**MiniSQL总体设计报告**

# MiniSQL系统概述

项目名称：MiniSQL数据库实现

## 背景

### 编写目的

通过MiniSQL的开发，深入理解数据库的技术实现，增强开发经验。

### 项目背景

《数据库系统设计》大作业

## 功能描述

-表定义中的列(属性)支持三种数据类型：整型(int)、浮点型(double)、字符型数组(char array)

-每张表必须有一个主键，且只有一个主键，单属性主键

-实现数据库的创建/删除，实现表的创建/删除

-实现记录的插入、删除、主键查找、非主键查找

功能列表：

- 数据库创建

- 数据库删除

- 表的创建

- 表的删除

- 插入记录

- 删除记录

- 更新记录

## 运行环境和配置

运行环境为windows环境。

可直接打开Mini\_SQL\Mini\_SQL\_VS文件夹下的.sln文件，作为VS项目直接运行，已经配置好项目和筛选器。

## 参考资料

<https://www.cnblogs.com/wade-luffy/p/6292784.html>

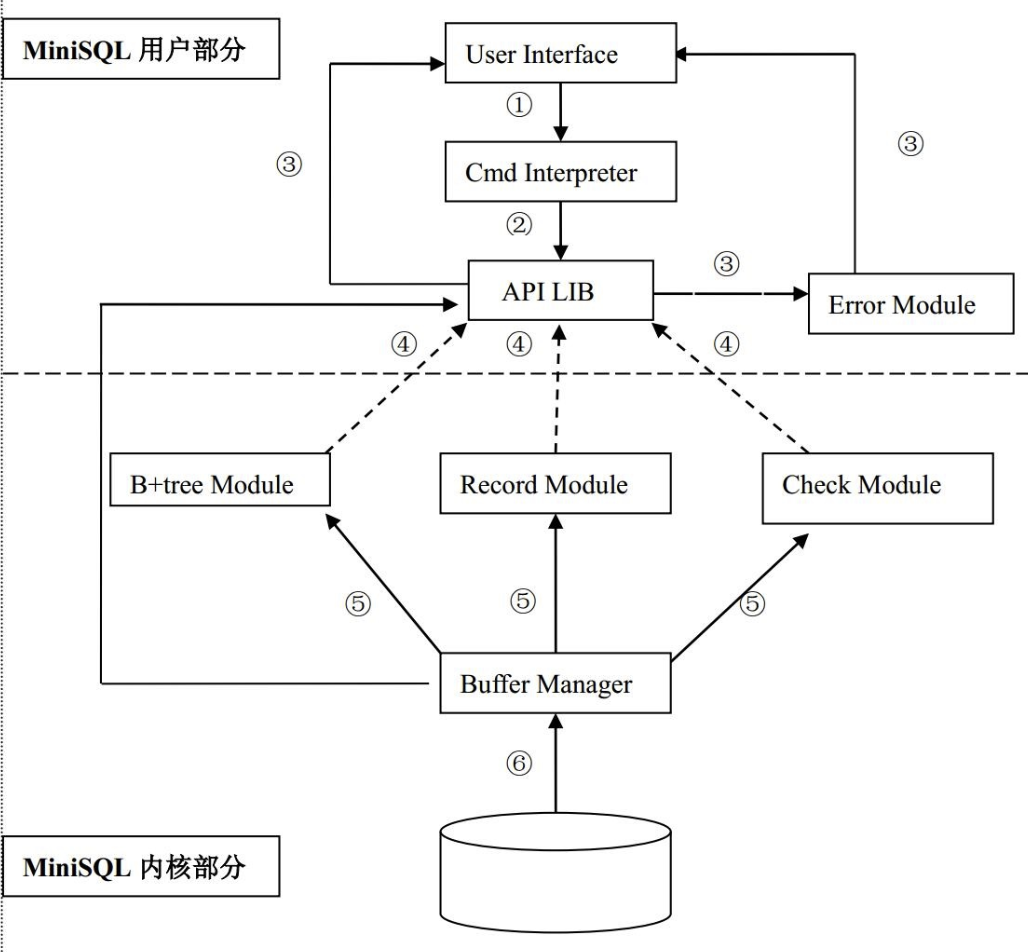
<https://www.jianshu.com/p/6f68d3c118d6>

