编译原理实验 Project-SQL 查询

21307174 刘俊杰

May 2024

1 题目描述

建立一个简单的教务管理数据库，用于存储学生的考试成绩并支持一些基本的查询操作。

1.1 数据格式

该数据库包含以下五张数据表：

Student(sid, dept, age)

学生信息表：sid 为主键，表示学生 ID；dept 表示学生所在院系名称；age 表示学生的年龄。

Course(cid, name)

课程信息表：cid 为主键，表示课程 ID；name 表示课程名称。

Teacher(tid, dept, age)

教师信息表：tid 为主键，表示教师 ID；dept 表示教师所在院系名称；age 表示教师的年龄。

Grade(sid, cid, score)

成绩信息表：sid 和 cid 分别米自 Student 表和 Course 表的主键（即每条记录中的 sid 和 cid 一定存在于相应的表中，下同），二者一起作为该表的主键；score 表示该学生这门课程的成绩。

Teach(cid, tid) 授课信息表:cid 和 tid 分别来自 Course 表和 Teacher 表的主键，二者一起作为为该表的主键；需要注意的是，一门课程可以有多个老师授课。

注：主键（Primary Key）可以唯一标识表中的每条记录，换言之在一张表中所有记录的主键互不相同。

在上述表中，age 和 score 是 [0,100] 的整数，其余都是长度小于等于 10 的非空字符串。其中三个主键 sid _cid_ 和 tid 由数字 0...9 组成，而 name 和 dept 则由大小写字母组成。

1.2 查询语句

（具体查询语句说明见 SQL 查询作业 PDF）

1.3 结果输出

对每条查询输出一张表，列由 COLUMNS 指定，每行为一条满足筛选条件 (EXPR) 的记录（不同的列间用一个空格分隔)。这里只需要输出所有满足条件的记录，不需要打印表头；查询结果为空则不输出；字符串类型无需输出双引号。若想输出所有的列，可以用 * 代替 COLUMNS；若 TABLES 中只有一张表，此时应输出该表的全部列；如果有第二张表，应输出第一张表的全部列与二张表的全部列的笛卡尔积：表内部的列按照上文定义时给出的顺序排序。

如果 TABLES 仅含一张表，则按该表的顺序依次输出符合条件的记录。若 TABLES 包含两张表（形如 A,B ），则先考虑 A 的顺序，再考虑 B 的顺序 (参考前文中两张表做笛卡尔积的例子)。

1.4 【输入格式】

从标准输入读入数据。

首先依次输入五张表 Student、Course、Teacher、Grade 和 Teach 的数据。

对于第 i 张表，第一行输入一个正整数 N_i，表示该表中记录的个数（亦即行数)。接下来 N_i 行，每行输入一条记录，其中不同列之间用一个空格分隔且字符出字段不会用双引号括起。

然后输入一个正整数 M，表示需要处理的查询语句个数。最后 M 行每行输入一条查询语句，保证每条查询语句均符合上述所有要求。

1.5 【输出格式】

输出到标准输出。

按要求输出每条查询语句的结果。

1.6 【样例 1 输入】

```
3
2019001 law 19
2019002 info 20
2019003 info 19
2
20190001 math
20190002 English
2
2019101 info 49
2019102 info 38
4
SELECT FROM Student
SELECT Student . dept FROM Student
SELECT Student.sid,Grade.sid,dept, score FROM Student,Grade
SELECT FROM Student, Grade WHERE Grade .sid=Student. sid
```

1.7 【样例 1 输出】

```
2019001 law 19
2019002 info 20
2019003 info 19
law
info
info
2019001 2019002 law 100
2019001 2019003 law 0
2019002 2019002 info 100
2019002 2019003 info 0
2019003 2019002 info 100
2019003 2019003 info 0
2019002 info 20 2019002 20190001 100
2019003 info 19 2019003 20190002 0
```

1.8 【样例 1 解释】

第 1-3 行、4-6 行、7-12 行和 13-14 行依次对应四条查询的F果。

每条查询最多涉及两张表，每个 expr 中最多一个 COLUMN = COLUMN 类型的 COND，且 $Ni \leqslant 10000$、$M \leqslant 10$ 。此外，所有测试数据保证每条查询的结果均不超过 10000 行。

