MiniSQL

成员信息:

巩德志 3200105088

贺嘉豪 3200102857

钱行健 3200103868

目录

分工说明

1. Buffer and disk manager ---------- 钱行健
2. Record manager---------- 钱行健
3. Index manager---------- 贺嘉豪
4. Catalog manager---------- 贺嘉豪 巩德志
5. Executor---------- 贺嘉豪 巩德志

第一章

1.1 MiniSQL实现功能分析

1）总功能：设计并实现一个精简型单用户SQL引擎MiniSQL，允许用户通过字符界面输入SQL语句实现基本的增删改查操作，并能够通过索引来优化性能。

2） 数据类型：要求支持三种基本数据类型：`integer`，`char(n)`，`float`。

3）表定义：一个表可以定义多达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。

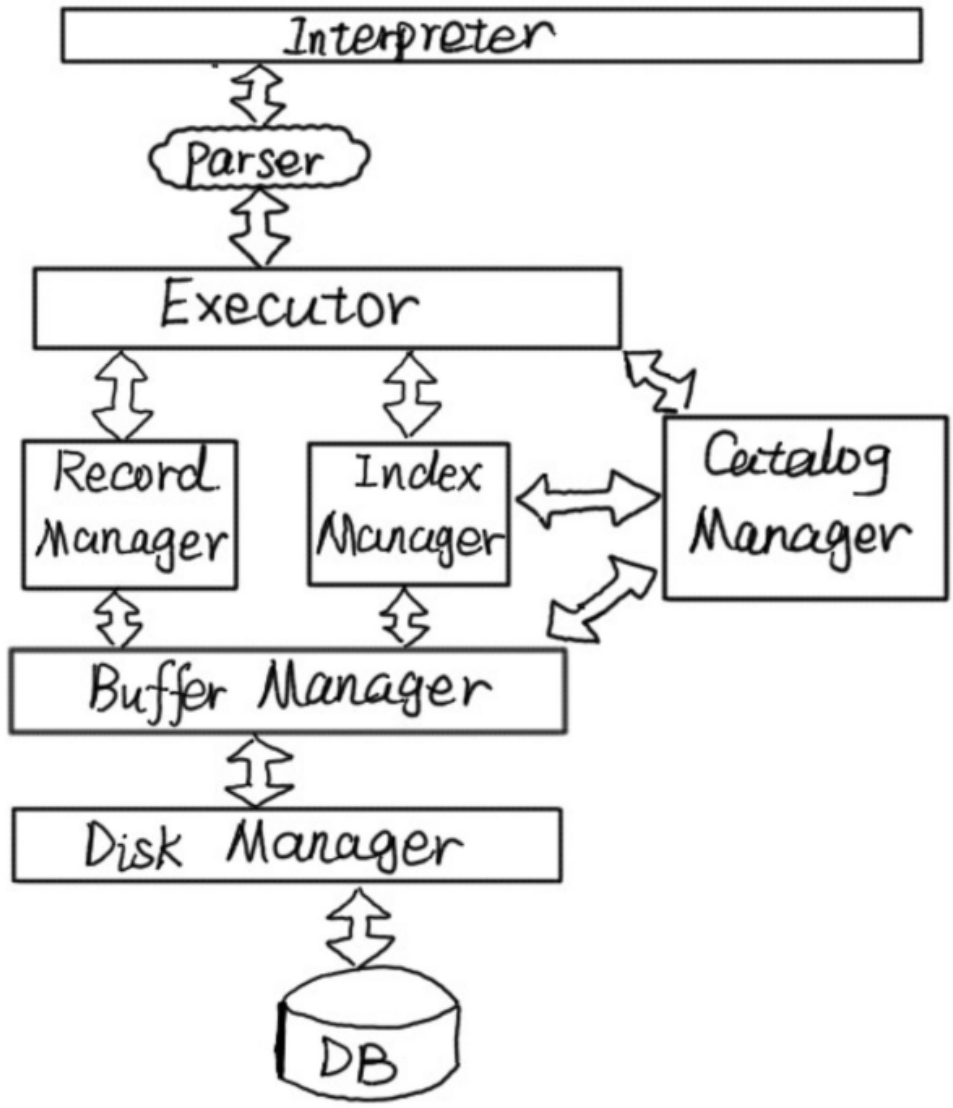
4）索引定义：对于表的主属性自动建立B+树索引，对于声明为`unique`的属性也需要建立B+树索引。

5）数据操作: 可以通过`and`或`or`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 MiniSQL系统体系结构

1.2.1 整体架构

首先，MiniSQL主要由7个部分组成:Parser、Executor、Catalog Manager、Index Manager、Record Manager、Buffer Pool Manager和Disk Managaer，其架构图如下所示:



1.2.2 各部分功能简述

(1) Parser:

● 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。

● 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，对正确的命令生成语法树，然后调用执行器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

(2) Executor:

● Executor（执行器）的主要功能是根据解释器（Parser）生成的语法树，通过Catalog Manager 提供的信息生成执行计划，并调用 Record Manager、Index Manager 和 Catalog Manager 提供的相应接口进行执行，最后通过执行上下文将执行结果返回给上层模块。

(3) Catalog Manager

● Catalog Manager 负责管理数据库的所有模式信息，包括：

a. 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。

b. 表中每个字段的定义信息，包括字段类型、是否唯一等。

c. 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，供执行器使用。

(4) Index Manager

● Index Manager 负责数据表索引的实现和管理，包括：索引（B+树等形式）的创建和删除，索引键的等值查找，索引键的范围查找（返回对应的迭代器），以及插入和删除键值等操作，并对外提供相应的接口。

● B+树索引中的节点大小应与缓冲区的数据页大小相同，B+树的叉数由节点大小与索引键大小计算得到

(5) Record Manager

● Record Manager 负责管理数据表中记录。所有的记录以堆表（Table Heap）的形式进行组织。

● Record Manager 的主要功能包括：记录的插入、删除与查找操作，并对外提供相应的接口。其中查找操作返回的是符合条件记录的起始迭代器，对迭代器的迭代访问操作由执行器（Executor）进行

(6) Buffer Pool Manager

● Buffer Manager 负责缓冲区的管理，主要功能包括：

a. 根据需要，从磁盘中读取指定的数据页到缓冲区中或将缓冲区中的数据页转储（Flush）到磁盘；

b. 实现缓冲区的替换算法，当缓冲区满时选择合适的数据页进行替换；

c. 记录缓冲区中各页的状态，如是否是脏页（Dirty Page）、是否被锁定（Pin）等；

d. 提供缓冲区页的锁定功能，被锁定的页将不允许替换。

(7) Disk Manager

● Disk Manager负责DB File中数据页的分配和回收，以及数据页中数据的读取和写入。

1.3 设计语言与运行环境

1. 设计语言： C++
2. 开发工具： VSCODE
3. 编译&开发环境：

   WSL-Ubuntu 20.04

gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0

g++ (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0

GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2

cmake version 3.16.3

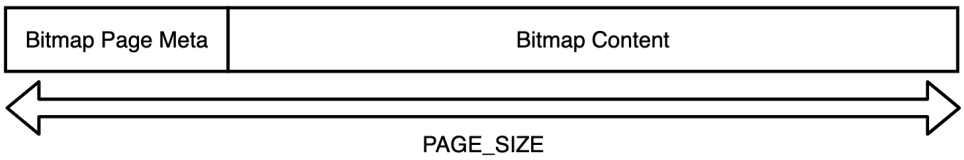
第二章

2.1 Disk and buffer manager

2.1.1 位图页

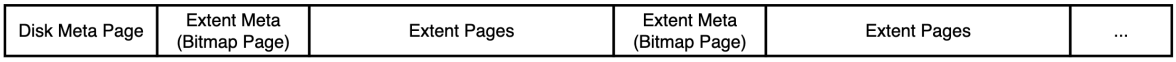
实现一个简单的位图页（Bitmap Page），位图页是Disk Manager模块中的一部分，是实现磁盘页分配与回收工作的必要功能组件。位图页与数据页一样，占用`PAGE_SIZE`（4KB）的空间，标记一段连续页的分配情况。

Bitmap Page由两部分组成，一部分是用于加速Bitmap内部查找的元信息（Bitmap Page Meta），它可以包含当前已经分配的页的数量（`page_allocated_`）以及下一个空闲的数据页（`next_free_page_`）。除去元信息外，页中剩余的部分就是Bitmap存储的具体数据。



2.1.2 磁盘数据页管理

在实现了基本的位图页后，我们就可以通过一个位图页加上一段连续的数据页（数据页的数量取决于位图页最大能够支持的比特数）来对磁盘文件（DB File）中数据页进行分配和回收。把一个位图页加一段连续的数据页看成数据库文件中的一个分区（Extent），再通过一个额外的元信息页来记录这些分区的信息。通过这种“套娃”的方式，来使磁盘文件能够维护更多的数据页信息。其主要结构如下图所示：



Disk Meta Page是数据库文件中的第`0`个数据页，它维护了分区相关的信息，如分区的数量、每个分区中已经分配的页的数量等等。接下来，每一个分区都包含了一个位图页和一段连续的数据页。

然而实际上真正存储数据的数据页是不连续的。

物理页号	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**...**
职责	磁盘元数据	位图页	数据页	数据页	数据页	位图页	数据页	
逻辑页号	/	/	0	1	2		3	

为了使得上层的Buffer Pool Manager对于Disk Manager中的页分配是无感知的（对于上层的Buffer Pool Manager来说，希望连续分配得到的页号是连续的0, 1, 2, 3...），在Disk Manager中需要对页号做一个映射（映射成上表中的逻辑页号）。

2.1.3 基于LRU替换策略的替换器

Buffer Pool Replacer负责跟踪Buffer Pool中数据页的使用情况，并在Buffer Pool没有空闲页时决定替换哪一个数据页。需要实现一个`LRUReplacer`。

LRU是Least Recently Used的缩写，即最近最少使用，是一种常用的页面置换算法，选择最近最久未使用的页面予以淘汰。记录每个页自上次被访问以来所经历的时间 t，当须淘汰一个页时，选择现有页中其 t 值最大的，即最近最少使用的页予以淘汰。

2.1.4 缓冲池管理

Buffer Pool Manager负责从Disk Manager中获取数据页并将它们存储在内存中，并在必要时将脏页面转储到磁盘中（如需要为新的页面腾出空间）。

