《数据库系统概论》项目实验报告

2012011303 计 23 李国豪
2012011350 计 24 周琳钧

一、 项目概述

本实验中我们实现了一个简单的关系型数据库，其中完成了全部的基本指令，和四种扩展：自己实现底层模块、索引表、聚集查询和分组聚集查询，并且对上课所讲过的单链表记录管理方式做出了一些改进，变成双链表结构，节约了存储空间，本项目共包含 39 个文件和 4100 行代码，是目前本科生涯中最为庞大的一项工程，极大的锻炼了我们的系统设计能力。在实现过程中，我们克服了许多困难，包括队友中期退课，时间不充裕，结构设计不合理等等，但我们最终都解决了它们，并完成了整个项目。虽然困难重重，但对我们的代码能力和设计能力有了极大程度的提高，这也是完成本项目的收获。

二、 编译&使用说明

开发环境：

Ubuntu

编译说明：

提供 Makefile 文件，使用 make 命令进行编译；

编译环境：Linux 系统，g++、Flex / Bison

使用说明：

除了标准的 SQL 语句之外，额外提供”quit”命令以退出程序

提供两种形式的使用方式

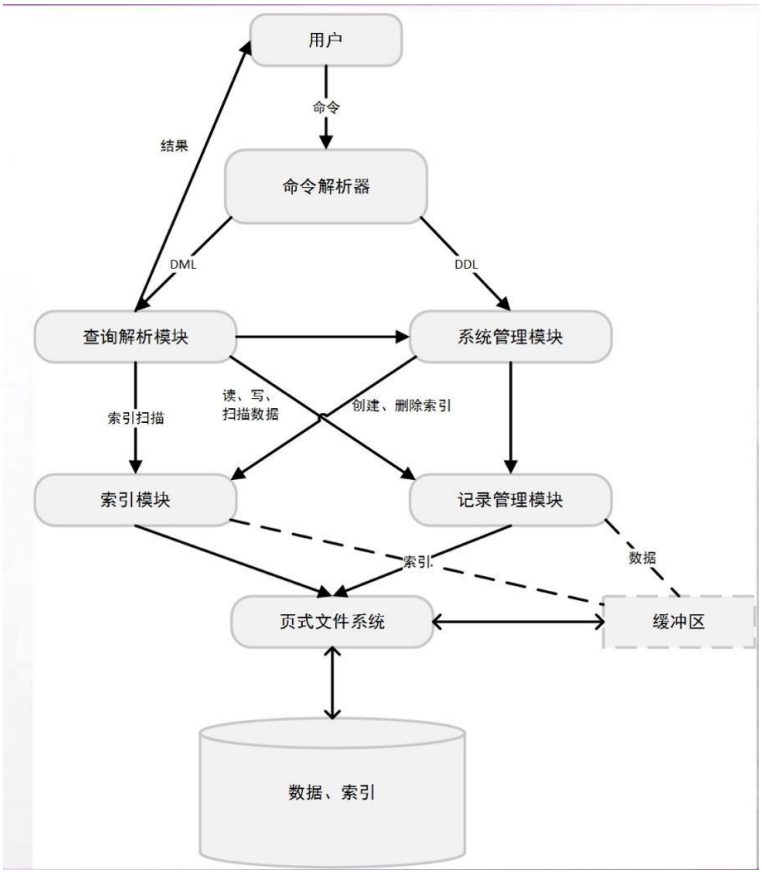
1、 无参数

默认以交互方式运行

2、 一个参数，为 sql 文件路径

注意在最后需要使用 quit 命令并且要求文件末尾有一个换行。

三、 系统结构设计



四、 主要模块设计

1. 错误状态列表（DBError.h 和 DBUtility.h）

错误号	错误原因
0 (DBOK)	程序正常运行
1 (CREATEFILEFAILURE)	无法创造文件
2 (FILEEXIST)	文件已存在
3 (FILENOTEXIST)	文件未存在
4 (FILENOTOPEN)	文件无法打开
5 (NOTSETFILEHEADER)	文件未设置文件头
6 (NOSUCHATTRIBUTE)	该文件中不包含此属性
7 (RECORDLENGTHERROR)	记录长度不符
8 (PRIMARYKEYERROR)	主键重复
9 (RECORDNOTEXIST)	记录不存在
10 (PAGEIDOVERFLOW)	页 ID 超出范围
11 (RIDOVERFLOW)	记录 ID 超出范围
12 (NOSUCHATTR)	该文件中不包含此属性
13 (ATTRNAMETOOLONG)	属性名称过长（>31）
14 (FOREIGNATTRNAMETOOLONG)	外键属性名称过长
15 (FOREIGNFILENAMETOOLONG)	外键文件名过长
16 (DBEXISTS)	数据库已存在

