

**系统能力综合培养实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学 院： | 计算机学院 | | |
| 专 业： | 计算机科学与技术 | | |
| 班 级： | ACM1601班、计算机1609班 | | |
| 学 号： |  | U201614774 | U201614767 |
| 姓 名： | 黄佳汇 | 杨祖容 | 王通 |
| 指导教师： | 谢美意、左琼 | | |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2019年 月 日

目 录

[1 实验任务概述 1](#_Toc29200685)

[1.1 实验目的 1](#_Toc29200686)

[1.2 实验要求 1](#_Toc29200687)

[1.3 HustBase体系结构介绍 2](#_Toc29200688)

[1.4 小组成员分工 4](#_Toc29200689)

[2 记录管理模块 5](#_Toc29200690)

[2.1 模块概述 5](#_Toc29200691)

[2.2 模块设计 5](#_Toc29200692)

[2.3 模块实现 7](#_Toc29200693)

[2.3.1记录文件管理 7](#_Toc29200694)

[2.3.2文件扫描函数 7](#_Toc29200695)

[2.3.3记录操作 8](#_Toc29200696)

[2.4 实验过程 9](#_Toc29200697)

[3 系统管理模块 9](#_Toc29200698)

[3.1 模块概述 9](#_Toc29200699)

[3.2 模块设计 9](#_Toc29200700)

[3.3 模块实现 9](#_Toc29200701)

[3.4 实验过程 9](#_Toc29200702)

[4 查询处理模块 9](#_Toc29200703)

[4.1 模块概述 9](#_Toc29200704)

[4.2 模块设计 10](#_Toc29200705)

[4.3 模块实现 11](#_Toc29200706)

[4.4 实验过程 12](#_Toc29200707)

[5 索引管理模块 13](#_Toc29200708)

[5.1 模块概述 13](#_Toc29200709)

[5.2 模块设计 13](#_Toc29200710)

[5.2.1 索引及其概念 13](#_Toc29200711)

[5.2.2 B+树及其概念 13](#_Toc29200712)

[5.2.3 模块和接口设计 14](#_Toc29200713)

[5.3 模块实现 16](#_Toc29200714)

[5.4 实验过程 19](#_Toc29200715)

[6 实验总结 20](#_Toc29200716)

[6.1 工作总结 20](#_Toc29200717)

[6.2 改进方案 20](#_Toc29200718)

[附录 22](#_Toc29200719)

# 实验任务概述

## 1.1 实验目的

1)了解数据库、操作系统、编译、数据结构等专业知识在关系数据库管理系统实现中的综合运用；

2)了解关系数据库管理软件的体系结构和设计方法；

3)了解关系数据库中数据和元数据的组织方法；

4)掌握关系数据的底层存储实现技术；

5)掌握基本关系操作在关系数据上的实现算法；

6)掌握基本SQL语句的实现方法；

7)掌握基本的查询优化技术。

## 1.2 实验要求

基于HustBase系统的总体设计架构，根据预先给定的系统框架、部分模块和接口要求，设计并实现系统中的其余模块功能，完成一个具有基本数据定义、数据操纵和数据查询功能的单用户关系数据库管理系统。

实验课时安排为四周，按周次划分为四个阶段。除去已预先提供的用户界面、语法分析和页面管理模块，要求学生自己设计实现记录管理、系统管理、查询处理和索引管理模块以及所有模块的联调，完成整个HustBase系统的开发。具体课程内容安排如表2.1所示。

表2.1 实验内容安排

|  |  |
| --- | --- |
| **周 次** | **内 容 安 排** |
| 第一周 | 学习HustBase的系统结构和工作原理，在已提供模块的基础上，开始记录管理模块、系统管理模块和索引管理模块的编码。 |
| 第二周 | 完成建库、建表功能的编码和联调；开始数据查询模块的编码。 |
| 第三周 | 完成数据操纵和数据查询功能的编码和联调。 |
| 第四周 | 完成索引管理功能，并与其他相关模块结合，完成系统整体联调。 |

HustBase系统的开发环境为Windows 操作系统及 Visual Studio 2010，开发语言为C语言。为保证良好的模块独立性、便于共同开发及阶段性检查，每个模块必须按文档规定的格式对外提供调用接口，但是模块内部的实现方案不做统一规定，由学生自行设计。

具体要求：

1. 实验内容采取2-3人小组的形式合作完成，每个学生在小组中均应承担合理的工作量，基本原则为“共同设计，分工编码”。
2. 考核方式为现场演示及答辩和测试程序验证，最终成绩根据系统完成情况、演示答辩情况和实验报告质量综合评分，同组学生的成绩相同。

