《高级数据库系统》

系统功能拓展项目文档

实验名称：	新数据类型及完整性约束支持
姓　　名：	季津天，黄文浩
学　　号：	22120385，22120381
日　　期：	2022.6.10

目 录

一、系统介绍

1.1 DBMS 整体结构

数据库管理系统（Database Management System，DBMS）是一种用于管理和组织数据的软件系统。它提供了一种结构化的方式来存储、操作和检索数据，使用户能够方便地访问和管理数据库。DBMS 中共有 5 个模块，按照其工作响应流程从上往下分别是：查询解析模块（QL, Query Language Component）、系统管理模块（SM, System Management Component）、索引模块（IX, IndexingComponent）、记录管理模块（RM, Record Management Component）、页式文件系统模块（PF, Paged File Component）。

1) QL 模块主要实现对 select、update、delete 和 insert 操作的处理功能（DML 语言）；
2) SM 模块主要维护数据库系统与数据库的模式（Schema），实现数据库、表、索引的创建与删除操作；
3) IX 模块主要是为了加快查询的速度，通过为表的特定属性创建 B+树索引来实现；
4) RM 模块主要为其他模块提供记录处理功能，主要实现是创建与删除数据表、将数据表的记录写进底层文件系统、对记录进行必要的修改删除等操作；
5) PF 模块是 DBMS 最底层的模块，主要功能是为其他模块进行数据存储的操作。

因此一个单用户的 DBMS 的整体工作流程如下：用户输入 SQL 语句→命令解析器解析命令→DML 命令输入 QL 模块/DDL 命令输入 SM 模块。命令传递至 QL 模块后，调用 RM、IX、SM 模块的相应接口；命令传递至 SM 模块后，调用 RM 或 IX 模块；RM 与 IX 同级，均调用 PF 模块，PF 模块与缓冲池进行交互。其整体流程如图 1 所示：

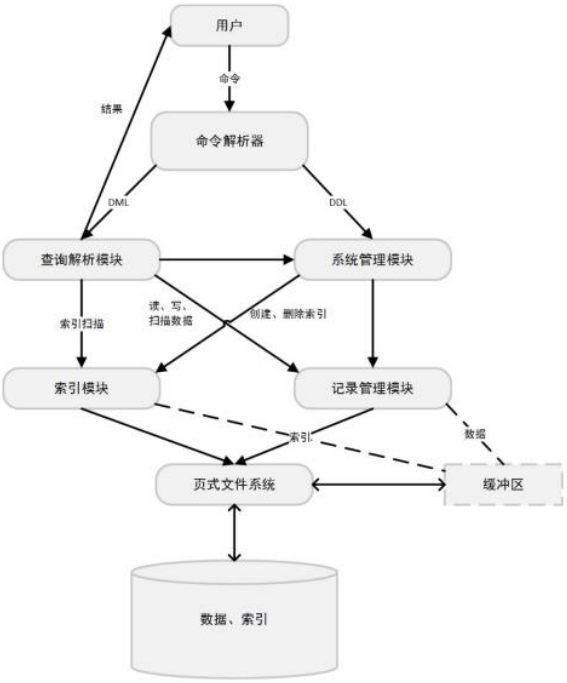

图 1 单用户 DBMS 整体结构

1.2 各模块介绍

1.2.1 PF 模块

A. 模块原理

PF 模块是 DBMS 最底层的模块，主要功能是为其他模块进行数据存储。它能够将数据与索引存储到某一个空闲的磁盘块中，或者根据其他模块的操作要求将某个磁盘块的数据或索引提取出来。

在本系统中，页式文件系统模块提供给其他模块对文件（file）和页（page）的管理功能，其他模块需要通过调用它的相关函数创建并打开文件，然后进行后续操作。需要注意的是，DBMS 的底层都是以 page 为读写操作单位，我们可以将一个文件（file）理解为一张表，file 中有很多 page，每个 page 中放很多条记录（record）。

B. 模块工作流程

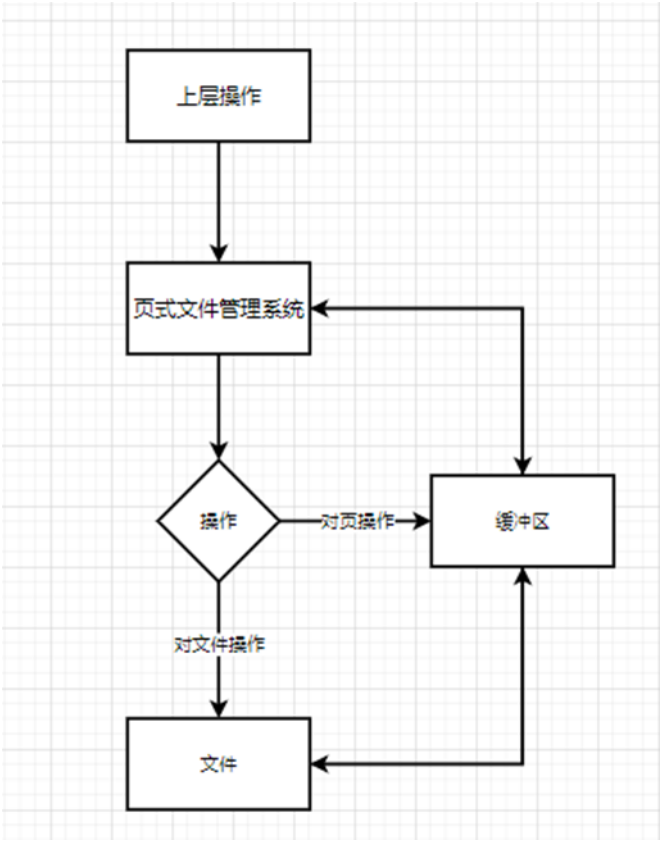

图 2 PF 模块工作流程

C. 模块结构体和主要类

该模块中主要有 PF_Manager、PF_FileHandle、PF_PageHandle、PF_HashTable 和 PF_BufferMgr 五个类。

- PF_Manager 的主要功能是初始化 PF_Manager 对象、创建及删除
- file、打开及关闭 file 等操作，在打开 file 时需要传入一个 PF_FileHandle 对象；
- PF_FileHandle 的主要功能是具体处理 file 中的操作，在 PF_Manager 类中新建打开一个 file 后，通过该类进行 file 中 page 的获取、创建、删除、标记 dirty 等；
- PF_PageHandle 的主要功能是处理具体某个 page 中的操作，针对某个特定 page 进行获取信息等；
- PF_HashTable 的主要功能是记录缓存池中的 file 的位置，便于通过一个 page 的 fd 和 pageNum 迅速的在缓存池中找到它；PF_BufferMgr 的主要功能是管理录缓存池，进行缓冲池的页面读入、分配、标记、写回等操作。

