

**系统能力综合培养实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学 院： | 计算机科学与技术学院 | | |
| 专 业： | 计算机科学与技术 | | |
| 班 级： | CS1904 | | |
| 学 号： |  | U201915052 |  |
| 姓 名： |  | 史宇辰 |  |
| 指导教师： | 左琼 | | |

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

2022年 11月 2日

目 录

[1 实验任务概述 1](#_Toc118730063)

[1.1 实验目的 1](#_Toc118730064)

[1.2 实验任务要求 1](#_Toc118730065)

[1.3 HustBase体系结构介绍 2](#_Toc118730066)

[1.4 小组成员分工 3](#_Toc118730067)

[2 记录管理模块 4](#_Toc118730068)

[3 系统管理模块 5](#_Toc118730069)

[4 查询处理模块 6](#_Toc118730070)

[4.1 模块概述 6](#_Toc118730071)

[4.2 模块设计 6](#_Toc118730072)

[4.3 模块实现 7](#_Toc118730073)

[4.4 实验过程 8](#_Toc118730074)

[5 索引管理模块 9](#_Toc118730075)

[5.1 模块概述 9](#_Toc118730076)

[5.2 模块设计 10](#_Toc118730077)

[5.2.1 索引文件的结构设计 10](#_Toc118730078)

[5.2.2 索引相关接口 11](#_Toc118730079)

[5.3 模块实现 13](#_Toc118730080)

[5.4 实验过程 16](#_Toc118730081)

[6 实验总结 17](#_Toc118730082)

[6.1 工作总结 17](#_Toc118730083)

[6.2 改进方案 17](#_Toc118730084)

[附录 17](#_Toc118730085)

# 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 实验任务要求

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/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，根据平时个人表现、互评情况，各人最终得分会有差异。

3) 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。

(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。

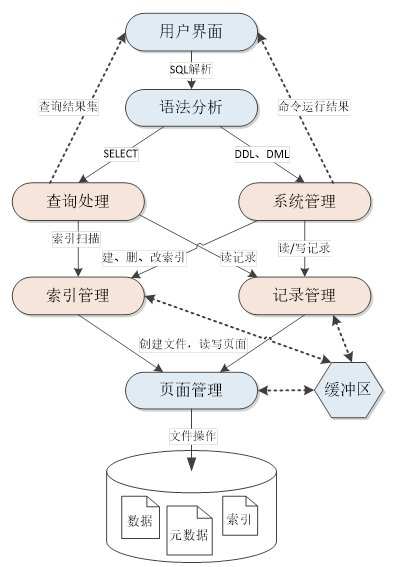


图 1.1 HustBase系统体系架构图

在图 1.1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

史宇辰：主要负责查询处理模块和索引管理模块的任务

高博文：主要负责系统管理模块和记录管理模块的任务

# 记录管理模块

此模块由本组的高博文实现

# 系统管理模块

此模块由本组的高博文实现

# 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，支持语法详见第9.1节。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，应尽可能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

本模块实现查询处理功能的基本过程为：首先对查询过程中涉及的表进行多表连接，然后根据各个条件选择符合条件的记录，最后通过投影操作选出所查询的属性名对应的列。

本模块对外提供的函数调用接口包括：

 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。

 RC query(char \*sql, SelResult \*res);

执行一条SELECT语句，查询结果集通过res参数返回。此函数是提供给测试程序专用的接口。

在查询过程中，使用到了记录管理模块中提供的接口，包括：

RC OpenScan(RM\_FileScan \*rmFileScan, RM\_FileHandle \*fileHandle, int conNum, Con \*conditions);

此接口用于打开一个文件扫描。本函数利用从第二个参数开始的所有输入参数初始化一个由参数rmFileScan指向的文件扫描结构，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。

RC GetNextRec (RM\_FileScan \*rmFileScan, RM\_Record \*rec);

此接口用于获取一个符合扫描条件的记录。如果该方法成功，返回值rec应包含记录副本及记录标识符。如果没有发现满足扫描条件的记录，则返回RM\_EOF。

RC CloseScan(RM\_FileScan \*rmFileScan);

此接口用于关闭一个文件扫描，释放相应的资源。

## 模块实现

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;

