SQLite with Dart

File-Based Relational Database

- What is SQLite?
    - Architecture Overview
- Dart SQLite
    - Project Setup
- SQLite execute and select
- FooBar Data Model
    - SQLite Dart Programming
    - CRUD
- SQL Query Fundamentals
- foobar project
    - Development Workflow
- Databases
    - Decision Framework
    - SQLite Limitations

What is SQLite?

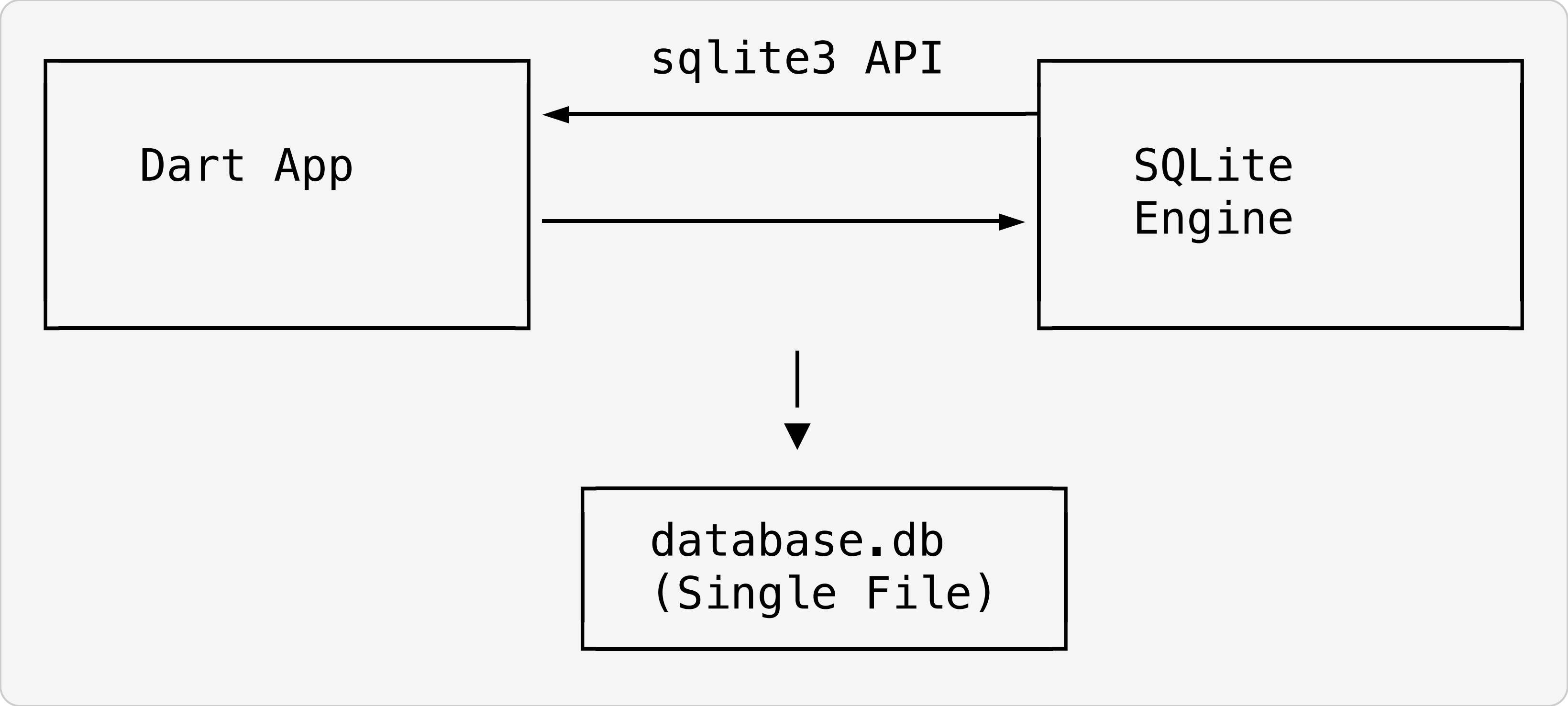
- 📁 **File-based database** - Single file contains entire database
- 🔄 **Relational database** with full SQL support
- 🚀 **Zero configuration** - No server setup required
- 📱 **Embedded** in applications (mobile, desktop, web)
- 🌍 **Most widely deployed** database engine in the world

Key Characteristics:

- Full ACID compliance (transactions)
- Complete SQL implementation
- Cross-platform compatibility
- Public domain (no licensing fees)
- Self-contained C library

Used by: Pocketbase, Android, iOS, browsers, desktop apps, embedded systems

Architecture Overview



Dart SQLite

- Using SQLite in Dart is making SQLite through the helper functions.
- We cannot use SQLite for client side web applications.
  - We can use SQLite for server side web applications.
  - For client side web application, we can use IndexedDB.

