**MiniSQL 总体设计报告**

作者：2016级软件工程专业

程浩然 傅诤哲 叶慕祈 钱程 石磊

# 1 引言

## 1.1编写目的

为了更好地实现我们的MiniSQL，我们对系统的进行了一个总体的分析，对系统的层次和构造进行了解析，希望能够提高我们的编码效率和质量，并且能使用户能够更清晰地了解到我们MiniSQL的构造过程和使用方法。

## 1.2项目背景

本项目由程浩然、傅诤哲、叶慕祈、钱程、石磊在数据库老师陈岭的指导下完成的，是希望以课堂大程的形式来独立完成一个MiniSQL的小型系统。

本组开发人员具有数据库应用与实现的知识，并且有较好的C++基础，而且有良好的团队精神。

## 1.3定义

* **C++**:是一种面向对象编程语言。它是C语言继承和发展后的产物，既可以实现C语言中面向过程的程序设计，又可以进行以抽象数据类型为特点的基于对象的程序设计，还可以进行以继承和多态为特点的面向对象的程序设计。它是一种使用范围非常广泛的程序设计语言。
* **SQL**:是一种访问数据库管理数据库的语言，是结构化查询语言(Structured Query Language)的简称，是一种高级的非过程化语言。

## 1.4参考资料

《数据库系统概念》 Abraham Silberschatz, Henry F.Korth, S. Sudarshan

# 2 任务概述

## 2.1目标

设计并实现一个精简型单用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。另外还有完善用户体验的各种语句，例如使用、退出、显示帮助等。在查找方面，不仅有定位查找，还有大于小于查找等等。

## 2.2运行环境

可在Windows系统上运行，内存需求为4G。需要安装VS17的运行库。

## 2.3需求概述

* 数据类型

需要支持三种基本数据类型：int，char(n)，float，其中char(n)满足 1 <= n <= 255。

* 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；支持单属性的主键定义。

* 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）。

* 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。

* 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。

## 2.4 SQL语句需求说明

MiniSQL支持标准的SQL语句格式，每一条SQL语句以分号结尾，一条SQL语句可写在一行或多行。为简化编程，要求所有的关键字都为小写。在以下语句的语法说明中，用黑体显示的部分表示语句中的原始字符串，如create就严格的表示字符串“create”，否则含有特殊的含义，如 表名 并不是表示字符串 “表名”，而是表示表的名称。

* **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 )

);

其中类型的说明见第二节“功能需求”。

若该语句执行成功，则输出执行成功信息；若失败，必须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 )

);

* **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，必须告诉用户失败的原因。

示例语句：

drop table student;

* **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，必须告诉用户失败的原因。

示例语句：

create index stunameidx on student ( sname );

* **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，必须告诉用户失败的原因。

示例语句：

drop index stunameidx;

* **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，必须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’;

* **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，必须告诉用户失败的原因。

示例语句：

insert into student values (‘12345678’,’wy’,22,’M’);

* **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，必须告诉用户失败的原因。

* **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8种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。

同时我们增加了一些语句：

* **使用数据库语句**

该语句的语法如下：

Use 数据库名;

可以开始使用我们的数据库。

* **停止使用数据库**

该语句的语法如下：

quit;

可以停止使用数据库并退出程序。

* **显示帮助文件**

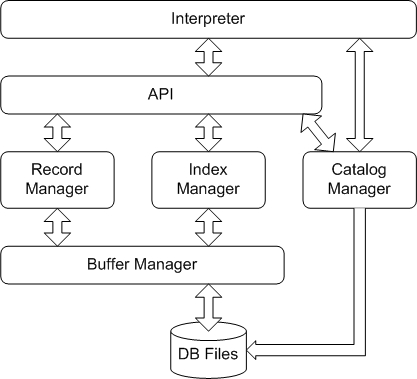
该语句的语法如下：

help;

可以显示我们为数据库系统写的帮助文件。

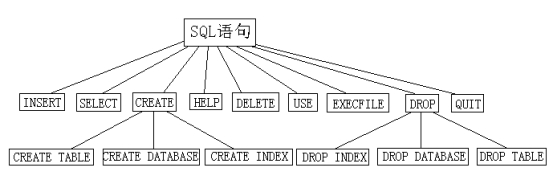
# 3 系统体系结构设计：

在系统的结构设计中，我们主要是按照实验设计指导书中给定的进行设计，各个模块之间的接口会在下面的模块接口设计中进一步阐述，设计结构图完全参照实验指导上的设计结构，主要分为Interpreter、API、Catalog Manager、Record Manager、Index Manager、Buffer Manager、Catalog Manager、Buffer Manager等模块。



