

浙江大学

数据库系统实验报告

MINISQL 小组实验报告

徐 震 3180105504 18888916826

曾一欣 3180105144 13372553368

毛一恒 3180103727 13587492148

指导教师

孙建伶

2020 年 6 月 21 日

目录

第一部分 正文	5
第一章 实验目的	6
1.1 总体实验目的	6
1.1.1 MINISQL	6
1.1.2 设计目的	6
1.2 模块实验目的	6
1.2.1 索引管理器	6
1.2.2 缓存管理器	6
1.2.3 记录管理器	7
1.2.4 目录管理器	7
1.2.5 接口管理器	7
1.2.6 命令行解释器	7
1.2.7 图形界面	7
第二章 系统需求	8
2.1 需求概述	8
2.1.1 数据类型	8
2.1.2 表定义	8
2.1.3 索引的建立和删除	8
2.1.4 查找记录	8
2.1.5 插入和删除记录	8
2.2 语法说明	8
2.2.1 创建表语句	9
2.2.2 删除表语句	9
2.2.3 创建索引语句	10
2.2.4 删除索引语句	10
2.2.5 选择语句	10

2.2.6	插入记录语句	11
2.2.7	删除记录语句	11
2.2.8	退出 MINISQL 系统语句	11
2.2.9	执行 SQL 脚本文件语句	11
2.2.10	关于返回信息	12
2.2.11	关于辅助性命令语句	12
2.2.12	关于图形界面	12
2.2.13	关于代码编辑器	13
2.2.14	关于错误提示	13
第三章	实验环境	14
第四章	系统设计	15
4.1	功能描述	15
4.1.1	建立/删除索引	15
4.1.2	查找/删除索引键	16
4.1.3	插入键值对	17
4.1.4	更新值内容	17
4.1.5	图形界面	18
4.1.6	缓存管理器	18
4.1.7	记录管理器	19
4.1.8	目录管理器	19
4.1.9	API	20
4.1.10	interpreter	20
4.2	主要数据结构	20
4.2.1	B+ 树	20
4.2.2	排序数组	21
4.2.3	数据块类 (Class DataBlock)	21
4.2.4	数据缓存类 (class DataBuffer)	22
4.2.5	缓存管理器中的其它数据结构	22
4.2.6	条件类 (Class Condition)	22
4.2.7	通用基本类	23
4.2.8	异常类	23
4.2.9	command 类和 CatalogManager 类	24
4.3	类图与类间关系	25
4.3.1	B+ 树	25
4.3.2	异常类型	25

4.3.3	索引管理器	27
4.3.4	图形界面	27
4.3.5	缓存管理器	27
4.3.6	其它	28
4.4	分工情况	28
第五章	系统实现分析及运行截图	29
5.1	创建表语句	29
5.1.1	执行流程	29
5.1.2	运行截图	29
5.2	删除表语句	30
5.2.1	执行流程	30
5.2.2	运行截图	30
5.3	创建索引语句	30
5.3.1	执行流程	30
5.3.2	运行截图	31
5.4	删除索引语句	31
5.4.1	执行流程	31
5.4.2	运行截图	32
5.5	选择语句	32
5.5.1	执行流程	32
5.5.2	运行截图	33
5.6	插入记录语句	36
5.6.1	执行流程	36
5.6.2	运行截图	37
5.7	删除记录语句	37
5.7.1	执行流程	37
5.7.2	运行截图	38
5.8	退出系统语句	39
5.9	执行脚本文件语句	39
5.9.1	执行流程	39
5.9.2	运行截图	40
5.10	show 语句	40
5.10.1	执行流程	40
5.10.2	运行截图	41

目录	4
第六章 总结	43
6.1 B+ 树, 索引管理器与 GUI	43
6.2 API, interpreter 与 catalog manager	43
6.3 record, buffer manager 与 DB Files	44
第二部分 附录	45
第一章 接口说明	46
A.1 index	46
A.2 API	48
A.3 interpreter	48
A.4 catalog manager	48
A.5 record manager	51
A.6 buffer manager	53
A.7 DB Files	54
第二章 插图, 表格与列表	57

第一部分

正文

第一章 实验目的

1.1 总体实验目的

1.1.1 miniSQL

设计并实现一个精简型单用户 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。

1.1.2 设计目的

通过对 MINISQL 的设计与实现, 提高学生的系统编程能力, 加深对数据库系统原理的理解。通过编程设计, 加深对数据库系统的理解并深入了解 B+ 树这一数据结构。以模块化方式构建大型计算机软件, 提高架构抽象能力并重视模块化和解耦合在软件设计中的作用。

1.2 模块实验目的

1.2.1 索引管理器

我们负责设计的索引管理器模块主要负责数据库系统中的索引管理, 提供基于 B+ 树的索引实现并提高数据库查询/插入/删除效率。并通过提供易用接口与其他模块整合实现有效功能。

1.2.2 缓存管理器

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。

1.2.3 记录管理器

负责管理记录表中数据的数据文件。主要功能为实现数据文件的创建与删除（由表的定义与删除引起）、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一个条件的查找（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，且不要求支持记录的跨块存储。

1.2.4 目录管理器

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。2. 表中每个字段的定义信息。3. 数据库中所有索引的定义。

1.2.5 接口管理器

接口管理器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，解释器层调用。解释器层解释生成的命令内部表示为输入，根据目录管理器提供的信息确定执行规则，并调用记录管理器、索引管理器，目录管理器提供的相应接口进行执行，最后返回执行结果给解释器模块。

1.2.6 命令行解释器

解释器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部分语义正确性，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。

1.2.7 图形界面

我们通过实现图形界面以环节用户对于命令行界面的恐惧（纵使这个图形界面中用户还是要敲写 SQL 语句），通过 PyQt 等易用接口实现一个简单的带有语法高亮的编辑器界面，并用对用户友好的方式输入输出各种信息，并提供对于一般编辑器操作的支持（例如多种快捷键和其他功能）。

第二章 系统需求

2.1 需求概述

2.1.1 数据类型

只要求支持三种基本数据类型：int，char(n)，float。

2.1.2 表定义

一个表可以定义多个属性，各属性可以指定是否为 unique；每一个表必须对某一个 unique 属性定义主键，否则报错。

2.1.3 索引的建立和删除

对于表的主键自动建立 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。

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。（where 条件是范围时删除多条）

2.2 语法说明

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”，其他非黑体显示的有其他的含义，如表名并不是表示字符串“表名”，而是表示表的名称。

2.2.1 创建表语句

该语句的语法如下：

```
1  -- 语法说明
2  create table 表名 (
3      列名 类型 ,
4      列名 类型 ,
5      列名 类型 ,
6      primary key ( 列名 )
7  );
8
9  -- 示例
10 create table student (
11     sno char(8),
12     sname char(16) unique,
13     sage int,
14     sgender char (1),
15     primary key ( sno )
16 );
```

Listing 2.1: Create Table Syntax and Example

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。

2.2.2 删除表语句

该语句的语法如下：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。

```
1  -- 语法说明
2  drop table 表名 ;
3  -- 示例语句：
4  drop table student;
```

Listing 2.2: Drop Table Syntax and Example

2.2.3 创建索引语句

该语句的语法如下：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。

```
1  -- 语法说明
2  create index 索引名 on 表名 ( 列名 );
3  -- 示例语句:
4  create index stunameidx on student ( sname );
```

Listing 2.3: Create Index Syntax and Example

2.2.4 删除索引语句

该语句的语法如下：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。

```
1  -- 语法说明
2  drop index 索引名 ;
3  -- 示例语句
4  drop index stunameidx;
```

Listing 2.4: Drop Index Syntax and Example

2.2.5 选择语句

该语句的语法如下：

```
1  -- 语法说明
2  select * from 表名 ;
3  -- 或:
4  select * from 表名 where 条件 ;
5  -- 示例语句
6  select * from student;
7  select * from student where sno = '88888888';
8  select * from student where sage > 20 and sgender = 'F';
```

Listing 2.5: Select Syntax and Example

其中“条件”具有以下格式：*op and op ...and op*。*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。

2.2.6 插入记录语句

该语句的语法如下：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。

```
1  -- 语法说明
2  insert into 表名 values ( 值1 , 值2 , ... , 值n );
3  -- 示例语句
4  insert into student values ('12345678','wy',22,'M');
```

Listing 2.6: Insert Syntax and Example

2.2.7 删除记录语句

该语句的语法如下：

```
1  -- 语法说明
2  delete from 表名 ;
3  -- 或：
4  delete from 表名 where 条件 ;
5  -- 示例语句
6  delete from student;
7  delete from student where sno = '88888888';
```

Listing 2.7: Delete From Syntax and Example

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。

2.2.8 退出 miniSQL 系统语句

该语句的语法如下：

```
1  -- 语法说明
2  quit;
3  exit;
```

Listing 2.8: Quit miniSQL Syntax

2.2.9 执行 SQL 脚本文件语句

该语句的语法如下：

```
1  -- 语法说明
2  execfile 文件名 ;
```

Listing 2.9: Execute File Syntax

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。

2.2.10 关于返回信息

返回信息的内容参考 mysql, 语言限定为英文, 主要包括执行 sql 返回的行数 (n row in set), 或者影响的行数 (n row affects), 必须给出执行 SQL 所花费的时间 Duration。执行出错需要指出错误的类型, 具体位置不做要求。错误类型包括 (SYNTAX ERROR, INVALID IDENTIFIER, INVALID VALUE, INVALID ATTR FOR INDEX), 即语法错误 (语法参考 mysql, miniSQL 不支持的数据类型和 SQL 等也报语法错误), 标识符错误 (不存在的表名, 字段名等), 值错误 (变量长度不足以容纳实际数据), 索引定义在非 unique 属性上。可以参考 mysql workbench 和 mysql terminal client 的返回信息。

2.2.11 关于辅助性命令语句

我们定义了如下的辅助性质语句来方便用户使用本数据库。

```
1  -- 语法说明
2  show tables;
3  show index from 表名;
4  show create table 表名;
5
6  -- 示例
7  show tables;
8  show index from student2;
9  show create table student2;
```

Listing 2.10: 辅助语句

2.2.12 关于图形界面

我们希望开发一个图形界面以管理用户的输入输出, 提供基本按钮和快捷键。并提供图标等。



图 2.1: GUI Requirements

2.2.13 关于代码编辑器

我们希望提供一个带有代码高亮功能，自动补全功能，自动换行功能，以及各种编辑器快捷键（复制一行，列操作，关键词操作）等的完整小型 sql 语句编辑视窗。同时，我们希望 MINISQL 支持多行语句执行的操作。

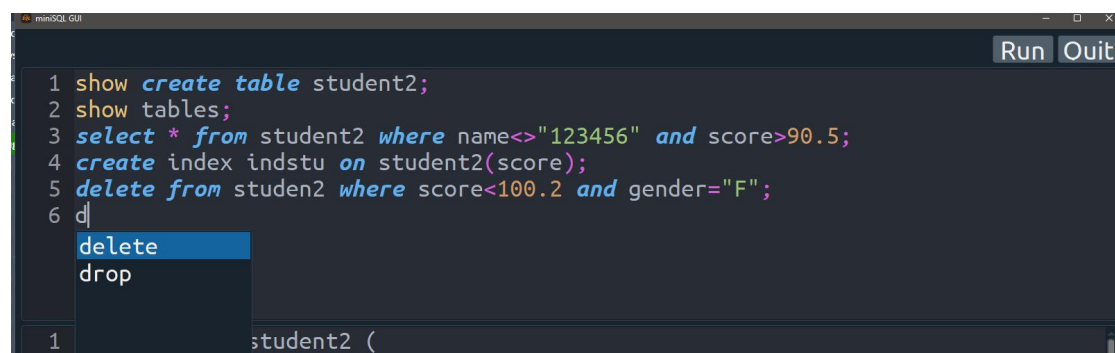


图 2.2: Editor Requirements

2.2.14 关于错误提示

我们希望错误提示以一种用户容易接受的格式呈现到程序上，例如在所有严重错误前加上 Error 标识符。结合 GUI 我们将会对这些格式化的错误信息进行高亮以方便用户识别。

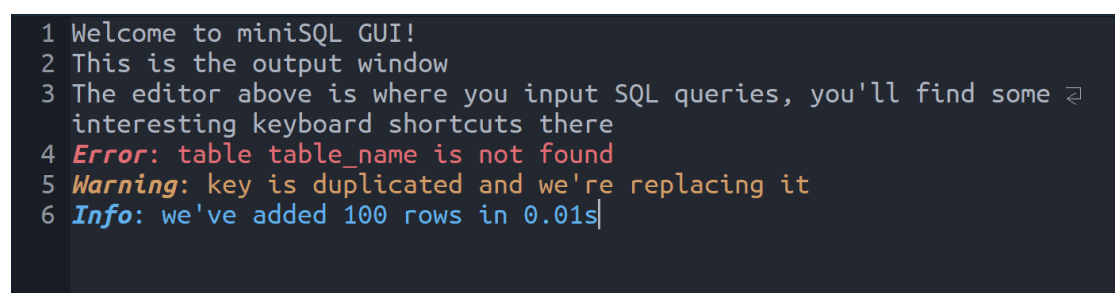


图 2.3: Error Message Requirements

第三章 实验环境

主要开发语言 Python 3.7.*/3.8.*

主要开发环境

- PyCharm 2020.1
- Visual Studio Code 1.45

经过测试的系统环境

- MICROSOFT WINDOWS [VERSION 10.0.18363.836/10.0.18362.592/10.0.18363.900]
- UBUNTU WSL2
- MACOS 10.15 CATALINA

Python 包要求 QScintilla, QDarkStyle, PyQt5, pickle, pandas, shutil, os, sys

实验处理器环境 Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 12 Logical Processors and Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz and Intel(R) Core(TM) i5 9300H CPU @ 2.40GHz (经过测试的)

