NineSQL 数据库报告

2013011413　计 35　王梓仲　2013011393　计 35　冯栩

一、实验目的

通过对数据的实现深入理解数据库的基本实现。了解对于命令的解析过程以及数据库底层的各种功能的实现过程，同时也通过这个过程提升我们的工程管理能力。

二、实验分工

王梓仲：命令解析器，解析查询模块的解析部分，系统管理模块。
冯栩：记录管理模块，索引模块，解析查询模块的查询部分。

三、实验详细

1、记录管理模块

此模块中包含了 DB_File 一个基本类，用于处理底层关于文件的各种操作。每一个 DB_File 类对应数据库中的一张表。

对于数据库表的存储，第 0 页存储各种主要信息，例如每一个 key 的基本信息（名称，类型，长度，是否为主键，是否 Not Null 等），之后的每一页都存储行的信息，页的第一位存储本页的结尾地址。而行的第一位存储该行是否为空。关于每一行的 SID，由于课程提供的管理系统有对应的页码，而每一行对应的在提供的 Buftype 数组中也有对应的位置，因此可以使用（page，index）的这样一个对来当做 SID 码。

关于表的使用：

新建：使用 createFile 创建表，之后使用 openFile 打开表，再使用 initFile 传入表的基本信息即可完成创建。

使用：使用 openFile 打开表，使用 resetKeys 将基本信息从文件中读取到内存里（加快速度），之后即可使用其它函数完成对于数据库表的操作。

删除：调用 deleteFile 即可完成对于表的删除。

由于当初设想的时候是将所有的对于行的解析转换实现在中层，因此底层部分的实现就极为简单。以下为有关数据表的实现：

每当插入操作时使用 addData 传入已经解析完成的数据即可。每次插入时从第一页向后寻找，直到找到某一页的末尾可插入的地方为止。

当删除操作时，调用 deleteData 传入一个（page，index）对，删除该条目，并将该页的最后一条转移到该位置上。

当更新操作时，使用 insertData 传入已经解析完的整条数据以及一个 SID，之后直接覆盖。

搜索操作时，传入需要查询的信息，之后底层完成搜索操作后范围包含一个 SID 数组的结果。

使用 getAllPiece 可以得到整个数据库中的所有行的 SID；

使用 getPiece 可以通过 SID 获得某一行。

可使用 testFile 输出所有行的信息，showStores 输入基本信息。

2、系统管理模块

此模块中包含了 DBT 和 DB 两个类。

DB 类是对针对每一个不同的数据库进行管理的类，一个数据库对应一个 DB。

在这一层结构中，DB 类的函数可以将当前的数据库名整理成系统中的绝对路径，连同之前解析好的各种数据传入底层，实现数据库内 create table、drop table、show tables 及对数据库中指定 table 的 desc 功能。

DB 类中会存储其数据库内所有 table 连成的链表，具体命令的对象（某个 table）的确定就是根据链表中的查询实现。由于每次重新运行程序前可能已经存在之前导入的数据库，于是提供了一个函数（initDB()）能够对数据库内的文件进行解析并将相应信息存入链表中。

DBT 类是对整个数据库系统进行管理的类，运行一次程序便对应着一个 DBT。

在这一层结构中，DBT 类的函数主要分为两大类型。对于 create database、drop database、use、show databases 这些功能，应直接在这一层完成。对于 create table、drop table、show tables、desc 这些功能，应根据目前所 use 的数据库传入到对应 DB 的相应函数中处理。

DBT 类中会记录所有数据库所在文件夹的绝对路径以及当前时刻 use 的数据库的名字和绝对路径。另外，DBT 类中会存储所有数据库连成的链表，上述的第一类函数中具体命令的对象（某个 database）的确定就是根据链表中的查询实现。而上述的第二类函数则是根据当前时刻 use 的数据库的名字在链表中查询，以确定调用具体哪一个 DB 类指针。由于每次重新运行程序前可能已经存在之前导入的数据库，于是在 DBT 类的构造函数中会为所有已经存在的数据库新建对应的 DB 并调用 initDB()函数，最后将 DB 加入链表中。

3、索引模块

我们所编写的数据库所使用的索引加速的技术为 B+树（加入了连接到 silbing 的标志），由于本课程已经给出了相应的文件管理模块，因此所编写的 B+树同样使用此模块尽心有关文件的管理。

由于和文件直接相关联的 B+树与使用指针完成的 B+树在很多地方都会有不同，因此在编写的时候我直接使用了记录管理模块中的 SID 来表征每一个 B+树节点的地址，并使用 getNode 和 writeBackNode 两个函数的联动来模拟指针的操作。