<http://hedengcheng.com/?p=525>

# MiniSQL系统结构设计

## 总体设计：

系统体系结构图



主要分为两大部分，不过各模块间相互联系。

其中标号含义：

1. 接收用户输入的字符串，并将输入传递给解释器；
2. 解释器用户的输入进行解析和语法检查，产生操作命令，并将命令传递给API模块；
3. API模块根据上层传递的命令数据，调用相关底层接口执行操作，输出命令执行结果并返回到用户界面；
4. B+tree、Record、Check类方法注册到APILIB，提供记录读写和类型检查；
5. B+tree、Record、Check类通过调用Buffer类实现对底层数据读写；
6. Buffer类方法对数据库文件进行直接读写操作；

## Interpreter 模块：

1. 目标：翻译用户命令，生成命令封装数据，调用下层模块执行命令
2. 程序文件：interpreter.h interpreter.cpp
3. 调用其他模块：APILIB模块
4. 主要数据结构及接口说明



## API模块：

1. 目标：根据底层模块提供的接口，实现所有数据库的功能操作
2. 程序文件：API.h API.cpp
3. 调用其他模块：Buffer模块 B+树模块 Record模块 Check模块
4. 主要数据结构及接口说明
5. // 记录查找关系符,用于查找记录时条件语句的关系运算
6. enum Operator\_Type
7. {B,BE,L,LE,E,NE};
8. // 比较单元，将每一个查找/删除 条件分装成比较单元，用于查找符合条件的记录
9. class CompareCell;
10. // 目录定位和切换，定位当前位置是否处于数据库中，用以检查数据库/表操作的合法性检查
11. class CatalogPosition;
12. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
13. \*    表创建信息
14. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
15. struct TB\_Create\_Info
16. {
17. using ColumnInfo = struct ColumnInfo // 新建表的字段信息
18. {
19. std::string name; // 字段名称
20. Column\_Type type; // 字段类型
21. bool isPrimary;   // 是否主键
22. int length;       // 字段数据长度
23. };
24. std::string table\_name;               // 新建的表名
25. std::vector<ColumnInfo> columns\_info; // 表的各个字段
26. };
27. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
28. \*    表插入信息
29. \*    插入一条新记录，包括 字段-值 的数组，没有给定插入值的字段初始化为默认值
30. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
31. struct TB\_Insert\_Info
32. {
33. using InsertInfo = struct
34. {
35. std::string column\_name;  // 插入的字段
36. std::string column\_value; // 插入的值
37. };
38. std::string table\_name;              // 插入的表名
39. std::vector<InsertInfo> insert\_info; // 插入字段集合
40. };
41. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
42. \*    表记录选择(查找)信息
43. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
44. struct TB\_Select\_Info
45. {
46. std::string table\_name;                        // 查找的表名
47. std::vector<std::string> name\_selected\_column; // 查找的字段名字
48. std::vector<CompareCell> vec\_cmp\_cell;         // 查找条件
49. };
50. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
51. \*    表更新信息
52. \*    eg.
53. \*    NewValue = {name,china}
54. \*    expr = {id,>,20}
55. \*    操作结果:将id大于20的所有字段的 "name"字段的值全部改为"china"
56. \*    若expr为数组，则最终修改的记录为满足所有的expr数组元素条件
57. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
58. struct TB\_Update\_Info
59. {
60. // 更新的字段名称以及新的值
61. using NewValue = struct
62. {
63. std::string field;
64. std::string value;
65. };
66. // 更新条件（要更新的记录满足的条件）
67. using Expr = struct
68. {
69. std::string field; // 条件的字段名称
70. std::string op;    // 条件的字段运算关系
71. std::string value; // 比较值
72. };
73. std::string table\_name;
74. std::vector<NewValue> field\_value; // 字段——值 向量
75. std::vector<Expr> expr;            // 跟新的字段条件
76. };
77. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
78. \*    表删除信息
79. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
80. struct TB\_Delete\_Info
81. {
82. using Expr = struct
83. {
84. std::string field;
85. std::string op;
86. std::string value;
87. };
88. std::string table\_name; // 删除目标的表名
89. std::vector<Expr> expr; // 删除的字段条件
90. };
91. // 数据库操作，一个数据库对应一个文件夹
92. // 数据库中的每张表对应数据库文件夹下的两个文件（索引文件.idx 和 数据文件.dbf）
93. bool CreateDatabase(std::string database\_name, CatalogPosition &cp);
94. bool DropDatabase(std::string database\_name, CatalogPosition &cp);
95. bool UseDatabase(std::string db\_name, CatalogPosition &cp);
96. // 表的相关操作见源码以及注释

