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=’C01’

select avg(grade)

from sc

where cno=’C01’

有和没有group by，比较其查询效率，并分析；

1. select sno,sname,bdate

from student as s1

where bdate=

(select max(bdate)

from student as s2

where s1.dept=s2.dept

)

create table tmp as(

select dept,max(bdate) as maxBdate

from student

group by dept);

select sno,sname,bdate

from student,tmp

where student.bdate=tmp.maxBdate 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,bdate

from student

where dept!=’计算机’ and bdate>all

(select bdate

from student

where dept=’计算机’

)

select sname,bdate

from student

where dept!=’计算机’ and bdate>

(select max(bdate)

from student

where dept=’计算机’

)；

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）

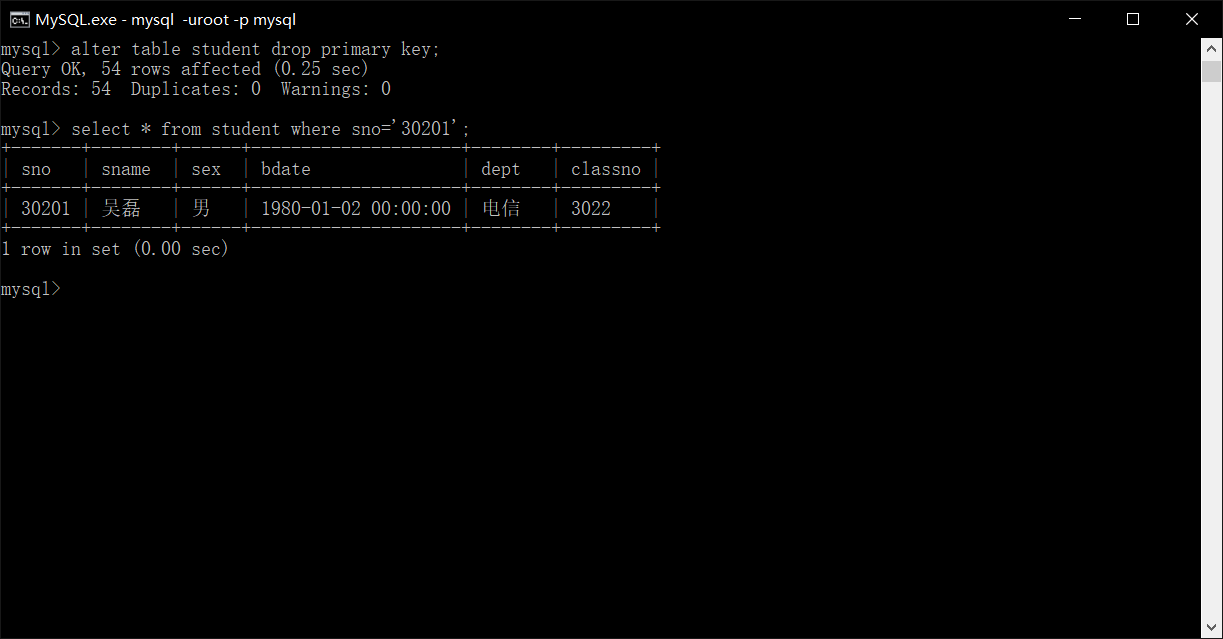
1. 实验结论和心得
2. 实验结论
   1. 对结果集只有一个元组的查询，分三种情况执行

以查询一个具体学生的信息为例：

1）不建立索引

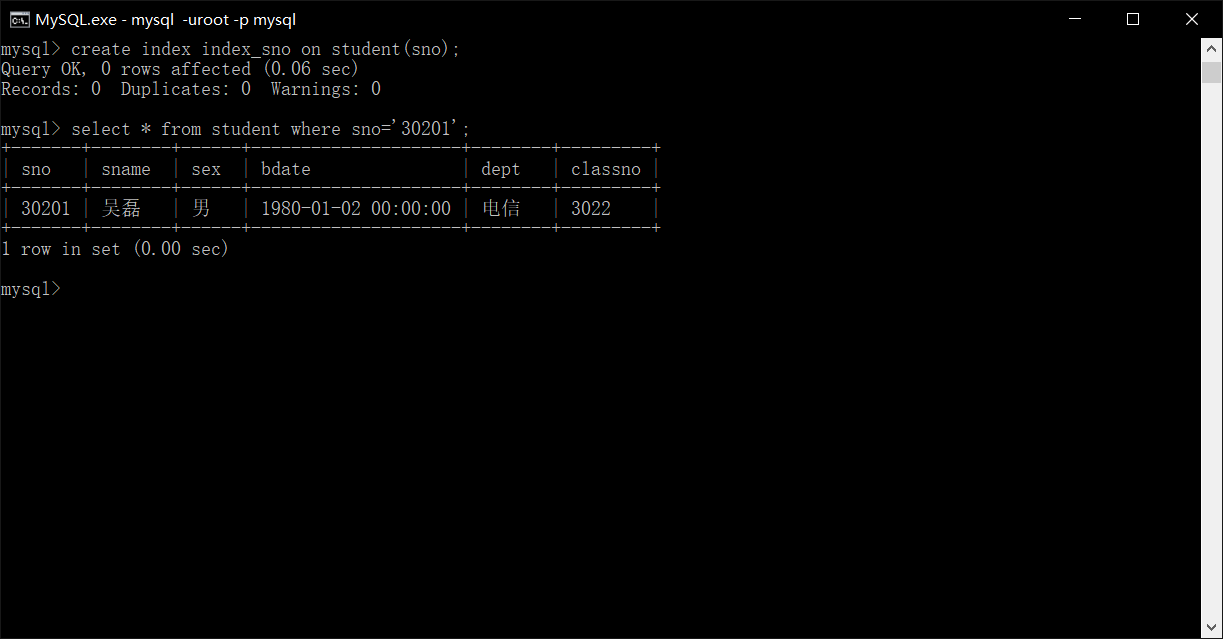
创建表时，由于设置了primary key，MySQL将自动生成主键索引，即聚集索引。

