各模块关键接口

目录

页式文件系统模块（PF）

PF_Manager Class

```
class PF_Manager
{
    public:
        PF_Manager    ();                                    //Constructor
        ~PF_Manager   ();                                    //Destructor
      RC CreateFile   (const char *fileName);                //Create a new file
      RC DestroyFile  (const char *fileName);                //Destroy a file
      RC OpenFile     (const char *fileName, PF_FileHandle   //Open a file
                       &fileHandle);
      RC CloseFile    (PF_FileHandle &fileHandle);           //Close a file
      RC AllocateBlock (char *&buffer);                      //Allocate a new scratch page in buffer
      RC DisposeBlock (char *buffer);                        //Dispose of a scratch page
};
```

- RC CreateFile (const char *fileName)

  该方法创建名为 fileName 的页式文件，该文件在系统中不存在。
- RC DestroyFile (const char *fileName)

  该方法销毁名为 fileName 的页式文件，该文件在系统中存在。
- RC OpenFile (const char *fileName, PF_FileHandle &fileHandle)

  该方法打开名为 fileName 的页式文件，该文件必须在系统中存在且由 CreateFile 方法创建。如果该方法成功执行，fileHandle 对象（详见下文）应指代该打开的文件。
- RC CloseFile (PF_FileHandle &fileHandle)

  该方法关闭 fileHandle 对象指代的文件，该文件必须曾通过 OpenFile 方法打开；当文件关闭时，它所有的文件页应移出缓存池。
- RC AllocateBlock (char *&buffer)

  在缓冲池中分配一个临时页，并将 buffer 指向该页。
- RC DisposeBlock (char *buffer)

  该方法清除缓冲池中 buffer 指向的临时页的内容，该页必须事先已由 AllocateBlock 分配。

PF_FileHandle Class

```
class PF_FileHandle
{
    public:
        PF_FileHandle              ();                                          //Default constructor
        ~PF_FileHandle             ();                                          //Destructor
        PF_FileHandle              (const PF_FileHandle &fileHandle);           //Copy constructor
        PF_FileHandle& operator=   (const PF_FileHandle &fileHandle);           //Overload =
      RC GetFirstPage              (PF_PageHandle &pageHandle) const;           //Get the first page
      RC GetLastPage               (PF_PageHandle &pageHandle) const;           //Get the last page
      RC GetNextPage               (PageNum current, PF_PageHandle              //Get the next page
                                    &pageHandle) const;
      RC GetPrevPage               (PageNum current, PF_PageHandle              //Get the previous page
                                    &pageHandle) const;
      RC GetThisPage               (PageNum pageNum, PF_PageHandle              //Get a specific page
                                    &pageHandle) const;
      RC AllocatePage              (PF_PageHandle &pageHandle)                  //Allocate a new page
      RC DisposePage               (PageNum pageNum);                           //Dispose of a page
      RC MarkDirty                 (PageNum pageNum) const;                     //Mark a page as dirty
      RC UnpinPage                 (PageNum pageNum) const;                     //Unpin a page
      RC ForcePages                (PageNum pageNum = ALL_PAGES) const;         //Write dirty page(s) to disk
};
```

- PF_FileHandle (const PF_FileHandle &fileHandle)
  该方法利用现有的 PF_FileHandle 对象创建一个新的 PF_FileHandle 对象，两个对象指代相同的文件。
- PF_FileHandle& operator= (const PF_FileHandle &fileHandle)
  该方法重载了=运算符，用来把一个 PF_FileHandle 对象赋给另一个，两个对象指代相同的文件。赋值前，等号左侧的 PF_FileHandle 对象不能指代一个已经打开的文件。
- RC GetFirstPage (PF_PageHandle &pageHandle)
  该方法把文件的第一页读入缓冲池，PF_PageHandle 对象（详见下文）应指代该页，用于访问该页的内容。
- RC GetLastPage (PF_PageHandle &pageHandle)
  该方法把文件的最后一页读入缓冲池，PF_PageHandle 对象应指代该页。
- RC GetNextPage (PageNum current, PF_PageHandle &pageHandle)
  该方法把页号为 current 的文件页之后的下一个有效页读入缓冲池，PF_PageHandle 对象应指代该页。
- RC GetPrevPage (PageNum current, PF_PageHandle &pageHandle)
  该方法把页号为 current 的文件页的前一个有效页读入缓冲池，PF_PageHandle 对象应指代该页。
- RC GetThisPage (PageNum pageNum, PF_PageHandle &pageHandle)
  该方法把页号为 pageNum 的文件页读入缓冲池，PF_PageHandle 对象应指代该页。
- RC AllocatePage (PF_PageHandle &pageHandle)

