

**《数据库系统原理》**

**课程设计指导书**

# 数据库课程组

目 录

[数据库课程组 1](#_Toc386443189)

[1. 课程设计目的 4](#_Toc386443190)

[2. 课程设计要求 4](#_Toc386443191)

[3. HustBase体系结构介绍 5](#_Toc386443192)

[4. 页面管理模块 7](#_Toc386443193)

[4.1 页面管理模块说明 7](#_Toc386443194)

[4.1.1 分页文件 7](#_Toc386443195)

[4.1.2 页面缓冲区 8](#_Toc386443196)

[4.2 页面管理模块对外提供的函数调用接口 9](#_Toc386443197)

[4.2.1 文件操作函数 9](#_Toc386443198)

[4.2.2 页面操作函数 9](#_Toc386443199)

[5. 记录管理模块 10](#_Toc386443200)

[5.1 记录管理模块说明 10](#_Toc386443201)

[5.1.1 控制页 10](#_Toc386443202)

[5.1.2 数据页 11](#_Toc386443203)

[5.1.3 记录操作 11](#_Toc386443204)

[5.2 记录管理模块对外提供的函数调用接口 13](#_Toc386443205)

[5.2.1 记录文件管理函数 13](#_Toc386443206)

[5.2.2 记录操作函数 13](#_Toc386443207)

[5.2.3 文件浏览函数 14](#_Toc386443208)

[6. 索引管理模块 14](#_Toc386443209)

[6.1 索引管理模块说明 14](#_Toc386443210)

[6.1.1 索引文件 15](#_Toc386443211)

[6.1.2 B+树 15](#_Toc386443212)

[6.1.3 B+树索引的操作 16](#_Toc386443213)

[6.2 索引管理模块对外提供的函数调用接口 19](#_Toc386443214)

[6.2.1 索引文件管理函数 19](#_Toc386443215)

[6.2.2 索引操作函数 19](#_Toc386443216)

[7. 系统管理模块 20](#_Toc386443217)

[7.1 系统管理模块说明 20](#_Toc386443218)

[7.2 系统管理模块对外提供的函数调用接口 21](#_Toc386443219)

[8. 查询处理模块 23](#_Toc386443220)

[8.1 查询处理模块说明 23](#_Toc386443221)

[8.2 查询处理模块对外提供的函数调用接口 23](#_Toc386443222)

[9. 语法分析模块 24](#_Toc386443223)

[9.1 HustBase支持的SQL语句 24](#_Toc386443224)

[9.2 语法分析流程 25](#_Toc386443225)

[9.3 SQL语句对应的数据结构 25](#_Toc386443226)

[9.4 语法分析模块对外提供的函数调用接口 29](#_Toc386443227)

# 课程设计目的

1. 了解关系数据库管理软件的体系结构和设计方法；
2. 了解关系数据库中数据和元数据的组织方法；
3. 掌握关系数据的底层存储实现技术；
4. 掌握基本关系操作在关系数据上的实现算法；
5. 掌握基本SQL语句的实现方法；
6. 掌握基本的查询优化技术。

# 课程设计要求

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它模块功能，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。

课程设计课时安排为四周，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、页面管理和记录管理模块，要求学生自己设计实现索引管理、系统管理和查询处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 数据库课程设计课程内容安排

|  |  |
| --- | --- |
| **周 次** | **内 容** |
| 第一周 | 学习HustBase的系统结构和工作原理，在给定模块的基础上，完成系统管理模块的编码和调试。 |
| 第二周 | 查询处理模块的编码和调试，联调完成基本的数据管理功能。 |
| 第三周 | 完成索引管理模块的编码和调试。 |
| 第四周 | 利用索引进行查询优化，完成系统整体联调。 |

HustBase系统的开发环境为Windows 操作系统及 Visual C++，开发语言为C语言。为保证良好的模块独立性，除用户界面模块外，其它模块均采用链接库的形式实现。每个模块必须按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. 课程设计内容采取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，基本原则是“共同设计，分工编码”。
2. 考核方式为公开演示及答辩，最终成绩根据系统完成情况、答辩情况和课程设计报告质量综合评分，同组学生的成绩相同。
3. 课程结束后，以小组为单位提交系统的源码、编译后的程序及课程设计报告。