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



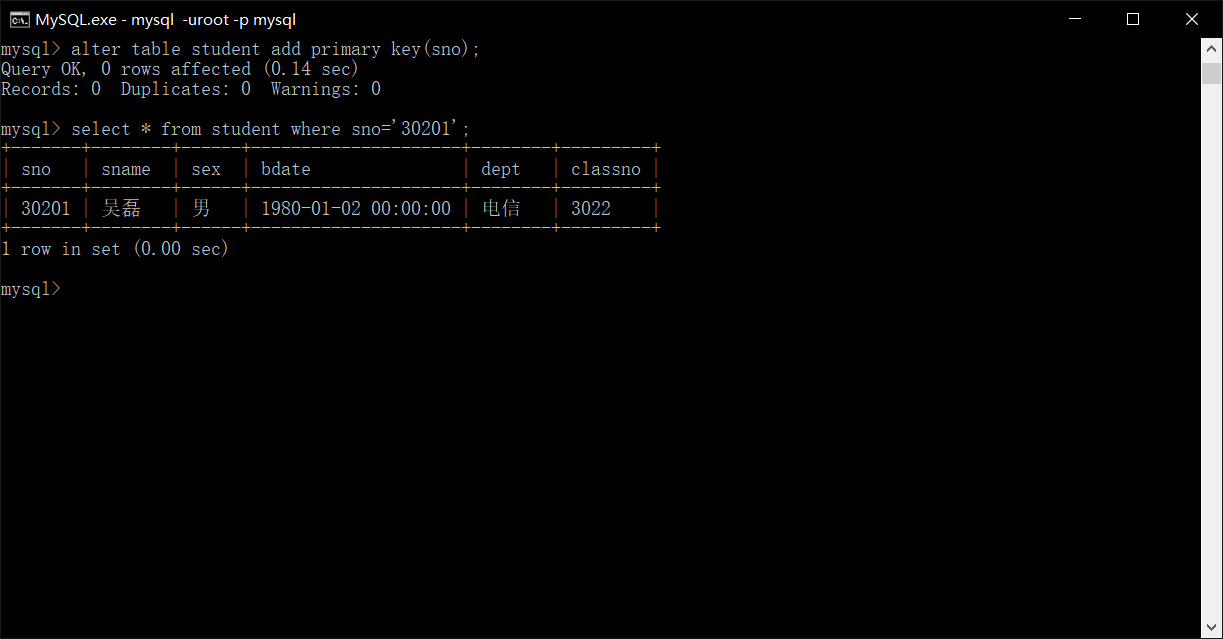
2）（学号上）建立非聚集索引

先建立非聚集索引，然后进行同样的查询操作：

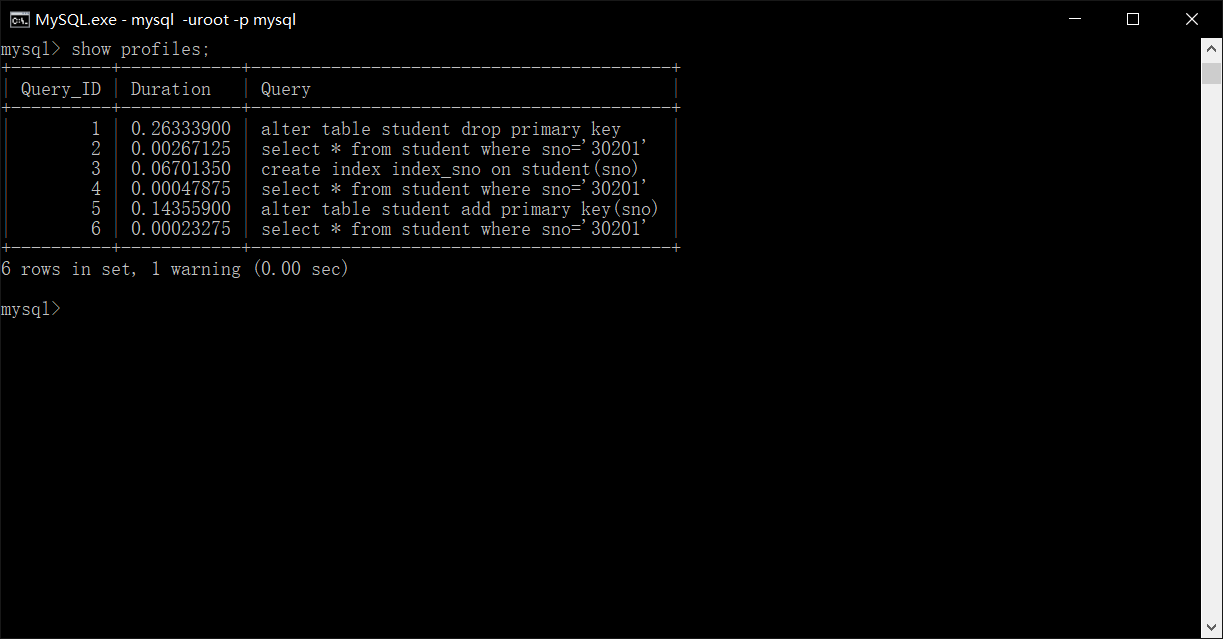


3）（学号上）建立聚集索引

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 | 0.00267125 | 时间最长 |
| （学号上）建立非聚集索引 | 0.00047875 | 时间较短 |
| （学号上）建立聚集索引 | 0.00023275 | 时间最短 |

