**门诊医疗预约系统**

优化报告

许正霖-518030910163

组长：周义天-518030910237

杨宇晗-518030910208

焦明胜-518030910249

曹韫琪-518030910242

**目录**

1、业务背景简述 3

2、优化目标描述 3

3、性能分析与测试过程、结果 4

3.1测试过程 4

3.2测试结果 4

3.3性能分析 6

4、优化方法描述及优化后性能 6

4.1优化方法 6

4.1.1分表 6

4.1.2添加索引 7

4.1.3子查询优化 7

4.1.4对sql语句进行优化 9

4.2优化后性能 9

5、其他发现 11

5.1缓存 11

5.2数据库引擎 13

1. **业务背景简述**

建立医院门诊预约系统，对提高工作效率，减少工作量以及解放劳动力具有重要意义。医院作为关系民生的重要机构，每天需要接待的人员基数巨大，尤其是现在人们健康意识的提高，因此，我们需要一个好的设计来构造医院的医疗门诊预约系统。本次实验在原来构建的项目系统基础上做出了优化，使得性能较原先有些许的提升。

1. **优化目标描述**

本次优化目标如下：

1. 使得长时间的查询过程能够缩短时间，希望对于长时间的查询能够尽量减少其查询时间。
2. 高并发的情况下减少查询时间，增加系统健壮度，保证系统的稳定性正确性。
3. 希望能够尽量降低数据库中表的复杂度，让对数据库操作的人员能够简便上手便于管理。
4. **性能分析与测试过程、结果**
   1. **测试过程**

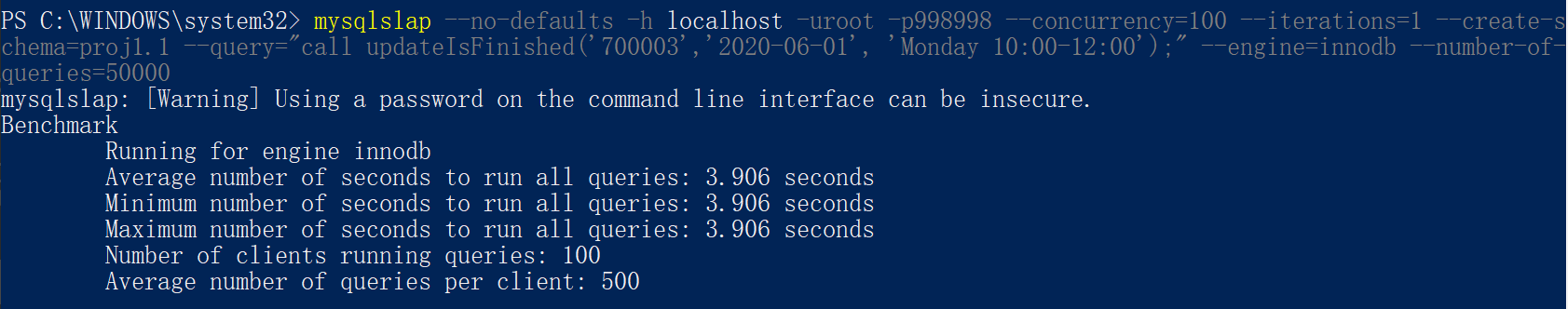
测试过程首先是使用sql的语句产生了大部分的数据量，其中大部分数据量主要是属于appointment patient部分（考虑到一个现行的医疗数据系统最有可能出现大数据量的地方便是预约和病人部分），其他处则采用了规定个数，类型的方式产生了一定的数据（比如规定科室有哪几种，有哪些门诊，产生了一百条随机的医生信息之类）。

其次采用mysqlslap增大了并行量进行查询，对之前编写过的函数或是简单的一条查询语句进行查询时间分析，利用explain分析一个查询子句进行时遍历了多少元组，是否使用了索引等操作。

