YourSQL: A relational database tool for pedagogical use

Emmeleia Arakleitou

A thesis presented for the degree of BSc

Computer Science (Hons)



School of Computing and communications

University of Lancaster

March 2019

Declaration

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all the associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School’s use of plagiarism detection system in order to check the integrity of assessed work.

I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date:

Signed:

Table of Contents

[Chapter 1 6](#_Toc2460733)

[Chapter 2 7](#_Toc2460734)

[2.1 SQL Language 7](#_Toc2460735)

[2.2 About SQLite 7](#_Toc2460736)

[2.3 SQLite Architecture 8](#_Toc2460737)

[2.3.1 SQLite Compiler 8](#_Toc2460738)

[2.3.2 Virtual database engine 8](#_Toc2460739)

[2.4 Uses of SQLite 8](#_Toc2460740)

[2.5 Relational Data Model 9](#_Toc2460741)

[2.5.1 Relations and attributes 9](#_Toc2460742)

[2.5.2 Relational Database Schema 10](#_Toc2460743)

[2.6 Relation Algebra 10](#_Toc2460744)

[2.6.1 Relational Algebraic operations 11](#_Toc2460745)

[2.7 Similar projects 12](#_Toc2460746)

[2.7.1 RALT – Relational Algebra Learning Tool 12](#_Toc2460747)

[2.7.2 Query Visualizer – QV 14](#_Toc2460748)

[2.8 Development Environment 15](#_Toc2460749)

[2.8.1 ANTLR – Another Tool For Language Recognition 15](#_Toc2460750)

[2.8.2 Java 16](#_Toc2460751)

[2.8.3 GitHub 16](#_Toc2460752)

[Chapter 3 17](#_Toc2460753)

[3.1 Requirements 17](#_Toc2460754)

[3.1.1 General Requirements 17](#_Toc2460755)

[3.1.2 Database Requirements 18](#_Toc2460756)

[3.1.3 Query Execution Requirements 18](#_Toc2460757)

[3.1.4 Query File Requirements 19](#_Toc2460758)

[3.1.5 Graphical User Interface Requirements 19](#_Toc2460759)

[3.1.6 Error Checking Requirements 21](#_Toc2460760)

[3.2 Use case Diagram 21](#_Toc2460761)

[3.3 Main Sub-System decomposition 22](#_Toc2460762)

Table of Figures

[Figure 2.1 Relation Students 10](#_Toc2441449)

[Figure 2.2 Relational algebra operations 12](#_Toc2441450)

[Figure 2.3 RALT Interface. This image is taken from Mitra (2009) 13](#_Toc2441451)

[Figure 2.4 Query visualization User Interface. This image is taken from Constantinou (2010) 15](#_Toc2441452)

[Figure 3.2 Use case diagram for YourSQL 21](#_Toc2441453)

List of tables

[Table 3.1 System’s General Requirements 17](#_Toc2442497)

[Table 3.2 System’s Database Requirements 18](#_Toc2442498)

[Table 3.3 System’s Query Execution Requirements 19](#_Toc2442499)

[Table 3.4 System’s Query File Requirements 19](#_Toc2442500)

[Table 3.5 System’s GUI requirements 20](#_Toc2442501)

[Table 3.6 System’s Error Checking Rewuiremtns 21](#_Toc2442502)

# 

Introduction

# 

Background

We begin this chapter with a small overview of the SQL language and the relational database management system (RDBMS), SQLite which is been used in the system to manage the databases. Also, we discuss the architecture of SQLite and how a statement is executed. Furthermore, we will be reviewing the concepts of Relational Algebra (RA) and the relational operations. In the next section, we will evaluate RALT and the Query Visualizer, two systems for teaching Relational Algebra and finally, the platforms and tools used to develop the relational database tool will be described in the last section.

## 2.1 SQL Language

SQL is the primary computer language responsible for data manipulation and relational database management. SQL stands for Structured Query Language and is one of the most popular languages used in the computer industry nowadays. The language was designed by Donald D. Chamberlin and Raymond F. Boyce, especially for communicating with databases and therefore it is deliberately made of a relatively small set of keywords and commands (Forta, 2013). All the interactions within a relational database are executed through SQL commands, as it provides a simple way of reading and writing data from a database, commands for table configurations, indexes and other data structures that can be found in the database. Despite the simplicity of the commands, SQL is a very powerful tool, as it allows application programmers with the appropriate knowledge of the SQL language to perform very complex and sophisticated operations.

