CSC3170

Project Report

Genshin Team
120090590 Chen Feiyang
120010042 Tao Zhen
120090460 Zhu Luokai
120090582 Chen Yiwen
119010166 Li Zhuoyi
120090447 Wang Yiwen
120090089 Wang Yuqi
120090588 Xiao Weizhao

2022. 12. 28

Contents

1 Project Background **3**

2 Project Overview **4**

3 Code Environment **5**

3.1 Dependencies 5

3.1.1 JDK Version 5

3.1.2 Junit Version 5

3.1.3 5

3.2 How to Run 5

4 Basic Functions **6**

4.1 Database 6

4.2 Condition 7

4.3 CommandInterpreter 9

4.4 Row 10

4.5 Table 11

5 Project Extension **12**

5.1 Structure Improvement 12

5.2 Structure Drawbacks 15

6 Presentation **16**

1 Project Background

In Project option3, we create a database management system (DBMS) using static programming language (Java). We give the database system the basic functions required, while adding more convenient and efficient instructions based on our technology and ideas.

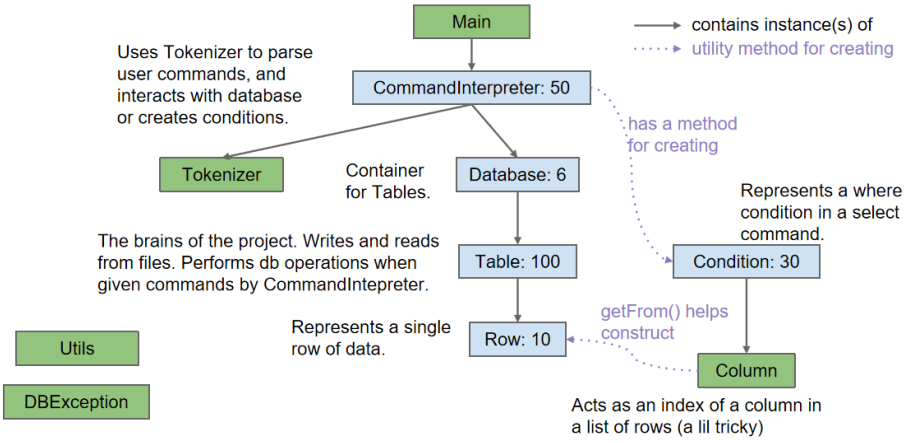


Figure 1: Project Organization

2 Project Overview

Our project object is a miniature relational database management system (DBMS) that stores tables of data in which we will set some number of labeled columns of data information. Our system will include a very simple query language for extracting information from these tables.

students

SID	Lastname	Firstname	SemEnter	YearEnter	Major
101	Knowles	Jason	F	2003	EECS
102	Chan	Valerie	S	2003	Math
103	Xavier	Jonathan	S	2004	LSUnd

Figure 2: Sample Table1, students

schedule

CCN	Num	Dept	Time	Room	Sem	Year
21228	61A	EECS	2-3MWF	1 Pimentel	F	2003
21231	61A	EECS	1-2MWF	1 Pimentel	S	2004
21229	61B	EECS	11-12MWF	155 Dwinelle	F	2003
21232	61B	EECS	1-2MWF	2050 VLSB	S	2004
21103	54	Math	1-2MWF	2050 VLSB	F	2003

Figure 3: Sample Table2, schedule

enrolled

SID	CCN	Grade
101	21228	B
101	21105	B+
101	21232	A-
101	21001	B

Figure 4: Sample Table3, enrolled

The main contributions are summarized as follows:

1) We implement the Database, Condition, CommandInterpreter, Row, Table class.

2) We added a lot of TestCases to test the performance of the application and made a classification.

3) We expanded the base application, adding additional features and a more interactive UI.

3 Code Environment

3.1 Dependencies

3.1.1 JDK Version

JDK 17

3.1.2 Junit Version

junit-4.13.2

3.1.3

3.2 How to Run

In terminal, type 'make' in the directory "project-genshin-teamdb61b", then type **java db61b.Main** in the directory "project-genshin-team".