17 (DBNOTEXISTS)	数据库不存在
18 (MULTIPLEDBCREATE)	在一个数据库中新建另一个数据库
19 (CREATEFILEPATHERROR)	文件路径错误
20 (ATTRNUMNOTEQUAL)	属性数量不符合该文件要求
21 (DBNOTCHOOSE)	未指定数据库
22 (NOSUCHRECORD)	无相应记录
23 (CAUCULATETYPEERROR)	聚集查询错误
24 (NORECORDMATCH)	无对应匹配记录
25 (KEYNULL)	待插入记录某属性值为空
26 (TYPEERROR)	带插入记录某属性值类型不合要求

错误函数调用过程：

DBPrintErrorPos(char*)：输出出错地点

DBPrintError(int)：根据错误号输出错误信息

2. 数据组织模块和记录管理模块

（1） 数据记录

a. **表结构**（DBFile.h 和 DBFileInfo.h）

表头（10KB）

- int recordLength: 记录总长度
- int attrNum： 属性数量
- DBAttribute attr[attrNum]: 属性
  - char* name: 属性名称
  - char* foreignKeyFileName：外键文件名称
  - char* foreignKeyAttr：外键属性名称
  - int length：属性长度
  - int offset：属性偏移量
  - int type：属性种类（0 为字符串，1 为整型，2 为空）
  - bool isNull：是否可以为空
  - bool isPrimary：是否为主键
  - bool isForeign：是否为外键
- int firstNotFullPageId：第一个非满的页
- int pageNum：页的数量
- bool isEmpty：该文件是否为空

表内容

b. 页结构（DBPage.h 和 DBPageInfo.h）

页头（64B）

- bool isNull：该页是否为空
- int firstEmptySlot：该页第一个空槽
- int pageID：该页的页号
- int slotNum：该页的槽数
- int nextEmptyPage：下一个为空的页

页内容（8128B）——记录槽

c. 记录结构（DBRecord.h）

记录头（8B）

- int nextEmptySlot：本页中下一个空槽
- bool isNull：该记录是否被删除

记录（根据文件头的 offset 可获得每一属性的值）

（2） 缓存管理（DBBufManager.h）

a. 概述

本实验中实现了一个简单的缓存管理机制，为方便起见，我们的缓存单位是文件，并采用了改进的 LRU 算法。本数据库维护了一个 100 个 DBFile 的列表作为缓存，当对文件进行改动时会在内存中进行改动，直到用户调用了 quit 命令或缓存文件超出系统能记录的范围时数据才会写回到文件系统中。

b. 改进的 LRU 算法

当缓存文件数大于系统承载力时面临着要将缓存文件写回，那么应该选择哪一个文件去写回呢？我们的策略是每一个缓存文件维护一个时间戳，每次进行文件改动的时候更新时间戳。当需要写回的时候，选择那个离当前时间最远的那个文件写回（即最早被改动的那个文件）。

c. 缓存管理过程——DBBufManager 类结构

成员变量：

- int bufnum：缓存文件个数
- map<char*, int> bufmap：文件名-缓存文件序号二元组集合
- map<int, char*> bufinvmap：缓存文件序号-文件名二元组集合
- vector<time_t> visittime：时间戳
- vector<bool> isNull：缓存文件是否为空列表

成员函数：

int AddBuf(char* filename)

作用：为 filename 新开启一个 Buf

int DeleteBuf()

作用：写回并删除一个时间戳较早的 Buf

int DeleteBufById(int fileid)

作用：定点写回并删除一个 Buf

int SearchBuf(char* filename)

作用：通过文件名查找缓存文件序号

int Fixopertime(int fileid, time_t opertime)