# HustBase体系结构介绍

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、查询处理和语法分析等主要模块构成。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。本模块提供的功能包括：创建、销毁、打开和关闭分页文件；扫描指定文件的所有页面；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：数据表中的数据以记录为单位存取。本模块包括一组函数，用于管理存储在一组文件中的数据表记录。记录管理模块依赖于底层的页面管理模块，页面管理模块以页面为单位实现文件级别的I/O，而记录管理模块则以记录为单位实现数据表级的读写。
  3. 索引管理：系统通过索引为查询提供快速访问路径。本模块用于管理存储在一组文件中的索引记录。与数据表不同的是，索引采用B+树结构来组织索引记录。索引管理模块同样依赖于底层的页面管理模块。
  4. 系统管理：本模块用于数据定义（建立数据表和索引）和数据操纵（插入、删除、修改记录）功能。系统管理模块的功能依赖于记录管理模块和索引管理模块。
  5. 查询处理：HustBase支持基本的SQL语句，包括简单的数据表定义（CREATE TABLE）、数据操纵（INSERT、DELETE、UPDATE）和数据查询（SELECT）语句。查询处理模块用于执行用户的SELECT命令，并向客户端返回最终的查询结果集。本模块的功能依赖于记录管理模块、索引管理模块和系统管理模块。



图3.1 HustBase系统体系架构图

* 1. 语法分析：语法分析模块提供对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。
  2. 客户端：一个图形化交互式工具，用于查看数据库模式信息、编辑SQL语句、并显示SQL语句执行结果。

# 页面管理模块

## 页面管理模块说明

### 分页文件

一个分页文件由若干个页（Page）构成，申请空间、读数据、写数据，均以页为单位。每页大小为4K字节。其中，前4个字节存放页号，后4092字节存放数据。对应数据结构为：

typedef struct {

PageNum pageNum; //页号(由0开始编号)

char pData[PF\_PAGE\_SIZE]; //数据

} Page;

每个文件的第0页为控制页，该页的数据区存放的信息为：

从data[0]开始，存放一个数据结构：

typedef struct {

PageNum pageCount; //尾页的页号

int nAllocatedPages; //已分配页的数目(包括控制页)

} PF\_FileSubHeader;

从data[PF\_FILESUBHDR\_SIZE]（PF\_FILESUBHDR\_SIZE = sizeof(PF\_FileSubHeader)）开始，存放一个位图。位图中的每一位对应着一个页面的分配情况：0表示该页为空闲页，1表示该页已分配。显然，每个分页文件的最大长度受位图大小的限制。

控制页在创建文件（CreateFile）时生成并初始化，之后每次申请新页面、释放页面、分配空闲页时，PF\_FileSubHeader结构和位图中的对应位都要做相应的修改。

使用一个分页文件前，需要先打开文件（OpenFile），获取一个文件句柄，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（CloseFile）释放资源。文件句柄的结构为：

typedef struct {

bool bOpen; //该文件句柄是否已经与一个文件关联

char \*fileName; //与该句柄关联的文件名

int fileDesc; //与该句柄关联的文件描述符

Frame \*pHdrFrame; //指向该文件头帧（控制页对应的帧）的指针

Page \*pHdrPage; //指向该文件头页（控制页）的指针

char \*pBitmap; //指向控制页中位图的指针

PF\_FileSubHeader fileSubHeader; //该文件的PF\_FileSubHeader结构

} PF\_FileHandle;

### 页面缓冲区

为提高页面访问效率，页面管理模块维护一个页面缓冲区。该缓冲区由以下数据结构表示：

typedef struct {

int nReads; //I/O统计已读

int nWrites; //I/O统计已写

Frame frame[PF\_BUFFER\_SIZE]; //缓冲池是一组帧页

} BF\_Manager;