实验内存环境 SODIMM 15.8GB/16.0GB and SODIMM 23.9GB/24.0GB and 7.73GB/8.00GB (经过测试的)

实验硬盘环境 KBG30ZMS512G NVMe TOSHIBA 512GB (经过测试的)

第四章 系统设计

4.1 功能描述

4.1.1 建立/删除索引

实际的数据库应用中，有在建立表时创建索引和表中有数据情况下创建索引的需求。区别在于，前者不需要其他模块为索引管理器提供数据，仅仅需要分配一个新索引所需的内存和磁盘空间；而后者要求发出建立索引请求的模块提供相应的数据块。默认情况下我们以记录的行号作为 B+ 树中键值对上的“值”，因此我们要求相应数据块是按照它们将来被查找的顺序提供的。

建立索引 正如图4.1所示的¹。为了方便演示，让我们用其中的第一列作为建立索引的数据集。我们会按顺序将提供的数据插入索引数据结构，因此在这一个键值对中，键为第一列数据值，如 10101，而值为行号，如 0。

<i>record 0</i>	10101	<i>Srinivasan</i>	<i>Comp. Sci.</i>	<i>65000</i>
<i>record 1</i>	12121	<i>Wu</i>	<i>Finance</i>	<i>90000</i>
<i>record 2</i>	15151	<i>Mozart</i>	<i>Music</i>	<i>40000</i>
<i>record 3</i>	22222	<i>Einstein</i>	<i>Physics</i>	<i>95000</i>
<i>record 4</i>	32343	<i>El Said</i>	<i>History</i>	<i>60000</i>
<i>record 5</i>	33456	<i>Gold</i>	<i>Physics</i>	<i>87000</i>
<i>record 6</i>	45565	<i>Katz</i>	<i>Comp. Sci.</i>	<i>75000</i>
<i>record 7</i>	58583	<i>Califleri</i>	<i>History</i>	<i>62000</i>
<i>record 8</i>	76543	<i>Singh</i>	<i>Finance</i>	<i>80000</i>
<i>record 9</i>	76766	<i>Crick</i>	<i>Biology</i>	<i>72000</i>
<i>record 10</i>	83821	<i>Brandt</i>	<i>Comp. Sci.</i>	<i>92000</i>
<i>record 11</i>	98345	<i>Kim</i>	<i>Elec. Eng.</i>	<i>80000</i>

图 4.1: 线性记录储存方式

¹图4.1来源于 *Database System Concepts 6th Edition Abraham Silberschatz 等*。

这样处理的原因在于，我们希望日后通过索引根据查找键快速找到记录对应的位置，而其在表格中的相对位置是最方便的寻址信息之一。若我们按照索引的自定义值储存插入的键，则索引失去加快搜索的作用；若我们插入的值为记录的绝对磁盘位置信息，则缓存管理器失效，且数据的移动会导致索引失效。

在内存中创建索引（并插入相应数据后），索引管理器会将索引的内存信息叫给缓存管理器，由其决定是否应将内存保留或者存储到磁盘中，同时返回给索引管理器一个唯一的索引标号（索引管理器会将其继续返回给上层模块），日后将根据这一唯一标识符从缓存管理器中取得相应索引（无论是通过读取磁盘文件还是直接获取内存指针）。

删除索引 我们通过上述的唯一标识符给缓存管理器发出删除信号，完成删除操作。

4.1.2 查找/删除索引键

查找和删除操作支持快速范围操作，并且两者在具体实现上有极大相似性，我们通过抽象两者的操作来提高代码复用率。我们首先直接判断用户进行的是范围还是单值查找，并且替前从缓存管理器中取得相应索引内容（内存或硬盘中）。

单值操作 我们调用 B+ 树相应查找接口获得应查找的值，并通过异常来进行错误通讯。若查找成功则直接返回，否则抛出相关异常。

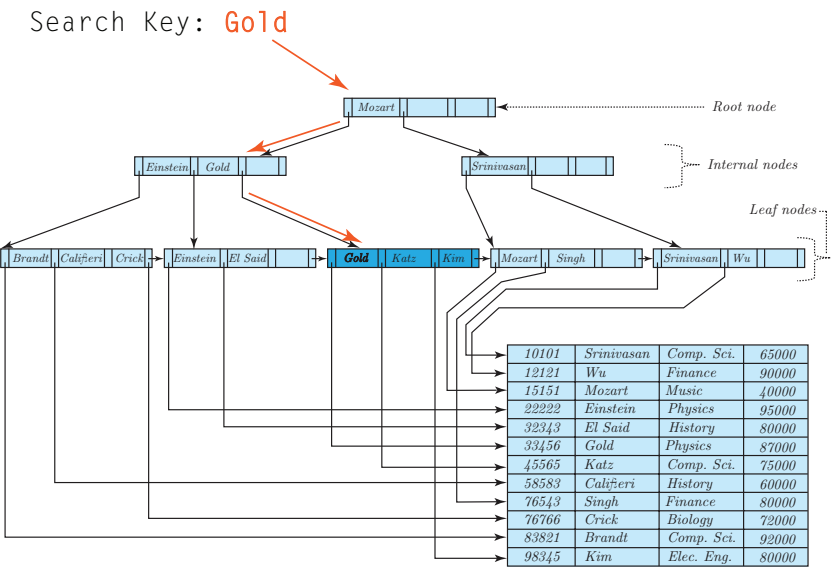


图 4.2: 单值操作

范围操作 我们首先会检查用户给予的范围是否有效²，同样的，我们使用异常来进行错误通讯，这使得接口模块能方便的实现错误处理。接着我们查询范围两端的值³并根据返回的节点情况进行相关操作，对于查找指令，这一操作是返回查询得到的相关信息；对于删除指令，这一操作是操作 B+ 树删除相关的值，并在操作全部完成后将修改后的索引叫给缓存管理器。如图4.3所示（加深部分为需要查找或删除的部分）。

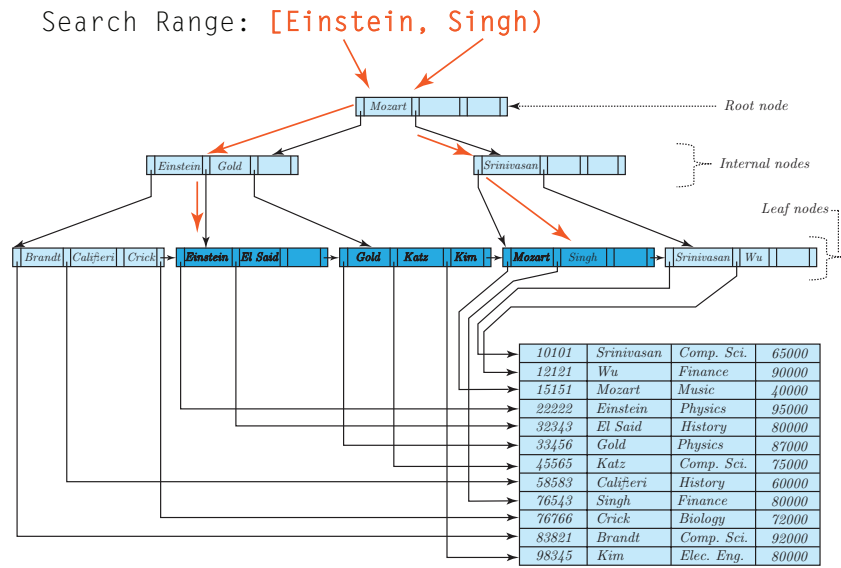


图 4.3: 范围操作

4.1.3 插入键值对

我们单独实现了插入键值对功能，因为其逻辑相对于查询和删除操作都有所不同（需检查重复元素等）。类似的，索引管理器会首先向缓存管理器请求相关索引内容。接着我们调用插入操作，将是否允许替换的信息传递给 B+ 树的相关函数直接进行操作⁴。

4.1.4 更新值内容

由于我们使索引中键值对中值指向记录在某张表中的位置，而这一位置在数据库运行过程中可能会发生很大变化⁵，我们给外界留出批量修改值的接口。

这一操作类似于范围操作中涉及到的内容，我们需要对 B+ 中的有效节点数目进行检查，根据节点数目选取不同的处理方式。

²例如，范围左右下标是否为左小右大，或被查找的树是否为空等
³我们默认范围查找的区间是左闭右开的。
⁴原因在于 B+ 树内部实现中也需要调用查找相关功能，与其规定执行流程，不如让 B+ 树具体实现获得最优执行流程
⁵例如闲时的数据库清理和记录重排序等，亦或记录管理器采取了不同的数据储存模型。

4.1.5 图形界面

我们拓展了 API 模块的接口，通过图形界面来提高用户体验。我们设计了一个编辑器界面，加入了语法提示与自动补全。我们通过 PyQt 来实现对图形界面的构建。

4.1.6 缓存管理器

我们估计一条记录占用空间大小为定值 16 字节，每块 (block) 大小固定为 4K，则每一个块中有固定记录条数 256 条。同时，设置内存中最多能够存储的数据块数为 2048 块，因此内存中数据占用的空间最大为 8M，处于可以接受的范围内。这样一来，当缓存管理器进行内部管理时，我们就可以用元组 [表名, 块位置, 记录位置] 来唯一标识任意一条记录的物理位置，如下图 1 所示 (事实上，每个表的块数也限制为 2048，即每个表最多可以有 50 万条记录，对于 miniSQL，这是足够大的)：

表名(String)	块位置(INT≥0)	记录位置(0≤INT≤255)
------------	------------	-----------------

图 4.4: 记录位置的唯一标识

为了方便模块间的信息传递，每一条记录也可以用元组 [表名, 位置] 来唯一标识，此时我们有：

$$\text{位置} = \text{块位置} \times 256 + \text{记录位置}$$

从而，无论是缓存和记录管理器之间，还是和索引管理器的信息交互，对于任一表中的任意一条记录，我们总是用一个整数‘位置’来唯一标识它，且可以和图 1 中的表示任意转化。

记录取得 (Get Blocks) 当某一表的记录被查询/删除/搜索时，缓存管理器收到记录管理器的信息，并返回所需要的块列表。由于记录在文件中的排列顺序相对随机，我们设计选择返回所需表的全部记录块给记录管理器。这可以通过遍历内存中的所有数据块和相应文件中的数据块 (如有必要) 实现。

当内存中不包含该表的全部记录信息 (即块信息) 时，必须要访问相应的表文件取得数据块。更进一步地，当内存中可以存储的块数满时，我们必须要换进换出数据块。需要注意的是，我们总是保证文件中的记录位置最大值大于等于缓存中的，这为脏数据块的写回 (write back) 提供的便利。

记录插入 (Insert Records) 当一条记录被插入时，缓存管理器收到记录管理器的信息。若该表中间的每一条记录都是有效 (包括无记录被删除过或被删除记录的位置已全部被新记录填充两种情况) 插入记录至表的最后一行并总是将记录插入到文件末尾 (记录插入文件通过调用文件管理器中相应接口完成)，否则插入记录至之前被删除记录的无效位置。

在记录被删除时，为防止其他记录的位置变动，我们选择懒惰删除的形式。这就需要有一个数据结构存储被删除的记录位置。因此，在插入记录时，我们必须要考虑之前是否曾经删除过记录。同时，为了搜索的方便（这是由记录取得 (Get Blocks) 的实现方式决定的），当在表的最后插入记录时，总是要求也将记录插入文件的末尾，以保证缓存的记录位置最大值不会超过文件中的。

插入完成后，缓存管理器返回新记录的‘位置’信息。

最近最少用 (Least Recently Used) 当内存中数据块个数达到预设的最大值时，需要选择合适的块进行替换。我们总是假设最近访问的块是最有可能再次访问的，因而选择使用最近最少用策略进行块替换。

具体而言，即选择访问时间最早的块进行替换操作，且当所选择的块为脏 (dirty) 块时（即该块中的内容比文件中新），需要先写回文件再进行替换。

保存/取得/删除索引 (Save/Get/Delete Indexes) 对于索引，更准确的说，是某个表上某属性的 B+ 树结构，为了尽量减少与文件系统的交互，提高效率，我们选择将全部索引信息存储在内存中，不作数量限制，这在表数不多时是可以接受的。只在 miniSQL 系统启动时从文件读入全部信息，在系统关闭（或空闲）时存储全部信息至文件。