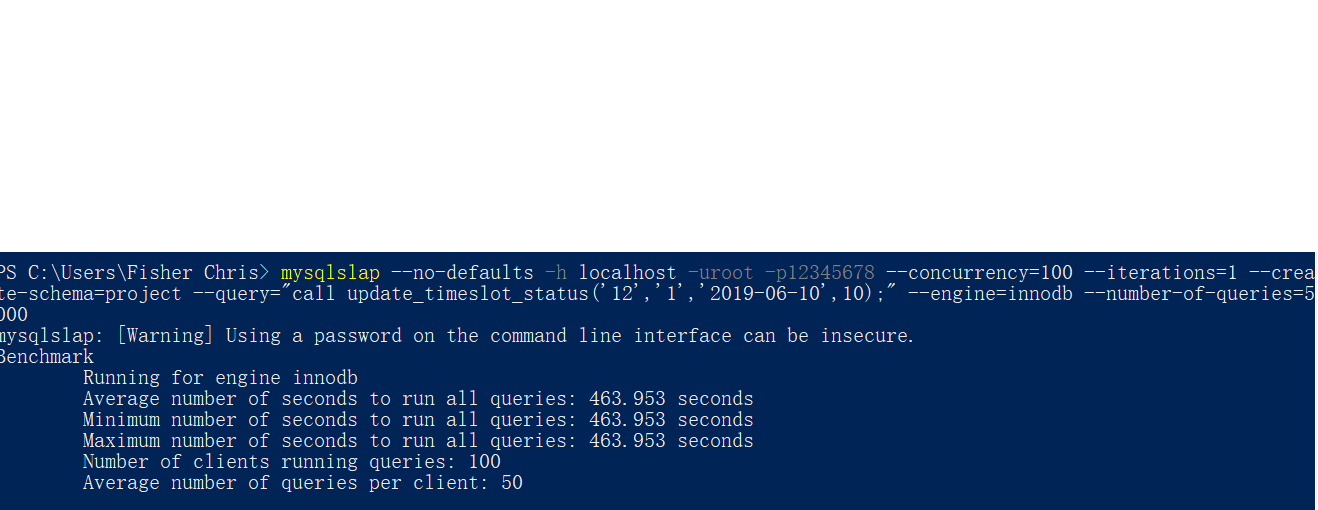
* 1. **测试结果**

以下展示了几个原先未优化版本的mysqlslap分析结果：

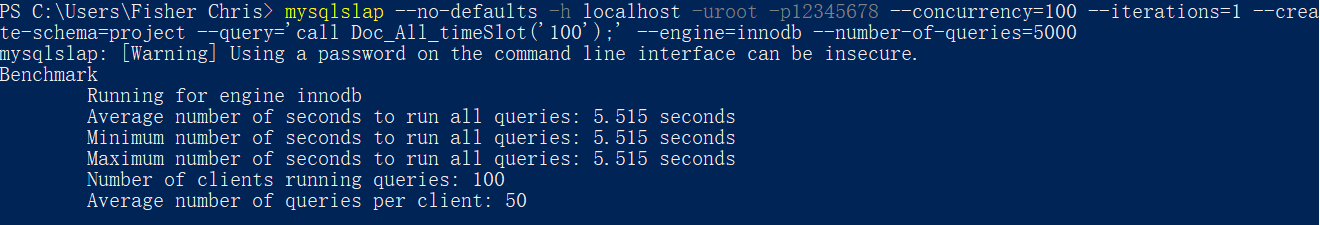
1、将已完成的预约条目更新



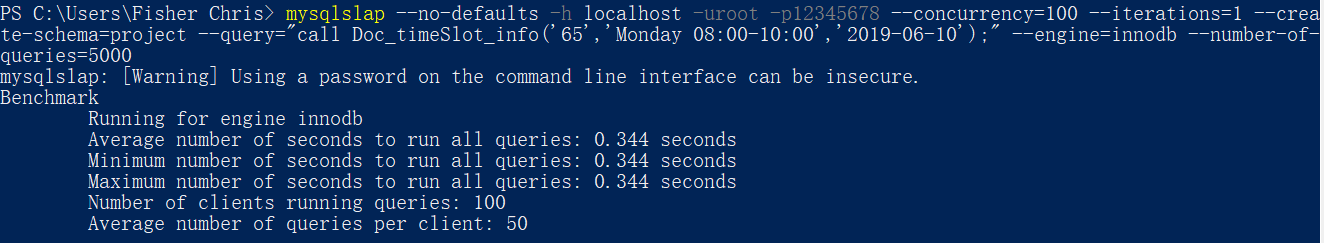
1. 更新某一天time\_slot状态



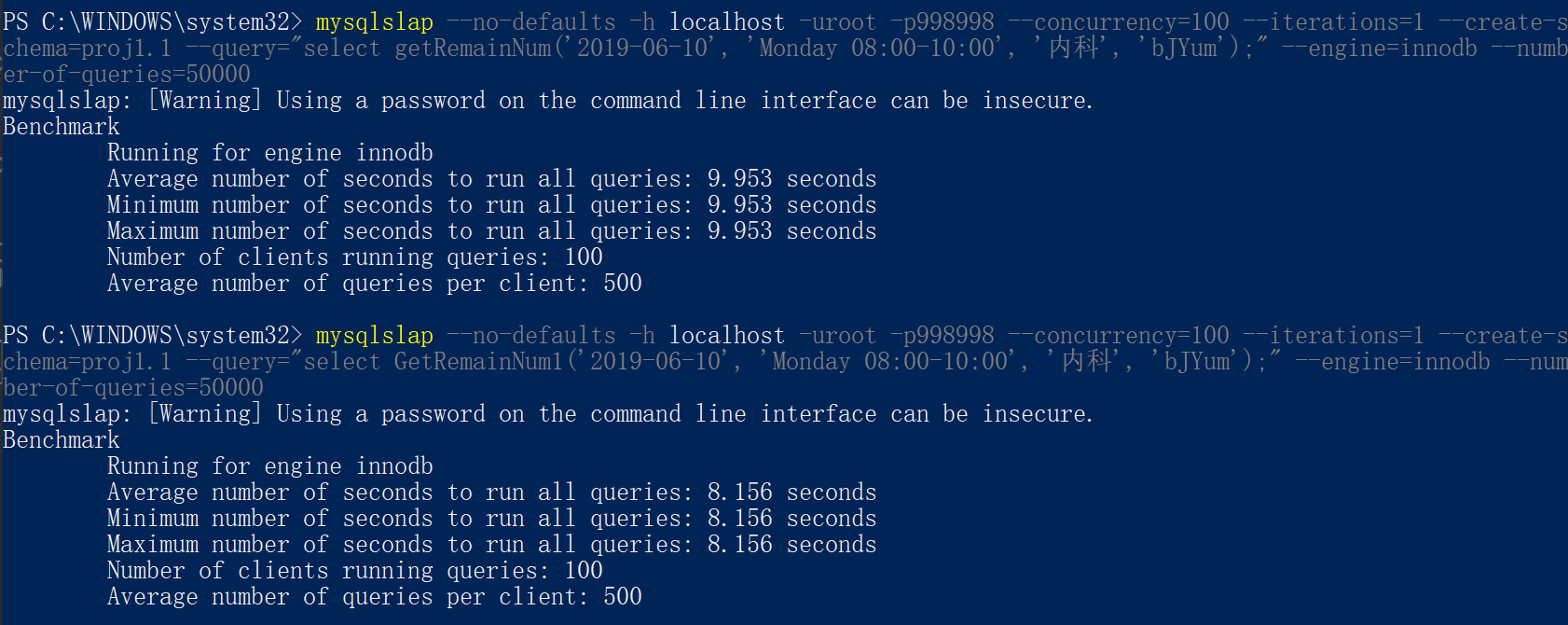
1. 查询一个医生的工作的所有时段



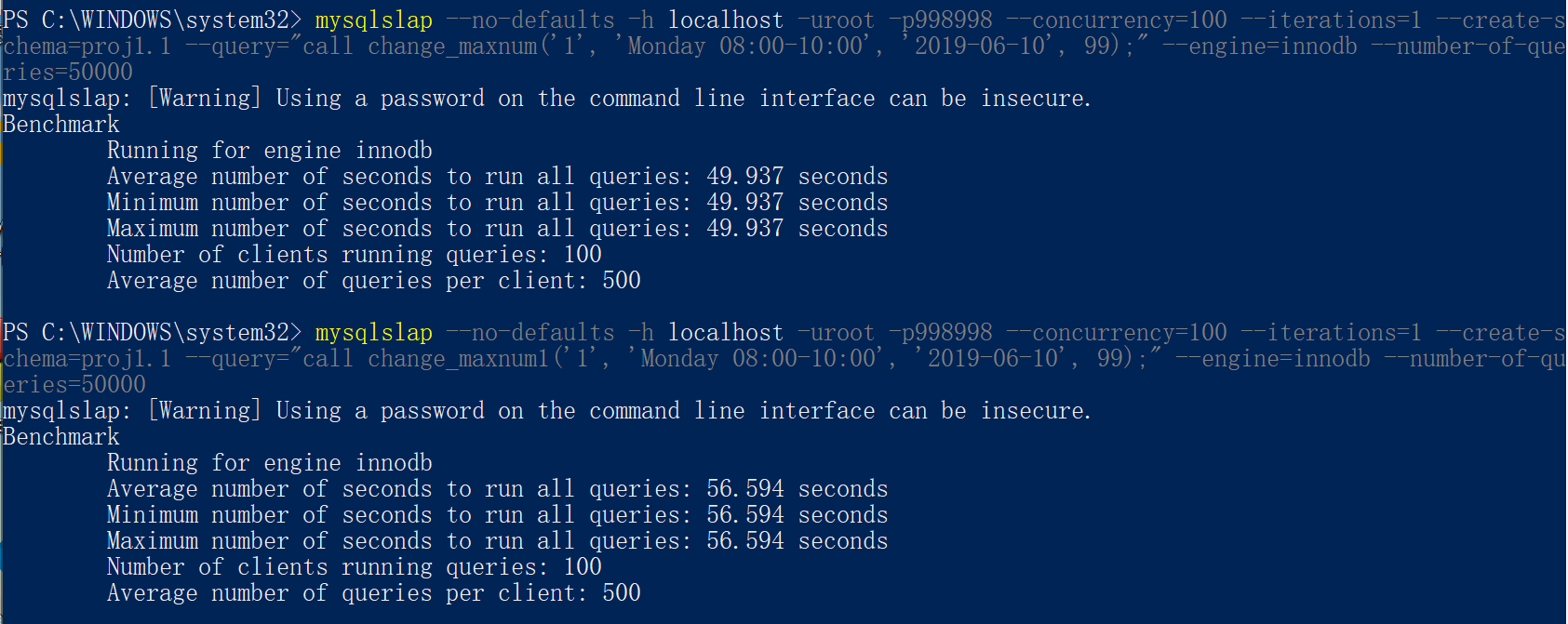