课程结束后，以小组为单位提交系统的源码、编译后的程序及实验报告。

## 1.3 HustBase体系结构介绍

HustBase是一个用于实践教学的简化版数据库管理系统，其设计目标是支持简单的SQL语句，提供基本的数据定义、数据操纵、数据查询和查询优化功能。为了便于学生在为期四周的实验周期内掌握系统框架并完成实验内容，与实际的商用DBMS相比，HustBase做了大量的功能简化，舍弃了如多用户、事务、故障恢复等常见特性，支持的SQL语法也非常有限。

HustBase的体系架构如图3.1所示。系统由页面管理、记录管理、索引管理、系统管理、查询处理、语法分析和用户界面等模块构成。各模块功能如下：

（1）页面管理：数据库中所有的数据、元数据和索引数据均以文件的形式存储在磁盘上。Windows操作系统支持的文件结构为流文件，而数据库中的数据是以记录为单位来进行存取的，为了实现对磁盘文件的高效I/O，需要将流文件转换为记录文件。这个过程分为两步：首先，将连续的流文件划分为由若干个固定大小的页面，将其抽象为分页文件；其次，将分页文件的页面划分为若干个固定大小的记录插槽或索引项插槽，将其抽象为记录文件或索引文件。本模块提供第一步所需功能，即将流文件抽象为分页文件，并提供以页面为单位的文件读写接口，具体包括：创建、销毁、打开和关闭分页文件；遍历指定文件中的所有页面；从指定文件中读取一个特定页面；在指定文件中添加、删除及修改页面等。

（2）记录管理：系统表和数据表中的数据均以记录为单位进行存取。本模块在页面管理模块的基础上，将文件中每个页面又划分为若干个固定大小的记录插槽，实现记录的插入、删除、修改和查找，为上层的数据操纵及数据查询功能提供支持。具体包括：创建、销毁、打开和关闭记录文件；插入、删除、修改记录；查找符合指定条件的记录等。

（3）索引管理：利用索引，系统可以为查询提供快速访问路径。本模块在页面管理模块的基础上，将文件中每个页面又划分为若干个固定大小的索引项插槽，提供对索引项的管理。索引文件与记录文件的不同之处在于，索引采用B+树结构来组织索引项，以便提高关键字查找的效率。具体包括：创建、销毁、打开和关闭索引文件；索引项的插入、删除、修改以及查找等。



图1.1 HustBase系统体系架构图

（4）系统管理：本模块提供对数据定义（CREATE/DROP TABLE，CREATE/DROP INDEX）和数据操纵（INSERT、DELETE、UPDATE）功能的支持。系统管理模块的实现依赖于记录管理模块和索引管理模块。

（5）查询处理：本模块用于提供对数据查询（SELECT）功能的支持，并向客户端返回最终的查询结果集。查询处理模块的功能依赖于记录管理模块和索引管理模块。

（6）语法分析：本模块提供对SQL命令的语法分析，并将分析结果以语法树的形式提供给系统管理模块和查询处理模块，以便其做进一步的分析和处理。

（7）用户界面：本模块为用户提供操作界面，接收用户输入的SQL命令，并向用户输出SQL命令的执行结果。

在图3.1显示的各模块中，蓝色模块的功能代码已经提供，红色模块则需要学生自行设计实现。学生根据自己的实际能力，也可以对蓝色模块的代码进行一定程度的修改以扩展系统功能，但原则上不应修改本文档中规定的模块接口。

## 1.4 小组成员分工

杨祖容：任务A

黄佳汇：任务B

王通 ：任务C

# 记录管理模块

## 模块概述

数据表中的数据由若干条记录构成，最终存储在磁盘文件中。本模块在分页文件的基础上，进一步将页面文件改造为记录文件，每个页面的数据区被划分为若干个记录插槽，每个记录插槽用于存放一条记录，并对外提供以记录为单位的记录文件读写接口。记录管理模块依赖于下层的页面管理模块，页面管理模块以页面为单位进行文件读写，而记录管理模块则以记录为单位进行文件读写，每条记录以页面号+插槽号为唯一标识。为简化实现，规定如下：1. 记录文件中存储的记录为定长记录；2. 同一个文件中的记录格式相同，即一个数据表对应一个文件；3. 记录不能跨页，且单条记录的长度不超过页面数据区大小