1.2.2 RM 模块

A. 模块原理

RM 理模块在 DBMS 中的位置相较 PF 模块来说更为上层。PF 模块侧重在磁盘中操作文件，主要针对文件中的 page 进行操作。而 RM 模块侧重通过一个文件来操控一张数据表的内容（包括各种参数、记录等），负责给索引模块、系统管理模块、查询解析模块提供记录处理的功能。其主要任务是创建与删除数据表、将数据表的记录写进底层文件系统、对记录进行必要的修改删除等操作。

B. 模块工作流程

在这一部分，我们总结记录管理模块的相关工作的流程，首先由上面的介绍可知, 每一个**记录**都用一个 **rid** 唯一表示，因此数据的增删改都是通过修改对应的 **rid**；其次查询记录涉及到条件查询，会独立出一个查询器类（RM_FileScan）来开发，通过提供调用接口来实现数据表的查询。对此，我们分别总结记录管理模块的相关操作的流程，主要是记录的添加，删除，修改和查询，具体流程如下：

a) **添加记录：**

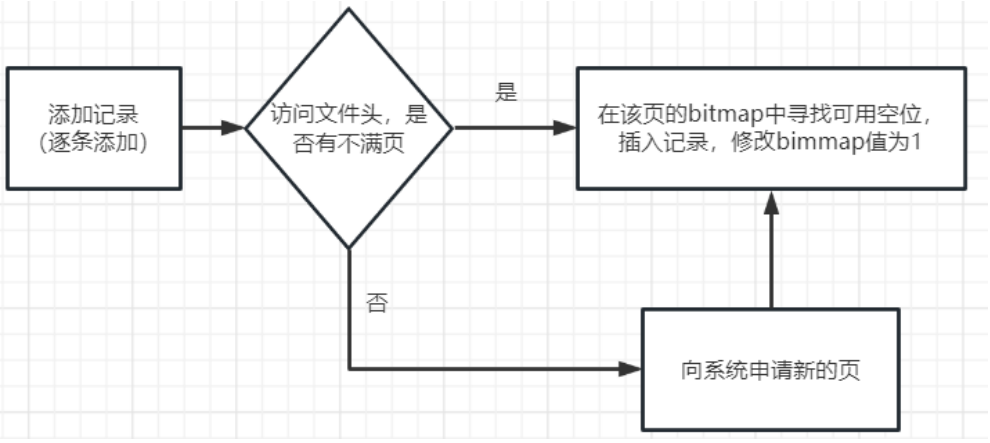

图 3 添加记录流程图

b) **删除记录：**利用要删除记录的 rid, 找到这条记录的位置，删除并将记录的 bitmap 值置为 0。

c) **修改记录：**利用要删除记录的 rid, 找到这条记录的位置，修改记录内容。

d) **查询记录：**

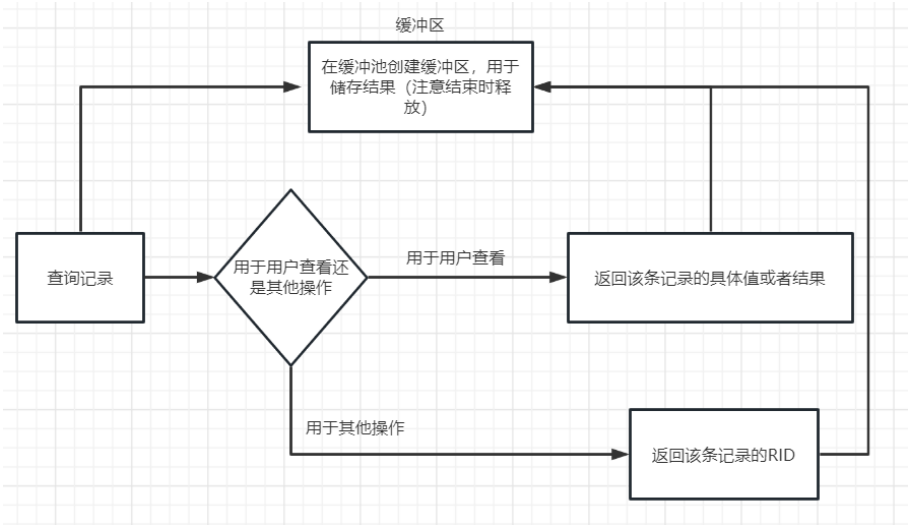

图 4 查询记录的流程图

C. 模块结构体和主要类

该模块主要涉及到数据表、记录、缓冲区的相关操作，因此这一模块主要有 RM_RID、RM_Record、RM_FileHandle、RM_FileScan、 RM_Manager 等类。

- RM_Manager 的主要功能是初始化 RM_Manager 对象，进行创建、删除、打开、关闭 file 等操作；
- RM_FileHandle 的主要功能是具体处理 file 中的操作，通过该类进行 file 中 record 的获取、插入、删除、更新等；
- RM_FileScan 的主要功能是扫描 record，返回满足条件的 record，主要用在上层进行条件查询操作时；
- RM_Record 为 record 类，主要功能为初始化一条具体的 record、修改一条 record 的具体数据等；
- RM_RID 为 record 的 id 类，主要功能为初始化一个 record 的 id，包括 record 所在的页号与槽号。

1.2.3 IX 模块

A. 模块原理

在 DBMS 中，IX（索引）模块与 RM（记录管理）模块处于相同的级别，相对于页式文件系统（PF）来说更高层次。IX 模块的整体工作原理与 RM 模块类

似，都是通过与 RF（关系模式）模块进行交互来完成特定功能。IX 模块的主要功能是加快查询速度，它通过为表的特定属性创建 B+树索引来实现。IX 模块依赖于底层的 PF 模块，利用 PF 模块提供的接口进行索引文件的存储和调用。索引文件中存储着属性值与记录 ID 的对应关系，使用 B+树来加速对属性值的查找。每个索引文件对应一棵 B+树，在 PF 模块中对应一个文件（file），B+树的每个节点对应 file 中的一个页（page）。在本系统中，索引模块的主要功能包括为指定属性创建或删除单独的索引文件、打开或关闭索引文件，以及在索引文件中快速定位满足条件的记录等操作。通过索引模块，可以提高查询效率和加速数据检索过程。

B. 模块工作流程

(一) 初始化对象、创建及打开索引文件

在程序最初运行时，首先初始化 IX_Manager 对象及 PF_Manager 对象，并调用 PF_Manager 的各方法创建、打开、初始化（为 file 分配 page）索引文件。其中需要注意的是，创建 file 时与 RM 模块一样，需要开辟两页的空间用于初始化 header；打开文件时传入 IX_IndexHandle 对象。上文已经介绍过，IX 与 RM 模块类似，在这里均不需要直接与缓冲区交互。