## 异常处理模块：

1. 目标：输出异常信息并写入日志
2. 程序文件：Error.h Error.cpp
3. 调用其他模块：无
4. 主要数据结构及接口说明
5. // 全局log日志
6. extern std::fstream log\_file;
7. // 异常处理基类，实现异常显示和日志写入
8. class BaseError
9. {
10. public:
11. virtual void PrintError() const;
12. virtual void WriteToLog() const;
13. protected:
14. std::string ErrorInfo;
15. std::string ErrorPos;
16. };

## B+Tree模块：

1. 目标：通过B+树，创建和读写文件索引，提供数据库快速检索
2. 程序文件：b+tree.h b+tree.cpp
3. 调用其他模块：Buffer模块
4. 主要数据结构及接口说明
5. 接口实现
   1. 创建和读写索引文件
   2. 实现对关键字的查找、插入和删除操作
   3. 打印B+树结点信息
6. 工作原理

关于B+树的具体定义，目前还有一些细节上的分歧。在本项目的B+树的实现中，\*\*B+树按照结点关键字个数和孩子结点个数一样多的定义实现\*\*。即在一个结点中，有N个关键字，k[0], k[1], ..., k[N-1]，就对应着N个孩子结点child[0], child[1], ..., child[N-1]。其中，k[0] <= child[0] < k[1] <= child[1], ..., <key[N-1] <= child[N-1]。