因此，保存/取得/删除索引的操作均只需要对内存进行操作即可，用 Python 自带的字典数据结构可以非常容易地根据索引编号找到对应的 B+ 树，进行所需的操作。

4.1.7 记录管理器

删除/查询记录 对从缓存管理器取得的数据块列表进行按条件分析筛选，取出符合条件的记录。筛选过程基本一致，区别在于最后删除记录返回的是被删除记录的‘位置’号列表（并加入 freeList 中），查询记录返回的是记录本身构成的列表。

插入记录 调用缓存管理器完成记录的插入，不需要返回任何信息。

创建/删除/清空表 调用缓存管理器中 DataBuffer 对象的成员函数完成，不需要返回任何信息。

4.1.8 目录管理器

管理元数据 对表格（表名，主键，表的属性）和索引元数据进行插入和删除。

查询元数据 根据 API 和 interpreter 的需求，对表名，表格属性（主键，unique，带索引的属性等），索引（是否存在，索引 id 等）进行查询。

4.1.9 API

调用各模块执行指令 对于以各种形式（字符串，控制台输入，文件输入）输入的指令，进行初步分解并调用 interpreter 进行解析；对于 interpreter 解析报错后的指令，按一定顺序调用对应模块进行查错和具体执行，并以要求的格式返回给输出或 GUI。

处理报错信息 处理来自各个模块不同形式的报错信息。

4.1.10 interpreter

解析指令 将字符串型的指令解析为方便 API 处理的类。

处理可能的报错 调用 catalog manager，根据指令信息和元数据信息进行基本的报错。

4.2 主要数据结构

4.2.1 B+ 树

⁶ B+ 树是一种树数据结构，通常用于数据库和操作系统的文件系统中。B+ 树的特点是能够保持数据稳定有序，其插入与修改拥有较稳定的对数时间复杂度。B+ 树元素自底向上插入，这与二叉树恰好相反。

B+ 树在节点访问时间远远超过节点内部访问时间的时候，比可作为替代的实现有着实在的优势。这通常在多数节点在次级存储比如硬盘中的时候出现。通过最大化在每个内部节点内的子节点的数目减少树的高度，平衡操作不经常发生，而且效率增加了。这种价值得以确立通常需要每个节点在次级存储中占据完整的磁盘块或近似的大小。

B+ 背后的想法是内部节点可以有在预定范围内的可变量目的子节点。因此，B+ 树不需要像其他自平衡二叉查找树那样经常的重新平衡。对于特定的实现在子节点数目上的低和高边界是固定的。例如，在 2-3 B 树（常简称为 2-3 树）中，每个内部节点只可能有 2 或 3 个子节点。如果节点有无效数目的子节点则被当作处于违规状态。

B+ 树的创造者 Rudolf Bayer 没有解释 B 代表什么。最常见的观点是 B 代表平衡 (balanced)，因为所有的叶子节点在树中都在相同的级别上。B 也可能代表 Bayer，或者是波音 (Boeing)，因为他曾经工作于波音科学研究实验室。

查找 查找以典型的方式进行，类似于二叉查找树。起始于根节点，自顶向下遍历树，选择其分离值在要查找值的任意一边的子指针。在节点内部典型的使用是二分查找来确定这个位置。

⁶ 此段内容摘自 [Wikipedia](#)

插入 节点要处于违规状态，它必须包含在可接受范围之外数目的元素。首先，查找要插入其中的节点的位置。接着把值插入这个节点中。如果没有节点处于违规状态则处理结束。如果某个节点有过多元素，则把它分裂为两个节点，每个都有最小数目的元素。在树上递归向上继续这个处理直到到达根节点，如果根节点被分裂，则创建一个新根节点。为了使它工作，元素的最小和最大数目典型的必须选择为使最小数不小于最大数的一半。

删除 首先，查找要删除的值。接着从包含它的节点中删除这个值。如果没有节点处于违规状态则处理结束。如果节点处于违规状态则有两种可能情况：它的兄弟节点，就是同一个父节点的子节点，可以把一个或多个它的子节点转移到当前节点，而把它返回为合法状态。如果是这样，在更改父节点和两个兄弟节点的分离值之后处理结束。亦或，它的兄弟节点由于处在低边界上而没有额外的子节点。在这种情况下把两个兄弟节点合并到一个单一的节点中，而且我们递归到父节点上，因为它被删除了一个子节点。持续这个处理直到当前节点是合法状态或者到达根节点，在其上根节点的子节点被合并而且合并后的节点成为新的根节点。

4.2.2 排序数组

本数据结构是为了配合主键而实现的，采用最普通的排序数组查找方式，并在数组内部采用二分查找进行相关操作。可从无限子树数目的 B+ 树抽象的到，因此我们可以较为方便的统一两者的接口。

值得注意的是，为了配合主键和记录管理器中数据的储存方式，我们往往使用一种特殊的类作为排序数组的内部容器：一种返回当前下标的特殊数组⁷。

4.2.3 数据块类 (Class DataBlock)

本数据结构是缓存管理器保存数据的重要组成部分，其数据成员如下：

- 锁位 (pin)：表示该块是否被锁住，即不能被替换。为布尔 (bool) 类型。
- 脏位 (dirty)：表示该快的内容是否比文件中更加新，即是否被操作改变过但还未写回文件。为布尔 (bool) 类型。
- 表名 (tablename)：该块所对应的表名。为字符串 (String) 类型。
- 位置 (position)：块位置。为整数 (int) 类型。
- 内容 (content)：块的内容，包含该块的全部 (最新) 记录信息。为列表 (list) 类型，其中的每个元素均为一条记录元组，在列表中的位置对应应在磁盘中的物理位置。
- 最近访问时间 (Last Access Time, LAT)：记录该块最近一次被访问的时间。

⁷我们可以利用这一特性而使得这种储存不占用任何空间，而同时保证接口的一致性。

4.2.4 数据缓存类 (class DataBuffer)

本数据结构是缓存管理器数据管理的重要部分，以数据块类为主体，通过对数据块与文件之间的交互管理，提高了执行效率，其数据成员和成员函数如下：

- 数据缓存 (buffer)：一个存储数据块类的列表 (list)，列表元素数最大为 2048。
- 数据块数 (blockNum)：当前缓存中已有的数据块数，即数据缓存列表的元素数，为了方便作为数据成员单独列出。
- 整个缓存查找替换 (LRU)：遍历数据缓存，找到访问时间最早的块，若是脏的，写回；最后将该块在列表中删除。
- 记录插入 (insert_record)：插入记录。
- 整个缓存除某表外替换 (LRU_except_one)：遍历除某一表外的数据缓存块，进行 LRU 操作。
- 记录取得 (get_blocks)：取得某表的全部数据块。
- 创建表 (create_table)：创建一个新的表 (调用文件管理器)。
- 删除表 (delete_table)：删除一个现有表 (调用文件管理器)。
- 清空表 (clear_table)：清空一个现有表的全部数据 (调用文件管理器)。

4.2.5 缓存管理器中的其它数据结构

freeList 为一个字典，键值对为 [表名: 对应已删除元素的列表]。

index_buffer 为一个字典，键值对为 [标号: 对应 B+ 树]。

maxrecordNum 为一个字典，键值对为 [表名: 该表记录最大‘位置’号]。

4.2.6 条件类 (Class Condition)

本数据结构对记录管理器筛选数据起着重要作用，其数据成员如下：

- 属性 (attribute)：记录是该表第几个属性上的条件，为一个整数。
- 类型 (type)：记录条件的类型 (等于，小于，大于等)，为一个整数。
- 值 (value)：条件中参与比较的值。

4.2.7 通用基本类

对 attribute, table, index 定义了三个模块间通用的基本类并存放在 minisqlclass.py 中, 通过 import 调用。

```

1 class Attribute:
2     def __init__(self, name="", type="", length=0, isP=False, isU=False):
3         self.name = name
4         self.type = type
5         self.length = length
6         self.isPrimary = isP
7         self.isUnique = isU
8 class Table:
9     def __init__(self, name="", attributeList=list(), size=0):
10        self.name = name
11        self.attributeList = attributeList[:]    #attribute 类
12        self.attributeNum = len(attributeList)
13        self.size = size
14 class Index:
15     def __init__(self, tableName="", indexName="", attrName="", id=-1):
16        self.table_name = tableName
17        self.index_name = indexName
18        self.index_id = id
19        self.attribute_name = attrName
20 class Condition:
21     def __init__(self, attribute_name="", op=' ', operand=None):
22        self.attribute_name = attribute_name
23        self.op = op
24        self.operand = operand

```

Listing 4.1: 通用基本类说明

4.2.8 异常类

定义了 error 类, 处理了 interpreter 能够发现的大部分错误类型。

```

1 class error_type:
2     def __init__(self):
3         self.NONE = ""
4         self.syn = "Error : syntax error"
5         self.ivld_cmd = "Error : no query specified"
6         self.exist_t = "Error : table {} already exists"    # todo: add format

```



```

7      self.exist_i = "Error : index on {} on table {} already exists" # todo: add
      ↪ format
8      self.exist_attrb = "Error : attribute {} on table {} is exist" # todo: add
      ↪ format
9      self.not_exist_t = "Error : table {} does not exist" # todo: add format
10     self.not_exist_i = "Error : index {} does not exist" # todo: add format
11     self.not_exist_a = "Error : attribute {} on table {} does not exist" # todo:
      ↪ add format
12     self.ivld_char = "Error : invalid char"
13     self.ivld_dt_tp = "Error : invalid data type"
14     self.no_prim_k = "Error : no primary key"
15     self.invalid_prim_k = "Error : invalid primary key"
16     self.not_exist_k = "Error : key {} on attribute {} on table {} does not exist"
      ↪ # todo: add format
17     self.ept_i = "Error : empty index {} on table {}" # todo: add format
18     self.ept_t = "Error : empty table {}" # todo: add format
19     self.no_unq_a = "Error : the attribute {} on table {} is not unique" # todo:
      ↪ add format
20     self.insert_not_match = "Error : inserted data not match"
21     self.out_of_range = "Error : data out of range"

```

Listing 4.2: 异常类说明

4.2.9 command 类和 CatalogManager 类

定义了 interpreter 中的 command 类和 catalog manager 类。

```

1  class command:
2      def __init__(self, catalog_manager):
3          self.error_tp = error.NONE
4          self.catalog = catalog_manager
5  class CatalogManager:
6      def __init__(self, table=[], index=[]):
7          self.table = table[:] #table 类
8          self.index = index[:] #index 类
9          self.tableNum = int(len(table))
10         self.indexNum = len(index)
11         self.indexcnt = 0

