OrientX 索引模块概要设计说明书

负责人：蒋瑜
编写人：谢敏
系统版本号：OrientX Version 2.5
完成时间：
开发单位：中国人民大学 IDKE 实验室 XML 工作组

1. 引言

本说明书主要介绍 OrientX 系统 2.5 版中索引模块（包括节点索引和纸索引）的概要设计。

以下各章节安排：第 2 节概述，介绍相关的背景知识，模块的主要功能及子模块构成；第 3 节介绍索引模块的 API；第 4 节简单说明索引底层的 BP-Tree 所使用的算法；第 5 节介绍后继的主要工作。

2. 概述

2.1 XML 索引概述

在 XML 上的查询（XQuery[1] / XPath[2]）需要通过在 XML 的树型结构上遍历来查找满足特定条件的节点，对应于这样的需求，我们总是希望能在 XML 文档上建立相应的索引结构来加速查询。之前在 XML 索引这方面的工作大致有以下几个方面，一方面是对的 XML 文档上的节点的带路径信息的编码（可以是区域编码或是其他任何一种编码）建立索引，然后在查询某条路径上的节点时，首先通过索引将路径上的所有节点取出来，然后用某种结构连接的方法将各个节点连接起来，这方面的工作包括 XISS(VLDB 2001)，XR-Tree(ICDE 2003)等；另一方面的工作是在考虑到过前面一方面的工作中结构连接的代价太高，于是考虑在索引的结构上，先将一条路径整体编码，然后索引这个路径的编码，从而避免连接路径上的节点,这方面的工作有 BLAS（SIGMOD 2004）等；还有一些工作将 XML 文档看作 Sequence，然后将在 XML 上的查询看作是在 Sequence 上作 Matching，在这种 Sequence 上建立 Suffix Tree 结构为基础的索引能有效地满足查询要求，这方面的工作有 VIST（SIGMOD 2003）等。

2.2 OrientX 的索引简介

OrientX 的索引模块包括两个部分，*路径索引*（*PathIndex*）和*值索引*（*ValueIndex*）。两者都是针对 XML 节点建立的索引：路径索引是在 XML 文档的 ElementNode 的 RegionCode 编码基础上，对指定的<setID, docID, eID>三元组，即同一 Dataset（setID）中某个文档(docID)的某个 RegionCode(eID)建立索引，由于 RegionCode 自身带有的路径信息，所以在

---

[1] S. Boag et. XQuery 1.0: An XML query language http://www.w3.org/TR/xquery, 2002
[2] J. Clark & S. DeRose XML path language(XPath) http://www.w3.org/TR/xpath, 1999

我们的工作中，称对这种编码的索引为*路径索引*（*PathIndex*）；值索引是针对 XML 文档的 ElementNode 的文本值或是 ElementNode 的属性值，通过 XML 文档的 Schema 获得相应的类型信息，然后在类型信息的指导下对 XML 文档中先前提到的两类值建立索引，我们称这部分的工作为*值索引*（*PathIndex*）。

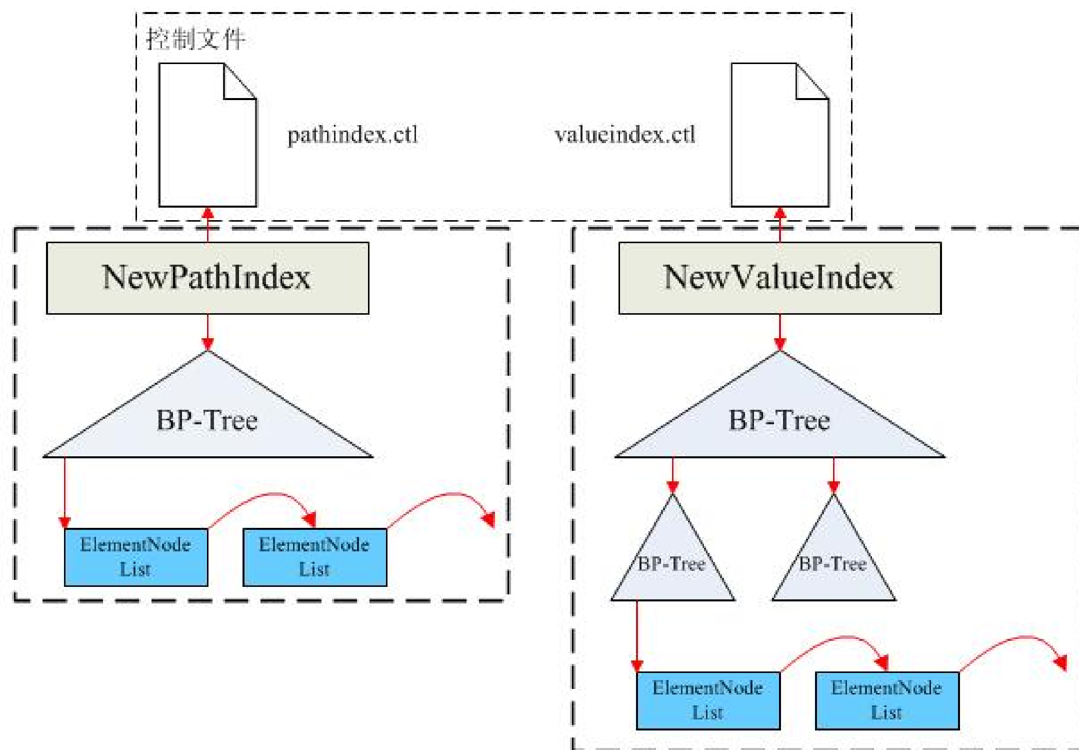
这部分的工作是在 IDKE 小组已经毕业成员王静编写的索引基础上完成，系统以前的索引请参见文档：