SQL is neither a stand-alone system nor a Database Management System (DBMS) its self, but it constitutes an integral part of almost every known DBMS, for managing the data in the relations that compose a database (Forta, 2013). For the implementation of this project and the execution of the SQL commands provided by the user, SQLite an already existing DBMS will be used to establish a connection between the database and the system.

## 2.2 About SQLite

In the simplest terms, SQLite is an Open Source, embedded relational DBMS[[1]](#footnote-1), that is contained in a C programming library and implements a small, fast, self-contained, high-reliability and full-featured SQL database engine (Kreibich, 2010).By the term embedded we mean that SQLite is not a stand-alone client/server database engine, instead it coexists into the application it servers. D. Richard Hipp is the developer of SQLite, and 18 years ago, in August 2000 SQLite 1.0 version was released (Allen & Owens, 2010) and in the later years 297 different versions in total were released (Leifer, 2019). The word “Lite” in SQLite, it characterizes the capabilities of the relational DBMS to provide a broad range of features with minimal costs in terms of complexity and resources, but it does not describe its potentials. Contrariwise, the word “Lite” as mentioned in (Kreibich, 2010) refers to the lightweight affect of SQLite in the respect of setup, complexity, network configuration and administrative overhead and resource usage.

## 2.3 SQLite Architecture

SQLite functionality is built into eight modules grouped into three major subsystems the Core, the Compiler and the Backend. In the following subsections the functionality of the Compiler and the Virtual database engine will be discussed in more details to comprehend the procedure that SQLite goes through when executing a statement (Allen & Owens, 2010).

### 2.3.1 SQLite Compiler

The compiler is composed of three modules that collaborate to compile the SQL statement from the text format into a data structure that the lower lever layers can manipulate. A Tokenizer is used to break the statement down into tokens that are then passed to the parser for a syntax and semantic evaluation. Furthermore, the parser assembles the tokens into a parser tree that is then passed to the code generator which is the last component of the compiler. Finally, the code generator analyses the tree and translates it into bytecode that will be used from the Virtual Machine to execute the statement (Allen & Owens, 2010).

### 2.3.2 Virtual database engine

It is considered as the heart of SQLite and it is designed for manipulating and processing the data stored in the database. It is a register-based subsystem that operates on bytecode. The bytecode generated is it a set of instructions that consists of various commands also known as opcodes. These instructions will be later executed to either perform a database operation or manipulate the data for the preparation of an operation (Allen & Owens, 2010).

## 2.4 Uses of SQLite

Due to the simplicity that SQLite provides, it is portable into any environment and this is one the major reasons that it became so popular. It can be easily used in a web environment to assist in managing complicated session information, in embedded devices such as televisions, game consoles, cameras etc. SQLite is the most widely deployed SQL database engine in the world as stated in Kreibich (2010). Although we cannot explicitly know the number of SQLite databases due to the lack of license agreement or disclosure requirement, there are several well-known users.

It is used in Safari web browser and it is the primary meta-data storage format for the Firefox web browser and the Thunderbird Email Reader from Mozilla (Leifer, 2019). Additionally, SQLite as an embedded system it does not require external server to access the database files, and consistently it does not require any administration. This is one of the factors that SQLite has become extremely popular in embedded platforms. In fact, Google’s Android mobile phone’s operating system and Chrome browser has adopted SQLite. Finally, Apple has made extensive use of SQLite in many native applications running on Mac OS-X desktops and servers (Allen & Owens, 2010).

## 2.5 Relational Data Model

Relational data model is the dominant data model used widely around word for data storage and processing. It was introduced by the English Computer Scientist Edgar F. Codd in 1970 (at IBM) (Stanczyk, Champion, & Leyton, 2001) . The relational model consists of three components parts; the “Structure”, which is a simple uniform data structure type named “Relation”, the “Manipulators” which are a set of specific operations that can transform a relation into another relation and finally the “Behaviours” which are a set of integrity constraints that imposes consistency between the relations of a database (Stanczyk, Champion, & Leyton, 2001). Relational data model uses a simple data structure, enabling users with different levels of expertise to use it.

