DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

CSCI 5408 Data Warehousing
Project Report

Submitted By:

Group 14

Chetanpreet Singh Sachdeva (B00853545)

Anqi Chen (B00838586)

Table of Contents

1 Team Members

1.1 Chetanpreet Singh Sachdeva

1.1.1 Roles

- Helped in the initial formulation of ideas when it came to Business logic.
- Helped in organizing the flow of work amongst the team.
- Worked around the actual processing of data and perform actions based on user queries.
- Handled any miscellaneous tasks too.

1.1.2 New Learning

- Developed major insight into how an actual DBMS stores data.
- Understood the user query and background data processing in detail.

1.1.3 Difficulties

Had trouble handling the additional workload as the team was working with one less member than other teams. Had to leave some requirements reluctantly to manage time.

1.2 Anqi Chen

1.2.1 Roles

- Understood business logic and created the flow chart of our application.
- Solved the problem of query parsing and extracting to the corresponding data type part.
- Built the logic about adding relationships between tables and wrote logs for users.
- Tested for functionalities.

1.2.2 New Learning

- Use Regex to parse SQL statements, trying to be as accurate as possible.
- Understand the constraints of adding a foreign key and creating the relationships between two tables.
- Find the way of using data types to represent and store data in DBMS.

1.2.3 Difficulties

As Chetan said, our biggest challenge would be the lack of time due to the reduction of team members. Even if we considered adding more features or taking data security measures, we still could not accomplish all tasks at the end given the time limit. Moreover, the debugging and testing of the application is more difficult than we expected, because every small error is taken into account when fetching the data, even typos. However, we gave our best.

2 Introduction

2.1 Project overview

The project requires us to design and implement a working DBMS using any programming language of our choice. A DBMS handles all the operations associated with a database while making sure that users have a simple operational experience. This meant that all the inner processing of the database should be abstracted well, and the user should not experience any difficulty even when performing complex work.

2.2 Block diagram

The block diagram describing how our application is designed to be organized is shown below (Figure 1).

