SE 3XA3: Software Requirements Specification
GoDBMS

Team #7, Databased
Eesha Qureshi, qureshe
Faiq Ahmed, ahmedf46
Kevin Kannammalil, kannammk

February 11, 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Feb 8, 2022	1.0	Initial Document
Feb 10, 2022	1.1	Finished Functional Requirements and Use Cases
Feb 11, 2022	1.2	Completed Document

This document describes the requirements for the project **GoDBMS** The template for the Software Requirements Specification (SRS) is a subset of the Volere template.

1 Project Drivers

1.1 The Purpose of the Project

The purpose of this project is to build on and improve SimpleDB, to accurately demonstrate how a **DBMS** works by implementing a minimalist **DBMS** while also maintaining the ability for the **system** to perform multiple tasks at once.

1.2 The Stakeholders

1.2.1 The Client

The clients for this project are the instructional staff for SFWRENG 3XA3, as they have requested the development of this software. The staff consists of Dr. Asghar Bokhari, the professor, and his teaching assistants Veerash Palanichamy and Oluwaseun Owojaiyo. The clients will oversee the schedule for deliverables, and meet with the developers weekly to provide input and assistance as well as next steps. They will also verify that the product adheres to the requirements outlined in this document.

1.2.2 The Customers

The customers for this product are developers or students who are interested in learning about the fundamentals of **DBMS** and their implementation. Additionally, customers also include those who wish to use a lightweight **DBMS** due to storage costs, memory constraints or any other reason.

1.2.3 Other Stakeholders

Other stakeholders include the developers and quality assurance testers of the project. This consists of Eesha Qureshi, Faiq Ahmed and Kevin Kannammalil, as they will be writing, maintaining, testing and deploying all the source code for this project. Their software engineering skills are one of the resources invested in the project and it's successful development is in their best interest. Additionally, the owner of the original repository that this project is based on is also a stakeholder as they have left the code openly sourced in the interest of other developers building on or recreating it.

1.3 Mandated Constraints

Users of **GoDBMS** require MacOS, Windows or Linux on their machine.

1.4 Naming Conventions and Terminology

- DBMS: database management system
- Concurrency/Concurrent: multiple tasks running at the same
- SQL: structured query language, used to describe tasks to perform in a database
- Query/Queries: a descriptive data request, usually written in SQL
- Transaction: a specific job that the database management system is instructed to perform
- Table: a defined structure in a database that data can be inserted into. The inserted data must abide by this structure
- System: in this case the system refers to our project and the overarching database management system
- Record: a specific entry of values in a database table
- Field: a column of a table in the database
- Schema: The structure of a table in the database, this includes all fields and constraints of the table

1.5 Relevant Facts and Assumptions

1.5.1 Assumptions

- Users have access to a computer and necessary hardware such as keypad and mouse
- Users have basic knowledge of SQL and queries
- Users have working English proficiency as other languages are not supported
- User has at least $STORAGE\_SIZE$ space available on PC

2 Functional Requirements

2.1 The Scope of the Work and the Product

The **DBMS** being built is a simple **concurrent** database that can be used by inputting **SQL queries** from the command line. This **DBMS** will be implemented in a more efficient manner as compared to the original project by adding **concurrency** and the ability to

handle multiple **transactions** at the same time. The project will focus on adding support for only a few primitive data types such as strings and integers and having a very basic **SQL** parser to convert the user inputs into commands the database can execute. This **SQL** parser will only support basic insert, delete and select commands with functionalities such as nested **queries** stripped.

2.1.1 The Context of the Work

The context of this database management **system** is going to be a command line application running on Windows, Linux, or MacOS. It will use the Go compiler to compile the program to a binary so it can natively run on the appropriate operating **system**. The user will directly interact with this application use the command line and it does not interact with other **system**s aside form the operating **system**.

2.1.2 Work Partitioning

Event Name	Input/Output	Summary
1. Create a new **table**	**Input: SQL query** **Output:** Display the newly created **table** or errors	The system uses the **query** to create a **table** and display it back to the user. Alternatively error messages are displayed if the **query** is incorrect.
2. Delete a **table**	**Input: SQL query** **Output:** Success message or errors	The system uses the **query** to delete a **table** and displays a message indicating success or an error.
3. List all **tables**	**Input: SQL query** **Output:** A text list of all **table** names or error message	The system uses the **query** to display all stored **tables**. Alternatively error messages are displayed if the **query** is incorrect.
4. Add **record** to **table**	**Input: SQL query** **Output:** Success message or errors	The system uses the **query** to store a **record** into a specified **table**. Alternatively error messages are displayed if the **query** is incorrect.
5. Modify a **record**	**Input: SQL query** **Output:** A display of the modified **record** or errors	The system uses the **query** to modify a specific **record** in a **table** and displays it back to the user. Alternatively error messages are displayed if the **query** is incorrect.
6. Delete a **record**	**Input: SQL query** **Output:** Success message or errors	The system uses the **query** to delete a **record** and displays a message indicating success or an error.
7. Search for **record**s	**Input: SQL query** **Output:** A display of the resulting **record**s or errors	The system uses the **query** to find and displays all **record**s that fit the search criteria. Alternatively error messages are displayed if the **query** is incorrect.
8. Process multiple **transactions**	**Input: SQL queries** **Output:** All **query** outputs	The system takes in multiple **query** inputs and is able to process and return the correct output for all of them at the same time.