## 模块设计



图2.1 记录文件结构示意图

记录文件是在页面文件的基础上改造的，所有每个记录文件同样由若干个页面构成，其中第0页为页面信息控制页，存放页面使用情况信息。

1）记录信息控制页

根据图2.1，记录文件的第1页为记录信息控制页。该页的主要功能是控制管理整个页面的数据页分配，包括当前记录总数、每个记录的长度以及每页可以装载的记录数，从pData[0]开始，存放一个数据结构：

typedef struct {

int nRecords; //当前文件中包含的记录数

int recordSize; //每个记录的大小

int recordsPerPage; //每个页面可以装载的记录数量

} RM\_FileSubHeader;

接着，从pData[sizeof(RM\_FileSubHeader)]开始，存放一个位图，用来标记记录文件中哪些页面为满页，哪些页面为非满页。非满页是指页面中还存在空闲的记录插槽的那些页面。位图中相应位为0表示非满页，为1表示满页。该位图与第0页中的位图相似，但是所表示的含义不同。位图的初始值为3，以表示页面信息控制页以及记录信息控制页是满页。

2）数据页

从分页文件的第2页开始为数据页面，每个数据页面被划分为若干个记录插槽，记录在数据页面中采用顺序存储方式。此外，为了区分当前页面中哪些插槽已经存放了记录数据，哪些插槽为空，还需要记录相关的控制信息。

在每个数据页面中，从pData[0]开始存放一个int 大小的值，用来表示当前数据页中包含的记录数，所有的数据页的记录数量之和即是记录信息控制页中的总记录数值。

然后存放一个位图，用于管理该页面中记录插槽的使用情况。如果一个数据页面能够存放n个记录，则需要一个大小为 Sup(n/8) 字节的位图。若某个插槽含有有效记录，位图中相应的位为1；若某个插槽为空插槽，相应的位为0。位图后的空间按记录长度划分为若干个插槽。注意，位图的大小取决于一个页面能够存放的记录数，而一个页面能够存放的记录数又是由记录的大小决定。因此，每个记录文件在创建时都需要根据其存储的记录长度来计算位图所需的字节数。

最后就是存放所有记录的数组，其中记录的数据结构如下所示：

typedef struct {

PageNum pageNum; //记录所在页的页号

SlotNum slotNum; //记录的插槽号

bool bValid; //true表示为一个有效记录的标识符

}RID;

typedef struct{

bool bValid; // False表示还未被读入记录

RID rid; // 记录的标识符

char \*pData; //记录所存储的数据

}RM\_Record;

每一条记录被分为3个部分。首先是一个bool型的bValid，表示这条记录槽是否已经使用，false表示未被使用。然后是这条记录的标识符，最后就是该条记录的数据部分。

## 模块实现

### 2.3.1记录文件管理

创建记录文件函数RM\_CreateFile会传入文件名以及记录的长度，然后以此来初始化记录信息控制页的各项基本信息，从而成功的创建一个记录文件。

使用一个记录文件前，需要先打开文件（RM\_OpenFile），获取一个记录文件句柄，后续的文件操作均以该句柄为标识，不再使用该文件时，要关闭文件（RM\_CloseFile）释放资源。该句柄的数据结构如下所示：

typedef struct{

bool bOpen;//句柄是否打开（是否正在被使用）

PF\_FileHandle \*pf\_fileHandle;//底层页面控制句柄

RM\_FileSubHeader \*rm\_fileSubHeader;//页面控制信息

char \*rBitmap;//记录信息控制页的位图指针

}RM\_FileHandle;

### 2.3.2文件扫描函数

打开一个文件扫描是在当前的句柄基础上，将扫描条件等一些数据一起添加到扫描句柄，从而开始扫描，扫描句柄的数据结构如下所示：

typedef struct{

bool bOpen; //扫描是否打开

RM\_FileHandle \*pRMFileHandle; //扫描的记录文件句柄

int conNum; //扫描涉及的条件数量

Con \*conditions; //扫描涉及的条件数组指针

PF\_PageHandle PageHandle; //处理中的页面句柄

PageNum pn; //扫描即将处理的页面号

SlotNum sn; //扫描即将处理的插槽号

}RM\_FileScan;

typedef struct

{

int bLhsIsAttr,bRhsIsAttr;//条件的左、右分别是属性（1）还是值（0）

AttrType attrType;//该条件中数据的类型

int LattrLength,RattrLength;//若是属性的话，表示属性的长度

int LattrOffset,RattrOffset;//若是属性的话，表示属性的偏移量

CompOp compOp;//比较操作符

void \*Lvalue,\*Rvalue;//若是值的话，指向对应的值

}Con;

