DeepWiki emanavas/PadelFlow

Index your code with Devin Share

Database Layer

Relevant source files

Purpose and Scope

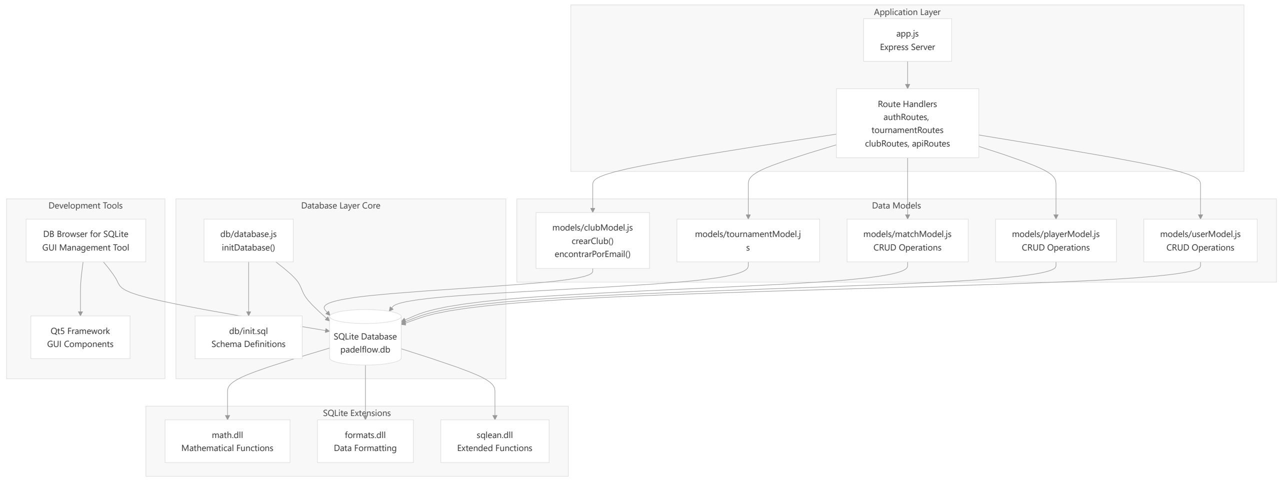
The Database Layer provides persistent data storage and management capabilities for the PadelFlow tournament management platform. This document covers the SQLite database system, its configuration, extensions, and integration with the application layer. It includes the core database setup, data models, and development tools that enable reliable storage and retrieval of tournament, club, player, and match information.

For specific implementation details of database models and CRUD operations, see Core Application Architecture. For development tools and environment setup, see Development Environment.

Database Architecture Overview

PadelFlow uses SQLite as its primary database engine, enhanced with multiple extensions to provide advanced functionality. The database layer is structured around a centralized database instance with modular model files that handle specific entity operations.

