)

The DATABASE-SCHEMA.md documentation lists additional indexes that may not be in the current `db.ts` SQLITE_SCHEMA string:

Index Name	Table	Column(s)	Status

`idx_merchants_org`	`merchants`	`org_number`	Documented but covered by UNIQUE constraint
`idx_cards_user`	`cards`	`user_id`	Documented, may not be in db.ts
`idx_spending_limits_user`	`spending_limits`	`user_id`	Documented, may not be in db.ts
`idx_spending_limits_card`	`spending_limits`	`card_id`	Documented, may not be in db.ts
`idx_consents_user`	`consents`	`user_id`	Documented, may not be in db.ts
`idx_data_requests_user`	`data_access_requests`	`user_id`	Documented, may not be in db.ts
`idx_complaints_status`	`complaints`	`status`	Documented, may not be in db.ts

Recommendation: Reconcile DATABASE-SCHEMA.md with actual `db.ts` code. Add missing indexes to the schema if the queries justify them.

Recommended Additional Indexes

Based on query pattern analysis, the following indexes should be added:

Proposed Index	Table	Column(s)	Justification
`idx_transactions_user_created`	`transactions`	`(user_id, created_at DESC)`	Optimizes paginated transaction history (Q8) -- avoids sort after index lookup
`idx_complaints_status`	`complaints`	`status`	Admin dashboard query for open complaints (Q30)
`idx_consents_user`	`consents`	`user_id`	DSAR export needs all consents for user
`idx_data_requests_user`	`data_access_requests`	`user_id`	DSAR tracking per user
`idx_audit_log_resource`	`audit_log`	`(resource_type, resource_id)`	Resource-specific audit trail lookup

Partial Index Opportunities (PostgreSQL)

These are PostgreSQL-specific optimizations to add after migration:

Proposed Index	Table	Column(s)	Condition	Justification

`idx_sessions_active`	`sessions`	`user_id`	`WHERE revoked = 0`	Auth middleware only queries active sessions
`idx_aml_alerts_open`	`aml_alerts`	`created_at`	`WHERE status IN ('open','investigating')`	Dashboard shows only open alerts
`idx_notifications_unread`	`notifications`	`user_id`	`WHERE read = 0`	Badge count for unread notifications
`idx_users_active`	`users`	`email`	`WHERE deleted_at IS NULL`	Login only checks non-deleted users

Connection Pooling Configuration

> **Note (ADR-014, 2026-03-03):** Drop uses PostgreSQL 16 in ALL environments. SQLite and the dual-driver layer have been removed. The section below reflects the current PostgreSQL-only configuration.

PostgreSQL 16 (All Environments)

PostgreSQL uses Drizzle ORM with connection pooling:

Setting	Value	Source	Notes
Pool library	`pg.Pool`	`db.ts:16-21`	Node-postgres built-in pool
Connection string	`DATABASE_URL` env var	`db.ts:18`	Standard PostgreSQL URL format
Max connections	Default (10)	pg.Pool default	Adjust based on App Runner instance count
Idle timeout	10,000ms	pg.Pool default	Close idle connections after 10s
Connection timeout	0 (no timeout)	pg.Pool default	Wait indefinitely for connection

Recommended Production Pool Configuration

```
// Recommended pg.Pool configuration for production
const pool = new pg.Pool({
    connectionString: process.env.DATABASE_URL,
```

```
    max: 20,                    // Max connections per instance
    idleTimeoutMillis: 30000,   // Close idle after 30s
    connectionTimeoutMillis: 5000, // Fail if no connection in 5s
    ssl: { rejectUnauthorized: true } // Require SSL for RDS
});
```

Parameter	Recommended Value	Rationale
`max`	20	Balance between connection availability and RDS connection limits. With 2-3 App Runner instances, total connections = 40-60 (well under RDS default 100).
`idleTimeoutMillis`	30,000	Close idle connections to free RDS slots, but keep them long enough to avoid reconnection overhead for bursty traffic.
`connectionTimeoutMillis`	5,000	Fail fast on connection issues rather than hanging. API should return 503 to client.
`ssl`	`{ rejectUnauthorized: true }`	Encrypt connections to RDS. Required for compliance.

PgBouncer Consideration

At current projected scale (3,000 users, ~100 concurrent connections), direct `pg.Pool` is sufficient. PgBouncer should be evaluated when:

- Connection count exceeds RDS limits
- Multiple services need to share the same database
- Transaction-mode pooling would reduce connection overhead

EXPLAIN ANALYZE Examples

Transaction History Query (most common)

```
-- PostgreSQL 16 (all environments — ADR-014)
EXPLAIN ANALYZE
SELECT * FROM transactions
WHERE user_id = 'usr_demo1'
```

```
ORDER BY created_at DESC
LIMIT 20 OFFSET 0;
-- Expected: Index Scan using idx_transactions_user on transactions
--           Sort Key: created_at DESC
--           Rows Removed by Index: 0 (all rows match user_id)
```

Session Validation (every request)

```
-- PostgreSQL 16
EXPLAIN ANALYZE
SELECT * FROM sessions
WHERE token_hash = 'abc123...'
AND revoked = FALSE
AND expires_at > NOW();
-- Expected: Index Scan using idx_sessions_token on sessions (token_hash=?)

-- PostgreSQL
EXPLAIN ANALYZE
SELECT * FROM sessions
WHERE token_hash = 'abc123...'
AND revoked = 0
AND expires_at > '2026-02-21T00:00:00';
-- Expected: Index Scan using idx_sessions_token on sessions (cost=0.28..8.30)
--           Filter: (revoked = 0 AND expires_at > ...)
```

Audit Log by User (investigation)

```
-- PostgreSQL
EXPLAIN ANALYZE
SELECT * FROM audit_log
WHERE user_id = 'usr_demo1'
ORDER BY timestamp DESC
LIMIT 100;
-- Expected: Index Scan Backward using idx_audit_log_user
```

Performance Monitoring

Key Metrics to Track

Metric	Target	Alert Threshold	Query
Session validation latency	< 5ms	> 20ms	`SELECT * FROM sessions WHERE token_hash = ?`
Transaction list latency	< 50ms	> 200ms	`SELECT * FROM transactions WHERE user_id = ? ORDER BY created_at DESC LIMIT 20`
Audit log write latency	< 10ms	> 50ms	`INSERT INTO audit_log (...)`
Index bloat	< 20%	> 50%	`pg_stat_user_indexes`
Sequential scans on large tables	0	Any	`pg_stat_user_tables.seq_scan` for transactions, audit_log

Periodic Index Maintenance (PostgreSQL)

```
-- Check index usage
SELECT schemaname, tablename, indexname, idx_scan, idx_tup_read
FROM pg_stat_user_indexes
ORDER BY idx_scan ASC;

-- Check for bloated indexes
SELECT pg_size_pretty(pg_relation_size(indexrelid)) as size, indexrelid::regclass
FROM pg_stat_user_indexes
ORDER BY pg_relation_size(indexrelid) DESC;

-- Reindex if bloated
REINDEX INDEX CONCURRENTLY idx_transactions_user;
```

Cross-References

- **Database schema:** DATABASE-SCHEMA.md
- **Database design:** database-design.md
- **Audit architecture:** audit-architecture.md
- **Data architecture:** data-architecture.md
- **Migration strategy:** migration-strategy.md (PostgreSQL-specific optimizations)
- **Drizzle ORM schema:** `src/shared/db/schema.ts`