缓冲池大小为PF\_BUFFER\_SIZE个帧（Frame），对应数据结构为：

typedef struct {

bool bDirty; //True 表示页面内容已经被修改，False表示没有被修改

unsigned int pinCount; //访问当前页面的线程数

time\_t accTime; //页面最后访问时间

char \*fileName; //访问页面所在的文件名

int fileDesc; //与该文件相关联的文件描述符

Page page; //从硬盘中获取的实际页面

} Frame;

每次从文件中读一个页面时，从页面缓存区为该页分配一个帧（采用LRU算法），将该页的内容读到帧中。每个页面对应一个页面句柄，作为该页面的标识。页面句柄的数据结构为：

typedef struct {

bool bOpen; //True 表示页面句柄被打开

Frame \*pFrame; //指向当前对象句柄的指针

}PF\_PageHandle;

## 页面管理模块对外提供的函数调用接口

### 文件操作函数

/\*创建一个分页文件\*/

RC CreateFile (const char \*fileName);

/\*打开一个分页文件，返回它的句柄指针\*/

RC openFile(char \*fileName, PF\_FileHandle \*fileHandle);

/\*关闭给定句柄的分页文件\*/

RC CloseFile (PF\_FileHandle \*fileHandle);

### 页面操作函数

/\*获取指定页面，返回其句柄指针\*/

RC GetThisPage (PF\_FileHandle \*fileHandle, PageNum pageNum, PF\_PageHandle \*pageHandle);

/\*分配一个新的页面，返回它的句柄（如果文件中有空闲页，就直接分配一个空闲页，否则文件扩展一个页面）\*/

RC AllocatePage (PF\_FileHandle \*fileHandle, PF\_PageHandle \*pageHandle);

/\*返回指定页面对应的页面号\*/

RC GetPageNum (PF\_PageHandle \*pageHandle, PageNum \*pageNum);

/\*返回指定页面对应的数据区指针\*/

RC GetData (PF\_PageHandle \*pageHandle, char \*\*pData);

/\*丢弃文件中编号为pageNum的页面，将其变为空闲页面\*/

RC DisposePage (PF\_FileHandle \*fileHandle, PageNum pageNum);

/\*标记页面为"脏"\*/

RC MarkDirty(PF\_PageHandle \*pageHandle);

/\*编号为pageNum的页面不再需要被驻留在内存中\*/

RC UnpinPage(PF\_PageHandle \*pageHandle);

# 记录管理模块

## 记录管理模块说明

数据表中的数据以记录为单位存取。本模块包括一组函数，用于管理存储在一组文件中的数据表记录。记录管理模块依赖于底层的页面管理模块，页面管理模块以页面为单位实现文件级别的I/O，而记录管理模块则以记录为单位实现数据表级的读写，记录根据页面号和插槽号以顺序方式存取。为简化实现，规定文件中存储的记录为定长，且记录格式相同。即：一个数据表对应一个文件。

### 控制页

每个文件包含若干页面，其中第0页为分页信息控制页，具体内容见4.1节。文件的第1页为记录信息控制页，从data[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; //一个分页可以装载的记录数

int firstRecordOffset; //第一个记录的偏移值

} RM\_FileSubHeader;

从pData[sizeof(RM\_FileSubHeader)]开始的区域是一个位图，用来标记记录文件中哪些页面为满页哪些页面为非满页，其中相应位为0表示非满页，为1表示满页。它和分页文件第0页中的位图非常相似，但是是在不同的层次上，所表示的意思也不一样。

### 数据页

从分页文件的第2页开始为数据页面，记录在数据页面中采用顺序存储方式。在一个记录文件中，通过（页面号，插槽号）可以唯一确定一个记录。记录ID的数据结构如下：

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;

在每个数据页面中，从data[0]开始存放一个位图，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 (n+7)/8 字节的位图。当某个插槽含有有效的记录，位图中相应的位为1；当某个插槽为空插槽时，相应的位为0。注意此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记录大小计算此位图所需的字节数。

### 记录操作

记录对应的数据结构为：

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;

1. 插入记录

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，要更新控制页的页头，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。如果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。