其中pn需要根据位图等信息，计算出第一条有效记录所在的页面号，sn默认初始值为-1。

在使用中，用户应先调用此函数初始化文件扫描结构，然后再调用GetNextRec函数来逐个返回文件中满足条件的记录。如果条件数量conNum为0，则意味着检索文件中的所有记录。如果条件不为空，则要对每条记录进行条件比较，只有满足所有条件的记录才被返回。注意GetNextRec函数只是返回满足条件的记录RID还需要调用记录操作函数GetRec来获取记录数据。

关闭扫描操作便是简单的关闭即可，同时释放相应的资源。

### 2.3.3记录操作

在以上文件结构设计方案的基础上，记录管理模块需要实现对记录文件的各种读写操作：

1）插入记录InsertRec

插入记录时，首先在文件中找到一个非满页（若没有则需要申请一个新的页面），在该页中找到一个空插槽并插入记录。记录插入后，要更新控制页，即将文件包含的记录数加1，同时将插槽位图中的相应位置1，还要更新该数据也上的记录总数。如果记录插入后，该页面已经没有空插槽，则将该页面标记为满页，即在记录文件的控制页的位图上将该页面所对应的位置1。最后将插入位置的rid返回。

2）删除记录DeleteRec

删除记录时，对于一个给定的记录ID，先检查该记录ID的有效性，如果无效，将返回一个错误值。如果有效，则找到相应的记录。记录删除并不需要将页面中记录所占用的区域清零，只需要将插槽位图中将相应的位置0，同时将该数据槽的bValid值置为false，之后再更新当前页面的记录数、满页情况、分配情况、总记录数等信息。

3）修改记录UpdateRec

修改记录时，对给定的记录ID，先检查该记录ID的有效性，然后找到相应的记录，用新的记录值替换原记录值。

4）查找记录GetRec

根据传入的文件句柄以及需要RID，找到相应的记录数据，并将数据返回即可。当然在获取记录之前，还是要检查该记录的有效性。

## 实验过程

1.扫描搜索时开始位置的控制

开始设计时，是直接找到第一条有效记录的位置，然后赋值给sn，但是在设计GetNextRec时，为了方便循环，设计的是先查找下一条有效记录记录的位置，然后判断记录是否满足条件，然后返回。因此每次查询都会略过第一条记录，导致结果不准确，因此后序直接将扫描条件的起始位置中sn置为-1，这样就可以保证获取到所有满足条件的记录。

2．获取记录数据时，长度错误

在比较记录是否满足条件的函数中，需要先获取记录中的数据，由于数据包含int，float和char型，在获取记录时，需要根据不同的类型来控制获取数据的长度，当需要获取的数据为chars型时，需要分配一个属性长度+1的数组来保存数据，因为字符串末尾有‘\0’。当为int或者float时，直接分配一个属性长度大小的数据即可。

# 系统管理模块

## 模块概述

## 模块设计

## 模块实现

## 实验过程

# 查询处理模块

## 模块概述

查询处理模块负责select语句的处理。HustBase只支持简单的select语句。根据关系模型理论，最基本的查询处理方法为先计算关系的笛卡尔集，然后选择并投影出最终的结果集。但是在本模块的设计实现中，考虑利用代数优化方法及索引对查询过程进行优化。

## 模块设计

本模块涉及的主要数据结构包括：

struct RelAttr {

char \*relName; //表名

char \*attrName; //属性名

};

typedef struct \_Condition Condition;

struct \_Condition{

int bLhsIsAttr; //操作符左边是属性为1，是值时为0

Value lhsValue; //当左边是值时，保存该值的信息

RelAttr lhsAttr; //当左边是属性时，保存属性信息

CompOp op; //比较运算符，详见5.2.1

int bRhsIsAttr; //操作符右边是属性为1，是值时为0

RelAttr rhsAttr; //当右边是属性时，保存属性信息

Value rhsValue; //当左边是值时，保存该值的信息

};

typedef struct SelResult{

int col\_num; //结果集的字段数，最多20个字段

int row\_num; //结果集的记录数

AttrType type[20]; //结果集各字段的数据类型

int length[20]; //结果集各字段值的长度

char fields[20][20]; //结果集各字段的字段名，字段名不超过20个字符

char \*\*res[100]; //结果集数据，每个链表节点最多记录100条记录

SelResult \*next\_res; //指向下一个节点的指针

}SelResult;

本模块只有一个接口，select接口，该接口设计较为复杂，将会详细地解释分析并且关注与具体细节上。

首先我们可以把所有的条件分为三种类型，值与值、值与参数和参数与参数。其中前两种是可以在合并成笛卡尔集之前进行求值化简，具体操作为：

1. 对于值与值，如果这个值与值判断是正确的，将会在删除这个条件，如果这个判断是错误的，将会直接返回空集，表示没有任何记录满足；

2. 对于值与参数来说，将会根据这个参数是否是索引进行扫描。如果是索引的话，就可以通过索引进行扫描，如果不是，就简单通过记录进行扫描。如果在这步之前，已经把这个参数对应的数据表扫描过了，那么仅在扫描过后的集合中再次选择确认，其余的丢弃，否则就创建一个新的扫描集。

