



miniSQL 系统设计报告

组员：张小爽（3110104779）

许 多（3110000090）

王哲峰（3110000026）

班级：计科 1101 班

指导老师：孙建伶

助教：金鑫

时间：2013-11-13

提要

本系统设计并实现了一个精简型单用户 SQL 引擎(DBMS)MiniSQL, 允许用户通过字符界面输入 SQL 语句实现表的建立/删除; 索引的建立/删除以及表记录的插入/删除/查找。经过严格的测试, 证明本系统可以在 windows 平台下成功编译并顺利运行。本报告详细说明了该系统的外部接口和内部实现原理, 为该系统日后的升级和维护按照软件工程方法留下了比较详尽的文档资料。

目录

第一章 问题描述.....	4
第二章 需求分析.....	4
2.1 需求概述.....	4
2.2 系统功能具体实现要求(语句)	5
第三章 设计方法.....	8
3.1 系统体系结构.....	8
3.2 模块概述.....	8
Interpreter	8
API	9
Catalog Manager	9
Record Manager.....	9
Index Manager	10
Buffer Manager	11
DB Files.....	11
第四章 功能实现.....	12
4.1 创建表.....	12
4.2 插入数据.....	13
4.3 执行 SQL 脚本	13
4.4 选择语句的执行.....	14
4.5 索引的创建.....	15
4.6 索引的删除.....	16
4.7 删除语句.....	16
4.8 表的删除.....	16
4.9 退出:	16
第五章 系统分析.....	17
5.1 功能亮点.....	17
5.2 系统不足.....	17
第六章 分工&申明.....	17

第一章 问题描述

随着现代社会的信息量不断增加，对各类数据库的需求日益加大，如何有效管理庞大的数据信息成为了摆在人们面前的一大难题。作为一名计算机专业的学生，在学习过数据库系统设计这门课程之后，要将所学到的知识学以致用，来解决生活中的问题。

本系统正是基于这样一个目的所设计的，它是一个精简型单用户SQL引擎，可以使用户通过字符界面输入SQL语句实现表的建立/删除；索引的建立/删除以及表记录的插入/删除/查找。通过该数据库系统的设计与实现，可以提高学生的系统编程能力，加深对数据库系统原理及应用的理解。

第二章 需求分析

2.1 需求概述

2.1.1 数据类型

支持四种基本数据类型: int, char(n), float, date, 其中 char(n) 满足 $1 \leq n \leq 255$ 。

2.1.2 表定义

一个表最多可以定义 32 个属性，各属性可以指定是否为 unique；支持单属性的主键 (Primary Key) 定义。

2.1.3 索引的建立和删除

对于表的主属性自动建立 B+树索引，对于声明为 unique 的属性可以通过 SQL 语句由用户指定建立/删除 B+树索引（因此，所有的 B+树索引都是单属性单值的）。

2.1.4 查找记录

可以通过指定用 and 连接的多个条件进行查询，支持等值查询和区间查询。实现基本连接功能，不实现外连接（不支持 NULL 值）支持集函数查询。

2.1.5 插入和删除记录

支持每次一条记录的插入操作；支持每次一条或多条记录的删除操作。

2.2 系统功能具体实现要求(语句)

2.2.1 创建表语句

该语句的语法如下：

```
create table 表名 (  
    列名 类型 ,  
    列名 类型 ,  
  
    列名 类型 ,  
    primary key ( 列名 )  
);
```

若该语句执行成功，则输出执行成功信息；若失败，告诉用户失败的原因。

示例语句：

```
create table student (  
    sno char(8) ,  
    sname char(16) unique ,  
    sage int ,  
    sgender char (1) ,  
    primary key ( sno )  
);
```

2.2.2 删除表语句

该语句的语法如下：

```
drop table 表名 ;
```

若该语句执行成功，则输出执行成功信息；若失败，告诉用户失败的原因。

示例语句：

```
drop table student;
```

2.2.3 创建索引语句

该语句的语法如下：

```
create index 索引名 on 表名 ( 列名 );
```

若该语句执行成功，则输出执行成功信息；若失败，告诉用户失败的原因。

示例语句：

```
create index stunameidx on student ( sname );
```

2.2.4 删除索引语句

该语句的语法如下：

```
drop index 索引名 ;
```

若该语句执行成功，则输出执行成功信息；若失败，告诉用户失败的原因。

示例语句：

```
drop index stunameidx;
```

2.2.5 选择语句

该语句的语法如下：

```
select * from 表名 ;
```

或：

```
select * from 表名 where 条件 ;
```

其中“条件”具有以下格式：列 op 值 and 列 op 值 ... and 列 op 值。

op 是算术比较符：= <> < > <= >=

若该语句执行成功且查询结果不为空，则按行输出查询结果，第一行为属性名，其余每一行表示一条记录；若查询结果为空，则输出信息告诉用户查询结果为空；若失败，告诉用户失败的原因。