2 实验思路与核心代码

2.1 对表中记录的存储

```
1 // TABLE根据表名存储不同表的记录
2 unordered_map<string, vector<vector<string>>> TABLE;
3
4 // 列与记录索引的映射
5 unordered_map<string, int> col_to_index = {
6     {"sid", 0}, {"cid", 1}, {"tid", 2}, {"score", 3},
7     {"name", 4}, {"dept", 5}, {"age", 6}
8 };
```

首先为了实现对输入的记录数据的存储，使用 unordered_map 哈希表 TABLE 来存储，以表名作为键、插入的记录为值，插入的记录用 vector<vector<string 存储。

注意的是这里为了实现方便,每个表的记录都有 6 列,即 sid，cid,tid,score,name,dept,age,，如果该表中某列不存在则用”” 替代，后续如果优化可以对这部分进行优化，减少空间开销

同时用另外一个哈希表 col_to_index 记录该列名与记录索引（第几列）的映射。

2.2 对输入记录的处理

针对不同表的输入记录，将记录插入对应的表中，实现代码如下：

```
1 针对不同
2 // 处理输入记录，存储到TABLE对应的表中
3 void process_input() {
4     for (int i = 0; i < 5; i++) {
```

```
5           int n;
6           cin >> n;
7           while (n > 0) {
8               n--;
9               switch (i) {
10                  case 0: { // Student(sid,dept,age)
11                      long long sid, age;
12                      string dept;
13                      cin >> sid >> dept >> age;
14                      vector<string> temp = {to_string(sid),
15                      "", "", "", "", dept, to_string(age)};
16                      TABLE["Student"].push_back(temp);
17                      break;
18                  }
19                  case 1: { // Course(cid,name)
20                      long long cid;
21                      string name;
22                      cin >> cid >> name;
23                      vector<string> temp = {"", to_string(cid),
24                      "", "", name, "", ""};
25                      TABLE["Course"].push_back(temp);
26                      break;
27                  }
28                  case 2: { // Teacher(tid,dept,age)
29                      long long tid, age;
30                      string dept;
31                      cin >> tid >> dept >> age;
32                      vector<string> temp = {to_string(tid), "",
33                      "", "", "", dept, to_string(age)};
34                      TABLE["Teacher"].push_back(temp);
35                      break;
36                  }
37                  case 3: { // Grade(sid,cid,score)
38                      long long sid, cid, score;
39                      cin >> sid >> cid >> score;
```

```
40                      vector<string> temp = {to_string(sid),
41                      to_string(cid), "", to_string(score), "", "", ""};
42                      TABLE["Grade"].push_back(temp);
43                      break;
44                  }
45                  default: { // Teach(cid,tid)
46                      long long cid, tid;
47                      cin >> cid >> tid;
48                      vector<string> temp = {"", to_string(cid),
49                      "", "", "", "", to_string(tid)};
50                      TABLE["Teach"].push_back(temp);
51                      break;
52                  }
53              }
54          }
55      }
56  }
```

2.3 对查询语句的预处理

2.3.1 对字符串的预处理函数

以下是三种不同功能的字符串处理函数，方便后续在对查询语句的与处理中用到：

```
1  // 移除字符串中的所有空格
2  string removeSpaces(const string& str) {
3      string result;
4      for (char ch : str) {
5          if (!isspace(static_cast<unsigned char>(ch))) { // 使用isspace检查是否为空格
6              result += ch;
7          }
8      }
9      return result;
10 }
11
```

```
12 // 分割字符串的函数，基于提供的分隔符(char)，同时去除无用的空格
13 vector<string> splitString (const string& str, char delimiter ) {
14     vector<string> tokens;
15     string token;
16     istringstream tokenStream(str);
17 
18     while ( getline (tokenStream, token, delimiter )) {
19         tokens.push_back(removeSpaces(token));
20     }
21 
22     return tokens;
23 }
24 
25 // 分割字符串的函数，使用整个字符串作为分隔符(string)，同时去除无用的空格
26 vector<string> splitString_str (const string& str, const string& delimiters ) {
27     vector<string> result ;
28     size_t pos = 0;
29     size_t delimiterPos ;
30     // 只要还有分隔符，就继续分割
31     while (( delimiterPos = str. find ( delimiters , pos)) != string :: npos) {
32         // 如果分隔符前面有文本，添加到结果中
33         if ( delimiterPos != pos) {
34             result .push_back(removeSpaces(str.substr(pos, delimiterPos - pos)));
35         }
36         // 更新位置，跳过分隔符
37         pos = delimiterPos + delimiters . length ();
38     }
39     // 添加剩余的文本
40     if (pos != str . length ()) {
41         result .push_back(removeSpaces(str.substr(pos )));
42     }
43     return result ;
44 }
```

