CSC3170 Project Report

121090003 Jingzhi Bao,
120090792 Zimeng Huang, 120090108 Wei Zhou,
120040061 Ningyuan Wang, 119010306 Ruiyi Wang,
119020401 Jiaming Wang, 120090784 Qihua Yin

December 2022

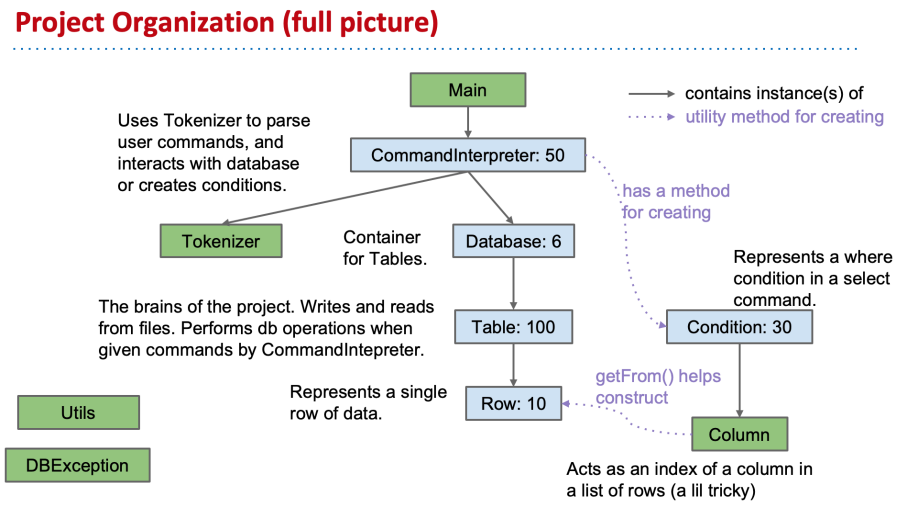
The Chinese University of Hong Kong, Shenzhen

1 Introduction

The communication with the database system using an artificial notation usually known as a *language*, although it is much simpler than any human language. The definition and processing of such languages is an important skill for any computer scientists. We normally think of programming languages such as Java, but there are many other contexts where small, domain- specific languages (DSLs) are appropriate engineering solutions to a design or implementation problem.

Thus, in this way, this project tries to implement a miniature relational database management system (DBMS) that stores tables of data, where a table consists of some number of labeled columns of information. Our system will include a very simple query language for extracting information from these tables.

2 Design



Our project mainly focuses on the *Command Interpreter* module and database structures of database system. It will basically support these functions:

1. Interpret SQL language.
2. Support the basic structures of database system (e.g. database files, tables, rows, columns).
3. Do fundamental queries on database system, including *select*, *from*, *where*, *insert into*, *create table* clauses.

The main function is in a public class called db61b.Main, which is the most direct way to invoke all the functions. The first line of output of our program identifies the program, and of course it may contain anything. Before reading the first command and on reading each subsequent end-of-line or comment, our program will print the prompt '> ' (which is followed by a blank as well), or '...' to indicate a continuation. The output standards can be referred in exactly the format shown in our results, and when encounters an error, the warning messages will be split on separate lines, starting with the word *error*. In addition, the program will exit without further output when it encounters a quit (or exit) command or an end-of-file in the input. When printing out the contents of a table, the final version of outputs may not follow the order of rows, but it will be in the order specified in the .db file or in the columns of the select command from which the table came. Besides, we also provide a *makefile* which is set up to compile everything on the command *gmake* and to run all the tests on the command *gmake* check.

3 Implementation

3.1 Command Interpreter

With the help of the given class **Tokenizer**, we implement the *CommandInterpreter()*, where we can break up all the inputs of the users and turn them into command that our program can execute.

We mainly have 7 kinds of commands here, which are: *Create*, *Load*, *Exit*, *Insert*, *Print*, *Store*, and *Select*.

Create: When command "create table" is found, read the table name, create an object of Table class and name it after the inputted name.

Load: Use *database.put()* to load the data into a existed table.

Exit: Simply exit a statement.

Insert: When "insert into values" is found, read the actual values that the users want to insert next. Create a Row object to catch these values, and insert the new row into a table.

Print: Print the data in a specific Table object.

Store: Use *table.writeTable()* to store the table as a database(.db) file.

Select: When "select" is found, execute *SelectClause()*, which works as follows:

1. Read the column names that the users want to select, until "from" is found.
2. Read the table names that the users want to select the columns from, until "where" is found.
3. Join all the tables.
4. Read all the conditions and use the *ConditionClause()* to validate them.
5. Select the columns in the joined table with the conditions.

3.2 Database

Database is the class that handles table instances. Specifically, the database class supports operations to store, retrieve, and delete tables. We utilize two ArrayList to store table instances and table names.