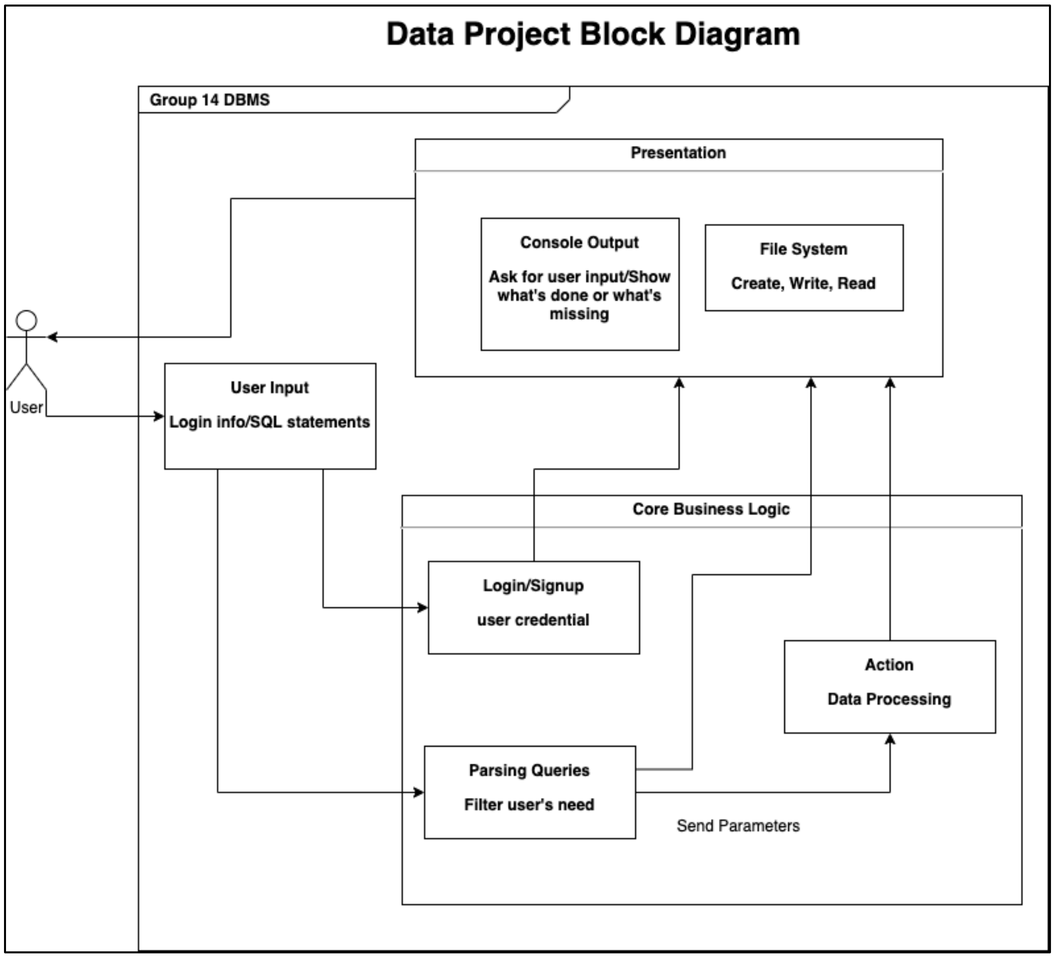


Figure 1 Block Diagram of our Application

2.3 Possible Scenarios

Some usual user scenarios presented in a DBMS can include:

- Adding a new user
- Creating Tables
- Inserting new Values
- Selecting values from existing tables
- Updating those values
- Deleting some rows in one table
- Dropping the entire table
- Assigning Primary and Foreign Keys to the tables
- Getting information on relationships between tables created

Some usual administrator scenarios presented in a DBMS can include:

- Parsing user's queries in standard SQL format and processing data to do corresponding operations
- Using the data dictionary to store structural information
- Managing and retrieving data in different tables
- Tracing query execution time in general logs
- Tracing user's input and database changes in event logs
- Creating simple ERD to see relationships between tables in DBMS

3 Conceptual Framework

3.1 Understanding of the problem

To demonstrate a working DBMS, we needed to decide on some Data Structures as our program would have to work in a way that is not usually used in actual DBMS from a user's perspective. Our application should be able to handle user's requests, provide a command-line based user interface and perform several (if not all) functions that a simple DBMS would offer.

We were working with a programming language not directly used in DBMS, so our first problem was to handle the user commands and convert them accordingly so that our program can understand and function accordingly. Besides, the other part was to handle the Storing and processing of Data in a coherent and persistent manner that doesn't disrupt the flow of the program.

3.2 Initial Flow Diagram

Our initial flow diagram is built as below (Figure 2), it clearly shows the whole process of a basic workflow that user can perform in our system.

