Software requirements specification for project SQLite-like DB

1. Authors

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

2. Introduction

This project is about developing our own DB. This DB is SQLite-like. SQL is a declarative language with a context-sensitive grammar. The parser parses the query and provides the necessary recursive parsing classes in an imperative manner. The driver core sequentially executes the steps to implement the query, providing an internal abstraction of query execution.

The project is designed for storing, retrieving and modifying data in the form of tables without the need for a separate server.

For our project we chose to work with Java. It was chosen because it:

- provides platform independence thanks to the JVM;
- supports reliable memory and exception management, which is important for stable operation of the file system and transactions;
- suits well for modular architecture (splitting into different files);
- has a high level of code security and readability, which simplifies project maintenance and development.

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.

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

In our project there is only one actor. It is CLI User, who is a developer or administrator using DB through the console. His purpose is to create tables, insert, and retrieve data.

5. Functional requirements

5.1 Use-cases for CLI User

Use-Case 5.1.1: Creating the DB structure

Actors: CLI User

Goals: Create a new DB or its structural components.

Preconditions: The system environment is installed and running correctly.

Trigger condition: The actor enters a valid DDL query (CREATE, DROP, etc.)

Main success scenario:

1. The actor inputs a valid query.
2. The system parses the query and returns a query object.
3. The query object is passed to the core driver.
4. The core driver creates the internal representation of the query.
5. The driver passes control to the file manager.
6. The file manager creates a minimal transaction log (except for SELECT).
7. The file manager creates the required file structure.
8. On success, control is returned to the parser, which waits for the next query.

Alternative scenario A — Invalid query:

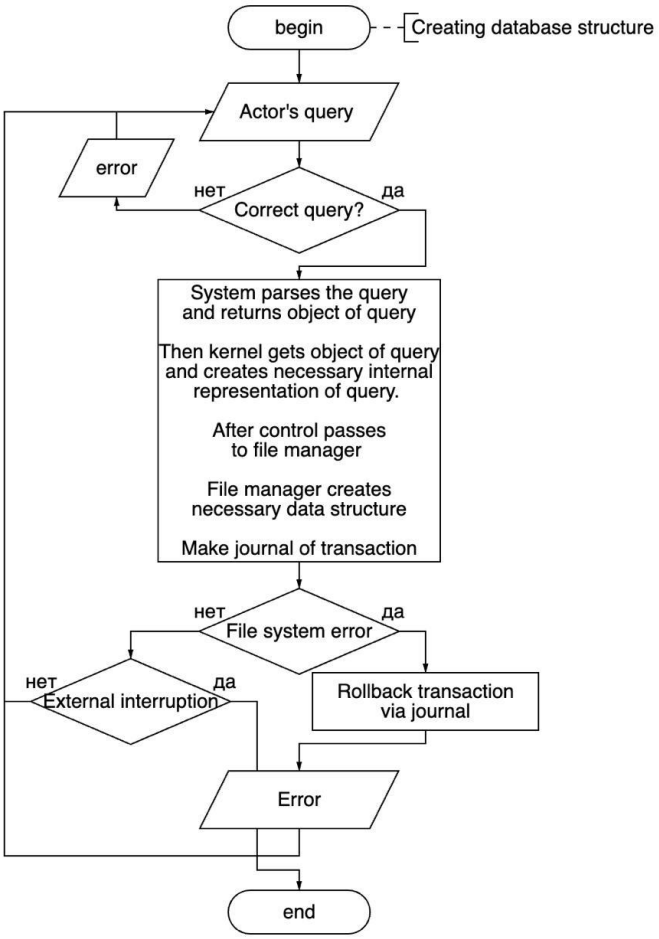
Trigger: The actor enters an invalid query.

1. The parser detects a syntax or semantic error.
2. The parser displays an error message.
3. The system continues working normally.

Alternative scenario B — File system error:

Trigger: A failure occurs during file system operations.

1. The transaction is rolled back using the log.
2. The error is displayed.
3. The system continues working normally.



Use-Case 5.1.2: Modifying the DB structure

Actors: CLI User

Goals: Modify the DB structure or data using valid queries.

Preconditions: The database exists and its file representation is valid.

Trigger condition: The actor enters a valid DML or DDL query (ADD, INSERT, UPDATE, ALTER, etc.)

Main success scenario:

1. The actor inputs a valid query.
2. The system parses the query and returns a query object.
3. The query object is passed to the core driver.
4. The core driver builds an internal representation of the query.
5. Control is passed to the file manager.

6. The file manager creates a minimal transaction log (except for SELECT).
7. The file manager updates the file structure accordingly.
8. On success, control is returned to the parser to await the next query.