（Jing Wang, Xiaofeng Meng, Shan Wang. SUPEX A Schema-Guided Path Index for XML Data. Proc. of Very Large Database Conference(VLDB 2002),September 2002）

这部分索引模块在建立好后，其功能主要在于，结合 OrientX 系统中的查询（XQuery/XPath）处理模块，有效地加速结构连接中对满足某一位置条件的节点的查找，也能够很好的处理查询满足某一谓词的节点（通过在 B+Tree 上的 RangeQuery）的操作，这部分的相关信息可以参见系统查询模块的概要说明书。

2.3 OrientX 索引的设计

索引模块的设计图示如下：



- 路径索引的设计：

  路径索引模块为 NewPathIndex，命名的方式为了区别于以前小组成员得工作。这部分索引以<setID, docID, eleID>这样的三元组作为查找关键字，所以在设计这部分的时候，使用了比较直接的方法，将<setID, docID, eleID>直接组成一个字符串作为关键字，然后将这些关键字组织成 BP-Tree 的节点关键字， 查找的时候就直接用这个三元组组成的字符串作为关键字到 *BP-Tree*（系统所实现的 B+Tree）中查找。三元组对应的所有 ElementNode Item 都按照链表的方式组织在一起。

- 值索引的设计：

值索引模块为 NewValueIndex，命名的方式为了区别于以前小组成员的工作。值索引以<setID, docID, eleID>三元组对应的 ElementNode 的文本值或是<setID, docID, aID>对应的 AttributeNode 的属性值作为查找关键字，在设计这部分的时候，使用了两层的 BP-Tree 结构，第一层 BP-Tree 使用<setID, docID, nodeID>作为关键字，在 BP-Tree 中找到对应的第二层 BP-Tree 根节点后，再用查询的值作为关键字到对应的第二层 BP-Tree 中查找符合查询条件的 ElementNode 序列。索引值对应的所有 ElementNode Item 也都按链表的方式组织在一起。

- 控制文件的设计：

索引的控制文件中除了存储一些必要的索引控制信息如 BP-Tree 的根页号外，还存储了一些必要的统计信息:

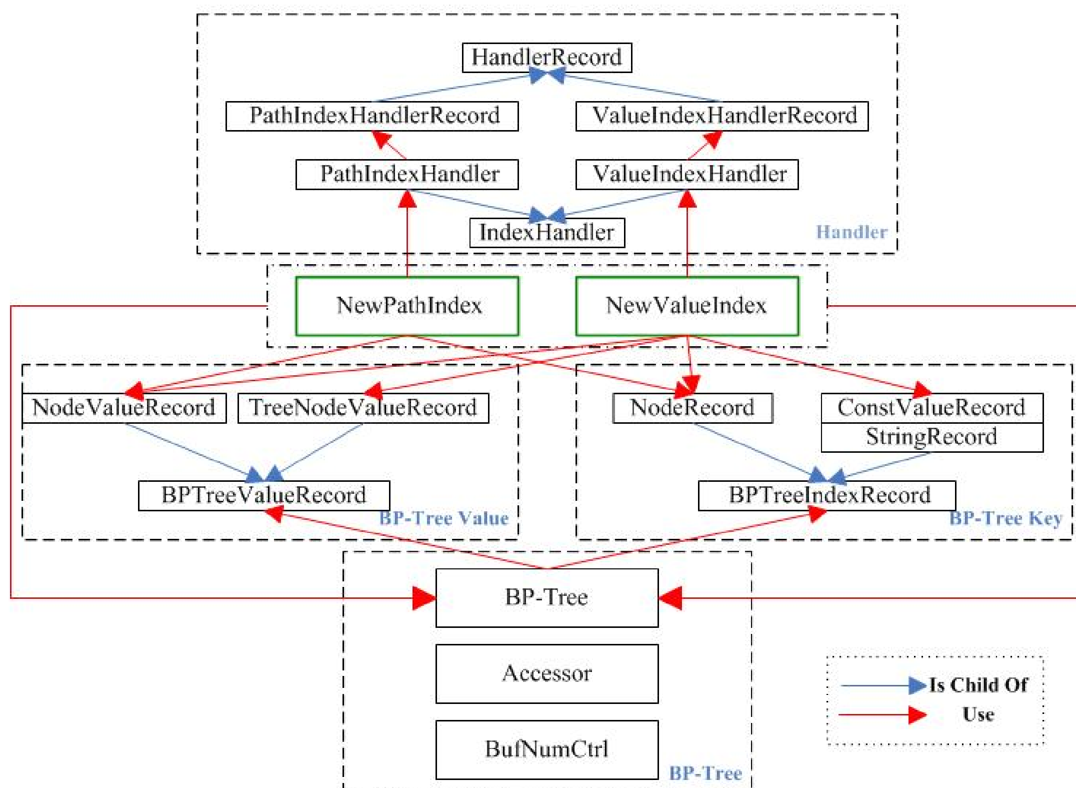
- ✓ 整个索引模块所索引的节点（ElementNode）数目，以及这些节点所耗费的数据页数目
- ✓ 整个索引模块所索引的节点中属于每个<setID>（DataSet）的节点数目，以及这些节点所耗费的数据页数目
- ✓ 整个索引模块所索引的节点中属于每个<setID, docID>（Dataset 对应的某个 Document）的节点数目，以及这些节点所耗费的数据页数目

这部分的统计信息服务于上层的查询模块，可以在查询过程中辅助查询模块选择适当的查询策略。

2.4 OrientX 索引模块的类结构层次

索引模块的类大致可以分为五个部分: 1. 索引接口 2. BP-Tree 接口 3. BP-Tree Key 4. BP-Tree Value 5. Handler 接口