2.3.2 对查询语句的预处理

实现代码如下：

```
int n; // 查询语句的个数
    cin >> n;
    cin.ignore(); // 处理输入n后的换行符

    // 处理查询
    while (n > 0) {
        n--;
        string s;
        // 读取每个查询
        while (getline(cin, s)) {
            // 对查询语句的不同部分进行分割
            string attribute_str;
            string table_str;
            string where_str;
            int a = s.find("SELECT");
            int t = s.find("FROM");
            int w = s.find("WHERE") == -1 ? s.size() + 1 : s.find("WHERE");
            attribute_str = s.substr(a + 7, t - a - 8);
            table_str = s.substr(t + 5, w - t - 5);
            where_str = (w == s.size() + 1 ? "" : s.substr(w + 6));

            vector<string> table = splitString(table_str, ','); // 查询的表集合
            vector<string> attribute = splitString(attribute_str, ','); // 查询的列集合
            vector<string> where = splitString_str(where_str, "AND"); // 查询的条件集合

            // 处理查询并输出结果
            process_queries(table, attribute, where);
            break;
        }
    }
```

① 首先，对每一句查询语句针对不同的部分进行分割：

attribute_str 是 SELECT 到 FROM 之间的子字符串，即查询的列；

table_str 是 FROM 到 WHERE 之间的子字符串，即查询的表；

where_str 是 WHERE 之后的子字符串（若无 WHERE，则该 where_str 为""），即查询的条件。

② 使用前面的字符串处理函数，分别对步骤 ① 中的 attribute_str、table_str 和 where_str，用，来分割 attribute_str 和 table_str 不同列或表并去除空格，用 AND 来分割 where_str 来获得不同的条件并去除空格，得到：

attribute 是查询的列集合；

table 是查询的表集合；

where 是查询的条件集合。

③ 最后将查询语句的预处理结果传入到 process_queries 函数来处理查询并输出结果。

2.4 查询处理并输出结果

分针对 1 张表的查询和针对 2 张表的查询分别处理（后续如果有优化，可以优化该部分，使得可以对任意张表进行处理，不用分情况处理）：

```
1  // 处理查询信息
2  void process_queries(const vector<string>& table,
3  const vector<string>& attribute, const vector<string>& where) {
4      if (table.size() == 1) { // 查询一张表
5          process_queries_for_one_table(table, attribute ,where);
6      }
7      else { // 查询两张表
8          process_queries_for_two_table(table, attribute ,where);
9      }
10 }
```

2.4.1 对 1 张表的查询处理并输出结果

对一个条件的左右值简化：由于对查询一张表的条件的左右值进行处理，查询内容的列中可去除表名，即 TABLE.COLUMN 只用保留 COLUMN。同时为了处理与字符串的比较，如“1999”，要将其中的双引号去除。

实现代码如下：

```
// 对查询一张表的条件的左右值进行处理
// w : 条件字符串 str: 判别符
vector<string> process_condition_for_one_table(string w, string str) {
    vector<string> t;
    t = splitString_str(w, str);
    string x = t[0], y = t[1];
    // 针对一张表的查询 查询内容中可去除表名
    vector<string> xt = splitString(x, '.');
    vector<string> yt = splitString(y, '.');
    string xx = xt.size() == 1 ? xt[0] : xt[1];
    string yy = yt.size() == 1 ? yt[0] : yt[1];
    // 处理引号
    if (xx[0] == '\"' && xx.back() == '\"') {
        xx = xx.substr(1, xx.size() - 2);
    }
    if (yy[0] == '\"' && yy.back() == '\"') {
        yy = yy.substr(1, yy.size() - 2);
    }
    return {xx, yy};
}
```