该方法给文件分配一个新页，并把该页读入内存，PF_PageHandle 对象应指代这个新页。

- RC DisposePage (PageNum pageNum)

  该方法清除文件中页号为 pageNum 的页。

- RC MarkDirty (PageNum pageNum)

  该方法把文件中页号为 pageNum 的页标记为“脏页”，以表明该页的内容已经或者即将发生改变。当文件页从缓冲池移出时，脏页中的内容会重新写入磁盘。

- RC UnpinPage (PageNum pageNum)

  该方法指明页号为 pageNum 的页已经不需要放在内存中。

- RC ForcePages (PageNum pageNum = ALL_PAGES)

  该方法把页号为 pageNum 的脏页中的内容从缓冲池复制到磁盘，该页继续保存在缓冲池中但不再标记为脏页。如果没有提供具体的页号（比如 pageNum = ALL_PAGES）,那么缓冲池中该文件所有的脏页都将写入磁盘且不再标记为脏页。

PF_PageHandle Class

```
class PF_PageHandle
{
    public:
        PF_PageHandle              ();                                  //Default constructor
        ~PF_PageHandle             ();                                  //Destructor
        PF_PageHandle              (const PF_PageHandle &pageHandle);   //Copy constructor
        PF_PageHandle& operator=   (const PF_PageHandle &pageHandle);   //Overload =
     RC GetData                    (char *&pData) const;                //Set pData to point to the
                                                                        //     page contents
     RC GetPageNum                 (PageNum &pageNum) const;            //Return the page number
};
```

- PF_PageHandle (const PF_PageHandle &pageHandle)

  该方法是利用现有的 PF_PageHandle 对象创建一个新的 PF_PageHandle 对象。
- PF_FileHandle& operator= (const PF_FileHandle &fileHandle)

  该方法重载了=运算符，用来把一个 PF_PageHandle 对象赋给另一个 PF_PageHandle 对象。
- RC GetData (char *&pData) const

  该方法用来访问文件页中的具体内容，该页必须保存在缓冲池中。若方法成功执行，pData 将指向该页的内容。
- RC GetPageNum (PageNum &pageNum) const

  该方法把 pageNum 设置为 PF_PageHandle 对象指代的文件页的页号。

记录管理模块（RM）

RM_Manager Class

```
class RM_Manager
{
    public:
        RM_Manager    (PF_Manager &pfm);                          //Constructor
        ~RM_Manager   ();                                         //Destructor
      RC CreateFile   (const char *fileName, int recordSize);     //Create a new file
      RC DestroyFile  (const char *fileName);                     //Destroy a file
      RC OpenFile     (const char *fileName, RM_FileHandle
                       &fileHandle);                              //Open a file
      RC CloseFile    (RM_FileHandle &fileHandle);                //Close a file
};
```

- RC CreateFile (const char *fileName, int recordSize)

  该方法通过调用 PF_Manager::CreateFile 创建名为 fileName 的页式文件，该文件中的记录大小为 recordSize。
- RC DestroyFile (const char *fileName)

  该方法通过调用 PF_Manager::DestroyFile 销毁名为 fileName 的页式文件。
- RC OpenFile (const char *fileName, RM_FileHandle &fileHandle)