Project Setup

pubspec.yaml

```yaml
dependencies:
  sqlite3: ^2.4.0    # Pure Dart SQLite interface
  path: ^1.8.3       # Cross-platform path handling
```

```dart
import 'package:sqlite3/sqlite3.dart';
import 'package:path/path.dart' as p;

final dbPath = p.join('data', 'my_database.db');
final db = sqlite3.open(dbPath);
```

SQLite execute and select

Understand how to run SQL commands safely and efficiently.

- Use "db.execute" to **change** data.
- Use "db.select" to **retrieve** data.
- Use "prepare + execute/select" to stay **safe and efficient**.

- *Quick Execution:* *`db.execute`*
- 🔹 Runs raw SQL directly
- 🔹 Good for one-time commands
- ⚠️ Risk: SQL injection if inputs are embedded directly

```
db.execute("INSERT INTO students (name, age)
            VALUES ('Alice', 22)");
```

- Safe & Reusable: `db.prepare + stmt.execute`
- 🔐 Uses placeholders (`?`) to prevent injection
- 🔁 Efficient for repeated queries
- 🧹 Requires cleanup with `dispose()`

```
final stmt = db.prepare(
  "INSERT INTO students (name, age) VALUES (?, ?)");
stmt.execute(['Alice', 22]);
stmt.dispose();
```

db.select vs db.execute

Feature	db.execute	db.select
Purpose	INSERT/UPDATE/DELETE	SELECT queries
Returns	Nothing or metadata	List of rows/maps
Example	db.execute("DELETE ...")	db.select("SELECT * FROM ...")

- For Queries, use `db.select`.
- For other actions, use `db.execute`.

FooBar Data Model

- The FooBar class

```
import 'package:sqlite3/sqlite3.dart';
class FooBar {
  String foo;
  int bar;
  FooBar({
    required this.foo,
    required this.bar,
  });
```

Conversion from/to SQLite

```
factory FooBar.fromRow(Row row) {
  return FooBar(
    foo: row['foo'] as String,
    bar: row['bar'] as int,
  );
}

Map<String, dynamic> toMap() => {
      'foo': foo,
      'bar': bar,
    };
```

The == operator for object comparison

```
  @override
  bool operator ==(Object other) {
    if (identical(this, other)) return true;
    return other is FooBar && other.foo == foo && other.bar == bar;
  }

  @override
  int get hashCode => foo.hashCode ^ bar.hashCode;
}
```

SQLite Dart Programming

FooBarCrudSQLite is the class for SQLite programming.

```dart
Database? _database;
final String _databaseName = 'foobar.db';
final String _tableName = 'foobars';
final String _dataDirectory = 'data';
```

database property

```
Future<Database> get database async {
  if (_database != null) return _database!;

  // Ensure data directory exists
  final dataDir = Directory(_dataDirectory);
  if (!await dataDir.exists()) {
    await dataDir.create(recursive: true);
  }

  String dbPath = join(_dataDirectory, _databaseName);
  _database = sqlite3.open(dbPath);
  _createTableIfNotExists();
  return _database!;
}
```

create table

- Fromm the database, we create table if not exists.

```
void _createTableIfNotExists() {
   final db = _database!;
   db.execute('''
     CREATE TABLE IF NOT EXISTS $_tableName (
       id INTEGER PRIMARY KEY AUTOINCREMENT,
       foo TEXT NOT NULL,
       bar INTEGER NOT NULL,
       created_at DATETIME DEFAULT CURRENT_TIMESTAMP
     )
   ''');
 }
```

CRUD

- CREATE: INSERT INTO
- READ: SELECT * FROM
- UPDATE: UPDATE
- DELETE: DELETE FROM

CREATE

```dart
Future<int> create(FooBar foobar) async {
  final db = await database;

  final stmt = db.prepare('''
    INSERT INTO $_tableName (foo, bar)
    VALUES (?, ?)
  ''');

  stmt.execute([foobar.foo, foobar.bar]);
  stmt.dispose();

  return db.lastInsertRowId;
}
```

Read

```
/// READ: Get a FooBar by ID
/// Returns null if not found
Future<FooBar?> read(int id) async {
  final db = await database;

  final stmt = db.prepare(
    'SELECT * FROM $_tableName WHERE id = ?');
  final result = stmt.select([id]);
  stmt.dispose();

  if (result.isEmpty) return null;

  return FooBar.fromRow(result.first);
}
```

read all

```
/// READ: Get all FooBar records
/// Returns a list of all FooBar objects
Future<List<FooBar>> readAll() async {
  final db = await database;

  final ResultSet resultSet =
    db.select('SELECT * FROM $_tableName ORDER BY id');

  return resultSet.map((row) =>
    FooBar.fromRow(row)).toList();
}
```