4 Basic Functions

4.1 Database

The function first creates an empty database. Then we can use the get function to enter the table name to quickly find the return table or null, or use the put function to legally change the table name.

```
// The main function in Database
    public Table get(String name) {
        if ( _database.containsKey(name) ){
            return _database.get(name);
        }
        return null;
    }

    public void put(String name, Table table) {
        if (name == null || table == null) {
            throw new IllegalArgumentException("null argument");
        }
```

4.2 Condition

We first read the user's symbol input, and then judge the logical symbol correspondence of input to get the target to build the relationship expression and find the user. A Condition representing COL1 relation 'VAL2', where COL1 is a column designator, VAL2 is a literal value (without the quotes), and relation is one of the strings "<", ">", "<=", ">=", "=", or "!=".

```
// The main function in Condition
    boolean test(Row... rows) {
        String c1, c2;
        c1 = _col1.getFrom(rows);
        if (_col2 != null) {
            c2 = _col2.getFrom(rows);

        }
        else{
            c2 = _val2;
        }

        Pattern pattern = Pattern.compile("-?\\d+(\\.\\d+)?");
        int result = -1;
        if (pattern.matcher(c1).matches()) {
            if (c1.contains(".") || c2.contains(".")) {
                result = Double.compare(Double.parseDouble(c1),
                    Double.parseDouble(c2));
            } else {
                result = Integer.compare(Integer.parseInt(c1),
                    Integer.parseInt(c2));
            }
        } else {
            result = c1.compareTo(c2);
        }
        if (_relation.equals("<")) {
            return (result == -1);
        }
        if (_relation.equals(">")) {
            return (result == 1);
        }
        if (_relation.equals("<=")) {
            return (result <= 0);
        }
```

```
33          if (_relation.equals(">=")) {
34              return (result >= 0);
35          }
36          if (_relation.equals("=")) {
37              return (result == 0);
38          }
39          if (_relation.equals("!=")) {
40              return (result != 0);
41          }
42          if (_relation.equals("notIn")) {
43              return (result != 0);
44          }
45          if (_relation.equals("in")){
46              return (result == 0);
47          }
48          return false;
49      }
```

4.3 CommandInterpreter

We create an interpreter that uses the recursive descent principle, converts the bnf syntax, unties the input side, and uses tokens to create statements from the remaining sequences and perform operations, returning values. Specifically, we prepare table(e.g.create, exit, insert, load, store, print, select) for the statement. In summary, in this section, we will identify the various forms of instructions entered by the user and convert them into functional code modules.

```
// Some functions in CommandInterpreter
    /** Parse and execute a load statement from the token stream. */
    void loadStatement() {
        _input.next("load");
        String name_buffer=this.name();
        Table table_buffer=Table.readTable(name_buffer);
        _database.put(name_buffer,table_buffer);
        System.out.printf("Loaded %s.db%n",name_buffer);
        _input.next(";");
    }

    /** Parse and execute a store statement from the token stream. */
    void storeStatement() {
        _input.next("store");
        String name = _input.peek();
        Table table = tableName();
        table.writeTable(name);
        System.out.printf("Stored %s.db%n", name);
        _input.next(";");
    }

    /** Parse and execute a print statement from the token stream. */
    void printStatement() {
        _input.next("print");
        String tableName = _input.peek();
        Table table_buffer=tableName();
        _input.next(";");
        System.out.printf("Contents test of %s:%n", tableName);
        table_buffer.print();
    }
```

4.4 Row

We create a class named row. Users can set and update the value of the row through the function, and return the relevant information of the row (e.g.number,value). We also added equals function to judge by hashcode. We implement the Row(List<Column> columns, Row... rows) constructor.

```
// Equal functions in Row
   public boolean equals(Object obj) {
       if (!(obj instanceof Row objRow)) {
           return false;
       }

       if (this.hashCode() != objRow.hashCode()) {
           return false;
       }

       if (_data.length != objRow.size()) {
           return false;
       }

       for (int i = 0; i < this.size(); i++) {
           if (!_data[i].equals(objRow.get(i))) {
               return false;
           }
       }

       return true;
   }
```

4.5 Table

This function contains the basic functions associated with Table. Users can find and return labels, values, and so on for specified rows and columns. We set up the function of reading and output table, and print the search results in standard form; At the same time, the filter table function is set to filter the tables that meet the requirements based on the row and column values.

```
// Select functions in Table
    Table select(Table table2, List<String> columnNames,
                List<Condition> conditions) {
        Table result = new Table(columnNames);
        List<Column> temp_columns = new ArrayList<Column>();
        Iterator<String> it_columnNames = columnNames.iterator();
        while(it_columnNames.hasNext()){
            String temp_name = it_columnNames.next();
            temp_columns.add(new Column(temp_name, this, table2));
        }

        Iterator<Row> it_rows_1 = this.iterator();
        while(it_rows_1.hasNext()){
            Iterator<Row> it_rows_2 = table2.iterator();
            Row temp_row_1 = it_rows_1.next();
            while(it_rows_2.hasNext()){
                Row temp_row_2 = it_rows_2.next();
                if(Condition.test(conditions, temp_row_1, temp_row_2)){
                    result.add(new Row(temp_columns, temp_row_1,
                        temp_row_2));
                    break;
                }
            }
        }
        return result;
    }
```

5 Project Extension

5.1 Structure Improvement

We would like to add some extensions of our own ideas to the features required by the project.

