# ADR-004: SQLite as Primary Database

- Architecture Diagrams
- Context and Problem Statement
- Decision Drivers
- Considered Options
- Decision Outcome
  - Positive Consequences
  - Negative Consequences
- Pros and Cons of the Options
  - SQLite
  - PostgreSQL
  - MySQL/MariaDB
  - MongoDB
  - JSON Files
- Links
- Implementation Notes
- Review History

## Architecture Diagrams

**Status:** Accepted **Date:** 2025-09-20 (estimated) **Deciders:** System Architect, Database Engineer **Technical Story:** Database selection for subject/session tracking

## Context and Problem Statement

TOSCA requires persistent storage for: - Subject records (demographics, treatment history) - Session data (treatment sessions, events, outcomes) - Protocol definitions (treatment parameters) - Safety logs (immutable audit trail) - Calibration data (device calibrations over time)

The system is single-user (one technician at a time), desktop-based (Windows 10/11), and requires HIPAA-compliant encryption in production. Which database should TOSCA use?

## Decision Drivers

- Single-user, single-machine deployment (no network access required)
- Medical device data integrity requirements (ACID compliance)
- Zero-configuration deployment (embedded database preferred)
- Research mode development (encryption deferred to Phase 6)
- Easy backup/restore for data migration
- Support for complex queries (session history, treatment analytics)
- File-based storage for portability

## Considered Options

- SQLite (embedded, file-based, zero-config)
- PostgreSQL (client-server, full-featured, requires setup)
- MySQL/MariaDB (client-server, widely used, requires setup)
- MongoDB (NoSQL, document-based, schema-less)
- JSON files (flat file storage, no query capability)

## Decision Outcome

Chosen option: “SQLite”, because it provides zero-configuration embedded database with full ACID compliance, perfect for single-user desktop applications. SQLCipher extension available for Phase 6 encryption. File-based storage simplifies backup/restore and data portability.

### Positive Consequences

- **Zero Configuration:** No database server to install/manage
- **Embedded:** Database file lives with application data (`data/tosca.db`)
- **ACID Compliance:** Full transaction support, data integrity guarantees
- **SQLAlchemy ORM:** Excellent Python ORM support for cleaner code
- **Alembic Migrations:** Database schema versioning and migrations
- **SQLCipher Ready:** Encryption extension available for Phase 6 (transparent AES-256-GCM)
- **Portable:** Single file, easy backup/restore
- **Proven:** Used in medical devices, aerospace, automotive (highly reliable)

### Negative Consequences

- **Single-User Limitation:** No concurrent write access (acceptable for TOSCA use case)
- **No Built-In Encryption:** Phase 6 requires SQLCipher migration (plaintext in research mode)
- **Performance at Scale:** Large datasets (millions of events) may require optimization
- **No User Management:** Application-level authentication required (Phase 6)

## Pros and Cons of the Options

### SQLite

- Good, because zero-configuration deployment (no database server)
- Good, because ACID compliance ensures data integrity
- Good, because file-based storage simplifies backup (copy single file)
- Good, because SQLCipher extension provides transparent encryption
- Good, because excellent Python support (sqlite3 built-in, SQLAlchemy ORM)
- Good, because proven reliability in medical devices and safety-critical systems
- Bad, because no built-in encryption (requires SQLCipher for HIPAA compliance)
- Bad, because single-writer limitation (not a concern for single-user system)
- Bad, because large binary data (videos) should be stored in filesystem, not database

### PostgreSQL

- Good, because most feature-rich open-source database
- Good, because built-in encryption options (pgcrypto)
- Good, because excellent performance at scale
- Good, because multi-user support with fine-grained access control
- Bad, because requires database server installation and configuration
- Bad, because overkill for single-user desktop application
- Bad, because network configuration adds complexity
- Bad, because backup requires pg_dump or continuous archiving

### MySQL/MariaDB

- Good, because widely used, large community
- Good, because multi-user support
- Good, because proven in production environments
- Bad, because requires database server installation
- Bad, because more complex than SQLite for single-user use case
- Bad, because network configuration required even for localhost

### MongoDB

- Good, because schema-less (flexible data model)
- Good, because JSON-native (natural fit for event logging)
- Good, because horizontal scaling (not needed for TOSCA)
- Bad, because requires MongoDB server installation
- Bad, because no ACID compliance for multi-document transactions (until v4.0+)
- Bad, because less mature Python ORM support than SQL databases
- Bad, because encryption requires MongoDB Enterprise (commercial license)

### JSON Files

- Good, because simplest possible implementation (no dependencies)
- Good, because human-readable for debugging
- Good, because easy backup (copy files)
- Bad, because no query capability (must load entire file)
- Bad, because no transaction support (data corruption risk)
- Bad, because no relational integrity (foreign keys, constraints)
- Bad, because poor performance for large datasets
- Bad, because no built-in indexing

## Links

- SQLite Documentation
- SQLCipher Encryption
- SQLite in Medical Devices
- Related: [ADR-008-security-architecture.md] (Phase 6 SQLCipher migration)
- Related: [02_database_schema.md] (Complete schema documentation)

## Implementation Notes

**Current Status (v0.9.12-alpha):** - Database file: `data/tosca.db` (UNENCRYPTED - plaintext) - SQLAlchemy ORM used for all database operations - Alembic migrations configured but not yet used (single schema version) - Database created on first launch if missing

**Phase 6 Migration Plan (SQLCipher):** 1. Install SQLCipher Python bindings (`pip install sqlcipher3`) 2. Generate 256-bit encryption key (store securely, NOT in code) 3. Migrate existing `tosca.db` to encrypted format:

```
sqlcipher tosca.db
sqlite> PRAGMA key = 'encryption-key';
sqlite> ATTACH DATABASE 'encrypted.db' AS encrypted KEY 'encryption-key';
sqlite> SELECT sqlcipher_export('encrypted');
sqlite> DETACH DATABASE encrypted;
```

4. Update SQLAlchemy connection string to use SQLCipher
5. Implement key management (hardware token, Windows DPAPI, or secure key store)

**Two-Tier Logging Strategy:** - **High-frequency data (100Hz+):** JSONL files in session folders - Photodiode readings, camera metadata, real-time interlock states - Rationale: SQLite write performance degrades with 100+ inserts/second - **Event-based data:** SQLite database - Protocol steps, power changes, safety triggers, user actions - Rationale: Queryable, relational integrity, transaction support

**Backup Strategy:** 1. Automated daily backup: Copy `tosca.db` to `data/backups/tosca_YYYYMMDD.db` 2. Before schema migrations: Backup + verify integrity 3. Session data: Backup session folders to external drive/cloud monthly

## Review History

| Date | Reviewer | Notes |
|---|---|---|
| 2025-09-20 | System Architect | Initial SQLite selection |
| 2025-10-30 | Security Review | Confirmed SQLCipher migration path for Phase 6 |
| 2025-11-04 | Documentation Review | Formalized as ADR-004 |

---

**Template Version:** 1.0 **Last Updated:** 2025-11-04