作用：调整某缓存文件时间戳

int AllWriteBack()

作用：将全部缓存文件写回到磁盘。

（3） 缓存管理（DBBufManager.h）

a. 概述

文件管理有一系列函数操控着文件的生灭过程。

b. 管理过程

int CreateFile(char* filename)

作用：创造一个文件

int DestroyFile(char* filename)

作用：删除一个文件

int OpenFile(char* filename)

作用：打开一个文件

int CloseFile(char* filename)

作用：关闭一个文件

DBFileInfo* getFileHeader(char* filename)

作用：获取文件头

int SetFileHeader(DBFileInfo* fileinfo, char* filename)

作用：设置文件头

（4） 记录管理（DBFile.h）

a. 概述

记录管理是数据库中最核心的部分，它涉及到对记录的底层操控，本实验中我们改进了 SQL 中的链表实现记录插入、更改、删除的过程，使整个数据库节约了空间，并加快了记录操作的性能，此外，我们还实现了聚集查询的功能。

b. 双链表记录管理原理

在 SQL 中，记录管理原理如下图所示：

这样，需要在文件头和记录中都设置一个（pageid，rid）对来存储下一条待插记录位置，消耗空间为 2*4（int）*recordNum，这样造成了信息冗余，在我们的设计中，只需消耗约 4*recordNum 空间即可。

我们的原理是维护两个链表，其中横向链表是指未满页链表（存放在页头），纵向链表是指一页中的空槽链表（存放在记录头），具体的实现原理和维护方法类似上述单链表操作，但稍麻烦一些，可以参考程序，此处不再赘述。

c. 查询过程

查询的标准形式是：CondEntry opA CondEntry opA CondEntry…，其中 CondEntry 为 attr opB keyvalue，opA 为等于、大于、小于三种，opB 为与、或两种。查询过程为：如果为单属性查询，通过索引表返回满足条件的（pageid，rid）集合，如果是多属性查询，则通过分解属性遍历表中每条记录得到每个属性的满足情况，再根据 opA 连接从而得知该条记录是否满足整个条件，最后返回（pageid，rid）集合。

d. 管理过程

char* getPage(int page)

作用：通过页号获得页头位置

char* getRecord(int page, int rid)

作用：通过页号和记录号获得记录头位置

int SearchRecord(char** keyattr, int* style, int* oper, char** keyword, int paranum, vector< pair<int, int> >& re)

作用：查询记录，参数分别代表属性名称，查询单个属性方式（等于，大于，小于），查询属性间连接符（与、或），查询属性对应关键词，查询属性个数， 返回结果列表的引用。

int AddRecord(char* record, int length)

作用：添加一条记录

int DeleteRecord(int pageid, int rid)

作用：删除一条记录

int UpdateRecord(char* keyattr, char* keyword, int pageid, int rid)

作用：更新一条记录

int show(vector< pair<int, int> >rlist, vector<int>attrindex)

作用：文件标准打印输出

int calculate(vector< pair<int, int> >rlist, char* attrname, int mode, int &resultflag, int &resultInteger, char* &resultLiteral)

作用：聚集查询和分组聚集查询

3. 系统管理模块（OrderPack.h, OrderPack.cpp）

（1） 命令转换（OrderPack.cpp）

命令常量	作用
CREATE_DB	创造一个新数据库
DROP_DB	删除一个数据库
USE	进入某数据库
CREATE_TB	创造一个表
DROP_TB	删除一个表

SHOWDBS	展示所有数据库
SHOWTBS	展示某数据库中所有表
DESC	展示某个表结构
INSERT	插入记录
DELETE	删除记录
UPDATE	更新记录
SELECT	查询记录
CREATE_INDEX	创建一个索引
DROP_INDEX	删除一个索引

针对每个指令，我们将顶层包装好的变量集合通过转换，解析成一个一个底层函数需要的参数，在通过调用底层函数完成相应操作

（2） 顶层解析-底层操作接口（OrderPack.h）

OrderType type：命令种类
string dbname：数据库名
string tbname：表名
Schema schema：模式属性
vector<Value> values：值集合
bool allAttrs：查询时是否为全属性查询
vector<Attr> attrs：属性列表
vector<string> tables：查询时表列表
Condition condition：查询条件
Attr groupbyAttr：聚集查询属性
string updateAttr：记录待更新属性
Value updateValue：记录待更新值
string indexAttr：索引属性