Table 2: Work Partitioning Events and Summaries

2.1.3 Individual Product Use Cases

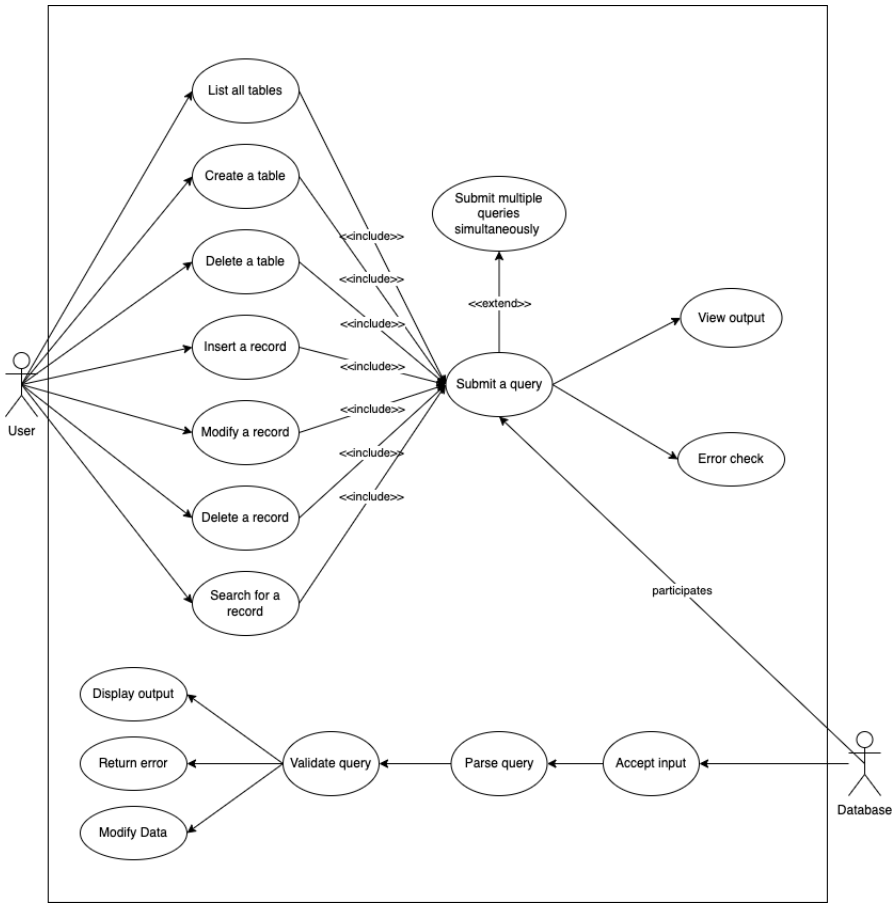

Figure 1: Individual Use Cases

2.2 Functional Requirements

BE1. The user wants to create a **table**

- FR1. The system must accept a command line input from the user
- FR2. The system must validate the syntax of the input to create a **table**
- FR3. The system must parse the input to identify the necessary **field**s
- FR4. The system must validate that the user provided a valid name for the **table** that does not already exist
- FR5. The system must validate that the user provided valid information about the **schema** of the **table**
- FR6. The system must display any errors in the input to the user
- FR7. The system must create a new **table** and store the **table**

FR8. If the **table** is successfully created the system must display a message to the user to inform them

BE2. The user wants to delete a **table**

FR1. The system must accept a command line input from the user

FR2. The system must validate the syntax of the input to delete a **table**

FR3. The system must parse the input to identify the necessary **field**s

FR4. The system must validate that the user provided a valid name for the **table** and the **table** exists

FR5. The system must display any errors in the input to the user

FR6. The system must create a new delete the **table** and remove it from storage

FR7. If the **table** is successfully deleted the system must display a message to the user to inform them

