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`

Revision #8
Created 2026-02-21 05:59:07 UTC by John
Updated 2026-05-23 10:57:21 UTC by John