1. 主要数据结构及接口说明:
2. // 索引文件头信息结点
3. //MostLeftNode指向最左边的叶子结点，用于遍历数据和按照范围查找关键字。
4. //RecordInfo记录了表中的一条记录的各个字段的类型和顺序。比如：
5. /\*
6. RecordInfo数据                              记录类型
7. IFD             一条记录三个字段，类型依次为：int,float,doouble
8. IC10D           一条记录三个字段，类型依次为：int, char, double。其中char字段为10个字符的字符串\*/
9. class IndexHeadNode
10. {
11. public:
12. FileAddr root;                     // the address of the root
13. FileAddr MostLeftNode;             // the address of the most left node
14. char KeyType;                      // 关键字类型
15. char RecordInfo[RecordInfoLength]; // 记录字段相关信息
16. };
17. /\* B+树的结点结构体。其中ROOT标志只在B+树的根结点关键字未满的情况下使用。在根节点分裂后，整个B+树只存在内结点和叶子结点两种类型。\*/
18. // define B+tree Node
19. enum class NodeType
20. {
21. ROOT,
22. INNER,
23. LEAF
24. };
25. class BTNode
26. {
27. public:
28. NodeType node\_type;  // node type
29. int count\_valid\_key; // the number of key has stored in the node
30. KeyAttr key[MaxKeyCount];         // array of keys
31. FileAddr children[MaxChildCount]; // if the node is not a leaf node, children store the children pointer
32. // otherwise it store record address;
33. FileAddr next; // if leaf node
34. void PrintSelf();
35. };
36. // B+tree
37. class BTree
38. {
39. public:
40. BTree(char \*idx\_name, char \_KeyType = 'i', char \*\_RecordInfo = "i"); // 创建索引文件的B+树
41. ~BTree() { delete idx\_name; }
42. FileAddr Search(KeyAttr search\_key);   // 查找关键字是否已经存在
43. bool Insert(KeyAttr k, FileAddr k\_fd); // 插入关键字k
44. FileAddr Delete(KeyAttr k);            // 返回该关键字记录在数据文件中的地址
45. void PrintBTreeStruct();               // 层序打印所有结点信息
46. void PrintAllLeafNode();
47. private:
48. FileAddr DeleteKeyAtInnerNode(FileAddr x, int i, KeyAttr key); // x的下标为i的结点为叶子结点
49. FileAddr DeleteKeyAtLeafNode(FileAddr x, int i, KeyAttr key);  // x的下标为i的结点为叶子结点
50. void InsertNotFull(FileAddr x, KeyAttr k, FileAddr k\_fd);
51. void SplitChild(FileAddr x, int i, FileAddr y);                 // 分裂x的孩子结点x.children[i] , y
52. FileAddr Search(KeyAttr search\_key, FileAddr node\_fd);          // 判断关键字是否存在
53. FileAddr SearchInnerNode(KeyAttr search\_key, FileAddr node\_fd); // 在内部节点查找
54. FileAddr SearchLeafNode(KeyAttr search\_key, FileAddr node\_fd);  // 在叶子结点查找
55. BTNode \*FileAddrToMemPtr(FileAddr node\_fd);                     // 文件地址转换为内存指针
56. private:
57. char \*idx\_name;
58. int file\_id;
59. IndexHeadNode idx\_head;
60. };

## Record模块：

1. 目标：负责记录数据的读写
2. 程序文件：Record.h Record.cpp
3. 调用其他模块：Buffer模块
4. 主要数据结构及接口说明
5. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
6. \*
7. \*    定义记录各个字段的类型
8. \*    字段类型 I---int  C---字符串  D---Doouble
9. \*
10. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
11. enum class Column\_Type
12. {
13. I,
14. C,
15. D
16. };
17. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
18. \*
19. \*    联合的数据结构定义 字段的值
20. \*
21. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
22. union Column\_Value {
23. int IntValue;       //整形值
24. double DoubleValue; //浮点型值
25. char \*StrValue;     //字符串指针
26. };
27. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
28. \*
29. \*    定义索引文件关键字类型
30. \*
31. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
32. class KeyAttr
33. {
34. public:
35. using Key\_Value = union {
36. char StrValue[ColumnNameLength]; //字符串指针
37. int IntValue;                    //整形值
38. double DoubleValue;              //浮点型值
39. };
40. Column\_Type type;
41. Key\_Value value;
42. bool operator<(const KeyAttr &rhs) const;
43. bool operator>(const KeyAttr &rhs) const;
44. bool operator==(const KeyAttr &rhs) const;
45. bool operator<=(const KeyAttr &rhs) const;
46. bool operator>=(const KeyAttr &rhs) const;
47. bool operator!=(const KeyAttr &rhs) const;
48. };
49. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
50. \*
51. \*    定义每个字段的单元数据
52. \*    字段单元记录了该字段的数据类型、数据的值、以及该字段的下一个字段指针
53. \*    注意拷贝和赋值操作
54. \*    如果保存的字段是字符串类型，则字符串的前三个字符表示表创建时定义好字符串长度
55. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
56. class Column\_Cell;
57. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
58. \*
59. \*    定义一条记录的头结构，唯一标志一条记录
60. \*    记录的头记录了该记录的第一个字段的地址
61. \*    记录各个字段以链表的形式形成整条记录
62. \*    若是字符串类型字段，字符串前三个字符是表示该字段字符串的长度
63. \*
64. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
65. class RecordHead
66. {
67. public:
68. RecordHead();
69. RecordHead(const RecordHead &rhs);            // 拷贝构造
70. RecordHead &operator=(const RecordHead &rhs); // 拷贝赋值
71. RecordHead(RecordHead &&rhs);                 // 移动构造
72. RecordHead &operator=(RecordHead &&rhs);      // 移动赋值
73. ~RecordHead();
74. void AddColumnCell(const Column\_Cell &cc);
75. size\_t size() const; // 返回整条记录的大小
76. Column\_Cell \*GetFirstColumn() const;
77. private:
78. Column\_Cell \*phead; // 指向记录的第一个字段
79. Column\_Cell \*pLast;
80. };
81. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
82. \*
83. \*    记录类:  定义所有的记录数据操作。
84. \*    注意:    记录读写包括索引部分(.idx)和数据部分(.dbf)。
85. \*             索引部分的修改由B+树对应的模块负责
86. \*             本模块的所有操作只修改记录的实际数据部分。
87. \*
88. \*    说明：   本类所有的记录均以文件地址唯一标志。删除和查找记录的地址由索引树提供
89. \*
90. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
91. class Record
92. {
93. public:
94. // 插入新的记录，返回新插入记录的所在数据文件的地址
95. FileAddr InsertRecord(const std::string dbf\_name, const RecordHead &rd);
96. // 删除记录，返回删除的记录所在数据文件的地址
97. FileAddr DeleteRecord(const std::string dbf\_name, FileAddr fd, size\_t);
98. // 更新整条记录,若是字符串类型字段，字符串前三个字符是表示该字段字符串的长度
99. // 实际写入时并不将字符串的长度写进文件
100. bool UpdateRecord(const std::string dbf\_name, const RecordHead &rd, FileAddr fd);
101. private:
102. // 将以链表的形式传入的记录数据读取并保存为一个整数据块以便于写入数据文件
103. // tuple的第一个元素为记录数据的大小，第二个元素为数据的指针
104. std::tuple<unsigned long, char \*> GetRecordData(const RecordHead &rd);
105. };