所有类的层次结构图如下：



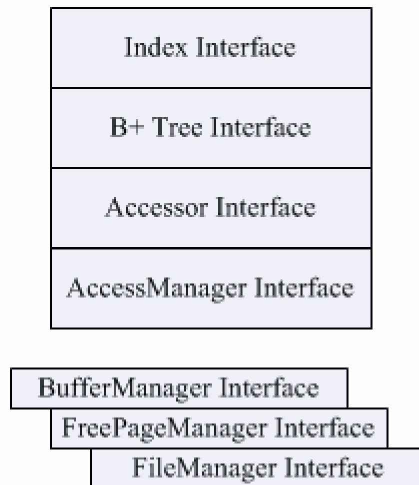
索引模块的类概要：

- 索引接口：（用于其他模块访问索引）
  - NewPathIndex ：
    路径索引所对应的类，包含所有的路径索引的创建，删除，查询操作，并可以通过这个模块访问相应的路径索引控制信息
  - NewValueIndex ：
    值索引对应的雷，包含所有的值索引的创建，删除，查询操作，并可以通过这个模块访问相应的值索引控制信息
- BP-Tree 接口：（对应索引底层数据结构）
  - BP-Tree ：
    系统所实现的 B+树数据结构，是整个索引模块的基础，这个类包含 BP-Tree 操作接口，向 B+树中的插入，删除和查询操作，另外还有一组 RanagQuery 接口可以在 B+Tree 上查找（<,>,=,!=,>=,<=）某个指定值的所有元组
  - Accessor ：
    BP-Tree 访问底层存储的中间接口，处于 BP-Tree 于 AccessManager 中间，用来封装 BP-Tree 对节点的实际操作
  - BufNumCtrl ：
    BP-Tree 使用的缓存控制模块，可以控制索引模块使用的 Buffer 的数目
- BP-Tree Key: （BP-Tree 的 Key，是 B+树上查找单位）
  - NodeRecord :
    将<setId, docID, eID> 三元组组织成关键字
  - ConstValueRecord :
    将常量（char,int,long,float,double）值组织成关键字
  - StringRecord :
    将字符串常量组织成关键字
- BP-Tree Value: (BP-Tree 的 Value,是 B+树上叶子节点所索引的值)
  - NodeValueRecord :
    将 ElementNode List 的首页号，ElementNode List 中的节点数目，消耗的页数组织成值 Item
  - TreeNodeValueRecord :
    将一个 BP-Tree 的 root 和 leaf 所在数据页号,下层所有的 ElementNode List 中的节点数目，消耗的页数组织成值 Item
- Handler 接口：（返回值的接口，包括顺序依次获取所有结果以及获取全部结果接口）
  - IndexHandler ：所有 Handler 的祖先类
    - PathIndexHandler ：
      路径索引的返回值 Handler
    - ValueIndexHandle ：
      值索引的返回值 Handler
  - HandlerRecord ：所有 HandlerRecord 的祖先类
    - PathIndexHandlerRecord :
      PathIndexHandler 对应的 Item
    - ValueIndexHandlerRecord :
      ValueIndexHandler 对应的 Item

3. BP-Tree 模块说明

BP-Tree(OrientX 系统中设计的 B+Tree)是整个索引模块的基础,在这里特别对 OrientX 中所实现的 BP-Tree 进行一些说明。

下图是索引模块以及下层模块的层次结构图



在上图中，上层的 Index Interface，B+Tree Interface，Accessor 三个层次对应于索引模块中处理的三个层次。下面的 AccessManager Interface 是系统中的访问控制模块，直接操作系统下面的 BufferManager（缓存管理模块），FreePageManager（空闲页管理模块）和 FileManager（文件管理模块）。可以看出 BP-Tree（即图中 B+ Tree Interface）在模块中起着中间层的重要作用。

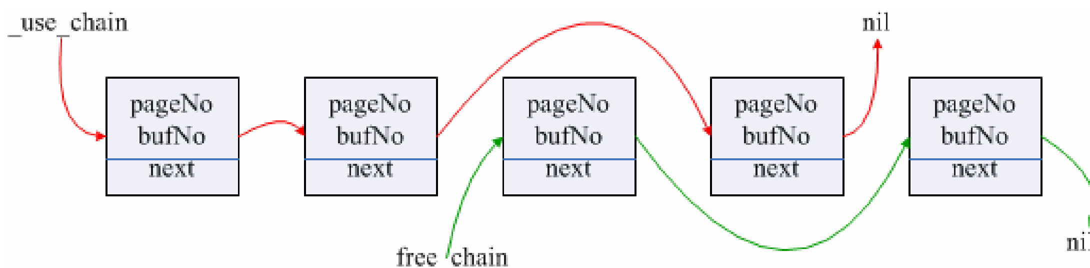
OrientX 中的 BP-Tree 有三个层次构成，最上层 BP-Tree 类用于与索引模块（NewPathIndex，NewValueIndex）交互，中间的 Accessor 用于抽象 BP-Tree 对节点的操作，目的是 BP-Tree 只是对结点进行操作，怎么操作由抽象出的 Accessor 来完成。最后在 Accessor 之下是 BufNumCtrl 类，这个类的工作十分简单，就是用一个链表结构来记录已使用的 Buf，控制总的 Buf 使用数目在一定范围之内，同时期望能够避免下层的 buffer 块号查找时间,能够在较快的速度下返回 buffer 块号。

B+树是一种有效的外存索引结构，在 B+树的设计中，有两个问题十分重要，就是对 B+树节点的分裂和合并的处理策略。OrientX 系统中的 BP-Tree 的实现中对于节点的分裂和合并采取以下策略，对于节点的分裂，直接将节点分成 lower[m/2]和 m-lower[m/2]+1 两个部分(m 是 BP-Tree 节点所能容纳的关键字上限)，然后再按上述过程递归向上分裂；而对于节点的合并，当节点的 key 数目少于 lower[m/2]时，首先看兄弟节点是否有多余的 key，如果兄弟节点有多余的 key 直接进行 Rotate 操作，避免节点的合并操作，如果兄弟节点都没有多余的 key 就 Collapse，并按上述过程递归向上。