Alternative scenario A — Invalid query:

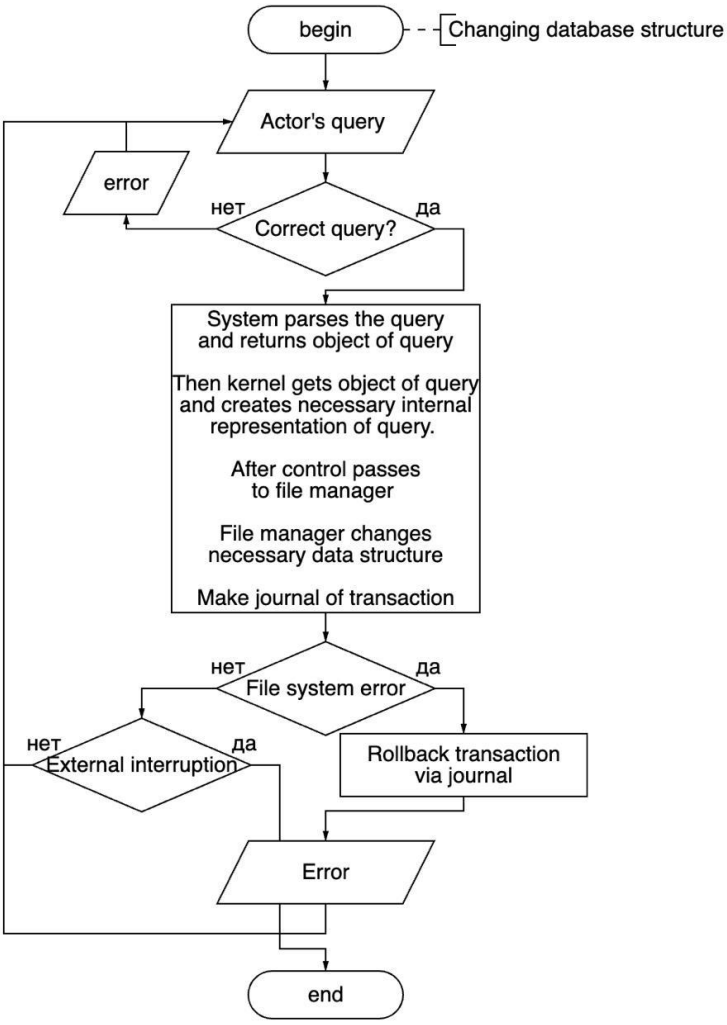
Trigger: Invalid query entered.

1. The parser detects an error and displays it.
2. The system continues normal operation.

Alternative scenario B — File system error:

Trigger: File system operation fails.

1. The transaction is rolled back via the log.
2. The error message is shown.
3. The system continues normal operation.



Use-case 5.1.3: Displaying information about the DB structure

Actors: CLI User

Goals: Retrieve and display information about the current DB structure.

Preconditions: The DB exists and its file representation is valid.

Trigger condition: The actor enters a valid DQL query.

Main success scenario:

1. The actor inputs a valid query.
2. The system parses the query and returns a query object.
3. The query object is passed to the core driver.
4. The driver creates the internal representation of the query.

5. The driver passes control to the file manager.
6. The file manager retrieves the requested information.
7. The data is passed to the output module.
8. The user receives the displayed information.

Alternative scenario A — Invalid query:

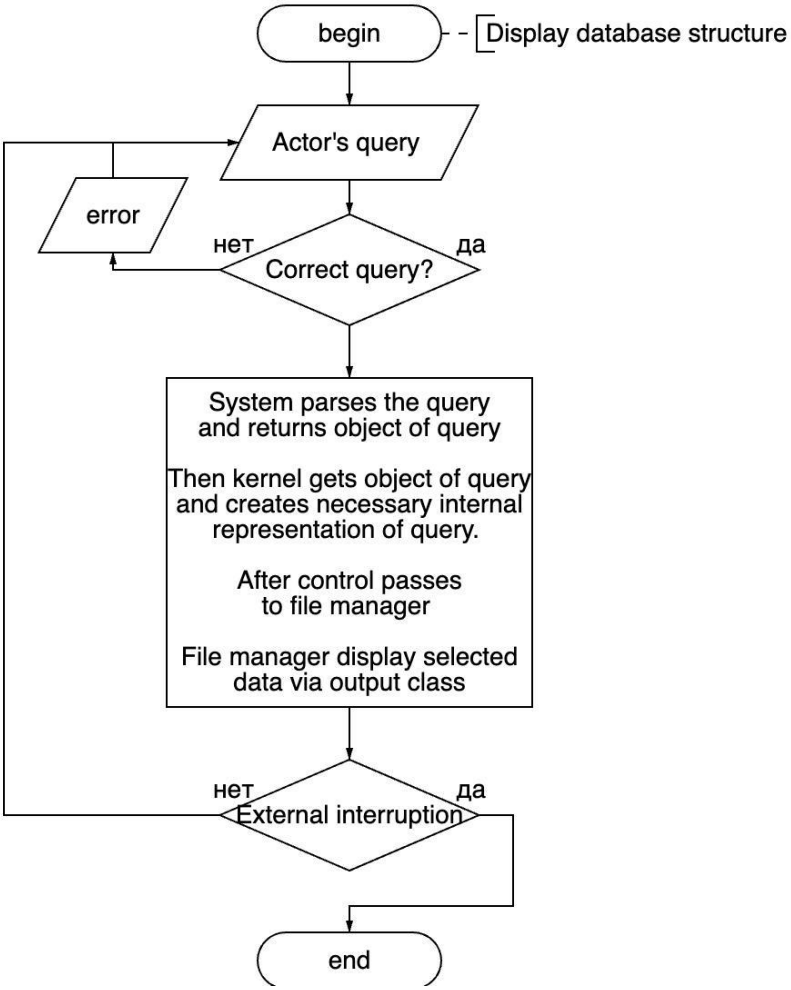
Trigger: Invalid query entered.