```

Listing 4.3: command 类和 CatalogManager 类说明

4.3 类图与类间关系

4.3.1 B+ 树

B+ 树的类图实现如图4.5所示⁸⁹。

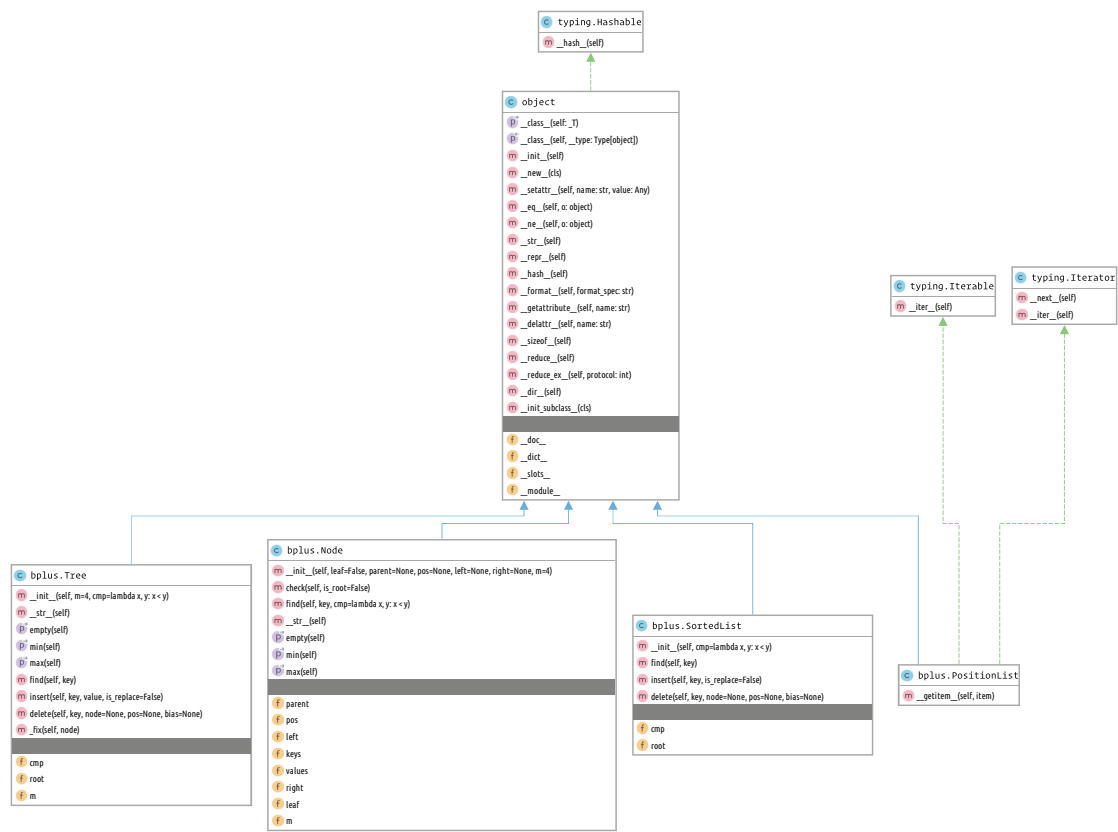


图 4.5: B+ 树的类图与类间关系

4.3.2 异常类型

程序使用的异常类图如4.6所示。

⁸为了类图完整性，我们也列举了除用户自定义类型以外的类型。

⁹我们使用矢量图渲染了字体，若图表过小请放大查看。

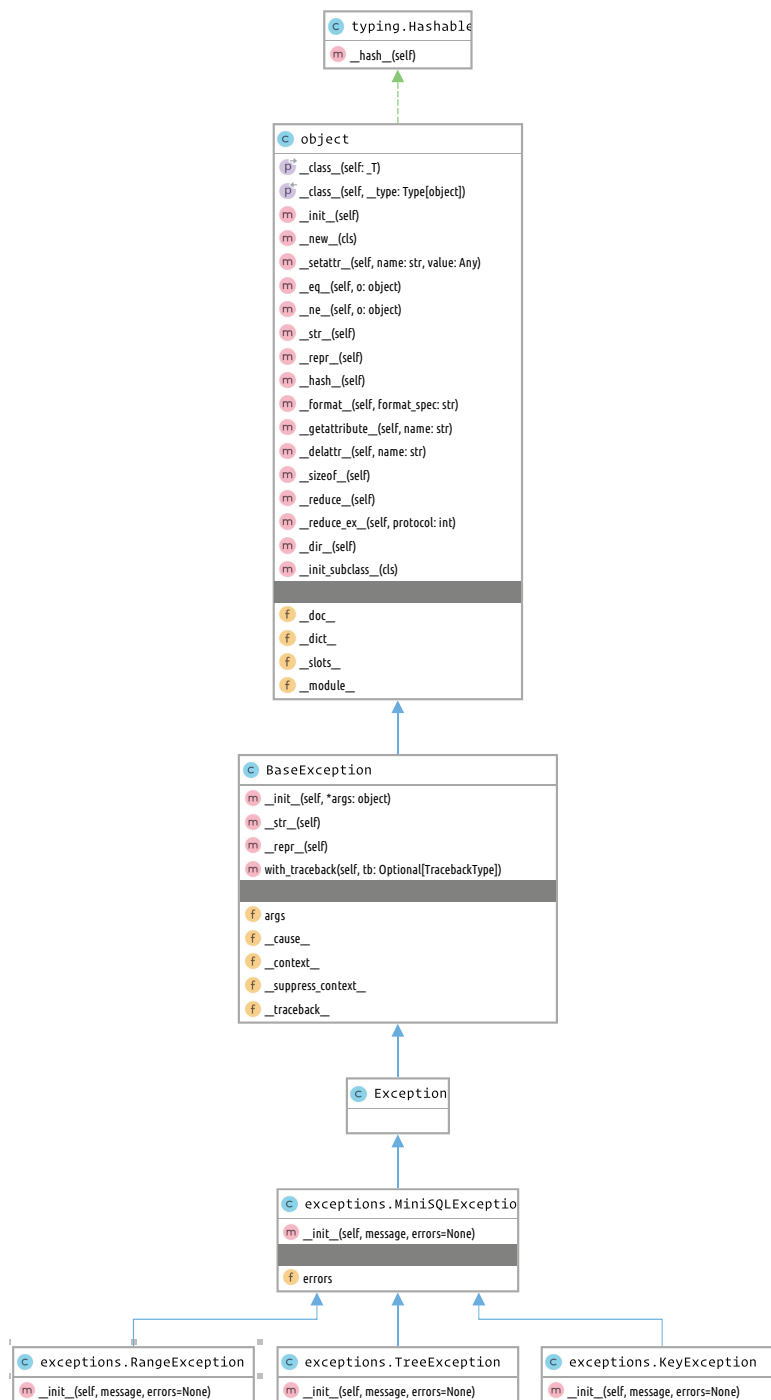


图 4.6: 异常的类型与类间关系

4.3.3 索引管理器

在具体的索引管理器实现上，我们采用了模块层面的抽象而非类层面的，这更贴合 Python 语言的风格。这样能保证尽量大的抽象层次与代码复用率。值得注意的是，我们在实现 B+ 树相关操作的时候也使用了静态函数来抽象部分内容¹⁰。

4.3.4 图形界面

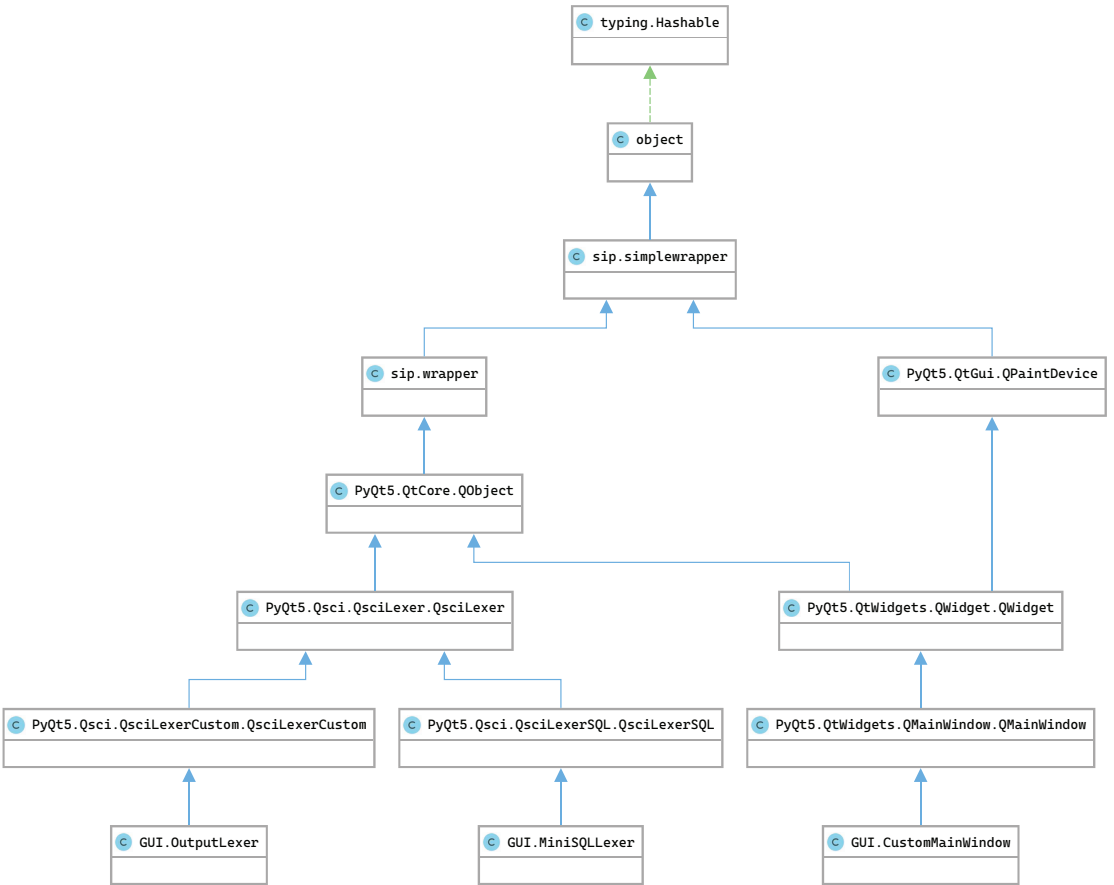


图 4.7: 图形界面类图

4.3.5 缓存管理器

在具体的缓存管理器实现上，我们采用了类层面的抽象，尽可能多的利用类成员函数解决问题。

¹⁰我们将在下一部分详细阐述实现细节

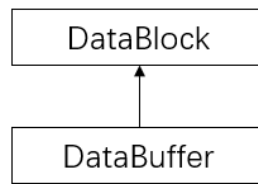


图 4.8: 缓存管理器类间关系

4.3.6 其它

对通用基本类，类间关系为 table 类使用了 attribute 类。

对 command 类和 catalog manager 类，类间关系为 command 包括了 catalog 类，而 catalog 类包括了上面描述的三种基本类。

4.4 分工情况

徐震 负责索引管理器，图形界面模块。

曾一欣 负责 interpreter，API，目录管理器模块。

毛一恒 负责记录管理器，缓存管理器，文件系统模块。

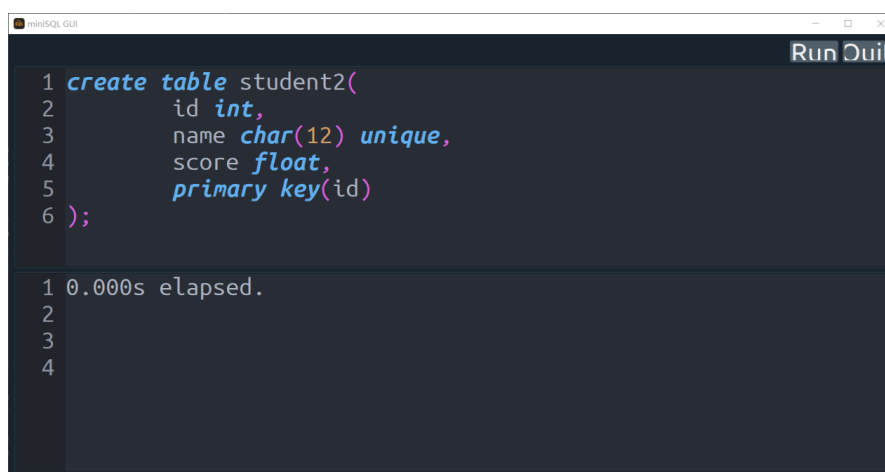
第五章 系统实现分析及运行截图

5.1 创建表语句

5.1.1 执行流程

- 1.GUI 读入，调用 API；
- 2.API 调用 interpreter 解析并处理错误；
3. 返回 API，调用 catalog manager，record manager 和 index manager；
- 4.record manager 调用 buffer manager，再由 buffer manager 调用 file manager 建表；
- 5.index manager 调用 buffer manager 建主键 index；
- 6.API 计算运行时间，返回给 GUI 输出。

5.1.2 运行截图



The screenshot shows a window titled "minSQL GUI" with a dark background. The top right corner has "Run" and "Quit" buttons. The main area displays a SQL command: `1 create table student2(`, `2 id int,`, `3 name char(12) unique,`, `4 score float,`, `5 primary key(id)`, `6);`. Below the command, the output shows: `1 0.000s elapsed.`, `2`, `3`, and `4`.