  该方法通过调用 PF_Manager::OpenFile 打开名为 fileName 的页式文件。如果该方法成功执行，fileHandle 对象（详见下文）应指代该打开的文件。
- RC CloseFile (RM_FileHandle &fileHandle)

  该方法通过调用 PF_Manager::CloseFile 关闭 fileHandle 对象指代的文件。

RM_FileHandle Class

```
class RM_FileHandle
{
    public:
        RM_FileHandle    ();                                          //Constructor
        ~RM_FileHandle   ();                                          //Destructor
      RC GetRec          (const RID &rid, RM_Record &rec) const;     //Get a record
      RC InsertRec       (const char *pData, RID &rid);              //Insert a new record, return record id
      RC DeleteRec       (const RID &rid);                           //Delete a record
      RC UpdateRec       (const RM_Record &rec);                     //Update a record
      RC ForcePages      (PageNum pageNum = ALL_PAGES) const;        //Write dirty page(s) to disk
};
```

- RC GetRec (RID &rid, RM_Record &rec)

  该方法从文件中检索出标识符为 rid 的记录。如果该方法成功执行，rec 应该包含该记录的副本及其记录标识符（详见下文的 RM_Record）。应注意的是，RM_FileHandle 对象必须指代一个打开的文件，否则此方法以及该类中的所有方法都会报错。

- RC InsertRec (char *pData, RID &rid)

  该方法把 pData 指向的数据作为一条新的记录插入文件。如果该方法成功执行，返回的参数&rid 应指向这条新记录的标识符。

- RC DeleteRec (RID &rid)

  该方法从文件中删除标识符为 rid 的记录。如果删除记录后，包含该记录的页为空页，则可以通过调用 PF_Manager::DisposePage 删除该空页。

- RC UpdateRec (RM_Record &rec)

  该方法把文件中与 rec 有相同标识符的记录（详见下文的 RM_Record）更新为当前 rec 中的内容。

- RC ForcePages (PageNum pageNum = ALL_PAGES) const

  该方法通过调用 PF_Manager::ForcePages 把该文件中某个或者所有的脏页从缓冲池拷贝到磁盘。

RM_FileScan Class

```
class RM_FileScan
{
    public:
        RM_FileScan   ();                                    //Constructor
        ~RM_FileScan  ();                                    //Destructor
      RC OpenScan     (const RM_FileHandle &fileHandle,      //Initialize file scan
                       AttrType attrType, int attrLength,
                       int attrOffset, CompOp compOp,
                       void *value, ClientHint pinHint = NO_HINT);
      RC GetNextRec   (RM_Record &rec);                      //Get next matching record
      RC CloseScan    ();                                    //Terminate file scan
};
```

- RC OpenScan (const RM_FileHandle &fileHandle, AttrType attrType, int attrLength, int attrOffset, CompOp compOp, void *value, ClientHint pinHint = NO_HINT)

  该方法初始化一个 fileHandle 文件的记录扫描器。扫描过程中，仅检索指定属性满足指定条件（与 value 进行比较）的那些记录。如果 value 为空，那么条件也为空，在扫描过程中将检索所有记录。参数 attrType 和 attrLength 指明了待比较属性的类型和长度，包括 4 字节整型（INT）、4 字节浮点数（FLOAT）或者长度在 1~MAXSTRINGLEN 字节之间的字符串（STRING）。参数 attrOffset 指明了待比较属性在记录中的位置。参数 compOp 指明了属性值和 value 的比较方式。

CompOp	
EQ_OP	等于（attribute = value）
LT_OP	小于（attribute < value）
GT_OP	大于（attribute > value）
LE_OP	小于等于（attribute <= value）
GE_OP	大于等于（attribute >= value）
NE_OP	不等于（attribute <> value）
NO_OP	无比较（value 为空）