我所编写的 B+树，每一颗都对应一个 BPlusTree 的实例。

关于 B+树的存储，第 0 页存储根节点的信息以及现在已经获得的最大页码，之后每页第一位存储当前页的最大地址，剩下的用于存储节点的具体信息。

关于 B+树的具体实现是将网上找到的源码进行了自己的重构并且加以修改之后实现的。

关于索引与原数据库的链接，在经过网上查询之后是有两种方式，一种是按照主键进行建立 B+树并存储每一行的信息，而其他的键就建立一颗映射到相应地址的 B+树。而另一种就是将所有的键都建立成映射到地址的 B+树。由于之前记录管理模块的实现方式并咩有设计 B+树，因此选择了后面

的一种方式来建立索引。

由于在查询使可能会涉及到一些范围查询，因此我在编写 B+树的搜索时编写了两种搜索方式，一种是查询到等于 k 的第一个位置，另一种是找到大于 k 的第一个位置。这样一来就能够将所有的范围查找实现了。

同时，因为会涉及到查询某一行所对应在 B+树的位置以进行操作（更新、删除等），因此还编写了一个通过表中每行的 SID 查询到所在 B+树位置的函数。

对于字符串的索引值，我采用将其前四位取出（不足补 0）的方式，按照 256 进制换算成一个整数进行插入。

在将 B+树编写完成后对于原始的记录管理模块进行了修改。由于设想中既然要加速就应该加速得彻底一些，因此我对于每一个表在一开始就建立了对于所有键的索引。每当插入、删除、更新行的时候，直接更新所有的索引树。而查找也由原来的暴力查找改为了根据 B+树实现的查找。具体效率会在第四大块中体现。

4、命令解析模块

解析部分：

使用 flex 进行词法分析器的生成，使用 GNU Bison 进行语法分析器的生成。

对于词法分析，应先将不同类型（包括数字、运算符、关键字、可行的命名等）的词进行定义。对于关键字，为了方便调试时命令的输入，将其定义为大小写均能识别。具体添加的定义如下所示。

```
NEWLINE   (\r|\n|\r\n)
DIGIT     ([0-9])
INTEGER       ({DIGIT}+)
IDENTIFIER    ([A-Za-z][_0-9A-Za-z]*)
OPERATOR(","|"."|"*"|"("|")"|"="|"<"|">"|";")
WHITESPACE    ([ \t]+)
EXIT          ("EXIT"|"exit")
LTE           ("<=")
GTE           (">=")
NEQ           ("<>")
CREATE        ("CREATE"|"create")
DB        ("DATABASE"|"database")
DBS           ("DATABASES"|"databases")
DROP      ("DROP"|"drop")
USE           ("USE"|"use")
SHOW      ("SHOW"|"show")
TBS           ("TABLES"|"tables")
TB        ("TABLE"|"table")
DESC      ("DESC"|"desc")
NOT       ("NOT"|"not")
IS        ("IS"|"is")
NULL      ("NULL"|"null")
```

```
IN          ("IN"|"in")
PRIMARY     ("PRIMARY"|"primary")
FOREIGN         ("FOREIGN"|"foreign")
KEY             ("KEY"|"key")
CHECK       ("CHECK"|"check")
REFER       ("REFERENCES"|"references")
INSERT          ("INSERT"|"insert")
INTO        ("INTO"|"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")
INT             ("INT"|"int")
CHAR        ("CHAR"|"char")
VCHAR       ("VARCHAR"|"varchar")
```

对于语法分析，关键是语法规则的定义。约定所有命令以分号";"结尾，因此如果出现错误命令，会在检测到分号后出现提示，并过滤掉该分号及它和上一个分号间的所有输入。语法规则的定义如下所示（忽略语义动作）。

```
Program             :   Program Stmt
                    |   %empty
                    ;
Stmt                :   error ';'
                    |   EXIT ';'
                    |   CREATE DB IDENTIFIER ';'
                    |   DROP DB IDENTIFIER ';'
                    |   USE IDENTIFIER ';'
                    |   SHOW DBS ';'
                    |   SHOW TBS ';'
                    |   CREATE TB IDENTIFIER '(' AttrDefList ')' ';'
                    |   DROP TB IDENTIFIER ';'
                    |   DESC IDENTIFIER ';'
                    |   INSERT INTO IDENTIFIER VALUES ValueListList ';'
                    |   DELETE FROM IDENTIFIER WhereClause ';'
                    |   SELECT AttrList FROM TableList WhereClause ';'
```

```
                | UPDATE IDENTIFIER SET SetList WhereClause ';'
                ;
WhereClause     :   %empty
                | WHERE CondList
                ;
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
                ;
ValueListList   :   '(' ValueList ')'
                | ValueListList ',' '(' ValueList ')'
                ;
ValueList       :   ValueList ',' ValueItem
                | ValueItem
                ;
ValueItem       :   INTEGER
                | LITERAL
                | NUL
                ;
SetList         :   IDENTIFIER '=' ValueItem
                | SetList ',' IDENTIFIER '=' ValueItem
                ;
CondList        :   CondList AND Cond
                | CondList OR Cond
                | Cond
                ;
Cond            :   Expr '=' Expr
                | Expr '>' Expr
                | Expr '<' Expr
                | Expr LTE Expr
                | Expr GTE Expr
                | Expr NEQ 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
                ;
```

