MiniSQL 总体设计报告

1 MiniSQL 系统概述

1.1 背景

1.1.1 编写目的

本学期数据库系统原理的综合实验要求设计并实现一个单用户 SQL 数据库：MiniSQL，要求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
- 表记录的插入/删除/查找

本项目的编写即是为了完成该项任务。

1.1.2 项目背景

为全面提高学生创新和实践能力，浙江大学数据库系统原理课程分为课堂教学和综合性实验两部分。综合性实验采取分组形式完成，每 3~4 个学生为 1 组，分别设有组长、主程序员、程序员、测试员、文档员等角色，全面锻炼学生的系统设计与实现能力、系统编程能力、测试能力、组织文档能力、以及团队合作能力，并加深学生对数据库管理系统事先技术的理解。

综合性实验要求设计并实现一个单用户 SQL 数据库：MiniSQL，实验内容贯穿全部重要知识点，通过 2 个步骤的阶段性验收，锻炼学生综合运用每个环节所学知识解决实际问题的能力。

1.2 功能描述

1.2.1 表

- 一个表最多可以定义 32 个属性
- 各属性可以指定是否为 unique
- 支持单属性的主键定义

1.2.2 索引

- 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
- 所有 B+ 树索引都是单属性单值

1.2.3 数据类型

支持以下数据类型：
- integer
- char(n) (1 <= n <= 256)
- float

1.2.4 数据操作

- 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
- 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

1.2.5 SQL 语句支持：

- 本项目预计将支持多数常见的标准 SQL 语句，包括但不仅限于：
  - ✧ 创建表和删除表
  - ✧ 创建索引和删除索引
  - ✧ 选择，插入，删除
  - ✧ 退出系统
  - ✧ 执行 SQL 脚本
- 支持的关键字包括但不仅限于：
  - ✧ select
  - ✧ from
  - ✧ where
  - ✧ natural join
  - ✧ join using
  - ✧ distinct
  - ✧ as
  - ✧ order by
  - ✧ union
  - ✧ intersect
  - ✧ except
  - ✧ avg
  - ✧ min
  - ✧ max
  - ✧ sum
  - ✧ count
  - ✧ group by
  - ✧ having
  - ✧ some
  - ✧ all
  - ✧ exists
  - ✧ unique
  - ✧ < = >
  - ✧ char
  - ✧ integer
  - ✧ float
  - ✧ create table
  - ✧ primary key
  - ✧ foreign key
  - ✧ references
  - ✧ not null
  - ✧ insert into values
  - ✧ delete from
  - ✧ drop table
  - ✧ update set
- 每一条 SQL 可以一行或多行，以分号结尾，要求其中关键字均为小写。
- 每一条 SQL 语句最多支持一层嵌套。
- 不支持定义视图，事务，函数，触发器等 SQL 高级特性。

1.3 运行环境和配置

项目在 Linux 平台下采用 C/C++编写，通过 g++编译，故优先支持 Linux 平台。但预计同样可在 Windows/Mac OS 等支持 C++编译运行的平台运行。

项目编写过程中采用校内自建 Git 服务器进行版本管理和团队协作。

项目预计仅使用 C++标准库，故无需安装配置第三方库。

1.4 参考资料

- 数据库系统概念 / （美）Abraham Silberschatz 等著；杨冬清等译. —北京：机械工程出版社， 2012.11
- C++ Primer 中文版 / （美）Lippman,S.B 等著；王刚等译. —北京：电子工程出版社，2013.9