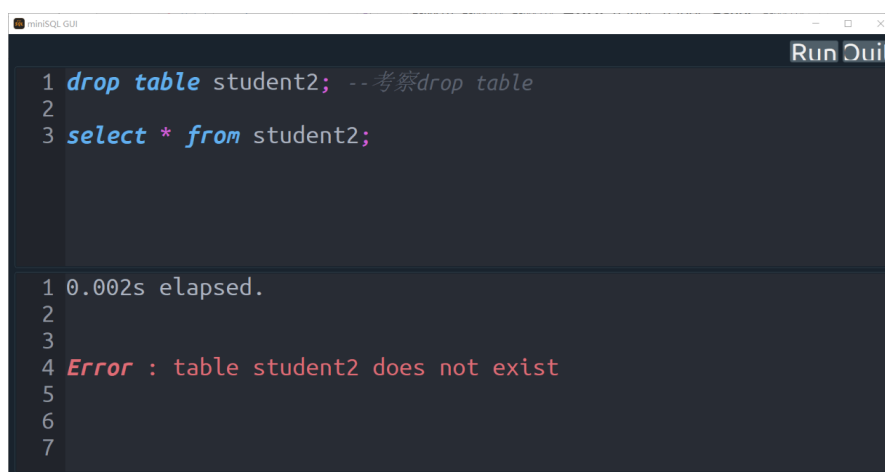
图 5.1: create table

5.2 删除表语句

5.2.1 执行流程

- 1.GUI 读入, 调用 API;
- 2.API 调用 interpreter 解析并处理错误;
3. 返回 API, 调用 catalog manager 得到要删除的表的 index, 调用 index manager 删除这些 index (index manager 将调用 buffer manager);
4. 调用 catalog manager 和 record manager 删除表 (record manager 将调用 buffer manager 并由 buffer manager 调用 file manager 进行删除);
- 5.API 统计执行时间, 返回给 GUI 输出。

5.2.2 运行截图



```
miniSQL GUI
1 drop table student2; --考察drop table
2
3 select * from student2;

1 0.002s elapsed.
2
3
4 Error : table student2 does not exist
5
6
7
```

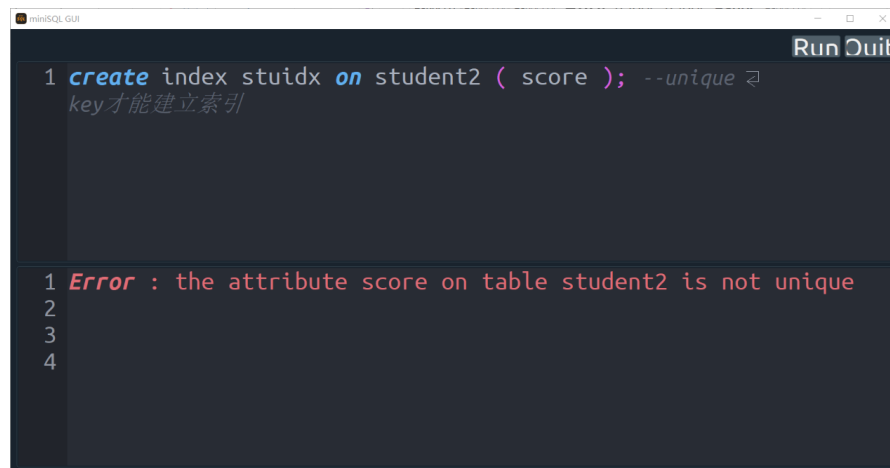
图 5.2: drop table

5.3 创建索引语句

5.3.1 执行流程

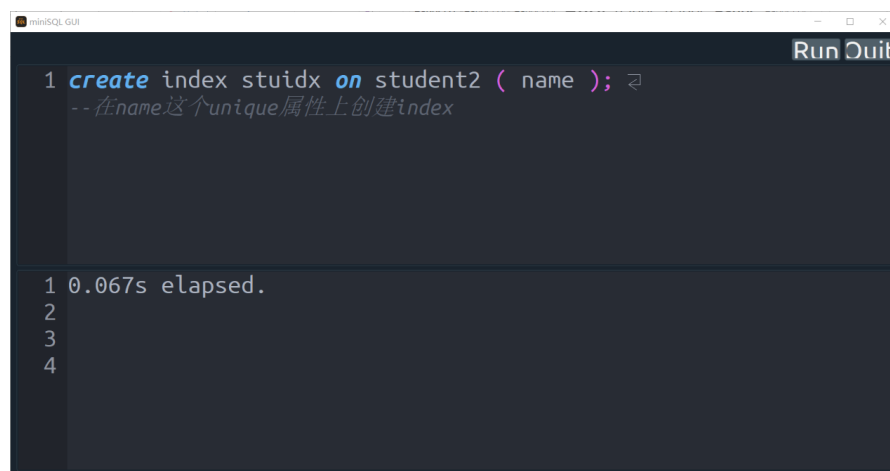
- 1.GUI 读入, 调用 API;
- 2.API 调用 interpreter 解析并处理错误;
3. 返回 API, 调用 catalog manager 建立 index 并返回 index id, 调用 record manager 得到要建 index 的 attribute 的数据, 调用 index manager 对这些数据建立 index;
- 4.API 统计执行时间, 返回给 GUI 输出。

5.3.2 运行截图



The screenshot shows the minSQL GUI window. The command entered is `1 create index stuidx on student2 (score); --unique`. Below the command, there is a comment in Chinese: `key才能建立索引`. The output shows an error: `1 Error : the attribute score on table student2 is not unique`. The window has a 'Run' button and a 'Quit' button in the top right corner.

图 5.3: create index-failed



The screenshot shows the minSQL GUI window. The command entered is `1 create index stuidx on student2 (name);`. Below the command, there is a comment in Chinese: `-- 在name这个unique属性上创建index`. The output shows the command was successful: `1 0.067s elapsed.`. The window has a 'Run' button and a 'Quit' button in the top right corner.

图 5.4: create index-succeed

5.4 删除索引语句

5.4.1 执行流程

- 1.GUI 读入, 调用 API;
- 2.API 调用 interpreter 解析并处理错误;
3. 返回 API, 调用 catalog manager 得到要删除的 index 的 id 并删除 index, 调用 index manager 根据 id 删除 index (index manager 将调用 buffer manager);

4.API 统计执行时间，返回给 GUI 输出。

5.4.2 运行截图

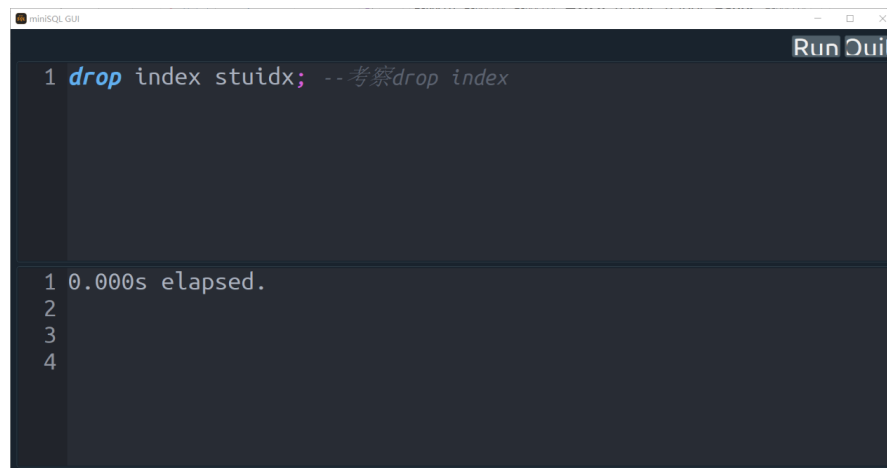


图 5.5: drop index

5.5 选择语句

5.5.1 执行流程

- 1.GUI 读入，调用 API;
- 2.API 调用 interpreter 解析并处理错误;
3. 返回 API，这里分为两种类型，一种是查询的 attribute 包含 index 的，一种是不包含 index 的;
4. 对于包含 index 的查询，调用 index manager 进行查询并返回查询结果的行号（将调用 buffer manager），将行号和（可能有的）剩余的查询要求交给 record manager 进行查询（可能需调用 buffer manager，并由 buffer manager 调用 file manager 得到数据 block);
5. 对于不包含 index 的查询，直接调用 record manager 同上;
- 6.API 统计执行时间，返回给 GUI 输出。

5.5.2 运行截图

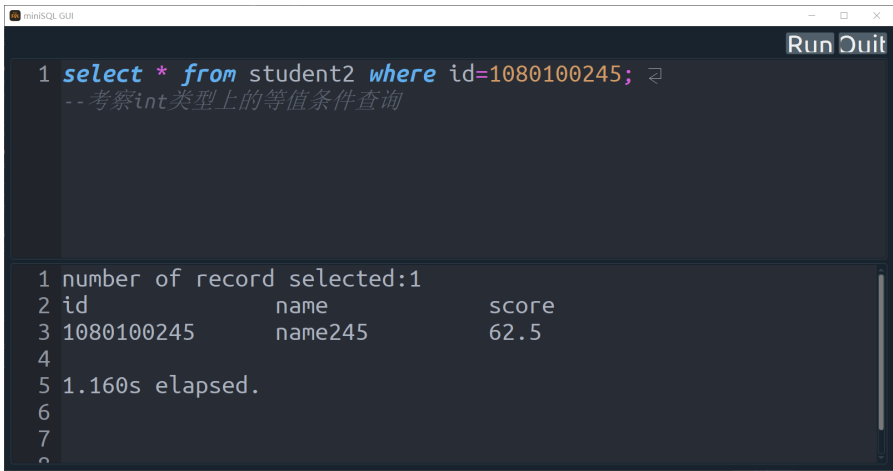


图 5.6: select-int+'='

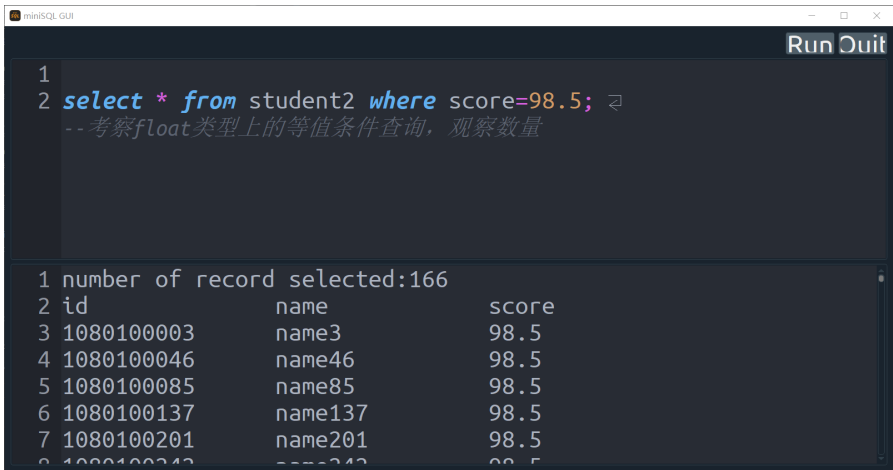


图 5.7: select-float+'='

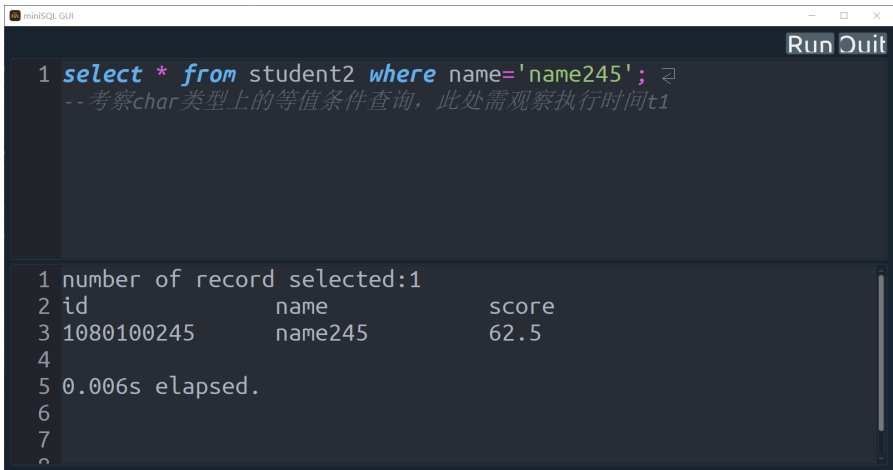


图 5.8: select-char+'='

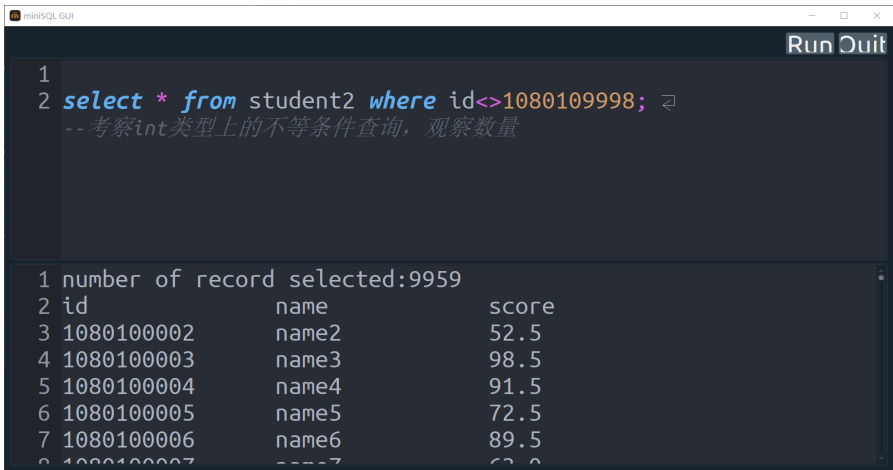


图 5.9: select-int+'<>'

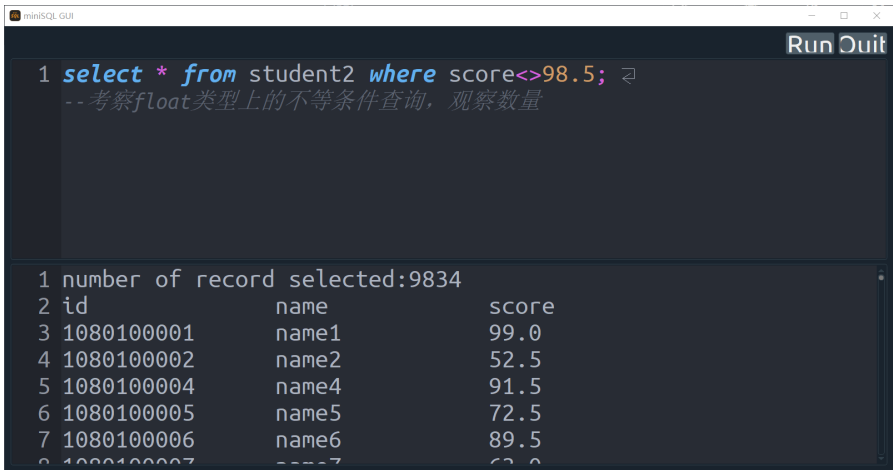


图 5.10: select-float+'<>'