## Buffer模块：

1. 目标：实现物理文件的读写，包括两类接口：

一类实现对任意文件任意位置读写任意长度的数据

另一类实现对固定长度数据的读写操作，并自动管理文件空间

1. 程序文件：Buffer.h Buffer.cpp
2. 调用其他模块：全局变量模块 异常处理模块
3. 功能：

使其他模块与物理磁盘完全隔开，使之完全转换为内存操作

提供系统缓冲， 模拟操作系统内存管理功能，加快数据文件的读取和

1. 接口实现：

实现文件的创建和删除

实现数据（记录）在文件中的读取和写入

实现在文件中添加和删除数据时的空间自动管理

提供对上层模块的统一接口

1. 工作原理：

所有对于文件的读写都通过Buffer类提供的接口实现

Buffer类将磁盘文件以页的形式换入内存缓冲

当待读写的文件数据所在的文件页之前已经被读入内存，则直接对该缓冲数据进行操作

当待读写的文件数据所在的文件页不存在内存中

如果内存缓冲页未满，则将需要读写的文件页读入内存缓冲页，执行读写操作

如果内存缓冲页已满，则通过内存页置换算法，将部分之前被使用的内存页写回到原文件，将待读写的文件页换入内存页

1. 文件格式：

索引文件和数据文件有统一的文件格式

每个文件都以若干固定大小的文件页的形式存在

除了零号页面，所有文件页都由\*\*页头\*\*和\*\*数据区\*\*两部分组成

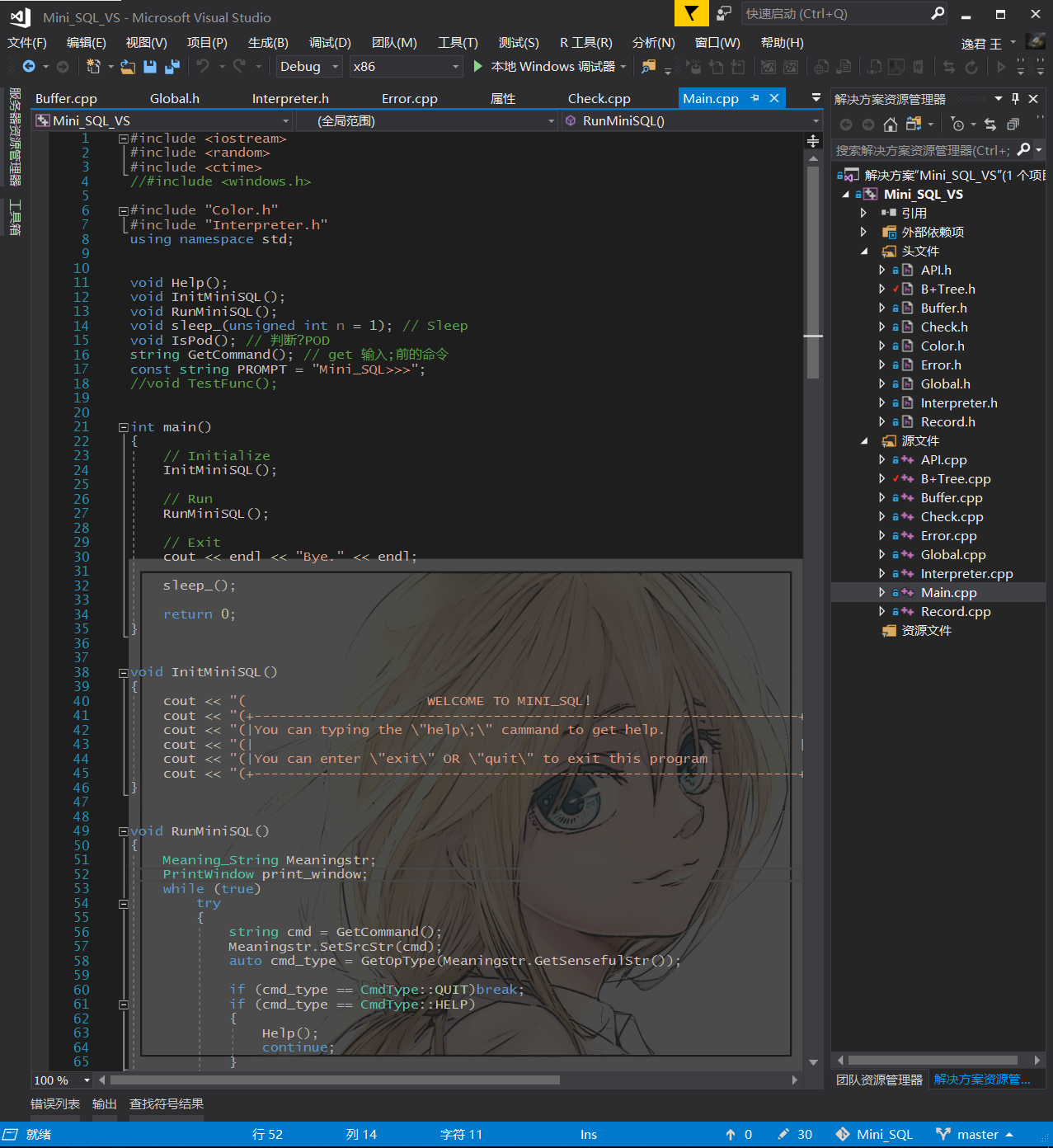
页头由\*\*页号\*\*和\*\*常驻内存标志\*\*组成。页号表示该页在文件中的位置，常驻内存标志用于内存页置换算法

零号文件页除了页号和常驻内存标志外，还拥有\*\*文件头信息数据区\*\*，文件头信息记录了该文件可以写入新数据的位置信息，以及被删除的数据的位置信息。此外，还有一块预留区域保留以后使用