处理对 1 张表的查询并输出查询结果：

① 首先遍历查询表中的每一个记录，用 flag 标志记录查询语句中的条件是否成立，成立 flag=1，否则 flag=0。

② 针对每一个记录判断是否满足查询的所有条件约束，先遍历判断所使用的判别符（如 =、!=、>= 等等），用判别符分割查询条件，得到左值和右值的字符串。

③ 调用 process_condition_for_one_table 对左右值进行简化。

④ 对简化后的结果根据判别符判断是否成立，不成立则修改 flag 并退出。

⑤ 若 flag=1 即所有条件都成立，则根据查询语句中的表和列，输出对应的查询结果；否则查询结果为空。

实现代码如下：

```
// 处理对1张表的查询
void process_queries_for_one_table(const vector<string>& table,
const vector<string>& attribute, const vector<string>& where) {
    for (auto& it : TABLE[table[0]]) {
            int flag = 1; // 条件是否成立标志

            // 判断where之后的条件是否成立
            for (auto& w : where) {
                // 针对不同判别符处理
                if (w.find("!=") != -1) { // !=
                    vector<string> t = process_condition_for_one_table(w, "!=");
                    string xx = t[0], yy = t[1];
                    if (xx == yy) {
                        flag = 0;
                        break;
                    }
                } else if (w.find(">=") != -1) { // >=
                    vector<string> t = process_condition_for_one_table(w, ">=");
                    string xx = t[0], yy = t[1];
                    if ( stoll (xx) < stoll (yy)) {
                        flag = 0;
                        break;
                    }
                } else if (w.find("<=") != -1) { // <=
                    vector<string> t = process_condition_for_one_table(w, "<=");
                    string xx = t[0], yy = t[1];
                    if ( stoll (xx) > stoll (yy)) {
                        flag = 0;
                        break;
                    }
                } else if (w.find("=") != -1) {// =
                    vector<string> t = process_condition_for_one_table(w, "=");
```

```
33                  string xx = t[0], yy = t[1];
34                  if (xx != yy) {
35                      flag = 0;
36                      break;
37                  }
38              } else if (w.find(">") != -1) {// >
39                  vector<string> t = process_condition_for_one_table(w, ">");
40                  string xx = t[0], yy = t[1];
41                  if ( stoll (xx) <= stoll(yy)) {
42                      flag = 0;
43                      break;
44                  }
45              } else if (w.find("<") != -1) { // <
46                  vector<string> t = process_condition_for_one_table(w, "<");
47                  string xx = t[0], yy = t[1];
48                  if ( stoll (xx) >= stoll(yy)) {
49                      flag = 0;
50                      break;
51                  }
52              }
53          }
54
55          // 如果条件成立,输出查询结果
56          if (flag) {
57              if ( attribute [0] == "*") { // 输出查询表的所有列
58                  for (int i = 0; i < 7; i++) {
59                      if (it[i] != "") {
60                          cout << it[i] << " ";
61                      }
62                  }
63                  cout << endl;
64              } else { // 输出查询的列
65                  for (int i = 0; i < attribute.size(); i++) {
66                      // 由于是对一张表的查询,可以只关注.之后的列名
67                      string a = attribute[i];
```

```
                    vector<string> t = splitString(a, '.');
                    string s;
                    if (t.size() == 1) s = t[0];
                    else s = t[1];
                    cout << it[col_to_index[s]];
                    if (i != attribute.size() - 1) cout << " ";
                }
                cout << endl;
            }
        }
    }
}
```

2.4.2 对 2 张表的查询处理并输出结果

对查询 2 张表的条件的左右值进行处理：首先将所有查询条件的左右值形如 TABLE.COLUMN，用 . 分割得到表名和列名，由此可以根据 table1 或 table2 的记录得到对应的值，再将存在的”” 双引号去除，最后返回左值和右值。