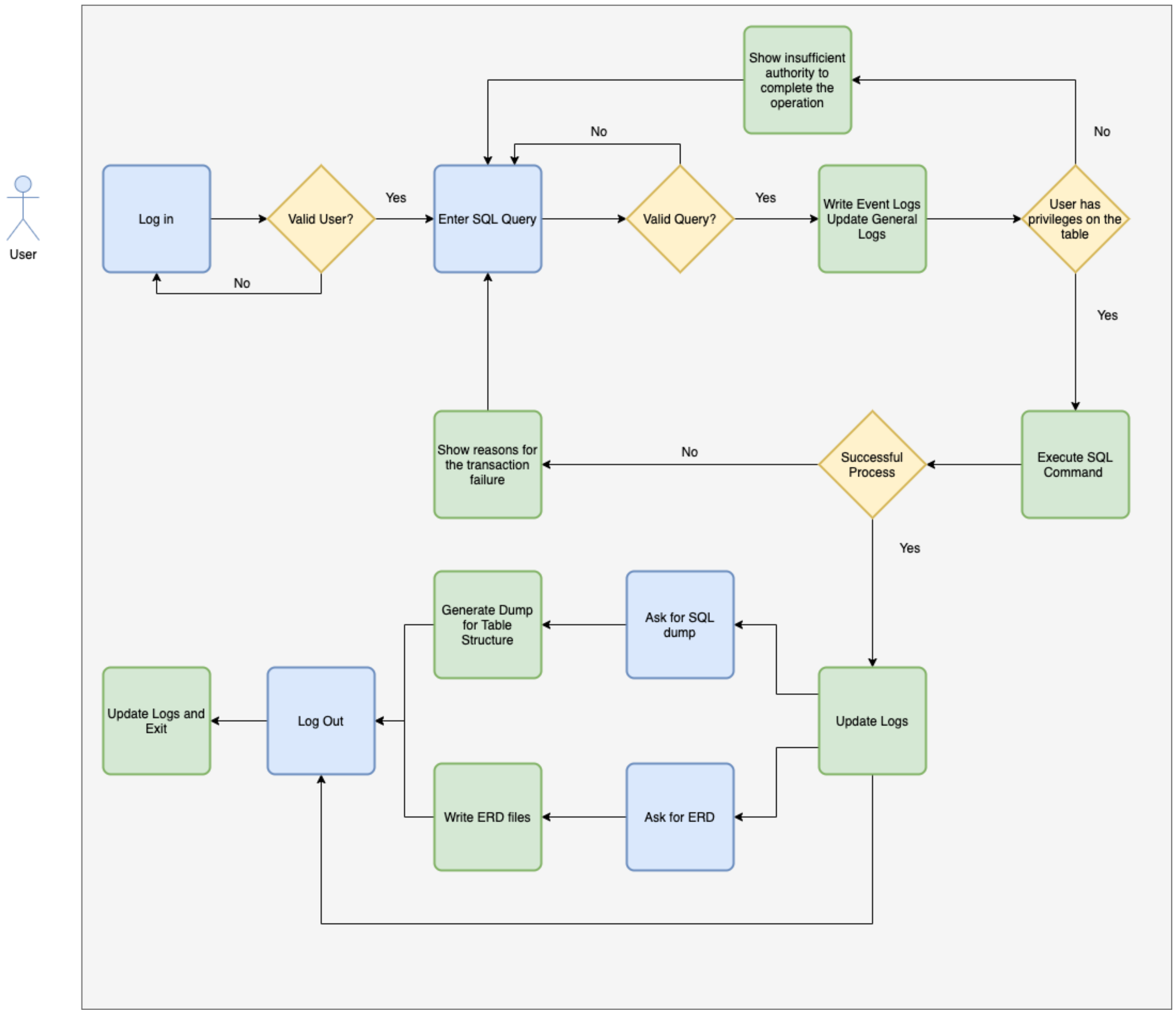

User
Log in
Valid User?
Yes
No
Enter SQL Query
Valid Query?
No
Yes
Write Event Logs Update General Logs
User has privileges on the table
No
Show insufficient authority to complete the operation
Yes
Execute SQL Command
Successful Process
No
Show reasons for the transaction failure
Yes
Update Logs
Ask for SQL dump
Generate Dump for Table Structure
Ask for ERD
Write ERD files
Log Out
Update Logs and Exit

Figure 2 Initial flow diagram

3.3 Initial Algorithm

We initially identified the need for the following Algorithms before we started working on Code:

1. For checking if an attribute has a unique value in an Array list

a. Select the Array List.
b. Select the Attribute that you want to Check.

c. Traverse the Array list through the attribute of that entity.
d. If a duplicate entry is found, the return fails.
e. If not, continue traversal.
f. Return Success if no duplicate entry is found.