### 2.5.1 Relations and attributes

A relation is a set of tuples, and each tuple consists of multiple fields named attributes. As defined by the mathematical definition, a set is a collection of distinct unordered objects (Stoll, 1963), therefore each tuple in a relation must be uniquely identified by one or more attributes. A tabular form it is commonly used to represent a relation. Figure2.1 is an example of a relation named Students. Each row in Students is a distinct tuple that represents a group of related data values and each column represents an attribute. Attributes are the properties that define a relation, and in the case of the Student relation, the first and last name, the age, the student and course id are the properties that are needed to define a student. Each one of these attributes draws their values from a specific domain, also known as data type that describe the values that can appear under each attribute. For example, the domain for attribute “Age” is the set of values {20,21,19,22} (Stanczyk, Champion, & Leyton, 2001).

As already mention above, all tuples in a relation must be uniquely identified by a set of attributes which is defined as the primary key. The primary key may be a singleton set, which means that is a set with only one value and thus the primary key will be represented by a single attribute. These attributes are used to enforce entity integrity, a constrain to ensure that every tuple will only appear once in the relation. It is worth mentioning that the primary key cannot hold the null values because they cannot be used to identify a tuple (Watt & Eng, 2014).

Tuples

Attributes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Students | StudentId | LastName | FirstName | Age | CourseId |
| 1 | Smith | James | 20 | COO1 |
| 2 | Williams | Alan | 21 | COO2 |
| 3 | Jones | David | 19 | COO1 |
| 4 | Green | Bob | 21 | COO4 |
| 5 | Green | Taylor | 22 | COO1 |

Figure 2.1 Relation Students

### 2.5.2 Relational Database Schema

So far, we discussed the properties of a single relation in a database. A relation’s schema represents the name and the attributes of a relation e.g., Students (StudenId, LastName, FirstName, Age, CourseId). Although the primary aim of relational databases is to represent real world objects and the relationships among them. As a result, relational databases, can have more than one relation and the schema is called relational schema. Within the Relational Schema there may be associations between relations that are used to connect related information that are stored into different tables. By this, a tuple in one relation refers to a tuple in another relation. To maintain these references, the concept of the foreign key is introduced. A Foreign key of a relation A is a set of attributes that references a relation B only if it satisfies these two conditions:

* All attributes in the foreign key of relation A have same domain as the attributes of the primary key of relation B and
* A value in a tuple tA where tA∈A, either appears as a value in the primary key for a tuple tB where tB∈B, or the value is null.

If those conditions are true, then we can say that relation A is referencing relation B

## 2.6 Relation Algebra

“A static model of an information system of whatever complexity is of little use” (Stanczyk, Champion, & Leyton, 2001). A system’s main functionality is to process the data stored in its database to produce meaningful information. Therefore, the need of operations applicable to relations arises.

Relational algebra is a procedural language used to perform operations on relational databases. Its main application is to the theoretical foundations of relational databases and to help understand the concepts of query execution and query optimization. In a relational mode, as already discussed in the previous sections, the fundamental structure is a relation, also known as a set of tuples. Equitably, the processing primitives used to manipulate the data stored in the database are set oriented, that is processing multiple sets of tuples simultaneously, using a single operation. (Stanczyk, Champion, & Leyton, 2001). Finally, the fundamental property of relational algebra is that every operation accepts as arguments relation instances and returns a relation instance as a result. (Raghu & Johannes, 2003)

### 2.6.1 Relational Algebraic operations

Relational algebra has two different types of operators, which are called the passive and the active operators. The active operators can perform actions that modify the data stored in the database, e.g., insert a new tuple into an already existing relation, modification of one or more values stores in tuples, and finally, remove tuples that are no longer useful in the current relation. On contrary, passive primitives permit derivation of information by associating and retrieving different data values from one or more relations. (Stanczyk, Champion, & Leyton, 2001) Figure2.2 represents all actions that form a complete set of operations that can be applied to a relational schema.