Accessor 是一个抽象操作的中间类，定义了一组 BP-Tree 可能用到的对节点及其关键字的操作，操作包括 B+树节点的载入和删除，节点头信息（是否是根节点，是否是叶节点）的获取和改写，节点 key 的获取和改写以及节点 value 的获取和改写，所有 Accessor 接口定义的方法列举如下，具体含义请参考头文件注释。

```
class Accessor
{
public:
        void Init(int datasetID, int keylen, int vallen);

        //==============================================

        bool IsRoot(short bufNo);
        bool IsLeaf(short bufNo);

        //==============================================

        int GetNewNode();
        short LoadNode(int pageNo);
        void FreeNode(short bufNo);

        //==============================================

        unsigned char GetNodeMask(short bufNo);
        short GetNodeRecNum(short bufNo);
        int GetNodeNextPage(short bufNo);

        void WriteNodeMask(short bufNo, unsigned char mask);
        void WriteNodeRecNum(short bufNo, short recnum);
        void WriteNodeNextPage(short bufNo, int nextpage);

        void CopyKeyFromN2N(short srcbufno, short desbufno, int src_i, int des_i);
        void CopyInfoN2N(short srcbufno, short desbufno, int src_i, int des_i, int rep);
        void MoveRecord(short bufNo, int st_i, int new_i, int rep);

        int GetInterNodeInfo(short bufNo, int index);
        void WriteInterNodeInfo(short bufNo, int index, int pageNo);
        void GetLeafNodeInfo(short bufNo, int index, ValType &val);
        void WriteLeafNodeInfo(short bufNo, int index, ValType &val);
        void GetNodeKey(short bufNo, int index, KeyType &key);

        void WriteNodeKey(short bufNo, int index, KeyType &key);
};
```

此外还考虑到树结点的缓存问题，所以实验在底层的 BufferManager 的缓存管理机制基础上又增加了一个 BufNumCtrl 类用作缓存数量的控制，在初始化时可以选择使用它作为控制或者不用，实现使用空闲链表来管理缓存内容，这部分可以控制缓存的数目以及可以在已经换存页的情况下较快给出 bufno，这部分图示如下：



4. 索引模块 API 说明

4.1 NewPathIndex 部分 API 说明

函数名称：Open ()

函数说明：在使用 PathIndex 之前，需要先调用此函数将控制信息读入

输入参数：
输出参数：

函数名称：Close（）
函数说明：在使用 PathIndex 结束之后需要调用此函数将一些必要的控制信息刷出
输入参数：
输出参数：

函数名称：CreatePathIndex(int datasetID, int docID)
函数说明：结合 XML 的 Schema 信息对一个给定的文档上的所有节点建立路径索引
输入参数：datasetID dataset 编号
docID doc 编号
输出参数：0 表示正确，其他表示错误

函数名称：CreatePathIndex（int datasetID, int docID, DTDNodeCode &eID）
函数说明：对一个给定文档上的指定节点建立索引
输入参数：datasetID dataset 编号
docID doc 编号
eID 指定节点的编码
输出参数：0 表示正确，其他表示错误

函数名称：DropPathIndex（int datasetID, int docID）
函数说明：结合 XML 文档的 Schema 信息释放一个文档对应的所有节点的索引
输入参数：datasetID dataset 编号
docID doc 编号
输出参数：0 表示正确，其他表示错误

函数名称：DropPathIndex（int datasetID, int docID， DTDNodeCode &eID）
函数说明：释放对应文档上指定节点的索引
输入参数：datasetID dataset 编号
docID doc 编号
eID 指定节点的编码
输出参数：0 表示正确，其他表示错误

函数名称：GetHandler(int datasetID,
int docID, DTDNodeCode &eID, PathIndexHandler &handler)
函数说明：获得指定节点的索引所记录的 ElementNode List，并将结果存入 handler
输入参数：datasetID dataset 编号
docID doc 编号
eID 指定节点编码
handler 返回值的容器，存储了对应的 List 的首页号
输出参数：0 表示正确，其他表示错误