2. Conversion of SQL query to method

a. Parse the first word of the query. Assume it start with “select”. Store it temporarily.
b. Now select the word following ‘from’ so we know which table we are talking about. We need to store this information as well.
c. Now go to the method that displays all records and pass in the table name for reference.
d. Traverse the entire table and display records to the user.

3.4 Initial Data Structure Details

We looked at different Data Structures based on their advantages and disadvantages. The main use cases helped us narrow down the scope of our research. As we are planning to work with Java, we thought of using Graphs and Array lists for several reasons:

- Graphs can represent Entities with their vertices and Relationships using their edges.
- Use different java classes to represent each entity, in which variables of that entity can act as attributes. Once we use a class object to represent one set of attributes (like a row of the table), we can use an array list of array list to create an entire table.
- Here the Outer Array list acts as a row number and the inner one holds the class object/entity for one member(row).

4 Final Implementation

We had to make certain choices when it comes to the practical application compared to our initial thoughts regarding the problem. So here we focus on the actual implementation which differed from what we had considered initially to be our solution.

4.1 Implementation Environment

We’ve chosen JAVA as our programming language to develop this application, IntelliJ is used as the common platform to support development. GitLab is used for collaboration.

4.2 Data Structure

We’ve made use of our knowledge of OOPS concepts and the Java programming language while working on the problem. We used text files as a tool to store persistent data and then used different classes to run the queries as required by the user. Throughout this program, we’ve used Data Structures (Arrays, Lists etc.) to process data in and out of the aforementioned text files. However, the use of Data structures is flexible as long as we have the right persistent data stored in a text file.

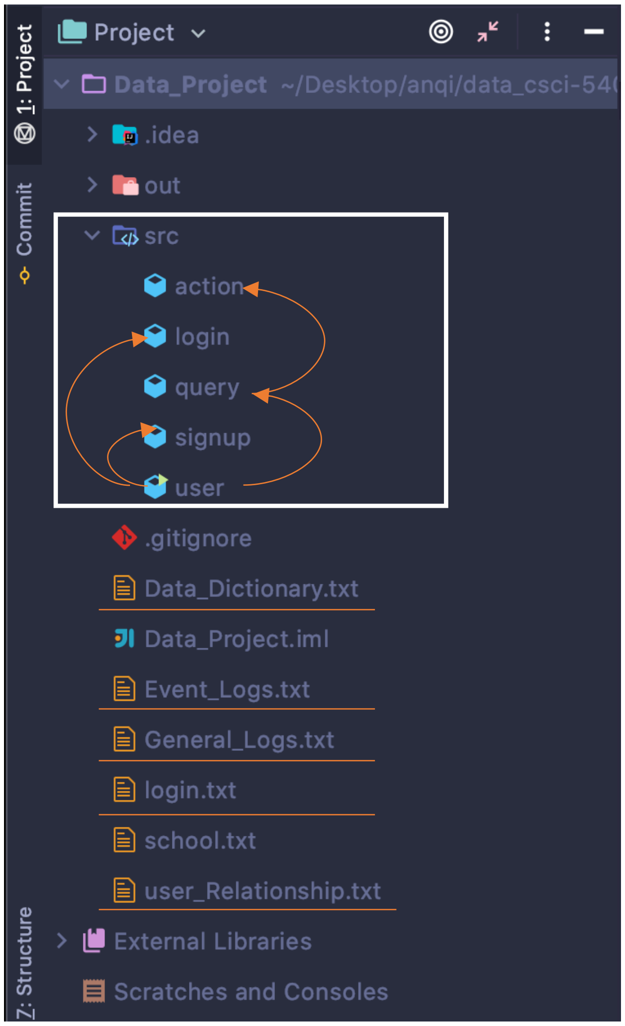


Figure 3 Flow and Different Text files generated

4.3 Implementation details

1. **Check user credential**
   For this, we store every new user and its credential in a text file. So, every time a user tries to log in, we can quickly look at their credentials in the text file. If it is a new user, we can always ask them to sign up and then they'll be added to the user record. Also, the user that creates a table is the one allowed to make significant changes to the table (Add primary key, drop table etc.). Every other user can easily read the data.