BINARY OPERATIONS

UNARY OPERATIONS

RESTRICTION

INTERSECTION

DIFFERENCE

PRODUCT

DIVISION

PROJECTION

SELECTION

UNION

JOIN

Figure 2.2 Relational algebra operations

As we can see from the above figure the operations are divided into two categories, the binary and the unary. The difference is that a unary operation can be applied to a single relation, but a binary operation in order to be applied at least two relations are needed. For this project, the selection, projection and join operators will be discussed as in more details in the later sections along with the aspects or relational algebra, and query optimization.

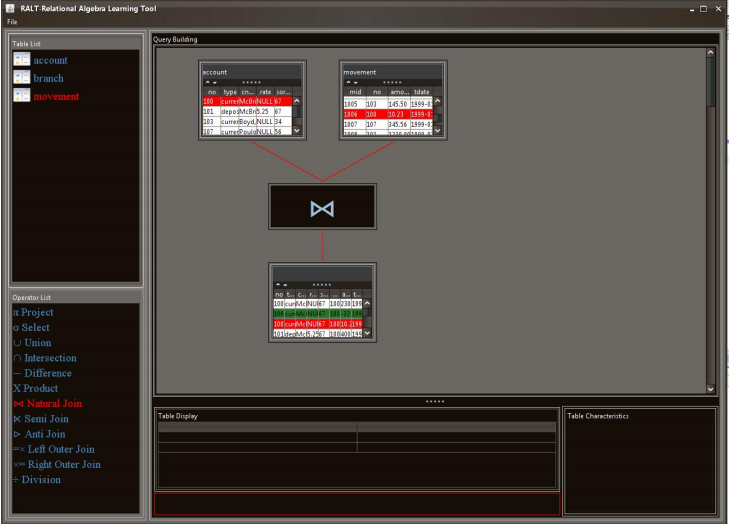
## 2.7 Similar projects

In this section we investigate two systems, RALT and the Query Visualizer. Despite the fact that SQL language and Relational algebra are widely used in the computing industry, there is a lack of tools that combine both these topics. The majority of the existing tools are built solely for either SQL or Relational algebra. Due to the limitation of existing projects that are under the same category as the project presented in this report, this section presents two systems that do not serve the same purpose as YourSQL, but they have similar features. We will discuss the strengths and weaknesses of each system along with their features to outline any implications with the current system.

### 2.7.1 RALT – Relational Algebra Learning Tool

RALT is a Java application that was developed by Pritam Mitra in 2009 (Mitra, 2009). Through an interactive user interface, RALT aid teaching of relational algebra in a more interactive and enjoyable manner. The user can construct query statements by dragging relational operators into the “Query Visualization Panel” (Figure 2.3), hence the users avoid manually typing the commands. This is considered an advantage of this system for the following reasons:

* Eliminates the possibility of a syntax mistake and
* allows the user to focus on the comprehending the concepts of relational algebra rather composing the query in the correct syntax (Mitra, 2009).



Error Console panel

Table Characteristic Panel

Table Data Display Panel

Query Visualization panel

Operator Panel

Table name panel

Figure 2.3 RALT Interface. This image is taken from Mitra (2009)

On top of that functionality RALT allows to the user to observe the results of every intermediate step towards the execution of the statement by drawing the results into the “Query Visualization Panel” in the form of a table as we can see from Figure 2.3 and thus enabling immediate feedback on the user’s operations even though the query it is not fully composed. Moreover, RALT makes use of data lineage and this what makes this system exceptional, because as Mirta stated “None of the existing tools for teaching relational algebra make use of data lineage” (2009). Data lineage allows the user track how data are retrieved and in the terms of relational algebra indicates which rows contribute to compose a row. It offers great benefits to students as they can have a walkthrough of the execution process and learn how the Relational Algebra operations are executed. Furthermore, it provides a mean to students to diagnose the position of errors made in the query as the resulting records will differ from the expected ones. These features constitute a great benefit to the learner, and this are what makes the system exceptional.

Data lineage is deployed by selecting a row from the resulting table and all the rows that contributed to compose that row are highlighted in red along with the selected row. We can see this functionality in Figure 2.3. This functionality assists users to better enhancing how Relational Algebra works.

