



清华大学

数据库系统概论

数据库课程项目报告

作者：

桥本优、钱雨杰、董胤蓬

指导教师：

冯建华、张勇

清华大学

计算机科学与技术系

January 3, 2016

Contents

1	项目任务	1
2	系统结构设计	2
2.1	系统架构	2
3	模块设计	3
3.1	记录管理模块	3
3.2	系统管理模块	5
3.3	查询解析模块	5
3.4	索引模块	5
3.5	扩展功能	7
3.5.1	域完整性约束	7
3.5.2	外键约束	7
3.5.3	模糊匹配	8
3.5.4	三个表以上的连接	8
3.5.5	聚集查询	8
3.5.6	图形用户界面	8
4	实验成果	9
5	小组分工	10
6	实验总结	11

Chapter 1

项目任务

本项目是清华大学计算机科学与技术系开设的《数据库系统概论》课程项目。

项目的任务是实现一个单用户的关系数据库管理系统。该项目分为四个功能模块：

- (1) 记录管理模块：该模块是 DBMS 的文件系统，管理存储数据库记录以及元数据的文件。该模块依赖于我们预先给定的一个页式文件 I/O 系统，在此基础上扩展而成。
- (2) 索引模块：为存储在文件中的记录建立 B+ 树索引，加快查找速度。
- (3) 系统管理模块：实现基本的数据定义语言（DDL），实现解析器来解析命令行。
- (4) 查询解析模块：解析 SQL 语句，能将输入的 SQL 语句解析成关系代数表达式，并生成查询执行计划，访问文件系统执行查询，输出查询结果。

在上述功能的基础上，还可以对该系统进行个性化的功能扩展及性能优化，内容包括但不限于：

- (1) 查询优化：基于对查询计划代价的估计，为给定查询选择最有效的查询执行计划。
- (2) 支持属性域约束和外键约束。
- (3) 创建数据表时支持更多的数据类型。例如 decimal, date 等。
- (4) 支持三个或以上表的连接。
- (5) 支持更多 SQL 语句。例如聚集查询 AVG, SUM, MIN, MAX, GROUP BY 等。
- (6) 支持模糊查询。例如 LIKE 关键字以及“%, *, ?”等通配符。
- (7) 提供类似 MySQL front 的图形化 UI。

Chapter 2

系统结构设计

2.1 系统架构

Chapter 3

模块设计

3.1 记录管理模块

记录管理模块主要负责在数据库中创建、打开、删除一个数据表，在表中插入、删除、更新一条记录，以及遍历表找到所有符合条件的记录。

我们在项目中把每张数据表存在独立的文件中。文件均由页式文件系统管理，每个数据表文件的第一页是表文件头页，记录整个数据表的一些信息；之后的页均为数据页，每个数据页的页头记录该页的一些信息。

具体地，表文件头页按以下格式组织：

```
struct FileHead
{
    int recordSize;      //记录长度
    int pageNumber;     //页的个数
    int recordPerPage;   //每页记录个数
    int recordNumber;    //记录的总数
};
```

表文件头页中依次记 4 个整数，分别表示该表中记录长度、页的个数、每页的记录个数和表中记录总数。文件头页其余位置留空，可以用作其他扩展。

数据页每页 8KB，前 96 byte 为页头，记录该页的一些信息，之后依次是固定长度的槽，每个槽可以放一条记录。数据页头按以下格式组织：

```
struct PageHead
{
    int usedSlot;        //已用槽的个数
    char slotMap[84];    //每个槽的状态4
};
```

数据页头首先存放一个整数，记录本页已经放了几条记录。之后 84 byte 中每一个 bit 依次对应之后每个槽的状态，0 表示为空，1 表示已经有记录。

本模块的实现主要包括 3 个类，分别是 RM_Manager、RM_FileHandle 以及 RM_FileScan。在实现各个功能时，使用和维护之前所述的文件头、页头的信息。

```
class RM_Manager
{
public:
    RM_Manager      (FileManager *pfm, BufPageManager* bpm);
    int CreateFile   (const char *fileName, int recordSize);
    int DestroyFile  (const char *fileName);
    int OpenFile     (const char *fileName, RM_FileHandle &fileHandle);
    int CloseFile    (RM_FileHandle &fileHandle);
};
```

RM_Manager 提供的接口包括：(1) 新建一个文件，生成表文件头页；(2) 删除一个文件；(3) 打开一个文件，得到一个 RM_FileHandle，可以通过其进行对记录的操作；(4) 关闭一个文件。

```
class RM_FileHandle
{
public:
    RM_FileHandle   ();
    ~RM_FileHandle  ();
    int GetRec       (const RID &rid, RM_Record &rec) const;
    int InsertRec    (const char *pData, RID &rid);
    int DeleteRec    (const RID &rid);
    int UpdateRec    (const RM_Record &rec);
};
```

RM_FileHandle 提供的接口包括：(1) 根据 RID 获得记录内容；(2) 插入一条记录，返回位置标识 RID；(3) 根据 RID 删除一条记录；(4) 更新一个记录（知晓 RID）。

```
class RM_FileScan {
public:
    RM_FileScan(FileManager *pfm, BufPageManager* bpm);
    ~RM_FileScan();
    int OpenScan   (const RM_FileHandle &fileHandle,
                    AttrType attrType,
                    int attrLength,
                    int attrOffset,
                    CompOp compOp,
                    void *value);
    int GetNextRec (RM_Record &rec);
    int CloseScan() {return 0;}
};
```