2 MiniSQL 系统结构设计

2.1 总体设计

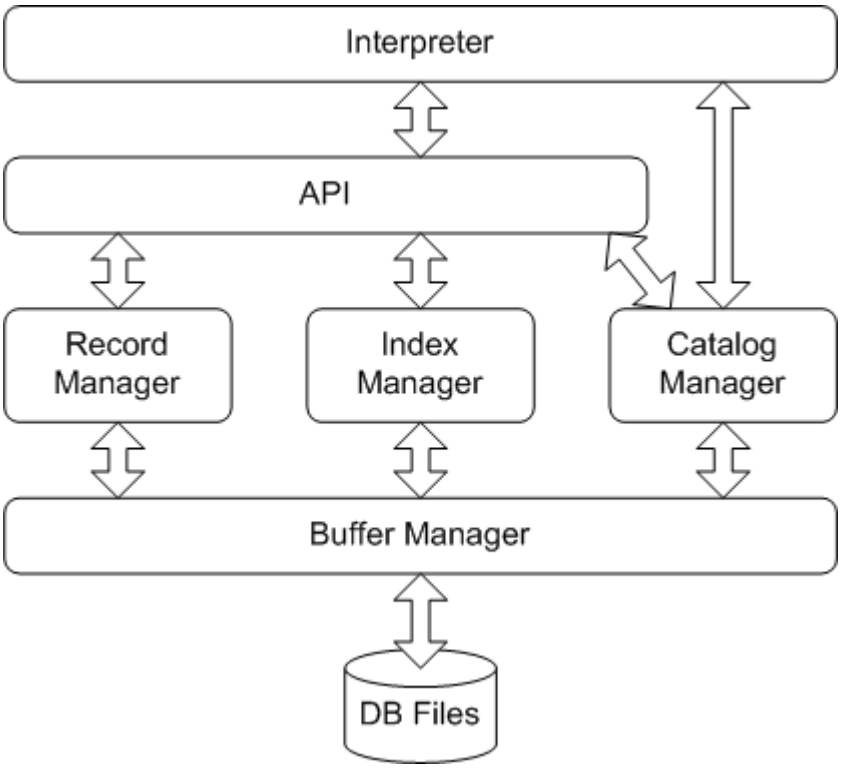

图 1 总体设计

2.2 Interpreter 模块

Interpreter 模块直接与用户交互，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。

Interpreter 模块负责接受命令，接收到命令后解释器会将命令格式化构建成相应的对象，对象中会保存再进一步处理时需要的数据，包括表名、条件等。之后根据对象的类型将该对象传入 API 模块中的函数进行相应的处理。如果这个过程中出现 SQL 语言的语法错误，程序将抛出一个异常，以便提示用户出现了异常。

该模块中将使用以下类来处理 SQL 语言。

- UseDatabase
- ShowDatabase
- CreateDatabase
- DropDatabase
- ShowTables
- CreateTable
- DropTable
- CreateIndex
- DropIndex
- Selcet
- Insert
- Delete
- Update

2.3 API 模块

API 模块是整个系统的核心，其主要功能为提供执行 SQL 语句的接口，供 Interpreter 层调用。该接口以 Interpreter 层解释生成的命令内部表示为输入，根据 Catalog Manager 提供的信息确定执行规则，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。

在这个模块中将为 Create,Drop,Insert 等在 Interpreter 中的不同类提供处理接口。根据不同类型的语句，API 模块将调用 Record Manager、Index Manager 和 Catalog Manager 提供的相应接口进行处理。例如要创建一个表，在 API 模块的 CreateTable 函数中，要调用 Record Manager 中的函数来创建文件，并调用 Catalog Manager 中的对应函数把表的属性写入文件中。

2.4 CatalogManager 模块

Catalog Manager 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。

Catalog Manager 还必需提供访问及操作上述信息的接口，供 Interpreter 和 API 模块使用。

这一模块中应包含以下类：

- Database
- Table
- Index
- Attribute

在这些类中将提供 Create, Drop, Use, Show 等函数来处理相应的命令，以便 Interpreter 和 API 模块调用；同时提供接口来访问各种模式信息。

2.5 RecordManager 模块

Record Manager 主要功能有数据文件的创建与删除，记录的插入、删除与查找操作，并对外提供相应的接口。这一模块与 API 和 Buffer Manager 连接。

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，只要求支持定长记录的存储，且不要求支持记录的跨块存储。

这一模块用 RecordManager 类来实现，其中应该包含 Insert, Select, Update, Delete, DeleteRecord, UpdateRecord 等组成部分。