(二) **对特定索引文件进行操作**

主要是通过 IX_IndexHandle 对象对索引文件进行各类操作。首先初始化 B+点，为其根结点分配 page，然后是进行如下不同的操作

1) B+树插入节点

a) 寻找保存键值的叶子结点；
b) 应插入结点已满时，申请新结点；
c) 同时调整应插入但未插入结点中的键值记录，使其均衡存放于两个叶结点中；
d) 调整指针使其指向新叶子结点；
e) 寻找指向新叶子结点的非叶结点；
f) 若应插入结点已满，则申请新结点，同时调整应插入但未插入结点的键值记录，使其均衡存放于两个非叶结点中，调整各结点指针使其指向正确，依次向上调整，直到根结点。

2) B+树删除节点

a) 叶子结点中寻找等于键值的记录，删除相应的指针及主文件中对应的记录；

此时一种情况下执行：

b) 调整其左侧或右侧结点及本届点中的键值记录，使其均衡存放于两个叶结点中；

c) 如有调整，则进一步调其上层非叶结点，重新确定其键值，以满足大于等于键值的记录都在其右侧；

另一种情况下执行：

b) 将本结点的剩余键值，移动到左侧或右侧结点，此空结点将被删除，调整叶结点指针；

c) 如有调整，则进一步调整其上层非叶结点，重新确定其键值，以满足大于等于键值的记录都在右侧，依次向上调整，直到根结点。

需要注意的是吗，在实现的 DBMS 中，不再使用标记删除发，当删除某个属性值时，将会在叶子结点对应的索引块中查找，若找到，则将该块内其后的元素前移，覆盖被删除键值。

C. 模块结构体和主要类

该模块中主要有 IX_Manager、IX_IndexHandle、IX_IndexScan 三个类。

- IX_Manager 的主要功能是管理索引文件，通过初始化 IX_Manager 对象，进行创建、删除、打开、关闭索引文件等操作；
- IX_IndexHandle 的主要功能是具体处理对索引的操作，增加、删除 B+树结点等；
- IX_IndexScan 的主要功能是扫描索引项，根据指定条件对索引进行一次查询。

1.2.4 SM 模块

A. 模块原理

系统管理模块（SM, System Management Component）是数据库管理系统（DBMS）中负责管理数据库系统的核心组件之一，他的位置在记录管理模块（RM）和索引模块（IX）的上层。其在 DBMS 中的整体工作基本是通过调用 RM 与 IX 模块实现的。

系统管理模块维护数据库系统与数据库的模式（Schema），主要实现了如下功能：调用系统关于文件目录的命令，实现创建、删除数据库等操作；调用 RM 模块的相关函数，实现创建、删除表等操作；调用 IX 模块的相关函数，实现创建、删除索引等操作；维护 relcat、attrcat 两个基础信息文件。

B. 模块工作流程

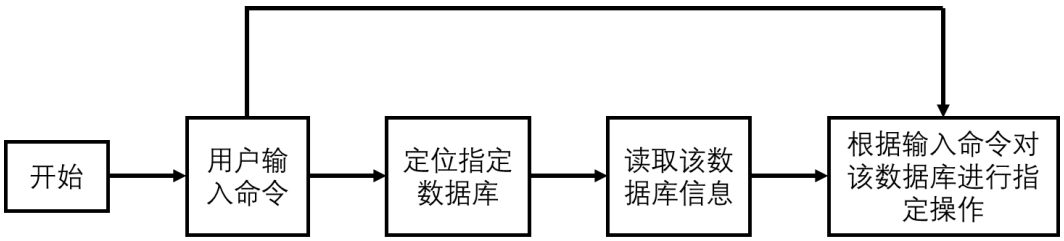

图 5 SM 模块工作流程图

C. 模块结构体和主要类

该模块中主要有 SM_Manager、Statistics、StatisticsMgr、Printer、LinkList 五个类.

- SM_Manager 的主要功能为管理数据库系统，通过初始化全局的唯一一个 SM_Manager 对象，实现创建、删除数据库等操作，实现创建、删除表等操作，实现创建、删除索引等操作，实现维护 RelCatEntry、AttrCatEntry 基础信息文件的操作；
- Statistics 的主要功能是统计信息;
- StatisticsMgr 的主要功能是跟踪一组统计信息；
- Printer 的主要功能为打印，输出用户需要的内容；
- LinkList 主要提供链表用以存储统计信息。

1.2.5 QL 模块

A. 模块原理

查询解析模块（QL, Query Language Component）在 DBMS 中的位置相较记录管理模块（RM）、索引模块（IX）、系统管理模块（SM）来说都更为上层。其在 DBMS 中的整体工作基本是通过调用 RM、IX 与 SM 模块实现的。

查询解析模块对用户的命令生成查询结果，主要实现对 select、update、delete 和

insert 操作的处理功能（DML 语言）。对于插入、删除和更新操作，本模块会调用 RM 模块对物理磁盘上的数据进行更新，并对改变的数据表进行检查，如果修改的属性上有索引的话会调用 IX 模块对该属性的索引进行相应更新。

B. 模块工作流程

QL 模块的工作流程可以用如下流程图表示：

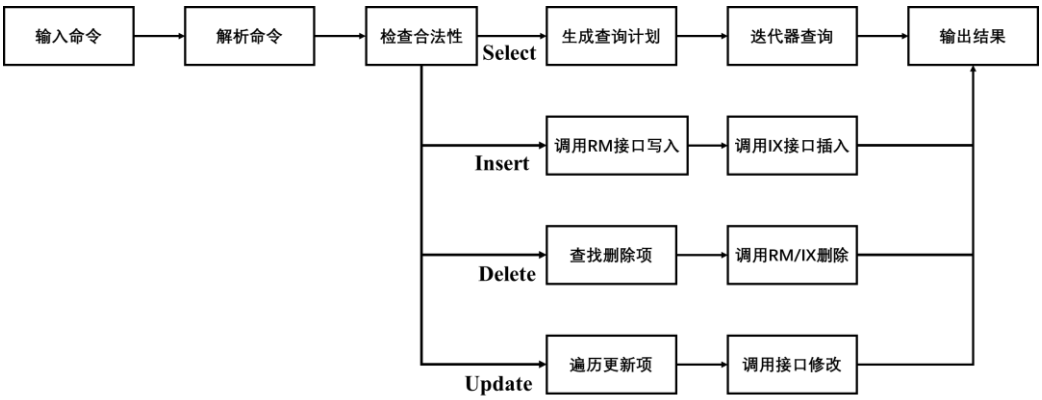

图 6 QL 模块工作流程图

C. 模块结构体和主要类

本模块中主要有 QL_Manager、QL_Iterator、QL_FileScanIterator、QL_SelectionIterator、QL_ProjectionIterator、QL_IndexSearchIterator、QL_NestedLoopJoinIterator、QL_IndexedJoinIterator、QL_DisjointSet 九个类。

- QL_Manager 的主要功能为提供查询解析操作，通过初始化全局的唯一一个 QL_Manager 对象，实现用户输入的查询、插入、删除、修改等操作；
- QL_Iterator 的主要功能是派生迭代器类以执行查询计划，
- QL_FileScanIterator、QL_SelectionIterator、QL_ProjectionIterator、QL_IndexSearchIterator、QL_NestedLoopJoinIterator 及 QL_IndexedJoinIterator 均为 QL_Iterator 的派生类；
- QL_DisjointSet 主要实现并查集数据结构，在生成查询计划时使用。