在`BufferPoolManager`的实现中，需要用到此前已经实现的`LRUReplacer`或是其它的`Replacer`，它将被用于跟踪`Page`对象何时被访问，以便`BufferPoolManager`决定在Buffer Pool中没有空闲页可以用于分配时替换哪个数据页。

2.2 Record manager

2.2.1 数据的序列化和反序列化

为了能够持久化存储上面提到的`Row`、`Field`、`Schema`和`Column`对象，我们需要提供一种能够将这些对象序列化成字节流（`char*`）的方法，以写入数据页中。与之相对，为了能够从磁盘中恢复这些对象，我们同样需要能够提供一种反序列化的方法，从数据页的`char*`类型的字节流中反序列化出我们需要的对象。总而言之，序列化和反序列化操作实际上是将数据库系统中的对象（包括记录、索引、目录等）进行内外存格式转化的过程，前者将内存中的逻辑数据（即对象）通过一定的方式，转换成便于在文件中存储的物理数据，后者则从存储的物理数据中恢复出逻辑数据，两者的目的都是为了实现数据的持久化。

为了确保数据能够正确存储，在上述提到的`Row`、`Schema`和`Column`对象中都引入了魔数`MAGIC_NUM`，它在序列化时被写入到字节流的头部并在反序列化中被读出以验证在反序列化时生成的对象是否符合预期。

需要完善`Row`、`Schema`和`Column`对象各自的`SerializeTo`、`DeserializeFrom`和`GetSerializedSize`方法。

2.2.2 堆表

堆表（`TableHeap`）是一种将记录以无序堆的形式进行组织的数据结构，不同的数据页（`TablePage`）之间通过双向链表连接。堆表中的记录通过`RowId`进行定位。`RowId`记录了该行记录所在的`page_id`和`slot_num`，其中`slot_num`用于定位记录在这个数据页中的下标位置。

堆表中的每个数据页都由表头（Table Page Header）、空闲空间（Free Space）和已经插入的数据（Inserted Tuples）三部分组成。表头在页中从左往右扩展，记录了`PrevPageId`、`NextPageId`、`FreeSpacePointer`以及每条记录在`TablePage`中的偏移和长度；插入的记录在页中从右向左扩展，每次插入记录时会将`FreeSpacePointer`的位置向左移动。

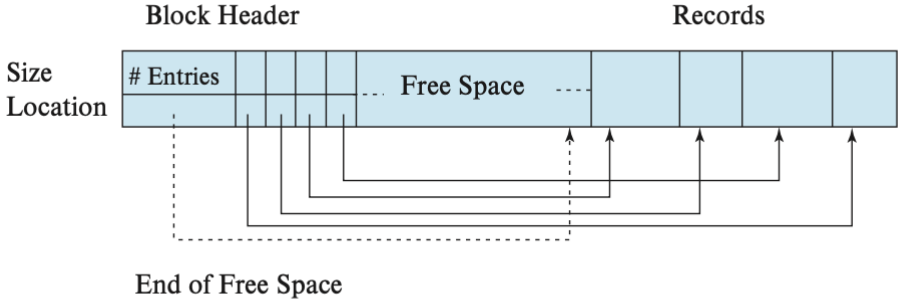


Figure 13.6 Slotted-page structure.

当向堆表中插入一条记录时，一种简单的做法是，沿着`TablePage`构成的链表依次查找，直到找到第一个能够容纳该记录的`TablePage`（*First Fit* 策略）。当需要从堆表中删除指定`RowId`对应的记录时，框架中提供了一种逻辑删除的方案，即通过打上Delete Mask来标记记录被删除，在之后某个时间段再从物理意义上真正删除该记录（本节中需要完成的任务之一）。对于更新操作，需要分两种情况进行考虑，一种是`TablePage`能够容纳下更新后的数据，另一种则是`TablePage`不能够容纳下更新后的数据，前者直接在数据页中进行更新即可，后者的实现方式留给同学们自行思考。此外，在堆表中还需要实现迭代器`TableIterator`，以便上层模块遍历堆表中的所有记录。

2.3 Index manager

2.3.1 索引键的序列化和反序列化

由于B+树的每个结点最后都要存储在disk中，因此要实现信息的持久化存储，需要采用序列化与反序列化的方式。

即向索引中插入或进行删除操作时，需要将对应的需要修改的结点页面通过buffer pool manager取出，然后将取出的页面中存储的数据进行反序列化，恢复出相应的B+树结点，在对取出的结点做出更改之后，需要将该节点进行序列化进行存储，从而实现信息的落盘与存取。

而B+树结点页面的序列化在generic_key.h中实现。

2.3.2 B+树的相关操作

2.3.2.1 B+树的建立

在所有的逻辑页中，页号为1的页面为INDEX_ROOT_PAGE，该页中记录着所有索引的根节点信息，以"index_id root_page_id"的形式进行存储，即一个index_id对应一个根节点的root_page_id。

因此，当我们需要新建一个B+树时，首先需要先向buffer pool manager申请一个新页，然后用该新页存储我们的叶节点数据，同时我们需要将该B+树的叶节点对应页面的page_id存储到INDEX_ROOT_PAGE中，并与index_id一一对应。在需要调用索引时，我们只需根据相应的index_id取出对应的叶节点。

2.3.2.2 B+树元素插入

在B+树进行插入时，我们首先需要判断当前树是否为空，若为空，则我们需要建立一个新树，若不为空，则插入叶节点中。

```
if(tree is empty){
   创建新树;
}else{
   插入叶节点;
}
```

而在向叶节点中进行插入时，我们首先需要根据插入的key获得相应的叶节点，然后再向其中插入。此时会出现两种情况，一种是插入之后叶节点未超过容量，此时我们成功完成了插入；若叶节点中的元素个数超过了容量，则要将该叶节点分裂成两个，并获取到第二个叶节点的首元素，递归插入到父亲结点中。

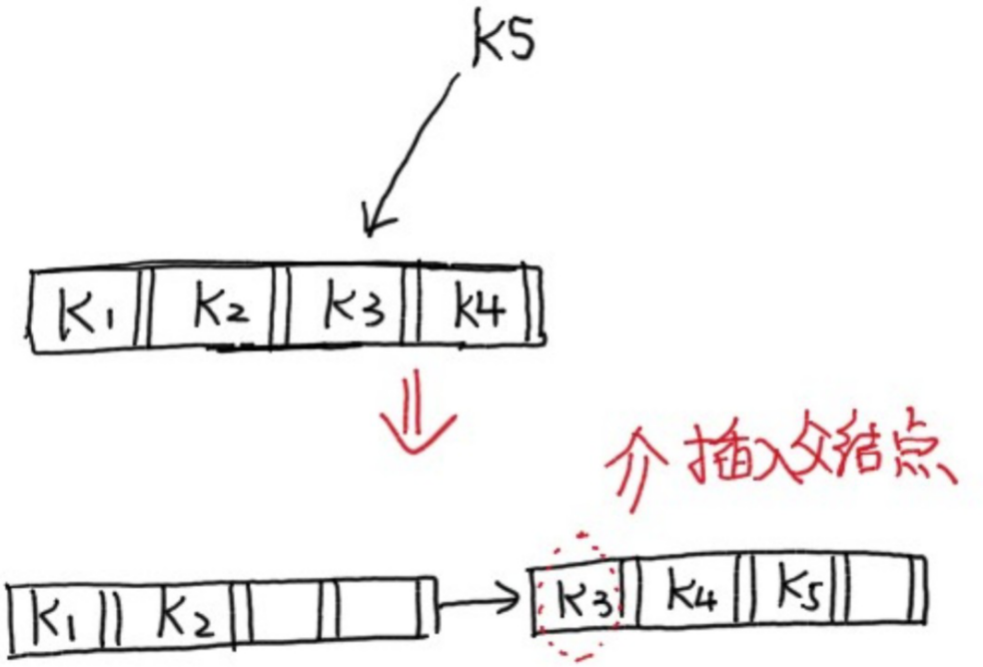
```
根据key值找到要插入的叶节点leaf
if(未找到满足条件的叶节点){    //待插入的key值与已有的值重复
   return false;
}else{
   if(leaf的大小+1为超过容限){
      直接在叶节点中进行插入;
   }else{
```

```
        分裂leaf,并得到一个新的页结点new_leaf;
        if(待插入的key小于new_leaf的首元素)
            将key插入leaf中;
        else
            将key插入new_leaf中;

        更新new_leaf的parent_page_id和、next_page_id;
        更新leaf的next_page_id;

        获得new_leaf的首元素middle_key;
        InsertIntoParent(middle_key); //递归调整父节点
    }
    return true;
}
```

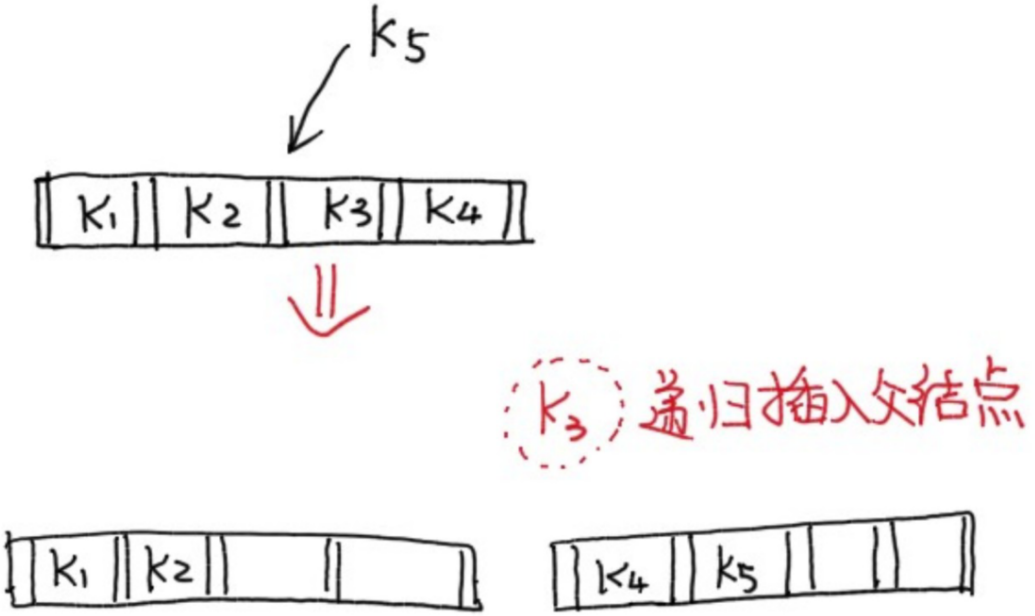
在向父节点中插入key值时，我们也需要考虑分裂的问题，但是由于父节点是中间结点，因此中间结点的分裂与叶节点的分裂稍有不同，因为对于中间结点而言，在分裂时需要将分裂而得的new_node的首元素、即middle_key从new_node中删除，然后再将middle_key递归插入自己的父节点之中。而对于叶节点，在分裂时middle_key并不需要从新产生的new_leaf中删除。

```
if(current_node的大小+1 < 容量){
    直接插入
}{
    将current_node中的键值对左移一位; //因为key[0]不存储真实值
    将代入的key插入current_node中;
    将current_node进行分裂，获得new_node;
    current_node中的键值对整体右移一位;
    new_node中的键值对整体左移一位;

    更新new_node的parent_page_id和next_page_id;
    更新current_node的next_page_id;
    更新new_node的各个子页的parent_page_id;

    获得分裂后的middle_key; //即new_node的key[0]

    InsertIntoParent(middle_key); //继续进行递归调用调整
}
```