## 3.1 Interpreter模块

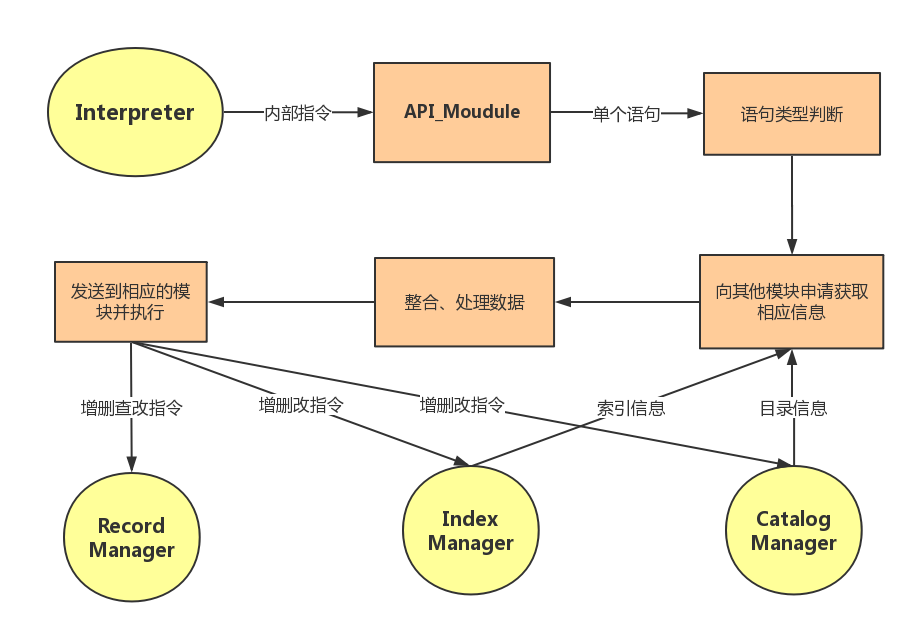
**Interpreter 模块**的主要功能是接受用户输入的SQL语句及其他命令语句，并检验用户输入的SQL语句及其他命令语句的格式，语法和语义的正确性。同时将符合要求的语句转化为内部形式，供API及Catalog模块使用；而对不符合要求的语句，显示其出错信息，供用户参考。SQL语句的命令解析如下图所示：



## 3.2 API模块

**API模块**是整个系统的核心。其主要功能是根据Interpreter层解释生成的内部命令，将其进行解析之后，统筹调度Record Manager、Index Manager和Catalog Manager完成相应的功能操作。

API模块会首先解析Interpreter模块发来的SQL字符串指令，并分辨其指令类型并验证其合法性；随后，API模块将向Catalog Manager或Index Manager模块发送请求，以获取完成SQL指令功能所需要的数据；在数据准备步骤完成之后，API模块会对获取到的数据进行整合，并分发给Catalog Manager、Index Manager与Record Manager模块，以完成对底层数据库文件的相应操作，从而最终实现SQL相应的功能。



本模块接收Interpreter发来的指令，并解析成相应类型的指令：创建数据表/创建索引/删除数据表/删除索引/查询数据/插入数据/删除数据/帮助/退出指令。对每个单独的指令，本模块会预先调用其他的子模块做好数据准备，并将准备好的数据整合之后，发送给相应的子模块进行执行。

## 3.3 Catalog Manager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。
4. 数据表中的记录条数及空记录串的头记录号。
5. 数据库内已建的表的数目。

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。

为减小模块之间的耦合，Catalog模块采用直接访问磁盘文件的形式，不通过Buffer Manager，Catalog中的数据也不要求分块存储。

数据字典文件格式：C:\Users\ASUS\AppData\Local\Temp\ksohtml\wpsB6B4.tmp.jpg

## 3.4 Record Manager模块

**Record Manager**负责管理记录表中数据的数据文件。主要功能为实现记录的插入、删除与查找操作，并对外提供相应的接口。其中记录的查找操作要求能够支持不带条件的查找和带条件的查找（包括等值查找、不等值查找和区间查找）。

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（一个文件中只存储一个表或索引的内容），且不要求支持记录的跨块存储。