2.6 IndexManager 模块

Index Manager 负责 B+树索引的实现，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+树中节点大小应与缓冲区的块大小相同，B+树的叉数由节点大小与索引键大小计算得到。

这一模块中应包含以下类：

- BPlusTreeNode
- BPlusTree
- IndexManager

2.7 BufferManager 模块

Buffer Manager 负责缓冲区的管理，主要功能有：

1. 根据需要，读取指定的数据到系统缓冲区或将缓冲区中的数据写出到文件
2. 实现缓冲区的替换算法，当缓冲区满时选择合适的页进行替换
3. 记录缓冲区中各页的状态，如是否被修改过等
4. 提供缓冲区页的 pin 功能，及锁定缓冲区的页，不允许替换出去

为提高磁盘 I/O 操作的效率，缓冲区与文件系统交互的单位是块，块的大小应为文件系统与磁盘交互单位的整数倍，一般可定为 4KB 或 8KB。

此模块包含类 BufferManager，具体的函数有 getBlock 和 writeBlock。

3 测试方案和测试样例

3.1 测试方案

测试系统：Ubuntu 16.04

测试目标：核实 MiniSQL 能够实现预期功能并在大数据读取输入时能够稳定运行

工具与方法：手工进行黑盒测试，批量产生大数据进行压力测试

3.1.1 黑盒测试

主要包括功能测试和语法测试。主要试图发现下列几类错误：功能不正确或遗漏、输入和输出错误、性能错误和初始化及终止错误。

功能测试：为了检测程序的正确性和鲁棒性，我们将通过给出正确的语法来测试以下功能：

- show databases
- use database
- create database
- drop database
- show tables
- create table
- drop table
- create index
- drop index
- select
- insert
- delete
- update

- quit
- execfile

语法测试：主要目的是保证在用户输入错误的命令时，系统不会因无法处理命令而崩溃。上一部分对数据库的功能进行测试，以保证系统的正确性，这一部分对语法进行测试，以保证系统在处理错误输入时能正确输出。其中语法测试包括两个部分：关键字和命令格式。关键字测试主要检查在输入错误的关键字时，系统能否识别出语法错误。由于该 MiniSQL 仅支持小写命令，故输入大写的命令时系统也将输出错误；命令格式测试主要检查在输入错误的命令格式时，系统能否检测出语法错误。该部分主要检测命令的关键字位置是否正确及关键字搭配是否正确。在发生语法错误时，系统应该输出“ERROR: Syntax error”

3.1.2 压力测试

大数据无法通过手工进行单条语句输入，因此将采用用代码产生大量插入操作语句，然后在数据库中执行的方法进行压力测试。在压力测试中，我们主要测试在大批量数据的导入和读取时程序的稳定性。

在压力测试中为了充分检测程序性能，相同数据量的插入操作将分为三种：插入到同一数据库的同一表中、插入到同一数据库的不同表中、插入到不同数据库的表中。

压力测试的样例将通过代码生成，在下一节中暂时不会给出。

3.2 测试样例

3.2.1 功能测试

1. 测试目的：测试 show databases 功能

   测试代码：

```
show databases;
```

   期望结果：如果当前系统没有数据库，则输出“Error: No databases”；否则程序输出当前系统中所有数据库。

2. 测试目的：测试 use database 功能

   测试代码：

```
use <database_name>;
```

   期望结果：如果不存在该数据库，则输出“Error: No such database”；否则程序输出输出“Database changed”。

3. 测试目的：测试 create database 功能

   测试代码：

```
create database <database_name>;
```

   期望结果：如果已经存在同名数据库，则输出“Error: The database with same name exists”；否则程序输出“Create database successfully”。

4. 测试目的：测试 drop database 功能

测试代码：

```
drop database <database_name>;
```

期望结果：如果不存在该数据库，则输出“Error: No such database”；否则程序输出“Drop database successfully”。

5. 测试目的：测试 show tables 功能

测试代码：

```
show tables;
```

期望结果：如果当前未选择数据库，则输出“Error: No databases selected”；如果已选择数据库但该数据库中不存在表，则输出“Error: No tables”；如果已选择数据库且该数据库中存在表，则输出所有表的名称。