1. 根据某一条件查询一个医生工作时段的信息



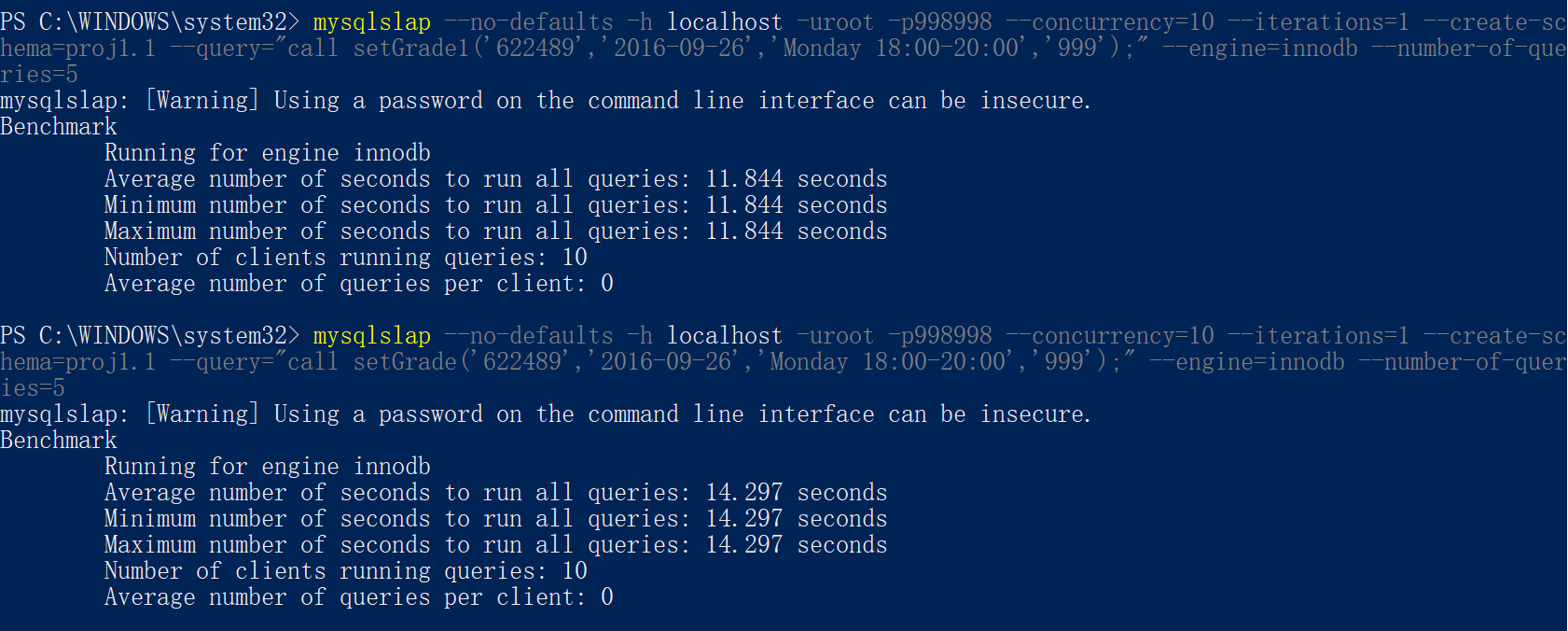
5、根据某一时段查询该时段预约人数信息



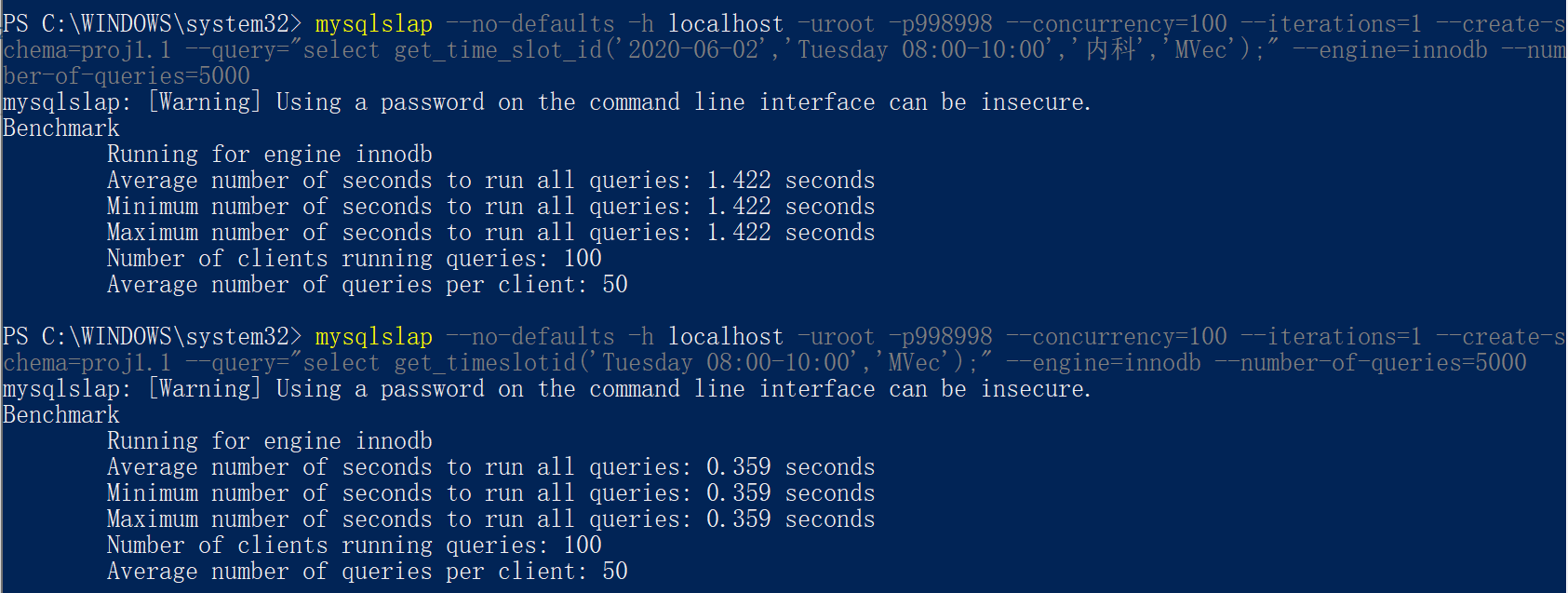
6、改变某一时段的最大人数



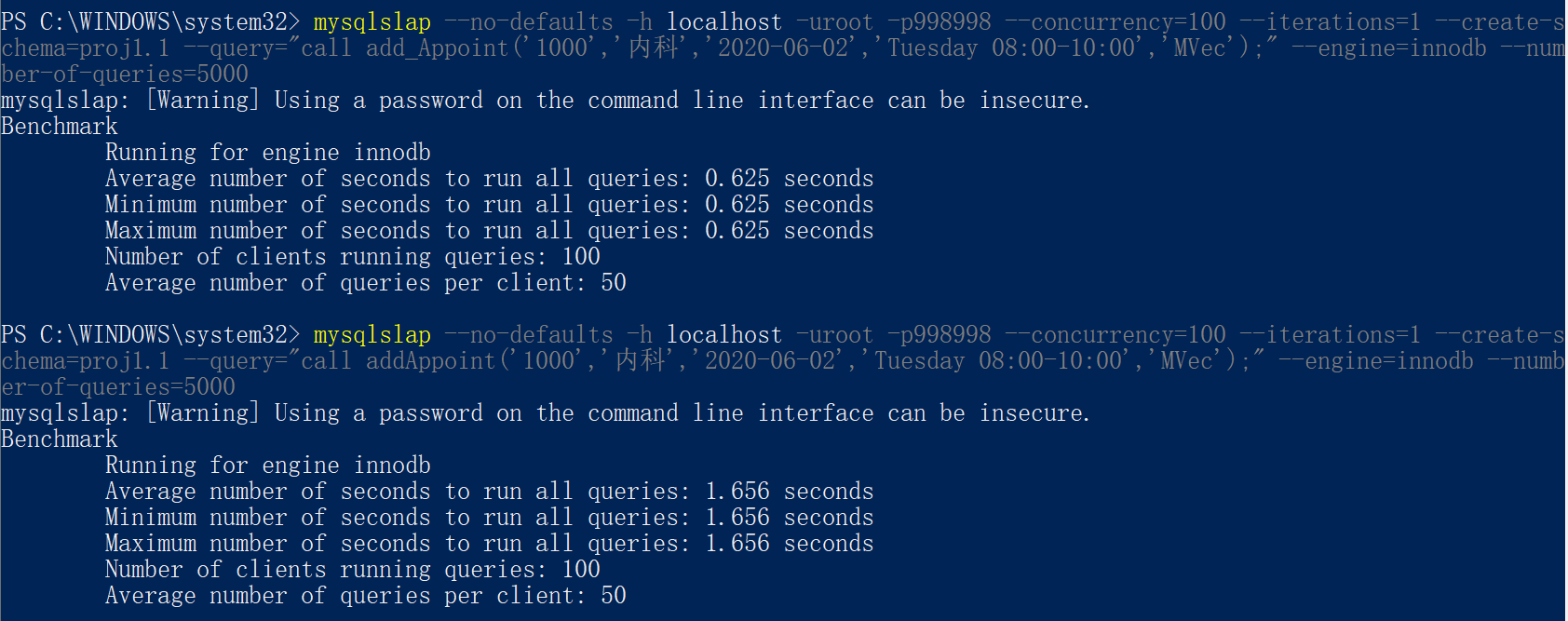
7、将已完成的appoint设置成绩



8、得到time\_slot\_id



9、增加一条预约



* 1. **性能分析**

从测试结果看出，查询的时间代价基本都是较大的，很多处查询都超过了至少5s，这对于一个能够正常工作的系统来说的确是令人遗憾的。其中时间过长的地方基本集中在关于appointment的操作上（appointment中的数据为一百万条）。