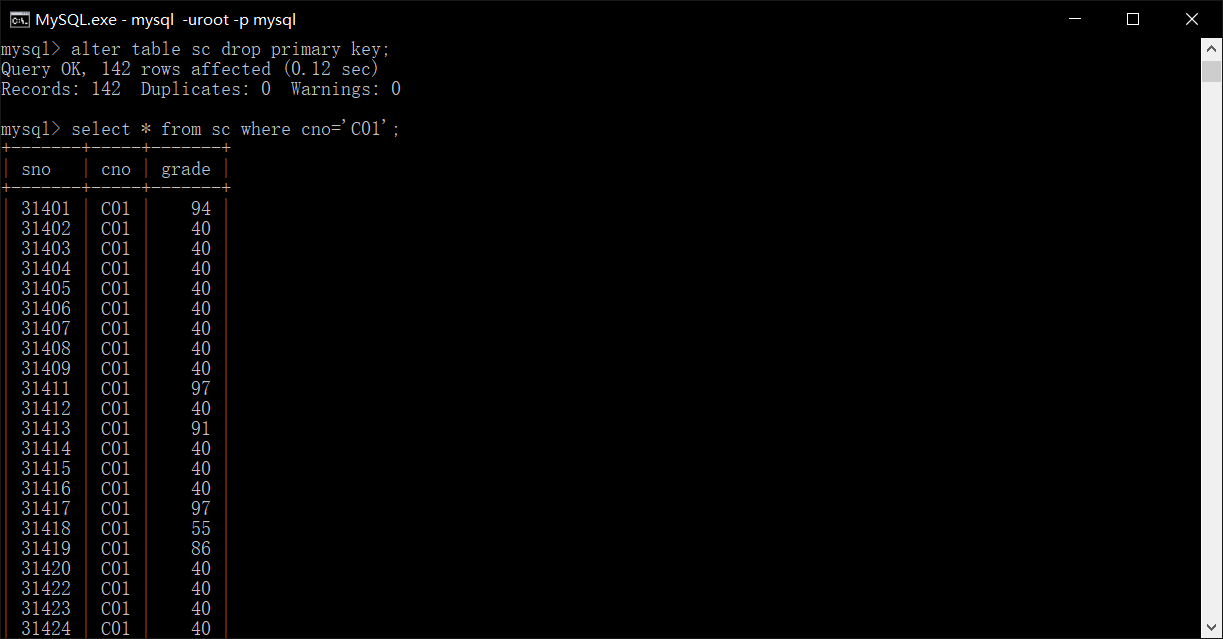
* 1. 对结果集中有多个元组的查询，分三种情况执行

以查询某课程的成绩表为例：

1）不建立索引

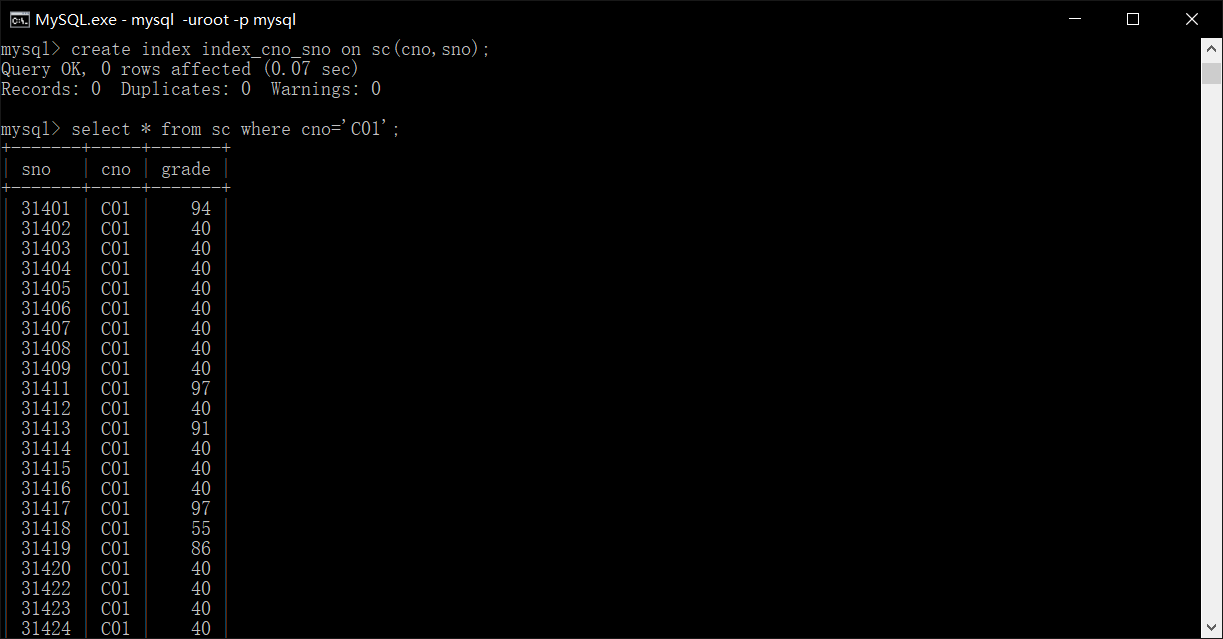
为了测试在不建立索引的情况下，查询成绩表的执行时间，首先将主键索引删除。然后查询课程号为

C01的课程的成绩表信息：



2）（课程号，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：

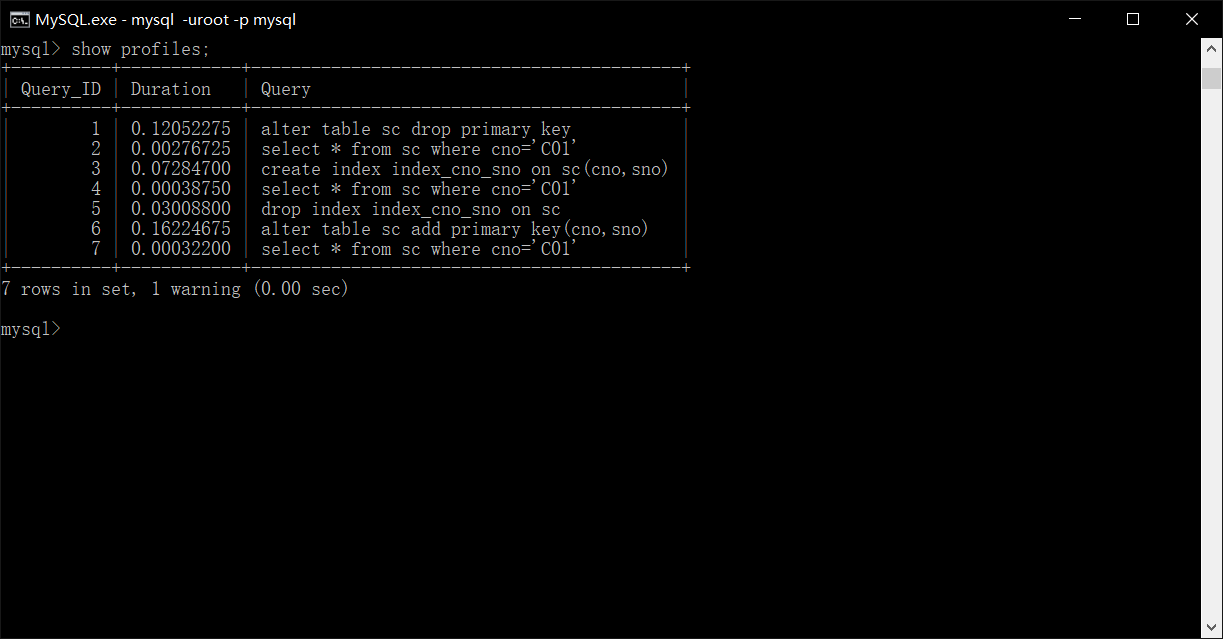


3）（课程号，学号上）建立聚集索引

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 | 0.00276725 | 时间最长 |
| （课程号，学号上）建立非聚集索引 | 0.00038750 | 时间较短 |
| （课程号，学号上）建立聚集索引 | 0.00032200 | 时间最短 |