6. 测试目的：测试 create table 功能

测试代码：

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

期望结果：如果当前未选择数据库，则输出“Error: No databases selected”；如果已选择数据库但该数据库中存在同名表，则输出“Error: The table with same name exists”；如果已选择数据库且该数据库中不存在同名表表，则输出“Create table successfully”。

7. 测试目的：测试 drop table 功能

测试代码：

```
drop table <table_name>;
```

期望结果：如果当前未选择数据库，则输出“Error: No databases selected”；如果已选择数据库但该数据库中不存在该表，则输出“Error: No such table”；如果已选择数据库且该数据库中存在同名表，则输出“Drop table successfully”。

8. 测试目的：测试 create index 功能

测试代码：

```
create index <index_name> on <table_name>(<column_name>);
```

期望结果：如果当前未选择数据库，则输出“Error: No databases selected”；如果已选择数据库但该数据库中不存在该表，则输出“Error: No such table”；如果已选择数据库且该数据库中存在同名表但不存在该列，则输出“Error: No such column”;如果已经选择数据库且存在该列，但已经存在同名索引，则输出“Error: The index with same name exits”；否则，输出“Create index successfully”。

9. 测试目的：测试 drop index 功能

测试代码：

```
drop index <index_name>;
```

期望结果：如果当前未选择数据库，则输出“Error: No databases selected”；如果已选择数据库但该数据库中不存在该索引，则输出“Error: No such index”；如果已选择数据库且该数据库中存在同名索引，则输出“Drop index successfully”。

10. 测试目的：测试 select 功能

测试代码：

```
select <name> from <table _name>;
select * from <table_name> where <condition>;
//condition: <column_name> op <values> and <column_name> op <values> …
//here ‘op’ stands for ‘=’ ‘<>’ ‘<’ ‘>’ ‘<=’ ‘>=’
//maybe there are the following statements
select <name> from <table_name> group by <column_name> having <conditon>;
//<name> can be the aggregate functions, such as sum(), avg(), max(), count(), etc.
```

期望结果：如果当前未选择数据库，则输出“Error: No databases selected”；如果已选择数据库但该数据库中不存在该表，则输出“Error: No such table”；如果已选择数据库且该数据库中存在同名表，则按照实际结果输出：若查询结果为空则输出“Empty Set”；否则按照第一行为属性名，其余每一行表示一条记录的形式输出查询结果。

11. 测试目的：测试 insert 功能

测试代码：

```
insert into <table_name> values (<value1>, <value2>, …);
```

期望结果：如果当前未选择数据库，则输出“Error: No databases selected”；如果已选择数据库但该数据库中不存在该表，则输出“Error: No such table”；如果已选择数据库且该数据库中存在同名表，但值的数量与表的列数不同，则输出“Column count doesn’t match value count”;否则输出“Insert successfully”。

12. 测试目的：测试 delete 功能

测试代码：

```
delete from <table_name>;
delete from <table_name> where <condition>;
//condition: <column_name> op <values> and <column_name> op <values> …
//here ‘op’ stands for ‘=’ ‘<>’ ‘<’ ‘>’ ‘<=’ ‘>=’
```

期望结果：如果当前未选择数据库，则输出“Error: No databases selected”；如果已选择数据库但该数据库中不存在该表，则输出“Error: No such table”；如果已选择数据库且该数据库中存在同名表，则根据是否满足条件进行输出，输出格式为“delete successfully, <number> rows affected”。

13. 测试目的：测试 update 功能

测试代码：

```
update <table_name> set <column_name> = <value> where <condition>;
```

期望结果：如果当前未选择数据库，则输出“Error: No databases selected”；如果已选择数据库但该数据库中不存在该表，则输出“Error: No such table”；如果已选择数据库且该数据库中存在同名表，但无匹配列名，则输出“Error: No such column”；否则根据是否满足条件输出“Update successfully, <number> rows affected”。