1. 删除记录

删除记录时，给定一个记录ID，通过页面号和插槽号找到相应的记录，在查找记录之前要检查该记录ID的有效性，先判断页面号所对应的页面是否已经分配，以及该页面的给定插槽号所对应的记录插槽已经写入了记录。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，并把该页面标记为非满页。同时将控制页头中的记录个数减1。

1. 记录扫描

记录扫描为用户提供基于特定条件的扫描记录文件中记录的功能，在扫描期间，只有那些满足条件的记录将被获取。

记录扫描的数据结构：

typedef struct{

bool bOpen; //扫描是否打开

RM\_FileHandle \*pRMFileHandle; //指向记录文件操作的指针

int conNum; //涉及的条件数量

Con \*conditions; //涉及的条件

ClientHint pinHint; //用于记录管理组件指定页面的固定策略

int N; // 固定在缓冲区中的页，与指定的页面固定策略有关

int pinnedPageCount; // 实际固定在缓冲区的页面数

PF\_PageHandle pfPageHandles[PF\_BUFFER\_SIZE]; // 固定在缓冲区页面所对应的页面操作列表

int phIx; //当前被扫描页面的操作索引

SlotNum snIx; //下一个被扫描的插槽的插槽号

PageNum pnLast; //上一个固定页面的页面号

}RM\_FileScan;

涉及条件的数据结构：

typedef struct

{

int bLhsIsAttr,bRhsIsAttr;//标明条件的左、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，标明属性的长度

int LattrOffset,RattrOffset; //若是属性的话，标明属性的偏移量

CompOp compOp; //用于比较的操作符

void \*Lvalue,\*Rvalue; //若是值的话，保存该值

}Con;

其中attrType和LattrLength/RattrLength分别表示所要进行比较的属性的类型和长度。当属性的类型为整型或浮点型时，LattrLength/RattrLength固定为4。当属性的类型为字符串类型时，LattrLength/RattrLength表示的是字符串的长度。LattrOffset/RattrOffset表示所指定的属性在记录中的偏移量，通过它就能给在记录中找到所要比较的属性值。

比较操作符的数据结构为：

typedef enum {

EQual, LessT, GreatT, // 等于，小于，大于

NEqual, LEqual, GEqual, // 不等于，小于等于，大于等于

NO\_OP // 无比较

} CompOp;

## 记录管理模块对外提供的函数调用接口

### 记录文件管理函数

/\*创建一个记录文件，该文件中每条记录的大小为recordSize \*/

RC RM\_CreateFile (char \*fileName, int recordSize);

/\*打开文件，返回它的句柄指针 \*/

RC RM\_OpenFile (char \*fileName, RM\_FileHandle \*fileHandle);

/\*关闭给定句柄的文件 \*/

RC RM\_CloseFile (RM\_FileHandle \*fileHandle);

### 记录操作函数

/\*检索文件中标识符为rid的记录\*/

RC GetRec (RM\_FileHandle \*fileHandle, RID \*rid, RM\_Record \*rec);

/\*插入一个新的记录到fileHandle指向的数据文件中，pData指向新纪录的内容，并返回该记录的标识符rid\*/

RC InsertRec (RM\_FileHandle \*fileHandle, char \*pData, RID \*rid);

/\*从文件中删除标识符为rid的记录\*/

RC DeleteRec (RM\_FileHandle \*fileHandle, RID \*rid);

/\*更新与rec有关联的文件中的记录内容\*/

RC UpdateRec (RM\_FileHandle \*fileHandle, RM\_Record \*rec);

### 文件浏览函数

/\*初始化文件扫描。该方法初始化由文件句柄引用的打开文件中记录的扫描。扫描中，仅满足所有指定条件的记录被检索。如果条件的数量为0，条件是一个空指针，所有记录被检索。如果条件不为空，则要进行全表扫描并比较，满足所有条件的记录被检索。\*/

RC OpenScan(RM\_FileScan \*rmFileScan,

RM\_FileHandle \*fileHandle,

int conNum,

Con \*conditions,

ClientHint pinHint);