  为了访问文件中的记录，文件页必须首先调入内存，当内存中的缓冲区已满，就需要利用文件页替换算法来清理缓冲区。参数 pinHint 的引入是为了上层模块调用记录管理模块的文件扫描算法时，可以指明缓冲区的文件页替换算法。页式文件系统模块已实现了 Least-Recently-Used（LRU）策略，您可以实现其它算法，或者传入 NO_HINT 以表明使用默认替换算法。

- RC GetNextRec (RM_Record &rec)

  该方法用于检索出满足扫描条件的下一条记录。如果该方法成功执行，rec 应包含这条记录的副本及其记录标识符。

- RC CloseScan ()

  该方法用于关闭文件扫描。

RM_Record Class

```
class RM_Record
{
    public:
        RM_Record       ();                             //Constructor
        ~RM_Record      ();                             //Destructor
      RC GetData        (char *&pData) const;           //Set pData to point to the record's contents
      RC GetRid         (RID &rid) const;               //Get the record id
};
```

- RC GetData (char *&pData) const

  在调用该方法前，应首先创建 RM_Record 对象。该方法提供对记录内容的访问，若该方法成功执行，pData 应指向记录的内容。

- RC GetRid (RID &rid) const

  若该方法成功执行，rid 应包含记录的标识符。

RID Class

```
class RID
{
    public:
        RID             ();                             //Default constructor
        ~RID            ();                             //Destructor
        RID             (PageNum pageNum, Slot slotNum); //Construct RID from page and slot number
      RC GetPageNum     (PageNum &pageNum) const;       //Return page number
      RC GetSlotNum     (SlotNum &slotNum) const;       //Return slot number
};
```

- RID (PageNum pageNum, SlotNum slotNum)

  该方法通过页号和槽号创建一个新的记录标识符对象，一个记录标识符通过记录所在的页号和该页内的槽号唯一标识一条记录。

- RC GetPageNum (PageNum &pageNum)

  若该方法成功执行，pageNum 应为该标识符标识的记录所在的页号。

- RC GetSlotNum (SlotNum &slotNum)

  若该方法成功执行，slotNum 应为该标识符标识的记录所在的槽号。

索引模块（IX）

IX_Manager Class

```
class IX_Manager
{
    public:
        IX_Manager     (PF_Manager &pfm);                          //Constructor
        ~IX_Manager    ();                                         //Destructor
      RC CreateIndex   (const char *fileName, int indexNo, AttrType attrType,   //Create new index
                        int attrLength);
      RC DestroyIndex  (const char *fileName, int indexNo);        //Destroy index
      RC OpenIndex     (const char *fileName, int indexNo,         //Open index
                        IX_IndexHandle &indexHandle);
      RC CloseIndex    (IX_IndexHandle &indexHandle);              //Close index
};
```

- RC CreateIndex (const char *filename, int indexNo, AttrType attrType, int attrLength)

  该方法为文件 fileName 创建标号为 indexNo 的索引，该索引可以保存在文件 fileName.indexNo 里。被索引属性的类型和长度分别在参数 attrType 和 attrLength 中指明。与记录管理模块一样，它们可以是 4 字节整型（INT）、4 字节浮点数（FLOAT）或者长度在 1~MAXSTRINGLEN 字节之间的字符串（STRING）。

- RC DestroyIndex (const char *fileName, int indexNo)

  该方法通过删除页式文件系统中的索引文件来删除 fileName 上标号为 indexNo 的索引。

- RC OpenIndex (const char *fileName, int indexNo, IX_IndexHandle &indexHandle)

  该方法通过打开页式文件系统中的索引文件来打开 fileName 上标号为 indexNo 的索引。如果成功执行，indexHandle 对象应指代该打开的索引。indexHandle 对象用来添加和删除索引中的项（详见下文），也可传入 IX_IndexScan 的构造函数来扫描索引。用户可以通过创建多个的 indexHandle 来多次打开同一个索引，但同一时刻只有一个 indexHandle 能修改该索引。

- RC CloseIndex (IX_IndexHandle &indexHandle)