## 3.5 Index Manager模块

**Index Manager**是程序的索引部分,它负责维护一个B+树的数据结构，并优化数据库的查询效率。它通过Buffer Manager提供的文件块的操作，在外存的文件中维护索引。在Record Manager对数据进行查询时，Index Manager可以为record manager提供对数时间优化的查询。于此同时，Index Manager还需要对所有的索引进行维护。在数据库中的表发生增加、删除和修改的时候，Index Manager需要通过Buffer Manager对保存索引的B+树进行修改，并储存入磁盘。

Index Manager会将数据库索引以B+树的数据结构储存在外存中。当需要建议索引时，Index Manager将通过Record Manager获得建索引的键与地址的对应关系数组。当索引需要修改时，Index Manager将从Record Manager处获得需要修改的相关数据的信息。在维护Index Manager时，主要需要实现三大类操作：插入节点、删除节点和查询节点。在本系统中，我们采用分块储存的方式在外存中维护了一个5阶B+树。

## 3.6 Buffer Manager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。

记录管理模块（Record Manager）和索引管理模块（Index Manager）向缓冲区管理申请所要的数据，缓冲区管理器首先在缓冲区中查看数据是否存在，若存在，直接返回，否则，从磁盘中将数据读入缓冲区，然后返回。

最近最少使用(LRU)算法：用一个链表记录所有的缓冲块，每次访问到一个缓冲块就将它插入到链表的头部，这样链表尾的缓冲块就是最近最少使用的块，在需要的时候就可以替换出去。

改进：考虑增加脏块的设置。

# 4 接口设计

## 4.1外部接口

本系统的外部接口为控制台。

使用者通过控制台的字符界面输入查询语句使用MiniSQL。输入方式为键盘输入。

## 4.2内部接口

本系统内部模块间采用函数调用的方式。具体描述如下。

Catalog Manager为Interpreter提供各种与创建、删除、查询表有关的函数接口，Interpreter调用相应的接口来实现用户的要求。调用时，Record Manager对表进行读写，并且通过Index Manager来更新维护数据库中的表。Index Manager通过Record Manager提供的数据，对索引进行建立、删除和维护，并为Record Manager提供索引查询的接口。Buffer Manager为Record Manager提供查找文件的接口。

## 4.3 接口清单

* **Catalog Manager**

API会根据interpreter的指令，调用Catalog M的以下函数：

1．建立和删除记录

|  |  |
| --- | --- |
| **建立数据表** | Create\_Table(string Table\_Name); |
| **建立表元素** | Create\_Attr(string Table\_Name, string Attr\_Name, string Attr\_Type, string Attr\_Argu)  其中Attr\_Argu为char(n)中的n，类型为string |
| **删除数据表记录** | Drop\_Table(string Table\_Name); |

注：这里建立与删除的都是数据表/元素的目录信。具体的数据（DBfile里的数据）的空间的释放是通过IndexManager和RecordManager实现的。

2.查询相关信息

|  |  |
| --- | --- |
| **表信息相关** | |
| **检查数据库是否存在** | int DBExists(string database) |
| **检查表是否存在** | int tableExists(string database, string table) |
| **询问表信息** | Table get\_table\_info(string table); |
| **检查列信息** | Int attrExists(string database, string table, sstring attr) |
| **检查某字段是否唯一** | int isUnique(string table, string attr) |
| **索引相关** | |
| **检查某字段是否有索引** | int hasIndex(string table, string attr) |
| **返回字段所有索引** | Index indexName(string table, string attr) |
| **检查某表是否有主键** | int isPK(string table, string attr) |
| **寻找表中主键** | CString pkOnTable(string table) |
| **获取某索引信息** | get\_index\_info(string index) |
| **获取所有索引信息** | get\_all\_index() |
| **返回索引的总数** | int indexCount(string table) |
| **获取索引所在表** | CString tableIndexOn(string index) |
| **获取索引所在字段** | CString attrIndexOn(string index) |