查询处理算法：

在查询处理过程中，首先对输入参数进行判断，不合法则返回错误信息。然后根据属性名参数判断是否为查询所有列，若是则将标记变量值置为1。接着遍历所有属性名，若属性名对应的表名缺失，则寻找属性名所在的表，并将表名加入，最后若对应表名缺失，返回错误信息；接着对每个条件的左属性和右属性进行同样操作。然后开始进行正式查询操作，首先将所有表进行连接，若只有一个表，则不需要连接；否则每次将需要连接的下一个表连接到已经连接好的表后，同时将属性名更改为“表名.属性名”的格式。然后根据各个条件对记录进行扫描，扫描到满足条件的记录就插入到结果集中。最后对是否查询所有列的标记变量进行判断，如果查询所有列，则直接返回结果；否则根据各个属性名进行投影操作在结果集中选取出所需的列，返回结果。

## 实验过程

在对各个表进行连接的过程中，考虑到各个表中可能有相同的属性名，所以在连接的同时需要修改每一个属性名的格式为“表名.属性名”以进行区分，所以在最开始需要对每一个属性结构变量补充其所在表的表名，以便后续操作的进行。

在实现查询处理模块的过程中，需要使用到记录管理模块中的函数接口对记录进行扫描，来获取符合查询条件的记录，但是查询处理过程中对条件的结构定义Condition和记录管理模块中对条件的结构定义Con不同，所以在进行扫描之前需要将Condition类型的数组conditions转换为Con类型的数组cons，再进行记录扫描。

# 索引管理模块

## 模块概述

本模块用于管理存储在索引文件中的索引信息，索引文件也是在页面文件的基础上改造的，它对上层模块提供以索引项为单位的文件读写接口。索引管理类似于记录管理，但不同于记录管理模块中的顺序存储记录的做法，索引管理采用B+树结构来组织索引项。利用B+树索引，查询时可以快速定位某条记录所在的位置，从而避免对数据表做全面扫描。

B+树是一种多路平衡树，树中的每个节点包含一组有序的关键字和指针。非叶子节点可包含的最大分支数称为该B+树的序数（order）。如果一个B+树的序数为m，则B+树应满足下列的性质：

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。

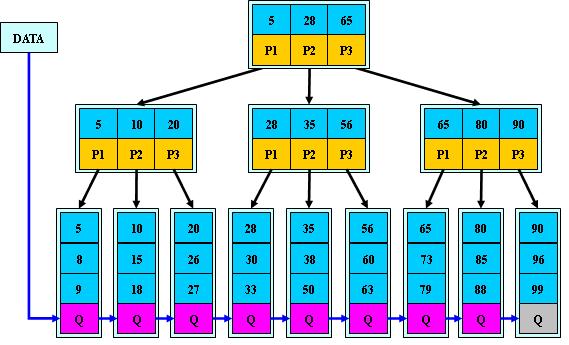


图 5.1 B+树示意图

图 5.1为一个序数为3的B+树示意图。

## 模块设计

本模块按照实验指导书给出的思路进行设计，主要包括索引文件的设计和索引相关接口的设计。

### 5.2.1 索引文件的结构设计

一个B+树索引存储为一个索引文件，文件由若干页面构成，每个索引节点占用一个页面。图 5.2为索引文件的设计方案。



图 5.2 索引文件结构示意图

类似于记录文件，索引文件的第0个页面也是存放页面控制信息。与记录文件不同的是，索引文件从第1页就开始存放索引数据。第1页与后续的索引页略有不同，该页从pData[0]开始，还存放了一个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。

紧跟在索引控制信息结构之后，从pData[sizeof(IX\_FileHeader)]开始，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;

紧接IX\_Node结构之后，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。

一个B+树节点占据一个页面，页面的大小固定，但是由于不同的索引关键字的大小不同，因此不同索引的B+树节点能够容纳的关键字的最大数量也就不同，即不同索引的B+树序数m不同。这是在索引创建时通过给定的属性值的大小来进行计算的。

计算公式为：

m=(PF\_PAGE\_SIZE-控制信息长度)/(2\*sizeof(RID)+atrrLength)

### 5.2.2 索引相关接口

本模块对外提供的函数调用接口包括：

 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。

 RC GetIndexTree(char \*fileName, Tree \*index);

获取由fileName指定的B+树索引内容，返回指向B+树的指针。此函数提供给测试程序调用，用于检查B+树索引内容的正确性。以下为相关数据结构定义：

typedef struct{

int keyNum; //本节点中包含的关键字（属性值）个数

char \*\*keys; //本节点中包含的关键字（属性值）数组

Tree\_Node \*parent; //父节点指针

Tree\_Node \*sibling; //右边的兄弟节点指针

Tree\_Node \*firstChild; //最左边的孩子节点指针

}Tree\_Node; //节点数据结构

typedef struct{

AttrType attrType; //B+树对应属性的数据类型

Int attrLength; //B+树对应属性值的长度

int order; //B+树的序数

Tree\_Node \*root; //B+树的根节点

}Tree;