示例语句：

```
select * from student;
```

```
select * from student where sno = '88888888';
```

```
select * from student where sage > 20 and sgender = 'F';
```

2.2.6 插入记录语句

该语句的语法如下：

```
insert into 表名 values ( 值1 , 值2 , ... , 值n );
```

若该语句执行成功，则输出执行成功信息；若失败，告诉用户失败的原因。

示例语句：

```
insert into student values ('12345678','wy',22,'M');
```

2.2.7 更新语句

该语句的语法如下：

```
update 表名 set 列名=值 where 条件 ;
```

若该语句执行成功，则输出执行成功信息；若失败，告诉用户失败的原因。

示例语句：

```
update student set sno = '12345678' where sname = 'zt';
```

2.2.8 删除记录语句

该语句的语法如下：

```
delete from 表名 ;
```

或：

```
delete from 表名 where 条件 ;
```

若该语句执行成功，则输出执行成功信息，其中包括删除的记录数；若失败，告诉用户失败的原因。

示例语句：

```
delete from student;
```

```
delete from student where sno = '88888888';
```

2.2.9 退出 MiniSQL 系统语句

该语句的语法如下：

```
quit;
```

2.2.10 执行 SQL 脚本文件语句

该语句的语法如下：

```
execfile 文件名 ;
```

SQL 脚本文件中可以包含任意多条上述 SQL 语句，MiniSQL 系统读入该文件，然后按序依次逐条执行脚本中的 SQL 语句。

第三章 设计方法

3.1 系统体系结构

根据以上功能需求，MiniSQL 体系结构如下：

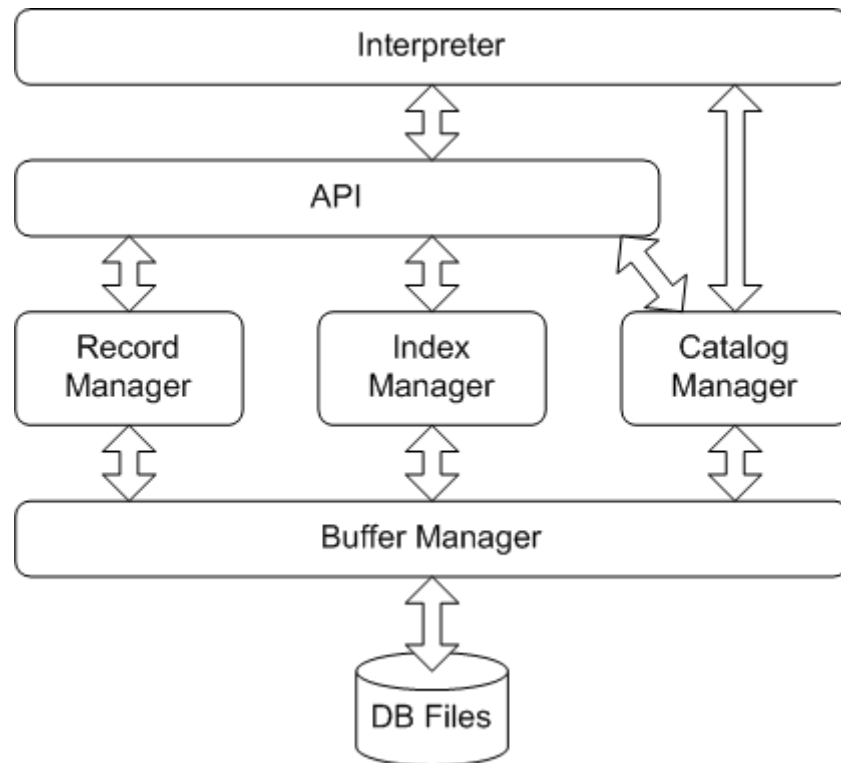


图 1 MiniSQL 体系结构

从文件底层开始，miniSQL 总体架构分为 DB files, Buffer, Record/Index/Catalog Manager, API 和 Interpreter。在实验指导书的 miniSql 体系结构基础上，我们做了以下修改：