/\*获得下一个匹配的记录副本。该方法检索满足文件扫描的扫描条件的下一个记录的副本。如果该方法成功，返回值rec应包含记录副本及记录标识符。如果没有满足扫描条件的记录，返回RM\_EOF。\*/

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

# 索引管理模块

## 索引管理模块说明

本模块用于管理存储在一组文件中的索引记录。索引管理类似于记录管理，但不同于记录管理中的顺序存储数据表记录，索引管理采用B+树结构来组织索引记录。索引管理模块同样依赖于底层的页面管理模块。

### 索引文件

一个B+树索引是一个索引文件，由若干页面构成，索引节点的大小固定，每个页面存储一定的索引节点。此部分类似于记录管理。

索引文件的第1个页面存放控制信息，从data[0]开始，存放一个数据结构：

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

int order; //B+树的阶数

}IX\_FileHeader;

索引文件的文件头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。关于keyLength的解释，具体见6.1.3节，“非唯一索引的重复键值处理”。

### B+树

B+树是一种多路平衡树，在B+树中的节点通常被表示为一组有序的元素和子指针。如果B+树的序数（order）是m，则B+树要满足下列的性质：

1. 根结点只有1个，关键字个数的范围为[1,m-1]，分支数量范围[2,m]；
2. 除根以外的内部结点，每个结点包含分支数范围为[[m/2],m]，即关键字字数的范围是[[m/2]-1,m-1]，其中[m/2]表示取大于等于m/2的最小整数；
3. 所有的叶子结点中包含了全部[关键字](http://baike.baidu.com.cn/view/390935.htm)的信息，及指向含这些关键字记录的[指针](http://baike.baidu.com.cn/view/159417.htm)，且叶子结点本身依关键字的大小自小而大顺序链接。并且所有的叶子节点都在同一层。
4. 叶子节点包含的关键字个数范围为[[m/2],m-1]。

B+树中节点的数据结构为：

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

char \*keys; //指向关键字的指针

RID \*rids; //指向记录标识符的指针

}IX\_Node;

B+树中的每个节点都是存储在分页文件中的一页上，这样每个页面号就能够唯一的标志一个B+树节点。由于在索引文件中并没有单独的分配一个页面用来存放索引头文件，而是把它放在第1页的数据区的开始部分，这样每个B+树节点的有效数据都是从页面pData加上sizeof(IX\_FileHeader)大小的偏移开始的。

其中关键字是紧跟着IX\_Node这个结构体存放的，而指针则是紧接着关键字区域存放的。B+树的节点每次从磁盘读到页面缓冲区的位置不一定相同，因此每次读入所需要节点的时候都要重新给keys和rids 赋值，使它们指向正确的位置。

每个B+树节点都占据一个页面，页面的大小固定为4K，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+树节点能够容纳的最大的关键字个数不同，即不同索引的B+树阶数不同。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-sizeof(IX\_FileHeader)-sizeof(IX\_Node))/(2\*sizeof(RID)+atrrLength)

其中，sizeof(RID)乘以2的原因是本文通过在属性值中加上记录的标志来使得每个关键字都唯一。

### B+树索引的操作

1. **B+树的查找**

对B+树可以进行两种查找运算：

1. 从最小关键字起顺序查找；
2. 从根节点开始，进行随机查找。

在查找时，如内部节点上的关键字等于给定值，查找并不终止，而是继续向下直到叶子节点。这是因为内部节点并不包含指向记录的指针。因此，在B+树中，不管查找成功与否，每次查找都是走了一条从根到叶子节点的路径。

1. **B+树的插入**

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。如果结点处于合法状态，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，每个结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，否则则将中间元素插入到它的父节点当中。再递归的处理父结点的插入，直到到达根节点，如果根节点被分裂，则创建一个新根节点。

1. **B+树的删除**

首先，查找到要删除的值，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。
3. **非唯一索引的重复键值处理**

由于建索引的列不一定是唯一列，因此可能出现多个元组的索引列值相同的问题。解决方法为：将索引列的值和记录的ID值组合成索引关键字，由于记录ID是唯一的，因此组合而成的关键字一定是唯一的。因为上述原因，索引关键字的长度keyLength为索引列值的长度+记录ID的长度。关键字比较过程中，先比较属性值，然后在属性值相同的情况下比较记录ID。这样就使得索引在总体上按关键字从小到大排列，在关键字相同的情况下，按记录ID的大小从左向右排。