  该方法通过关闭页式文件系统中的索引文件来关闭 indexHandle 对象指代的索引。

IX_IndexHandle Class

```
class IX_IndexHandle
{
    public:
        IX_IndexHandle    ();                                          //Constructor
        ~IX_IndexHandle   ();                                          //Destructor
      RC InsertEntry      (void *pData, const RID &rid);               //Insert new index entry
      RC DeleteEntry      (void *pData, const RID &rid);               //Delete index entry
      RC ForcePages       ();                                          //Copy index to disk
};
```

- RC InsertEntry (void *pData, const RID &rid)

  调用该方法时 IX_IndexHandle 对象必须指代一个打开的索引。该方法在 IX_IndexHandle 指代的索引中插入一个新项(*pData, rid)，参数 pData 指向待插入的属性值，参数 rid 是具有该属性值的记录的标识符。

- RC DeleteEntry (void *pData, const RID &rid)

  调用该方法时 IX_IndexHandle 对象必须指代一个打开的索引。该方法把(*pData, rid)从 IX_IndexHandle 指代的索引中删除。

- RC ForcePages ()

  该方法把 IX_IndexHandle 对象所在的数据页全部从内存写回磁盘。

IX_IndexScan Class

```
class IX_IndexScan
{
    public:
        IX_IndexScan    ();                                  //Constructor
        ~IX_IndexScan   ();                                  //Destructor
     RC OpenScan        (const IX_IndexHandle &indexHandle,  //Initialize index scan
                         CompOp compOp, void *value,
                         ClientHint pinHint = NO_HINT);
     RC GetNextEntry    (RID &rid);                          //Get next matching entry
     RC CloseScan       ();                                  //Terminate index scan
};
```

- RC OpenScan (const IX_IndexHandle &indexHandle, CompOp compOp, void *value, ClientHint pinHint = NO_HINT)

  该方法初始化一个 indexHandle 指代的索引的条件扫描器，该扫描器应返回所有满足条件的记录的标识符。参数 compOp 和 value 的含义与 RM_FileScan::OpenScan 方法中的含义相同，具体可查阅记录管理模块。

- RC GetNextEntry (RID &rid)

  该方法把参数 rid 设置为索引扫描中下一条记录的标识符。

- RC CloseScan ()

  该方法终止索引扫描。

系统管理模块（SM）

The CREATE DATABASE Command

创建数据库 DBname 可以通过在文件系统中创建目录 DBname 来实现，数据库的相关数据都保存在该目录中。创建数据库的示例代码如下。

```
main (int argc, char **argv){
    //argv[0] points to the name of the command, argv[1] points to argument DBname
    char *dbname;
    char command[80] = "mkdir ";

    if (argc != 2) {
        cerr << "Usage: " << argv[0] << " dbname \n";
        exit(1);
    }

    //The database name is the second argument
    dbname = argv[1];

    //Create a subdirectory for the database
    system (strcat (command, dbname));

    if (chdir(dbname) < 0) {
        cerr << argv[0] << " chdir error to " << dbname << "\n";
        exit(1);
    }
}
```

创建数据库 DBname 后，应生成两个元数据文件 relcat 和 attrcat，保存在目录 DBname 中。relcat 用于记录数据库 DBname 中关系表的相关信息，对于每个关系表，可保存的信息有：

relName	关系表名称
tupleLength	每个元组的最大长度（以字节为单位）
attrCount	关系表中的属性个数
indexCount	关系表中有索引的属性个数

attrcat 用于记录数据库所有关系表中的所有属性，对于每个属性，可保存的信息有：

relName	属性所在的关系表名称
attrName	属性名称

offset	到元组开始位置的字节偏移量
attrType	属性类型
attrLength	属性长度
indexNo	索引号，如果没有被索引则为-1

The DROP DATABASE Command

删除数据库 DBname 可以通过删除目录 DBname 来实现。

```
char command[80] = “rm -r ”;
system (strcat (command, dbname));
```