3.3 Table

We consider *Column* to be an important attribute in the *Table*, and it is only constructed when the database loads '.db' files or executes 'create' clause (for convenience, we call theses tables *database tables*, which are stored directory in heap). Namely, the *Column* is constructed at the same time as the 'initial' table.

Idea: the columns of a new table obtained by the 'select clause' is a subset of the 'database tables'. (We define a 'temporary table' is a table created by a 'select clause', otherwise 'standard table'. More specifically, the temporary table is one without explicit name)

To maintain this property, the Table should support two ways of construction:

1. Table(String[] columnTitles), used for *standard table*. (The columns are initialized by string array)
2. Table(List¡Column¿ columns), used for *temporary table*. (The table borrows columns from the table in database)

By emphasizing the importance of *Column*, the role of '_columnTitles' (type: String[]) is diminished. We will just use them as column titles to display to the user.

3.4 Row and Column

3.4.1 Row

Row is the basic storage unit in the form of a string list. We fill in the methods that are given by the original framework. The main addition is a join method, which returns the join of multiple rows. It simply merges rows together with a head-to-tail approach, and supports the join method in Table.

3.4.2 Column

To reinforce the importance of the column, the *Column* class needs to record:

1. it belongs to which table,
2. its name,
3. the full name ('tableName_columnName')

With feature 3, we can solve the problem of attribute name duplication.

To support condition filtering, we need to know where the rows in the table after Cartesian product come from which table in 'from clause'. We define a 'super column' having the attribute of 'offset', which records the offset in the ultimate columns after Cartesian product. Specifically, the 'offset' attribute is derived when we do table join (Cartesian product).

3.5 Condition

We follow the instructions and fill out the framework provided by this project. In class *Condition*, we design several constructors:

1. **Condition**(Column col1, String relation, Column col2)

   It is a condition representing COL1 RELATION COL2, where COL1 and COL2 are column navigators. and RELATION is one of the strings "<", ">", "<=", ">=", "=", or "!=".

2. **Condition**(Column col1, String relation, String val2)

   An overloaded Constructor that calls Condition(Column col1, String relation, Column col2), used when the third parameter is a literal value.

3. **boolean test**(Row... rows)

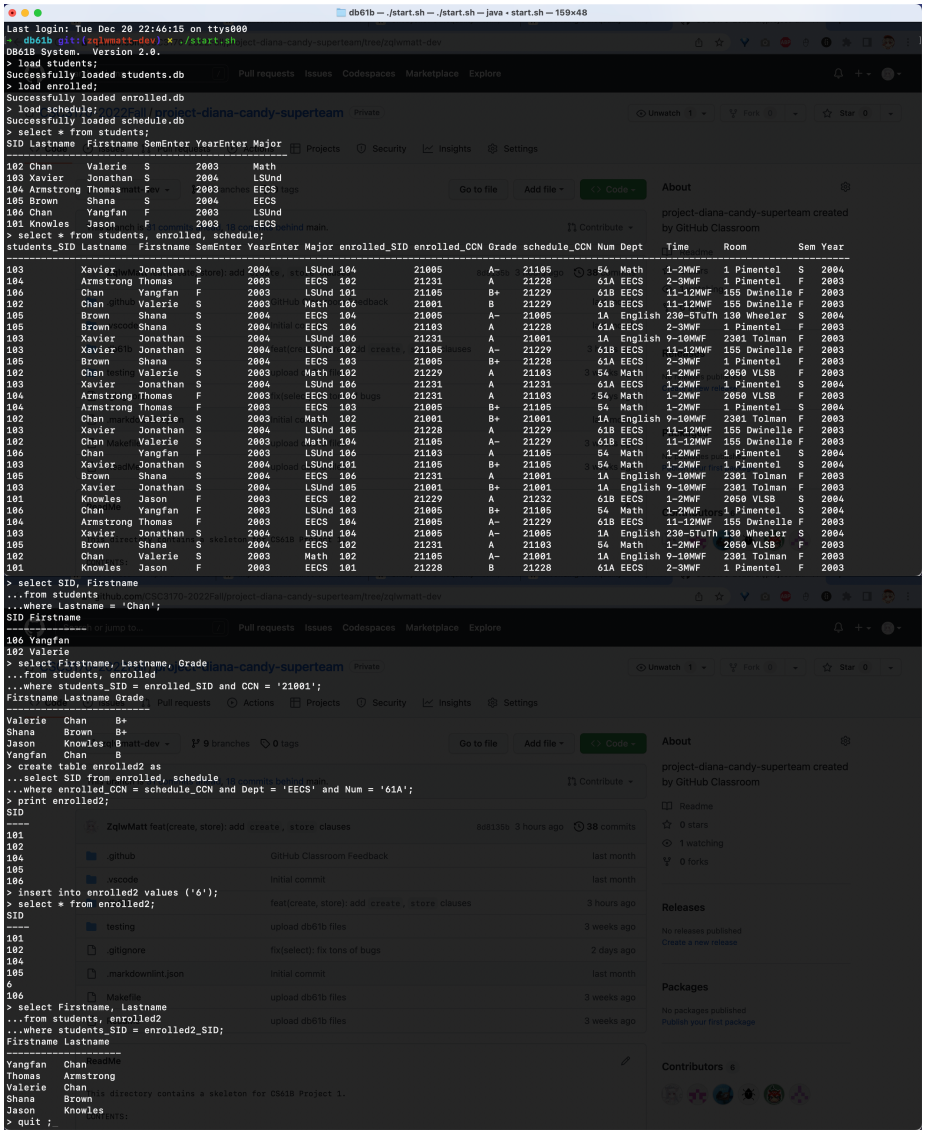
   Assuming that ROWS are rows from the respective tables from which columns are selected, returns the result of performing the test denoted. Only allows non-numeric type when checking equality or inequality. Any attempts applying "<", ">", "<=", or ">=" between non-numeric types will throw *NumberFormatException* error.