Query construction using the interface it is an easy and effective process although it becomes more complicated as the query becomes more complex. Also, due to the rules imposed by the programmer, when a change occurs in the query, if often requires the user to rebuild the entire statement, e.g., when a node with no parent is deleted, the entire tree is deleted as well (Mitra, 2009). To provide a clearer example consider the tree composed in Figure2.3, if the user wishes to test the result of the JOIN operator with relations “account” and “branch” instead of “movements”, then by deleting the node named “movement” to replace it with the new node will delete the entire tree and the user will have to re-build it. Moreover, another drawback of the system is that when the user selects a row to apply data lineage, the resulting rows are indicated using colours. This functionality will be of no use for users with monochromatic vision[[2]](#footnote-2) and may be confusing with people of any king of colour blindness.

Even though this system is especially designed for teaching only Relational algebra concepts and it is not related to SQL in any way, I believe that some features of this tool can be used to improve the project presented by this report. Data lineage is a feature that YourSQL can adopt to educate students and enhance the system’s attempt to identify the user’s mistakes.

### 2.7.2 Query Visualizer – QV

Query visualizes is a tool that aims to assist in teaching both relational algebra and SQL language while providing with the explanation and details of how each result is retrieved for every query rather than just presenting them. The system provides the user two options, to build a database through the graphical user interface (GUI) or load an existing one. When a relational algebra statement is entered, it is evaluated the by Relational Algebra (RA) parser and when a SQL statement is entered it is converted into a tree of relational algebra operations that will be then evaluated by the RA parser. As already mentioned, QV provides the user with feedback through all the execution process, by illustrating the result in a tree format that each node in the tree represents the intermediate result of an execution step. Finally, it is worth mentioning that QV not only allows simple optimizations for the queries, but it provides the user with timing and memory usage statistics (Constantinou, 2010).

Figure 2.4 displays the GUI of QV when a Relational algebra query is executed. The query is translated into a query tree and displayed along with the results. This system has some similarities with YourSQL, as in both systems the query it is displayed using a canonical tree, and the user is allowing to see the results of an intermediate step by clicking a node from the tree. Although they both implement this functionality, QV it is more efficient as it temporarily stores the result of every step and present them to the user when requested. YourSQL implements this functionality by calling a select statement on the relation that needs to be presented every time this request is made.

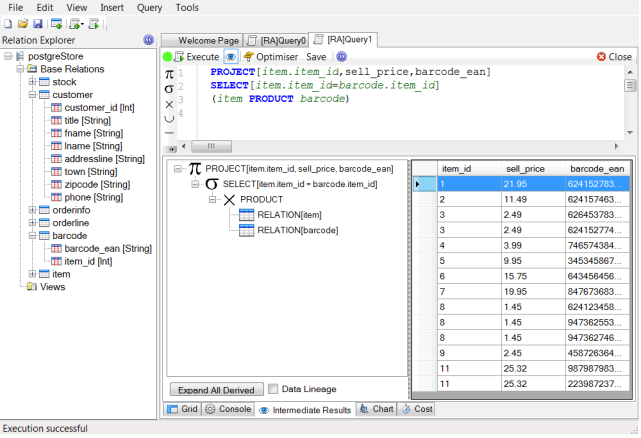


Figure 2.4 Query visualization User Interface. This image is taken from Constantinou (2010)

## 2.8 Development Environment

This section will give an overview of the platform and technologies used to implement the relational database tool and the reasons that each one was the suitable for this project. Notice that further information on the actual implementation process will be described in a following chapter in more details.

### 2.8.1 ANTLR – Another Tool For Language Recognition

When the system accepts an input, it must be validated that it is written in the language specified by the program and ANTLR[[3]](#footnote-3) is a compiler-compiler tool that is used in this project to apply this functionality. ANTLR is a powerful parser generator for reading, processing and executing or even translating structured text that is given through the format of a grammar file (Parr, 2013). Moreover, ANTLR provides pre-defined grammars for SQLite which is adjusted to provide the rules needed to evaluate the subset of commands that will be implemented in this project. The original grammar file can be found in Kiers (2014).

### 2.8.2 Java

Java is an Object-Oriented programming language that can is considered highly portable as it can be installed into systems that support different OS. Besides this, there are other reasons that can justify this decision which are specified bellow.