二、指定拓展功能

2.1 支持更多的数据类型

在这一模块我们组实现了 DECIMAL, VARCHAR, DATE, DATETIME 等 4 个新的数据类型，修改的代码如下：

A. 修改 redbase.h 文件

在该文件中增加这四种数据类型：

```
enum AttrType {
    INT,
    FLOAT,
    STRING,
    DATE,
    DATETIME,
    VARCHAR,
    DECIMAL,
};

enum ValueType {
    VT_NULL,
    VT_INT,
    VT_FLOAT,
    VT_STRING,
    VT_DATE,
    VT_DATETIME,
    VT_VARCHAR,
    VT_DECIMAL,
};
```

B. 修改 PARSE

- **首先修改 scan.l 中的正则式**，以确保 DATE 的输入类型为’ 2023-11-23’这种形式，且 DATETIME 的输入形式为’2023-11-23 9:00:00’这种形式，增加的代码如下：

```
\'{digit}{4}\-{digit}{1,2}\-{digit}{1,2}\'    {   yylval.sval = get_qstring(yytext, yyleng);
                                    printf("T_DATE: %s \n",yylval.sval);
                                    return T_DATE;}

\'{digit}{4}\-{digit}{1,2}\-{digit}{1,2}[ ]{digit}{1,2}\:{digit}{1,2}\:{digit}{1,2}\' {
                                    yylval.sval = get_qstring(yytext, yyleng);
                                    printf("T_DATETIME: %s \n",yylval.sval);
                                    return T_DATETIME;}
```

- **然后修改 parse.y**，主要是为了增加符号表示和增加四种类型的输出：

符号表示：

```
%token   <ival>   T_INT

%token   <rval>   T_REAL
         T_DECIMAL

%token   <sval>   T_STRING
       T_QSTRING
       T_SHELL_CMD
       T_DATE
       T_DATETIME
       T_VARCHAR
```

针对四种数据类型的符号进行赋予值：

```
 Value
: T_QSTRING
   {
     $$ = value_node(STRING, (void *) $1);
   }
  | T_DATE
   {
     $$ = value_node(DATE, (void *) $1);
   }
  | T_DATETIME
   {
     $$ = value_node(DATETIME, (void *) $1);
   }
  | T_INT
   {
     $$ = value_node(INT, (void *)& $1);
   }
  | T_REAL
   {
     $$ = value_node(FLOAT, (void *)& $1);
   }
  | T_DECIMAL
   {
     $$ = value_node(DECIMAL, (void *)& $1);
   }
  | T_VARCHAR
   {
     $$ = value_node(VARCHAR, (void *)& $1);
   }
```

下面三个都是输出时增加这四种类型数据的输出。

```
ostream &operator<<(ostream &s, const DataAttrInfo &ai)
{
  return
    s << " attrName=" << ai.attrName
    << " attrType=" <<
    (ai.attrType == INT ? "INT" :
     ai.attrType == FLOAT ? "FLOAT" :
     ai.attrType == DECIMAL ? "DECIMAL" :
     ai.attrType == STRING ? "STRING" :
     ai.attrType == VARCHAR ? "VARCHAR" :
     ai.attrType == DATE ? "DATE" : "DATETIME"
     )
    << " attrDisplayLength=" << ai.attrDisplayLength;
}
```

```
ostream &operator<<(ostream &s, const Value &v)
{
  s << "AttrType: " << v.type;
  switch (v.type) {
    case INT:
      s << " *(int *)data=" << *(int *)v.data;
      break;
    case FLOAT:
      s << " *(float *)data=" << *(float *)v.data;
      break;
    case DECIMAL:
      s << " *(float *)data=" << *(float *)v.data;
      break;
    case STRING:
      s << " (char *)data=" << (char *)v.data;
      break;
    case VARCHAR:
      s << " (char *)data=" << (char *)v.data;
      break;
    case DATE:
      s << " (char *)data=" << (char *)v.data;
      break;
    case DATETIME:
      s << " (char *)data=" << (char *)v.data;
      break;
    default:
      break;
```

```
28.    }
29.    return s;  }
30. ostream &operator<<(ostream &s, const AttrType &at)
31. {
32.   switch(at){
33.     case INT:
34.       s << "INT";
35.       break;
36.     case FLOAT:
37.       s << "FLOAT";
38.       break;
39.     case DECIMAL:
40.       s << "DECIMAL";
41.       break;
42.     case STRING:
43.       s << "STRING";
44.       break;
45.     case VARCHAR:
46.       s << "VARCHAR";
47.       break;
48.     case DATE:
49.       s << "DATE";
50.       break;
51.     case DATETIME:
52.       s << "DATETIME";
53.       break;
54.   }
55.   return s;  }
```

- **然后修改 node.cc,** 修改 Value_node 分配函数，增加四种数据类型的情况。

```
1.     NODE *value_node(AttrType type, void *value) {
2.      NODE *n = newnode(N_VALUE);
3.
4.      n->u.VALUE.type = type;
5.      switch (type) {
6.      case INT:
7.          n->u.VALUE.ival = *(int *)value;
8.          break;
9.      case FLOAT:
10.         n->u.VALUE.rval = *(float *)value;
11.         break;
12.     case DECIMAL:
13.         n->u.VALUE.rval = *(float *)value;
```

```
14.         break;
15.     case STRING:
16.         n->u.VALUE.sval = (char *)value;
17.         break;
18.     case VARCHAR:
19.         n->u.VALUE.sval = (char *)value;
20.         break;
21.     case DATE:
22.         n->u.VALUE.sval = (char *)value;
23.         break;
24.     case DATETIME:
25.         n->u.VALUE.sval = (char *)value;
26.         break;
27.     }
28.     return n;
29. }
```