1. **优化方法描述和优化后性能**
   1. **优化方法**

**4.1.1分表**

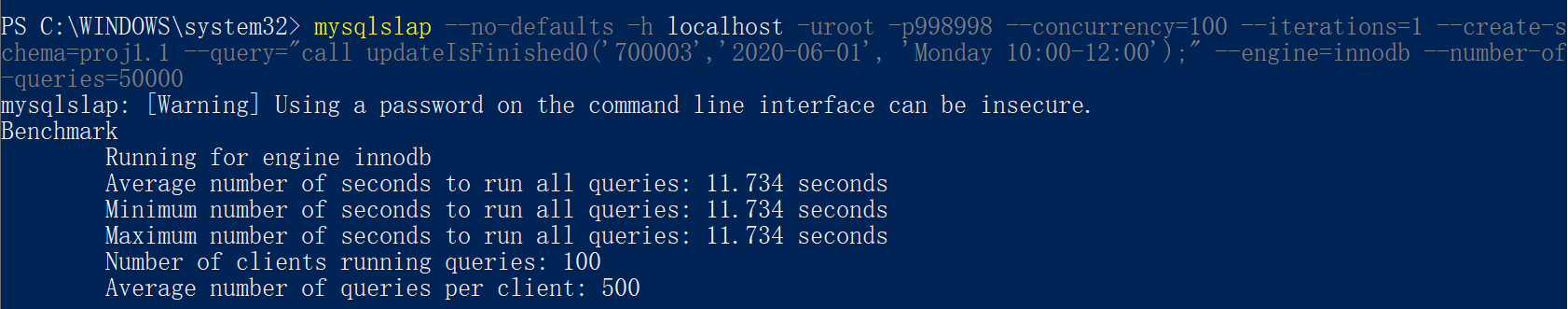
首先想到的优化方法就是分表。

当一张的数据达到几百万时，查询一次所花的时间会变多，如果有联合查询的话，会花费更多时间。

分表的目的就在于此，减小数据库的负担，缩短查询时间。尤其是对一个表中适用区域明显不同的两部分数据。

在我们的项目中，最明显的可用于分表的表为appointment：可以将未完成的预约和已完成的预约分成两个表，这样查询相关预约时只需要从某一个表中查询即可。

分表后新修改的更新预约状态的函数性能：



可以看出性能比原来差了许多，这是因为在原先表的基础上只需要将一个名为isfinished从0改为1即代表appointment完成，但是在新表中需要将未完成预约表中的一项删除，再将这项的内容插入已完成预约表中成为新的一项。操作的繁琐带来的自然也是性能的下降。但一旦更新完成，appointment的查询时间会大大降低。

在助教学长的帮助下，我们把更新预约状态函数从即时调整数据所在表，改为每隔一周把未完成表中已完成的数据移入已完成表中。由于病人需要在完成的订单状态被更新至appointment\_finished表之前完成评分，所以必须限制病人在就诊完成后的一周内完成评分。