4. 命令解析模块

顶层的命令解析模块使用 Flex / Bison 开源工具，足以解析基础和扩展功能要求中的所有给出的语法形式。

(1) 词法分析

定义了 30 个左右的关键字，列举如下

OPERATOR	(","|"."|"+"|"-"|"*"|"/"|"("|")"|"="|"<"|">"|";")
WHITESPACE	([\t]+)
NEWLINE	(\r|\n|\r\n)
INTEGER	([0-9]+)
IDENTIFIER	([A-Za-z][_0-9A-Za-z]*)
QUIT	("quit")
CREATE	("CREATE"|"create")
DB	("DATABASE"|"database")
DBS	("DATABASES"|"databases")
DROP	("DROP"|"drop")

USE	("USE"|"use")
SHOW	("SHOW"|"show")
TBS	("TABLES"|"tables")
TB	("TABLE"|"table")
INDEX	("INDEX"|"index")
DESC	("DESC"|"desc")
NOT	("NOT"|"not")
IS	("IS"|"is")
NUL	("NULL"|"null")
IN	("IN"|"in")
PRIMARY	("PRIMARY"|"primary")
FOREIGN	("FOREIGN"|"foreign")
KEY	("KEY"|"key")
CHECK	("CHECK"|"check")
REFER	("REFERENCES"|"references")
INS_INTO	("INSERT INTO"|"insert into")
VALUES	("VALUES"|"values")
DELETE	("DELETE"|"delete")
WHERE	("WHERE"|"where")
UPDATE	("UPDATE"|"update")
SET	("SET"|"set")
SELECT	("SELECT"|"select")
FROM	("FROM"|"from")
LIKE	("LIKE"|"like")
AND	("AND"|"and")
OR	("OR"|"or")
SUM	("SUM"|"sum")
AVG	("AVG"|"avg")
MAX	("MAX"|"max")
MIN	("MIN"|"min")
GRP_BY	("GROUP BY"|"group by")
INT	("INT"|"int")
CHAR	("CHAR"|"char")
VCHAR	("VARCHAR"|"varchar")

(2) 语法分析

规约时使用的产生式如下所示

```
Program :
        Program Stmt
      | /* empty */
      ;
```

```
Stmt :
        ENDLINE
      | QUIT ENDLINE
      | CREATE DB IDENTIFIER ';' ENDLINE
      | DROP DB IDENTIFIER ';' ENDLINE
      | USE IDENTIFIER ';' ENDLINE
      | SHOW DBS ';' ENDLINE
      | SHOW TBS ';' ENDLINE
      | CREATE TB IDENTIFIER '(' AttrDefList ')' ';' ENDLINE
      | DROP TB IDENTIFIER ';' ENDLINE
      | DESC IDENTIFIER ';' ENDLINE
      | INS_INTO IDENTIFIER VALUES '(' ValueList ')' ';' ENDLINE
      | DELETE FROM IDENTIFIER WhereClause ';' ENDLINE
      | SELECT AttrList FROM TableList WhereClause GroupClause ';' ENDLINE
      | UPDATE IDENTIFIER SET IDENTIFIER '=' ValueItem WhereClause ';' ENDLINE
      | CREATE INDEX IDENTIFIER '(' IDENTIFIER ')' ';' ENDLINE
      | DROP INDEX IDENTIFIER '(' IDENTIFIER ')' ';' ENDLINE
      | error ENDLINE
        ;

WhereClause :
        /* empty */
      | WHERE CondList
      ;
GroupClause :
        /* empty */
      | GRP_BY Attr
      ;
AttrDefList :
        AttrDefList ',' AttrDefItem
      | AttrDefItem
      ;
AttrDefItem :
      IDENTIFIER Type '(' INTEGER ')'
      | IDENTIFIER Type '(' INTEGER ')' NOT NUL
      | PRIMARY KEY '(' IDENTIFIER ')'
      | CHECK '(' CondList ')'
      | FOREIGN KEY '(' IDENTIFIER ')' REFER IDENTIFIER '(' IDENTIFIER ')'
        ;
Type :    INT
      | CHAR
      | VCHAR
      ;
```

```
ValueList :
        ValueList ',' ValueItem
      | ValueItem
        ;
ValueItem :
        INTEGER
      | LITERAL
      | NUL
      ;
CondList :
        CondList AND Cond
      | CondList OR Cond
      | Cond
      ;
Cond :
        Expr '=' Expr
      | Expr '>' Expr
      | Expr '<' Expr
      | Attr IS NUL
      | Attr LIKE LITERAL
      | Attr IN '(' ValueList ')'
      ;
Expr :
        Attr
      | INTEGER
      | LITERAL
      ;
AttrList :
        '*'
      | AttrList ',' AttrAggr
      | AttrAggr
      ;
AttrAggr :
        Attr
      | SUM '(' Attr ')'
      | AVG '(' Attr ')'
      | MAX '(' Attr ')'
      | MIN '(' Attr ')'
      ;
Attr :
        IDENTIFIER
      | IDENTIFIER '.' IDENTIFIER
      ;
```

```
TableList :
    TableList ',' IDENTIFIER
  | IDENTIFIER
  ;
```