SM_Manager Class

```
//Used by CreateTable
struct AttrInfo {
    char *attrName;                 //Attribute name
    AttrType attrType;              //Type of attribute
    int attrLength;                 //Length of attribute
};

//Used by Printer class
struct DataAttrInfo {
    char relName[MAXNAME+1];        //Relation name
    char attrName[MAXNAME+1];       //Attribute name
    int offset;                     //Offset of attribute
    AttrType attrType;              //Type of attribute
    int attrLength;                 //Length of attribute
    int indexNo;                    //Attribute index number
};

class SM_Manager
{
    public:
        SM_Manager      (IX_Manager &ixm, RM_Manager &rmm);     //Constructor
        ~SM_Manager     ();                                     //Destructor
      RC OpenDb         (const char *dbName);                   //Open database
      RC CloseDb        ();                                     //Close database
      RC CreateTable    (const char *relName, int attrCount,    //Create relation
                         AttrInfo *attributes);
      RC DropTable      (const char *relName);                  //Destroy relation
      RC CreateIndex    (const char *relName, const char *attrName);  //Create index
      RC DropIndex      (const char *relName, const char *attrName);  //Destroy index
      RC Load           (const char *relName, const char *fileName);  //Load utility
      RC Help           ();                                     //Help for database
      RC Help           (const char *relName);                  //Help for relation
      RC Print          (const char *relName)                   //Print relation
      RC Set            (const char *paramName, const char *value)    //Set system parameter
};
```

- RC OpenDb (const char *dbName)

  该方法把系统当前目录切换到*dbName 所在的目录。

- RC CloseDb ()

  该方法关闭当前数据库中所有打开的文件，并把缓冲区中所有的相关数据库数据写入磁盘。

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

该方法创建一个新的关系表*relName。参数 attrCount 指明了关系表中的属性个数（1 到 MAXATTRS 之间）；参数 attributes 是长度为 attrCount 的数组，第 i 个元素 attributes[i]保存了第 i 个属性的属性名、类型和长度。此外，该关系表的相关信息应写入 relcat，每个属性的相关信息应写入 attrcat。

- RC DropTable (const char *relName)

  该方法删除关系表*relName 以及关系表的所有索引。其中，关系表文件是通过调用 RM_Manager::DestroyFile 删除，而索引文件通过调用 IX_Manager::DestroyIndex 删除。此外，关系表*relName 以及所有相关索引应从 relcat 和 attrcat 中删除。

- RC CreateIndex (const char *relName, const char *attrName)

  该方法通过调用 IX_Manager::CreateIndex 为关系表*relName 的属性*attrName 创建索引，并更新 relcat 和 attrcat。

- RC DropIndex (const char *relName, const char *attrName)

  该方法通过调用 IX_Manager::DestroyIndex 删除关系表*relName 中属性*attrName 的索引，并更新 relcat 和 attrcat。

- RC Load (const char *relName, const char *fileName)

  该方法把文件*fileName 里的所有元组插入到关系表*relName 中。它通过调用 RM_Manager::OpenFile 打开关系表文件，通过 IX_Manager::OpenIndex 打开每个索引，然后依次读取文件*fileName 里的各个元组，调用 RM_FileHandle::InsertRec 把元组插入到关系表*relName 中，调用 IX_IndexHandle::InsertEntry 在索引中创建相关的索引项。插入所有元组后，关闭所有打开的文件。

- RC Help ()

  该方法从 relcat 中读取所有的关系表信息，并利用 Printer 类输出到前端。

- RC Help (const char *relName)

  该方法从 attrcat 中读取关系表*relName 的所有属性的信息，并利用 Printer 类输出到前端。

- RC Print (const char *relName)

  该方法打开关系表*relName，利用 Printer 类输出表中的所有内容，然后关闭该关系表。

- RC Set (const char *paramName, const char *value)

  该方法将 paramName 标识的系统参数设置为*value 指定的值。

Printer Class

```
class Printer
{
    public:
        Printer (const DataAttrInfo *attributes, const int attrCount);
        ~Printer ();
        void PrintHeader (ostream &c) const;
        void Print (ostream &c, const char * const data);
        void Print (ostream &c, const void * const data[]);
        void PrintFooter (ostream &c) const;
};
```