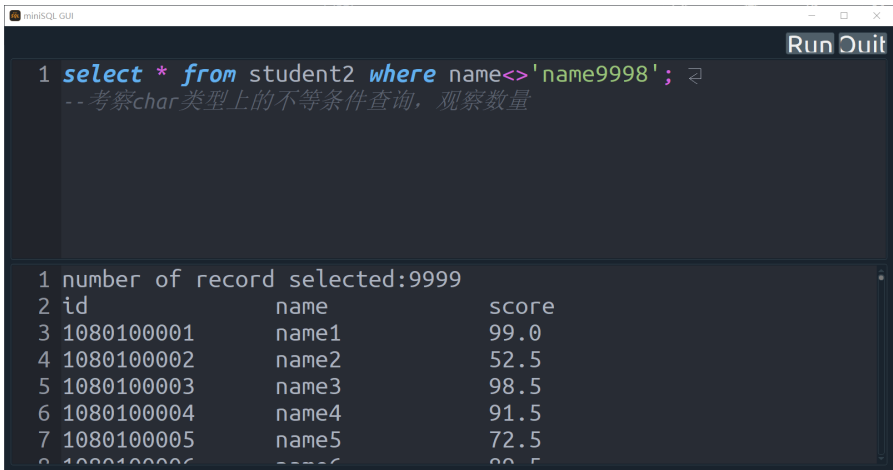


图 5.11: select-char+'<>'

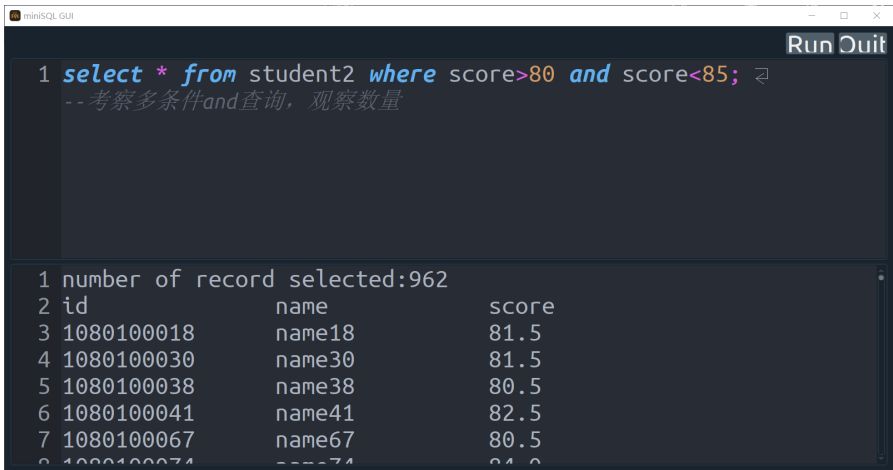


图 5.12: select-and

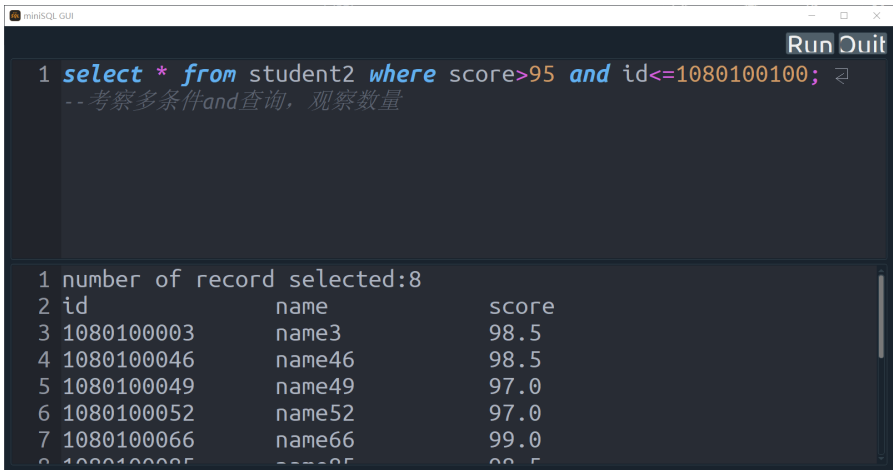


图 5.13: select-and

5.6 插入记录语句

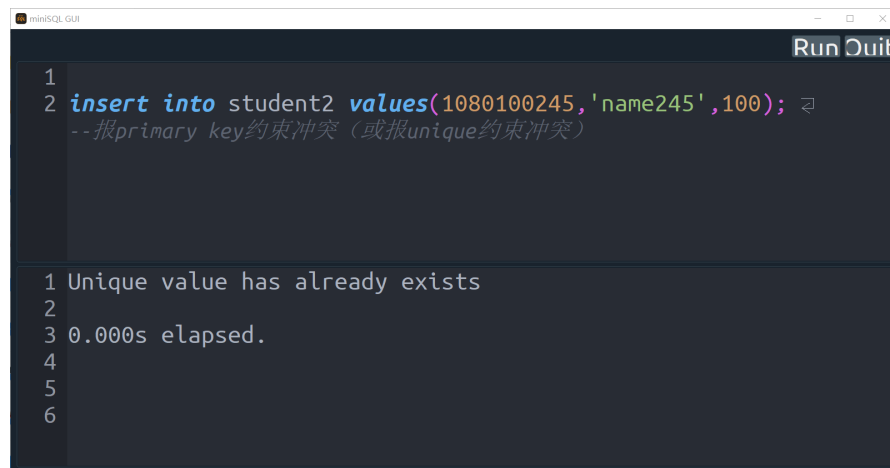
5.6.1 执行流程

- 1.GUI 读入, 调用 API;
- 2.API 调用 interpreter 解析并处理错误;
3. 返回 API, 调用 catalog manager 得到 unique 并进行检查, 得到 index id 并调用 index manager 进行 insert (将调用 buffer manager);
4. 调用 record manager 进行 insert (将调用 buffer 并可能调用 file manager 存取数据)

;

5.API 统计执行时间，返回给 GUI 输出。

5.6.2 运行截图



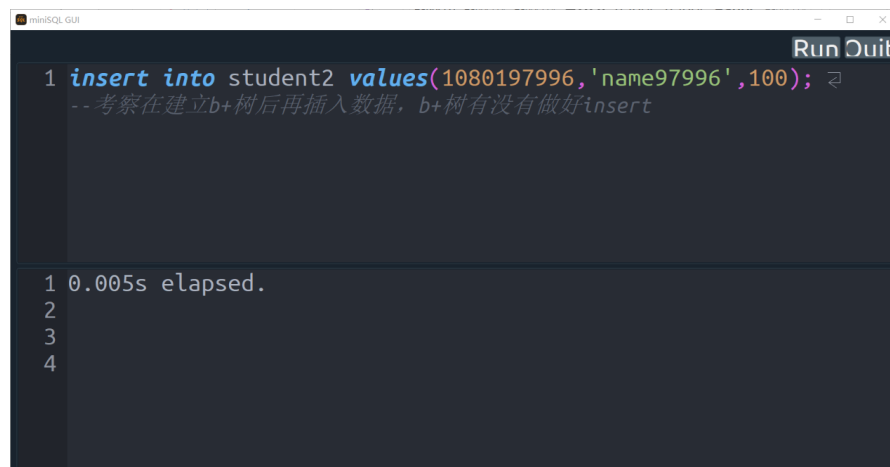
The screenshot shows the miniSQL GUI window. The input area contains the following SQL command and comment:

```
1  
2 insert into student2 values(1080100245,'name245',100);  
--报primary key约束冲突 (或报unique约束冲突)
```

The output area shows the following messages:

```
1 Unique value has already exists  
2  
3 0.000s elapsed.  
4  
5  
6
```

图 5.14: insert-failed



The screenshot shows the miniSQL GUI window. The input area contains the following SQL command and comment:

```
1 insert into student2 values(1080197996,'name97996',100);  
-- 考察在建立b+树后再插入数据，b+树有没有做好insert
```

The output area shows the following messages:

```
1 0.005s elapsed.  
2  
3  
4
```

图 5.15: insert-succeed

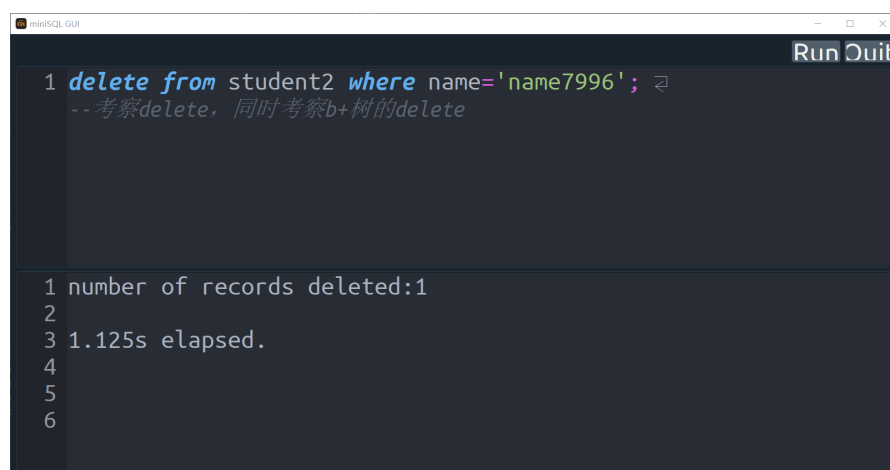
5.7 删除记录语句

5.7.1 执行流程

1.GUI 读入，调用 API;

- 2.API 调用 interpreter 解析并处理错误;
3. 返回 API, 调用 catalog manager 得到 index id 并调用 index manager 对对应 index 的数据进行删除 (将用到 buffer manager), 调用 record manager 进行删除 (将用到 buffer manager 并可能调用 file manager 进行文件读写);
- 4.API 统计执行时间, 返回给 GUI 输出。

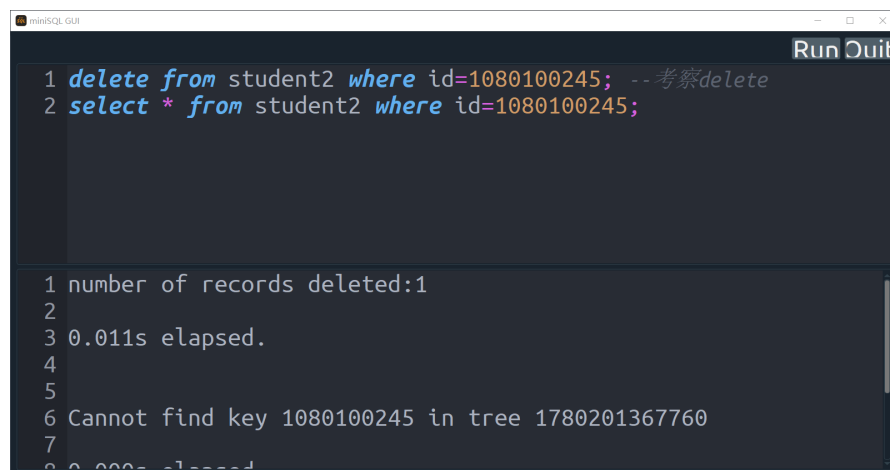
5.7.2 运行截图



```
minSQL GUI
1 delete from student2 where name='name7996';
-- 考察delete, 同时考察b+树的delete

1 number of records deleted:1
2
3 1.125s elapsed.
4
5
6
```

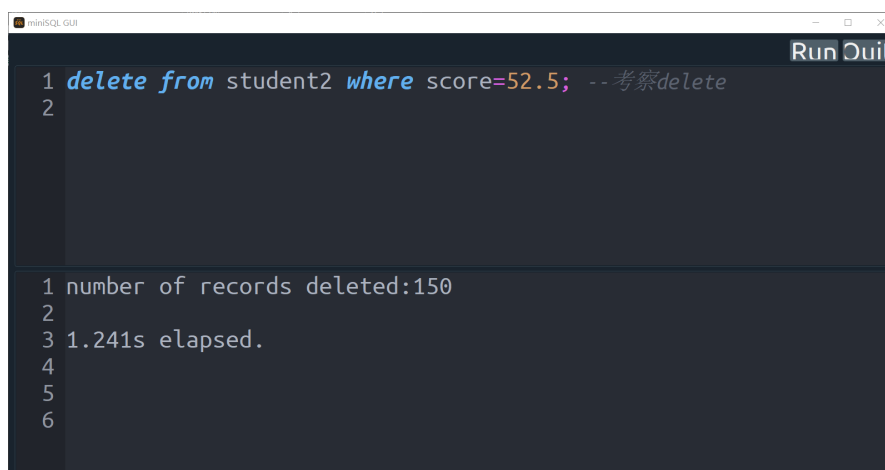
图 5.16: delete-use name(index)



```
minSQL GUI
1 delete from student2 where id=1080100245; -- 考察delete
2 select * from student2 where id=1080100245;

1 number of records deleted:1
2
3 0.011s elapsed.
4
5
6 Cannot find key 1080100245 in tree 1780201367760
7
8
```