4.2 NewValueIndex 部分 API 说明

函数名称：Open（）
函数说明：在使用 ValueIndex 之前，需要先调用此函数将控制信息读入
输入参数：
输出参数：

函数名称：Close（）
函数说明：在使用 ValueIndex 结束之后需要调用此函数将一些必要的控制信息刷出
输入参数：
输出参数：

函数名称：CreateValueIndex(int datasetID, int docID)
函数说明：结合 XML 文档的 Schema 信息对一个给定文档上所有节点及属性值建立值索引
输入参数：datasetID dataset 编号
docID   doc 编号
输出参数：0 表示正确，其他表示错误

函数名称：CreateValueIndex（int datasetID, int docID, DTDNodeCode &nodeid）
函数说明：对一个给定文档上的指定节点或属性建立索引
输入参数：datasetID dataset 编号
docID   doc 编号
nodeid   指定节点或属性的编码
输出参数：0 表示正确，其他表示错误

函数名称：DropValueIndex（int datasetID, int docID）
函数说明：释放一个文档对应的所有节点或属性的值索引
输入参数：datasetID dataset 编号
docID   doc 编号
输出参数：0 表示正确，其他表示错误

函数名称：DropValueIndex（int datasetID, int docID， DTDNodeCode &nodeid）
函数说明：释放对应文档上指定节点的索引
输入参数：datasetID dataset 编号
docID   doc 编号
nodeid   指定节点或属性的编码
输出参数：0 表示正确，其他表示错误

函数名称：GetHandler(int datasetID, int docID, DTDNodeCode &eID,
OperatorType op,CConstVaslue &value,
ValueNodeHandler &handler)
函数说明：获得指定节点或属性的索引所记录的 ElementNode List，并将结果存入 handler
输入参数：datasetID dataset 编号
docID   doc 编号
eID   指定节点编码
op   指定的过滤操作符（>,<,=,!=,>=,<=）

value　　操作符 op 对应的比较常量
handler　返回值的容器，存储了对应的 List 的首页号
输出参数：0 对应成功，其他值表示错误（没有建立索引等）

4.3 BP-Tree 部分 API 说明

BP-Tree 模板类的两个模板参数 KeyType 分别对应 Key 类型和 Value 类型，在模块中分别由 IndexRecord 和 ValueRecord 的子类表示。

函数名称：Open(int datasetID, int pageno, int leafpageno, int keylen, int vallen)
函数说明：根据参数来初始化 BP-Tree 内部结构
输入参数：datasetID　dataset 编号，是 BP-Tree 申请数据页对应的数据集
pageno　BP-Tree 根节点数据页号
leafpageno　BP-Tree 第一个叶节点的数据页号
keylen　BP-Tree 的 key 长度
vallen　BP-Tree 的 value 长度
输出参数：

函数名称：Close（）
函数说明：清理 BPTree
输入参数：
输出参数：

函数名称：Insert（KeyType &key, ValueType &val）
函数说明：插入关键字为 key，值为 val 的元组
输入参数：key 关键字
Val 对应的索引值
输出参数：0　表示插入成功
-1 表示插入失败
-2 表示 key 值重复

函数名称：Search（KeyType &key, ValueType *&val）
函数说明：返回满足 key 的 value
输入参数：key 关键字
Val 对应的索引值
输出参数：0　　表示查找成功
非 0　表示查找失败
函数名称：Search（KeyType &key, ValueType *&val, int &nodeno, int &nodeindex）
函数说明：返回满足 key 的 value, 以及 key 所在的叶子的页号和第一个大于等于 key 的元组位置
输入参数：key　　关键字
Val　　对应的索引值
nodeno　key 所在的叶子的页号
nodeindex 大于等于 key 的第一个元组的 index

输出参数：0　　　表示查找成功
　　　　　非 0　表示查找失败

函数名称：Delete（KeyType &key, ValueType *&val）
函数说明：删除 key 对应的元组
输入参数：key 关键字
　　　　　Val 返回对应的索引值
输出参数：0　　　表示删除成功
　　　　　非 0　表示删除失败

Range Query 接口说明：

```
int FindEqualRecord(KeyType &key, NxdbList &retlist);
int FindNotEqualRecord(KeyType &key, NxdbList &retlist);
int FindLessRecord(KeyType &key, NxdbList &retlist);
int FindGreatRecord(KeyType &key, NxdbList &retlist);
int FindLtEqualRecord(KeyType &key, NxdbList &retlist);
int FindGtEqualRecord(KeyType &key, NxdbList &retlist);
```

分别对应（=, !=, <, >, <=, >=）六种操作，retlist 是返回结果的 list
返回 0 是成功，其他是不成功

5. 后续工作