实现代码如下：

```
// 对查询2张表的条件的左右值进行处理
// t:左右值 table:表名集合 it1: table1的记录 it2: table2的记录
vector<string> process_condition_for_two_table(const vector<string>& t,
const vector<string>& table,const vector<string>& it1,const vector<string>& it2) {
    string x = t[0], y = t[1];
    string xx, yy;
    vector<string> xt = splitString(x, '.');
    vector<string> yt = splitString(y, '.');
    if (xt.size() == 1) {
        if (!col_to_index.count(xt[0])) {
            xx = xt[0];
            if (xx[0] == '\"' && xx.back() == '\"') xx = xx.substr(1, xx.size() - 2);
        } else xx = it1[col_to_index[xt[0]]] != "" ? it1[col_to_index[xt[0]]] : it2[col_to_index[xt[0]]];
    } else {
        if (xt[0] == table[0]) xx = it1[col_to_index[xt[1]]];
        else xx = it2[col_to_index[xt[1]]];
```

```
    }
    if (yt.size() == 1) {
        if (!col_to_index.count(yt[0])) {
            yy = yt[0];
            if (yy[0] == '\"' && yy.back() == '\"') yy = yy.substr(1, yy.size() - 2);
        } else yy = it1[col_to_index[yt[0]]] != "" ? it1[col_to_index[yt[0]]] : it2[col_to_index[yt[0]]];
    } else {
        if (yt[0] == table[0]) yy = it1[col_to_index[yt[1]]];
        else yy = it2[col_to_index[yt[1]]];
    }
    return {xx,yy};
}
```

