北京邮电大学 计算机学院

《数据库系统原理》实验报告

姓名 王睿嘉

学号2015211906

班级2015211307

实验六 数据查询分析

1. 实验内容和环境描述
2. 实验目的
3. 通过对不同查询语句执行情况的分析，巩固和加深对查询和查询优化相关理论知识的理解，提

高优化数据库系统的实践能力；

1. 熟悉了解MySQL中查询分析器的使用，进一步提高编写复杂查询程序的能力。
2. 实验内容
3. 索引对查询的影响
   1. 对结果集只有一个元组的查询，分三种情况执行：

不建立索引

（学号上）建立非聚集索引

（学号上）建立聚集索引

对执行步骤和结果进行分析比较；

* 1. 对结果集中有多个元组的查询，分类似①的三种情况进行比较；
  2. 对查询条件为一个连续范围的查询，分类似①的三种情况进行比较，注意系统处理的选择；
  3. 索引代价

在有索引和无索引的情况下，分别插入数据，比较执行效率。

1. 分析比较相同查询功能的查询语句
2. select avg(grade)

from sc

group by cno

having cno=100

select avg(grade)

from sc

where cno=100

有和没有group by，比较其查询效率，并分析；

1. select sno,sname,age

from student s1

where age=

(select max(age)

from student s2

where s1.dept=s2.dept

)

select dept,max(age) as maxAge into tmp

from student

group by dept

select sno,sname,age

from student,tmp

where student.age=tmp.maxAge and tmp.dept=student.dept

drop table tmp

重写后的查询一定比原始查询更优吗？通过执行分析结果；

1. 对下面两个查询进行比较

select sname,age

from student

where dept!=10 and age>all

(select age

from student

where dept=10

)

select sname,age

from student

where dept!=10 and age>

(select max(age)

from student

where dept=10

)；

1. 查询优化

除了建立适当索引，对SQL语句重写外，还有其他手段进行查询调优，例如调整缓冲区大小、事

先建立视图等。设计实现下列查询，使之运行效率最高。

写出查询形式及调优过程，并说明最优情况下的运行时间：

* 1. 查找选修了每一门课的学生；
  2. 查找至少选修了数据库原理和操作系统的学生的学号。

1. 实验环境

数据库系统：5.7.21 MySQL Community Server（GPL）

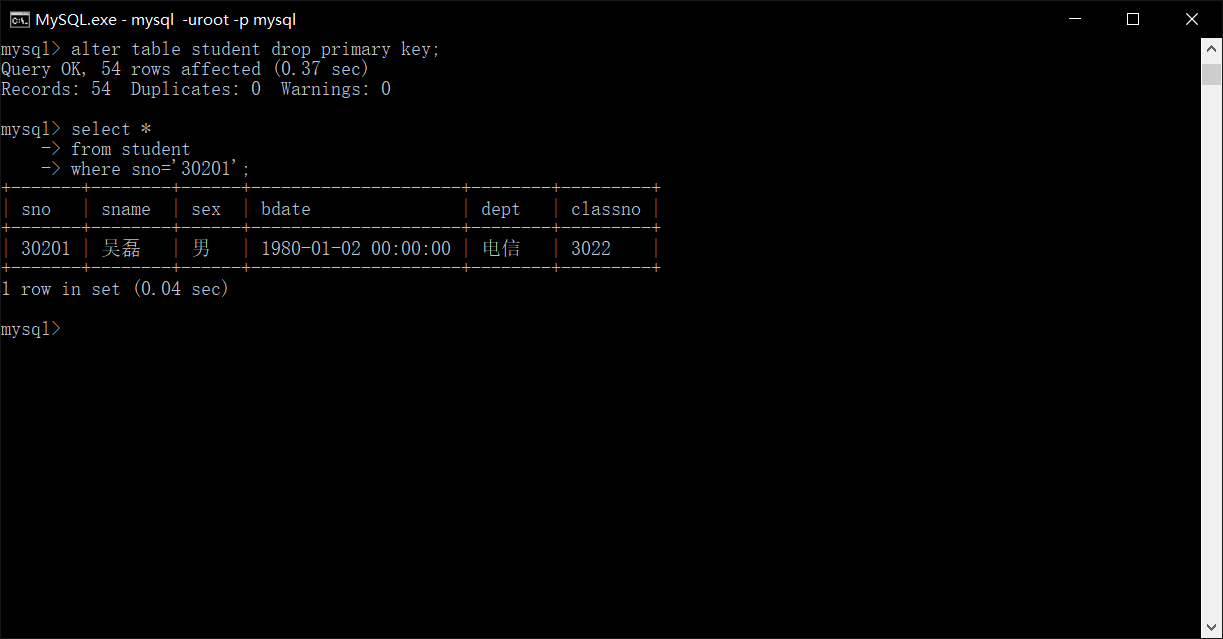
1. 实验结论和心得
2. 实验结论
   1. 对结果集只有一个元组的查询，分三种情况执行