- **GUI** We try to create a fancy and user-friendly GUI that makes it easy for users to select the features they need, with helpful tips (e.g. Show possible paths as users type).
- **Basic mathematical functions** We improve the program to support functions "max", "min", "avg", "count" and "sum" to calculate the statistical information of rows and columns

```
> select max(SID), count(Major) from students;
Search results:
+--------+--------+
| SID    | Major  |
+--------+--------+
| 106    | 6      |
+--------+--------+
```

Figure 5: Basic mathematical functions

- **Group by** We set more signature information(e.g.primary key) for the table, make it easier for the user to find. We will also try to add sort to rows and columns. In addition, we will also try to incorporate group statistics features such as group by.

```
//an example of implementing group by, order by, and function calls
        System.out.println("Search results:");
        Table table = selectClause();
        ArrayList<LinkedHashSet<Row>> groupRow = new
            ArrayList<LinkedHashSet<Row>>();
        boolean whetherGroup = false;
        String groupColumnName = "";
        if (_input.nextIf("group")) {
            if (_input.nextIf("by")) {
                groupColumnName = _input.next();
                whetherGroup = true;
                groupRow = table.group(groupColumnName);
            } else {
                throw error("The correct syntax should group by
                    <attr>");
            }
        }
```

```
> select max(YearEnter),Major from students group by Major order by Major;
Search results:
+--------------+--------+
| YearEnter    | Major  |
+--------------+--------+
| 2004         | EECS   |
| 2004         | LSUnd  |
| 2003         | Math   |
+--------------+--------+
```

Figure 6: Group by

- **Multi-select** In order to get the table with the required information step by step, we will expand on the basis of select so that the value returned by the lookup can be multi-tiered and step-by-step.

```
> load students;
Loaded students.db
> select * from students;
Search results:
+--------+--------------+--------------+--------------+--------------+--------+
| SID    | Lastname     | Firstname    | SemEnter     | YearEnter    | Major  |
+--------+--------------+--------------+--------------+--------------+--------+
| 102    | Chan         | Valerie      | S            | 2003         | Math   |
| 103    | Xavier       | Jonathan     | S            | 2004         | LSUnd  |
| 104    | Armstrong    | Thomas       | F            | 2003         | EECS   |
| 105    | Brown        | Shana        | S            | 2004         | EECS   |
| 106    | Chan         | Yangfan      | F            | 2003         | LSUnd  |
| 101    | Knowles      | Jason        | F            | 2003         | EECS   |
+--------+--------------+--------------+--------------+--------------+--------+
>
```

Figure 7: Multi-select

5.2 Structure Drawbacks

The basic structure of this structure can be improved, too.

- **Error Detect** The command interpreter uses some limited patterns to match input commands. This structure makes it difficult to deal with errors. Any unexpected input will block the whole program and we have to restart. Otherwise, it will cause unexpected output.

- **Underlying Data Storage** The project was designed to use hash sets to store rows and tables. This makes the project simpler but also reduces efficiency. Maybe we can turn to B+ tree for further exploration.

6 Presentation

Video Link:
https://www.bilibili.com/video/BV1ZG4y1J7k9/?vd_source=002be158841360a7c32c6b138889ea4b

CSC3170 Project_1 Option_3

Database Management System (DBMS)

Genshin Team

陈飞飏 | 陈宣文 | 朱骆锴 | 陶震
李卓毅 | 王奕文 | 王雨奇 | 肖玮钊

CONTENTS

1 **Background**

What is DBMS?

2 **Intro**

How should we implement a DBMS?

3 **Functions**

What functions have we implemented?

4 **Extensions**

What sort of robustness and enhancement we've done?