14. 测试目的：测试 quit 功能

测试代码：

```
quit;
```

期望结果：退出当前程序。

15. 测试目的：测试执行 SQL 脚本文件功能

测试代码：

```
execfile <file_name>;
```

期望结果：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。如果不存在该文件，则输出“No such file”，否则按照每条 SQL 语句进行输出。

3.2.2 语法测试

1. 测试目的：测试能否判断“大写输入正确关键字”为错误语法
   测试代码：

```
SHow databases;
UsE <database_name>;
create Database <database_name>;
Drop database <database_name>;
sHow tables;
CREATE TABLE <table_name> (
column_name type_name);
drOp table <table_name>;
CreaTe index <index_name> on <table_name>(<column_name>);
drOp index <index_name>;
SELECT * from <table_name>;
select * From <table_name>;
select * from <table_name> WHERE <condition>;
INsert into <table_name> values (<value1>, <value2>, …);
DELEte from <table_name>;
delete from <table_name> wHere <condition>;
UPDATE <table_name> set <column_name> = <value> where <condition>;
QUIt;
Execfile;
```

期望结果：输出“Error: Syntax error.”

2. 测试目的：测试能否判断“小写输入错误关键字”为错误语法
   测试代码：

```
show database; //databases
uses <database_name>; //use
creat database <database_name>; //create
create databases <database_name>; //database
dropt database <database_name>; //drop
drop databases <database_name>; //database
show table; //tables
create tables <table_name> (
column_name type_name); //table
create tables <table_name> (
```

```
column_name inte); //int
create tables <table_name> (
column_name varchar); //char
create tables <table_name> (
column_name double); //float

drop tables <table_name>; //table
create indes <index_name> on <table_name>(<column_name>); //index
drop indexs <index_name>; //index
selete * from <table_name>; // select
select * form <table_name>; // from
select * from <table_name> what <condition>; //where
insert to <table_name> values (<value1>, <value2>, …); //into
deleted from <table_name>; //delete
update <table_name> sett <column_name> = <value> where <condition>;//set
quite; //quit
execfiles; //execfile
```

期望结果：输出“Error: Syntax error.”

3. 测试目的：检测命令的关键字位置和搭配是否正确
   测试代码：

```
<database_name> create database; //开头为非关键字
create <database_name> database; //create 后跟非 database, table, index
drop <database_name> database; //drop 后跟非 database, table, index
drop database; //没有选定数据库

create table <table_name> (
type_name column_name);        // create table 中类型在前，列名在后
drop table;            //没有选定表

create index <index_name>; //没有选定表
create index on <table_name>(<column_name>); //缺少索引名
create index <index_name> on <table_name>; //缺少列名
create index <index_name> <table_name>(<column_name>); //缺少 on
drop index; //没有选定索引

select *; //没有选定表
select * where <condition> from <table_name>; //where 与 from 顺序颠倒
select from <table_name>; //缺少结果的列名
from <table_name> select *; //select 与 from 位置颠倒
select * from <table_name> group by <column_name> where <condition>;
//group by 与 where 搭配错误
```

```
insert <table_name> values (<value1>, <value2>, …); //缺少 into
insert into values (<value1>, <value2>, …); //没有选定表
insert into <table_name> (<value1>, <value2>, …); //缺少 values

delete from where <condition>; //没有选定表
delete <table_name>; //缺少 from
delete where <condition> from <table_name>; //from 和 where 顺序颠倒

update set <column_name> = <value> where <condition>; //没有选定表
update <table_name> <column_name> = <value> where <condition>; //缺少 set
update <table_name> set <column_name> where <condition>; //缺少赋值
update <table_name> where <condition>; //缺少 set
update <table_name> set <column_name> = <value> <condition>;//缺少 where
```

期望结果：输出“Error: Syntax error.”

4 分组与设计分工

本组成员：
董玮豪 3150104577
刘子旋 3150104437
王大鑫 3150103559

本系统的分工如下：
董玮豪：IndexManager, DBFiles
王大鑫：API, Interpreter, 测试
刘子旋：BufferManager, CatalogManager, RecordManager