1. **索引扫描**

索引扫描是基于条件来扫描一个索引中的索引项，它是对B+树查找功能的进一步加强。

索引扫描的数据结构：

typedef struct{

bool bOpen; //扫描是否打开

IX\_IndexHandle \*pIXIndexHandle; //指向索引操作的指针

CompOp compOp; //用于比较的操作符

char \*value; //与属性值进行比较的值

ClientHint pinHint; //用于记录管理组件指定页面的固定策略

int N; // 固定在缓冲区中的页，与指定的页面固定策略有关

int pinnedPageCount; // 实际固定在缓冲区的页面数

PF\_PageHandle pfPageHandles[PF\_BUFFER\_SIZE]; // 固定在缓冲区页面所对应的页面句柄数组

int phIx; //当前被扫描页面的页面句柄索引

int ridIx;

PageNum pnNext; //下一个将被写入缓冲的页面号

}IX\_IndexScan;

pinHint，pinnedPageCount和phIx与记录文件扫描的意义是一致的。其中需要说明的是ridIx和pnNext。

由于比较操作符只有六种：大于，大于等于，小于，小于等于，等于，以及无比较操作。这里的无比较操作具有其特殊的意义，就是可以将记录按照某个属性值的从小到大的顺序扫描到。

通过分析所有叶子节点关键字的排列顺序特性可知，找到第一个符合条件的索引项后，就可以顺序扫描，直到遇到第一个不符合条件的索引项就可以停止扫描，比如小于比较符，如果扫描到一个索引项的属性值大于或等于所给定的属性值，则这个索引项后面的索引项的属性值都大于等于这个给定的属性值。

因此对于索引扫描初始化时，最重要的工作就是初始化ridIx和pnNext的值，对于小于和小于等于比较操作符，ridIx为0，pnNext为第一个叶子节点所在的页面（在这里的实现一直会是第1个页面）。对于等于号，通过B+树查找操作找到叶子节点，然后顺序比较属性值，如果找到相等的就将ridIx以及pnNext置为正确的值，如果找到了一个大于的值，则返回错误。对于大于和大于等于操作也是相同的原理。

## 索引管理模块对外提供的函数调用接口

### 索引文件管理函数

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

此方法创建一个名为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)

此方法打开名为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。

### 索引操作函数

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

此方法向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)

此方法从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, ClientHint pinHint=NO\_HINT);

此方法用于在indexHandle指向的一个已打开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为指向扫描返回的结果的指针。

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

此方法用于从IX\_IndexScan句柄对应的索引扫描中获得下一个由参数RID标识的索引。

# 系统管理模块

## 系统管理模块说明

系统管理模块负责数据定义（建立数据表和索引）和数据操纵（插入、删除、修改记录），其功能依赖于记录管理模块和索引管理模块。

一个数据库由包含在指定目录中的数据库文件构成。这些数据库包括系统表文件、系统列文件、若干记录文件和若干索引文件。模块的主要数据结构如下：

typedef struct \_ AttrInfo AttrInfo;

struct \_AttrInfo {

char \*attrName; // 属性的名字

AttrType attrType; // 属性的类型

int attrLength; // 属性的长度

};

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

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 {

char \*relName; //表名，可能为NULL

char \*attrName; //属性名

}RelAttr;

typedef struct \_Value Value;

struct \_Value{

AttrType type; // 值的类型

Void \*data; // 值

};

系统表文件中记录的结构如下：

|  |  |
| --- | --- |
| 表名 tablename | 该表中属性的数量 attrcount |

表名占21个字节，即表名的最大长度为20个字节。表中属性的数量占用sizeof(int)共4个字节。

系统列文件中记录的结构如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| tablename | attrname | attrtype | attrlength | attroffset | ix\_flag | indexname |