- **最后修改 interp.cc**，完善四种类型数据在底层功能函数中的代码，这些代码在建表和插入时有调用。

```
1.     static int mk_attr_infos(NODE *list, int max, AttrInfo attrInfos[]) {
2.   ...
3.         if (!strcmp(type_str, "int")) {
4.             type = INT;
5.         } else if (!strcmp(type_str, "char")) {
6.             type = STRING;
7.         } else if (!strcmp(type_str, "varchar")) {
8.             type = VARCHAR;
9.         } else if (!strcmp(type_str, "float")) {
10.            type = FLOAT;
11.        } else if (!strcmp(type_str, "decimal")) {
12.            type = DECIMAL;
13.        } else if (!strcmp(type_str, "datetime")) {
14.            type = DATETIME;
15.        } else if (!strcmp(type_str, "date")) {
16.            type = DATE;
17. //           printf("date yes\n");
18.        } else {
19.            return E_INVFORMATSTRING;
20.        }
```

```
1.     static void mk_value(NODE *node, Value &value) {
2.   ...
3.         case INT:
4.             value.type = VT_INT;
```

```
5.          value.data = (void *)&node->u.VALUE.ival;
6.          break;
7.      case FLOAT:
8.          value.type = VT_FLOAT;
9.          value.data = (void *)&node->u.VALUE.rval;
10.         break;
11.     case DECIMAL:
12.         value.type = VT_DECIMAL;
13.         value.data = (void *)&node->u.VALUE.rval;
14.         break;
15.     case STRING:
16.         value.type = VT_STRING;
17.         value.data = (void *)node->u.VALUE.sval;
18.         break;
19.     case VARCHAR:
20.         value.type = VT_VARCHAR;
21.         value.data = (void *)node->u.VALUE.sval;
22.         break;
23.     case DATETIME:
24.         value.type = VT_DATETIME;
25.         value.data = (void *)node->u.VALUE.sval;
26.         break;
27.     case DATE:
28.         value.type = VT_DATE;
29.         value.data = (void *)node->u.VALUE.sval;
30.         break; } } }
```

C. 修改系统管理模块（SM）的代码

- **首先修改 SM_Manager.cc，**希望在建表时除 int 外其他类型都采用 attrLength+1 的大小。

```
1.  RC SM_Manager::CreateTable(const char *relName, int attrCount, AttrInfo *attributes) {
2.  ...
3.  attrEntry.attrSize = attributes[i].attrType == INT ? 4 : attributes[i].attrLength + 1;
```

之后将内容从指定文件加载到指定关系中

```
1.  RC SM_Manager::Load(const char *relName, const char *fileName) {
2.  ...
3.              case FLOAT: {
4.                  char *end = NULL;
5.                  *(float *)(data + attributes[i].offset) = strtof(buffer + p, &end);
6.                  if (end == data + attributes[i].offset) {
```

```
                    std::cerr << cnt + 1 << ":" << q << " " << "incorrect float" << std::endl;
                    return SM_FILE_FORMAT_INCORRECT;  }
                break;    }
            case DECIMAL: {
                char *end = NULL;
                *(float *)(data + attributes[i].offset) = strtof(buffer + p, &end);
                if (end == data + attributes[i].offset) {
                    std::cerr << cnt + 1 << ":" << q << " " << "incorrect float" << std::endl;
                    return SM_FILE_FORMAT_INCORRECT;   }
                break;   }
            //string
            case STRING: {
                if (q - p > attributes[i].attrDisplayLength) {
                    std::cerr << cnt + 1 << ":" << q << " " << "string too long" << std::endl;
                    return SM_FILE_FORMAT_INCORRECT;   }
                strcpy(data + attributes[i].offset, buffer + p);
                break;   }
            case VARCHAR: {
                if (q - p > attributes[i].attrDisplayLength) {
                    std::cerr << cnt + 1 << ":" << q << " " << "string too long" << std::endl;
                    return SM_FILE_FORMAT_INCORRECT;  }
                strcpy(data + attributes[i].offset, buffer + p);
                break;  }
            //date
            case DATE: {
                if (q - p > attributes[i].attrDisplayLength) {
                    std::cerr << cnt + 1 << ":" << q << " " << "date too long" << std::endl;
                    return SM_FILE_FORMAT_INCORRECT;  }
                strcpy(data + attributes[i].offset, buffer + p);
                break;  }
            case DATETIME: {
                if (q - p > attributes[i].attrDisplayLength) {
                    std::cerr << cnt + 1 << ":" << q << " " << "dateTIME too long" << std::endl;
                    return SM_FILE_FORMAT_INCORRECT;  }
                strcpy(data + attributes[i].offset, buffer + p);
                break;  }
```

- **然后修改 printer.cc 文件，**主要是增加四种数据类型的输出

```
Printer::Printer(const std::vector<DataAttrInfo> &attributes_) {
```

```
...
 if (attributes[i].attrType == STRING)
        spaces[i] = min(attributes[i].attrDisplayLength, MAXPRINTSTRING);
    else if (attributes[i].attrType == VARCHAR)
        spaces[i] = min(attributes[i].attrDisplayLength, MAXPRINTSTRING);
    else if (attributes[i].attrType == DATE)
        spaces[i] = min(attributes[i].attrDisplayLength, MAXPRINTSTRING);
    else if (attributes[i].attrType == DATETIME)
        spaces[i] = min(attributes[i].attrDisplayLength, MAXPRINTSTRING);
```

```
 //打印函数
void Printer::Print(ostream &c, const char * const data, bool isnull[]) {
...
    else if (attributes[i].attrType == VARCHAR || this_isnull) {
        //打印出前 MAXNAME + 10 个字符
        memset(str, 0, MAXPRINTSTRING);

        const char* str_to_print = this_isnull ? "NULL" : data + attributes[i].offset;

        if (attributes[i].attrDisplayLength > MAXPRINTSTRING) {
            strncpy(str, str_to_print, MAXPRINTSTRING - 1);
            str[MAXPRINTSTRING - 3] = '.';
            str[MAXPRINTSTRING - 2] = '.';
            c << str;
            Spaces(MAXPRINTSTRING, strlen(str));
        } else {
            strncpy(str, str_to_print, (size_t)(this_isnull ? 4 : attributes[i].attrDisplayLength));
            c << str;
            if (attributes[i].attrDisplayLength < (int) strlen(psHeader[i]))
                Spaces((int)strlen(psHeader[i]), strlen(str));
            else
                Spaces(attributes[i].attrDisplayLength, strlen(str));
        }
    }
    else if (attributes[i].attrType == DATE || this_isnull) {
        //打印出前 MAXNAME + 10 个字符
        memset(str, 0, MAXPRINTSTRING);

        const char* str_to_print = this_isnull ? "NULL" : data + attributes[i].offset;

        if (attributes[i].attrDisplayLength > MAXPRINTSTRING) {
```

```
32.                 strncpy(str, str_to_print, MAXPRINTSTRING - 1);
33.                 str[MAXPRINTSTRING - 3] = '.';
34.                 str[MAXPRINTSTRING - 2] = '.';
35.                 c << str;
36.                 Spaces(MAXPRINTSTRING, strlen(str));
37.             } else {
38.                 strncpy(str, str_to_print, (size_t)(this_isnull ? 4 : attributes[i].attrDisplayLength));
39.                 c << str;
40.                 if (attributes[i].attrDisplayLength < (int) strlen(psHeader[i]))
41.                     Spaces((int)strlen(psHeader[i]), strlen(str));
42.                 else
43.                     Spaces(attributes[i].attrDisplayLength, strlen(str));
44.             }
45.         } else if (attributes[i].attrType == DATETIME || this_isnull) {
46.             //打印出前 MAXNAME + 10 个字符
47.             memset(str, 0, MAXPRINTSTRING);
48.
49.             const char* str_to_print = this_isnull ? "NULL" : data + attributes[i].offset;
50.
51.             if (attributes[i].attrDisplayLength > MAXPRINTSTRING) {
52.                 strncpy(str, str_to_print, MAXPRINTSTRING - 1);
53.                 str[MAXPRINTSTRING - 3] = '.';
54.                 str[MAXPRINTSTRING - 2] = '.';
55.                 c << str;
56.                 Spaces(MAXPRINTSTRING, strlen(str));
57.             } else {
58.                 strncpy(str, str_to_print, (size_t)(this_isnull ? 4 : attributes[i].attrDisplayLength));
59.                 c << str;
60.                 if (attributes[i].attrDisplayLength < (int) strlen(psHeader[i]))
61.                     Spaces((int)strlen(psHeader[i]), strlen(str));
62.                 else
63.                     Spaces(attributes[i].attrDisplayLength, strlen(str));  }   }
64.         }else if (attributes[i].attrType == DECIMAL) {
65.             memcpy (&b, (data + attributes[i].offset), sizeof(float));
66.             //sprintf(strSpace, "%f", b);
67.             snprintf(strSpace, attributes[i].attrSize, "%f", b);
68.             c << strSpace;
69.             if (strlen(psHeader[i]) < 12)
70.                 Spaces(12, strlen(strSpace));
71.             else
72.                 Spaces((int)strlen(psHeader[i]), strlen(strSpace));
```