- Printer (const DataAttrInfo *attributes, const int attrCount)

  Printer 的构造函数需要两个参数：参数 attributes 是长度为 attrCount 的数组，描述了待输出元组的模式信息；参数 attrCount 指明了元组的属性个数。
- void PrintHeader (ostream &c) const

  该方法将输出每个属性的名称，然后打印一行破折号。在输出元组的具体内容前，应先调用该方法。
- void Print (ostream &c, const char * const data)

  void Print (ostream &c, const void * const data[])

  该方法将输出 char* data 指向的元组；对于查询解析模块，该方法还可以接收一个 void*指针数组，其中的每个指针都指向一个属性值。
- void PrintFooter (ostream &c) const

  该方法把总共被打印的元组个数作为“页脚”信息输出到前端。在最后一个元组被输出后，应调用该方法。

以下是 Print 类的示例。

```
AttrInfo *attributes;
int attrCount;
RM_FileHandle rfh;
RM_Record rec;
char *data;

// Fill in the attributes structure, define the RM_FileHandle
…

// Instantiate a Printer object and print the header information
Printer p (attributes, attrCount);
p.PrintHeader (cout);

// Open the file and set up the file scan
```

```
if ((rc = rmm → OpenFile (relName, rfh)))
        return (rc);

RM_FileScan rfs;

if ((rc = rfs.OpenScan (rfh, INT, sizeof(int), 0, NO_OP, NULL)))
        return (rc);

// Print each tuple
while (rc != RM_EOF){
        rc = rfs.GetNextRec(rec);

        if (rc != 0 && rc != RM_EOF)
                return (rc);

        if (rc != RM_EOF){
                rec.GetData(data);
                p.Print(cout, data);
        }
}

// Print the footer information
p.PrintFooter(cout);

// Clost the scan, file, delete the attributes pointer, etc.
…
```

查询解析模块（QL）

QL_Manager Class

```
struct RelAttr {
    char *relName;          //relation name (may be NULL)
    char *attrName;         //attribute name
    friend ostream &operator<<  (ostream &s, const RelAttr &ra);
};

struct Value {
    AttrType type;          //type of value
    void     *data;         //value
    friend ostream &operator<<  (ostream &s, const Value &v);
};

struct Condition{
    RelAttr    lhsAttr;     //left-hand side attribute
    CompOp op;              //comparison operator
    int        bRhsIsAttr;  //TRUE(T) if right-hand side is an attribute
    RelAttr    rhsAttr;     //right-hand side attribute (bRhsIsAttr = T)
    Value      rhsValue;    //right-hand side value (bRhsIsAttr = F)
    friend ostream &operator<<  (ostream &s, const Condition &c);
};

class QL_Manager
{
    public:
        QL_Manager          (SM_Manager &smm, IX_Manager &ixm,   //Constructor
                             RM_Manager &rmm);
        ~QL_Manager         ();                                  //Destructor
     RC Select              (int             nSelAttrs,          //# attrs in Select clause
                             const RelAttr selAttrs[],           //attrs in Select clause
                             int             nRelations,         //# relations in From clause
                             const char * const relations[],     //relations in From clause
                             int             nConditions,        //# conditions in Where clause
                             const Condition conditions[]);      //conditions in Where clause

     RC Insert              (const char *relName,               //relation to insert into
                             int           nValues,              //# values to insert
                             const Value values[]);              //values to insert

     RC Delete              (const char *relName,               //relation to delete from
```

```
                              int          nConditions,          //# conditions in Where clause
                              const Condition conditions[]);     //conditions in Where clause

    RC Update                 (const char *relName,              //relation to update
                              const RelAttr &updAttr,            //attribute to update
                              const int bIsValue,                //0/1 if RHS of = is attribute/value
                              const RelAttr &rhsRelAttr,         //attr on RHS of =
                              const Value &rhsValue,             //value on RHS of =
                              int nConditions,                   //# conditions in Where clause
                              const Condition conditions[]);     //conditions in Where clause
};
```