表名(tablename)与属性名(attrname)各占21个字节，即表名与属性名的最大长度均为20个字节。属性的类型(attrtype)占sizeof(AttrType)共4个字节。属性的长度(attrlength)与偏移量(attroffset)均占sizeof(int)共4个字节。该属性列上是否存在索引的标识(ix\_flag)占1个字节，’1’表示存在索引，’0’表示不存在索引。索引的名称(indexname)占21个字节，即索引名的最大长度为20个字节。

## 系统管理模块对外提供的函数调用接口

* RC CreateDb (char \*dbPath ,char \*dbName)

该函数在dbPath的路径下创建一个名为dbName的空库，生成相应的数据库文件。

* RC DropDb (char \*dbName)

该函数删除名为dbName的目录下的数据库文件。

* RC OpenDb (char \*dbName)

该函数改变当前目录名为dbName，然后打开目录包含的数据库文件。

* RC CloseDb ()

该函数关闭所有在当前数据库中打开的文件。关闭文件操作将自动使所有相关的缓冲区更新到磁盘。

* RC CreateTable (char \*relName, int attrCount, AttrInfo \*attributes)

此函数创建一个新的名为relName的表。参数attrCount表示关系中属性的数量（取值为1到MAXATTRS之间）。参数attributes是一个长度为attrCount的数组。对于新关系中第i个属性，attributes数组中的第i个元素包含名称、类型和属性的长度（见AttrInfo结构定义）。

* RC DropTable (char \*relName)

该函数用来销毁名为relName的表和所有在其上建立的索引。

* RC CreateIndex (char \*indexName,char \*relName, char \*attrName)

该函数在关系relName的属性attrName上创建索引，索引名为indexName，并建立关系当前内容的索引。函数首先检查在标记属性上是否已经存在一个索引，如果存在，则返回一个非零的错误码。否则，创建该索引。创建索引的工作包括：①打开索引；②扫描被索引的记录，插入索引项；③关闭索引。

* RC DropIndex (char \*indexName)

该函数用来删除名为indexName的索引。函数首先检查索引是否存在，如果不存在，则返回一个非零的错误码。否则，销毁该索引。

* RC Insert (char \*relName, int nValues, Value \*values)

该函数用来在relName表中插入具有指定属性值的新元组。函数根据给定参数构建一条元组，调用记录管理模块的函数插入该元组，然后在该表的每个索引中为该元组创建合适的索引项。

* RC Delete (char \*relName, int nConditions, Condition \*conditions)

该函数用来删除relName表中所有满足指定条件的元组以及该元组对应的索引项。如果没有指定条件，则此方法删除relName关系中所有元组。如果包含多个条件，则这些条件之间为与关系。

* RC Update (char \*relName, char \*attrName, Value \*Value, int nConditions, Condition \*conditions);

该函数用于更新relName表中所有满足指定条件的元组，在每一个更新的元组中将属性attrName的值设置为一个新的值。如果没有指定条件，则此方法更新relName中所有元组。如果要更新一个被索引的属性，应当先删除每个被更新元组对应的索引条目，然后插入一个新的索引条目。

# 查询处理模块

## 查询处理模块说明

查询处理模块负责select语句的处理。Select语句的语法详见第9.1节。本模块涉及的主要数据结构包括：

struct RelAttr {

char \*relName; // 表名，可以为NULL

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

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

};

## 查询处理模块对外提供的函数调用接口

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

该方法的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。

返回给用户的查询结果res应当是一个关系：关系头+一组元组+返回元组数。

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。但是应尽可能利用代数优化及索引对查询过程进行优化。

# 语法分析模块

## HustBase支持的SQL语句

HustBase目前暂支持下面9种SQL语句，不区分大小写。语法如下：

1. create table relName(attrName1 Type1(len1),   
    attrName2 Type2(len2),  
    ...,   
    attrNameN TypeN(lenN));