2. **Converting SQL Queries**

   We know that the user-input will be in an SQL format and for our Database Management System (DBMS) to work with it, we would need some form of conversion.

   As our queries will be simple and straightforward, we can create direct methods for a particular set of queries. For example: For a simple select* form of query, we have a method that can display the contents of the entire designated table.

   Regex Patterns are used to match the standard SQL statements, then Matchers are used to putting queries into different types of operations. Use the group method in Matcher to extract different parts of Queries. Regex is also used to split some parts into smaller pieces. Finally, we store the processed data using String or the Array List of String, then send those parameters to call the corresponding methods in the action class.

3. **Identifying different Keys**

   Every Database holds multiple types of keys in each table. To solve this problem, we can implement certain algorithms that make sure unique values are present in certain attributes.

   We've implemented a check for Primary keys in each table and also enable the user to add Foreign Keys. This generates a relationship text file that shows us which tables are connected and through which keys.

4. **Working with Data**

   As we are working with a conversion-based environment when it comes to handling data, we used text files as a means to store data persistently on our memory. This means that the data stays there even when the program is not running.

   Using our knowledge of Java and OOPS we create different classes that would handle different things: Queries (to handle and process user queries), Action (To perform the action that user requested) etc.

   The main functions of the database take place within the Action class. Here we get the processed user queries which are converted as per Java. We then have different methods defined for different purposes (Select, Update, Delete etc.). Each method is carefully made so that it adheres to all possible scenarios we might have in a Database (Entering duplicate values in a primary key etc.). We also check for user credentials so that no one can destroy the data in a table created by someone else. Reading the data is allowed to every user. After the requisite processing, each method returns 1 if the query was processed and 0 if there was something wrong.

   Using this type of environment we were able to make significant performance gains too. We also made sure our code is as clean and concise as possible.

5. **Generating Logs**

   Logs are generated in two ways:

   General Logs: Store the execution time of a query and user details.

   Event Logs: Gets recorded only when a change is made to the table. Example: User deletes a row etc.

5 Repository

Our project code can be found on GitLab. We've created separate branches and merged them into Master Branch to get the Final Clean code onto our Master Branch.

https://git.cs.dal.ca/csachdeva/data_csci-5408_project

6 Product Features

Based on the project specification file given, our application can perform the following:

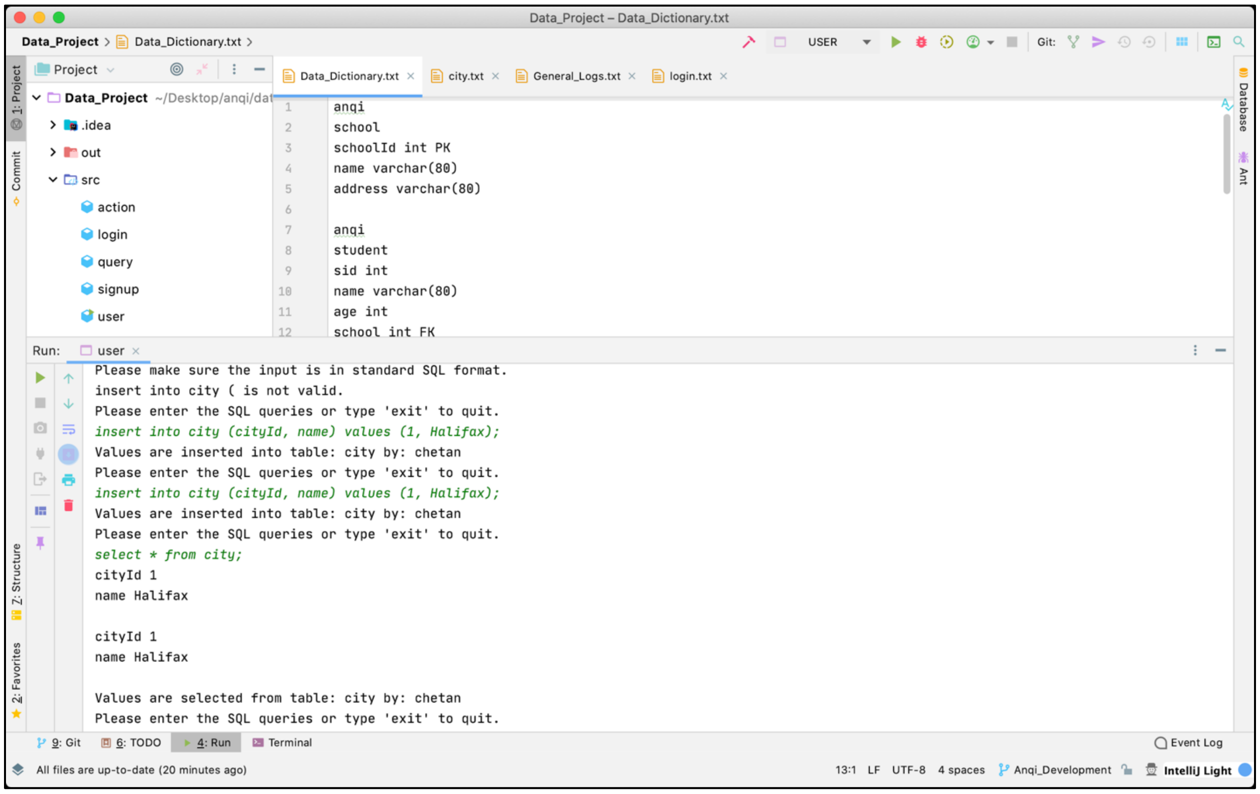
- Add a new user
- Create tables
- Insert values into tables
- Select values from existing tables
- Update values in specific rows of data
- Delete records in one table
- Drop tables
- Assigning Primary and Foreign Keys to the tables
- Getting information on relationships between tables created

The features contain the following aspects:

- Quick response time, only taking little operating memory.
- Ignores unnecessary spaces in queries.
- Easy-to-use user experience.
- Shows reasons for operation failures in detail.

6.1 Final Product Screenshots

We choose to display some of the features and files of our program as screenshots here for initial references.

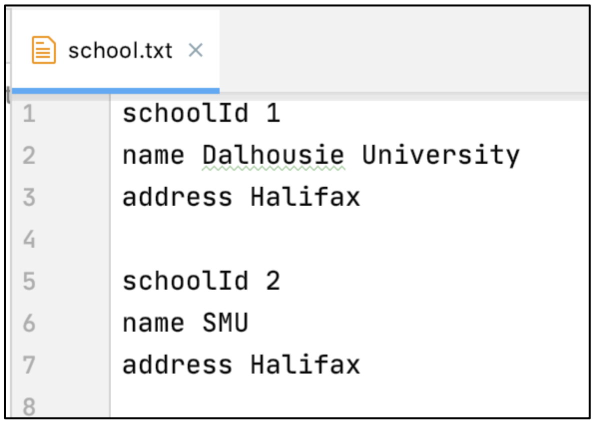


Data_Project – Data_Dictionary.txt
Data_Project > Data_Dictionary.txt >
USER
Git:
Project
Data_Project ~/Desktop/anqi/dat
.idea
out
src
action
login
query
signup
user
Data_Dictionary.txt
city.txt
General_Logs.txt
login.txt
Run: user
9: Git
6: TODO
4: Run
Terminal
Event Log
All files are up-to-date (20 minutes ago)
13:1 LF UTF-8 4 spaces Anqi_Development IntelliJ Light

```
anqi
school
schoolId int PK
name varchar(80)
address varchar(80)

anqi
student
sid int
name varchar(80)
age int
school int FK
```