Database Architecture Components

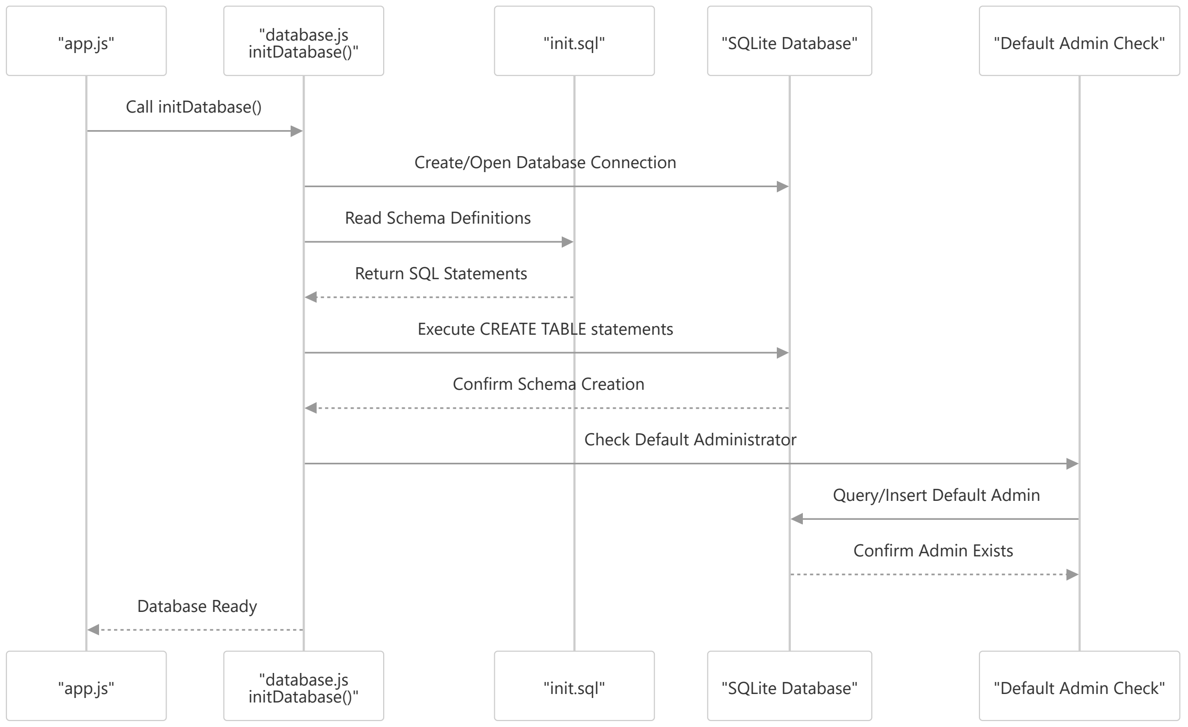


Sources: README.md 84-99 High-level system diagrams

Core Database Components

The database layer is initialized through the `initDatabase()` function which sets up the SQLite connection and executes the schema definitions. The system uses a modular approach where each entity type has its own model file containing specialized database operations.

Database Initialization Flow



The core database components include:

Component	File	Purpose
Database Initialization	`db/database.js`	Connection management and schema setup
Schema Definitions	`db/init.sql`	Table structures and relationships
Club Operations	`models/clubModel.js`	Club creation and management
Tournament Management	`models/tournamentModel.js`	Tournament CRUD operations
Match Handling	`models/matchModel.js`	Match data and score management

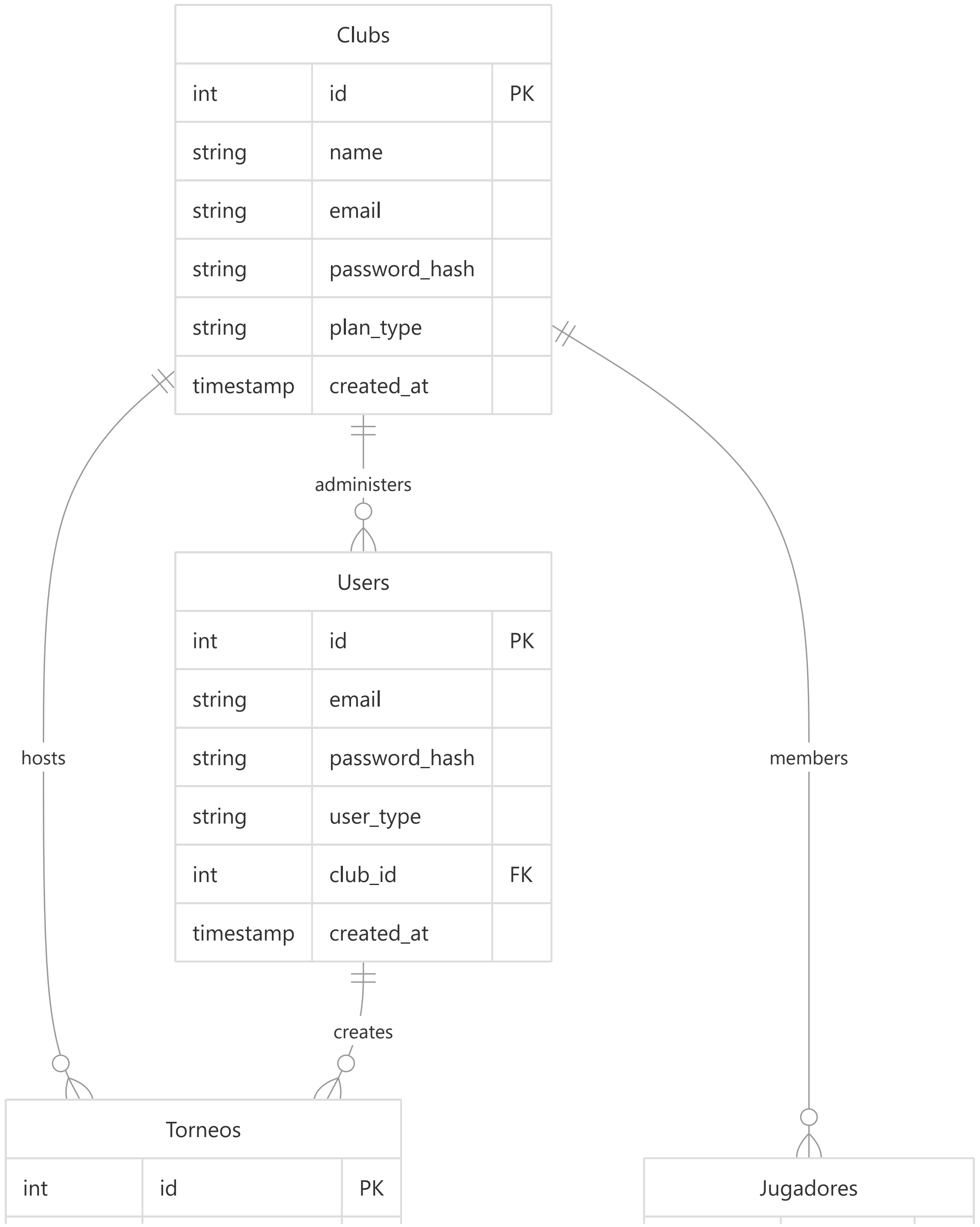
Component	File	Purpose
Player Management	`models/playerModel.js`	Player registration and profiles
User Authentication	`models/userModel.js`	User accounts and role management