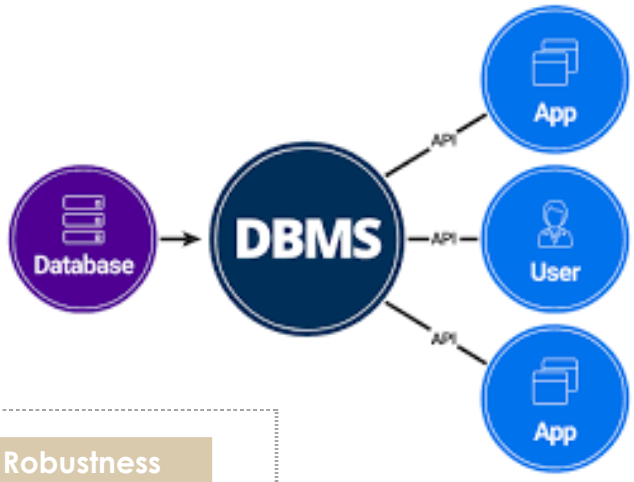
5 **Conclusion**

What we can summarize and learn from this project?

Background

What is DBMS ?

A database management system (DBMS) is system software for **creating and managing databases**. A DBMS makes it possible for end users to **create, protect, read, update and delete data** in a database.



Work to be done

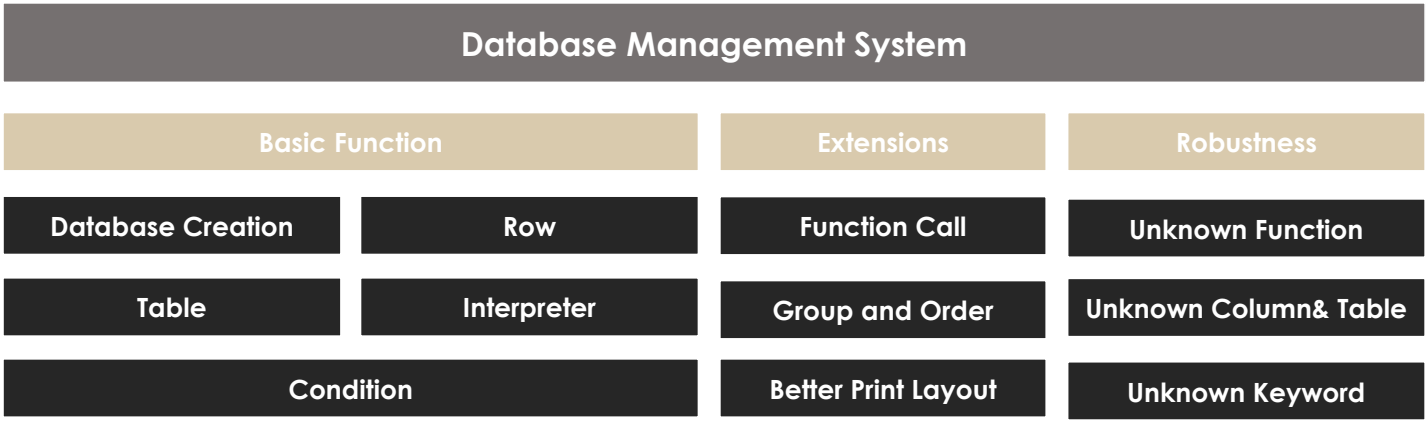
- System Implementation
- Query Language Design
- Robustness & Extensions

Introduction

Our Goal

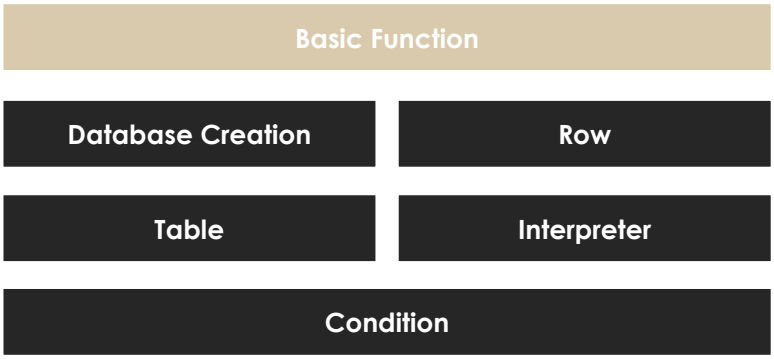
Design a **well-implemented** miniature relational database management system with **necessary extensions** and **great robustness**.

System framework



Functions

Real machine demonstration



Extensions & Robustness

Real machine demonstration

Extensions	Robustness
Function Call	Unknown Function Type
Group and Order	Unknown Column and Table
Better Print Layout	Unknown Keyword

Conclusion

What else can we do?

- Nice GUI
- Other Complementary Functions
- ……..

What we've learned?

- Cooperation of software within a database management system
- How query language are processed and understood by System
- Using Github
- Teamwork is crucial
- Hard work!
- ……..

Q&A

We are Team Genshin!

Ask us anything about our project !