处理对两张表的查询并输出查询结果：

① 首先两重循环遍历查询两张表中的每一个记录，用 flag 标志记录查询语句中的条件是否成立，成立 flag=1，否则 flag=0。

② 针对每一个记录判断是否满足查询的所有条件约束，先遍历判断所使用的判别符（如 =、!=、>= 等等），用判别符分割查询条件，得到左值和右值的字符串。

③ 调用 process_condition_for_two_table 对左右值进行简化，并分出从属哪一张表，进而获得最终的左值和右值。

④ 对简化后的结果根据判别符判断是否成立，不成立则修改 flag 并退出。

⑤ 若 flag=1 即所有条件都成立，则根据查询语句中的表和列（这里同样需要对查询内容从属哪一张表做处理），输出对应的查询结果；否则查询结果为空。

实现代码如下：

```
// 处理对两张表的查询
void process_queries_for_two_table(const vector<string>& table,
const vector<string>& attribute, const vector<string>& where) {
    for (auto& it1 : TABLE[table[0]]) {
        for (auto& it2 : TABLE[table[1]]) {
            int flag = 1; // 判断where之后的条件是否成立的标志

            // 判断where之后的条件是否成立
```

```
9                for (auto& w : where) {
10                   vector<string> t;
11                   // 针对不同的判别符进行处理
12                   if (w.find("!=") != -1) { // !=
13                       t = splitString_str (w, "!=");
14                       vector<string> temp = process_condition_for_two_table(t,table,it1 , it2 );
15                       string xx = temp[0],yy = temp[1];
16                       if (xx == yy) {
17                           flag = 0;
18                           break;
19                       }
20                   } else if (w.find(">=") != -1) { // >=
21                       t = splitString_str (w, ">=");
22                       vector<string> temp = process_condition_for_two_table(t,table,it1 , it2 );
23                       string xx = temp[0],yy = temp[1];
24                       if ( stoll (xx) < stoll (yy)) {
25                           flag = 0;
26                           break;
27                       }
28                   } else if (w.find("<=") != -1) { // <=
29                       t = splitString_str (w, "<=");
30                       vector<string> temp = process_condition_for_two_table(t,table,it1 , it2 );
31                       string xx = temp[0],yy = temp[1];
32                       if ( stoll (xx) > stoll (yy)) {
33                           flag = 0;
34                           break;
35                       }
36                   } else if (w.find("=") != -1) { // =
37                       t = splitString_str (w, "=");
38                       vector<string> temp = process_condition_for_two_table(t,table,it1 , it2 );
39                       string xx = temp[0],yy = temp[1];
40                       if (xx != yy) {
41                           flag = 0;
42                           break;
43                       }
```

```
44                  } else if (w.find(">") != -1) { // >
45                      t = splitString_str (w, ">");
46                      vector<string> temp = process_condition_for_two_table(t,table,it1 , it2 );
47                      string xx = temp[0],yy = temp[1];
48                      if ( stoll (xx) <= stoll(yy)) {
49                          flag = 0;
50                          break;
51                      }
52                  } else if (w.find("<") != -1) {// <
53                      t = splitString_str (w, "<");
54                      vector<string> temp = process_condition_for_two_table(t,table,it1 , it2 );
55                      string xx = temp[0],yy = temp[1];
56                      if ( stoll (xx) >= stoll(yy)) {
57                          flag = 0;
58                          break;
59                      }
60                  }
61              }
62
63              // 如果条件成立,输出查询结果
64              if (flag) {
65                  if ( attribute [0] == "*") { // 输出查询表中的所有列
66                      for (int i = 0; i < 7; i++) {
67                          if (it1[i] != "") {
68                              cout << it1[i] << " ";
69                          }
70                      }
71                      for (int i = 0; i < 7; i++) {
72                          if (it2[i] != "") {
73                              cout << it2[i] << " ";
74                          }
75                      }
76                      cout << endl;
77                  } else { // 输出查询表中的查询对应的列
78                      for (int i = 0; i < attribute . size (); i++) {
```

```
79                          string a = attribute[i];
80                          // 判断 TABLE.COLUMN ,输出对应TABLE的COLUMN
81                          vector<string> t = splitString(a, '.');
82                          if (t.size() == 1) {
83                              string x = it1[col_to_index[t[0]]] == "" ? it2[col_to_index[t[0]]] :
84                              it1[col_to_index[t[0]]];
85                              cout << x << " ";
86                          } else {
87                              string x = t[0] == table[0] ? it1[col_to_index[t[1]]] :
88                              it2[col_to_index[t[1]]];
89                              cout << x << " ";
90                          }
91                      }
92                      cout << endl;
93                  }
94              }
95          }
96      }
97  }
```

3 测试结果

3.1 【1.in 测试结果】

3.1.1 【1.in 测试数据】

```
3
2019001 law 19
2019002 info 20
2019003 info 19
2
20190001 math
20190002 English
2
2019101 info 49
```

```
2019102 info 38
2
2019002 20190001 100
2019003 20190002 0
1
20190001 2019102
4
SELECT * FROM Student
SELECT Student . dept FROM Student
SELECT Student.sid,Grade.sid,dept,score FROM Student,Grade
SELECT * FROM Student, Grade WHERE Grade .sid=Student. sid
```

3.1.2 【1.out 标准输出】

```
2019001 law 19
2019002 info 20
2019003 info 19
law
info
info
2019001 2019002 law 100
2019001 2019003 law 0
2019002 2019002 info 100
2019002 2019003 info 0
2019003 2019002 info 100
2019003 2019003 info 0
2019002 info 20 2019002 20190001 100
2019003 info 19 2019003 20190002 0
```