RM_FileScan 提供的接口包括：(1) 打开一个 Scan，指定数据类型、长度、位置、约束条件；(2) 获得一条符合条件的记录；(3) 关闭一个 Scan。

3.2 系统管理模块

3.3 查询解析模块

3.4 索引模块

索引模块的功能是为表中的某一属性建立索引，提高查询速度。

我们项目中索引使用 B+ 树的数据结构实现，每个索引存放在一个独立的文件中。索引文件的命名方式是 `RelName.AttrName.index`。索引文件由页式文件系统管理，有三种页，分别是索引文件头页、中间节点页、叶子节点页。

具体地，索引文件头页按以下方式组织：

```
struct IX_FileHead
{
    AttrType attrType;
    int attrLength;
    int maxN;
    int pageNum;
    int firstEmptyPage;
    int root;
};
```

索引文件头页中记录索引数据的类型、长度、每页最多放多少索引项、已使用页的数量、根节点页等。

之后的每个页页头也需要记录一些内容，包括

```
struct IX_PageHead
{
    IX_PageType type; // 0 for node, 1 for leaf
    int n;
};
```

`type` 表示该页是中间节点还是叶子节点，`n` 表示该页当前有的索引项个数。

中间节点页与叶子节点页的数据结构是相同的，如下图所示。



其中 P 为指针， K 为索引码值。区别在于，如果是中间节点，指针指向子节点（索引文件内的一个页），即 P 存储的是本文件的一个 RID；如果是叶子节点，每个

索引码值对应一条数据记录，P 存储的是该记录在数据表页中的 RID；叶子节点最后一个索引码值后面的指针 P 指向后面一个叶子节点，这样一个索引文件中存储的所有索引项按顺序可以形成一个列表。

索引的插入、删除、查询等功能按照 B+ 树的要求实现。

索引模块的实现仿照记录管理模块,主要包括 3 个类,即 IX_Manager、IX_IndexHandle 以及 IX_IndexScan。

```
class IX_Manager
{
public:
    IX_Manager (FileManager *pfm_, BufPageManager* bpm_)
        : pfm(pfm_), bpm(bpm_) {}
    ~IX_Manager ();
    int CreateIndex (const char *fileName,           // Create new index
                    const char *indexName,
                    AttrType attrType,
                    int attrLength);
    int DestroyIndex (const char *fileName,          // Destroy index
                     const char *indexName);
    int OpenIndex   (const char *fileName,          // Open index
                     const char *indexName,
                     IX_IndexHandle &indexHandle);
    int CloseIndex  (IX_IndexHandle &indexHandle); // Close index
private:
    FileManager *pfm;
    BufPageManager *bpm;
};
```

IX_Manager 提供的接口包括新建索引、打开索引、关闭索引和删除索引。

```
class IX_IndexHandle
{
public:
    IX_IndexHandle ();
    ~IX_IndexHandle ();
    int InsertEntry (void *pData, const RID &rid); // Insert new index entry
    int DeleteEntry (void *pData, const RID &rid); // Delete index entry
    int ForcePages  ();                          // Copy index to disk
    void PrintEntries ();
    int GetLowerBound (void *pData, RID &indexid);
    int GetUpperBound (void *pData, RID &indexid);
    int GetFirst      (RID &indexid);
};
```

IX_IndexHandle 提供的接口包括插入索引项、删除索引项，以及为了支持查询提供首项、LowerBound 和 UpperBound 的查询。B+ 树的主要功能实现在这个类中完成。


```

class IX_IndexScan
{
public:
    IX_IndexScan () : data(NULL) {}           // Constructor
    ~IX_IndexScan () {}                       // Destructor
    int OpenScan (IX_IndexHandle &indexHandle, // Initialize index scan
                CompOp compOp,
                void *value);
    int GetNextEntry (RID &rid);              // Get next matching entry
    int CloseScan ();                         // Terminate index scan
};

```

IX_IndexScan 提供的接口和 RM_FileScan 完全一致，区别在于 IX_IndexScan 的查询操作是通过索引在 B+ 树上完成的，效率更高。

其他的模块在执行新建索引以及插入、删除、更新记录的同时，要更新相关的索引。

3.5 扩展功能

3.5.1 域完整性约束

域完整性约束，即建表时有 CHECK 关键字，约束某一个属性可能的取值。

为实现这一功能，我们对每个有域完整性约束的属性，新建一个和索引文件一样的约束文件，命名方式为 RelName.AttrName.check.index，也用 IX_Manager 管理。在这个文件中，将该属性可能的取值作为索引项依次插入。

插入记录时，检查所有有域完整性约束的属性值是否在其约束文件中存在。如果存在，允许插入，否则拒绝并报错。

3.5.2 外键约束

外键约束，用于与另一张表的关联，以保持数据的一致性。外键一定是另一张表的主键，因此一定是建好索引的。

所以，在更新一个有外键约束的记录时，打开关联表的主键索引，检查更新后的值是否存在。如果存在，允许更新，否则拒绝并报错。

3.5.3 模糊匹配

3.5.4 三个表以上的连接

3.5.5 聚集查询

3.5.6 图形用户界面

图形用户界面使用 Qt 实现，在文本框中输入 SQL 语句，点击后通过命令行执行，然后将输出的结果以表格形式显示。

Chapter 4

实验成果

Chapter 5

小组分工

Chapter 6

实验总结