以查询一个具体学生的信息为例：

1）不建立索引

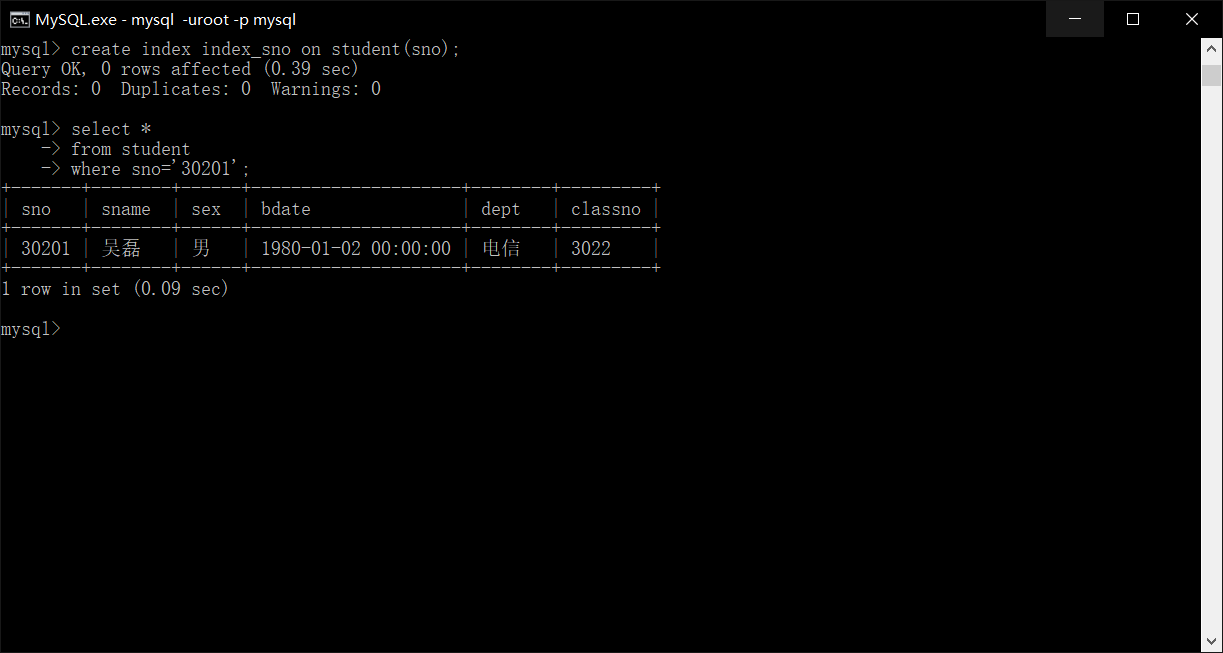
创建表时，由于设置了primary key，MySQL将自动生成主键索引，即聚集索引。