D. 修改查询解析模块（QL）

- 首先修改 **ql.manager.cc,**

```
/**
 * 判断左属性与右常量的数据类型是否匹配
 */
static bool can_assign_to(AttrType rt, ValueType vt, bool nullable) {
    //if(vt == VT_DATE) printf("vt == VT_DATE\n");
    //if(rt == DATE) printf("rt == DATE\n");
    //LOG(WARNING) << "attrtype = " << vt;
    return (vt == VT_NULL && nullable) ||
           (vt == VT_INT && rt == INT) ||
           (vt == VT_INT && rt == FLOAT) ||
           (vt == VT_FLOAT && rt == FLOAT) ||
           (vt == VT_DECIMAL && rt == DECIMAL) ||
           (vt == VT_FLOAT && rt == DECIMAL) ||
           (vt == VT_STRING && rt == STRING) ||
           (vt == VT_VARCHAR && rt == VARCHAR) ||
           (vt == VT_STRING && rt == VARCHAR) ||
           (vt == VT_DATE && rt == DATE) ||
           (vt == VT_STRING && rt == DATE) ||
           (vt == VT_STRING && rt == DATETIME) ||
           (vt == VT_DATETIME && rt == DATETIME);  }
```

```
std::ostream &operator <<(std::ostream &os, const QL_Condition &condition) {

...
        if (condition.bRhsIsAttr) {
            os << condition.rhsAttr.relName << "." << condition.rhsAttr.attrName;

        } else {
            switch (condition.rhsValue.type) {
                case VT_INT:
                    os << *(int *)condition.rhsValue.data;
                    break;
                case VT_FLOAT:
                    os << *(float *)condition.rhsValue.data;
                    break;
                case VT_DECIMAL:
                    os << *(float *)condition.rhsValue.data;
                    break;
                case VT_STRING:
```

```
                os << (char *)condition.rhsValue.data;
                break;
            case VT_VARCHAR:
                os << (char *)condition.rhsValue.data;
                break;
            case VT_DATE:
                os << (char *)condition.rhsValue.data;
                break;
            case VT_DATETIME:
                os << (char *)condition.rhsValue.data;
                break;
...
```

```
RC QL_Manager::Insert(const char *relName, int nValues, const Value *values) {
...
            case DECIMAL: {
                *(float *)dest = *(float *)value;
                break;
            }
            case STRING: {
                char *src = (char *)value;
                // 字符串长度超过限制
                if (strlen(src) > attr.attrDisplayLength) return QL_STRING_VAL_TOO_LONG;
                strcpy(dest, src);
                break;
            }
            case VARCHAR: {
                char *src = (char *)value;
                // 字符串长度超过限制
                printf("INSERT-VARCHAR = %s\n",src);
                if (strlen(src) > attr.attrDisplayLength) return QL_STRING_VAL_TOO_LONG;
                strcpy(dest, src);
                break;
            }
            case DATE: {
                char *src = (char *)value;
                printf("INSERT-DATE = %s\n",src);
                // 字符串长度超过限制
                if (strlen(src) > attr.attrDisplayLength) return QL_STRING_VAL_TOO_LONG;
                strcpy(dest, src);
                break;
```

```
29.             }
30.             case DATETIME: {
31.                 char *src = (char *)value;
32.                 printf("INSERT-DATETIME = %s\n",src);
33. //                  // 字符串长度超过限制
34.                 if (strlen(src) > attr.attrDisplayLength) return QL_STRING_VAL_TOO
    _LONG;
35.                 strcpy(dest, src);
36.                     break;
```

这里传入的 values 对象，记录的要插入的数据，this_values 则指向了当前要插入的某一行，某一条数据的指针。遍历每个属性，根据 attrType，强制转换 value 指针的类型，因为 value 定义的时候是 void*，为了对所有数据类型都通用。判断一下是否超过限制，没有的话拷贝到 dest。而这里的 dest 是目标要存的地址，赋值相当于在 Data 加偏移的地方改变值。

```
1.  bool checkSatisfy(char *lhsData, bool lhsIsnull, char *rhsData, bool r
    hsIsnull, const QL_Condition &condition) {
2.  ...
3.      case DECIMAL: {
4.          float lhs = *(float *)lhsData;
5.          float rhs = *(float *)rhsData;
6.          switch (condition.op) {
7.              case EQ_OP:
8.                  return lhs == rhs;
9.              case NE_OP:
10.                 return lhs != rhs;
11.             case LT_OP:
12.                 return lhs < rhs;
13.             case GT_OP:
14.                 return lhs > rhs;
15.             case LE_OP:
16.                 return lhs <= rhs;
17.             case GE_OP:
18.                 return lhs >= rhs;
19.             default:
20.                 CHECK(false);
21.         }
22.     }
23. ...
```

这里仅展示部分代码。

2.2 支持更多的属性域约束

在这一模块我们实现了唯一性约束（UNIQUE）和 CHECK 约束，其中我们的 CHECK 约束仅仅约束 INT 类型的数据，CHECK(a) 表示输入的数字必须小于 a 才能正常插入，修改的核心代码如下：

A. 修改 redbase.h 的文件，主要是增加这两种约束

```
1.  enum AttrSpec {
2.    ATTR_SPEC_NONE = 0x0,
3.    ATTR_SPEC_NOTNULL = 0x1,
4.    ATTR_SPEC_PRIMARYKEY = 0x2,
5.    ATTR_SPEC_UNIQUE = 0x4,
6.    ATTR_SPEC_CHECK = 0x8,
```