1. The parser detects an error and displays it.
2. The system continues normal operation.

Alternative scenario B — External interruption:

Trigger: External interruption occurs.

1. The system terminates critically.



Use-Case 5.1.4: Rolling back DB structure.

Actors: CLI User

Goals: Restore the database structure to a previous consistent state using transaction control commands.

Preconditions: The transaction log and file structures are consistent and available.

Trigger condition: The actor enters valid TCL queries (BEGIN TRANSACTION, COMMIT, ROLLBACK, etc.)

Main success scenario:

1. The actor inputs a valid query.
2. The system parses the query and returns a query object.
3. The query object is passed to the core driver.
4. The driver creates an internal representation of the query.
5. Control is passed to the file manager.
6. The file manager uses the journal to roll back or commit changes.
7. On success, control is returned to the parser.
8. The system waits for the next query.

Alternative scenario A — Invalid query:

Trigger: Invalid query entered.

1. The parser detects an error and reports it.
2. The system continues normal operation.

Alternative scenario B — File system error:

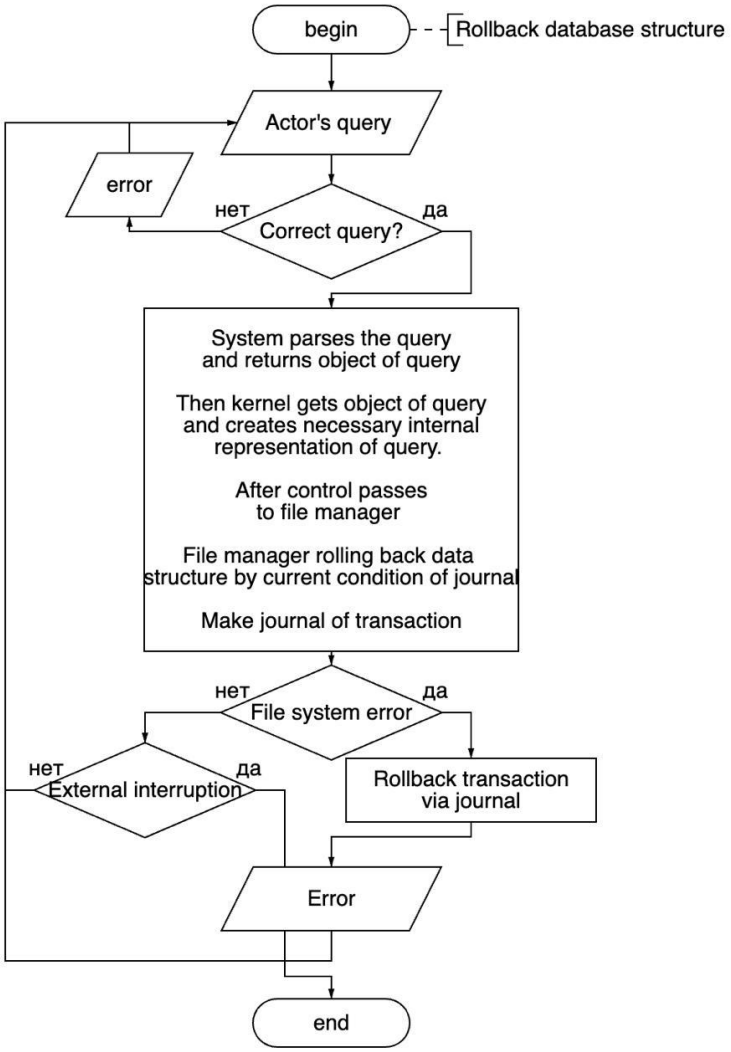
Trigger: Rollback operation fails due to file system error.

1. The system notifies the user of rollback failure.
2. The system continues operating.

Alternative scenario C — External interruption:

Trigger: External interruption occurs.

1. The system terminates critically.



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

Our project is used as a static library. CLI User runs it in a console without a graphical interface. The system parses queries and executes them.

Operating systems required:

- Linux Ubuntu (ver. 24.04+),
- MacOS (ver. 13+)
- Windows (ver. 10+)

Implementation language: Java (ver. 17+)

7.2. Reliability

Recovery is implemented using simple logging. We will log every request except SELECTs.

7.3. Extensibility

Potential expansion to network mode.