* Based on my previous experience with programming I felt more confident using java for this particular project as I tend to be more familiar with this language and due to the time limitation, I preferred to choose tools that I was knowledgeable of.
* Java provides many open source libraries that would appear to be useful for this project. The two main libraries used in this project.
  + JDBC - Java Database Connectivity API: Used to establish a connection between the SQLite database management system and the java application
  + JavaFX: Used to develop the GUI that the user will be using to interact with the system. Even though I was familiar with Swing I choose JavaFX instead due to the numerous tutorials that can be found online and the consistency that it provides across multiple platforms.

### 2.8.3 GitHub

Is a web-based hosting service for version control that uses Git. Git is distributed free and open source version control system that is designed to track changes in source code files and to support distributed workflows (Software Freedom Conservancy, n.d.). I used this tool for managing the source code and restore previous versions of my program in case that something when wrong. Also, GitHub gave me the opportunity to work in different environments as I could connect to my repository from different machines.

# 

Software Requirements

Small summary….

## 3.1 Requirements

In this section, the software requirements that define the features and functionality of YourSQL are described using a table format. These requirements should convey the user’s needs and expectations for this system since the aim of gathering this information is to help develop a sophisticated system that encompass these needs in order to improve a user’s quality of work in a domain. Although, they must not be specific in terms of the implementation process.

These requirements are divided into two categories, the functional and non-functional. The functional requirements as specified by the name, are linked to the functional aspects of the system and they are used to define the functionality and features of a software system. On the other hand, requirements that are specified as non-functional, are more implicit and describe the characteristic of a system such as performance, or storage capacity, security etc.

The tables below illustrate the requirements of the system and specifies the type of each requirement, functional or Non-Functional. Also is specifies whether a requirement is fully supported (Completed), partially supported (Partially Completed) or not supported at all (Non-Completed) by the system.

### 3.1.1 General Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Requirement Description** | **Type** | **Status** |
| **1.1** | The program shall be run by clicking on the application’s icon/shortcut. | Functional | Completed |
| **1.2** | The system must be able to run on any host with Java version 8 installed. | Non-Functional | - |
| **1.3** | The system shall have simple GUI that will allow simple navigation through the system. | Non-Functional | - |
| **1.4** | The system shall be easy to learn and use by user’s with basic knowledge of SQL. | Non-Functional | - |
| **1.5** | The system’s code shall be well written and documented. | Non-Functional | - |

Table 3.1 System’s General Requirements

### 3.1.2 Database Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Requirement Description** | **Type** | **Status** |
| **2.1** | The system shall enable the user to create a new database. | Functional | Not-Completed |
| **2.2** | The system shall enable the user to load an existing database file from a directory in the host machine into to the application. | Functional | Completed |
| **2.3** | The system shall enable the user to delete and modify a database. | Functional | Partially-Completed |
| **2.3.1** | The user shall be able to delete a database during runtime. | Functional | Not-Completed |
| **2.4** | The system shall allow the user to save and reset the database’s state. | Functional | Completed |
| **2.4.1** | The user shall be able to save any changes made to the database during runtime. | Functional | Completed |
| **2.4.2** | The user shall be able to restore the latest version of the database file during runtime. | Functional | Completed |
| **2.5** | The user shall be informed when any of the operations described by requirement 2.3.1, 2.4.1 and 2.4.2 is performed. | Functional | Completed |
| **2.5** | Upon exiting the user shall be asked to either commit or discard any modification made to the database file before closing the application. | Functional | Completed |
| **2.6** | The system shall enable the user to switch between databases. | Functional | Completed |
| **2.7** | The system shall allow the user to view the relations that exists in the database. | Functional | Not-Completed |
| **2.8** | The system shall allow the user to view the schema of a particular relation. | Functional | Not- Completed |
| **2.9** | The system shall be able to load a database file of any size | Non-Functional | - |