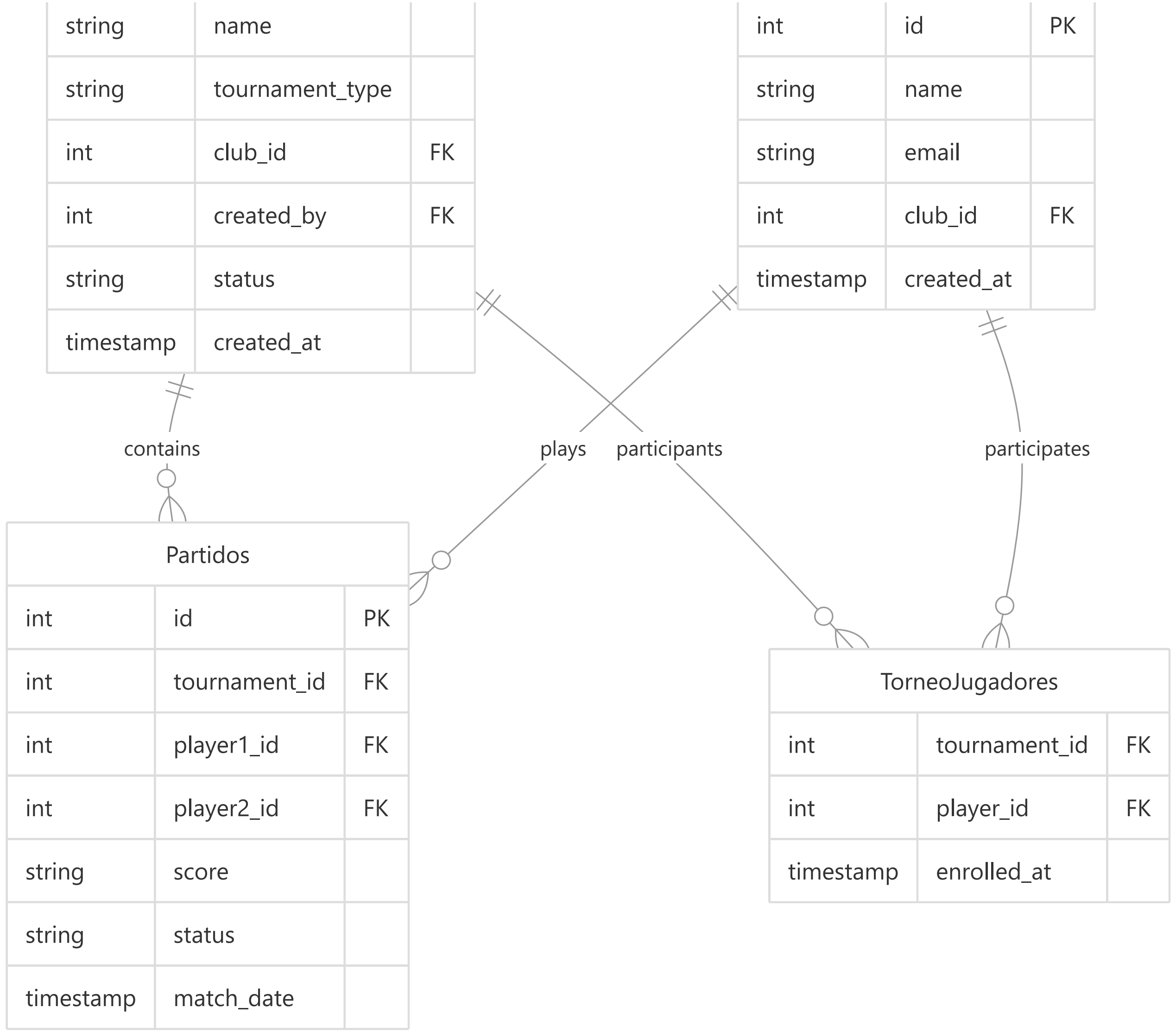
Sources: README.md 84-99 README.md 45-50