以上产生式中，忽略了规约时执行的语义计算代码。

这里使用的是采用 LR 分析技术进行 S-属性文法的语义计算。简单的说，语义计算代码片段基本上是在做一件事情：生成语法分析树节点，并将综合属性沿语法分析树向上传递。

语法分析树中的节点应包含所有的综合属性（语法信息），bison 中默认是 YYSTYPE 类型（其实是 int 类型），在这里重新定义成类 SemValue.

(3) 错误处理与类型检查

当输入错误命令（无法按照某一个产生式进行规约）时，期望的行为是放弃这一条错误命令，给出报错信息并等待新的命令输入。bison 里面“error”是一个保留的终结符，对应的编号是 256。有“error”存在的产生式，会丢弃之前的输入符号的方式强行归约。使用产生式 stmt： error ENDLINE。当错误发生时，其一定会使用此产生式归约。程序中利用这一点实现实现期望的错误处理行为。

命令解析模块中，仅对语句内部进行粗糙的类型检查，更加细致的类型检查（比如插入记录是否和该表的 Schema 匹配）则留给底层的模块进行处理。

5. 索引模块

使用 B+树，对表中的 Primary 属性进行快速检索和修改，支持检索关键字为数字或者字符串类型，支持点查询和区间查询，支持快速的插入查找和删除。

(1) BNode　B 树节点

成员变量

```
enum NodeType{LEAF, STEM}; // 两种类型的 B 树节点
static int MaxSize;   // 程序中取值为 50
NodeType type;   // 节点类型
 int number; // 节点编号
std::vector< pair<std::string, int> > values;   //
int pNextLeaf; // 下一个叶子节点编号，仅在叶节点中有效
int pPreLeaf; // 上一个叶子节点编号，仅在叶节点中有效
int pFirstNode; //指向的第一个子节点，仅在非叶节点中有效
int parent; // 父节点编号
```

(2) BTree　B 树

成员变量

```
enum KeyType { DIGIT, STRING }; // 根据键值的不同分为两种树
int root;　　// 根节点编号
KeyType type;　// 树的类型
std::vector<BNode*> ptrs; // 树中所有的节点
```

方法

```
/* 根据树的键值类型的不同，函数均有两个版本 */
pair<int, int> search(int);
pair<int, int> search(const string&);
/* 搜索一段记录，分为相等、大于、小于三种模式 */
std::vector< pair<int, int> > search_zone(int, int);
std::vector< pair<int, int> > search_zone(const string&, int);
int insert(int, pair<int, int>);
int insert(const string&, pair<int, int>);
int remove(int);
int remove(const string&);
```