4. **static boolean test**(List¡Condition¿ conditions, Row... rows)

   This function will iteratively call boolean test(Row... rows). Return true if and only if ROWS satisfies all CONDITIONS.

4 Achievement

4.1 Result



4.2 Highlight

In addition to the features mentioned in the design idea, we also made optimization in these aspects:

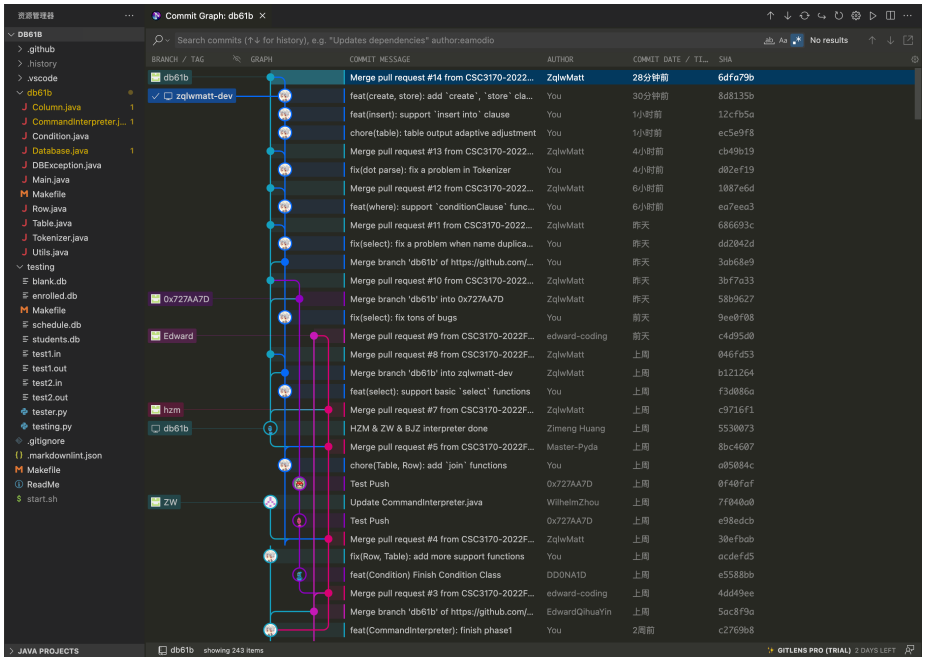
1. **Table.toString**: print table with adaptive adjustment.

2. **Table.select**(List<String> columnNames, List<Condition> conditions): supports column names and conditions filtering, and we increase the robustness of the function.

3. **Column**(String name, Table... tables): If the length of tables = 1, we are constructing a database table. Otherwise, we are constructing a super column (robustness).

4. **Column._offset**: use offset in super columns to derive the data after Cartesian product.

4.3 Timeline



5 Contribution

Each member of our group contribute equally on both report and presentation in this project. We follow the instructions in project description and divide tasks in six parts, everyone is responsible for one of them.

Name	Work in charge
Jingzhi Bao	whole project design and coding, presentation preparation and report writing
Zimeng Huang	*Command Interpreter* part, presentation preparation and report writing
Wei Zhou	*Command Interpreter* part, presentation preparation and report writing
Ruiyi Wang	*Table* part, presentation preparation and report writing
Jiaming Wang	*Table* part, presentation preparation and report writing
Ningyuan Wang	*Condition* part, presentation preparation and report writing
Qihua Yin	*Row* part, presentation preparation and report writing