本模块在实现过程中调用了页面管理模块中的多种接口函数，例如：

RC CreateFile (const char \*fileName);

创建一个名称为指定文件名的分页文件。

 RC OpenFile(char \*fileName, int \* FileID);

根据文件名打开一个分页文件，返回文件ID。

 RC CloseFile (int FileID);

关闭FileID对应的分页文件。

RC GetThisPage (int FileID, PageNum pageNum, PF\_PageHandle \*pageHandle);

根据文件ID和页号获取指定页面到缓冲区，返回页面句柄指针。

 RC AllocatePage (int FileID, PF\_PageHandle \*pageHandle);

在指定文件中分配一个新的页面，并将其放入缓冲区，返回页面句柄指针。分配页面时，如果文件中有空闲页，就直接分配一个空闲页；如果文件中没有空闲页，则扩展文件规模来增加新的空闲页。

 RC MarkDirty(PF\_PageHandle \*pageHandle);

标记指定页面为“脏”页。如果修改了页面的内容，则应调用此函数，以便该页面被淘汰出缓冲区时系统将新的页面数据写入磁盘文件。

 RC UnpinPage(PF\_PageHandle \*pageHandle);

此函数用于解除pageHandle对应页面的驻留缓冲区限制。在调用GetThisPage或AllocatePage函数将一个页面读入缓冲区后，该页面被设置为驻留缓冲区状态，以防止其在处理过程中被置换出去，因此在该页面使用完之后应调用此函数解除该限制，使得该页面此后可以正常地被淘汰出缓冲区。

## 模块实现

模块实现过程中主要用到了以下几个数据结构：

索引句柄：

typedef struct {

bool bOpen; //该索引句柄是否已经与一个文件关联

int fileID; //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;

索引控制信息：

typedef struct {

int attrLength; //建立索引的属性值的长度

int keyLength;

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;

节点控制信息：

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;

索引扫描：

typedef struct {

bool bOpen; //扫描是否打开

IX\_IndexHandle\* pIXIndexHandle; //指向索引文件句柄的指针

CompOp compOp; //用于比较的操作符

char\* value; //与属性值进行比较的值

PF\_PageHandle PageHandle; //处理中的页面句柄

PageNum pn; //扫描即将处理的页面号

int ridIx; //扫描即将处理的索引项编号

}IX\_IndexScan;

节点数据结构：

typedef struct Tree\_Node{

int keyNum; //节点中包含的关键字（属性值）个数

char \*\*keys; //节点中包含的关键字（属性值）数组

Tree\_Node \*parent; //父节点

Tree\_Node \*sibling; //右边的兄弟节点

Tree\_Node \*firstChild; //最左边的孩子节点

}Tree\_Node; //节点数据结构

B+树数据结构：

typedef struct{

AttrType attrType; //B+树对应属性的数据类型

int attrLength; //B+树对应属性值的长度

int order; //B+树的序数

Tree\_Node \*root; //B+树的根节点

}Tree;

实现索引管理模块的各个接口的算法如下:

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

首先根据文件名创建对应页面文件，然后打开创建的文件，获取fileID，再根据fileID分配第一页，并在该页加入索引相关控制信息，最后调用页面处理函数处理，关闭文件。

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

根据文件名打开文件，得到fileID，根据fileID获取到对应页面的第一页，获取到索引控制信息，并进行相应的修改。

RC CloseIndex(IX\_IndexHandle \*indexHandle);

将索引句柄的bOpen设置为false，然后关闭fileID对应的文件。

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

首先在B+树中查找到可能进行插入操作的叶子节点，获取到该叶子节点所在页面的页面句柄，同时得到该页面的页号，然后根据插入属性值和页号插入到对应的叶子节点中。首先，查找到要插入其中的叶子节点的位置，接着把值插入到这个节点当中。如果节点处于合法状态，则处理结束。如果该节点有过多的元素，即元素的个数达到了m个，则把它平均分裂为两个节点，每个节点都有至少为最小数目个元素。如果是叶子节点，则将分裂后的后面一颗树的第一个元素插入到它的父节点当中，否则将中间元素插入到它的父节点当中。再递归的处理父节点的插入，直到到达根节点，如果根节点被分裂，则创建一个新根节点。