Database Schema and Entity Relationships

The PadelFlow database schema supports a multi-tenant architecture with hierarchical user roles and comprehensive tournament management capabilities. The schema defines relationships between clubs, users, tournaments, players, and matches.

Entity Relationship Model


Clubs
int | id | PK
string | name
string | email
string | password_hash
string | plan_type
timestamp | created_at
administers
hosts
members
Users
int | id | PK
string | email
string | password_hash
string | user_type
int | club_id | FK
timestamp | created_at
creates
Torneos
int | id | PK
Jugadores



The schema supports multiple tournament types including:

- **Round Robin (Liguilla)**: All participants play against each other
- **Single Elimination (Eliminatoria Directa)**: Bracket-style elimination
- **League (Liga)**: Season-based competition with multiple rounds

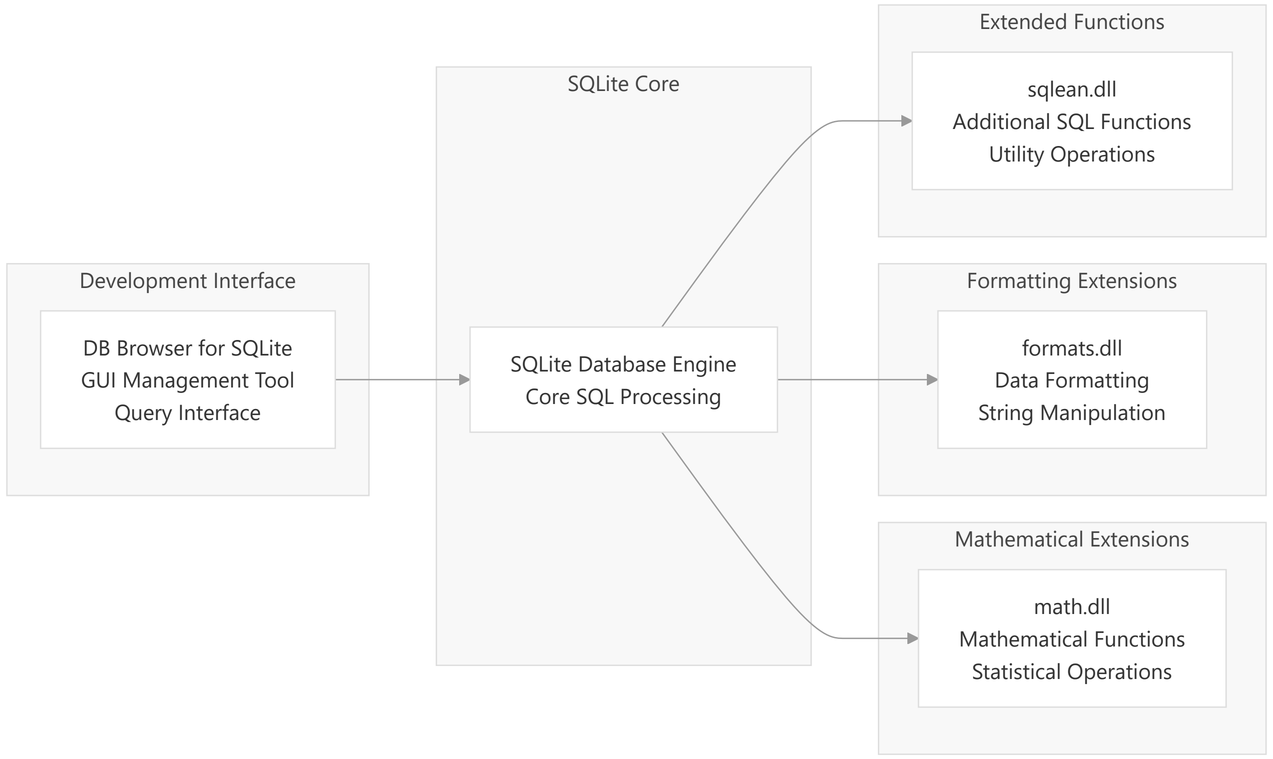
- **Americana Clásica**: Rotating pairs tournament format

Sources: README.md 84 README.md 199-216

Database Extensions and Development Tools

The SQLite database is enhanced with multiple extensions that provide advanced functionality beyond standard SQL operations. These extensions enable mathematical calculations, data formatting, and extended utility functions required for tournament management.