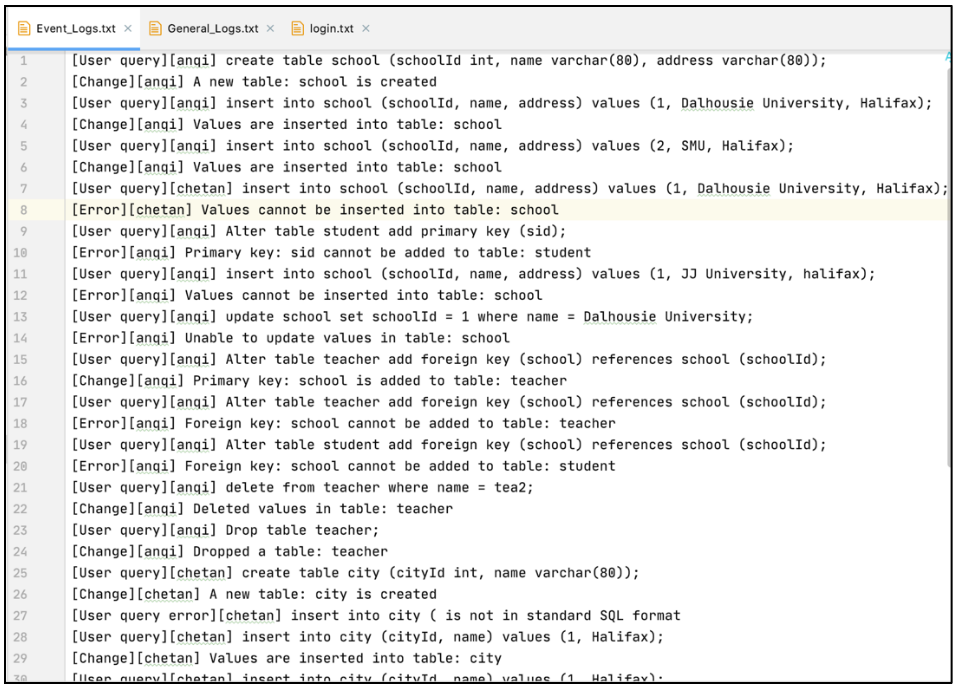
```
Please make sure the input is in standard SQL format.
insert into city ( is not valid.
Please enter the SQL queries or type 'exit' to quit.
insert into city (cityId, name) values (1, Halifax);
Values are inserted into table: city by: chetan
Please enter the SQL queries or type 'exit' to quit.
insert into city (cityId, name) values (1, Halifax);
Values are inserted into table: city by: chetan
Please enter the SQL queries or type 'exit' to quit.
select * from city;
cityId 1
name Halifax

cityId 1
name Halifax

Values are selected from table: city by: chetan
Please enter the SQL queries or type 'exit' to quit.
```

Figure 4 Overview of our final product



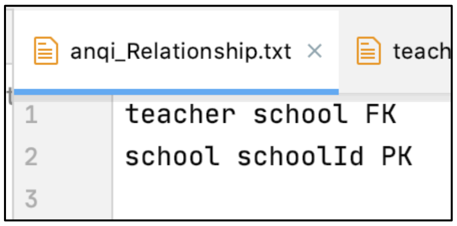
```
schoolId 1
name Dalhousie University
address Halifax

schoolId 2
name SMU
address Halifax