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

首先根据属性值和RID查找需要删除的索引项所在的页面，获取到对应的页面句柄，同时得到该页面的页号，然后根据删除属性值和页号对指定节点进行删除操作。首先，找到要删除的值，接着从包含它的节点中删除这个值。如果没有节点处于非法状态则处理结束。如果节点处于非法状态，则有两种可能情况：

(1) 它的兄弟节点有多余子节点。此时可以把兄弟节点的一个或是多个子节点转移到当前节点，从而把它变为合法状态。这种情况下，在更改父节点和两个兄弟节点的分离值之后处理结束。

(2) 它的兄弟节点由于处在最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，并递归处理父节点（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是到达根节点，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。

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

首先根据属性值在B+数中进行查找，找到可能所在的叶子节点，得到所在页面的页面句柄，从该页面中获取到节点控制信息，然后根据运算符进行判断，找到开始处理的页面，对索引扫描进行初始化。

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

先从索引扫描中获取到页面句柄，再从页面句柄中获取节点控制信息，然后先对节点中的索引项进行判断，如果扫描到最后一个索引项就切换到下一页面，如果扫描到最后一个叶子节点的最后一个索引项就结束扫描。扫描过程中根据操作符对结果进行处理。

RC CloseIndexScan(IX\_IndexScan \*indexScan);

将索引扫描的bOpen值置为false，解除扫描的页面句柄的固定。

RC GetIndexTree(char \*fileName, Tree \*index);

根据文件名打开指定索引，对index中的attrType，attrLength，order进行修改，修改为打开的索引中对应的值。然后从打开的索引中获取到指向根节点的指针，赋值给index中的root。

## 实验过程

在进行索引B+树的插入操作的过程中，插入新的内容时若节点已经存放有m个元素，则需要进行分裂，分裂为两个节点，先创建一个新节点，将旧节点的后半部分内容存入新节点，但同时需要考虑新的内容应该插入到哪个节点当中，因此需要事先找到新节点在原节点中的顺序，如果在m/2前，则插入到旧节点中，否则插入新节点中，最后将新节点的首个项插入到父节点中，开始对父节点进行递归判断。

在进行索引B+数的删除操作的过程中，删除节点内容后若节点存放内容少于m/2，则需要进行处理。若兄弟节点可以容纳下该节点剩余内容，则将两个节点合并，然后递归处理父节点的删除操作；若兄弟节点有多余的子节点，则将兄弟节点的一个或是多个子节点转移到当前节点，从而把它变为合法状态。这种情况下，在更改父节点和两个兄弟节点的分离值之后处理结束。

# 实验总结

## 工作总结

主要工作：

1. 实现了基础的单表查询与多表查询功能，完成了查询处理模块的编写
2. 与系统管理模块联调，测试了查询处理功能的正确性
3. 完成了索引管理模块的编写，实现了该模块对外提供的各个接口的功能
4. 与系统管理模块联调，测试了索引相关基本操作的正确性

心得体会：

在这次课程设计中，我们组依照实验任务书的要求完成了基于HuseBase系统的单用户关系数据库系统，我主要负责其中查询处理模块和索引管理模块的实现。其中我不仅学习到了数据库中查询处理功能的大致流程，也学习到了索引B+树的相关知识，对数据库相关知识也有了进一步的理解与掌握。这次课程设计的内容也是对上学期所学习的数据库基本知识的进一步延伸与对这些知识的实际运用能力的一种考察。同时这次课设不仅考验了个人的编程能力，也考验了团队配合能力，在小组中各个成员实现自己负责的模块功能的同时，也需要使用到其他团队成员所设计的模块功能，因此需要团队成员之间积极的沟通与配合，才能成功完成各个模块的对接与整个数据库系统的联调，这个过程也是对今后正式面对工作项目时的一种简单模拟，只有各个组员分工协调，配合默契，才能有条不紊地合作完成复杂的项目。

## 改进方案

目前在我们实现的HustBase数据库系统中，在查询处理的过程中还没有进行查询优化处理，可以通过一些代数优化手段，例如先进行符合条件的记录的筛选，再进行所需列的投影操作，最后再进行连接操作，这样可以减少连接操作中笛卡尔积运算的规模，从而提高查询过程的效率。还可以通过使用索引来扫描记录更进一步地提高查询过程的效率。

# 附录