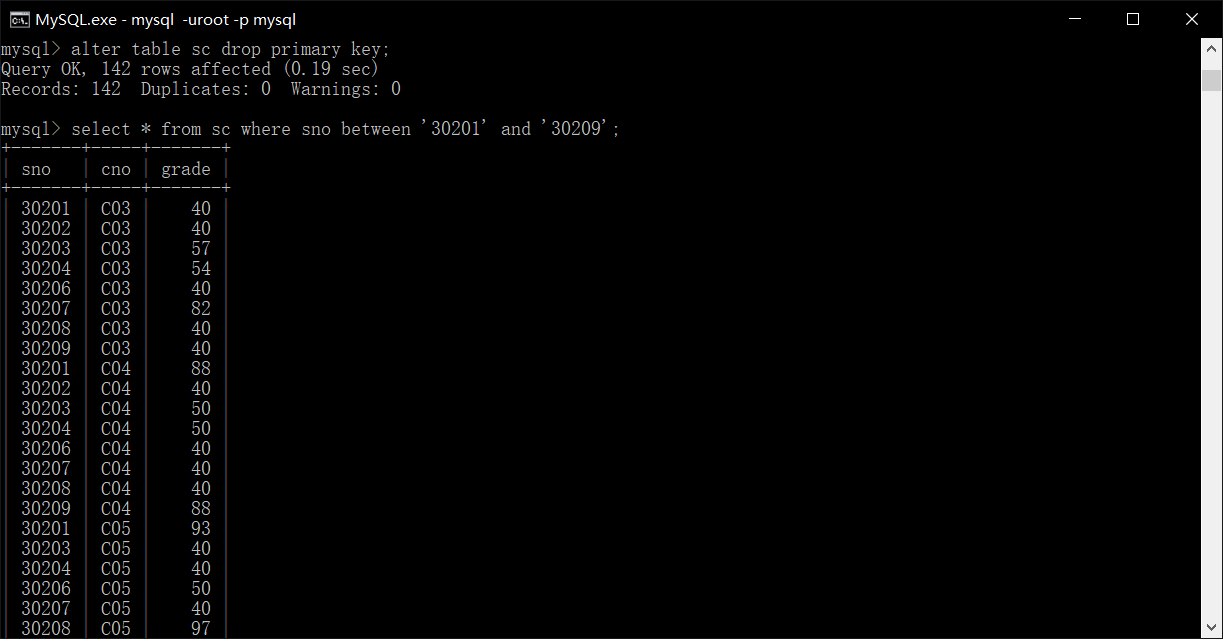
* 1. 对查询条件为一个连续范围的查询，分三种情况执行

以查询学号在某范围内的学生选课情况为例：

1）不建立索引

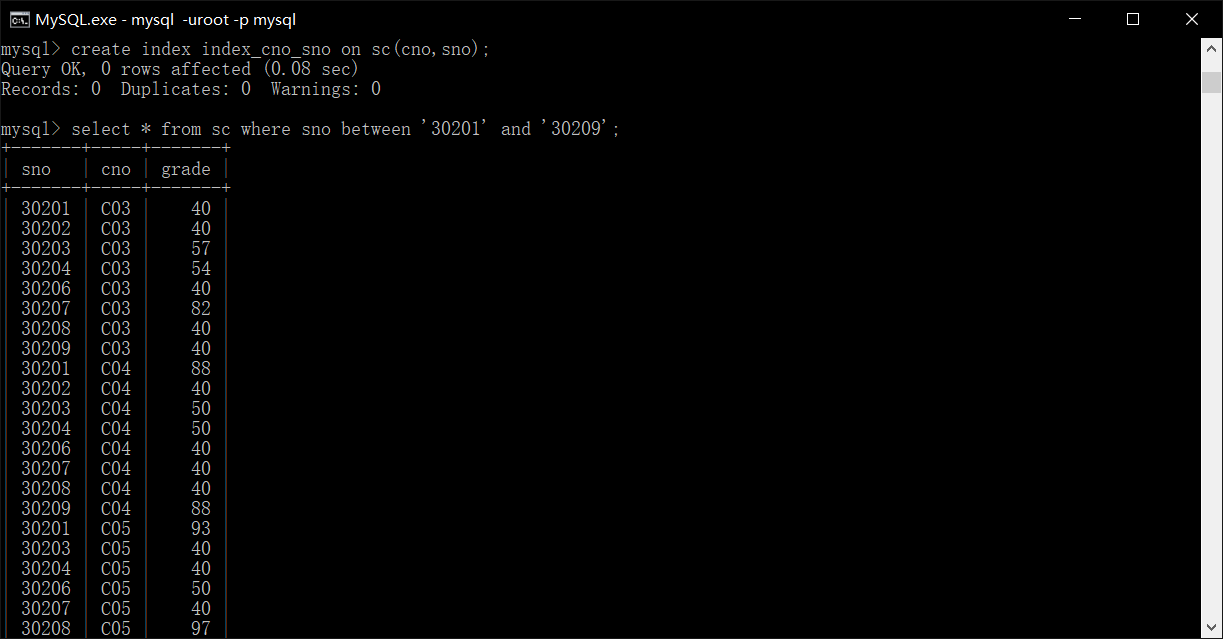
为了测试在不建立索引的情况下，查询学生选课情况的执行时间，首先将主键索引删除。然后查询学