为了测试在不建立索引的情况下，查询学生信息的运行时间，，首先将主键索引删除。然后查询学号为30201的学生的信息：



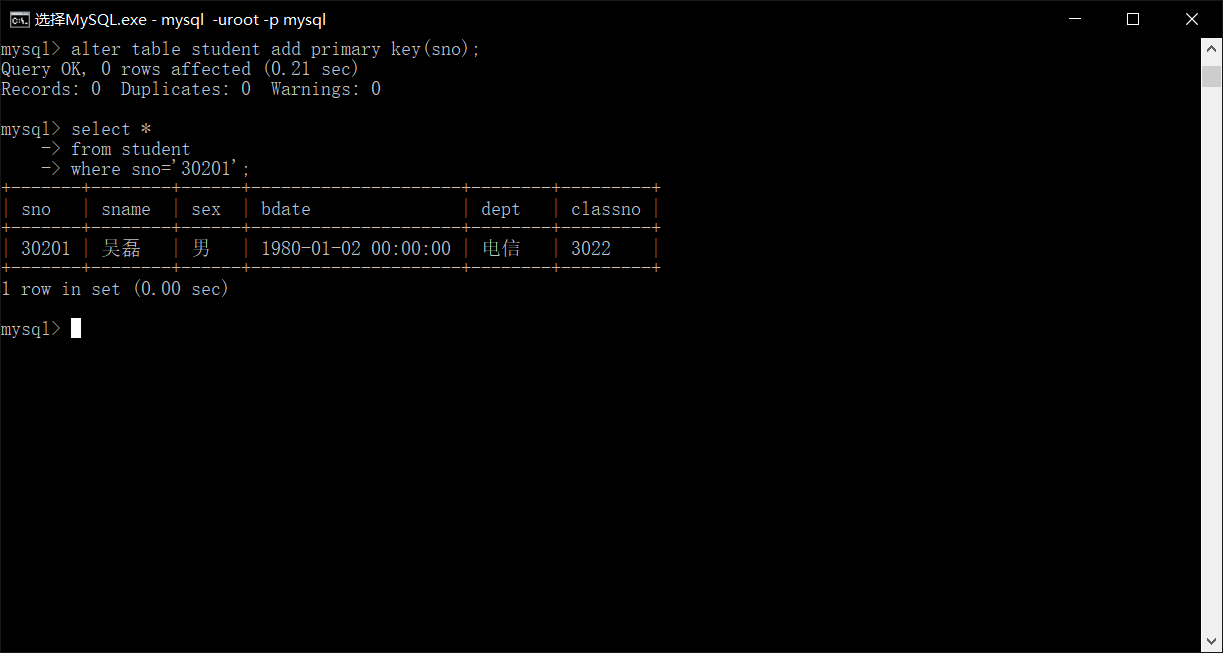
2）（学号上）建立非聚集索引

先建立非聚集索引，然后进行同样的查询操作：



3）（学号上）建立聚集索引

重新设置该表的主键，即聚集索引，然后进行同样的查询操作：



三种查询语句的时间对比如下：

|  |  |  |
| --- | --- | --- |
| 查询方法 | 时间 | 比较 |
| 不建立索引 | 0.04 |  |
|  |  |  |
|  |  |  |
|  |  |  |

* 1. ODBC实现方法

**初始化ODBC，获取相应环境句柄**

SQLHENV henv = SQL\_NULL\_HENV; //初始环境句柄

retcode = SQLAllocHandle(SQL\_HANDLE\_ENV, SQL\_NULL\_HANDLE, &henv); //初始化ODBC环境，分配环境句柄

retcode = SQLSetEnvAttr(henv, SQL\_ATTR\_ODBC\_VERSION, (SQLPOINTER)SQL\_OV\_ODBC3, SQL\_IS\_INTEGER);

无论程序建立多少个ODBC连接，此过程只需执行一次。

**与ODBC数据源建立连接**

SQLHDBC hdbc = SQL\_NULL\_HDBC; //初始化连接句柄

retcode = SQLAllocHandle(SQL\_HANDLE\_DBC, henv, &hdbc);

//为连接句柄分配内存并返回

retcode = SQLConnect(hdbc, szDSN, SQL\_NTS, userID, SQL\_NTS, passWORD, SQL\_NTS);

//连接数据库

**向ODBC数据源提交SQL语句，实现数据库操作**

SQLHSTMT hstmt = SQL\_NULL\_HSTMT; //初始化语句句柄

retcode = SQLAllocStmt(hdbc, &hstmt); //为语句句柄分配内存，并返回

retcode = SQLExecDirect(hstmt, sqlOper, SQL\_NTS);