**4.1.2添加索引**

添加索引之后，DB在根据query进行查询的过程中可以根据现有的一些index达到快速查询的效果。

**4.1.3子查询优化**

子查询可以一次性完成逻辑上需要多个步骤才能完成的SQL操作，但是在两种情况下性能可能较差：

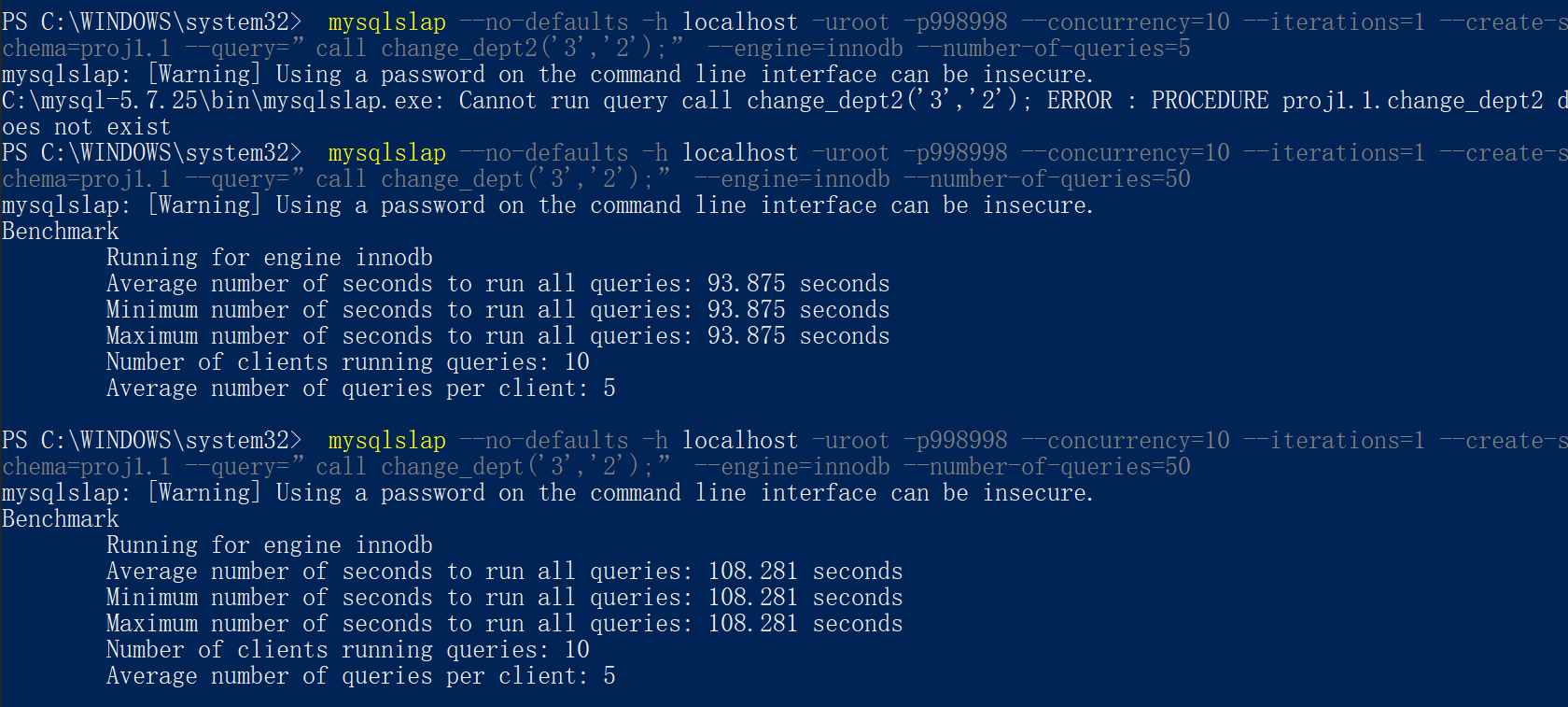
1、select count(\*)语句后接子查询

2、In和exist后接子查询

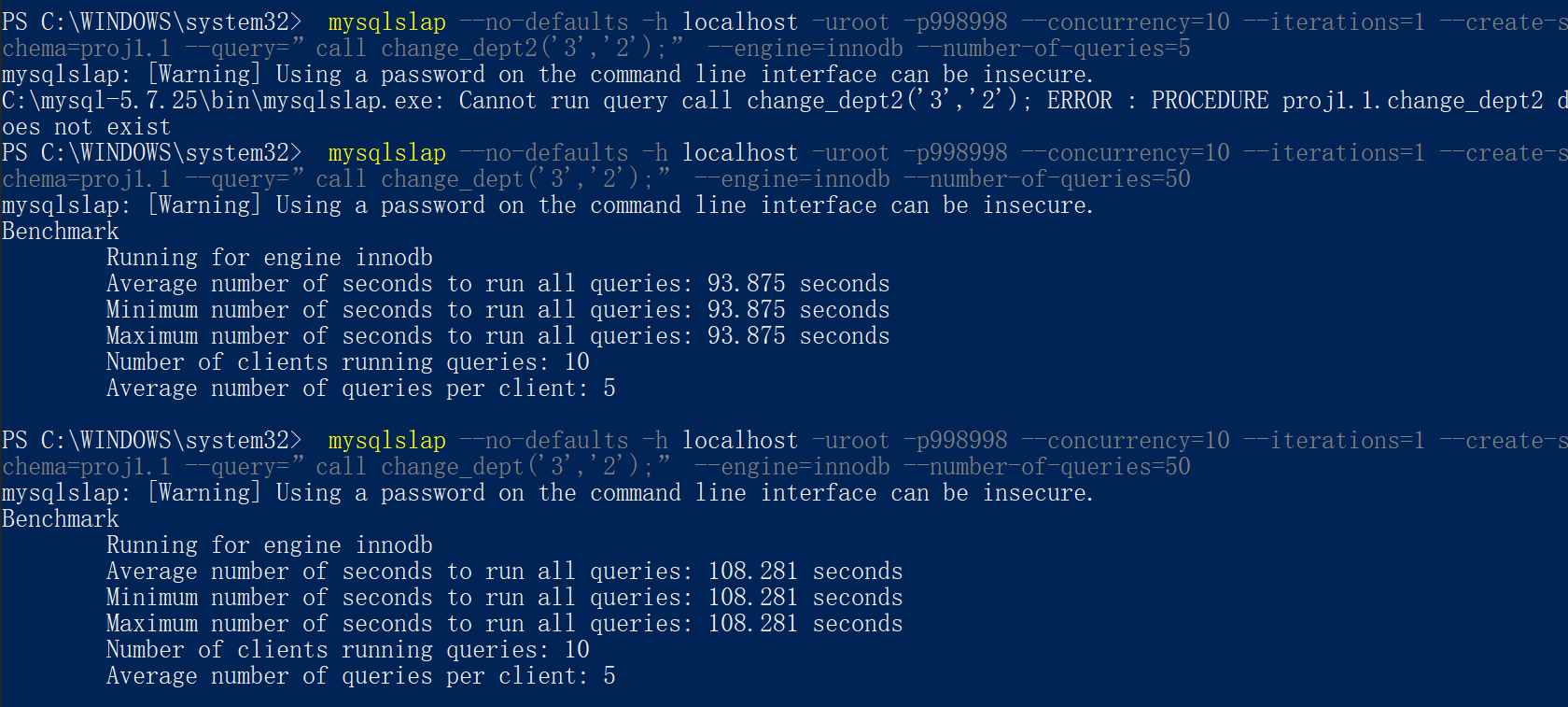
此次优化过程中我们主要对In和exist中的子查询进行了优化。

利用之前的一个业务SQL进行实验：改变科室

使用in方法所需时间：94s



使用exists方法所需时间：108s



实验表明：

形如 SELECT   \*  FROM A WHERE id IN (SELECT id FROM B);in会将B的所有查询做完后进行缓存，再将每次A的查询结果与缓存的结果对比