实现原理

查找：由于 B 树类似平衡二叉树的性质，可以 O(logn)递归找到其可能所在的叶节点，然后顺序遍历查找。

删除：这里采取了非常简单的做法，仅仅为删除项打上标记，而并非真正在 B 树中移除，其位置可以在之后为新插入的节点所占据。

插入：插入数据的过程比较复杂，首先也是找到其应该在的叶节点，如果插入后 B 树节点超过了指定的上限，该叶节点将进行分裂，并且可能引起其父节点的连锁分裂。

(3) IndexManager 索引（B 树）管理

成员变量

```
std::map<std::string, BTree*> trees;
```

方法

```
int storeIndex(const std::string&, BTree*);
void addBTree(const std::string&, BTree*);
int removeBTree(const std::string&);
BTree* getBTree(const std::string&);
int insertRecord(DBFileInfo*, char*, int, int);
int deleteRecord(DBFileInfo*, char*);
int updateRecord(DBFileInfo*, char*, char*, char*);
```

五、 实验结果

本工程在 src 目录下使用 make 编译。使用./main 运行程序。

1. **文件管理**

   可正确运行下列文件中的代码，包含创建表，插入数据等

   src/data/drop.sql

   src/data/create.sql

   src/data/book100.sql

   src/data/order100.sql

2. **DML**

   show databases

   use [DatabaseName]

   show tables

   desc [FileName]

```
~/Course/Database/SQL/src$ ./main

 sql> show databases;

 lgh
 orderDB


 sql> use orderDB;

 sql> show tables;

 orders
 customer
 book
 publisher


 sql> desc customer;
id: int(4) Not Null PRIMARY
name: char(25) Not Null
rank: int(4) Not Null
```

3. **record_operation**

   实现功能：

   insert into [filename] values …

   delete from [filename] where Condition

   update [filename] set … where Condition

```
 sql> use orderDB;

 sql> insert into orders values(315000, 200001, 'eight');
Insert Resolution: Error: The data's type cannot match the attribute's type..

 sql> insert into customer values(307001, 'chad cabello', 3);

 sql> delete from publisher where nation = 'USA';
Delete Record: Error: Delete record failed. No record matched..

 sql> update book set title = 'nine times nine' where id = 200001;

 sql> select title from book where id = 200001;

                  title|
         nine times nine|

 sql> delete from book where id = 200001;
```

4. **空值检测**

实现功能：not null

```
sql> use lgh;

sql> create table NULT (id int(10) not null, name varchar(20));

sql> insert into NULT values (NULL, 'haha');
Insert Record: Error: There is at least one NULL in your record, but the attributes don't support NULL..
```

5. **主键检测**

实现功能：primary key

```
sql> use lgh;

sql> create table NT (id int(10), name varchar(20), primary key (id));

sql> insert into NT values (233, 'huhu');

sql> insert into NT values (233, 'lala');
Add record: Error: Create record failed. The record's primary key is equal to some else in this document..
```

6. **单条件 select**

实现功能：

select * from [filename] where Condition

```
sql> use orderDB;

sql> select * from book where id < 200010;

    id|                   title|                 authors|publisher_id| copies
|
200002|  Standing in the Shadows|     Richard Bruce Wright| 101787|   2900|
200003|  Children of the Thunder|            Carlo DEste| 102928|   3447|
200004|  The Great Gilly Hopkins|         Gina Bari Kolata| 101339|     39|
200005|  Meine Juden--eure Juden|          E. J. W. Barber| 103089|    206|
200006|  You Can Draw a Kangaroo|                 Amy Tan| 101850|   5296|
200007|  The Little Drummer Girl|           Robert Cowley| 104382|   1006|
200008|  A Walk Through the Fire|             Scott Turow| 102008|   8795|
200009|  The Nursing Home Murder|          David Cordingly| 102866|   7380|

sql> select authors from book where id = 200006;

               authors|
               Amy Tan|
```

7. **多条件 select**

实现功能：

select [Attrname] from [filename] where ConditionGroup

```
sql> use orderDB;

sql> select id, title, copies from book where id < 200100 and id > 200090 and copies > 2000;

    id|                   title| copies|
200091|    T??dliche Versuchung.|   6794|
200092|  A Suitable Boy: A Novel|   6228|
200094|  Autumn of the Patriarch|   8673|
200096|  Diet for a Small Planet|   6576|
200097|  Beneath a Midnight Moon|   5814|
200098|  Deadlines and Datelines|   8459|
200099|   Gulac War (Sobs, No 6)|   9538|
```