叶节点的分裂图示如下:



中间结点的分裂图示如下:



2.3.2.3 B+树元素删除

对于B+树的删除，与插入一样，首先我们要获得待删除元素所在的叶节点，然后从叶节点中删除该元素。

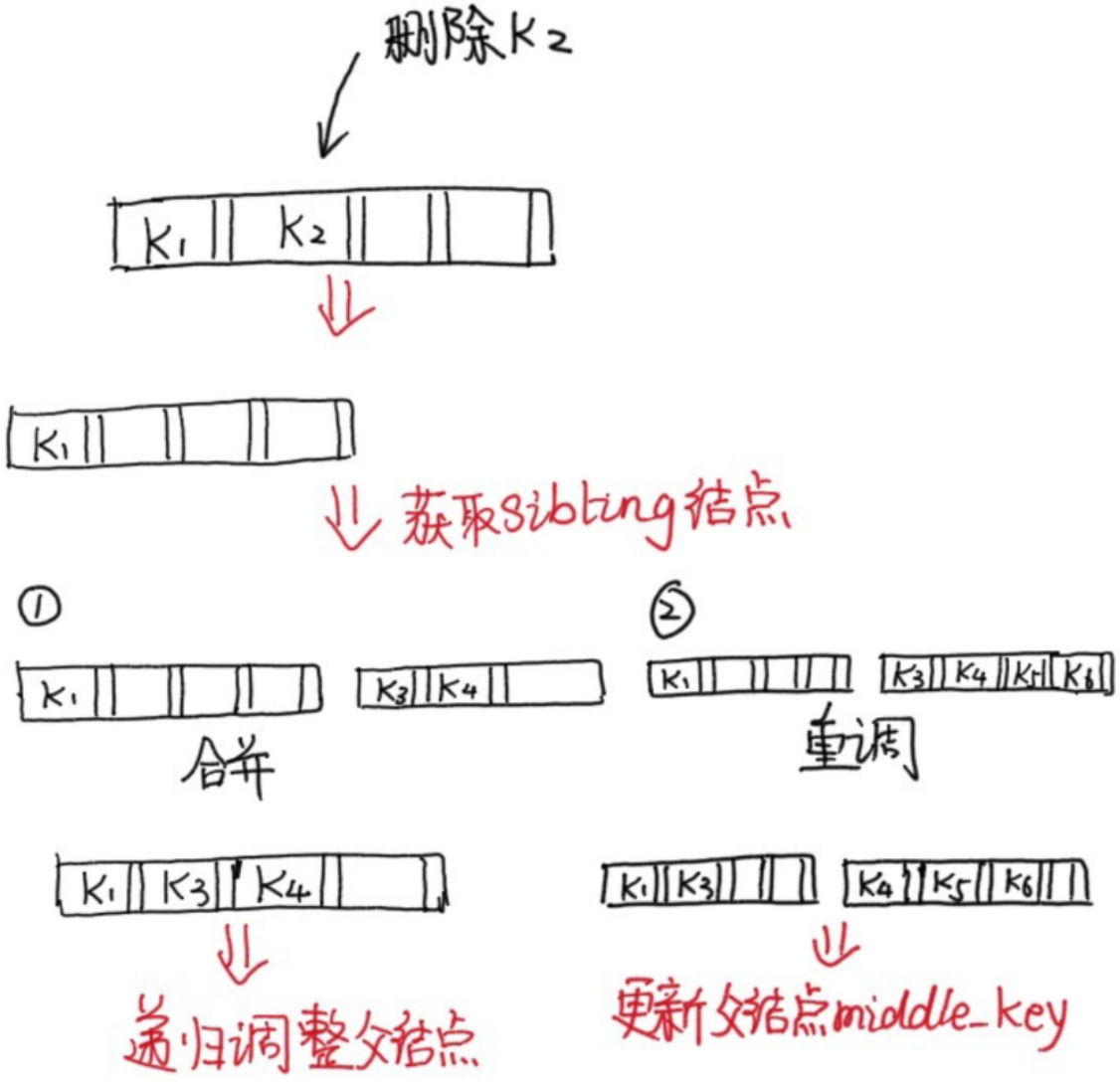
删除该元素之后，我们需要判断是否需要对结点进行调整。如果删除一个元素之后，该叶节点中的元素个数小于所规定的下限，我们则需要获得它的兄弟叶节点sibling,如果sibling的元素个数和leaf的元素个数之和小于叶节点中元素个数的最大限制，则对sibling和leaf结点进行合并;如果sibling结点和leaf结点的元素个数之和大于最大限制，则从sibling中取出一个元素插入到leaf中。

此时，如果我们仅仅进行了重调、即从sibling中取出了一个结点加入leaf中，则我们只需要对父节点的middle_key进行调整;如果我们将sibling结点与leaf结点进行了合并,则意味着父节点失去了一个子节点，则我们需要对父节点中的元素进行删除，而父节点的元素进行删除后也需要进行递归地判断是否需要合并与重调。

```
根据待删除的key获取相应的叶节点leaf;
```

```
if(leaf不存在){
    return false;
}else{
    从leaf中删除key;
    if(leaf的大小 > 最小限制)
        return true;
    else if(leaf的大小 < 最小限制){
        if(leaf在父节点中的位置为0)
            sibling = leaf的下一个兄弟节点;
        else
            sibling = leaf的上一个兄弟节点;

        if(sibling的大小 + leaf的大小 > 最大限制){
            从sibling中取出一个元素插入leaf中;
            调整父节点的middle_key;
        }else{
            合并sibling和leaf;
            从父节点中删除存储索引sibling的键值对;
            (通过调用Remove，从而实现递归调整)
        }
    }
}
```

删除操作的图示:



2.3.3 B+树索引迭代器

B+树还为上层提供索引迭代器,包括三种迭代器:begin(),begin(key)和end()。

其中第一种begin()是获取最左边的叶节点。

第二种begin(key)是获取key值所在的叶节点。

第三种end()是获得最右边的叶节点,即最后一个叶节点。

2.4 Catalog manager

2.4.1 对表、索引、目录源信息的序列化和反序列化

数据库中定义的表、索引和目录在内存中以`TableInfo`、`IndexInfo`和`CatelogInfo`的形式表现，分别维护了`TableMetadata`、`IndexMetadata`和`CatalogMeta`，各个源信息分别实现了序列化和反序列化，从而将表、索引和目录的所有定义信息持久化到数据库文件并在重启时从数据库文件中恢复。

2.4.2 对表的维护和管理

2.4.2.1 建立表

根据传入的表名、模式创建一个表，若创建成功，返回信息创建成功；若该表已经存在，则返回信息该表已经存在

2.4.2.2 获取表

1. 传入参数表名，获得该表。若获得成功，则返回信息获得成功；若获得失败，则返回信息该表不存在。
2. 传入参数表的序号，获得该表。若获得成功，则返回信息获得成功；若获得失败，则返回信息该表不存在。
3. 获得目录下的所有表。若获得成功，则返回信息获得成功；若获得失败，则返回信息获得失败。

2.4.2.3 删除表

根据传入的表名，删除该表。若删除失败，则返回信息该表不存在；若删除成功，则返回信息删除成功。

2.4.3 对索引的维护和管理

2.4.3.1 建立索引

根据传入的表名、索引名、键值，创建一个该表上的索引。若创建成功，则返回信息创建成功；若该表已经存在，则返回信息该表已经存在

2.4.3.2 获得索引

传入表名、索引名，获得对应的索引。若获得成功，则返回信息获得成功；若获得失败，则返回信息获得失败。

2.4.3.3 删除索引

传入表名、索引名，删除对应的索引。若删除成功，则返回信息删除成功；若删除失败，则返回信息获删除失败。

2.5 Executor

2.5.1 对数据库的维护和管理

1. 创建数据库
   根据语法树解析结果，创建数据库。若创建成功，返回信息创建成功；若创建失败，返回信息创建失败。
2. 删除数据库
   根据语法树解析结果，删除数据库。若删除成功，返回信息删除成功；若删除失败，返回信息删除失败。
3. 查看数据库
   根据语法树解析结果，查看数据库，打印所有数据库的名称。若查看成功，返回信息查看成功；若查看失败，返回信息查看失败。
4. 使用数据库

   根据语法树解析结果，使用该数据库。若使用成功，返回信息使用成功；若使用失败，返回信息该数据库不存在。

2.5.2 对表的维护和管理

1. 查看表
   根据语法树解析结果，查看当前数据库中所有表，打印所有表名。若查看成功，返回信息查看成功；若查看失败，返回信息查看失败。
2. 创建表
   根据语法树解析结果，创建数据库。若创建成功，返回信息创建成功；若创建失败，返回信息创建失败。
3. 删除表
   根据语法树解析结果，删除表。若删除成功，返回信息删除成功；若删除失败，返回信息删除失败。