Find By

```
  /// READ: Find FooBar records by foo field (like a search)
  /// Returns a list of matching FooBar objects
  Future<List<FooBar>> findByFoo(String foo) async {
    final db = await database;

    final stmt = db.prepare(
      'SELECT * FROM $_tableName WHERE foo LIKE ? ORDER BY id');
    final result = stmt.select(['%$foo%']);
    stmt.dispose();

    return result.map((row) => FooBar.fromRow(row)).toList();
  }

}
```

Update

```
/// UPDATE: Update an existing FooBar record
/// Returns true if successful, false if record not found
Future<bool> update(int id, FooBar foobar) async {
  final db = await database;
  final stmt = db.prepare('''
    UPDATE $_tableName
    SET foo = ?, bar = ?
    WHERE id = ?
  ''');
  stmt.execute([foobar.foo, foobar.bar, id]);
  stmt.dispose();
  return db.updatedRows > 0;
}
```

Delete

```
/// DELETE: Remove a FooBar record by ID
/// Returns true if successful, false if record not found
Future<bool> delete(int id) async {
  final db = await database;
  final stmt = db.prepare(
    'DELETE FROM $_tableName WHERE id = ?');
  stmt.execute([id]);
  stmt.dispose();
  return db.updatedRows > 0;
}
```

Delete All

```
/// DELETE: Remove all FooBar records (use with caution!)
/// Returns the number of deleted records
Future<int> deleteAll() async {
  final db = await database;
  db.execute('DELETE FROM $_tableName');
  return db.updatedRows;
}
```

SQL Query Fundamentals

Basic SELECT Queries

```sql
-- Get all students, ordered by ID
SELECT * FROM students ORDER BY id;

-- Get specific student by ID (parameterized)
SELECT * FROM students WHERE id = ?;

-- Get students by major
SELECT * FROM students WHERE major = 'Computer Science';

-- Get students in age range
SELECT * FROM students WHERE age BETWEEN 18 AND 25;
```

```sql
-- Count students by major
SELECT major, COUNT(*) as count
FROM students
GROUP BY major;

-- Get average age
SELECT AVG(age) as average_age FROM students;
```

Advanced Queries

```
-- Search by partial name match
SELECT * FROM students WHERE name LIKE '%John%';

-- Multiple conditions
SELECT * FROM students
WHERE age > 20 AND major = 'Computer Science';

-- Ordering and limiting results
SELECT * FROM students
ORDER BY age DESC, name ASC
LIMIT 10;
```

foobar project

```
sqlite/
├── lib/
│   ├── models/
│   │   └── foobar.dart              # Data model
│   └── services/
│       └── foobar_crud_sqlite.dart  # CRUD operations
├── test/
│   └── foobar_crud_test.dart        # Unit tests
├── data/
│   └── foobar.db                    # SQLite database
└── doc/
    └── crud_tutorial.md             # Documentation
```

Development Workflow

Running the Code

- Run the Application

```
cd /path/to/sqlite/project
dart run lib/main.dart
```

- Run the Tests

```
dart test
```

- Run Specific Test

```
dart test test/foobar_crud_test.dart
```

- Run with Verbose Output

```
dart test --reporter=expanded
```

- Check the Database File

```
# Database is created in data/ directory
ls -la data/
# Explore with SQLite CLI (if installed)
sqlite3 data/foobar.db
```

- You can use the SQLite and SQLite Viewer VSCode extension.

Databases

Feature	PocketBase	Firebase	SQLite	IndexedDB
Type	Server + SQLite	Cloud NoSQL	File-based SQL	Browser NoSQL
Location	Self-hosted	Google Cloud	Local file	Browser storage
Real-time	✅ Built-in	✅ Built-in	❌ None	❌ None
Authentication	✅ Built-in	✅ Complete	❌ Manual	❌ Manual
Scalability	⚠️ Manual	✅ Automatic	❌ Single user	❌ Single user
Queries	✅ REST API	✅ Rich NoSQL	✅ Full SQL	❌ Key-value
Offline	❌ Network only	✅ Smart sync	✅ Always	✅ Always
Cost	🆓 Free hosting	💰 Pay-per-use	🆓 Free	🆓 Free

Decision Framework

- **Choose PocketBase** for: Self-hosted real-time apps, educational projects, MVPs, data control`
- **Choose IndexedDB** for: Browser-only applications, offline-first web apps, client-side caching

> **Choose SQLite** for: Single-user apps, offline-first, embedded applications

- **Choose Firebase** for: Global scale, automatic scaling, rapid development without hosting

SQLite Limitations

Concurrency Limitations:

- Single writer at a time
- Read-heavy workloads perform better
- Not ideal for high-concurrency applications
- Limited network database access

Scale Limitations:

- Database size practical limit (~281 TB theoretical)
- Single database file can become large
- No built-in replication or clustering
- Limited user management features

Feature Limitations:

- No stored procedures or user-defined functions
- Limited data types compared to full SQL databases
- No Complex JOIN
  - No RIGHT OUTER JOIN or FULL OUTER JOIN