在以上步骤之后，实则已经进行了单表查询，在这步之后，对于select中选定了但是仍然没有被扫描的数据表全部加入到扫描集中，最后使得每一个选定的数据表都有自己对应的一个扫描集，作为合并笛卡尔集的基础。

下一步将会通过查询系统列文件的所有列记录，确定一个长记录（包含了数据表所有信息笛卡尔集的一个记录）的格式、每一列长度、每一列偏移值等信息。同时还会计算出计算笛卡尔集的时候一些例如连续写入次数、循环写入次数、数据表偏移值、数据表长度等信息。

最后根据得到的信息，构建出一个笛卡尔集。

然后根据最后一种参数与参数的条件，筛选出最后满足所有条件的记录，从而能都得到的查询结果。

## 模块实现

RC Select (int nSelAttrs, RelAttr \*selAttrs, int nRelations, char \*\* relations, int nConditions, Condition \*conditions, SelResult \* res)

前六个参数逻辑上可分为三组。每一组的第一个参数是一个整型n，表示本组中第二个参数的条目数量；第二个参数是一个包含实际条目数的为n的数组。例如，参数nSelAttrs包含所选属性的数量，而参数selAttrs是包含实际属性的长度为nSelAttrs的数组，其余类似。具体而言，第一组参数表示查询涉及的属性；第二组表示查询涉及的表；第三组表示查询条件。最后一个参数res用于返回查询结果集。

其中第二个扫描集中，status表示这一条rid是否满足所有条件，在每一次判断单表条件进行扫描的时候，把所有的active的变为inactive，而每扫描到一条对应的数据，就把它的status再次变成active，扫描完之后，把所有剩余的inactive的记录变为death。这样就能筛选下所有的记录。

单表查询的主要思路已经在上面模块设计中提到过，接下来再具体描述笛卡尔集的求法。经过观察可以发现，笛卡尔集中每一个值都会出现相同的次数，但是每一个值出现的规律不一样。对于第一列来说，是连续出现N次，但是只会出现一次，而对于最后一列来说，它只会每次出现一次，但是会循环着N次写入。所以我们要通过两次循环才能计算出笛卡尔集的写入顺序与次数。

## 实验过程

1. 关于出现不等于符号的时候单表查询的选择

一旦出现不等于号的时候，不管这个列是否是索引，一定不能使用索引操作。因为索引基于B+树，对不等于符号的判断没有任何加速意义，所以出现不等于号的时候，直接使用记录文件扫描即可。

2. 对于各种select操作中的潜在的语义错误判断

一、select a from A

如果A表中没有a这个列的话，将会返回语法错误；

二、select a from A, B

如果A，B表中同时有这个a列的话，将会返回语法错误；

三、selectA.a from A, B

如果A中没有a，但是B中有A的话，在这里我们仍旧视为是正确的，作为一个简单的冗余支持；

四、select a from A, B where A.b>B.c

如果此时A.b与B.c的类型不同，即无法进行对应的判断，将会返回语法错误；

五、selectafromA, B where C.a>A.a

此时由于根本没有fromC表，该句将会返回语法错误；

六、select a from A, B where A.a> “a”

如果此时A.a的数据类型不是字符串的话，将会返回语法错误。

# 索引管理模块

## 模块概述

索引模块是HustBase的一个加速模块，调用索引模块可以提高在单表查询时的搜索效率，一般上将会从O(N)提高到O(LogM(N))的速度。其中M的值是B+树的order值，即每一层的节点数目，当order值是2的时候，B+树就是一棵二叉搜索树。在本系统中，这一模块是难度最大的，其每一个节点是页面模块中提供的每一页，这样就能实现高速查询与实现。

## 模块设计

### 5.2.1 索引及其概念

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。如果想按特定职员的姓来查找他或她，则与在表中搜索所有的行相比，索引有助于更快地获取信息。