2.5.3 对索引的维护和管理

1. 查看索引
   根据语法树解析结果，查看所有表上的所有索引，打印所有索引名。若查看成功，返回信息查看成功；若查看失败，返回信息查看失败。
2. 创建索引
   根据语法树解析结果，在表上创建索引。若创建成功，返回信息创建成功；若创建失败，返回信息创建失败。
3. 删除索引
   根据语法树解析结果，删除表上的索引。若删除成功，返回信息删除成功；若删除失败，返回信息删除失败。

2.5.4 查找操作

根据语法树解析结果进行查找。查找分为四种情况，分别是无投影且无条件、有投影且无条件、无投影且有条件、有投影且有条件。当进行有条件查询时，判断是否可以通过索引查找。若查找成功，则返回信息查找成功，打印查找信息。若查找失败，则返回信息查找失败。

2.5.5 插入操作

根据语法树解析结果插入数据，并更新索引。若插入成功，则返回信息插入成功。若插入失败，则返回信息插入失败。

2.5.6 删除操作

根据语法树解析结果删除数据，并更新索引。若删除成功，则返回信息删除成功。若删除失败，则返回信息删除失败。

2.5.7 更新操作

根据语法树解析结果更新数据，并判断是否需要更新索引。若更新的field为索引键值，则需要更新索引。若更新成功，则返回信息更新成功。若更新失败，则返回信息更新失败。

2.5.8 终止操作

退出程序

第三章

3.1 Disk and buffer manager

3.1.1 位图页

- `template<size_t PageSize> bool BitmapPage<PageSize>::AllocatePage(uint32_t &page_offset)`：分配一个空闲页，并通过`page_offset`返回所分配的空闲页位于该段中的下标（从`0`开始）
- `template<size_t PageSize> bool BitmapPage<PageSize>::DeAllocatePage(uint32_t page_offset)`：回收已经被分配的页`page_offset`
- `template<size_t PageSize> bool BitmapPage<PageSize>::IsPageFree(uint32_t page_offset) const`：判断给定的页`page_offset`是否是空闲（未分配）的

3.1.2 磁盘数据页管理

- `page_id_t DiskManager::AllocatePage()`：从磁盘中分配一个空闲页，并返回空闲页的**逻辑页号**
- `void DiskManager::DeAllocatePage(page_id_t logical_page_id)`：释放磁盘中**逻辑页号**对应的物理页
- `bool DiskManager::IsPageFree(page_id_t logical_page_id)`：判断该**逻辑页号**对应的数据页是否空闲
- `page_id_t DiskManager::MapPageId(page_id_t logical_page_id)`：可根据需要实现。在`DiskManager`类的私有成员中，该函数可以用于将逻辑页号转换成物理页号

3.1.3 基于LRU替换策略的替换器

- `bool LRUReplacer::Victim(frame_id_t *frame_id)`：替换（即删除）与所有被跟踪的页相比最近最少被访问的页，将其页帧号（即数据页在Buffer Pool的Page数组中的下标）存储在输出参数`frame_id`中输出并返回`true`，如果当前没有可以替换的元素则返回`false`
- `void LRUReplacer::Pin(frame_id_t frame_id)`：将数据页固定使之不能被`Replacer`替换，即从`lru_list_`中移除该数据页对应的页帧。`Pin`函数应当在一个数据页被Buffer Pool Manager固定时被调用
- `void LRUReplacer::Unpin(frame_id_t frame_id)`：将数据页解除固定，放入`lru_list_`中，使之可以在必要时被`Replacer`替换掉。`Unpin`函数应当在一个数据页的引用计数变为`0`时被Buffer Pool Manager调用，使页帧对应的数据页能够在必要时被替换
- `size_t LRUReplacer::Size()`：此方法返回当前`LRUReplacer`中能够被替换的数据页的数量

3.1.4 缓冲池管理

- `Page *BufferPoolManager::FetchPage(page_id_t page_id)`：根据逻辑页号获取对应的数据页，如果该数据页不在内存中，则需要从磁盘中进行读取；
- `Page *BufferPoolManager::NewPage(page_id_t &page_id)`：分配一个新的数据页，并将逻辑页号于`page_id`中返回；
- `bool BufferPoolManager::UnpinPage(page_id_t page_id, bool is_dirty)`：取消固定一个数据页；
- `BufferPoolManager::FlushPage(page_id)`：将数据页转储到磁盘中；
- `bool BufferPoolManager::DeletePage(page_id_t page_id)`：释放一个数据页；
- `bool BufferPoolManager::FlushPage(page_id_t page_id)`：将所有的页面都转储到磁盘中。

对于`FetchPage`操作，如果空闲页列表（`free_list_`）中没有可用的页面并且没有可以被替换的数据页，则应返回`nullptr`。`FlushPage`操作应该将页面内容转储到磁盘中，无论其是否被固定。

3.2 Record manager

3.2.1 数据的序列化和反序列化

在本节中你需要完成如下函数：

- `uint32_t Row::SerializeTo(char *buf, Schema *schema) const`：将该`Row`序列化到buf中，返回`buf`指针向前推进了的字节数（字段类型无需序列化，反序列化时从传入的schema获取字段类型）
- `uint32_t Row::DeserializeFrom(char *buf, Schema *schema)`从buf中反序列化，返回`buf`指针向前推进了的字节数
- `uint32_t Row::GetSerializedSize(Schema *schema) const`：返回序列化该`Row`需要的字节数
- `uint32_t Column::SerializeTo(char *buf) const`：将该`Column`序列化到buf中，返回`buf`指针向前推进了的字节数
- `uint32_t Column::DeserializeFrom(char *buf, Column *&column, MemHeap *heap)`：从buf中反序列化，通过`column`传回，返回`buf`指针向前推进了的字节数
- `uint32_t Column::GetSerializedSize() const`：返回序列化该`Column`需要的字节数
- `uint32_t Schema::SerializeTo(char *buf) const`：将该`Schema`序列化到buf中，返回`buf`指针向前推进了的字节数
- `uint32_t Schema::DeserializeFrom(char *buf, Schema *&schema, MemHeap *heap)`：从buf中反序列化，通过`schema`传回，返回`buf`指针向前推进了的字节数
- `uint32_t Schema::GetSerializedSize() const`：返回序列化该`Schema`需要的字节数

对于`Row`类型对象的序列化，可以通过位图的方式标记为`null`的`Field`(即 *Null Bitmaps*)，对于`Row`类型对象的反序列化，在反序列化每一个`Field`时，需要将自身的`heap_`作为参数传入到`Field`类型的`Deserialize`函数中，这也意味着所有反序列化出来的`Field`的内存都由该`Row`对象维护。对于`Column`和`Schema`类型对象的反序列化，将使用`MemHeap`类型的对象`heap`来分配空间，分配后新生成的对象于参数`column`和`schema`中返回。

3.2.2 堆表

- `bool TableHeap::InsertTuple(Row &row, Transaction *txn)`: 向堆表中插入一条记录，插入记录后生成的`RowId`需要通过`row`对象返回（即`row.rid_`）
- `bool TableHeap::UpdateTuple(Row &row, const RowId &rid, Transaction *txn)`：将`RowId`为`rid`的记录`old_row`替换成新的记录`new_row`，并将`new_row`的`RowId`通过`new_row.rid_`返回
- `void TableHeap::ApplyDelete(const RowId &rid, Transaction *txn)`：从物理意义上删除这条记录
- `bool TableHeap::GetTuple(Row *row, Transaction *txn)`：获取`RowId`为`row->rid_`的记录
- `void TableHeap::FreeHeap()`：销毁整个`TableHeap`并释放这些数据页
- `TableIterator TableHeap::Begin(Transaction *txn)`：获取堆表的首迭代器
- `TableIterator TableHeap::End()`：获取堆表的尾迭代器
- `TableIterator &TableIterator::operator++()`：移动到下一条记录，通过`++iter`调用
- `TableIterator TableIterator::operator++(int)`：移动到下一条记录，通过`iter++`调用

3.3 Index manager

3.3.1 新建

```
INDEX_TEMPLATE_ARGUMENTS
void BPLUSTREE_TYPE::StartNewTree(const KeyType &key, const ValueType &value)
```

新建一个根节点，并将根节点通过UpdateRootPageId(),进行记录，然后向根节点中插入对于的(key,value)对。

```
INDEX_TEMPLATE_ARGUMENTS
void BPLUSTREE_TYPE::UpdateRootPageId(int insert_record)
```

当insert_record非0时，表示新建了一棵B+树，此时需要获取INDEX_ROOT_PAGE_ID所对应的页面，然后向其中插入index_id和root_page_id所构成的键值对。

当insert_record为0时，表示更改了当前索引的根节点页，则需要对INDEX_ROOT_PAGE中相应索引对应的root_page_id进行调整。

3.3.2 插入

```
INDEX_TEMPLATE_ARGUMENTS
bool BPLUSTREE_TYPE::Insert(const KeyType &key, const ValueType &value,
Transaction *transaction)
```

插入的详细实现在第二章中已有阐述，此处给出它所使用的几个接口，并对尚未阐述的接口进行详细说明。

```
INDEX_TEMPLATE_ARGUMENTS
bool BPLUSTREE_TYPE::InsertIntoLeaf(const KeyType &key, const ValueType &value,
Transaction *transaction)
```

```
INDEX_TEMPLATE_ARGUMENTS
template <typename N>
N *BPLUSTREE_TYPE::Split(N *node)
```

对于Split分裂的实现，首先我们需要通过buffer pool manager获取一个新页，然后将旧页中的一般数据转移到新页中，即可实现对于旧节点的分裂。

```
INDEX_TEMPLATE_ARGUMENTS
void BPLUSTREE_TYPE::InsertIntoParent(BPlusTreePage *old_node, const KeyType
&key, BPlusTreePage *new_node,Transaction *transaction)
```

```
INDEX_TEMPLATE_ARGUMENTS
Page *BPLUSTREE_TYPE::FindLeafPage(const KeyType &key, bool leftMost)
```

对于FindLeafPage的实现,首先我们需要根据key一路进行索引，从根节点向叶节点索引，逐次Fetch这些节点所在的页，并且在查询完之后Unpin该页，直到我们找出key所在的页节点并返回。

3.3.3 删除

对于删除的具体实现，在第二章中也已进行了详细的阐释，因此在这里只给出相关的接口，对于未进行阐释的接口再给出实现说明。

```
INDEX_TEMPLATE_ARGUMENTS
void BPLUSTREE_TYPE::Remove(const KeyType &key, Transaction *transaction)
```

```
INDEX_TEMPLATE_ARGUMENTS
template <typename N>
bool BPLUSTREE_TYPE::CoalesceOrRedistribute(N *node, Transaction *transaction)
```