8. **两表连接**

实现功能：select [Attrname] from [fileA], [fileB] where ConditionGroup

```
~/Course/Database/SQL/src$ ./main data/create.sql
~/Course/Database/SQL/src$ ./main data/book100.sql
~/Course/Database/SQL/src$ ./main data/order100.sql
~/Course/Database/SQL/src$ ./main

 sql> use orderDB;

 sql> select book.title, orders.quantity from book, orders where
 book.id = orders.book_id and orders.quantity > 8;

              book.title|orders.quantity|
  Survival for Busy Women|      9|
The Ransom of Russian Art|     10|
Das Theorem des Papageis.|     10|
```

9. **索引管理**

实现功能：

create index([Attrname])

drop index([Attrname])

```
~/Course/Database/SQL/src$ ./main

 sql> use orderDB;

 sql> create index book(id);

 sql> select id, title from book where id > 202995;
BTree Searching book.id

     id|                    title|
 202996| Los Ricos Son Diferentes|
 202997| Wringer (Trophy Newbery)|
 202998| Cat Who Went Underground|
 202999| Die Landkarte der Liebe.|

 sql> insert into book values (334455, 'SQL', 'lgh', 223, 11);
BTree Searching book.id

 sql> update book set id = 304050 where id = 202997;
BTree Searching book.id
BTree Searching book.id

 sql> delete from book where id = 202998;
BTree Searching book.id

 sql> select id, title from book where id > 202995;
BTree Searching book.id

     id|                    title|
 202996| Los Ricos Son Diferentes|
 202999| Die Landkarte der Liebe.|
 304050| Wringer (Trophy Newbery)|
 334455|                     SQL|

 sql> drop index book(id);
```

10. **聚集查询**

实现功能：

select SUM([Attrname]) from …

select AVG([Attrname]) from …

select MAX([Attrname]) from …

select MIN([Attrname]) from …

```
sql> select SUM(quantity) from orders;

539

sql> select AVG(quantity) from orders;

5

sql> select MAX(quantity) from orders;

10

sql> select MIN(quantity) from orders;

0
```

11. 分组聚集查询

实现功能：

select … group by…

```
sql> insert into ll values("aa", 8);

sql> insert into ll values("bb", 12);
Syntax Error

sql> insert into ll values("bb", 12);

sql> insert into ll values("bb", 7);

sql> select * from ll;

                    name|     id|
                      aa|      5|
                      aa|      8|
                      bb|     12|
                      bb|      7|

sql> select name, SUM(id) from ll group by name;

aa  :  13
bb  :  19
```

12. 运行时情况说明

本工程内存消耗较大，我们尝试向一个文件中插入 3000 条记录，可以插入，但插入 50000 条的时候内存会爆掉，因此插入大数据时请通过多次运行程序分成较小块插入。

就性能而言，由于采用了索引结构，插入性能非常好， 在插入 3000 条数据时可以在 1 秒之内运行完成，搜索时，我们采用的方案是单属性时用索引，多属性时用遍历，尽可能保证其运行速度，在两表连接时，我们用的是双表遍历判断，因此速度较慢，大约 100 记录*100 记录的两表连接运行时间为 1s。

六、 小组分工

1. **周琳钧**
   - **(1) 底层模块、记录解析模块**
     - DBBufManager.h
     - DBBufManager.cpp
     - DBError.h
     - DBError.cpp
     - DBFile.h
     - DBFile.cpp
     - DBFileInfo.h
     - DBFileManager.h
     - DBFileManager.cpp
     - DBPage.h
     - DBPage.cpp
     - DBPageInfo.h
     - DBRecord.h
     - DBRecord.cpp
     - DBUtility.h
     - Global.h
     - Global.cpp
   - **(2) 系统管理模块**
     - OrderPack.cpp
2. **李国豪**
   - **(1) 命令解析模块：**
     - Attr.h
     - CondEntry.h
     - Condition.h
     - Condition.cpp
     - Expr.h
     - Lexer.l
     - OrderPack.h
     - Parser.y
     - Schema.h
     - Schema.cpp
     - SchemaEntry.h
     - SemValue.h
     - SemValue.cpp
     - Value.h

(2) 索引模块

BNode.h

BNode.cpp

BTree.h

BTree.cpp

IndexManager.h

IndexManager.cpp

七、 项目工程链接

https://github.com/lgh303/SQL