图 5.17: delete-use primary key

The screenshot shows a window titled "miniSQL GUI" with a dark background. At the top right, there are "Run" and "Quit" buttons. The main area is a text editor with line numbers 1 and 2 on the left. Line 1 contains the SQL command: `delete from student2 where score=52.5; --考察delete`. Below the editor, there is a text area showing the output of the command: `1 number of records deleted:150`, `2`, `3 1.241s elapsed.`, `4`, `5`, and `6` on separate lines.

```
miniSQL GUI
Run Quit
1 delete from student2 where score=52.5; --考察delete
2
1 number of records deleted:150
2
3 1.241s elapsed.
4
5
6
```

图 5.18: delete-use score

5.8 退出系统语句

可以通过“quit;”语句和 GUI 界面中的 quit 按钮进行。

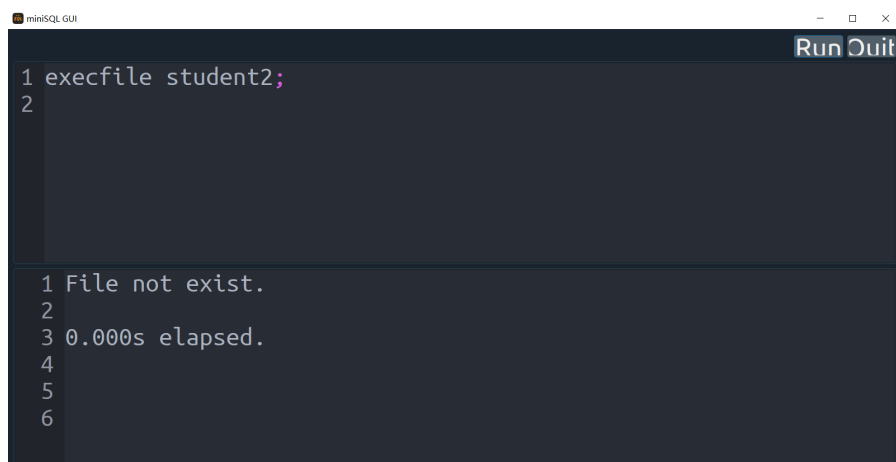
退出时调用 buffer manager->file manager 将 data, catalog 和 index 的数据都存入文件。

5.9 执行脚本文件语句

5.9.1 执行流程

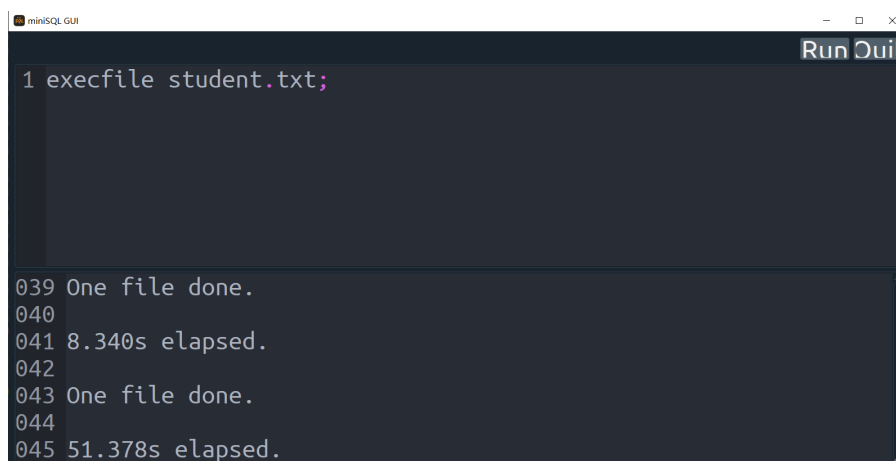
- 1.GUI 读入，调用 API;
- 2.API 调用 interpreter 解析并处理错误;
3. 返回 API，读文件，将文件中读到的指令传给 API 中相应处理函数;
- 4.API 统计执行时间，返回给 GUI 输出并输出“one file done”。

5.9.2 运行截图



The screenshot shows the miniSQL GUI window. The input field contains the command `1 execfile student2;`. The output field displays the following text: `1 File not exist.`, `2`, `3 0.000s elapsed.`, `4`, `5`, and `6`. The window has a title bar with 'miniSQL GUI' and standard window controls. A 'Run' button and a 'Quit' button are visible in the top right corner.

图 5.19: execfile-failed



The screenshot shows the miniSQL GUI window. The input field contains the command `1 execfile student.txt;`. The output field displays the following text: `039 One file done.`, `040`, `041 8.340s elapsed.`, `042`, `043 One file done.`, `044`, and `045 51.378s elapsed.`. The window has a title bar with 'miniSQL GUI' and standard window controls. A 'Run' button and a 'Quit' button are visible in the top right corner.

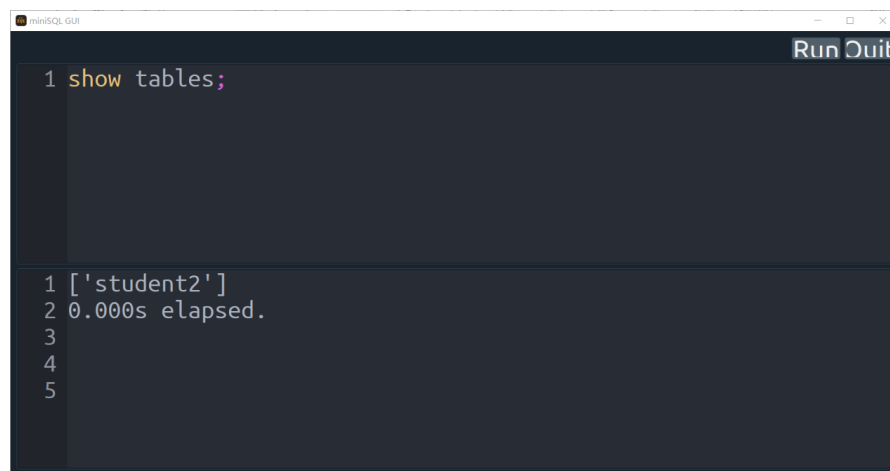
图 5.20: execfile-succeed

5.10 show 语句

5.10.1 执行流程

- 1.GUI 读入，调用 API；
- 2.API 调用 interpreter 解析并处理错误；
3. 返回 API，调用 catalog manager 对需要显示的信息进行查询；
- 4.API 统计执行时间，返回给 GUI 输出。

5.10.2 运行截图

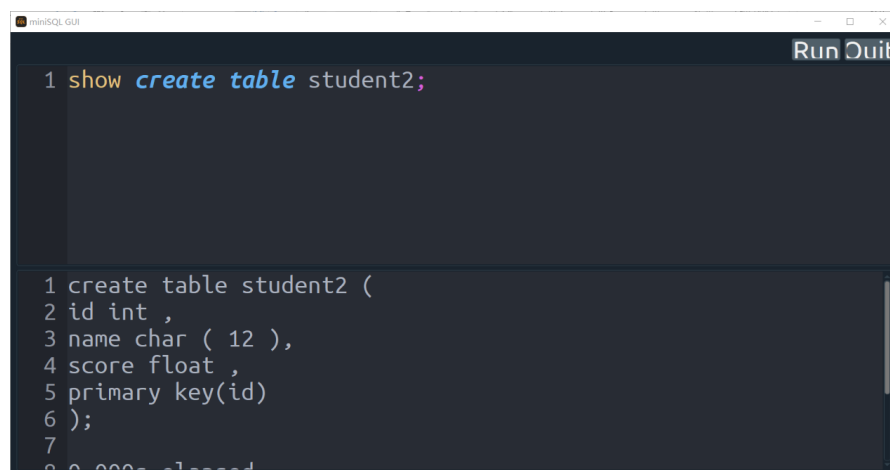


The screenshot shows a window titled "miniSQL GUI" with a dark background. The command "1 show tables;" is entered in the input area. The output area displays the results: "1 ['student2']", "2 0.000s elapsed.", "3", "4", and "5". A "Run" button and a "Quit" button are visible in the top right corner.

```
1 show tables;
```

```
1 ['student2']
2 0.000s elapsed.
3
4
5
```

图 5.21: show tables

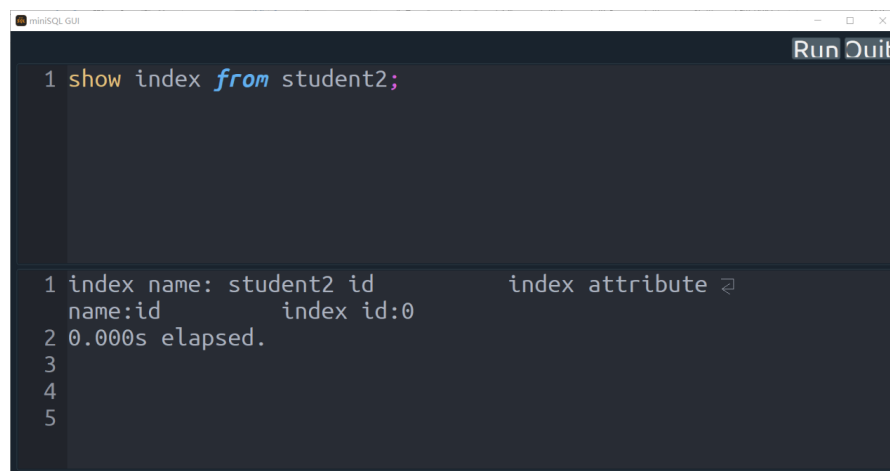


The screenshot shows a window titled "miniSQL GUI" with a dark background. The command "1 show create table student2;" is entered in the input area. The output area displays the table creation statement: "1 create table student2 (", "2 id int ,", "3 name char (12),", "4 score float ,", "5 primary key(id)", "6);", "7", and "8 0.000s elapsed.". A "Run" button and a "Quit" button are visible in the top right corner.

```
1 show create table student2;
```

```
1 create table student2 (
2 id int ,
3 name char ( 12 ),
4 score float ,
5 primary key(id)
6 );
7
8 0.000s elapsed.
```

图 5.22: show create table



```
miniSQL GUI
1 show index from student2;

1 index name: student2 id      index attribute ↗
  name:id                    index id:0
2 0.000s elapsed.
3
4
5
```

图 5.23: show index

第六章 总结

6.1 B+ 树，索引管理器与 GUI

在本实验中，我们实现了 B+ 树模块。借助 Python 的动态类型语言特性，我们的 B+ 树可以有效对各种 Python 支持的数据类型进行操作。通过进行单元测试，我们在 miniSQL 项目整体进行的前期就保证了 B+ 树模块的正确性，为后续调试整合过程带来了极大方便。

接着，我们实现了 B+ 树中的索引相关操作。并封装成了一个易用的模块，使得 API 可以方便的调用。

此次实验中我们深刻感受到了小组合作与提前定义接口的重要性，并且在 B+ 树具体实现过程中，我们对单元测试的作用有了更进一步的了解。

我们通过设计 GUI 进一步了解了程序设计过程中与人类交互方式的重要性。我们发现图形界面设计与命令行程序各有优劣，经过研究发现很多程序都采用命令行程序提供核心功能，接着通过图形界面将这些核心功能包裹。例如 Visual Studio 的 cl.exe 编译器

6.2 API, interpreter 与 catalog manager

在本实验中，我们实现了 API, catalog manager 和 interpreter 模块。这三个模块是所有模块中难度相对较低，但是需要细、和耐心以及对 minisql 的实现流程和具体要求足够了解的部分。

得益于小组另外两位成员的认真努力，调用封装好的 record manager 和 index manager 模块大大降低了 API 的实现难度。但在完成过程中我们也遇到了很多问题，如果能够更好更仔细地交流好接口和类，这些问题其实能够更轻松解决。

对于我个人来说，这是第一次通过 git 与小组成员合作完成代码，因为自己对这种合作模式的不熟悉给大家造成了不少麻烦；整个实验下来，最深的感受还是用更好的平台进行合作，确实能够方便小组交流合作，节省上传/下载/合并代码的时间。同时，在对 minisql 各种错误提示的处理中，也学到了利用 python 的 exception 和 error 类解决各种错误返回问题的方法，对如何写适用于可能状况百出的现实应用的代码有了基础的简单了解。

6.3 record, buffer manager 与 DB Files

在本实验中，我们实现了 Record Manager, Buffer Manager, DB Files 三个模块。这三个模块的关联性较大，难度适中，需要对数据库的内部有一定的理解，设计合适的存储形式并有耐心地一步步实现。

此次实验中，我深刻体会到了小组讨论与合作中提前定义接口的重要性。一开始，我不清楚应该从哪里入手，应该如何实现。幸运的是，在小组多次讨论后，终于有了一个比较清晰完整的实现思路。在小组成员的包容和帮助下，我最终实现了三个模块，并与小组成员实现的模块顺利对接接口。并在不断的打磨中尽量减少了 bug。