1. Catalog Manager 中对于文件的存取不通过 Buffer Manager，而是独立对 catalog file 进行维护，在构造函数中将信息读取出来放入缓存，析构函数中再将缓存中的内容写回文件。这样设计可以将 Catalog 模块独立出来作为一个专门的模块，更加便于维护。
2. 为了将数据表等设计得更加 OO，我们添加 global 模块，用于存放 table、attribute、tuple、exception、condition（筛选条件）、query（语句信息）等类的结构和简单操作函数，以及 enum 类型的 DATA_TYPE、OP_TYPE、QUERY_TYPE 作为全局变量，以满足实验指导书中对数据类型、选择操作判断条件、语句类型的支持。

3.2 模块概述

Interpreter

Interpreter 模块实现与用户交互，主要实现以下功能：

1. 程序流程控制，即“启动并初始化 → ‘接收命令、处理命令、显示命令结果’循环 → 退出”流程。
2. 接收并解释用户输入的命令，生成命令的内部数据结构表示，同时检查命令的语法正确性和语义正确性，对正确的命令调用 API 层提供的函数执行并显示执行结果，对不正确的命令显示错误信息。

本模块的主要设计方法为字符串的分析及处理。通过字符串流 `strstream` 以空格为分割，依次读取每一个词进行分析，获取相关参数，并调用相关的 API。

API

API 模块是整个系统的核心，主要功能是提供执行 SQL 语句的接口，供 Interpreter 层调用。该接口以 Interpreter 层解释生成的内部命令作为输入，根据 Catalog Manager 提供的表信息确定执行规则，并调用 Record Manager、Index Manager 和 Catalog Manager 提供的相应接口进行执行，最后返回执行结果给 Interpreter 模块。

Catalog Manager

Catalog Manager（模式信息管理器，下称 CM）负责管理数据库的所有模式信息，包括：

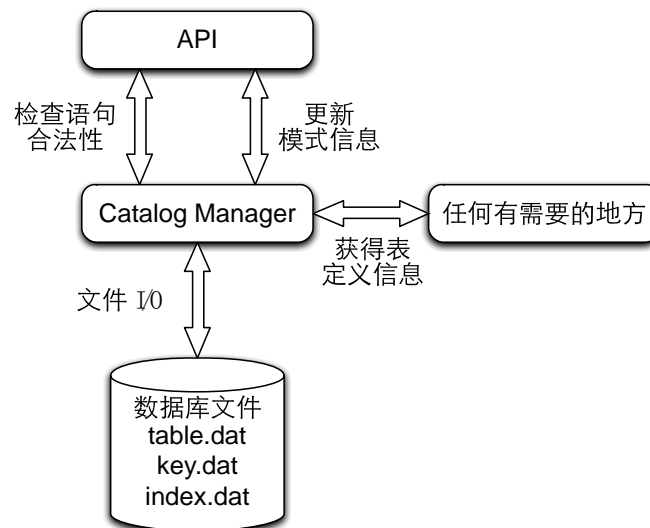
1. 数据库中所有表的定义信息，包括表的名称、表中字段（列）数、主键、定义在该表上的索引。
2. 表中每个字段的定义信息，包括字段类型、是否唯一等。
3. 数据库中所有索引的定义，包括所属表、索引建立在那个字段上等。

CM 还提供了访问及操作上述信息的接口，供 Interpreter 和 API 模块使用。此外，CM 还可以向 MiniSQL 的其他模块提供表定义等信息。

在初始化阶段，CM 会读取 `catalog` 目录下的模式信息文件，把表定义信息、字段定义信息和索引定义信息读入内存中的 `tableCatalog`、`keyCatalog` 和 `indexCatalog` 中。为了考虑效率，之后的更新模式信息操作都在内存中完成（在最后的析构阶段回写文件）。由于模式信息需要维护的文件远小于保存表记录的文件，故未使用缓冲区（buffer）来间接操作文件，而选择直接操作文件。

CM 的语句检查函数和更新模式信息函数均由上层的 API 模块调用。对于语句检查函数，如果合法性满足模式信息的要求，这些函数不返回值，否则函数抛出异常信息。而对于更新模式信息函数，调用后会对内存中的数据进行相应修改。MiniSQL 的其他部分在需要表定义等信息的时候，通过调用 CM 的相应函数就可以得到所需的信息。