号在30201-30209的学生的选课情况：



2）（课程号，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：

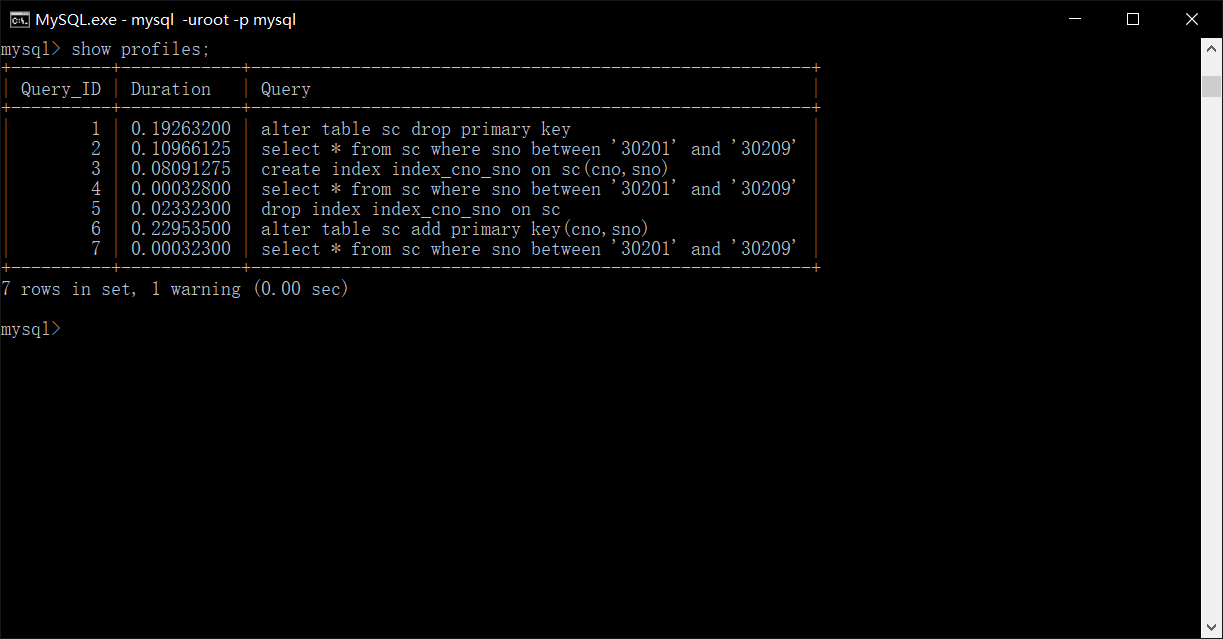


3）（课程号，学号上）建立聚集索引

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：



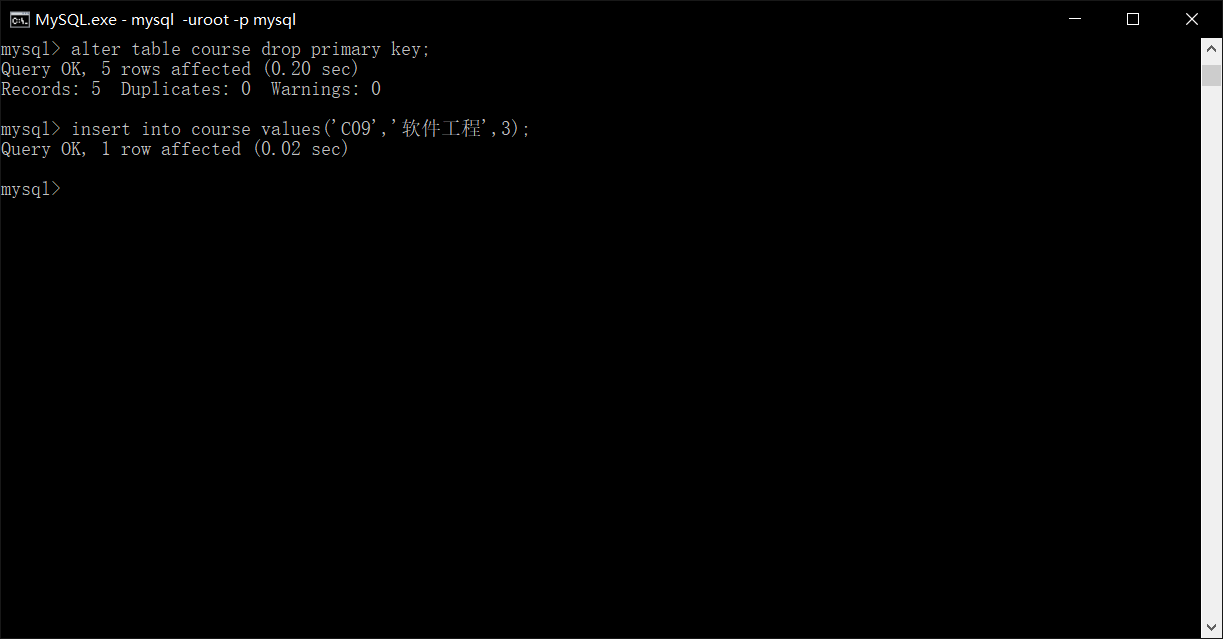
|  |  |  |
| --- | --- | --- |
| 查询方法 | 时间 | 比较 |
| 不建立索引 | 0.10966125 | 时间最长 |
| （课程号，学号上）建立非聚集索引 | 0.00032800 | 时间较短 |
| （课程号，学号上）建立聚集索引 | 0.00032300 | 时间最短 |