在 GNU Bison 中，语法分析树中的节点默认是 YYSTYPE 类型；根据程序实现的需要，将其改为自定义的 SemValue 类型。对于每一个解析完成的语句（Stmt），直接根据语句的类型生成一个 OrderPack，并调用 OrderPack 类中 process()函数进行语句的执行。由于在生成语法树节点时，综合属性也会向上传递，因此在 process()函数执行时，各模块需要的解析部分均顺势完成。

查询部分：

在本部分中使用另一个 DataDealer 类进行一些辅助性的操作。

关于插入、删除、更新等操作，基本就是对于原语句进行简单的解析之后直接调用底层代码进行插入即可。

关于 Select 语句，基本设想是先做完 where 语句的查找拿到所有的语句，之后调用 DataDealer 的 SelectColumn 函数。该函数的作用是将选定的行以及选定的数据插入到一张临时的新表当中去并对该表进行输出。

关于两表连接的部分，一开始的做法是直接对于两表计算笛卡尔积存入一个临时的新表，之后再对于该张表做 where 语句。后来发现这样的写法应该是过不了测例中的两表连接的，因此改为了先建立新表，在插入数据的过程中做第一条 where 语句，这样一来尽管时间很慢，但是确实能够将测例中对于两表连接的处理出来。

关于 SUM，AVG，MIN，MAX 在处理完 where 语句之后直接判断是否有这四个操作即可，之后对于 where 建立出来的新表进行聚集查询操作之后输出即可。

关于内部约束以及外部约束，都采用了在查询模块的高层进行文件的创

立以及检查（在更新及删除时），这样一来能够减少对于底层代码的修改。

关于 Like 的实现，是在检测到其中是 Like 语句时对目标字符串进行模糊查找。关于模糊查找的实现：1.首先将字符串中的字母都转成大写。2.之后如果用户输入的字符串并不是特别短（长度小于 4），判断其是否为正在比对的字符串的子串，如果是则返回相似。3.利用动态规划算法求解两字符串的 Levenshtein 距离。将 Levenshtein 距离与这两个字符串中较长一个的长度的比值与一个阈值进行比较，如果不超过这个阈值则返回相似。在本程序中，阈值暂定为 0.25。4.若上述情况均不满足，则返回不相似。这样一来就能够求出某一个串的相似串。

四、功能介绍

基础部分：

系统管理模块的功能全部实现。

查询解析模块的功能全部实现。

附加部分：

B+树索引模块（对于完成的 book 表进行单条查询，原时间为 1.401s，添加了 B+树索引后为 0.188s）。

域完整性约束。

外键约束。

Like 模糊查找。

SUM，AVG，MIN，MAX 聚集查询。

五、实验心得

最大的心得就是前期沟通的不顺畅导致后期的工作量激增，一开始编写的 DataDealer 到最后还在使用的函数也就只有一部分。其他的则是在最初对于语句理解的不够深入导致的功能缺失，因此在后期只能通过不断的增加新功能来达到最后的实现要求。总体来说这次大作业还是让我们学到了很多东西。

六、附件说明

src 下为工程的源码。

文件（夹）名称	功能简介
bufmanage/ fileio/ utils/	课程提供的文件管理系统
dataset_small/	课程提供的命令集
makefile	makefile
wf_file_manager.h	DB_File 类所在头文件
sys_manager.h	DBT,DB 类所在头文件
command_data.h	DataDealer 类所在头文件
bplus_tree.h	BPlusTree 类所在头文件
Lexer.l	flex 对应的文件
Parser.y	bison 对应的文件
其余	为解析部分对应的各类文件