注：int及float类型不需要注明长度

1. drop table relName;
2. create index indexName relName (attrName);
3. drop index indexName;
4. Select A1, A2, ..., Am  
   From R1, R2, ..., Rn  
   [Where A1' comp1 AV1 And A2' comp2 AV2 And ... And Ak' comp-k AVk];

comp支持6种比较运算符：=,<,>,>=,<=,<>。

1. Select \*  
   From R1, R2, ..., Rn  
   [Where A1' comp1 AV1 And A2' comp2 AV2 And ... And Ak' comp-k AVk];
2. Insert  
   Into relName   
   Values (V1, V2, ..., Vn);
3. Delete  
   From relName  
   [Where A1 comp1 AV1 And A2 comp2 AV2 And ... And Ak comp-k AVk];
4. Update relName  
   Set attrName = AV  
   [Where A1 comp1 AV1 And A2 comp2 AV2 And ... And Ak comp-k AVk];

## 语法分析流程

SQL语法分析的大致流程如图9.1所示。一条SQL语句通过词法器的解析，返回对应的单词句柄，然后语法分析器根据单词句柄和定义的语法规则进行语法分析，最后将分析的结果存入sql\_str结构中。

yacc\_sql.y

语法规则

SQL语句

yylex

token(s)

yyparse

存入

sql\_str结构

图9.1 SQL语法分析流程图

lex\_sql.l文件定义词法规则，使得词法分析程序能识别出输入字符串中的各个单词。yacc\_sql.y文件定义语法规则，根据该程序中定义的语法规则，语法分析程序能识别出合法的SQL语句，并根据语句类别将其语句信息存入相应的数据结构中。数据结构的具体定义见9.3节。

## SQL语句对应的数据结构

对应上述SQL语句的数据结构定义如下：

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

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

};

//struct of select

typedef struct {

int nSelAttrs; //select子句中属性的数量

RelAttr \*selAttrs[MAX\_NUM]; //select子句中的属性

int nRelations; //from子句中表的数量

char \*relations[MAX\_NUM]; //from子句中的表

int nConditions; //where子句中条件的数量

Condition conditions[MAX\_NUM];//where子句中的条件

}selects;

//struct of insert

typedef struct {

char \*relName; //要插入记录的表名

int nValues; //插入的值的数量

Value values[MAX\_NUM]; //要插入的值

}inserts;

//struct of delete

typedef struct {

char \*relName; //要删除记录的表名

int nConditions; //where子句中条件的数量

Condition conditions[MAX\_NUM]; //where子句中的条件

}deletes;

//struct of update

typedef struct {

char \*relName; //要修改记录的表名

char \*attrName; //要修改的属性名

Value value; //修改成的值

int nConditions; //where子句中条件的数量

Condition conditions[MAX\_NUM]; // where子句中的条件

}updates;

//struct of craete\_table

typedef struct {

char \*relName; //要创建的表的表名

int attrCount; //表中属性的数量

AttrInfo attributes[MAX\_NUM]; //属性的信息

}createTable;

//struct of drop\_table

typedef struct {

char \*relName; //要删除的表的表名

}dropTable;

//struct of create\_index

typedef struct {

char \*indexName; // 要创建的索引的名称

char \*relName; // 该索引涉及的表名

char \*attrName; // 该索引涉及的属性名

}createIndex;

//struct of drop\_index

typedef struct {

char \*indexName; // 要删除的索引的名称

}dropIndex;

**注：上述结构中凡涉及到多个条件、表名、属性等内容，均与输入的语句中的顺序相反。如select \* from table where con1 and con2 and con3;，对应的selects结构体中conditions[0]对应con3，conditions[1]对应con2，conditions[2]对应con1。**

union sqls{

selects sel;

inserts ins;

deletes del;

updates upd;

createTable cret;

dropTable drt;

createIndex crei;

dropIndex dri;

char \*errors;

};

typedef struct {

int flag; // 0--error; 1--select; 2--insert; 3--update;

// 4--delete; 5--create table; 6--drop table;

// 7--create index; 8--drop index;

union sqls sstr;

}sqlstr;

## 语法分析模块对外提供的函数调用接口

语法分析模块对外只提供一个语法分析函数：

RC parse(char\* st, sqlstr\* sqln);

该函数用于解析参数st传入的SQL语句，将解析的结果存入参数sqln中。若出错，返回错误类型RC。