* 1. 索引代价

以在课程信息表course中插入数据为例：

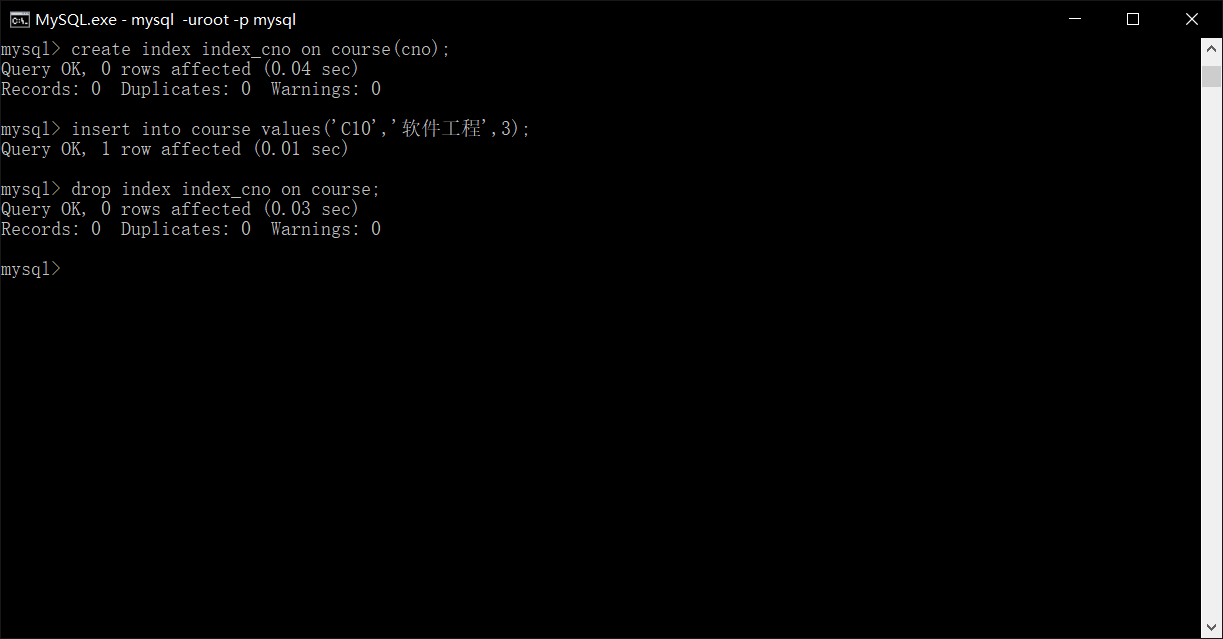
1）不建立索引

为了测试在不建立索引的情况下，插入课程信息的执行时间，首先将主键索引删除：



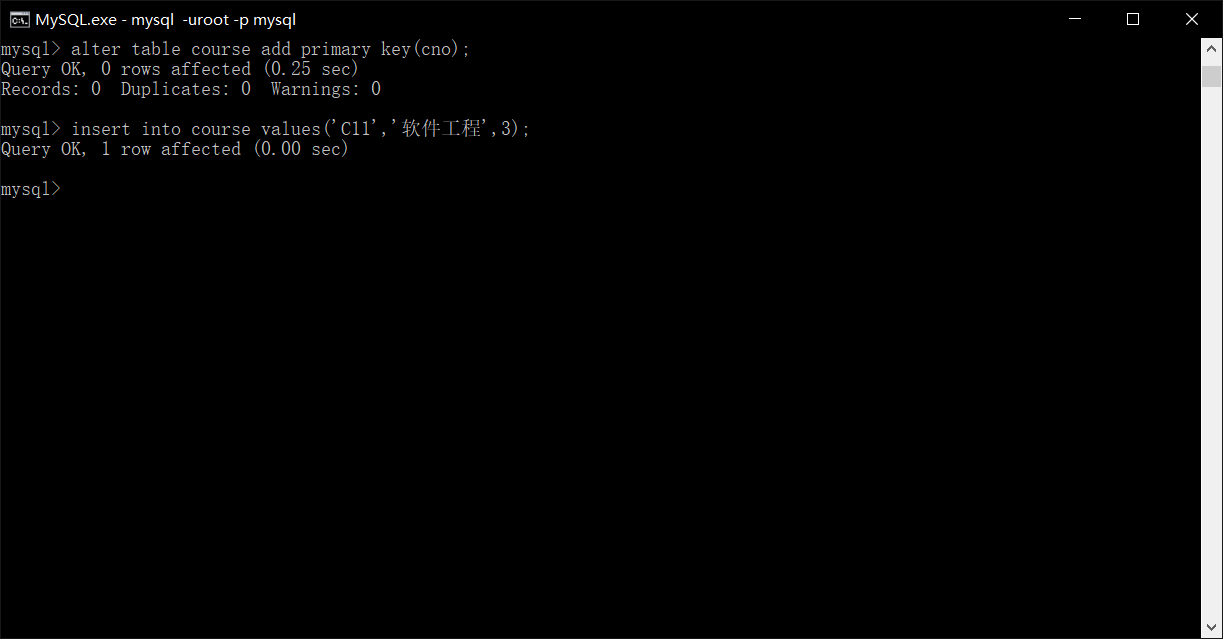
2）（课程号上）建立非聚集索引

先建立非聚集索引，然后进行同样的插入操作：

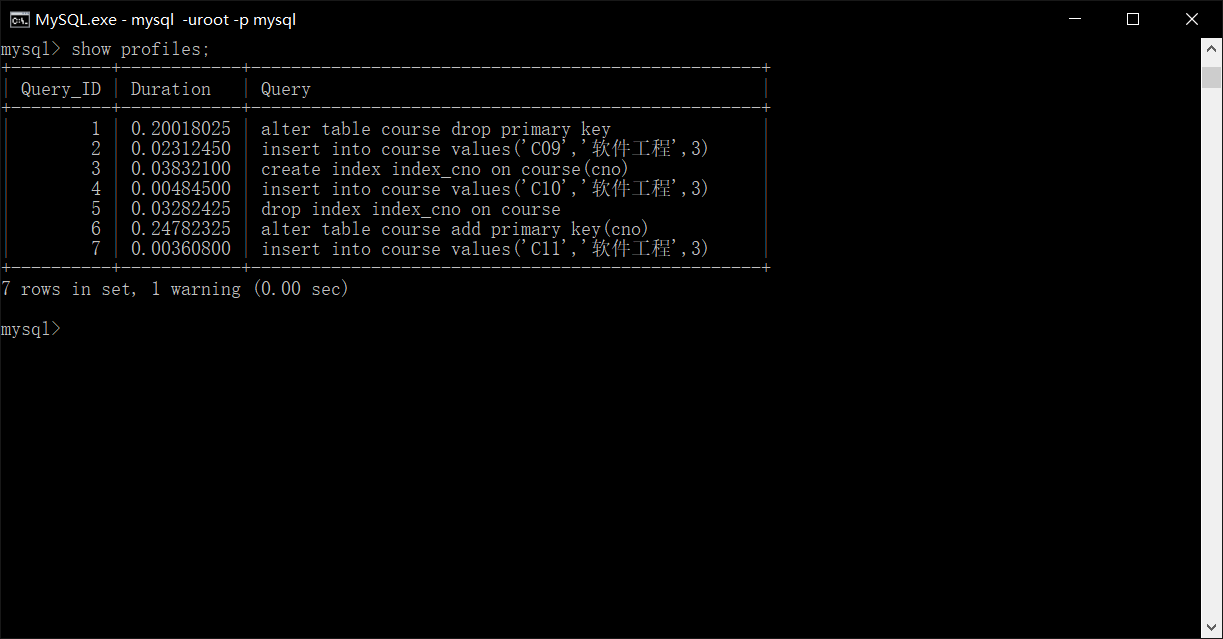


3）（课程号上）建立聚集索引

重新设置该表的主键，即聚集索引，然后进行同样的插入操作：



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 | 0.02312450 | 时间最长 |
| （课程号上）建立非聚集索引 | 0.00484500 | 时间较短 |
| （课程号上）建立聚集索引 | 0.00360800 | 时间最短 |

* 1. 有和没有group by，比较其查询效率

1）select avg(grade)

from sc

group by cno

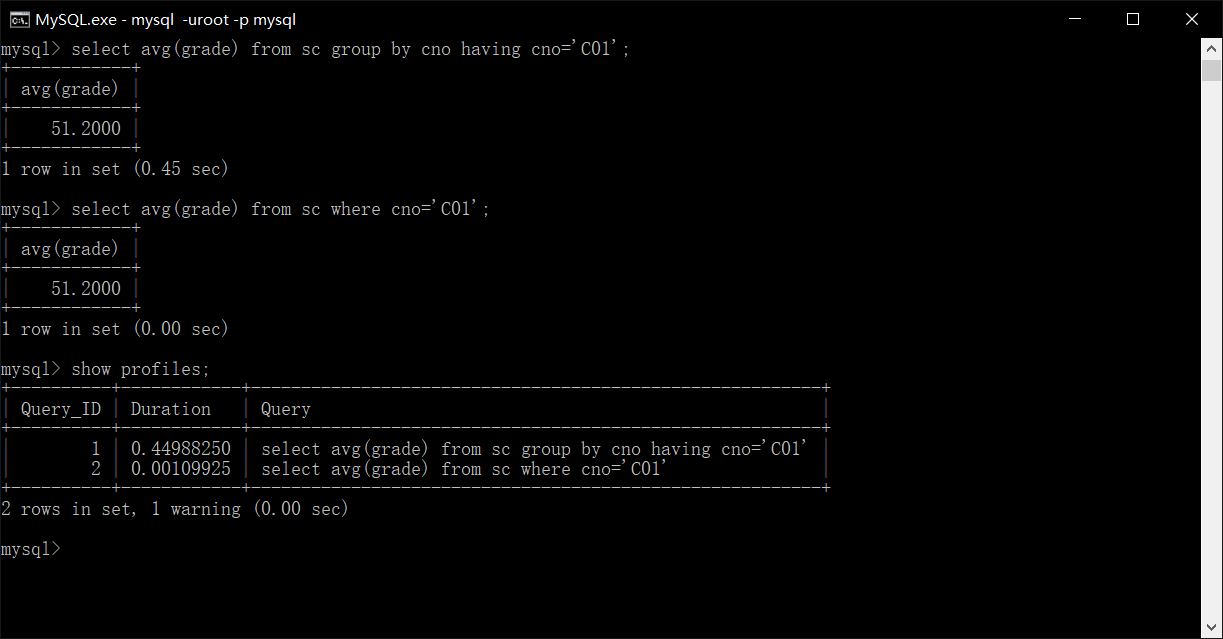
having cno=’C01’

2）select avg(grade)

from sc

where cno=’C01’

两条语句的运行结果相同，但是时间相差极大，如下图所示：



|  |  |  |
| --- | --- | --- |
| 查询语句 | 时间 | 比较 |
| 有group by | 0.44988250 | 长 |
| 没有group by | 0.00109925 | 短 |

* 1. 重写后的查询一定比原始查询更优吗？

1）select sno,sname,bdate

from student as s1

where bdate=

(select max(bdate)

from student as s2

where s1.dept=s2.dept

)

2）create table tmp as(

select dept,max(bdate) as maxBdate

from student

group by dept);