* **Index Manager**

API模块会根据Interpreter的指令，调用Index M的以下函数：

**1.新建删除索引文档**

|  |  |
| --- | --- |
| **建立索引** | **Create\_Index(string Table\_Name, int N,**  **int Attr\_N，int\* Attr\_Type, string\* Attr\_Name,**  **string Attr, string IndexName, int IndexType)**  注释：  Table\_Name是表名；  Attr\_Type是表内各个参数的类型 ( 1:int 2:float -n:char(n) );  Attr\_Name是表内各个参数的名字;  Int Attr\_N是表中的属性数（列数，也就是粉色数组的size）  string Attr, 索引所在参数  string IndexName, 索引名  int IndexType 索引的属性  IndexType定义规则：1为int，2为float，-n为char(n) |
| **删除索引** | **Delete\_Index(string IndexName)** |

**2.向索引文档添加/删除元素**

|  |  |
| --- | --- |
| **添加元素** | **Index\_InsertElement(int N, string IndexName, int IndexType,**  **int Key1, float Key2, string Key3)**  注释：  N为该元素在record文档中应在的地址  IndexType的定义规则同上  Key为元素的具体值（当Type为int时，Key1有效，Key2/Key3传入的值为空，不用管它；其它类型同理） |
| **删除元素** | 删除操作由record发出请求 |

* **Record Manager**

API模块会调用Record M的以下函数：

|  |  |
| --- | --- |
| **查询 SELECT** | |
| **不含Where的查询** | **Select\_All(int N, Table\_Name, int Attr\_N，**  **int\* Attr\_Type, string\* Attr\_Name)** |
| **含Where的查询** | **Select(int N, Table\_Name, int Attr\_N,**  **int\* Attr\_Type, string\* Attr\_Name,**  **Int op\_N, string\* Attr, int\* op, string\* Num,**  **string\* Index)**  注释：  Table\_Name是表名；  Attr\_Type是表内各个参数的类型 ( 1:int 2:float -n:char(n) );  Attr\_Name是表内各个参数的名字;  Int N是表中的记录数（行数）  Int Attr\_N是表中的属性数（列数）  *SELECT \* FROM Table\_Name WHERE (Attr[1] op[1] Num[1]) AND (…) AND (…) AND …*  op\_N是where后条件的个数  Attr是参与where的属性名  op是算术比较符：= <> < > <= >= 分别对应 0 1 2 3 4 5  Num是参与比较的值，跟Attr的类型有关（传过去的是string类型）  Index是某个Attr是否含有索引，有则返回索引名，无则为空  （查询操作中需要RecordM与IndexM进行交互） |
| **插入 INSERT** | |
| **插入数据** | **Insert(int N,Table\_Name, int Attr\_N,**  **int\* Attr\_Type, string\* Attr\_Name,**  **string\* value)**  这里的value是数据值（已经验证过每个数据与其对应的行属性的类型了，也就是说这个value是必定合法的） |
| **删除 DELETE** | |
| **删除所有数据** | **Delete\_All(int N,Table\_Name)** |
| **含Where的删除** | **Delete(int N,Table\_Name, int Attr\_N,**  **int\* Attr\_Type, string\* Attr\_Name,**  **Int op\_N, string\* Attr, int\* op,**  **string\* Num, string\* Index)**  （删除操作中需要RecordM与IndexM进行交互）  参数条件说明同SELECT |

# 5 出错处理设计

本系统出错主要在三个方面：一个是语法分析，主要有Interpreter模块进行检查；二是语句合法性（如查询的表不存在、插入属性与类型不符等），主要由API模块调度Catalog Manager模块进行验证；三是运行异常，主要由各个模块进行报错。

语法分析方面，在Interpreter模块只检查语法错误，若出现错误则会提示输入的不是有效字符，并重新开始程序，并不关闭程序。

语句合法性方面，在API模块会对传输进来的命令进行统一验证（与Catalog Manager模块所存储的目录信息进行统一比对），在验证合法之后才会指派给各个模块进行执行，否则会报错并执行下一条指令。

# 6 设计分工

|  |  |
| --- | --- |
| 组长 | **程浩然** |
| Interpreter模块 | **钱程** |
| API模块 | **程浩然** |
| Catalog Manager模块 | **傅诤哲** |
| Record Manager模块 | **叶慕祈** |
| Index Manager模块 | **石磊** |
| Buffer Manager模块 | **傅诤哲** |
| 测试 | **ALL** |
| 总体设计报告 | **钱程 石磊** |