索引的一个主要目的就是加快检索表中数据，亦即能协助信息搜索者尽快的找到符合限制条件的记录ID的辅助数据结构。

### 5.2.2 B+树及其概念

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+树性质：

1)根节点只有1个，分支数范围为[2,m]；

2)除根以外的每个内部节点的分支数范围为[Sup(m/2),m]；

3)叶子节点包含的关键字个数范围为[Sup(m/2),m]；

4)所有节点中的关键字按升序排列，即满足K[i] < K[i+1]；

5)非叶子节点中的指针P[i]指向关键字值属于[K[i], K[i+1])的子树；

6)所有的叶子节点都在同一层；

7)叶子节点中包含了全部关键字，及指向包含这些关键字记录的指针，且叶子节点本身依关键字的大小自小而大顺序链接。

图6.1为一个序数为3的B+树示意图。

一个普通的B+树如图5.1所示：

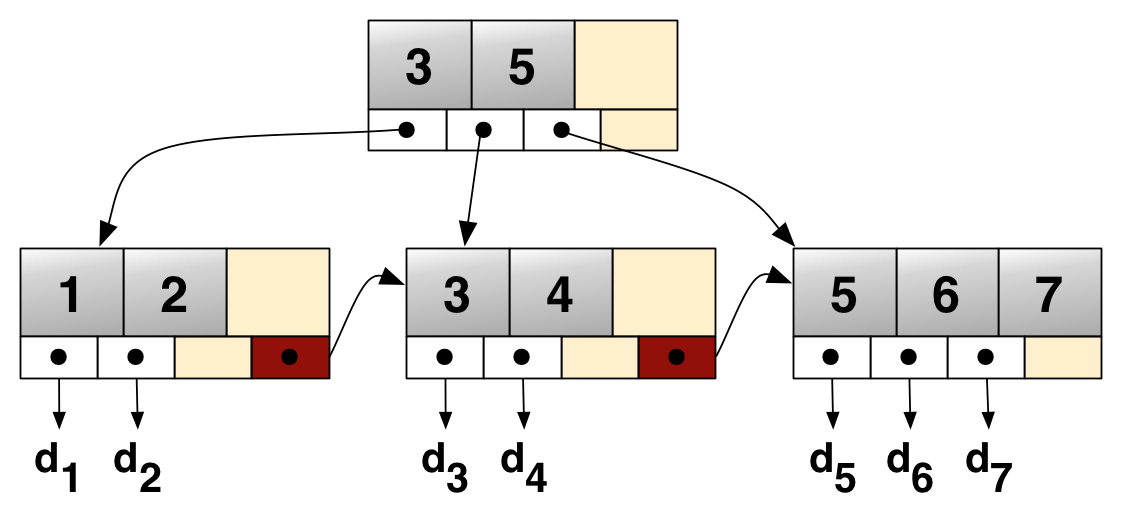


图5.1 B+树

### 5.2.3 模块和接口设计

因为给出的框架中已经有了接口设计和模块设计，因此这里引用任务书中给出的接口设计。

1. 数据结构设计：

描述索引控制信息的数据结构：

typedef struct{

int attrLength; //建立索引的属性值的长度

int keyLength; //B+树中关键字的长度

AttrType attrType; //建立索引的属性值的类型

PageNum rootPage; //B+树根节点的页面号

PageNum first\_leaf; //B+树第一个叶子节点的页面号

int order; //B+树的序数

}IX\_FileHeader;

除第1页外，其它索引数据页面的这片区域为空白区域，不存储有效信息。索引控制信息结构在索引建立的时候初始化，并且随着索引文件的变化，其中的rootPage会发生变化。

存放当前页面所存储节点的节点控制信息，对应的数据结构为：

typedef struct{

int is\_leaf; //该节点是否为叶子节点

int keynum; //该节点实际包含的关键字个数

PageNum parent; //指向父节点所在的页面号

PageNum brother; //指向右兄弟节点所在的页面号

char \*keys; //指向关键字区的指针

RID \*rids; //指向指针区的指针

}IX\_Node;

从pData[sizeof(IX\_FileHeader)+ sizeof(IX\_Node)]开始，存放B+树节点信息。每个节点中包含一组关键字和一组指针，关键字和指针的数量相等。对于非叶子节点来说，指针指向的是下层子树的一个根节点；对于叶子节点来说，指针指向的则是包含关键字的那条记录。

索引句柄的结构为：

typedef struct{

bool bOpen; //该索引句柄是否已经与一个文件关联

PF\_FileHandle fileHandle; //该索引文件对应的页面文件句柄

IX\_FileHeader fileHeader; //该索引文件的控制页句柄

}IX\_IndexHandle;

2. 接口设计

RC CreateIndex (const char \*fileName, AttrType attrType,int attrLength)

此函数创建一个名为fileName的索引。attrType描述被索引属性的类型，attrLength描述被索引属性的长度。

RC OpenIndex (const char \*fileName, IX\_IndexHandle \*indexHandle)