BE3. The user wants to list all created **tables**

FR1. The **system** must accept a command line input from the user

FR2. The **system** must validate the syntax of the input to list all **tables**

FR3. The **system** must parse the input to identify the necessary **field**s

FR4. The **system** must display any errors in the input to the user

FR5. If the input is valid the **system** must display all stored **tables** and their **schema**

BE4. The user wants to insert a **record** into a **table**

FR1. The **system** must accept a command line input from the user

FR2. The **system** must validate the syntax of the input to insert a **record**

FR3. The **system** must parse the input to identify the necessary **field**s

FR4. The **system** must validate that the user provided a valid and existing **table** name to insert the **record** into

FR5. The **system** must validate that the user provided a **record** that fits the **schema** of the **table** it is being inserted into

FR6. The **system** must display any errors in the input to the user

FR7. The **system** must create the new **record** in the **table** and add it to storage

FR8. If the **record** is successfully inserted the **system** must display a message to the user to inform them

BE5. The user wants to modify a **record** inserted into a **table**

FR1. The **system** must accept a command line input from the user

FR2. The **system** must validate the syntax of the input to modify the **record**

FR3. The **system** must parse the input to identify the necessary **field**s

FR4. The **system** must validate that the user provided a valid and existing **table** name to modify the **record** from

FR5. The **system** must validate that the user provided an existing, and uniquely identifiable **record** from the **table** to modify

FR6. The **system** must validate that the modified **record** fits the **schema** of the **table** it is being inserted into

FR7. The **system** must display any errors in the input to the user

FR8. The **system** must modify the **record** in the **table** and update it in storage

FR9. If the **record** is successfully updated the **system** must display a message to the user to inform them

BE6. The user wants to delete **record**s inserted into a **table**

FR1. The **system** must accept a command line input from the user

FR2. The **system** must validate the syntax of the input to delete the **record**s

FR3. The **system** must parse the input to identify the necessary **field**s

FR4. The **system** must validate that the user provided a valid and existing **table** name to delete the **record**s from

FR5. The **system** must display any errors in the input to the user

FR6. If any **record**s are deleted, the **system** must delete the **record**s from the **table** and remove them from storage

FR7. The **system** must display all deleted **record**s to the user

BE7. The user wants to search for **record**s inserted into **tables**

FR1. The **system** must accept a command line input from the user

FR2. The **system** must validate the syntax of the input to search for **record**s

FR3. The **system** must parse the input to identify the necessary **field**s

FR4. The **system** must validate that the user provided valid and existing **table** names to search the **record**s from

FR5. The **system** must display any errors in the input to the user

FR6. The **system** must display all **record** from the specified **tables** that fit the user specified search criteria

BE8. The users want to process multiple **transactions** at the same time

FR1. The **system** must accept and validate all inputs from the user

FR2. If a **transaction** is writing to a **table** or record, the **system** must stop all **transactions** from reading from that **table** or record until the write is complete

FR3. The **system** must allow multiple **transactions** to read from the same **table** or record at the same time

FR4. The **system** must allow multiple **transaction** write to different **tables** or records at the same time

FR5. The **system** must allow writes and reads from different **tables** or records at the same time

3 Non-functional Requirements

3.1 Look and Feel Requirements

3.1.1 Appearance Requirements

1. The **system** shall use a simple and minimal design
2. The **system** should add *OUTPUT_PADDING* of space after each output to easily distinguish different outputs.

3.1.2 Style Requirements

1. The **system** shall use the pre-existing styling in the command line terminal

3.2 Usability and Humanity Requirements

3.2.1 Ease of Use Requirements

1. The **system** shall be easy to use for anyone with a basic understanding of a database

3.2.2 Personalization and Internationalization Requirements

1. The **system** shall use English as the main language to communicate with the user

3.2.3 Learning Requirements

1. The **system** shall be straightforward to use for beginners
2. The **system** shall provide descriptive instructions to guide the user

3.2.4 Understandability and Politeness Requirements

N/A

3.2.5 Accessibility Requirements

1. The **system** shall use the accessibility features already existing in the command line

3.3 Performance Requirements

3.3.1 Speed and Latency Requirements

1. The **system** shall provide a response to the user in a reasonable time depending on the input. On average the **transactions** should respond within *RESPONSE_TIME*.