而形如SELECT \* FROM A WHERE EXISTS(SELECT 1 FROM B WHERE B.id  = A.id) 则会先进行A的查询，在再将查询结果带入B查询中进行判断

如果内表数据量大而外表数据量小，那么使用exists的子查询就更容易得到true的返回值，效果相对较好；如果内表数据量小而外表数据量大，那么使用in查询只对内表查询一次，可以避免反复读取内表中数据影响性能。

**4.1.4对sql语句进行优化**

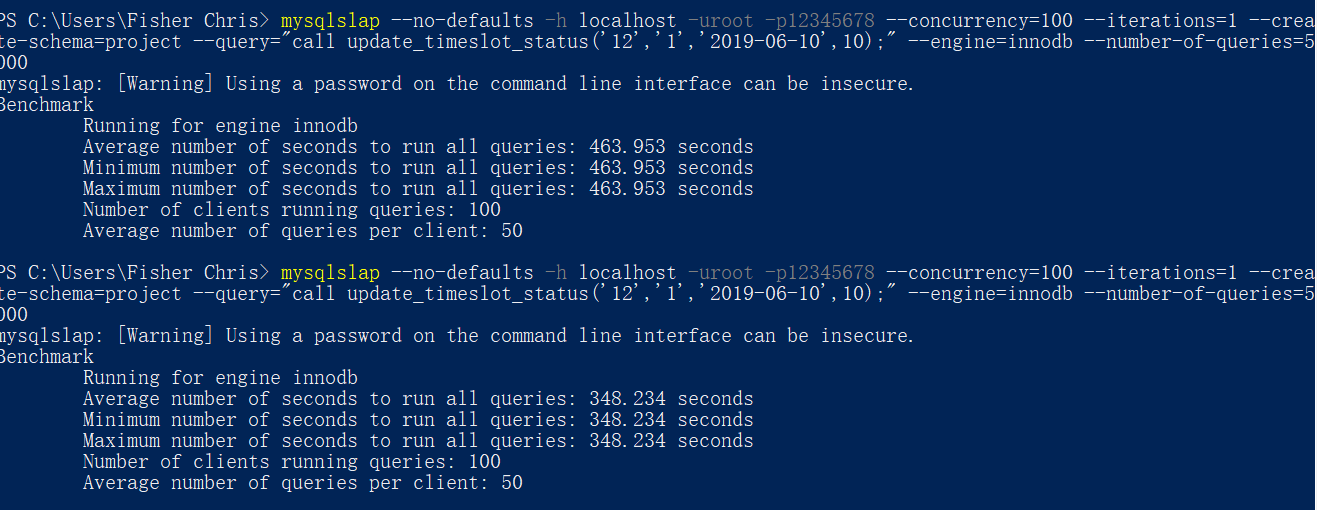
这一过程主要体现在优化先前写过的功能函数，过程和触发器上，通过将where子句提前和利用前三种方法，我们最终得到了优化后的业务SQL版本。

* 1. **优化后性能**

以下展示了优化后版本的mysqlslap分析结果：

1. 利用In和exists思想进行优化的函数

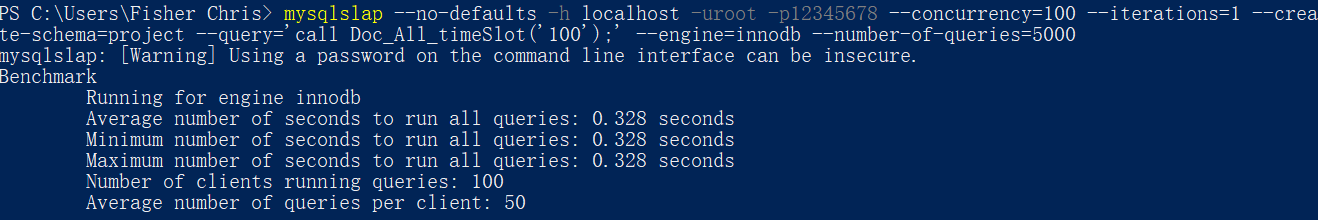
更改某一天time\_slot状态



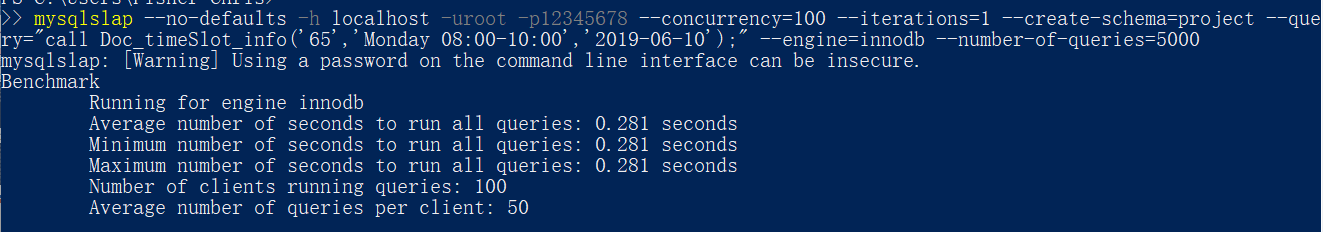
对这一过程的优化主要体现的是In和exist的思想，优化后的版本选择了exists，适合于外表小内表大的情况。