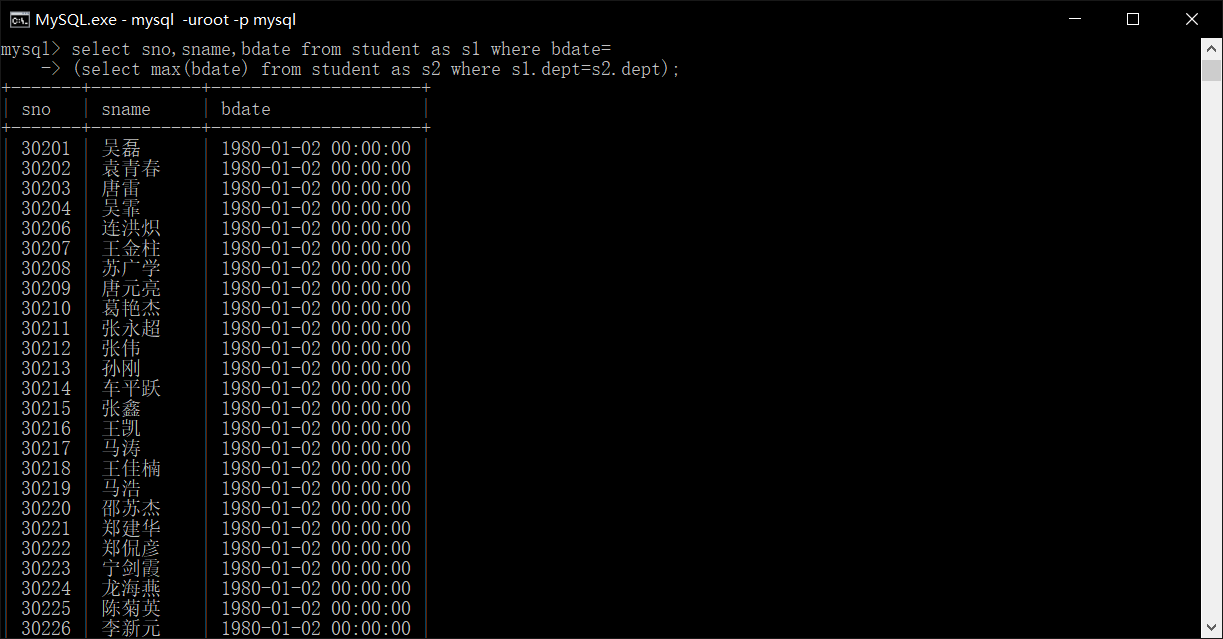
select sno,sname,bdate

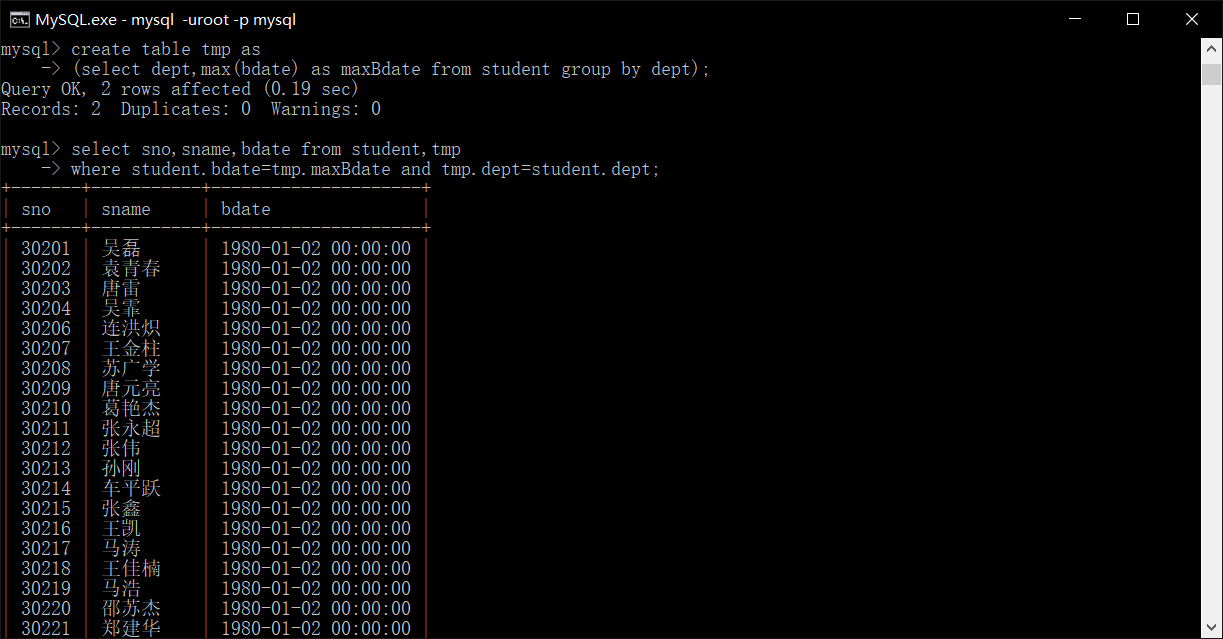
from student,tmp

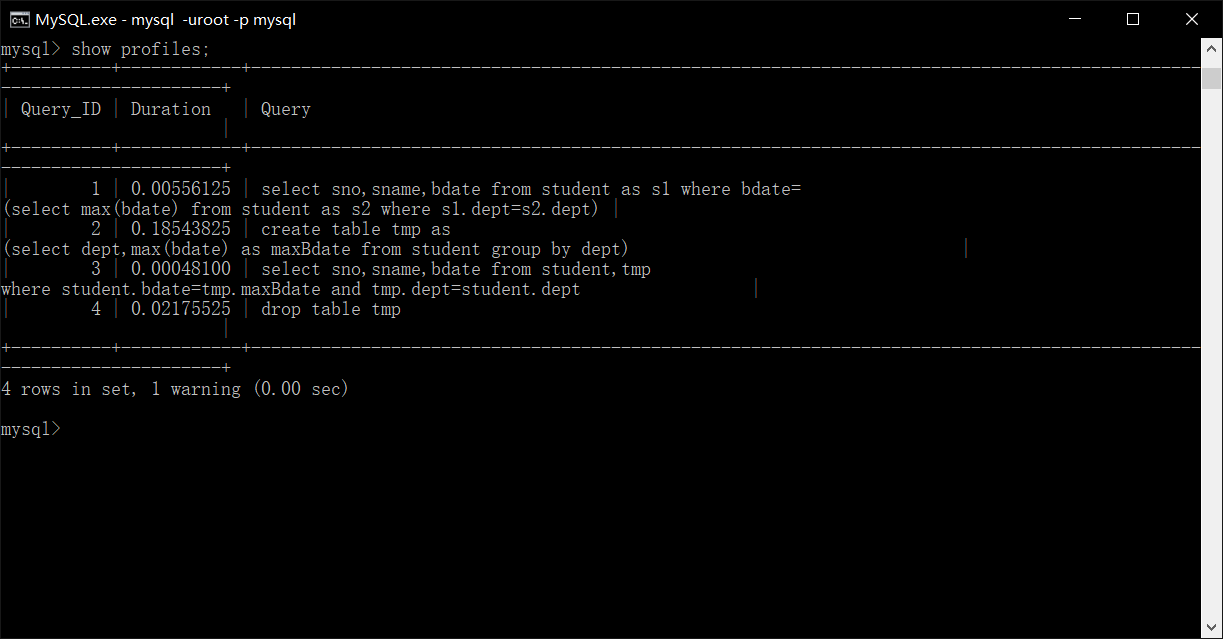
where student.bdate=tmp.maxBdate and tmp.dept=student.dept

drop table tmp

运行结果相同，但是时间相差极大，如下图所示：







第一种方法只包含一条语句，执行时间为0.00556125s；

第二种方法包含三条语句，执行时间分别为0.18543825s、0.00048100s和0.02175525s。其中，单

纯的查询时间为0.00048100s。

重写后，单纯的查询会比之前的用时少很多，甚至相差一个数量级。但是，重写后的查询需要建表和

删除表。如果没有对这个新表进行多次相关查询，那么额外的操作时间会使得这个优化得不偿失。

|  |  |
| --- | --- |
| 查询语句 | 时间 |
| 直接查询 | 0.00556125 |
| 建表查询 | 0.18543825（建表） |
| 0.00048100（查询） |
| 0.02175525（删表） |

* 1. 对下面两个查询进行比较

1）select sname,bdate

from student

where dept!=’计算机’ and bdate>all

(select bdate

from student

where dept=’计算机’