CM 与 MiniSQL 的其他部分之间的关系结构如图所示。



Record Manager

Record Manager 负责管理记录表中后缀为“.table”的数据文件。数据文件由一个或多个数据块组成，块大小应与缓冲区块大小相同。一个块中包含一条至多条记录，为简单起见，只支持定长记录的存储，且不支持记录的跨块存储。

主要功能如下：

1. 实现数据文件的创建与删除
2. 记录的插入
3. 记录的删除
4. 记录的查找。分为有 index 和没有 index（遍历）两种查找模式，能够支持不带条件的查找和多个条件的查找（包括等值查找、不等值查找和区间查找）。

Record Manager 向上对 API 提供相应的调用接口，向下通过调用 Buffer Manager 的函数，实现与 Buffer 中块信息的交互。

Index Manager

Index Manager 负责 B+树索引的实现，负责管理后缀为“.index”的，实现 B+树的创建和删除（由索引的定义与删除引起）、等值查找、插入键值、删除键值等操作，并对外提供相应的接口。

B+树中节点大小与块大小相同，B+树的叉数由节点大小与索引键大小计算得到。

每一个结点（块），存放若干分支点，每一个点存放当前列值，所在块，块中位置，以及一个指针值。

Buffer Manager

Buffer Manager 负责内存中数据缓冲区的管理，主要思想是在缓冲区中建立与文件系统交互的块，块的大小应为文件系统与磁盘交互单位的整数倍，我们将它定为 4KB。

Buffer Manager 的主要功能有：

1. 读取指定文件指定块的数据到 buffer 中，以及将 buffer 中的数据写回到指定文件的指定位置
2. 实现缓冲区的 LRU 替换算法，当缓冲区满时选择合适的页进行替换
3. 提供缓冲区页的锁定功能，不允许被锁定的块被替换出去

DB Files

DB Files 指构成数据库的所有数据文件，主要由记录数据文件(“.table”文件)、索引数据文件(“.index”文件)和 Catalog 数据文件组成，分别存放在不同的文件夹下，通过上层 buffer 对文件的操作，实现 DB 数据统一管理。

1. Record 数据记录存放的规则如下：

- a) 在每一个“.table”文件中，存放一张表的信息，支持定长的记录。
- b) 每条记录之后添加一个 1B 的标记位，占据（行的大小+1）B 的空间。多条记录均按格式顺序存放。
- c) 一个“.table”文件可以分割成若干个块，每一个块 4KB 大小，用从 0 开始顺序编码的块号作为标记。
- d) 不支持跨块存储，每个块末尾不够存放一条记录的空间被放弃，将新记录存放在下一个块中。

2. Catalog 数据记录存放的规则如下：

CM 使用三个文件对维护模式信息：table.dat、key.dat 和 index.dat。这三个文件分别存储了表定义信息、表中字段定义信息和索引定义信息，均位于名为 catalog 的子目录下。在 CM 的构造阶段，这三个文件的内容会被读入到 tableCatalog、keyCatalog、indexCatalog 这三个 vector 变量中；在析构阶段，则是从 vector 变量写回到文件。

文件使用二进制格式存储以节省空间，其中存储名称的元素以字符串的形式存储，其他元素以整数（默认是小端序）的形式存储。

a) table.dat 文件

该文件由连续的单位构成，每个单位记录一张表的定义信息，格式如下（长度的单位是字节）：

含义	标志	表名	属性(列)数	主键所在属性号	索引标志(列 i 若有索引, 则本标志的位 i 为 1)	第一条属性在 key.dat 中的下标	第一条索引在 index.dat 中的下标
长度	1	20	1	1	4	2	2

b) key.dat 文件

该文件由连续的单位构成，每个单位记录一个字段（属性、键）的定义信息，格式如下：

含义	标志	属性名	属性类型(0、1、2 分别代表整数、字符串、浮点数)	属性长度	该表下一条属性的下标(若无则置-1)
长度	1	20	1	1	2

c) index.dat 文件

该文件由连续的单位构成，每个单位记录一条索引的定义信息，格式如下：

含义	标志	索引名	索引所属表的下标	索引在所属表的属性号	该表下一条索引的下标(若无则置-1)
长度	1	20	2	1	2

第四章 功能实现

4.1 创建表

```
miniSQL>>create table student (
    sno char(8),
    sname char(16) unique,
    sage int,
    sgender char(1),
    primary key(sno)
);
Table student has been created successfully.
```

输入正确格式完成输入，可以成功执行表的创建。

```
miniSQL>>create tab student(
    a int,
    primary key(a));
Syntax error!
```