本接口实现对节点是否需要合并或者重调的判断，即通过第二章中所阐释的方式进行判断，然后分别调用以下两个分别用于重调和合并的接口:

```
INDEX_TEMPLATE_ARGUMENTS
template <typename N>
bool BPLUSTREE_TYPE::Coalesce(N *neighbor_node, N
*node,BPlusTreeInternalPage<KeyType, page_id_t, KeyComparator> *parent, int
index,Transaction *transaction)
```

该接口实现对sibling节点和leaf节点的合并，详细实现策略见第二章。

```
INDEX_TEMPLATE_ARGUMENTS
template <typename N>
void BPLUSTREE_TYPE::Redistribute(N *neighbor_node, N *node, int index)
```

该接口实现对sibling节点和leaf节点的重调，详细实现策略见第二章。

```
INDEX_TEMPLATE_ARGUMENTS
bool BPLUSTREE_TYPE::AdjustRoot(BPlusTreePage *old_root_node)
```

由于在递归重调时，可能会涉及到对根节点的调整，因此本接口的功能是当对根节点进行重调时，可以根据根节点当前情况的不同，从而做出相应不同的判断。

若我们需要删除根节点中最后一个key值，则需要把它的最后一个孩子节点设为根节点。

若我们需要删除整个B+树中的最后一个key值，则我们需要从INDEX_ROOT_PAGE中删除相应的记录。

3.3.4迭代器的实现

```
INDEX_TEMPLATE_ARGUMENTS
INDEXITERATOR_TYPE BPLUSTREE_TYPE::Begin()
```

调用`FindLeafPage()`获取最左边的叶节点，从而实现对begin()迭代器的构造。

```
INDEX_TEMPLATE_ARGUMENTS
INDEXITERATOR_TYPE BPLUSTREE_TYPE::Begin(const KeyType &key)
```

调用`FindLeafPage()`获取key所在的叶节点，从而实现对begin(key)迭代器的构造。

```
INDEX_TEMPLATE_ARGUMENTS
INDEXITERATOR_TYPE BPLUSTREE_TYPE::End()
```

调用`FindLeafPage()`获得最右边的一个迭代器，从而实现对end()迭代器的构造。

3.3.5内存回收的实现

```
INDEX_TEMPLATE_ARGUMENTS
void BPLUSTREE_TYPE::Destroy()
```

遍历B+树叶节点中存储的每一个键值对，将其从B+树中删除，删除的过程中即实现了对相应页面的回收(即DeletePage)。

3.4 Catalog manager

3.4.1 table

`uint32_t TableMetadata::SerializeTo(char *buf) const`

TableMetaData的序列化，将MAGIC_NUM、table_id_t、size_t、page_id_t、schema依次写入字节流buf，每次读出后buf增加相应的推进字节数，写入table_name前先写入其大小size_t，返回buf指针推进的字节数ofs

`uint32_t TableMetadata::DeserializeFrom(char *buf, TableMetadata *&table_meta, MemHeap *heap)`

TableMetaData的反序列化，将MAGIC_NUM、table_id_t、table_name、page_id_t、schema依次读出字节流buf，读出table_name前读出其大小size_t，并通过读出的各个参数构造TableMetaData。返回buf指针推进的字节数ofs

`uint32_t TableMetadata::GetSerializedSize() const`

获得TableMetaData的序列化长度，返回值等于序列化中的返回值。

3.4.2 index

`uint32_t IndexMetadata::SerializeTo(char *buf) const`

IndexMetaData的序列化，将MAGIC_NUM、index_id_t、index_name、table_id_t、key_map依次写入字节流buf，每次读出后buf增加相应的推进字节数，写入index_name、key_map前先写入其大小size_t，返回buf指针推进的字节数ofs

`uint32_t IndexMetadata::DeserializeFrom(char *buf, IndexMetadata *&index_meta, MemHeap *heap)`

IndexMetaData的反序列化，将MAGIC_NUM、index_id_t、index_name、table_id_t、key_map依次读出字节流buf，读出index_name、key_map前读出其大小size_t，并通过读出的各个参数构造TableMetaData。返回buf指针推进的字节数ofs

`uint32_t IndexMetadata::GetSerializedSize() const`

获得IndexMetaData的序列化长度，返回值等于序列化中的返回值。

3.4.3 catalog

`void CatalogMeta::SerializeTo(char *buf) const`

CatalogMeta的序列化，将MAGIC_NUM、table_meta_pages、index_meta_pages依次写入字节流buf，每次读入后buf增加相应的推进字节数，写入table_meta_pages、index_meta_pages前先写入其大小size_t

`CatalogMeta *CatalogMeta::DeserializeFrom(char *buf, MemHeap *heap)`

CatalogMeta的反序列化，将MAGIC_NUM、table_meta_pages、index_meta_pages依次读出字节流buf，每次读入后buf增加相应的推进字节数，读出table_meta_pages、index_meta_pages前先读出其大小size_t，根据读出的参数构造catalogmeta。

`uint32_t CatalogMeta::GetSerializedSize() const`

获得CatalogMeta的序列化长度，返回值等于序列化中的buf推进buf长度。

`CatalogManager::CatalogManager(BufferPoolManager *buffer_pool_manager, LockManager *lock_manager, LogManager *log_manager, bool init)`

CatalogManager构造函数，将catalogmeta进行反序列化，从page里取出来。遍历所有表的数据页，反序列化得到tablemetadata，创建tableheap，并初始化tableinfo，并压入tables_ 和 table_names_ 。遍历所有索引的数据页，反序列化得到indexmetadata，初始化indexinfo，并压入index_names_ 和 indexes_ 。

`CatalogManager::~CatalogManager()`

将catalog_meta序列化罗盘，并删除堆

`dberr_t CatalogManager::CreateTable(const string &table_name, TableSchema *schema, Transaction *txn,TableInfo *&table_info)`

调用GetNextTableId()得到新表id，push进table_names。调用内存池分配新页，调用TableInfo::Create给table_info分配内存，构造tablemetadata和tableheap，并初始化table_info，压入tables_。将tablemetadata和catalog_meta序列化落盘。

`dberr_t CatalogManager::GetTable(const string &table_name, TableInfo &table_info)`

遍历tables_names_ ，找到对应的table_id，返回*table_info* = tables_[table_id];

`dberr_t CatalogManager::GetTable(const table_id_t table_id, TableInfo &table_info)`

返回*table_info* = tables_[table_id];

`dberr_t CatalogManager::GetTables(vector<TableInfo > &tables) const`

遍历tables_，压入tables。

`dberr_t CatalogManager::DropTable(const string &table_name)`

在table_names_ ，talbles_ ，table_meta_pages中删除该表，释放table_metadata的数据页，将更新后的catalog_meta序列化落盘。

`dberr_t CatalogManager::CreateIndex(const std::string &table_name, const string &index_name, const std::vector<std::string> &index_keys, Transaction txn, IndexInfo &index_info)`

遍历index_names_ ，如果没有建立在table_name上的索引，则调用GetNextIndexId()，建立new_index，将new_index压入index_names_ 。如果存在建立在table_name上的索引，则将new_index压入index_names。调用内存池分配新页，调用IndexInfo::Create给index_info分配内存，构造indexmetadata，并初始化index_info，压入indexes_。将tablemetadata和catalog_meta序列化落盘。

`dberr_t CatalogManager::GetIndex(const std::string &table_name, const std::string &index_name, IndexInfo &index_info) const`

遍历index_names，查看是否有建立在该表上的索引。再遍历indexes，查看该索引是否存在。index_info = indexes_.find(new_index_id)->second;

`dberr_t CatalogManager::DropIndex(const string &table_name, const string &index_name)`

遍历index_names，查看是否有建立在该表上的索引。再遍历indexes，查看该索引是否存在。在index_names，indexes中删除该索引。释放数据页，将catalogmeta序列化落盘。

`dberr_t CatalogManager::GetTableIndexes(const std::string &table_name, std::vector<IndexInfo > &indexes) const`

遍历index_names， 找到对应表上的索引集，压入传入的indexes。

3.5 Executor

`dberr_t ExecuteCreateDatabase(pSyntaxNode ast, ExecuteContext *context);`

若dbs_ 中存在传入的db_name，返回创建失败。否则新建DBStorageEngine，并插入dbs_

`dberr_t ExecuteDropDatabase(pSyntaxNode ast, ExecuteContext *context);`

若dbs_ 中不存在传入的db_name，返回删除失败。否则在dbs_ 删除db_name，如果被删除是当前使用的数据库，则 current_db_.clear()

`dberr_t ExecuteShowDatabases(pSyntaxNode ast, ExecuteContext *context);`

判断dbs_ 是否为空。遍历dbs_，打印数据库名字

`dberr_t ExecuteUseDatabase(pSyntaxNode ast, ExecuteContext *context);`

判断db_name是否存在。将传入的db_name设为current_name

`dberr_t ExecuteShowTables(pSyntaxNode ast, ExecuteContext *context);`

在current_db中得到catalog，调用GetTables（）获得tables。遍历tables，打印表名。

`dberr_t ExecuteCreateTable(pSyntaxNode ast, ExecuteContext *context);`

在current_db中得到catalog，遍历语法树，得到column_name, type_, length, table_position, 得到nullable, unique的状态，构造column，再调用TableSchema构造schema，最后调用 CreateTable创建新表。

`dberr_t ExecuteDropTable(pSyntaxNode ast, ExecuteContext *context);`

在current_db中得到catalog，调用DropTable，删除表。

`dberr_t ExecuteShowIndexes(pSyntaxNode ast, ExecuteContext *context);`

在current_db中得到catalog，遍历tables，调用GetTableIndexes获得每张表上的表名并打印。

`dberr_t ExecuteCreateIndex(pSyntaxNode ast, ExecuteContext *context);`

在current_db中得到catalog，遍历语法树，得到table_name, index_name, index_keys，再调用CreateIndex构造indexinfo。遍历堆表，构造key_map上的entry，调用InsertEntry插入数据库条目。