//向ODBC数据源提交SQL语句，实现数据库操作

**结束应用程序**

应用程序完成数据库操作，退出运行之前，需要释放程序中使用的系统资源。这些资源包括：语句句柄、连接句柄和环境句柄等。

SQLFreeStmt(hstmt, SQL\_DROP); //释放语句句柄

SQLDisconnect(hdbc); //切断连接

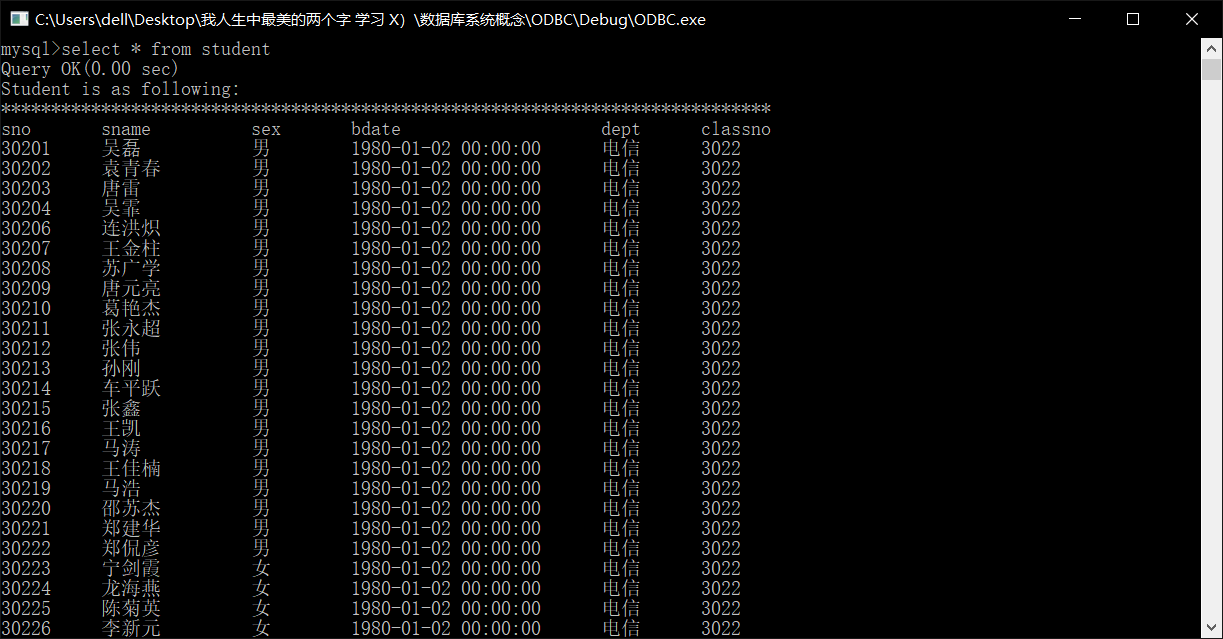
SQLFreeConnect(hdbc); //释放与连接句柄相关的资源

SQLFreeEnv(henv); //释放与环境句柄相关的资源

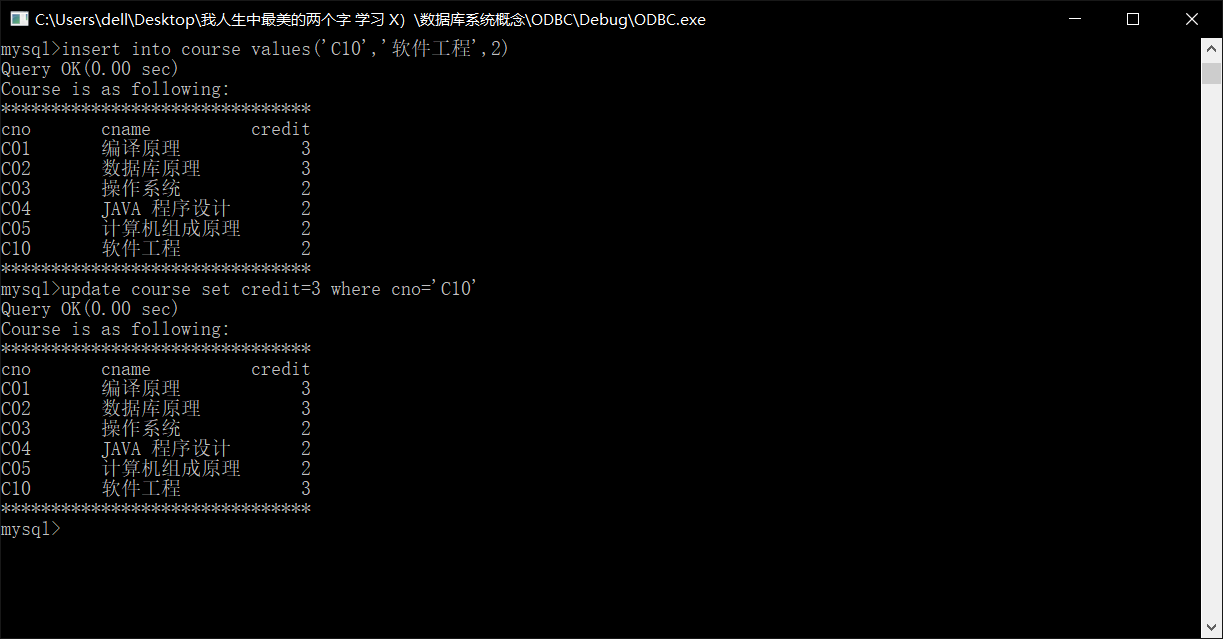
* 1. 程序源码

1. #include <iostream>
2. #include <string>
3. #include <windows.h>
4. #include <sql.h>
5. #include <sqlext.h>
6. #include <sqltypes.h>
7. #include <odbcss.h>
8. using namespace std;
9. #pragma comment (lib,"odbc32.lib")
10. #pragma comment (lib,"libmysql.lib")
11. #define MAXBUFLEN 255
12. /\*初始化 ODBC\*/
13. SQLHENV henv = SQL\_NULL\_HENV; //初始化环境句柄
14. SQLHDBC hdbc = SQL\_NULL\_HDBC; //初始化连接句柄
15. SQLHSTMT hstmt = SQL\_NULL\_HSTMT; //初始化语句句柄
16. /\*结构\*/
17. struct Course{
18. char cno[4];
19. char cname[15];
20. int credit;
21. };
22. struct Student{
23. char sno[7];
24. char sname[7];
25. char sex[3];
26. char bdate[20];
27. char dept[9];
28. char classno[5];
29. };
30. struct Sc{
31. char sno[7];
32. char cno[4];
33. int grade;
34. };
35. unsigned char sqlSelect[3][100] = { "select \* from course", "select \* from student", "select \* from sc" };
36. unsigned char sqlOper[MAXBUFLEN];
37. /\*打印数据库信息\*/
38. void priCourse(HDBC);
39. void priStudent(HDBC);
40. void priSc(HDBC);
41. int main(){
42. RETCODE retcode; //定义返回代码
43. UCHAR szDSN[10] = "mysqlodbc"; //数据源
44. UCHAR userID[5] = "root"; //用户名
45. UCHAR passWORD[7] = "142857"; //密码
46. string s;
48. retcode = SQLAllocHandle(SQL\_HANDLE\_ENV, SQL\_NULL\_HANDLE, &henv); //初始化ODBC环境，分配环境句柄
49. retcode = SQLSetEnvAttr(henv, SQL\_ATTR\_ODBC\_VERSION, (SQLPOINTER)SQL\_OV\_ODBC3, SQL\_IS\_INTEGER);
50. retcode = SQLAllocHandle(SQL\_HANDLE\_DBC, henv, &hdbc); //为连接句柄分配内存并返回
51. retcode = SQLConnect(hdbc, szDSN, SQL\_NTS, userID, SQL\_NTS, passWORD, SQL\_NTS);
52. //连接数据库
53. if ((retcode != SQL\_SUCCESS) && (retcode != SQL\_SUCCESS\_WITH\_INFO))
54. cout << "Connect failed!" << endl;
55. else
56. while (true) {
57. cout << "mysql>";
58. getline(cin, s);
59. if (s != "exit") {
60. strcpy\_s((char\*)sqlOper, MAXBUFLEN, s.c\_str());
61. retcode = SQLAllocStmt(hdbc, &hstmt);
62. retcode = SQLExecDirect(hstmt, sqlOper, SQL\_NTS);