此函数打开名为fileName的索引文件。如果方法调用成功，则indexHandle为指向被打开的索引句柄的指针。索引句柄用于在索引中插入或删除索引项，也可用于索引的扫描。

RC CloseIndex(IX\_IndexHandle \*indexHandle)

此函数关闭句柄indexHandle对应的索引文件。

RC InsertEntry (IX\_IndexHandle \*indexHandle,void \*pData, const RID \*rid)

此函数向IX\_IndexHandle对应的索引中插入一个索引项。参数pData指向要插入的属性值，参数rid标识该索引项对应的元组，即向索引中插入一个值为（\*pData，rid）的键值对。

RC DeleteEntry (IX\_IndexHandle \*indexHandle,void \*pData, const RID \*rid)

此函数从IX\_IndexHandle句柄对应的索引中删除一个值为（\*pData，rid）的索引项。

RC OpenIndexScan(IX\_IndexScan \*indexScan, IX\_IndexHandle \*indexhandle, CompOp compOp, char \*value);

此函数用于在indexHandle对应的索引上初始化一个基于条件的扫描。compOp和\*value指定比较符和比较值，indexScan为初始化后的索引扫描结构指针。

RC IX\_GetNextEntry(IX\_IndexScan \*indexScan, RID \*rid);

此函数用于继续IX\_IndexScan句柄对应的索引扫描，获得下一个满足条件的索引项，并返回该索引 项对应的记录的ID。

RC CloseIndexScan(IX\_IndexScan \*indexScan);

关闭一个索引扫描，释放相应的资源。

## 模块实现

1. CreateIndex

首先创建一个文件，然后分配第一个页面，在页面上填充IX\_FIleHeader的信息，根据传入的数据分配阶，然后初始化各个常量，关闭索引返回SUCCESS。

2. OpenIndex CloseIndex

打开关闭索引只需要简单的读入索引控制信息的相关数据，这样就可以完成索引的打开。底层文件相关接口均由页面模块提供的接口进行实现。需要注意的是，此时打开文件最好不要从控制页拷贝到内存而是直接将索引句柄的相关指针指向内存，这样可以方便数据的更新。

3. InsertEntry

B+树插入算法：

1）若为空树，创建一个叶子结点，然后将记录插入其中，此时这个叶子结点也是根结点，插入操作结束。

2）针对叶子类型结点：根据key值找到叶子结点，向这个叶子结点插入记录。插入后，若当前结点key的个数小于等于m-1，则插入结束。否则将这个叶子结点分裂成左右两个叶子结点，左叶子结点包含前m/2个记录，右结点包含剩下的记录，将第m/2+1个记录的key进位到父结点中（父结点一定是索引类型结点），进位到父结点的key左孩子指针向左结点,右孩子指针向右结点。将当前结点的指针指向父结点，然后执行第3步。

3）针对索引类型结点：若当前结点key的个数小于等于m-1，则插入结束。否则，将这个索引类型结点分裂成两个索引结点，左索引结点包含前(m-1)/2个key，右结点包含m-(m-1)/2个key，将第m/2个key进位到父结点中，进位到父结点的key左孩子指向左结点, 进位到父结点的key右孩子指向右结点。将当前结点的指针指向父结点，然后重复第3步。

具体实现：

1. 首先找到B+树需要插入的值放入的节点位置，然后把值用插入到该节点的对应位置中；

2. 检查当前节点是否非法，非法条件为：关键字的数量大于这个B+树的order数目。如果这个节点合法的话，该算法结束，否则进入下一步。

3. 取当前非法节点的最中间的那个关键字，把这个关键字上提作为其父节点的一个关键字（位置满足当前排序），然后在这个关键字的左右两边分别添加当前非法节点的左半边和右半边作为添加关键字多出的两个子节点。这样就完成了一次节点的分裂。

4. 将父节点作为当前节点，算法跳转到第2步，递归地进行插入结点，直到算法结束或者不存在父节点。如果不存在父节点，将会把这个上提的关键字作为一个新的节点，这个节点就是新的根节点。

需要注意的是，对于HustBase来说每一次分裂之后，将会销毁原来那个非法节点，每一次分裂操作之后需要重新安排所有节点的子节点的兄弟、父节点等信息。特别是当分裂的节点具有子节点的时候，其左右两边所有的子节点都需要重新分配父节点。