B. 修改 PARSER 模块文件

- **首先修改 parse.y 文件，**

```
1.  %token
2.  ...
3.  RW_UNIQUE
4.  RW_CHECK
```

执行建表时会用到这里的 attrType 文法，增加 RW_UNIQUE 和 RW_CHECK 结尾的文法。

```
1.    attrtype
2.  ...
3.    | T_STRING T_STRING '(' T_INT ')' RW_UNIQUE
4.    {
5.      $$ = attrtype_node($1, $2, $4, ATTR_SPEC_UNIQUE, NULL);
6.    }
7.    | T_STRING T_STRING '(' T_INT ')' RW_CHECK '(' T_INT ')'
8.    {
9.      $$ = attrtype_node($1, $2, $8, ATTR_SPEC_CHECK, NULL);
10.   }
```

- **然后修改 scanhelp.h 文件**

```
1.      static int get_id(char *s) {
2.  ...
3.    if (!strcmp(string, "unique"))
4.       return yylval.ival = RW_UNIQUE;
5.    if (!strcmp(string, "check"))
6.       return yylval.ival = RW_CHECK;
```

C. 修改系统管理模块 SM 的代码

- **修改 SM_Manager.cc 文件**

```
1.    RC SM_Manager::CreateTable(const char *relName, int attrCount, AttrInfo *attributes) {
2.  ...
3.        if (attrEntry.attrSpecs & ATTR_SPEC_PRIMARYKEY) {
4.            attrEntry.indexNo = indexNo++;
5.        } else if(attrEntry.attrSpecs & ATTR_SPEC_UNIQUE){
6.            attrEntry.indexNo = indexNo++;
7.        } else {
8.            attrEntry.indexNo = -1;
9.        }
10. ...
11.       if (attributes[i].attrSpecs == ATTR_SPEC_UNIQUE)
12.           TRY(ixm->CreateIndex(relName, relEntry.indexCount++, attributes[i].attrType, attribu
    tes[i].attrLength));  }
```

D. 修改查询解析模块 QL

- **修改 ql.manager.cc 文件**

```
1.
    RC QL_Manager::Insert(const char *relName, int nValues, const Value *values
    ) {
2.  ...
3.        bool duplicate = false;
4.        bool duplicate2 = false;
5.        bool duplicate3 = false;
6.        for (int i = 0; i < attrCount; ++i){
7.           // 遍历该记录中每个属性对应的值 对主键属性判断有无重复值
8.           //int gg = attributes[i].attrSpecs;
9.           // LOG(WARNING) << "attrspecs = " << gg;
10.          if (attributes[i].attrSpecs == ATTR_SPEC_PRIMARYKEY ) {
11.             // 该属性是主键
12.             TRY(pIxm->OpenIndex(relName, attributes[i].indexNo, indexHandle))
    ;
13.             RID rid;
14.             TRY(scan.OpenScan(indexHandle, EQ_OP, this_values[i].data));
15.             int retcode = scan.GetNextEntry(rid);
16.             TRY(scan.CloseScan());
17.             TRY(pIxm->CloseIndex(indexHandle));
18.             if (retcode != IX_EOF) {
19.                if (retcode != 0) return retcode;
20.                duplicate = true;
21.             }
22.          }else if (attributes[i].attrSpecs == ATTR_SPEC_UNIQUE) {
```

```
            // 该属性是 UNIQUE
            TRY(pIxm->OpenIndex(relName, attributes[i].indexNo, indexHandle));
            RID rid;
            TRY(scan.OpenScan(indexHandle, EQ_OP, this_values[i].data));
            int retcode = scan.GetNextEntry(rid);
            TRY(scan.CloseScan());
            TRY(pIxm->CloseIndex(indexHandle));
            if (retcode != IX_EOF) {
                if (retcode != 0) return retcode;
                duplicate2 = true;
            }
        }else if (attributes[i].attrSpecs == ATTR_SPEC_CHECK){
            if(*reinterpret_cast<int*>(this_values[i].data) > attributes[i].attrSize){
                duplicate3 = true;
                break;}
        }
    }
    // 主键值与已有值重复
    if (duplicate) { printf("Warning*****1 primary key error\n");return QL_DUPLICATE_PRIMARY_KEY;}
    if (duplicate2) { printf("Warning*****2 unique error\n");return QL_DUPLICATE_DISTINCT;}
    if (duplicate3){ printf("Warning*****3 CHECK error\n");return QL_INT_VAL_NOT_CHECK;}
```

- 然后修改 **ql.h** 文件

```
#define QL_DUPLICATE_DISTINCT       (START_QL_WARN + 9)
#define QL_INT_VAL_NOT_CHECK        (START_QL_WARN + 10)
```

- 最后修改 **ql.error.cc** 文件

```
//
// 错误表
//
const char *QL_WarnMsg[] = {
…
    "The attribute to insert the entry is unique, but the value already exists",
    "insert data does not meet the check condition",
};
```

这一部分主要是增加 UNIQUE 和 CHECK 约束不满足时的输出错误提示。

2.3 支持外键约束

在这一模块我们实现了形式上的外键约束 FOREIGNKEY，构建时的应当采用如下输入形式：’create table table2(a varchar(10) foreignkey table1)’，表示 a 列输入时依赖 table1。修改的核心代码如下：

A. 修改 redbase.h 文件，增加外键约束的类型

```
enum AttrSpec {
    ATTR_SPEC_NONE = 0x0,
    ATTR_SPEC_NOTNULL = 0x1,
    ATTR_SPEC_PRIMARYKEY = 0x2,
    ATTR_SPEC_UNIQUE = 0x4,
    ATTR_SPEC_CHECK = 0x8,
    ATTR_SPEC_FOREIGNKEY = 0x10
};
```

B. 修改 PARSER

- **首先修改 parser_internal.h 文件，**主要为属性结构体增加一个依赖表的变量，并且修改相应的函数

```
    struct {
     char *attrname;
     char *type;
     int size;
     enum AttrSpec spec;
     char *relname;
} ATTRTYPE;
NODE *attrtype_node(char *attrname, char *type, int size, enum AttrSpec spec,char *relname);
```

- **然后修改 node.cc 中的节点分配函数**

```
  //attrtype_node：分配，初始化并返回指向新指针的指针
NODE *attrtype_node(char *attrname, char *type, int size, enum AttrSpec spec,char *relname) {
    NODE *n = newnode(N_ATTRTYPE);

    n -> u.ATTRTYPE.attrname = attrname;
    n -> u.ATTRTYPE.type = type;
    n -> u.ATTRTYPE.size = size;
    n -> u.ATTRTYPE.spec = spec;
    n->u.ATTRTYPE.relname = relname;
    return n;
}
```

- **最后修改 parse.y 文件，增加外键约束类型的相关代码**

```
    %token
...
```

```
3.  RW_FOREIGNKEY
4.  ...
5.  | T_STRING T_STRING '(' T_INT ')' RW_FOREIGNKEY T_STRING

6.      {
7.          $$ = attrtype_node($1, $2, T_INT, ATTR_SPEC_FOREIGNKEY, $7);
8.      }
```