63. //向ODBC数据源提交SQL语句，实现数据库操作
64. if (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO)
65. cout << "Query OK(0.00 sec)" << endl;
66. else
67. cout << "ERROR" << endl;
68. SQLFreeStmt(hstmt, SQL\_DROP);
69. if (s.find("course") != string::npos)
70. priCourse(hdbc);
71. else if (s.find("student") != string::npos)
72. priStudent(hdbc);
73. else if (s.find("sc") != string::npos)
74. priSc(hdbc);
75. }
76. else {
77. cout << "Bye~" << endl;
78. SQLDisconnect(hdbc); //切断连接
79. SQLFreeConnect(hdbc); //释放与连接句柄相关的资源
80. SQLFreeEnv(henv); //释放与环境句柄相关的资源
81. break;
82. }
83. }
84. system("pause");
85. return 0;
86. }
87. void priCourse(HDBC hdbc){
88. HSTMT hstmt;
89. RETCODE retcode;
90. long lenOut1, lenOut2, lenOut3;
91. struct Course\* ccell;
92. ccell = (struct Course\*)malloc(sizeof(struct Course));
93. retcode = SQLAllocStmt(hdbc, &hstmt);
94. if (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO){
95. retcode = SQLBindCol(hstmt, 1, SQL\_C\_CHAR, ccell->cno, 4, &lenOut1);
96. //将应用程序的数据缓冲绑定到结果集各列
97. retcode = SQLBindCol(hstmt, 2, SQL\_C\_CHAR, ccell->cname, 15, &lenOut2);
98. retcode = SQLBindCol(hstmt, 3, SQL\_C\_SLONG, &ccell->credit, sizeof(int), &lenOut3);
99. retcode = SQLExecDirect(hstmt, sqlSelect[0], SQL\_NTS);
100. //将SQL语句送至数据库服务器，执行由其定义的访问
102. if (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO){
103. cout << "Course is as following:" << endl;
104. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
105. cout << "cno cname credit" << endl;
106. retcode = SQLFetch(hstmt); //相当于SQLFetchAdvances和SQLGetData两个函数的功能
108. while (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO){
109. printf("%-10s%-20s%-16d\n", ccell->cno, ccell->cname, ccell->credit);
110. retcode = SQLFetch(hstmt);
111. }
112. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
113. }
114. }
115. SQLFreeStmt(hstmt, SQL\_DROP); //释放语句句柄
116. }
117. void priStudent(HDBC hdbc){
118. HSTMT hstmt;
119. RETCODE retcode;
120. long lenOut1, lenOut2, lenOut3, lenOut4, lenOut5, lenOut6;
121. struct Student\* scell;
122. scell = (struct Student\*)malloc(sizeof(struct Student));
123. retcode = SQLAllocStmt(hdbc, &hstmt);
124. if (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO){