```

Figure 5 Data in the school table



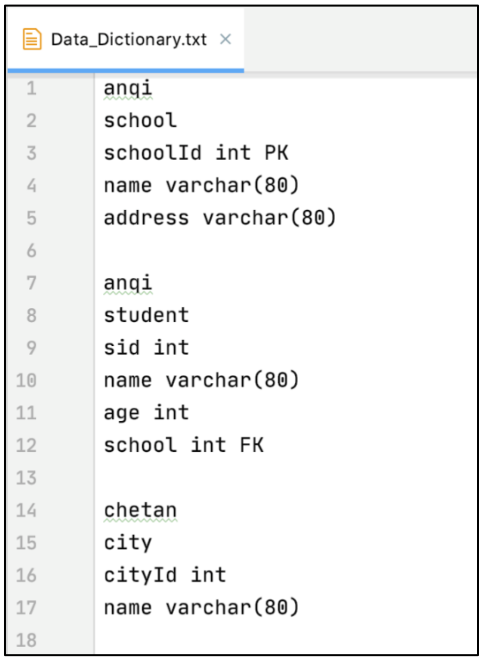
```
[User query][anqi] create table school (schoolId int, name varchar(80), address varchar(80));
[Change][anqi] A new table: school is created
[User query][anqi] insert into school (schoolId, name, address) values (1, Dalhousie University, Halifax);
[Change][anqi] Values are inserted into table: school
[User query][anqi] insert into school (schoolId, name, address) values (2, SMU, Halifax);
[Change][anqi] Values are inserted into table: school
[User query][chetan] insert into school (schoolId, name, address) values (1, Dalhousie University, Halifax);
[Error][chetan] Values cannot be inserted into table: school
[User query][anqi] Alter table student add primary key (sid);
[Error][anqi] Primary key: sid cannot be added to table: student
[User query][anqi] insert into school (schoolId, name, address) values (1, JJ University, halifax);
[Error][anqi] Values cannot be inserted into table: school
[User query][anqi] update school set schoolId = 1 where name = Dalhousie University;
[Error][anqi] Unable to update values in table: school
[User query][anqi] Alter table teacher add foreign key (school) references school (schoolId);
[Change][anqi] Primary key: school is added to table: teacher
[User query][anqi] Alter table teacher add foreign key (school) references school (schoolId);
[Error][anqi] Foreign key: school cannot be added to table: teacher
[User query][anqi] Alter table student add foreign key (school) references school (schoolId);
[Error][anqi] Foreign key: school cannot be added to table: student
[User query][anqi] delete from teacher where name = tea2;
[Change][anqi] Deleted values in table: teacher
[User query][anqi] Drop table teacher;
[Change][anqi] Dropped a table: teacher
[User query][chetan] create table city (cityId int, name varchar(80));
[Change][chetan] A new table: city is created
[User query error][chetan] insert into city ( is not in standard SQL format
[User query][chetan] insert into city (cityId, name) values (1, Halifax);
[Change][chetan] Values are inserted into table: city
[User query][chetan] insert into city (cityId, name) values (1, Halifax);
```

Figure 6 Event Logs



```
teacher school FK
school schoolId PK
```

Figure 7 Relationship File



```
anqi
school
schoolId int PK
name varchar(80)
address varchar(80)

anqi
student
sid int
name varchar(80)
age int
school int FK

chetan
city
cityId int
name varchar(80)
```

Figure 8 Data Dictionary with PK, FK set

6.2 Final Product Explanations

We believe the final product works well as a Standalone Database application. It responds quickly to user commands and the changes made are reflected almost immediately which saves a lot of time.

The storage limitations are based on the system on which it is loaded. It is also simple enough for the end-user as it accepts the general SQL query format and doesn't require any new knowledge.

The security is assured because every user has a defined login credential that is associated and kept in the same drive as the Database.

7 Conclusion and Future Work

In conclusion, we can happily say it has been a great learning experience. In the end, we were able to get a normal database working based on the Java programming language. It helped us gain a deeper insight into our Database concepts and improved our critical thinking ability even more.

The major problem we faced was having a member less while having the same workload as other groups. It meant we had to sacrifice some functionality (Concurrency Control, SQL Dump) to save time. Although all the members gave their best and probably more than what was asked of them at times, we fell short because of the aforementioned reason. If this wasn't the case since the beginning, we're sure we would've completed all of our tasks well. We hope this factor is considered when the entire project is evaluated.