- RC Select (int nSelAttrs, const RelAttr selAttrs[], int nRelations, const char * const relations[], int nConditions, const Condition conditions[])

  该方法的六个参数可以分为三组：每组的第一个参数是整数 n，指明了第二个参数里包含多少项数据；第二个参数是长度为 n 的数组，包含了实际的数据。例如，参数 nSelAttrs 指明了 Select 语句中选定属性的数量，而参数 selAttrs 是长度为 nSelAttrs 的数组，包含实际属性。（单个属性“*”表明是查询 Select *。）QL_Manager 类中的其它方法也使用了类似的传参方式。

  QL_Manager::Select 是查询解析模块的核心。当用户输入有效的 Select 语句时，应在屏幕上打印出正确的执行结果。一些设计要求和建议如下：您应构建一棵查询树作为逻辑查询计划，然后再将其转换为物理计划。查询执行策略应利用索引来提高查询效率。特别的，

  - 假设查询语句中包含选择条件 R.A = v，这里的 v 是一个常量。如果关系 R 在属性 A 上有索引，则可以使用索引模块的索引扫描以及对 RM_FileHandle::GetRec 的调用来获取关系表 R 的元组，而不是利用记录管理模块进行文件扫描。
  - 可以使用最简单的循环嵌套来实现联接或者笛卡尔积。假设需要基于属性相等将 R 和 S 连接起来，S 没有类似 S.A = v 的选择条件并且 S 在连接属性 A 上有索引，那么可以使用循环嵌套的内层循环来遍历 S 并在 S 上进行索引扫描而非文件扫描。

  查询结果返回给用户时应表示成一个关系表：表头，后跟一组元组，然后是元组数。这里必须使用系统管理模块提供的 Printer 类打印查询结果。

- RC Insert (const char *relName, int nValues, const Value values[])

  该方法在关系表*relName 里插入一个新的元组，该元组的属性值由传入的参数指定，你的代码需要确保参数 nValues 和 values 指定的属性数量和数据类型必须与关系表的模式相符合。该方法首先构建待插入的元组，然后调用 RM_FileHandle::InsertRec 插入元组，再调用 IX_IndexHandle::InsertEntry 在索引中为该元组创建合适的索引项。为了给用户提供反馈，该方法应通过 Printer 类打印插入的元组。

- RC Delete (const char *relName, int nConditions, const Condition conditions[])

  该方法删除关系表*relName 中所有满足指定条件的元组。如果没有指定条件，该方法将删除*relName 中的所有元组。

  令 R 表示关系表*relName。假设存在条件 R.A = v（v 是一个常量），并且 R 在属性 A 上有索引，那么可以通过索引的扫描快速找出 R 中满足 R.A = v 的元组，并检查它们是否满足其余的条件，若满足则删除相应的元组，删除元组后也需删除对应的索引项。为了给用户提供反馈，该方法应通过 Printer 类

打印删除的元组。

尽管 Delete 语句的执行策略通常比 Select 语句简单，但仍应为 Delete 语句构建物理查询计划，Select 语句中的某些物理运算符可以重复使用。

- RC Update (const char *relName, const RelAttr &updAttr, const int bIsValue, const RelAttr &rhsRelAttr, const Value &rhsValue, int nConditions, const Condition conditions[])

  该方法更新关系表*relName 中所有满足指定条件的元组，把它们的属性*updAttr 更新成特定的值。如果 bIsValue = 1，那么新的属性值应为 rhsValue 中的常量。否则，新的属性值应从属性 rhsRelAttr 中获得。

  如果要更新被索引的属性，请不要忘记更新对应的索引项。Update 语句也应构建物理查询计划，并同样可以重复使用某些物理运算符。为了给用户提供反馈，该方法应通过 Printer 类打印更新的元组。