125. retcode = SQLBindCol(hstmt, 1, SQL\_C\_CHAR, scell->sno, 7, &lenOut1);
126. retcode = SQLBindCol(hstmt, 2, SQL\_C\_CHAR, scell->sname, 7, &lenOut2);
127. retcode = SQLBindCol(hstmt, 3, SQL\_C\_CHAR, &scell->sex, 3, &lenOut3);
128. retcode = SQLBindCol(hstmt, 4, SQL\_C\_CHAR, &scell->bdate, 20, &lenOut4);
129. retcode = SQLBindCol(hstmt, 5, SQL\_C\_CHAR, scell->dept, 9, &lenOut5);
130. retcode = SQLBindCol(hstmt, 6, SQL\_C\_CHAR, scell->classno, 5, &lenOut6);
131. retcode = SQLExecDirect(hstmt, sqlSelect[1], SQL\_NTS);
133. if (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO){
134. cout << "Student is as following:" << endl;
135. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
136. cout << "sno sname sex bdate dept classno" << endl;
137. retcode = SQLFetch(hstmt);
139. while (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO){
140. printf("%-10s%-15s%-10s%-25s%-10s%-15s\n", scell->sno, scell->sname, scell->sex, scell->bdate, scell->dept, scell->classno);
141. retcode = SQLFetch(hstmt);
142. }
143. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
144. }
145. }
146. SQLFreeStmt(hstmt, SQL\_DROP);
147. }
148. void priSc(HDBC hdbc) {
149. HSTMT hstmt;
150. RETCODE retcode;
151. long lenOut1, lenOut2, lenOut3;
152. struct Sc\* cell;
153. cell = (struct Sc\*)malloc(sizeof(struct Sc));
154. retcode = SQLAllocStmt(hdbc, &hstmt);
155. if (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO) {
156. retcode = SQLBindCol(hstmt, 1, SQL\_C\_CHAR, cell->sno, 7, &lenOut1);
157. retcode = SQLBindCol(hstmt, 2, SQL\_C\_CHAR, cell->cno, 4, &lenOut2);
158. retcode = SQLBindCol(hstmt, 3, SQL\_C\_SLONG, &cell->grade, sizeof(int), &lenOut3);
159. retcode = SQLExecDirect(hstmt, sqlSelect[2], SQL\_NTS);
160. if (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO) {
161. cout << "Sc is as following:" << endl;
162. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
163. cout << "sno cno grade" << endl;
164. retcode = SQLFetch(hstmt);
165. while (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO) {
166. printf("%-10s%-16s%d\n", cell->sno, cell->cno, cell->grade);
167. retcode = SQLFetch(hstmt);
168. }
169. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
170. }
171. }
172. SQLFreeStmt(hstmt, SQL\_DROP);
173. }
     1. 运行结果