3.1.3 【1.in 运行结果】

运行结果与标准输出一致。

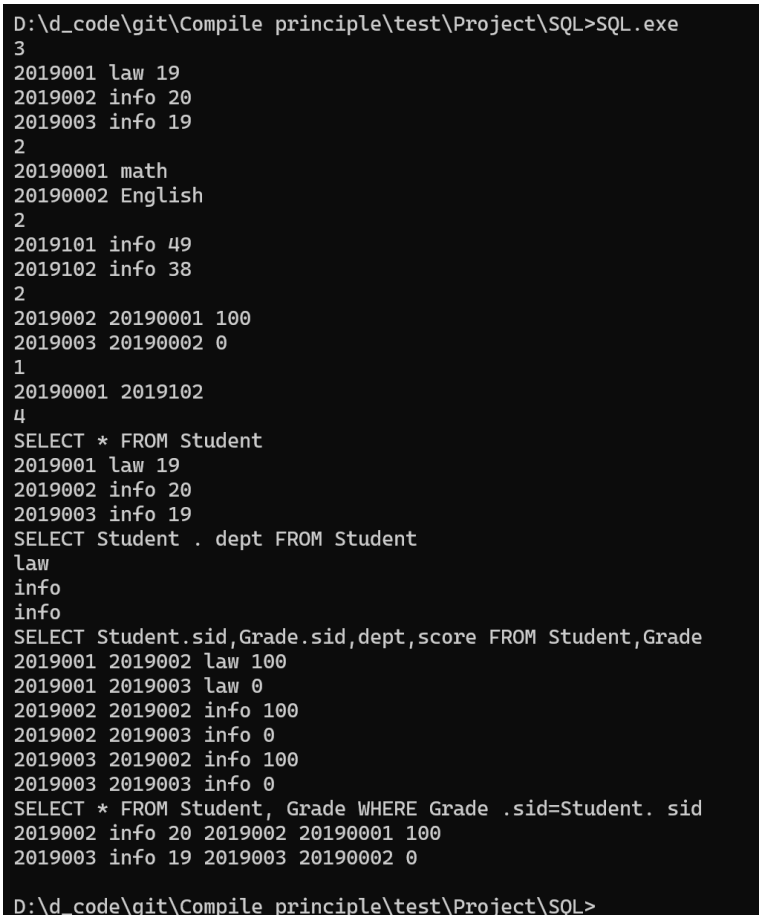
D:\d_code\git\Compile principle\test\Project\SQL>SQL.exe
3
2019001 law 19
2019002 info 20
2019003 info 19
2
20190001 math
20190002 English
2
2019101 info 49
2019102 info 38
2
2019002 20190001 100
2019003 20190002 0
1
20190001 2019102
4
SELECT * FROM Student
2019001 law 19
2019002 info 20
2019003 info 19
SELECT Student . dept FROM Student
law
info
info
SELECT Student.sid,Grade.sid,dept,score FROM Student,Grade
2019001 2019002 law 100
2019001 2019003 law 0
2019002 2019002 info 100
2019002 2019003 info 0
2019003 2019002 info 100
2019003 2019003 info 0
SELECT * FROM Student, Grade WHERE Grade .sid=Student. sid
2019002 info 20 2019002 20190001 100
2019003 info 19 2019003 20190002 0
D:\d_code\git\Compile principle\test\Project\SQL>

图 1: 【1.in 运行结果】

3.2 【2.in 测试结果】

3.2.1 【2.in 测试数据】

```
4
2019000001 law 19
2019000003 info 20
2019000002 info 19
2019010 math 28
4
2019010 math
2019011 math
2019012 Course
2019013 SELECT
4
2019010 math 0
2019000002 info 19
2019000001 math 100
2019000000 ABC 38
2
2019000001 2019010 90
2019010 2019010 99
1
2019013 2019000000
7
SELECT * FROM Grade
SELECT cid FROM Grade
SELECT cid,cid,Grade.cid, sid FROM Grade
SELECT * FROM Grade, Student
SELECT * FROM Grade, Student WHERE dept=" info"  AND Student.age
< 100 AND score> 90
SELECT Student.sid, dept, score FROM Grade,Student WHERE Student.sid
= Grade.sid
SELECT Student.sid, dept, score FROM Grade,Student WHERE Student.sid
= Grade.sid AND cid = " 2019"
```

3.2.2 【2.out 标准输出】

```
2019000001 2019010 90
2019010 2019010 99
2019010
2019010
2019010 2019010 2019010 2019000001
2019010 2019010 2019010 2019010
2019000001 2019010 90 2019000001 law 19
2019000001 2019010 90 2019000003 info 20
2019000001 2019010 90 2019000002 info 19
2019000001 2019010 90 2019010 math 28
2019010 2019010 99 2019000001 law 19
2019010 2019010 99 2019000003 info 20
2019010 2019010 99 2019000002 info 19
2019010 2019010 99 2019010 math 28
2019010 2019010 99 2019000003 info 20
2019010 2019010 99 2019000002 info 19
2019000001 law 90
2019010 math 99
```