Table 3.2 System’s Database Requirements

### 3.1.3 Query Execution Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Requirement Description** | **Type** | **Status** |
| **3.1** | The system shall enable the user to execute both data definition and data manipulation commands. | Functional | Partially-Completed |
| **3.2** | The system shall allow the user to enter the SQL statement for execution. | Functional | Completed |
| **3.3** | The system shall validate the user input and perform one of the actions below based on the validation result:   * Produce error message. * Inform user about an action taken. * Display trees and resulting relation | Functional | Completed |
| **3.4** | If applicable the system shall convert the user input into a canonical tree and display it graphically to the user. | Functional | Completed |
| **3.4.1** | The system shall execute the canonical tree step-by-step. | Functional | Completed |
| **3.4.2** | The system shall display the resulting relation to the user. | Functional | Completed |
| **3.4.3** | The system shall allow the user to view the records of every resulting relation of every intermediate execution step taken to execute the given query. | Functional | Completed |
| **3.5** | If applicable the system shall apply simple optimization algorithm to the canonical tree to obtain the optimized version if it exists. | Functional | Completed |
| **3.5.1** | Requirements 3.4.1 and 3.4.3 shall apply for the optimized tree as for the canonical. | Functional | Completed |

Table 3.3 System’s Query Execution Requirements

### 3.1.4 Query File Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Requirement Description** | **Type** | **Status** |
| **4.1** | The systems shall load a file containing a list of SQL Queries. | Functional | Completed |
| **4.2** | The system shall enable the user to view the queries. | Functional | Completed |
| **4.3** | The system shall enable the user to hide the queries. | Functional | Completed |
| **4.4** | The system shall enable the user to add a query to the list and essentially to the file. | Functional | Not- Completed |
| **4.5** | The system shall enable the user to delete a query from the list and essentially from the file. | Functional | Not- Completed |

Table 3.4 System’s Query File Requirements

### 3.1.5 Graphical User Interface Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Requirement Description** | **Type** | **Status** |
| **5.1** | The interface shall consist of 7 components | Functional | Partially-Completed |
| **5.1.1** | A menu bar with the following labels:   * File – Which contains two sub-labels:   + Load Database – Shall allow the user to load a database file.   + Exit – Shall allow the user to close the application. The user shall be asked to either commit or discard any modification made to the database file before closing the application or cancel this action.   + Create database – Shall allow the user to create a new database file to store the database. * Options – Which contains four sub-labels:   + Save Database – Shall allow to save the database.   + Reset Database – Shall allow to restore the previous version of the database file.   + View Queries – Shall allow the user to view the queries mentioned in requirement 4.2   + Hide Queries – Shall allow the user to hide the queries[[4]](#footnote-4). | Functional | Partially-Completed |
| **5.1.2** | An Input Field that should allow the user to type the statement that will be executed. | Functional | Completed |
| **5.1.3** | A canonical tree Panel that shall display the canonical tree. | Functional | Completed |
| **5.1.4** | An optimized tree Panel that shall display the optimized tree. | Functional | Completed |
| **5.1.5** | An output panel that should display the resulting relation and the errors messages. | Functional | Completed |
| **5.1.6** | A Query panel that should display the queries loaded from the query file. | Functional | Completed |
| **5.1.7** | A Database description panel should display the relations that compose a database if requested. | Functional | Not-completed |
| **5.2** | The system shall display the result of an intermediate step when requested in a new window. | Functional | Completed |
| **5.3** | The system shall display the schema of a particular relation when requested in a new window. | Functional | Not-Completed |
| **5.4** | The system shall enable the user to clear the window. | Functional | Completed |

Table 3.5 System’s GUI requirements

### 3.1.6 Error Checking Requirements

|  |  |  |
| --- | --- | --- |
| **No.** | **Requirement Description** | **Type** |
| **6.1** | The system shall evaluate the statement for syntax mistakes and indicate the error to the user. | Functional |
| **6.2** | It shall evaluate the statement for semantic mistakes and indicate the error to the user. | Functional |
| **6.3** | The system shall produce error messages and notifications based on the use’s interaction with the system and the database. | Functional |

Table 3.6 System’s Error Checking Requirements

These requirements provide a description of the system’s functionality and they specify in detail what the system must be able to do and what features will be available for the users. As we can see from the above tables some of the requirements are not yet supported by the system at all or they are not fully supported. Due to time limitations, the most important features where chosen to be implemented first so that primary aim of these projected will be implemented; query execution and optimization, and based on the progress additional features where added; reading queries from a file. Unfortunate there weren’t enough time to implement all the features described but they will be discussed in a later chapter of this report.