DeleteEntry

B+树删除算法：

如果叶子结点中没有相应的key，则删除失败。否则执行下面的步骤

1）删除叶子结点中对应的key。删除后若结点的key的个数大于等于Math.ceil(m-1)/2 – 1，删除操作结束,否则执行第2步。

2）若兄弟结点key有富余（大于Math.ceil(m-1)/2 – 1），向兄弟结点借一个记录，同时用借到的key替换父结（指当前结点和兄弟结点共同的父结点）点中的key，删除结束。否则执行第3步。

3）若兄弟结点中没有富余的key,则当前结点和兄弟结点合并成一个新的叶子结点，并删除父结点中的key（父结点中的这个key两边的孩子指针就变成了一个指针，正好指向这个新的叶子结点），将当前结点指向父结点（必为索引结点），执行第4步（第4步以后的操作和B树就完全一样了，主要是为了更新索引结点）。

4）若索引结点的key的个数大于等于Math.ceil(m-1)/2 – 1，则删除操作结束。否则执行第5步

5）若兄弟结点有富余，父结点key下移，兄弟结点key上移，删除结束。否则执行第6步

6）当前结点和兄弟结点及父结点下移key合并成一个新的结点。将当前结点指向父结点，重复第4步。

注意，通过B+树的删除操作后，索引结点中存在的key，不一定在叶子结点中存在对应的记录。

OpenIndexScan / CloseIndexScan

对于索引的扫描操作，实际上就是简单的找到索引扫描的起始点，然后一直往下顺序遍历，直到出现了第一个未满足条件的关键字即可。所以对于打开扫描来说，将会根据符号分为下面两种情况：

1. 对于小于、小于等于号，起始点就是第一个关键字；

2. 对于其他情况，起始点是根据B+树获取关键字位置，然后根据符号的不同，获取符号的前面一位或者下一位作为扫描的起始位置。

关闭索引扫描就是简单的初始化操作，在此不再累述。

GetNextEntry

获取下一个索引操作，仅需要根据当前位置往下依次遍历即可。每获取到下一个索引需要判断这个索引是否满足条件，如果满足条件的话继续，否则将会返回一个错误值以告知外层调用者，该扫描已经结束。

## 实验过程

首先编写简单的CreateIndex、OpenIndex、CloseIndex函数，这些函数都是建立文件、初始化或者关闭文件等等操作，相对来说比较简单。

对于插入删除来讲，首先我在网上学习了一下B+树的插入方法，跟着网上的博客一步一步理解B+树插入删除的过程，然后就开始编写代码。编写完毕后开始调试，打着断点以及在纸上一点一点跟踪数据的运行过程。

# 实验总结

## 工作总结

本次实验我们对数据库的底层原理有了很深的理解，知道了数据库的构成大概分为底层页面管理模块、记录管理模块、索引模块、系统模块、查询模块、语法分析模块。结合理论课学到的数据库系统原理，我们手动实现了这个数据库。 其中出现的问题是返回值没有按照要求，以至于测评时候总是没有通过。还有就是修改了一开始的接口，导致测试时候总是崩溃。经过和同学的讨论，我们成功解决了这个问题，并在规定时间内完成任务。让我印象最深刻的是，我们的数据库系统总是出现内存错误这个问题，我们的解决方法是打断点，一步一步调试，观察变量的数值和预期是否一致。发现错误的地方是内存分配时，分配了一个很大的数，原来是顶层Condition传参时没有正确的传进去。经过我们的修改，内存错误这个问题终于解决，我们小组非常开心。

在这个实验里，我们学会了协作完成任务的方法，学习了如何使用github来进行分工合作和版本控制，每当一个人完成自己的一个函数时，都需要将自己的代码push到远程的仓库中，然后其他同学再pull下来，如果出现冲突还需要解决冲突。在本次实验中，我们认为B同学的任务量最大，所以在完成自己的任务时，我还帮助了B同学。

## 改进方案

经过我们小组一个月的任务完成，我们发现了框架的一些bug，并将bug都记录下来，希望后面的同学能够参考：

1. EditArea.cpp 里面有SYSTABLES.xx SYSCOLUMNS.xx 应该没有后缀xx

2. TableList.cpp 里面有SYSTABLES 里面应该没有后缀xx

3. 部分变量没有初始化，编译器没办法通过，建议增加=NULL（或者数值型类型为0）

4.TableList.cpp 里面系统表查询函数conditions需要增加

LAttrLength,condition.LattrLength = ParentNode.GetLength();

同时，我们编写代码时也犯了一些错误，也记录下来，以后可以查看：

1. 扫描起始点传入的参数错误，修改扫描起始点

2. checkConditions函数中，分配leftValue = new char[currentCon.LattrLength];修改为：

leftValue = new char[currentCon.LattrLength+1]; 这样字符串后面会增加\0

# 附录