上面这一部分就能完成外键约束形式上的建立，具体功能需要在 ql.manager.cc 文件中增加相应的判断和和具体操作才能完成插入或者更新时的相应功能，但由于代码能力欠缺，该部分一直出错没有解决，因此仅完成外键约束形式上的功能。

三、功能测试

3.1 实现更多的的数据类型

- **建表：CREATE TABLE test (a int(10) NOT NULL, b varchar(1024), c decimal(5), d date，e datetime, PRIMARY KEY(a),)；**并用 **desc test** 查看建立的表：

```
MICROBASE >> create table test (a int(10) not null, b varchar(1024), c decimal(5), d date(10), e datetime(20), primary key(a));
create table test (a int(10) not null, b varchar(1024), c decimal(5), d date(10), e datetime(20), primary key(a));
## int int ## 0 10 1
## varchar varchar ## 1 1024 0
## decimal decimal ## 2 5 0
## date date ## 3 10 0
## datetime datetime ## 4 20 0
num is 5
INT
VARCHAR
DECIMAL
DATE
DATETIME

MICROBASE >> desc test;
help test;
relName                 attrName                offset      attrType    attrSize    attrDisplayLength attrSpecs   indexNo
-------------------------------------------------------------------------------------------------------------------------
test                    a                       0           0           4           10                3           0
test                    b                       4           5           1025        1024              0           -1
test                    c                       1032        6           6           5                 0           -1
test                    d                       1040        3           11          10                0           -1
test                    e                       1052        4           21          20                0           -1

5 tuple(s).
```

- **插入两条数据：insert into test values(1,'test1',22.36,'2022-6-13','2022-6-13 19:00:00'); 和 insert into test values(2,'test2',22.36,'2022-6-13','2022-6-13 19:00:00'); 并用 select * from test 查看是否插入成功。**

```
MICROBASE >>  insert into test values(1,'test1',22.36,'2022-6-13','2022-6-13 19:00:00');
T_STRING: test1
T_DATE: 2022-6-13
T_DATETIME: 2022-6-13 19:00:00
insert into test values (  1, 'test1', 22.360001,,);

MICROBASE >> insert into test values(2,'test2',50.36,'2022-6-13','2022-6-13 19:00:00');
T_STRING: test2
T_DATE: 2022-6-13
T_DATETIME: 2022-6-13 19:00:00
insert into test values (  2, 'test2', 50.360001,,);

MICROBASE >> select * from test;
select  *
 from  test
;
a           b                                                  c           d         e
-------------------------------------------------------------------------------------------------------------
1           test1                                              22.36       2022-6-13 2022-6-13 19:00:00
2           test2                                              50.36       2022-6-13 2022-6-13 19:00:00

2 tuple(s)
```

3.2 更多的属性域约束

- 创建表：**create table test2(a int(10) not null, b char(20) unique, c int(10) check(15));** 注意这里的 check（15）表示 c 的取值不能大于 15；然后用 desc test2 查看表属性；

```
MICROBASE >> create table test2(a int(10) not null, b char(20) unique, c int(10) check(15));
create table test2 (a int(10) not null, b char(20) unique, c int(15));
## int int ## 0 10 1
## char char ## 1 20 4
## int int ## 2 15 8
num is 3
INT
STRING
INT

MICROBASE >> desk test2;
syntax error

MICROBASE >> desc test2;
help test2;
relName                 attrName                offset      attrType    attrSize    attrDisplayLength attrSpecs   indexNo
-------------------------------------------------------------------------------------------------------------------------
test2                   a                       0           0           4           10                1           -1
test2                   b                       4           2           21          20                4           0
test2                   c                       28          0           4           15                8           -1

3 tuple(s)
```

- 插入两条数据：**insert into test2 values(1,'test1',5);**

  insert into test2 values(2,'test2',10);

  然后用 **select * from test2;** 查看插入情况；

```
MICROBASE >> insert into test2 values(1,'test1',5);
T_STRING: test1
insert into test2 values (  1, 'test1', 5);

MICROBASE >> insert into test2 values(2,'test2',10);
T_STRING: test2
insert into test2 values (  2, 'test2', 10);

MICROBASE >> select * from test2;
select  *
 from  test2
;
a           b                   c
-------------------------------------------
1           test1               5
2           test2               10

2 tuple(s).
```

- 验证 UNIQUE 约束，插入数据：**insert into test2 values(3,'test2',10);** 由于 ’test2’ 重复，会产生 UNIQUE 不通过错误

```
MICROBASE >> insert into test2 values(3,'test2',10);
T_STRING: test2
insert into test2 values (  3, 'test2', 10);
Warning*****2 unique error

MICROBASE >> QL warning: The attribute to insert the entry is unique, but the value already exists
```

- 验证 CHECK 约束， 插入数据：**insert into test2 values(3,'test3',20);** 由于 20 大于 15，会产生 check 不通过错误：

```
insert into test2 values(3,'test3',20);
T_STRING: test3
insert into test2 values (  3, 'test3', 20);
Warning*****3 CHECK error

MICROBASE >> QL warning: insert data does not meet the check condition
```

3.3 创建外键约束

- 创表：**create table test3(a int(10) not null, b int(10) foreignkey test1, primary**

key(a));这里 b 依赖于 test1，然后用 select * from test3 查看建立的表；

```
create table test3(a int(10) not null, b int(10) foreignkey test1, primary key(a));
create table test3 (a int(10) not null, b int(307) check, primary key(a));
## int int ## 0 10 1
## int int ## 1 307 16
num is 2
INT
INT

MICROBASE >> select * from test3;
select  *
 from  test3
;
a           b
------------------------

0 tuple(s).
```

四、项目心得

通过这次学习，我们对整个 DBMS 系统的运行流程有了更深入的理解。以前的实验课中，我们主要关注于具体模块的学习，了解各个类和函数的功能，更多地关注 DBMS 底层的运行逻辑和结构。但是我们没有从用户的角度来看待整个 DBMS 系统，即当用户输入一个命令时，DBMS 系统会进行哪些整体工作流程：从命令的解析，到根据不同的命令调用不同的模块来处理记录、索引等，最终输出结果。通过本次学习，我们对于如何解析输入的命令（特别是查询命令），在各个模块的各个类之间如何传递解析得到的属性、关系等信息，以及如何传递查询得到的相关元组信息等方面有了更充分的了解。我们现在能够更好地理解整个 DBMS 系统的工作流程，能够更好地处理用户输入的命令，并将其转化为对应的操作和结果输出。这样的学习让我们能够更全面地了解和应用 DBMS 系统，从用户的角度思考问题，并更好地理解和操作底层的各个模块，使我们能够更有效地开发和管理数据库系统。

最后是结课作业中，因为对 C++语言的不熟悉，在进行外键约束的实现上出现了很多 bug，因此我们仅仅展示了形式上的外键约束实现，更为具体的操作还需要进一步的学习和调整。