`dberr_t ExecuteDropIndex(pSyntaxNode ast, ExecuteContext *context);`

在current_db中得到catalog，调用GetTables获得tables，遍历tables，找到索引所在的表，调用DropIndex删除索引。

`bool DFS(pSyntaxNode ast, TableIterator &iter, Schema *schema)`

为了实现sql语句中的条件判断，在此实现了判断函数DFS。传入首个类型为kNodeCompareOperator或kNodeConnector的语法树节点ast，table迭代器iter，表的schema。递归方式如下，若connector为and，则返回子节点和子节点的右节点的返回结果的交集。若connector为or，则返回子节点和子节点的右节点的返回结果的并集。

```
if (strcmp(connector, "and") && DFS(ast->child_, iter, schema) && DFS(ast-
>child_->next_, iter, schema))
      return true;
    else if (strcmp(connector, "or") && (DFS(ast->child_, iter, schema) ||
DFS(ast->child_->next_, iter, schema)))
      return true;
```

递归出口为ast类型为kNodeCompareOperator，根据操作符不同进行不同的判断

```
if (strcmp(item, "=") == 0 && strcmp(l_value, r_value) == 0)
      return true;
    else if (strcmp(item, ">") == 0 && strcmp(l_value, r_value) > 0)
      return true;
    else if (strcmp(item, ">=") == 0 && strcmp(l_value, r_value) >= 0)
      return true;
    else if (strcmp(item, "<=") == 0 && strcmp(l_value, r_value) <= 0)
      return true;
    else if (strcmp(item, "<") == 0 && strcmp(l_value, r_value) < 0)
      return true;
    else if (strcmp(item, "<>") == 0 && strcmp(l_value, r_value) != 0)
      return true;
```

`dberr_t ExecuteSelect(pSyntaxNode ast, ExecuteContext *context);`

解析语法树，select分为四种情况，分别是无投影且无条件、有投影且无条件、无投影且有条件、有投影且有条件。

无投影无条件时，调用堆表迭代器，打印每条column的field。

有投影无条件时，记录投影的column_name，调用堆表迭代器，打印投影column_name对应的field

无投影有条件时，根据条件判断是否可以通过索引加速查找。调用堆表迭代器，调用DFS判断row是否符合条件，打印每条column的field。

有投影有条件时，记录投影的column_name，根据条件判断是否可以通过索引加速查找。调用堆表迭代器，调用DFS判断row是否符合条件，打印投影column_name对应的field

`dberr_t ExecuteInsert(pSyntaxNode ast, ExecuteContext *context);`

解析语法树，读入每一个field，构造row，调用InsertTuple插入tuple。调用InsertEntry插入索引条目。

`dberr_t ExecuteDelete(pSyntaxNode ast, ExecuteContext *context);`

解析语法树，读入每一个field，如果有条件约束，调用DFS判断是否满足条件。读入每一个field，构造row，调用UpdateTuple更新tuple。调用InsertEntry、RemoveEntry来更新索引条目。

`dberr_t ExecuteUpdate(pSyntaxNode ast, ExecuteContext *context);`

解析语法树，读入每一个field，如果有条件约束，调用DFS判断是否满足条件。读入每一个field，构造row，调用MarkDelete删除tuple。调用RemoveEntry删除索引条目。

`dberr_t ExecuteExecfile(pSyntaxNode ast, ExecuteContext *context);`

按行从文件中读入sql语句，借用main中的程序运行sql语句。

`dberr_t ExecuteQuit(pSyntaxNode ast, ExecuteContext *context);`

设置flag_quit，*context*->flag_quit_ = true;

第四章

4.1 各模块测试样例说明及结果

4.1.1 Disk and buffer manager

4.1.1.1 buffer_pool_manager_test

```
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from BufferPoolManagerTest
[ RUN      ] BufferPoolManagerTest.BinaryDataTest
[       OK ] BufferPoolManagerTest.BinaryDataTest (25 ms)
[----------] 1 test from BufferPoolManagerTest (25 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (25 ms total)
[  PASSED  ] 1 test.
```

4.1.1.2 lru_replacer_test

```
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from LRUReplacerTest
[ RUN      ] LRUReplacerTest.SampleTest
[       OK ] LRUReplacerTest.SampleTest (0 ms)
[ RUN      ] LRUReplacerTest.SampleTest2
[       OK ] LRUReplacerTest.SampleTest2 (27 ms)
[----------] 2 tests from LRUReplacerTest (27 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (27 ms total)
[  PASSED  ] 2 tests.
```

SampleTest小数据，SampleTest2大数据

4.1.1.3 disk_manager_test

```
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from DiskManagerTest
[ RUN      ] DiskManagerTest.BitMapPageTest
[       OK ] DiskManagerTest.BitMapPageTest (4 ms)
[ RUN      ] DiskManagerTest.FreePageAllocationTest
[       OK ] DiskManagerTest.FreePageAllocationTest (1888 ms)
[----------] 2 tests from DiskManagerTest (1893 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (1893 ms total)
[  PASSED  ] 2 tests.
```

4.1.2 Record manager

4.1.2.1 tuple_test

```
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from TupleTest
[ RUN      ] TupleTest.FieldSerializeDeserializeTest
[       OK ] TupleTest.FieldSerializeDeserializeTest (3 ms)
[ RUN      ] TupleTest.RowTest
[       OK ] TupleTest.RowTest (0 ms)
[ RUN      ] TupleTest.SchemaTest
[       OK ] TupleTest.SchemaTest (0 ms)
[----------] 3 tests from TupleTest (3 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (3 ms total)
[  PASSED  ] 3 tests.
```

4.1.2.2 table_heap_test

```
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from TableHeapTest
[ RUN      ] TableHeapTest.TableHeapSampleTest
[       OK ] TableHeapTest.TableHeapSampleTest (35 ms)
[----------] 1 test from TableHeapTest (35 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (35 ms total)
[  PASSED  ] 1 test.
```

4.1.3 Index Manager

(1) B+树索引键序列化和反序列化的测试

```
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from BPlusTreeTests
[ RUN      ] BPlusTreeTests.BPlusTreeIndexGenericKeyTest
[       OK ] BPlusTreeTests.BPlusTreeIndexGenericKeyTest (6 ms)
[ RUN      ] BPlusTreeTests.BPlusTreeIndexSimpleTest
[       OK ] BPlusTreeTests.BPlusTreeIndexSimpleTest (5 ms)
[----------] 2 tests from BPlusTreeTests (12 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (12 ms total)
[  PASSED  ] 2 tests.
```

测试目的:为检测B+树的索引键是否可以实现正常的序列化与反序列化。

测试方法:创建一个table,并向其中插入一些tuple，然后调用generic_key的序列化，再将其反序列化，比较前后tuple中各元素值是否相等，相等则说明序列化与反序列化成功。

(2) B+树插入、删除等基本操作的测试

```
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from BPlusTreeTests
[ RUN      ] BPlusTreeTests.SampleTest
[       OK ] BPlusTreeTests.SampleTest (7 ms)
[----------] 1 test from BPlusTreeTests (7 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (7 ms total)
[  PASSED  ] 1 test.
```

测试目的:检测B+树能否进行正常的建立、插入与删除。

测试方法:向B+树中插入n个随机排列的随机数，然后遍历B+树的元素检测插入是否成功。然后从B+树种删除n/2个元素，然后检测这n/2个元素是否被删除成功,剩下的n/2个元素是否还成功保留。每一步操作后都要调用`check()`检测所有页面是否都被Unpin。

(3) B+树迭代器功能的测试

```
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from BPlusTreeTests
[ RUN      ] BPlusTreeTests.IndexIteratorTest
[       OK ] BPlusTreeTests.IndexIteratorTest (9 ms)
[----------] 1 test from BPlusTreeTests (9 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (9 ms total)
[  PASSED  ] 1 test.
```

测试目的:测试获得的迭代器是否可以实现其功能。
测试方法:对B+树进行建立、插入、删除等操作，然后调用相应的迭代器验证是否可以通过相应的迭代器实现对叶节点的遍历。

4.1.4 catalog

```
[==========] Running 5 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 5 tests from CatalogTest
[ RUN      ] CatalogTest.CatalogMetaTest
[       OK ] CatalogTest.CatalogMetaTest (0 ms)
[ RUN      ] CatalogTest.IndexMetaTest
[       OK ] CatalogTest.IndexMetaTest (0 ms)
[ RUN      ] CatalogTest.TableMetaTest
[       OK ] CatalogTest.TableMetaTest (0 ms)
[ RUN      ] CatalogTest.CatalogTableTest
[       OK ] CatalogTest.CatalogTableTest (16 ms)
[ RUN      ] CatalogTest.CatalogIndexTest
[       OK ] CatalogTest.CatalogIndexTest (11 ms)
[----------] 5 tests from CatalogTest (28 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test suite ran. (28 ms total)
[  PASSED  ] 5 tests.
```

4.1.5 executor

database相关操作和测试结果如下：

创建三个数据库db0, db1, db2

展示数据库

删除数据库db2

展示数据库

使用数据库db0

```
minisql > create database db0;
[INFO] Sql syntax parse ok!
minisql > create database db1;
[INFO] Sql syntax parse ok!
minisql > create database db2;
[INFO] Sql syntax parse ok!
minisql > show databases;
[INFO] Sql syntax parse ok!
[DATABASE] db2
[DATABASE] db1
[DATABASE] db0
minisql > drop database db2;
[INFO] Sql syntax parse ok!
[INFO] Drop database successfully!
minisql > show databases;
[INFO] Sql syntax parse ok!
[DATABASE] db1
[DATABASE] db0
minisql > use db0;
[INFO] Sql syntax parse ok!
[INFO] Use successfully!
```

table相关操作和测试结果如下：

创建一系列表，格式不正确的表返回结果创建失败

```
minisql > create table t1(a int, b c
har(20) unique, c float, primary key
(a, c));
[INFO] Sql syntax parse ok!
[INFO] Create table successfully!
minisql > create table t2(a int, b c
har(0) unique, c float, primary key(
a, c));
[INFO] Sql syntax parse ok!
[INFO] Create table successfully!
minisql > create table t1(a int, b c
har(-5) unique, c float, primary key
(a, c));
[INFO] Sql syntax parse ok!
[ERROR] Create table failed!
minisql > create table t1(a int, b c
har(3.69) unique, c float, primary k
ey(a, c));
[INFO] Sql syntax parse ok!
[ERROR] Create table failed!
minisql > create table t1(a int, b c
har(-0.69) unique, c float, primary
key(a, c));
[INFO] Sql syntax parse ok!
[ERROR] Create table failed!
```