Extension Architecture



The development environment includes DB Browser for SQLite, a Qt-based application that provides:

- Visual database schema management
- Interactive query execution
- Data import/export capabilities
- Real-time database monitoring

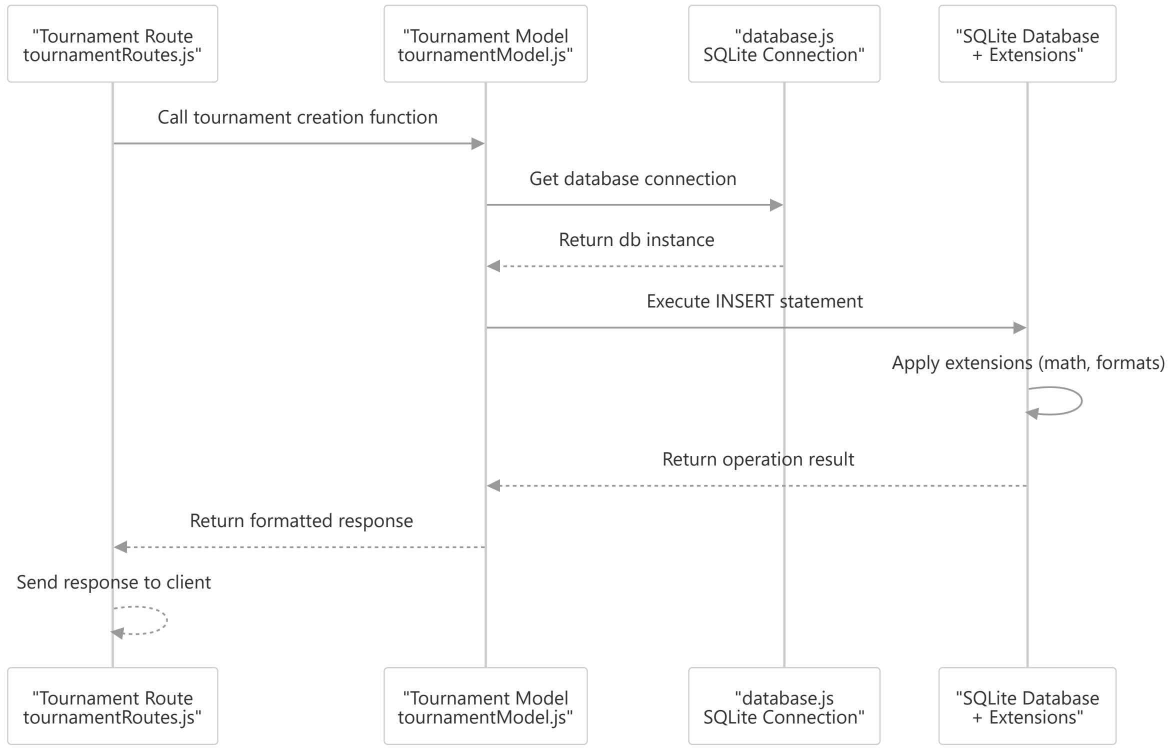
For detailed information about specific extensions and their capabilities, see Database Extensions. For Qt framework components powering the DB Browser, see Qt Framework Components.

Sources: High-level system diagrams, Database extension references

Integration with Application Layer

The database layer integrates seamlessly with the Express.js application through model files that encapsulate database operations. Each model provides a clean API for the route handlers to perform CRUD operations without direct SQL manipulation.

Application-Database Integration Pattern


"Tournament Route tournamentRoutes.js"
"Tournament Model tournamentModel.js"
"database.js SQLite Connection"
"SQLite Database + Extensions"
Call tournament creation function
Get database connection
Return db instance
Execute INSERT statement
Apply extensions (math, formats)
Return operation result
Return formatted response
Send response to client

The integration supports:

- **Session Management**: User sessions stored and validated against database
- **Real-time Updates**: Database changes trigger Server-Sent Events for live updates
- **Transaction Management**: Atomic operations for complex tournament operations
- **Freemium Restrictions**: Database queries enforce plan limitations

Key integration points include the `initDatabase()` function called during application startup and the model files that provide abstracted database operations for business logic implementation.

Sources: README.md 85-90 README.md 100-125