ADR-006: SQLite to PostgreSQL

ADR-006: SQLite for Development, PostgreSQL for Production

Status: SUPERSEDED by ADR-014: PostgreSQL-Only Architecture (2026-03-03) **Date:** 2026-02-21
Deciders: John (AI Director) **Category:** Database

Context

Drop requires a database strategy that supports rapid local development while being production-ready for a financial application. The key tension is between developer velocity (zero-config local setup) and production reliability (concurrent writes, ACID transactions, managed backups).

Database	Local Setup	Concurrent Writes	Managed Services	Financial Compliance
SQLite	Zero config, file-based	Poor (single writer)	None	Not suitable for production
PostgreSQL	Docker required	Excellent (MVCC)	AWS RDS, Aurora	Industry standard for fintech
MySQL	Docker required	Good	AWS RDS, Aurora	Common but less feature-rich

Drop's data model includes 19 tables with foreign keys, transactions requiring atomicity (balance deduction + transaction record), and compliance tables needing reliable concurrent access for audit logging. SQLite handles development workloads but cannot support concurrent writes from multiple App Runner instances.

Decision

Use SQLite (`better-sqlite3`) for development and PostgreSQL for production, with a dual-driver abstraction layer.

Driver detection is automatic based on environment:

```
const USE_PG = !!process.env.DATABASE_URL;
```

When `DATABASE_URL` is set (production), PostgreSQL is used. Otherwise, SQLite with WAL mode is used.

```
graph LR
    subgraph dev["Development"]
        app_dev["Drop App"] --> dal["Database Access Layer<br/>(db.ts)"]
        dal --> sqlite["SQLite<br/>(better-sqlite3)<br/>./data/drop.db"]
    end

    subgraph prod["Production"]
        app_prod["Drop App (x N)"] --> dal2["Database Access Layer<br/>(db.ts)"]
        dal2 --> pg["PostgreSQL<br/>(AWS RDS)<br/>DATABASE_URL"]
    end

    dal -.->|"Same API:<br/>query(), getOne(),<br/>run(), transaction()"| dal2
```

See ADR-010: Dual Database Driver for the abstraction layer details.

Consequences

Positive

- Zero-config local development: `npm run dev` just works, no Docker needed for DB
- Production-grade concurrent access with PostgreSQL MVCC
- AWS RDS provides automated backups, point-in-time recovery (critical for financial data)
- Same application code runs against both databases via abstraction layer
- SQLite WAL mode provides good read performance during development

Negative

- SQL compatibility layer adds complexity (see ADR-010)
- Subtle behavioral differences between SQLite and PostgreSQL (e.g., type coercion, datetime handling)

- Cannot test PostgreSQL-specific features locally without Docker
- Must test against both databases in CI

Risks

- **SQL dialect drift:** A query that works in SQLite may fail in PostgreSQL. Mitigation: dual-driver abstraction normalizes SQL; CI tests against both.
- **Performance characteristics differ:** SQLite is faster for single-connection workloads. Mitigation: performance testing against PostgreSQL before production launch.

References

- ADR-010: Dual Database Driver -- Abstraction layer implementation
- Database Schema -- Full schema documentation
- Database Design -- Database architecture decisions
- Migration Strategy -- SQLite to PostgreSQL migration plan

Revision #7
Created 2026-02-21 05:58:59 UTC by John
Updated 2026-05-23 10:56:42 UTC by John