如果格式有问题会给出错误提示。

```
miniSQL>>create table student(
    sno char(8),
    sname char(16) unique,
    sage int,
    sgender char(1),
    primary key(sno));
TableAlreadyExists: The table "student" already exists.
```

如果该表已经存在则也会给出错误提示。

4.2 插入数据

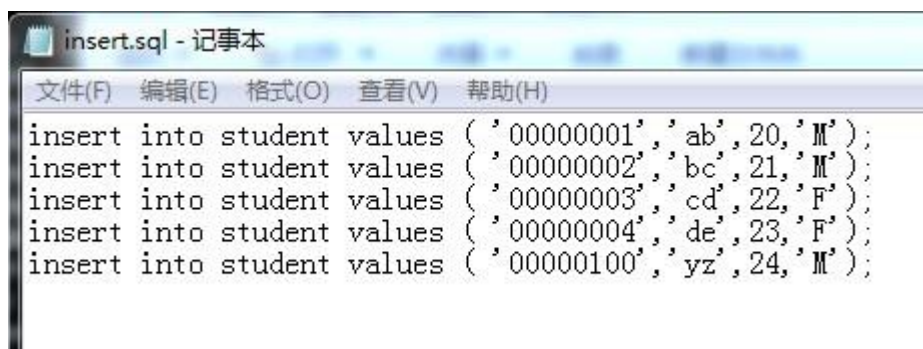
```
miniSQL>>insert into student values('12345678','wy',22,'M');  
This tuple has been inserted successfully.  
miniSQL>>insert student values('12345678','wy',22,'M');  
Syntax error!
```

```
miniSQL>>insert into stu values('12345678','wy',22,'M');  
TableNotExist: The table "stu" does not exist.  
miniSQL>>insert into student values('123456789','wy',22,'M');  
ValueIllegal: The value to insert is longer than the length of the key.
```

按照正确格式输入则可以正确对表进行插入,如果该表不存在,输入的属性信息不相符,或格式错误则都会给出相应的错误提示。

4.3 执行 SQL 脚本

测试脚本如下图所示,即对 student 表插入 5 条数据:



```
insert into student values ( '00000001','ab',20,'M');  
insert into student values ( '00000002','bc',21,'M');  
insert into student values ( '00000003','cd',22,'F');  
insert into student values ( '00000004','de',23,'F');  
insert into student values ( '00000100','yz',24,'M');
```

```
miniSQL>>execfile insert.sql;  
This tuple has been inserted successfully.  
This tuple has been inserted successfully.  
This tuple has been inserted successfully.  
This tuple has been inserted successfully.  
This tuple has been inserted successfully.
```

通过上图可以看到,系统成功读取了五条插入语句并分别执行。

4.4 选择语句的执行

```
miniSQL>>select * from student;
+-----+-----+-----+-----+
| sno    | sname    | sage | sgender |
+-----+-----+-----+-----+
| 12345678 | wy       | 22   | M       |
+-----+-----+-----+-----+
| 00000001 | ab       | 20   | M       |
+-----+-----+-----+-----+
| 00000002 | bc       | 21   | M       |
+-----+-----+-----+-----+
| 00000003 | cd       | 22   | F       |
+-----+-----+-----+-----+
| 00000004 | de       | 23   | F       |
+-----+-----+-----+-----+
| 00000100 | yz       | 24   | M       |
+-----+-----+-----+-----+
The time duration is: 0.078s
```

通过上图可以看到，可以对表进行全查找。

```
miniSQL>>select * from class;
TableNotExist: The table "class" does not exist.
```

通过上图可以看到，当表不存在时则可以给出错误信息。

```
miniSQL>>select * from student where sno = '12345678';
+-----+-----+-----+-----+
| sno    | sname    | sage | sgender |
+-----+-----+-----+-----+
| 12345678 | wy       | 22   | M       |
+-----+-----+-----+-----+
The time duration is: 0.063s
```

通过上图可以看到，可以对表进行单条件等号查找。

```
miniSQL>>select * from student where sno <> '12345678';
+-----+-----+-----+-----+
| sno    | sname    | sage | sgender |
+-----+-----+-----+-----+
| 00000001 | ab       | 20   | M       |
+-----+-----+-----+-----+
| 00000002 | bc       | 21   | M       |
+-----+-----+-----+-----+
| 00000003 | cd       | 22   | F       |
+-----+-----+-----+-----+
| 00000004 | de       | 23   | F       |
+-----+-----+-----+-----+
| 00000100 | yz       | 24   | M       |
+-----+-----+-----+-----+
The time duration is: 0.078s
```

