Software requirements specification for project SQLite-like DB

1. Authors

- Sofia Sidorenko, 24215
- Munkuev Vladislav, 24215
- Obraztsov Dmitry, 24214
- Romanenko Nokita, 24214

2. Introduction

This project is about developing our own DB. This DB is SQLite-like. The project is designed for storing, retrieving and modifying data in the form of tables without the need for a separate server.

The main goal is to implement a minimal relational database core with support for:

- an SQL-like query language,
- working with database files on disk,
- basic operations,
- simple indexing.

3. Glossary

1) DBMS - Database management system. A program that stores and manages structured data.
2) Relational model - A data model based on tables (relations) and the relationships between them.
3) SQL - Structured Query Language. This project uses a simplified subset of SQL.

4. Actors

Actor	***Role and goal***
CLI User	A developer or administrator using DB through the console. The purpose is to create tables, insert, and retrieve data.
Embedder Application	A program that uses DB as an embedded library. Its purpose is to execute SQL queries on local data without an external server.

File system	The environment where the database is stored. The system is responsible for storing and accessing database files.

5. Functional requirements

5.1. Strategic Use-cases

5.1.1. UC-S-1: Database Schema Management

Groups operations related to changing the database structure: CREATE TABLE, DROP TABLE, etc.

5.1.2. UC-S-2: Data Manipulation

Groups operations for adding, deleting, modifying, and selecting data: INSERT, UPDATE, DELETE, SELECT.

5.1.3. UC-S-3: Data Storage Management

Groups DB's interactions with the file system: creating, reading, writing, and deleting database files.

5.2. Use-cases for CLI User

5.2.1 Use-case UC-1-1: Create a table

Actors: CLI user

Goals: Create a new table with the specified columns and data types.

Precondition: Database is open.

Trigger condition: Entering the CREATE TABLE command.

Main success scenario:
1. The user enters the SQL command CREATE TABLE.
2. The parser checks the syntax.
3. The table schema is saved in the system table.
4. An empty table structure is created in the database file.

Alternative scenario:
Syntax error → message displayed to the user.

5.2.2. Use-case UC-1-2: Adding a record

Actors: CLI user

Goals: Add a new row to the table.

Precondition: Table exists.

Trigger condition: INSERT INTO command.

Main success scenario:

1. The user enters the INSERT SQL command.
2. The system parses the command.
3. Data is written to the first available free page of the database file.
4. The table index is updated (if any).

Alternative scenario:
Data type violation → error and operation rollback.

5.2.3. Use-case UC-1-3: Select Data

Actors: CLI User

Goals: Retrieve records from a table based on query conditions.

Precondition: The table exists and contains data.

Trigger Condition: The user enters a SELECT command.

Main Success Scenario:

1. The user executes the SELECT query.
2. DB parses the query.
3. Relevant records are read from the database file.
4. The results are printed to the terminal.

Alternative Scenario (Empty result):

DB returns “No rows found.”

5.2.4. Use-case UC-1-4: Updating or Deleting Data

Actors: CLI user

Goals: Modify or delete data in a table.

Trigger condition: UPDATE or DELETE command.

Main success scenario:

1. The user enters a query.
2. The system parses and checks the conditions.
3. Modifies or deletes records.
4. The updated data is saved to disk.

5.3 Use-cases for Application Embedding

5.3.1. Use Case UC-2-1: Executing an SQL Query via the API

Actors: Embedder application

Goals: Execute an SQL query and retrieve the result.

Precondition: DB library is enabled.

Trigger condition: Calling the execute_query(sql) function.

Main success scenario:

1. The application calls execute_query(sql).
2. DB parses and executes the query.
3. The results are returned as structured data.

Alternative scenario:

Syntax error or missing table → error code returned.

5.4 Use-cases for File System

5.4.1. Use-case UC-3-1: Creating a Database File

Actors: File system

Goals: Allow DB to create a new database file.

Precondition: User starts DB with a new database.

Trigger condition: CREATE DATABASE command or first run.

Main success scenario:

1. DB calls the system call to create a file.
2. The file system allocates disk space.
3. Returns a file descriptor.

Alternative scenario:
Access rights error → error message.

5.4.2. Use Case UC-3-2: Reading and Writing Pages

Actors: File System

Goals: Ensure DB can read and write data pages.

Precondition: The database file is open.

Main Success Scenario:

1. DB calls read() to read the page.
2. The file system returns the data.
3. DB modifies the page in memory.
4. DB calls write() to write it back.

Alternative Scenario:
Insufficient disk space → return error and rollback the transaction.

5.4.3. Use-case UC-3-4: Deleting a database file

Actors: File system

Goals: Delete a database file.

Trigger condition: DROP DATABASE command or directory cleanup.

Main success scenario:

1. DB calls unlink() / remove().
2. The file system deletes the file and frees up space.

Alternative scenario:

File is in use by another process → error returned.

6. System-wide functional requirements

Authorization:

- Not required (embedded DBMS).

Journaling:

- Simple journaling for crash recovery.

Storage format:

- Page-based, similar to SQLite (fixed blocks, for example 4 KB).

7. Non-functional requirements

7.1. Environment

Operating systems:

- Linux,
- macOS,
- Windows.

Implementation language: Java.

Libraries: Standard C library, optional others for file manipulation.

7.2. Reliability

In case of a failure, the system must be able to recover the database.

Integrity check on every file open.

7.4. Extensibility

Potential expansion to network mode.