)

2）select sname,bdate

from student

where dept!=’计算机’ and bdate>

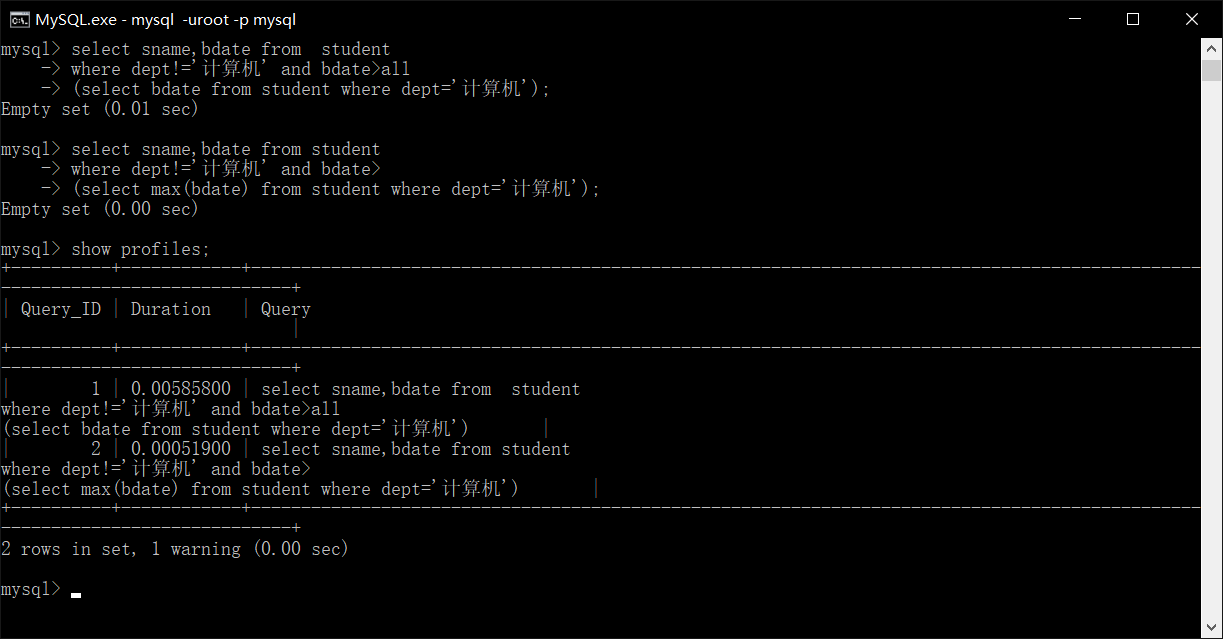
(select max(bdate)

from student

where dept=’计算机’

)；

运行结果相同，如下图所示：



|  |  |  |
| --- | --- | --- |
| 查询语句 | 时间 | 比较 |
| 直接比较 | 0.00585800 | 长 |
| 选择比较 | 0.00051900 | 短 |

* 1. 查找选修了每一门课的学生

1）直接查询

select sname

from student

where sno in(

select sno

from sc

group by sno

having count(\*)=(

select count(\*)

from course))

2）创建table

create table tmp as(

select distinct sno as sno1,count(cno) as amount

from sc

group by sno);

select sname

from tmp,student

where tmp.amount>=5 and tmp.sno1=student.sno

drop table tmp

3）创建view

create view tmp as(

select distinct sno as sno1,count(cno) as amount

from sc

group by sno);

select sname

from tmp,student

where tmp.amount>=5 and tmp.sno1=student.sno;

drop view tmp

4）嵌套子查询

select sname

from student

where not exists(

select \*

from course

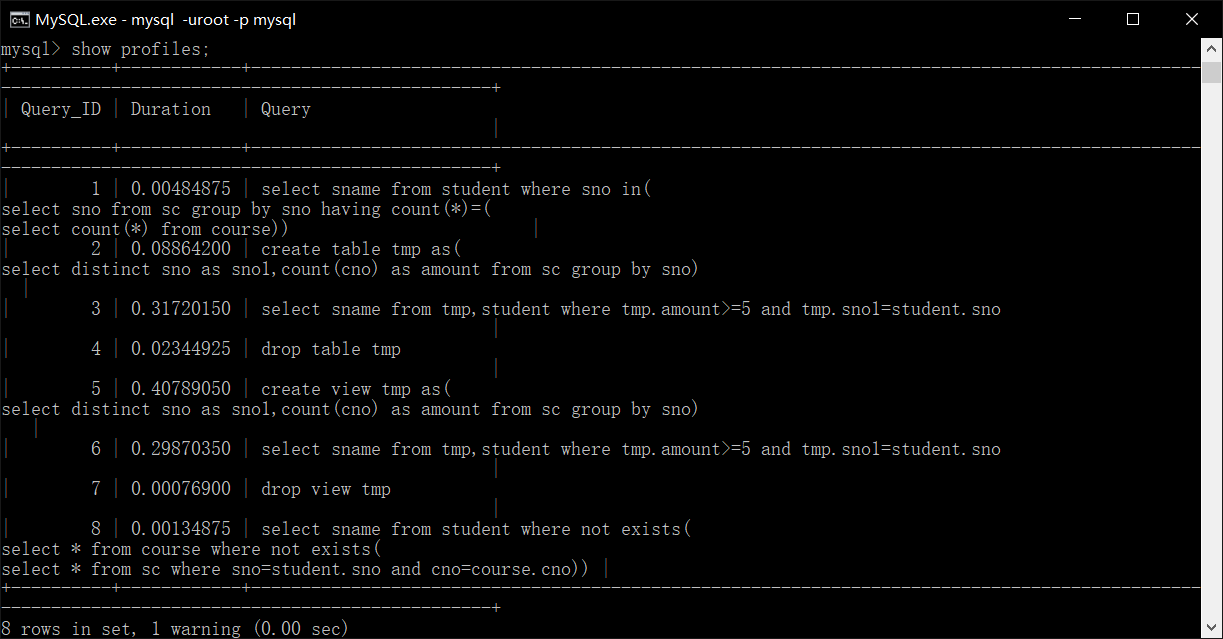
where not exits(

select \*

from sc

where student.sno=course.cno))

运行结果相同，时间如下：



|  |  |  |
| --- | --- | --- |
| 方法 | 时间 | 比较 |
| 直接查询 | 0.00484875 | 较短 |
| 创建table | 0.31720150 | 最长 |
| 创建view | 0.29870350 | 较长 |
| 嵌套子查询 | 0.00134875 | 最短 |

* 1. 查找至少选修了数据库原理和操作系统的学生的学号

1）创建table

create table tmp as(

select sc.sno as sno,cname

from sc,course

where sc.cno=course.cno and (course.cname=’数据库原理’or course.cname=’操作系统’));

select sno

from tmp

group by sno

having count(\*)>=2;

drop table tmp;

2）创建view

create view tmp as(

select sc.sno as sno,cname

from sc,course

where sc.cno=course.cno and (course.cname=’数据库原理’or

1. 实验心得

在本次实验中，遇到的主要问题有以下四点：

* 1. 时间精度。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. 执行时间与理论不符。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。