展示所有表

```
[TITLE] Tables_in_db0
[TABLE] t1
```

插入三条数据

```
insert into t1 values(1, "aaa",  2.33);
insert into t1 values(2, "bbb",  2.33);
insert into t1 values(3, "ccc",  2.33);minisql > [INFO] Sql syntax parse ok!
```

不同的select操作

无投影无条件

```
minisql > select * from t1;
[INFO] Sql syntax parse ok!
-------------------------------------------------------------
|a                  |b                  |c                  |
-------------------------------------------------------------
|1                  |aaa                |2.33000            |
-------------------------------------------------------------
|2                  |bbb                |2.33000            |
-------------------------------------------------------------
|3                  |ccc                |2.33000            |
-------------------------------------------------------------
```

有投影无条件

```
minisql > select a,b from t1;
[INFO] Sql syntax parse ok!
-----------------------------------------
|a                  |b                  |
-----------------------------------------
|1                  |aaa                |
-----------------------------------------
|2                  |bbb                |
-----------------------------------------
|3                  |ccc                |
-----------------------------------------
```

无投影有条件

```
minisql > select * from t1 where a=1;
[INFO] Sql syntax parse ok!
-------------------------------------------------------------
|a                  |b                  |c                  |
-------------------------------------------------------------
|1                  |aaa                |2.33000            |
-------------------------------------------------------------
```

有投影有条件

```
[INFO] Sql syntax parse ok!
-----------------------------------------
|a                  |b                  |
-----------------------------------------
|1                  |aaa                |
-----------------------------------------
```

无条件更新

```
minisql > update t1 set c = 3.0;
[INFO] Sql syntax parse ok!
```

查看结果

```
minisql > select * from t1;
[INFO] Sql syntax parse ok!
-------------------------------------------------------------
|a                  |b                  |c
-------------------------------------------------------------
|1                  |aaa                |3.00000            |
-------------------------------------------------------------
|2                  |bbb                |3.00000
-------------------------------------------------------------
|3                  |ccc                |3.00000
-------------------------------------------------------------
```

有条件更新

```
minisql > update t1 set c=2.33 where a=1;
[INFO] Sql syntax parse ok!
[INFO] Update successfully!
```

查看结果

```
minisql > select * from t1;
[INFO] Sql syntax parse ok!
-------------------------------------------------------------
|a                  |b                  |c                  |
-------------------------------------------------------------
|1                  |aaa                |2.33000            |
-------------------------------------------------------------
|2                  |bbb                |3.00000            |
-------------------------------------------------------------
|3                  |ccc                |3.00000            |
-------------------------------------------------------------
```

删除

```
minisql > delete from t1 where a = 3;
[INFO] Sql syntax parse ok!
```

查看结果

```
minisql > select * from t1;
[INFO] Sql syntax parse ok!
-------------------------------------------------------------
|a                  |b                  |c
-------------------------------------------------------------
|1                  |aaa                |2.33000
-------------------------------------------------------------
|2                  |bbb                |3.00000
-------------------------------------------------------------
```

创建并查看索引

```
minisql > create index idx01 on t1(a);
[INFO] Sql syntax parse ok!
[INFO] Create index successfully!
minisql > show indexes;
[INFO] Sql syntax parse ok!
[TITLE] Indexes Of Table t1
[INDEX] idx01
```

删除索引

```
minisql > drop index idx01;
[INFO] Sql syntax parse ok!
[INFO] Drop index successfully!
minisql > show indexes;
[INFO] Sql syntax parse ok!

[INFO] There aren't any indexes!
```

删除表的全部内容

```
minisql > delete from t1;
[INFO] Sql syntax parse ok!
minisql > select * from t1;
[INFO] Sql syntax parse ok!
-------------------------------------------------------------
|a                  |b                  |c
-------------------------------------------------------------
```

删除表

```
minisql > drop table t1;
[INFO] Sql syntax parse ok!
[INFO] Drop successfully!
minisql > show tables;
[INFO] Sql syntax parse ok!
[INFO] There aren't any tables!
```

4.2 验收样例测试结果

1.创建三个数据库:
create database db0;
create database db1;
create database db2;

```
minisql > create database db0;
[INFO] Sql syntax parse ok!
minisql > create database db1;
[INFO] Sql syntax parse ok!
minisql > create database db2;
[INFO] Sql syntax parse ok!
```

2.展示数据库:
show databases;
选定数据库:use db0;

```
minisql > show databases;
[INFO] Sql syntax parse ok!
[DATABASE] db2
[DATABASE] db1
[DATABASE] db0
minisql > use db0;
[INFO] Sql syntax parse ok!
[INFO] Use successfully!
```

3.创建两个表:

```
create table account(
id int,
name char(30) unique,
balance float,
primary key(id));
```

```
create table test(
id int,
cc float);
```

```
minisql >
create table account(
id int,
name char(30) unique,
balance float,
primary key(id));
[INFO] Sql syntax parse ok!
[INFO] Create table successfully!
minisql > create table test(
id int,
cc float);

[INFO] Sql syntax parse ok!
[INFO] Create table successfully!
```

4.展示所有表

```
show tables;
```

```
minisql > show tables;
[INFO] Sql syntax parse ok!
[TITLE] Tables_in_db0
2
[TABLE] test
[TABLE] account
```

5.插入二十条记录

```
execfile "/mnt/e/minisql/src/exe.txt";
```

```
minisql >
execfile "/mnt/e/minisql-1/src/exe.txt";
[INFO] Sql syntax parse ok!
1
1
[INFO] Insert successfully!
2
[INFO] Insert successfully!
3
[INFO] Insert successfully!
4
[INFO] Insert successfully!
5
[INFO] Insert successfully!
6
[INFO] Insert successfully!
7
[INFO] Insert successfully!
8
[INFO] Insert successfully!
9
[INFO] Insert successfully!
10
[INFO] Insert successfully!
11
[INFO] Insert successfully!
12
[INFO] Insert successfully!
13
[INFO] Insert successfully!
14
[INFO] Insert successfully!
15
[INFO] Insert successfully!
16
[INFO] Insert successfully!
17
[INFO] Insert successfully!
18
[INFO] Insert successfully!
19
[INFO] Insert successfully!
20
[INFO] Insert successfully!

total time:2.4002ms
```

6.显示全部的record查看插入结果

select * from account;

```
minisql > select * from account;
[INFO] Sql syntax parse ok!
-----------------------------------------------------------------
|id                  |name                |balance               |
-----------------------------------------------------------------
|12500000            |name0               |544.6199?             |
-----------------------------------------------------------------
|12500001            |name1               |57.41000              |
-----------------------------------------------------------------
|12500002            |name2               |264.6900?             |
-----------------------------------------------------------------
|12500003            |name3               |596.5900?             |
-----------------------------------------------------------------
|12500004            |name4               |34.29999              |
-----------------------------------------------------------------
|12500005            |name5               |542.5000?             |
-----------------------------------------------------------------
|12500006            |name6               |571.9199?             |
-----------------------------------------------------------------
|12500007            |name7               |528.4099?             |
-----------------------------------------------------------------
|12500008            |name8               |288.4400?             |
-----------------------------------------------------------------
|12500009            |name9               |919.6400?             |
-----------------------------------------------------------------
|12500010            |name10              |963.8499?             |
-----------------------------------------------------------------
|12500011            |name11              |739.8300?             |
-----------------------------------------------------------------
|12500012            |name12              |940.8900?             |
-----------------------------------------------------------------
|12500013            |name13              |775.7100?             |
-----------------------------------------------------------------
|12500014            |name14              |646.5000?             |
-----------------------------------------------------------------
|12500015            |name15              |276.5199?             |
-----------------------------------------------------------------
|12500016            |name16              |269.9599?             |
-----------------------------------------------------------------
|12500017            |name17              |72.69999              |
-----------------------------------------------------------------
|12500018            |name18              |622.7800?             |
-----------------------------------------------------------------
|12500019            |name19              |131.9299?             |
-----------------------------------------------------------------
```

7.点查询操作:
select * from account where id = 12500008;
select * from account where balance = 57.41000;
select * from account where name = "name6";

```
select * from account where id = 12500008;
[INFO] Sql syntax parse ok!
-------------------------------------------------------------
|id                  |name                |balance           |
-------------------------------------------------------------
|12500008            |name8               |288.4400          |
-------------------------------------------------------------
minisql > select * from account where balance = 57.41000;
[INFO] Sql syntax parse ok!
-------------------------------------------------------------
|id                  |name                |balance           |
-------------------------------------------------------------
|12500001            |name1               |57.41000          |
-------------------------------------------------------------
minisql > select * from account where name = "name6";
[INFO] Sql syntax parse ok!
-------------------------------------------------------------
|id                  |name                |balance           |
-------------------------------------------------------------
|12500006            |name6               |571.9199          |
-------------------------------------------------------------
```

select * from account where id <> 12500007;

```
minisql > select * from account where id <> 12500007;
[INFO] Sql syntax parse ok!
```

id	name	balance
12500000	name0	544.6199
12500001	name1	57.41000
12500002	name2	264.6900
12500003	name3	596.5900
12500004	name4	34.29999
12500005	name5	542.5000
12500006	name6	571.9199
12500008	name8	288.4400
12500009	name9	919.6400
12500010	name10	963.8499
12500011	name11	739.8300
12500012	name12	940.8900
12500013	name13	775.7100
12500014	name14	646.5000
12500015	name15	276.5199
12500016	name16	269.9599
12500017	name17	72.69999
12500018	name18	622.7800
12500019	name19	131.9299

select * from account where balance <> 57.41000;

```
minisql > select * from account where balance <> 57.41000;
[INFO] Sql syntax parse ok!
```

id	name	balance
12500000	name0	544.6199
12500002	name2	264.6900
12500003	name3	596.5900
12500004	name4	34.29999
12500005	name5	542.5000
12500006	name6	571.9199
12500007	name7	528.4099
12500008	name8	288.4400
12500009	name9	919.6400
12500010	name10	963.8499
12500011	name11	739.8300
12500012	name12	940.8900
12500013	name13	775.7100
12500014	name14	646.5000
12500015	name15	276.5199
12500016	name16	269.9599
12500017	name17	72.69999
12500018	name18	622.7800
12500019	name19	131.9299

select * from account where name <> "name0";

```
minisql > select * from account where name <> "name0";
[INFO] Sql syntax parse ok!
```

id	name	balance
12500001	name1	57.41000
12500010	name10	963.8499
12500011	name11	739.8300
12500012	name12	940.8900
12500013	name13	775.7100
12500014	name14	646.5000
12500015	name15	276.5199
12500016	name16	269.9599
12500017	name17	72.69999
12500018	name18	622.7800
12500019	name19	131.9299
12500002	name2	264.6900
12500003	name3	596.5900
12500004	name4	34.29999
12500005	name5	542.5000
12500006	name6	571.9199
12500007	name7	528.4099
12500008	name8	288.4400
12500009	name9	919.6400

select * from account where id >= 12500006;

```
minisql > select * from account where id >= 12500006;
[INFO] Sql syntax parse ok!
```