## Check模块：

1. 目标：负责命令数据的检查
2. 程序文件：check.h check.cpp
3. 调用其他模块：API模块
4. 主要数据结构及接口说明
5. // 创建新表信息检查
6. void Check\_TB\_Create\_Info(const TB\_Create\_Info &tb\_create\_info);
7. // 表插入信息检查
8. void Check\_TB\_Insert\_Info(const TB\_Insert\_Info &tb\_insert\_info);
9. // 表更新信息检查
10. void Check\_TB\_Update\_Info(const TB\_Update\_Info &tb\_update\_info);
11. // 删除信息检查
12. void Check\_TB\_Delete\_Info(const TB\_Delete\_Info &tb\_delete\_info);
13. // 查找信息检查
14. void Check\_TB\_Select\_Info(const TB\_Select\_Info &tb\_select\_info);

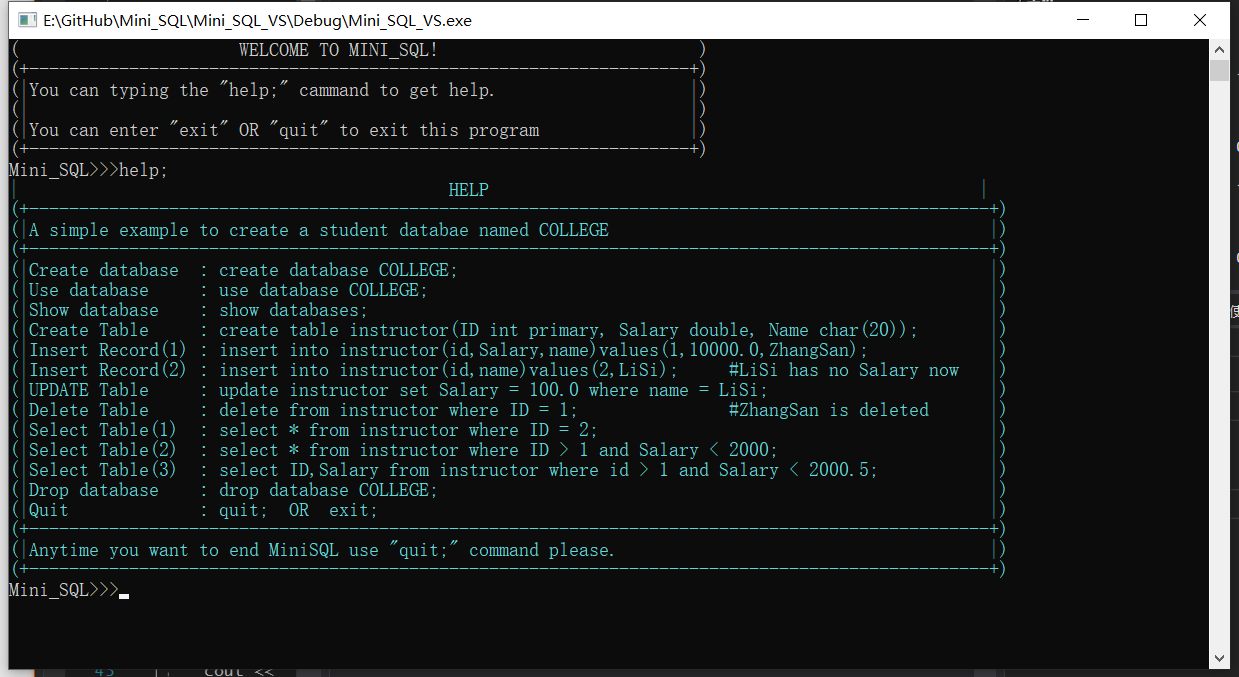
# 测试方案和测试样例



打开VS进行运行，Debug或者Release模式皆可

使用本地调试器

进入界面，如果键入help；则会弹出帮助窗口



# 分组与设计分工

本组成员（姓名王逸君 学号：3180105884）全部内容