3.2.3 【2.in 运行结果】

运行结果与标准输出一致。

4 后续可以进行的优化

只将表中所拥有的列插入表中，节省空间开销
针对对查询的处理函数，可以优化为对任意多张表的查询进行处理

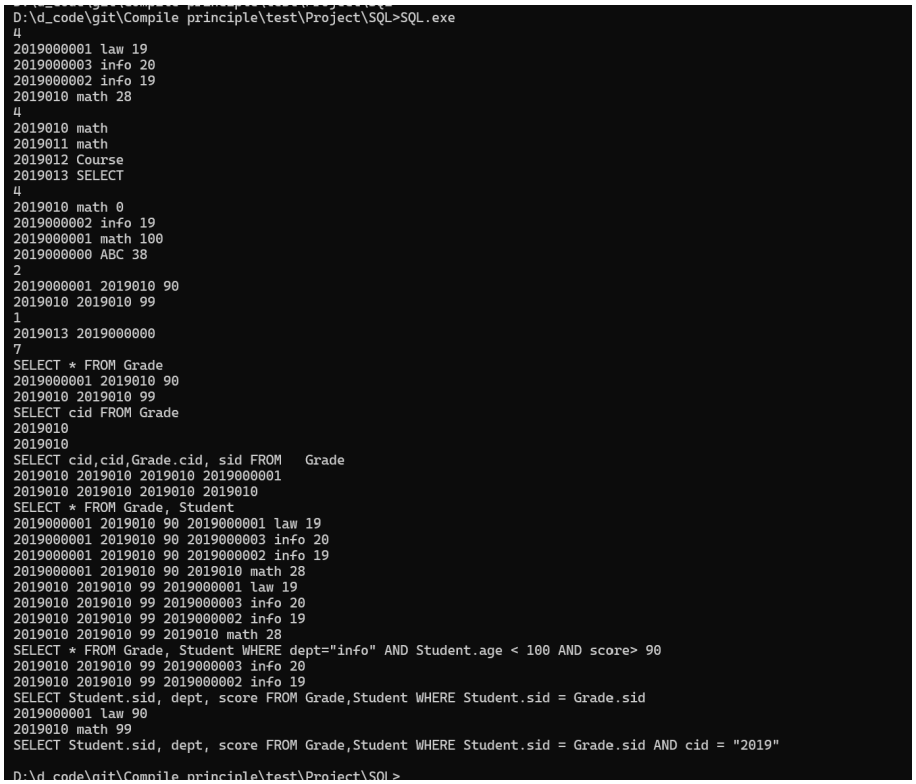
D:\d_code\git\Compile principle\test\Project\SQL>SQL.exe
4
2019000001 law 19
2019000003 info 20
2019000002 info 19
2019010 math 28
4
2019010 math
2019011 math
2019012 Course
2019013 SELECT
4
2019010 math 0
2019000002 info 19
2019000001 math 100
2019000000 ABC 38
2
2019000001 2019010 90
2019010 2019010 99
1
2019013 2019000000
7
SELECT * FROM Grade
2019000001 2019010 90
2019010 2019010 99
SELECT cid FROM Grade
2019010
2019010
SELECT cid,cid,Grade.cid, sid FROM Grade
2019010 2019010 2019010 2019000001
2019010 2019010 2019010 2019010
SELECT * FROM Grade, Student
2019000001 2019010 90 2019000001 law 19
2019000001 2019010 90 2019000003 info 20
2019000001 2019010 90 2019000002 info 19
2019000001 2019010 90 2019010 math 28
2019010 2019010 99 2019000001 law 19
2019010 2019010 99 2019000003 info 20
2019010 2019010 99 2019000002 info 19
2019010 2019010 99 2019010 math 28
SELECT * FROM Grade, Student WHERE dept="info" AND Student.age < 100 AND score> 90
2019010 2019010 99 2019000003 info 20
2019010 2019010 99 2019000002 info 19
SELECT Student.sid, dept, score FROM Grade,Student WHERE Student.sid = Grade.sid
2019000001 law 90
2019010 math 99
SELECT Student.sid, dept, score FROM Grade,Student WHERE Student.sid = Grade.sid AND cid = "2019"
D:\d_code\git\Compile principle\test\Project\SQL>

图 2: 【2.in 运行结果】