通过上图可以看到，可以对表进行单条件不等号查找。

```
miniSQL>>select * from student where sage > 20 and sgender = 'F';
+-----+-----+-----+-----+
| sno    | sname  | sage | sgender |
+-----+-----+-----+-----+
| 00000003 | cd     | 22   | F       |
+-----+-----+-----+-----+
| 00000004 | de     | 23   | F       |
+-----+-----+-----+-----+
The time duration is: 0.062s
```

通过上图可以看到，可以对表进行多条件查找。

4.5 索引的创建

```
miniSQL>>create index stunameidx on student (sname);
Index stunameidx has been created successfully.
```

索引成功创建。

```
miniSQL>>select * from student where sname = 'wy';
+-----+-----+-----+-----+
| sno    | sname  | sage | sgender |
+-----+-----+-----+-----+
| 12345678 | wy     | 22   | M       |
+-----+-----+-----+-----+
The time duration is: 0.031s
```

通过索引可以成功查找到相关元组，并且查找时间缩短近一倍。

```
miniSQL>>insert into student values('00000101','aa',22,'F');
This tuple has been inserted successfully.
```

添加新的元组以测试索引的插入。

```
miniSQL>>select * from student where sname = 'aa';
+-----+-----+-----+-----+
| sno    | sname  | sage | sgender |
+-----+-----+-----+-----+
| 00000101 | aa     | 22   | F       |
+-----+-----+-----+-----+
The time duration is: 0.032s
```

上图证明索引插入成功，新的元组也可以成功被索引查找。

```
miniSQL>>delete from student where sname = 'aa';
This tuple has been deleted successfully.
```

删除元组以测试索引的删除功能。

```
miniSQL>>select * from student where sname = 'aa';
There is no such tuple in this table.
The time duration is: 0.016s
```

上图证明索引删除成功。

4.6 索引的删除

```
miniSQL>>drop index stunameidx;  
Index stunameidx has been dropped successfully.
```

索引被成功删除

```
miniSQL>>select * from student where sname = 'wy';  
+-----+-----+-----+-----+  
| sno      | sname      | sage | sgender |  
+-----+-----+-----+-----+  
| 12345678 | wy         | 22   | M       |  
+-----+-----+-----+-----+  
The time duration is: 0.078s
```

Select 语句恢复为遍历查找，查找时间增加。

4.7 删除语句

```
miniSQL>>delete from student where sno = '12345678';  
This tuple has been deleted successfully.
```

```
miniSQL>>select * from student where sno = '12345678';  
There is no such tuple in this table.  
The time duration is: 0.047s
```

上图可以说明元组可以被成功删除。

```
miniSQL>>delete from student;  
This tuple has been deleted successfully.  
miniSQL>>select * from student;  
There is no such tuple in this table.
```

上图可以说明表中所有数据也可以被一并成功删除。

4.8 表的删除

```
miniSQL>>drop table student;  
Table student has been dropped successfully.  
miniSQL>>select * from student;  
TableNotExist: The table "student" does not exist.
```

整张表被成功删除，当再次查找该表时给出该表不存在的错误信息。

4.9 退出:

```
quit;
```

程序顺利退出。

第五章 系统分析

5.1 功能亮点

- 界面简洁美观，具有很好的用户体验。
- 程序架构清晰，各模块之间有明确的接口，便于日后维护。
- 在每次出错后都会给出详细的错误信息。
- 在每次查询的结尾都会给出本次查询所用时间，便于更好地进行测试。
- 经测试，在一千组数据量下程序也可以完美运行。
- ...

5.2 系统不足

由于本系统的开发时间较短，小组成员水平有限，相互之间配合不是很默契，对相关知识了解不足，导致了一下几点不足：

- 解析器对于某些错误输入不能够进行准确的识别。
- 系统整体查询速度较慢，还需进一步优化。
- 必须要正常输入quit退出后缓存中的内容才能被成功写入文件，如果直接关闭则缓存内容将全部丢失。
- 在插入大规模数据的时候的插入时间过于缓慢，主要是B+树的节点插入部分，还需在保证正确性的前提下进行优化。
- ...

第六章 分工

张小爽：	Record Manager, Buffer Manager, DB files
许多：	Catalog Manager, Index Manager, DB files
王哲峰：	API, Interpreter, Index Manager, DB files