3.3.2 Safety-Critical Requirements

N/A

3.3.3 Precision or Accuracy Requirements

1. The **system** shall not round any of the outputs to preserve the accuracy of the data

3.3.4 Reliability and Availability Requirements

N/A

3.3.5 Robustness or Fault-Tolerance Requirements

N/A

3.3.6 Capacity Requirements

1. The **system** will store all the data locally on the user's device

3.3.7 Scalability or Extensibility Requirements

1. The **system** can be vertically scaled with the hardware of the environment it is running in

3.3.8 Longevity Requirements

1. The **system** shall be developed to be easily maintainable in the long term

3.4 Operational and Environmental Requirements

3.4.1 Expected Physical Environment

1. The **system** is not dependent on an internet connection to work
2. The **system** should be used on a computer that can run the command line

3.4.2 Requirements for Interfacing with Adjacent Systems

N/A

3.4.3 Productization Requirements

N/A

3.4.4 Release Requirements

1. The **system** will have it's final iteration and release on April 12th 2021

3.5 Maintainability and Support Requirements

3.5.1 Maintenance Requirements

1. The **system** will be fully documented on how to use it
2. The **system** will have diagrams regarding the architecture and code
3. The **system** will use consistent styling

3.5.2 Supportability Requirements

1. The **system** will be open source
2. The **system** is supported to run on Windows, MacOS and Linux devices

3.5.3 Adaptability Requirements

N/A

3.6 Security Requirements

3.6.1 Access Requirements

1. The **system**'s code will only be view-able to the public through the git repository.
2. The **system** restricts it's editing privileges for the code only to the maintainers.

3.6.2 Integrity Requirements

N/A

3.6.3 Privacy Requirements

1. The **system** shall not collect any personal information of the users
2. The **system** will not require the user to create an account

3.6.4 Audit Requirements

N/A

3.6.5 Immunity Requirements

N/A

3.7 Cultural Requirements

N/A

3.8 Legal Requirements

3.8.1 Compliance Requirements

1. The **system** is not accountable for the legality of the data that is being stored
2. The **system** is not responsible for the loss of any data
3. The **system** is meant for learning purposes only and not for use in production
4. The **system** will not use any copyrighted information

3.8.2 Standards Requirements

1. The **system** shall follow the IEEE standards

3.9 Health and Safety Requirements

1. The **system** will not have any health and safety concerns for users

This section is not in the original Volere template, but health and safety are issues that should be considered for every engineering project.

4 Project Issues

4.1 Open Issues

There are no open issue in the actual SimpleDB repository and the last commit was in 2019. The original implementation is correct and functional. Since the original implementation is written in Java, it requires the Java Runtime Environment to function. Thus, only hardware that has the JRE installed can run the original implementation of this project.

4.2 Off-the-Shelf Solutions

There are various off the shelf solutions to a **DBMS** that are well documented, correct, easy to use, and open source. However, to improve productivity and functionality of the software, many additional features such as support for multiple different data types and custom functions tend to be added into these off-the-shelf solutions. On the other hand many of existing simplistic **DBMS** tend to not include features such as **concurrency** or **SQL** parsing in

them.

Though the code from the original repository can not be copied since we are implementing our **system** in a different language, many of the concepts and implementations can be referenced in our project.

4.3 New Problems

Since the project will not be compiled to a binary using Go compiler, the **system** would no longer need the Java Runtime Environment for the application to function as compared to the original project. However to be able to initially install the project, the Go compiler would need to be installed on the environment. It may be difficult for the user to install and compile from source code on their own.

4.4 Tasks

The tasks and schedule for this project can be found in our Gantt Chart

4.5 Migration to the New Product

Not applicable, since GoDBMS will be built from scratch and independent of the original project.

4.6 Risks

There are very few risks with this project. It runs independently on an operating **system** and is only meant to be used as learning tool rather than a production database. This project may have risks of using up excessive amounts of resources on the operating **system** which may effect the performance of the operating system. These risks can be minimized by testing and ensuring the correctness of the code.

4.7 Costs

There are no monetary costs associated with this project. All software used in the development and installation of this project is open source and free to use.

4.8 User Documentation and Training

Very minimal documentation is required for this project. Since we are removing certain features from the **SQL** parsing though, documentation should be provided to the user through the command line interface on which **SQL** features are available in this **DBMS**.

The user will need to be trained in the basics of **SQL** to be able to use this **DBMS**. There are various existing resources to learn about **SQL** already available. The user can given a link to the W3 SQL Tutorial to be trained in **SQL**.

4.9 Waiting Room

The following features can be added in future release to improve the functionality of the **DBMS** while maintaining its simplicity.

FR1. Nested **SQL queries**

FR2. Graphical User Interface to view and modify the database

4.10 Ideas for Solutions

A simple shell script or batch script can be included to assist users in the installation of this project, which includes installing the Go compiler and compiling the project to a binary.

5 Appendix

This section has been added to the Volere template. This is where you can place additional information.

5.1 Symbolic Parameters

The definition of the requirements will likely call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

$OUTPUT\_PADDING$ = 20px
$RESPONSE\_TIME$ = 5 seconds
$STORAGE\_SIZE$ = 1 GB