The requirements where aspired/derived from the aims of the projects. Based on the aims and the purpose that these project servers the appropriate requirements were derived. Consequently, it is crucial that these requirements are used as guidelines during the implementation process, so that the emerging system will satisfy the user’s needs and will best attend the purpose of these project.

## 3.2 Use case Diagram

To represent the user’s interaction with the relational database tool, we will be using a use case diagram. The following figure captures the dynamic aspect of the relational database tool and the potential uses of the system. It also illustrates how the all the components of the system corporate to perform the functionalities of the system. The user must load a database and then he can perform various actions. As illustrated by Figure 3.2 the user can load, save a database, clean the stage submit a query.

A picture containing text, map

Description automatically generatedFigure 3.2 Use case diagram for YourSQL

## 3.3 Main Sub-System decomposition

A screenshot of a cell phone

Description automatically generatedx

References

Allen, G., & Owens, M. (2010). In *The Definitive Guide to SQLite* (2nd ed.). New York, United States of America: Apress. Retrieved February 16th, 2019

Constantinou, G. (2010). *Relational Algebra and SQL Query Visualization.* Imperial College, London. Retrieved February 21th, 2019

Forta, B. (2013). In *Sams Teach Yourself SQL in 10 Minutes* (4th ed.). Indianapolis, Indiana: Sams. Retrieved February 16th, 2019

Kiers, B. (2014). *antlr/grammars-v4/sqlite/SQLite.g4*. Retrieved October 2018, from ANTLR 4 Documentation: https://github.com/antlr/grammars-v4/blob/master/sqlite/SQLite.g4

Kreibich, J. A. (2010). In M. Loukides (Ed.), *Using SQLite* (1st ed.). United States of America: O’Reilly Media. Retrieved February 16th, 2019

Leifer, c. (2019). *Appropriate Uses For SQLite*. (Small. Fast. Reliable.Choose any three) Retrieved February 19th, 2019, from SQLite: https://www.sqlite.org/whentouse.html

Leifer, c. (2019). *Release History*. (Small. Fast. Reliable.Choose any three) Retrieved February 19th, 2019, from SQLite: https://www.sqlite.org/changes.html

Mitra, P. (2009). *Ralational Algebra Learning Tool.* London. Retrieved February 21th, 2019

Parr, T. (2013). In L. The Pragmatic Programmers, & S. D. Pfalzer (Ed.), *The Definitive ANTRL 4 Reference* (1st ed.). United States of America, Texas: The Pragmatic Bookshelf. Retrieved October 2018, from https://pragprog.com/book/tpantlr2/the-definitive-antlr-4-reference

Raghu, R., & Johannes, G. (2003). In *Database Management Systems* (3rd ed.). Singapore: McGraw-Hill Higher Education.

Software Freedom Conservancy. (n.d.). *Git --everything-is-local*. Retrieved from Git --distributed-even-if-your-workflow-isnt : https://git-scm.com/doc

Stanczyk, S., Champion, B., & Leyton, R. (2001). In S. o. University, *Theory and Practice of Relational Databases* (2nd ed.). Oxford, United Kingdom : Taylor & Francis. Retrieved February 19th, 2019

Stoll, R. R. (1963). In *Set Theory and Logic* (2nd ed., pp. 2-3). New York, United States of America : Dover Publications, Inc. Retrieved February 19th, 2019

Watt, A., & Eng, N. (2014). In *Database Desing* (2nd ed.). BCcampus. Retrieved February 20th, 2019, from https://opentextbc.ca/dbdesign01/

1. Relational Database management systems are used to store user defined records in large tables. [↑](#footnote-ref-1)
2. People with monochromatic vision cannot see any colour and their perception of colour is limited to different shades of grey raging from black to white. Taken from <http://www.colourblindawareness.org/colour-blindness/types-of-colour-blindness/> [↑](#footnote-ref-2)
3. For this project ANTRL version 4 was used. [↑](#footnote-ref-3)
4. In case the there is no file to load both labels will be disabled. [↑](#footnote-ref-4)