本次实验我们全程通过 git 来实现代码的共享和历史版本保存，方便了小组的线上实时交流合作，节约了时间，提高了效率。从数据库的知识角度看，我在对 Buffer Manager 的设计中更加深入理解了 buffer 在数据库数据查询/删除等操作中起到的作用，同时对行存储、列存储等多种形式存储方式有了一定的了解。从编程语言 (python) 角度看，我从几乎一窍不通到可以写一些代码，了解了 python 的基本类型和语法，理解了 pickle, pandas 等 python 包的强大之处。

第二部分

附录

第一章 接口说明

A.1 index

```
1 def create_index(ind, data_list, cmp=dummy_cmp, is_primary=False):
2     """
3     :param ind: the id of the index to be saved to file
4     :param data_list: the data, as list, to create index on
5     :param cmp: the comparator of the index, defaults to operator<
6     :param is_primary: whether we're dealing with primary key, using sorted list
7     :return: index of the newly created table
8     """
9
10
11 def drop_index(ind):
12     """
13     :param ind: the id of the index
14     :return: currently nothing is returned
15     """
16
17
18 def insert(ind, key, value, is_replace=False):
19     """
20     :param ind: the id of the index
21     :param key: the key to insert into the index
22     :param value: the value of the B+ tree, probably the line number of the inserted
    ↪ item
23     :param is_replace: whether we should replace on duplication
24     :raise KeyException: duplication
25     """
26
27
28 def search(ind, key, is_greater=None, is_current=None, is_range=False,
    ↪ is_not_equal=False):
```

```

29      """
30      A thin wrapper around _operate
31      :param is_current: whether we want a single value range search with current node
32      :param is_greater: whether we want a single value range search of greater than
33      :param ind: the id of the index to be deleted on
34      :param key: the key/keys to be deleted (single or range)
35      :param is_range: are we searching in range?
36      :return: currently nothing is returned
37      """
38
39
40  def delete(ind, key, is_greater=None, is_current=None, is_range=False,
41  ↪ is_not_equal=False):
42      """
43      A thin wrapper around _operate
44      :param is_current: whether we want a single value range search with current node
45      :param is_greater: whether we want a single value range search of greater than
46      :param ind: the id of the index to be searched on
47      :param key: the key/keys to be searched (single or range)
48      :return: currently nothing is returned
49      """
50
51  def get_values(ind):
52      """
53      get every value from the bplus tree for printing all information fast enough
54      :param ind: id of the index whose value is to be updated
55      """
56
57
58  def update_values(ind, values):
59      """
60      updates information about an index
61      :param ind: id of the index whose value is to be updated
62      :param values: the new values to be set to the index
63      :return:
64      """

```

Listing A.1: 接口说明

A.2 API

```

1 def command_prompt(file_file=None,str_command=None):
2     """
3     :param file_file: the file this function is expected to load and prompt
4     :param str_command: the string this function is expected to load and prompt
5     :return: if str_command is not None, return the string type running result
6     """
7
8 def str_main(input):
9     """
10    :param input: string type instructions
11    :return: string type running result of input instructions
12    """
13
14 def main():
15     """
16    running in cmd and print the result
17    """

```

Listing A.2: 接口说明

A.3 interpreter

```

1 def translate(self, instruction):
2     """
3     :param instruction: string type instruction
4     :return: dictionary type information of instruction or (class) error
5     """

```

Listing A.3: 接口说明

A.4 catalog manager

```

1 def check_table(self, name):
2     """
3     :param name:string type name of table
4     :return: 1 if the table exists, 0 otherwise
5     """

```

```
6
7 def check_index(self, table_name, attribute_name):
8     """
9     :param table_name: string type name of table
10    :param attribute_name: string type name of attribute
11    :return: 1 if the attribute in this table has index, 0 otherwise
12    """
13
14 def check_unique(self, table_name, attribute_name):
15     """
16    :param table_name: string type name of table
17    :param attribute_name: string type name of attribute
18    :return: 1 if the attribute in this table is unique, 0 otherwise
19    """
20
21 def check_attribute(self, table_name, attribute_name):
22     """
23    :param table_name: string type name of table
24    :param attribute_name: string type name of attribute
25    :return: 1 if the attribute in this table exists, 0 otherwise
26    """
27
28 def get_unique(self, table_name):
29     """
30    :param table_name: string type name of table
31    :return: list of names of unique attributes in this table
32    """
33
34 def get_index(self, table_name):
35     """
36    :param table_name: string type name of table
37    :return: list of (class) index in this table
38    """
39
40 def create_table(self, table, index):
41     """
42    create a new table in catalog
43    :param table: (class) table
44    :param table: (class) index of the primary key
45    """
46
47 def create_index(self, index):
```

```
48     """
49     create a new index in catalog
50     :param table: (class) index
51     """
52
53 def drop_table(self, table_name):
54     """
55     drop a table in catalog
56     :param table_name: string type name of table
57     """
58
59 def drop_index(self, index_name):
60     """
61     drop a index in catalog
62     :param index_name: string type name of index
63     """
64
65 def check_index_name(self, index_name):
66     """
67     :param index_name: string type name of index
68     :return: 1 if the index exists, 0 otherwise
69     """
70
71 def get_attribute(self, table_name):
72     """
73     :param table_name: string type name of table
74     :return: list of [name,type,length] of attributes in this table
75     """
76
77 def get_primary(self, table_name):
78     """
79     :param table_name: string type name of table
80     :return: name of key attribtue
81     """
82
83 def get_index_attribute(self, table):
84     """
85     :param table_name: string type name of table
86     :return: list of attribtue name of index in this table
87     """
88
89 def get_index_info(self, index_name):
```

```

90     """
91     :param index_name: string type name of index
92     :return: [table name, attribute name, index id] of the index
93     """
94
95     def get_attribute_type(self, table_name, attr_name):
96         """
97         :param table_name: string type name of table
98         :param attr_name: string type name of attribute
99         :return: type of the attribute in this table
100        """
101
102     def get_index_id(self, table_name, attribute_name):
103         """
104         :param table_name: string type name of table
105         :param attribtue_name: string type name of attribute
106         :return: index id of the index of the attribute in this table
107        """
108
109     def get_attribute_cnt(self, table_name, attr_name):
110         """
111         :param table_name: string type name of table
112         :param attr_name: string type name of attribute
113         :return: number the arributes from 1 to n, return the number of required arribute
114        """
115
116     def get_all_table(self):
117         """
118         :return: list of table names in this database
119        """

```

Listing A.4: 接口说明

A.5 record manager

```

1     def insert_record(fname, record):
2         """
3         :param fname: the name of the table
4         :param record: a tuple of a record
5         :return: the insertion position of the record
6         """

```

```
7
8 def delete_record_with_Index(fname, indList, conditionList):
9     """
10    delete one or a serial of records using or not using Index
11    if use Index, the indList is a list, and the indList is 0 if not
12    :param fname: the name of the table
13    :param indList: the position of selected records from index manager
14    :param conditionList: a list of objects of class Condition
15    :return: the list of position of deleted records
16    """
17
18 def select_record_with_Index(fname, indList, conditionList):
19     """
20    select one or a serial of records using or not using Index
21    if use Index, the indList is a list, and the indList is 0 if not
22    :param fname: the name of the table
23    :param indList: the position of selected records from index manager
24    :param conditionList: a list of objects of class Condition
25    :return: the list of selected records
26    """
27
28 def create_table(fname):
29     """
30    :param fname: the name of the table
31    """
32
33 def delete_table(fname):
34     """
35    :param fname: the name of the table
36    """
37
38 def clear_table(fname):
39     """
40    :param fname: the name of the table
41    """
```

Listing A.5: 接口说明

A.6 buffer manager

```
1 def insert_record(self, tname, record):
2     """
3     :param tname: the name of the table
4     :param record: a tuple of a record
5     :return: the insertion position of the record
6     """
7
8 def get_blocks(self, tname):
9     """
10    :param tname: the name of the table
11    :return: a list of all the blocks in the table
12    """
13
14 def create_table(self, tname):
15     """
16    :param tname: the name of the table
17    """
18
19 def delete_table(self, tname):
20     """
21    not only delete the table file, but also in buffer
22    :param tname: the name of the table
23    """
24
25 def clear_table(self, tname):
26     """
27    not only clear the table file, but also in buffer
28    :param tname: the name of the table
29    """
30
31 def get_index(Ind):
32     """
33    :param Ind: the serial number of a B+ tree
34    :return: a B+ tree structure
35    """
36
37 def save_index(tree, Ind):
38     """
39    :param tree: a B+ tree structure
40    :param Ind: the serial number of a B+ tree
```

```

41     """
42
43 def delete_index(Ind):
44     """
45     :param Ind: the serial number of a B+ tree
46     """
47
48 def initialize_buffer():
49     """
50     initialize the freeList, index_buffer, maxrecordNum(get datas from files)
51     """
52
53 def quit_buffer():
54     """
55     save the freeList, index_buffer, maxrecordNum into files
56     """

```

Listing A.6: 接口说明

A.7 DB Files

```

1 def initialize_file():
2     """
3     if there is no Document Folders 'data', 'index', 'catalog', create them
4     """
5
6 def insert_new_record(fname, record):
7     """
8     :param fname: the name of the table
9     :param record: a tuple of a record
10    """
11
12 def write_back(data_block):
13     """
14     write the content of the data_block back into the file, called when LRU happens
15     :param data_block: an object of the class DataBlock
16     """
17
18 def get_block_data(fname, blockpos):
19     """
20     :param fname: the name of the table

```

```
21         :param blockpos: the position of the certain block in the file
22         :return: the content of one block in the file at the position of blockpos
23         """
24
25     def create_data_file(fname):
26         """
27         create a file of a table
28         :param fname: the name of the table
29         """
30
31     def delete_data_file(fname):
32         """
33         delete the file of a table
34         :param fname: the name of the table
35         """
36
37     def clear_data_file(fname):
38         """
39         clear the file of a table
40         :param fname: the name of the table
41         """
42
43     def save_index_file(index_buffer):
44         """
45         :param index_buffer: buffer of index in buffer manager
46         """
47
48     def get_index_file():
49         """
50         get the index_buffer from file when initializing
51         :return: the index_buffer
52         """
53
54     def save_catalog_file(catalogManager):
55         """
56         :param catalogManager: catalog(contain the metadata) in catalog manager
57         """
58
59     def get_catalog_file():
60         """
61         get the metadata for catalog manager from file when initializing
62         :return: the catalog(contain the metadata)
```



```
63     """
64
65 def save_freeList(freeList):
66     """
67     :param freeList: freeList in buffer manager to save
68     """
69
70 def get_freeList():
71     """
72     get the freeList for buffer manager from file when initializing
73     :return: the freeList
74     """
75
76 def save_maxrecordNum(maxrecordNum):
77     """
78     :param maxrecordNum: freeList in buffer manager to save
79     """
80
81 def get_maxrecordNum():
82     """
83     get the maxrecordNum for buffer manager from file when initializing
84     :return: the maxrecordNum
85     """
```

Listing A.7: 接口说明

第二章 插图，表格与列表

插图

2.1	GUI Requirements	13
2.2	Editor Requirements	13
2.3	Error Message Requirements	13
4.1	线性记录储存方式	15
4.2	单值操作	16
4.3	范围操作	17
4.4	记录位置的唯一标识	18
4.5	B+ 树的类图与类间关系	25
4.6	异常的类型图与类间关系	26
4.7	图形界面类图	27
4.8	缓存管理器类间关系	28
5.1	create table	29
5.2	drop table	30
5.3	create index-failed	31
5.4	create index-succeed	31
5.5	drop index	32
5.6	select-int+'='	33
5.7	select-float+'='	33
5.8	select-char+'='	34
5.9	select-int+'<>'	34
5.10	select-float+'<>'	35
5.11	select-char+'<>'	35
5.12	select-and	36
5.13	select-and	36
5.14	insert-failed	37

5.15 insert-succeed	37
5.16 delete-use name(index)	38
5.17 delete-use primary key	38
5.18 delete-use score	39
5.19 execfile-failed	40
5.20 execfile-succeed	40
5.21 show tables	41
5.22 show create table	41
5.23 show index	42

List of Listings

2.1	Create Table Syntax and Example	9
2.2	Drop Table Syntax and Example	9
2.3	Create Index Syntax and Example	10
2.4	Drop Index Syntax and Example	10
2.5	Select Syntax and Example	10
2.6	Insert Syntax and Example	11
2.7	Delete From Syntax and Example	11
2.8	Quit miniSQL Syntax	11
2.9	Execute File Syntax	12
2.10	辅助语句	12
4.1	通用基本类说明	23
4.2	异常类说明	24
4.3	command 类和 CatalogManager 类说明	24
A.1	接口说明	47
A.2	接口说明	48
A.3	接口说明	48
A.4	接口说明	51
A.5	接口说明	52
A.6	接口说明	54
A.7	接口说明	56