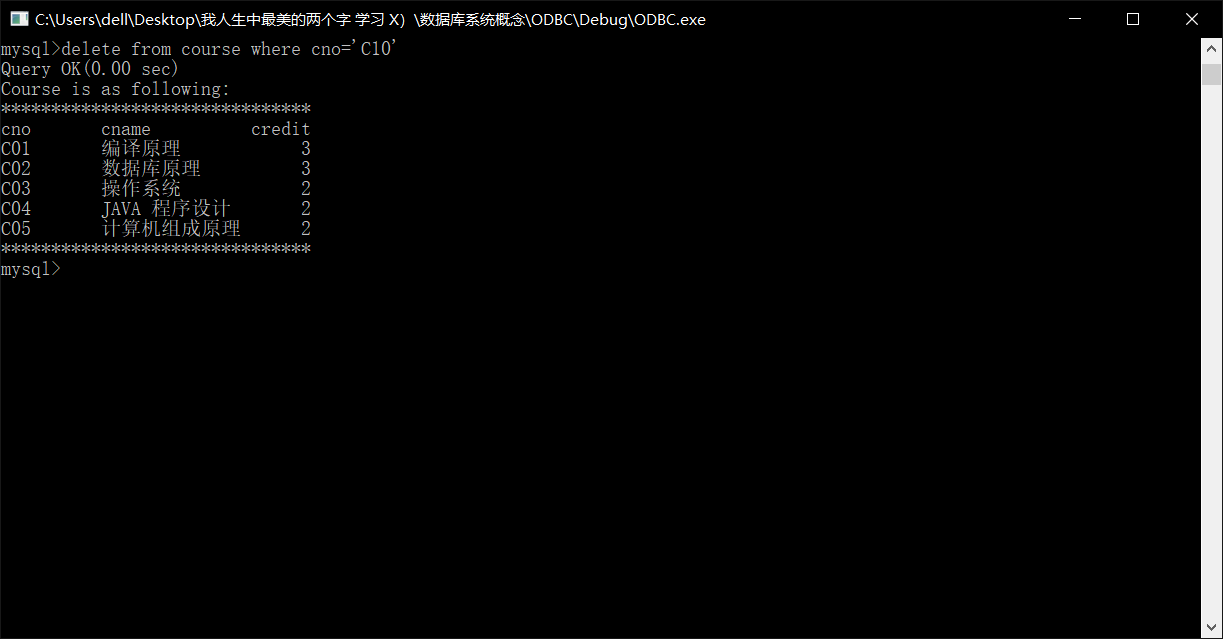
查询操作：



插入、更新操作：



删除操作：



应用程序退出：



1. 实验心得

在本次实验中，遇到的主要问题有以下四点：

* 1. ODBC数据源命名。根据网上相关教程，命名ODBC数据源是任意的。起初，我直接将其命名为ODBC，测试通过却始终出现无效DSN提示，可能是与关键词冲突导致。最终，更改名称，该问题得到解决；
  2. 数据源字长。起初使用64位ODBC，无法连接成功。最终，重新安装32位驱动，该问题得到解决；
  3. <mysql.h>库的使用。该库操作数据源极为方便，但64位头文件与32位IDE不匹配，最终决定采用实验指导书所推荐的<sql.h>；
  4. 中文乱码。数据库查询结果中，中文字符均为乱码。将数据源字符集改为GBK后，该问题得到解决；

实践出真知，本次数据库接口实验是对课堂和书本所学知识的补充。通过自己动手、亲力亲为编写数据库应用程序，加深了对ODBC的理解和记忆，收获颇丰。