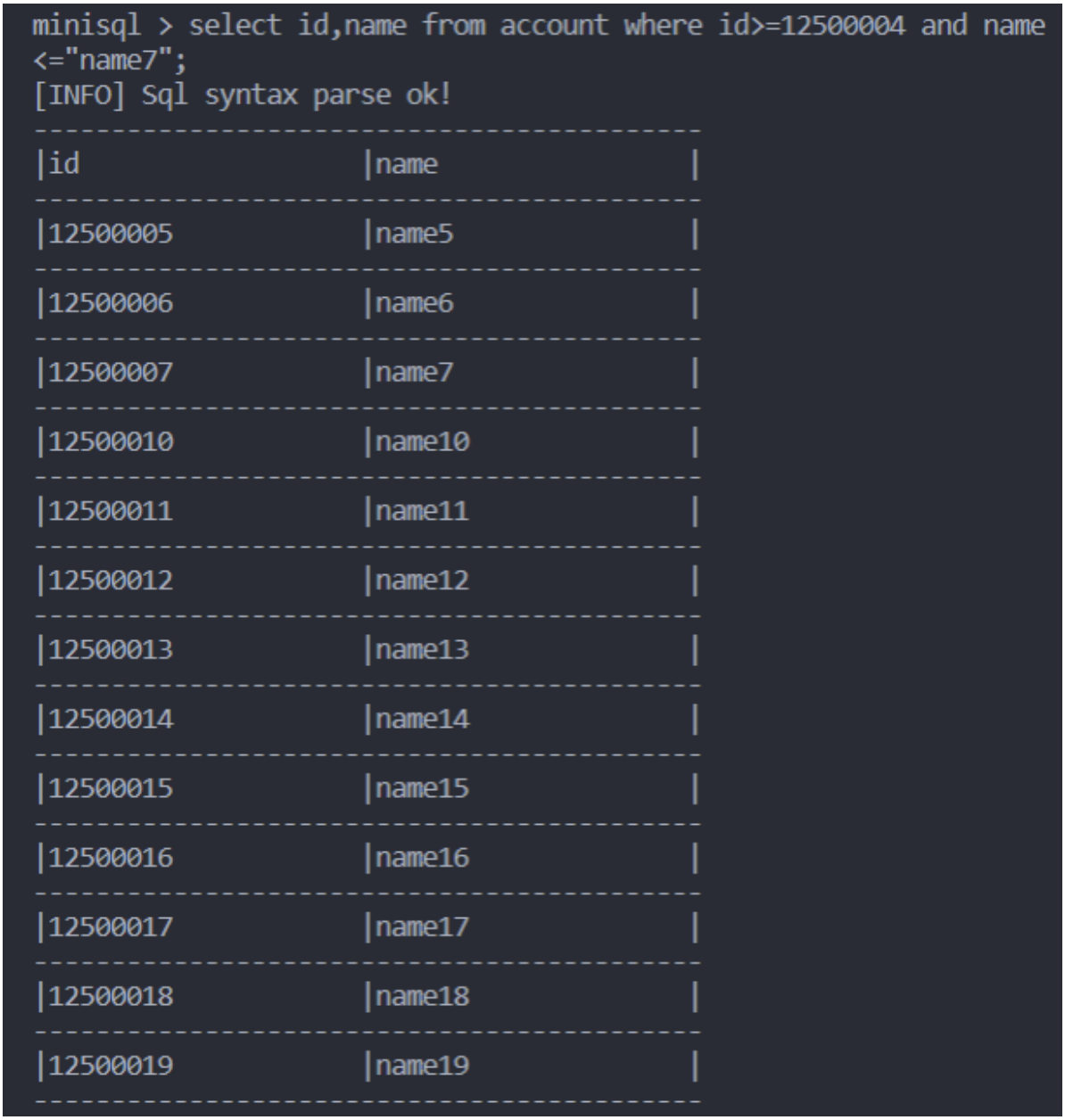
id	name	balance
12500007	name7	528.4099
12500008	name8	288.4400
12500009	name9	919.6400
12500010	name10	963.8499
12500011	name11	739.8300
12500012	name12	940.8900
12500013	name13	775.7100
12500014	name14	646.5000
12500015	name15	276.5199
12500016	name16	269.9599
12500017	name17	72.69999
12500018	name18	622.7800
12500019	name19	131.9299

select * from account where id <= 12500009;

```
minisql > select * from account where id <= 12500009;
[INFO] Sql syntax parse ok!
-----------------------------------------------------------------
|id                  |name                |balance               |
-----------------------------------------------------------------
|12500000            |name0               |544.6199?             |
-----------------------------------------------------------------
|12500001            |name1               |57.41000              |
-----------------------------------------------------------------
|12500002            |name2               |264.6900?             |
-----------------------------------------------------------------
|12500003            |name3               |596.5900?             |
-----------------------------------------------------------------
|12500004            |name4               |34.29999              |
-----------------------------------------------------------------
|12500005            |name5               |542.5000?             |
-----------------------------------------------------------------
|12500006            |name6               |571.9199?             |
-----------------------------------------------------------------
|12500007            |name7               |528.4099?             |
-----------------------------------------------------------------
|12500008            |name8               |288.4400?             |
-----------------------------------------------------------------
|12500009            |name9               |919.6400?             |
-----------------------------------------------------------------
```

8.多条件查询与投影

select id,name from account where id>=12500004 and name <="name7";



select name,id from account where id>=12500004 and name <="name7" or id = 125000008;

```
minisql > select name,id from account where id>=12500004 and name
<="name7" or id = 125000008;
[INFO] Sql syntax parse ok!
-----------------------------------------
|name                |id                  |
-----------------------------------------
|12500005            |name5               |
-----------------------------------------
|12500006            |name6               |
-----------------------------------------
|12500007            |name7               |
-----------------------------------------
|12500010            |name10              |
-----------------------------------------
|12500011            |name11              |
-----------------------------------------
|12500012            |name12              |
-----------------------------------------
|12500013            |name13              |
-----------------------------------------
|12500014            |name14              |
-----------------------------------------
|12500015            |name15              |
-----------------------------------------
|12500016            |name16              |
-----------------------------------------
|12500017            |name17              |
-----------------------------------------
|12500018            |name18              |
-----------------------------------------
|12500019            |name19              |
-----------------------------------------
```

select balance from account where name <> "name8" and id <> 125000009 or id <= 12500004;

```
minisql > select balance from account where name <> "name8" and id
 <> 125000009 or id <= 12500004;
[INFO] Sql syntax parse ok!
----------------------
|balance             |
----------------------
|544.6199?           |
----------------------
|57.41000            |
----------------------
|264.6900?           |
----------------------
|596.5900?           |
----------------------
|34.29999            |
----------------------
|542.5000?           |
----------------------
|571.9199?           |
----------------------
|528.4099?           |
----------------------
|919.6400?           |
----------------------
|963.8499?           |
----------------------
|739.8300?           |
----------------------
|940.8900?           |
----------------------
|775.7100?           |
----------------------
|646.5000?           |
----------------------
|276.5199?           |
----------------------
|269.9599?           |
----------------------
|72.69999            |
----------------------
|622.7800?           |
----------------------
|131.9299?           |
----------------------
```

9.唯一约束

insert into account values(12500000,"name21",23.3);

insert into account values(12500021,"name0",25.12);

```
minisql > insert into account values(12500000,"name21",23.3);
[INFO] Sql syntax parse ok!
[INFO] Insert successfully!
minisql > insert into account values(12500021,"name0",25.12);
[INFO] Sql syntax parse ok!
[INFO] Insert failed!
```

10.删除

delete from account where id = 12500000;

delete from account where id >=12500004 or id <=12500001;

11.更新

update account set balance = 12.5,name = "name22" where id = 12500002;

update account set balace = 17.5 where id = 12500003;

```
minisql > update account set balance = 12.5,name = "name22" where
id = 12500002;
[INFO] Sql syntax parse ok!
[INFO] Update successfully!
minisql > update account set balace = 17.5 where id = 12500003;
[INFO] Sql syntax parse ok!
[INFO] Update successfully!
minisql > select * from account;
[INFO] Sql syntax parse ok!
-------------------------------------------------------------
|id                  |name                |balance           |
-------------------------------------------------------------
|12500002            |name22              |12.50000          |
-------------------------------------------------------------
|12500003            |name3               |596.5900          |
-------------------------------------------------------------
|12500004            |name4               |34.29999          |
```

12.索引

show indexes;

```
minisql > show indexes;
[INFO] Sql syntax parse ok!
[TITLE] Indexes Of Table account
[INDEX] name
```

delete from account;

```
minisql > delete from account;
[INFO] Sql syntax parse ok!
minisql > select * from account;
[INFO] Sql syntax parse ok!
-------------------------------------------------------------
|id                  |name                |balance           |
-------------------------------------------------------------
```

第五章

5.1 堆表插入优化

当向堆表中插入一条记录时，一种简单的做法是，沿着`TablePage`构成的链表依次查找，直到找到第一个能够容纳该记录的`TablePage`。这是*First Fit* 策略，虽然这个策略的空间利用率会较好，但是单次插入可能会访问堆表中的所有页，这是我们不能接受的。利用空间换时间的思想，我们使用*Next Fit* 策略，每次从最后一个页开始插入。

效果是很明显的，10W条的插入从分钟级别变成了1.3s。