2、把where提前到子查询思想进行优化的函数

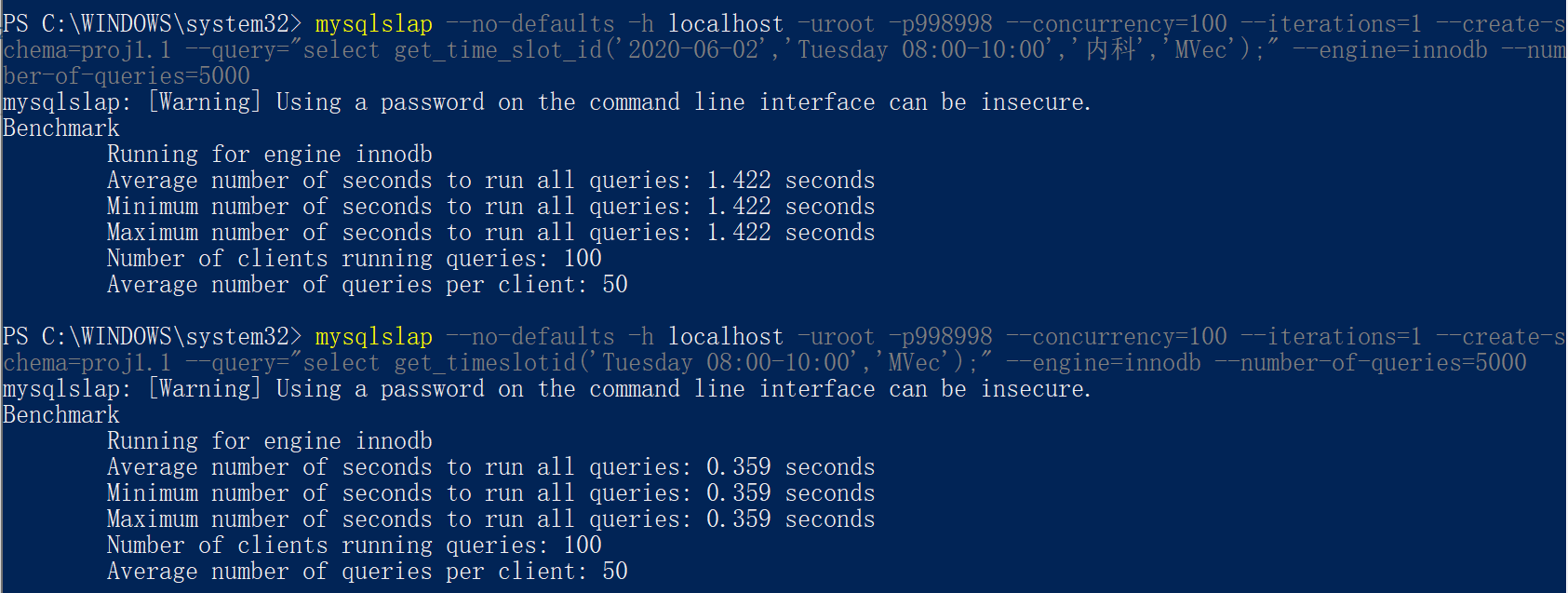
查询一个医生的工作的所有时段



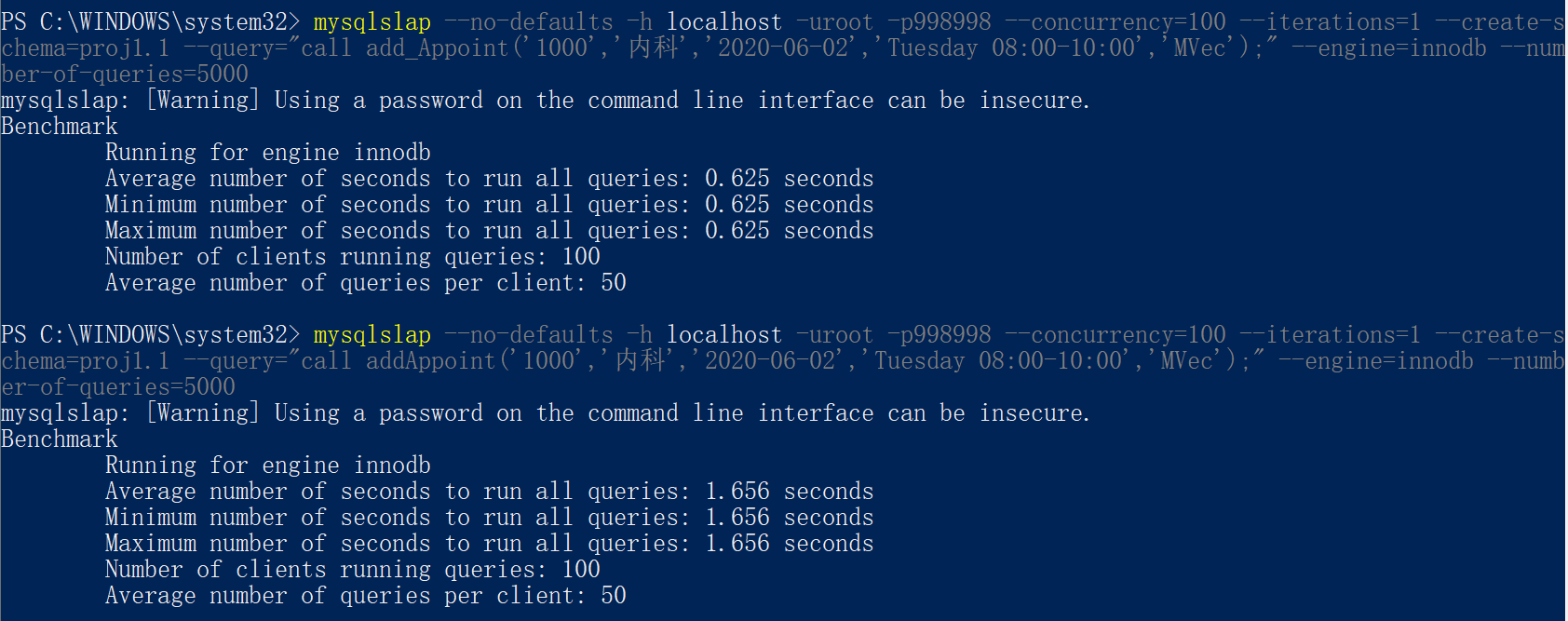
根据某一条件查询一个医生工作时段的信息



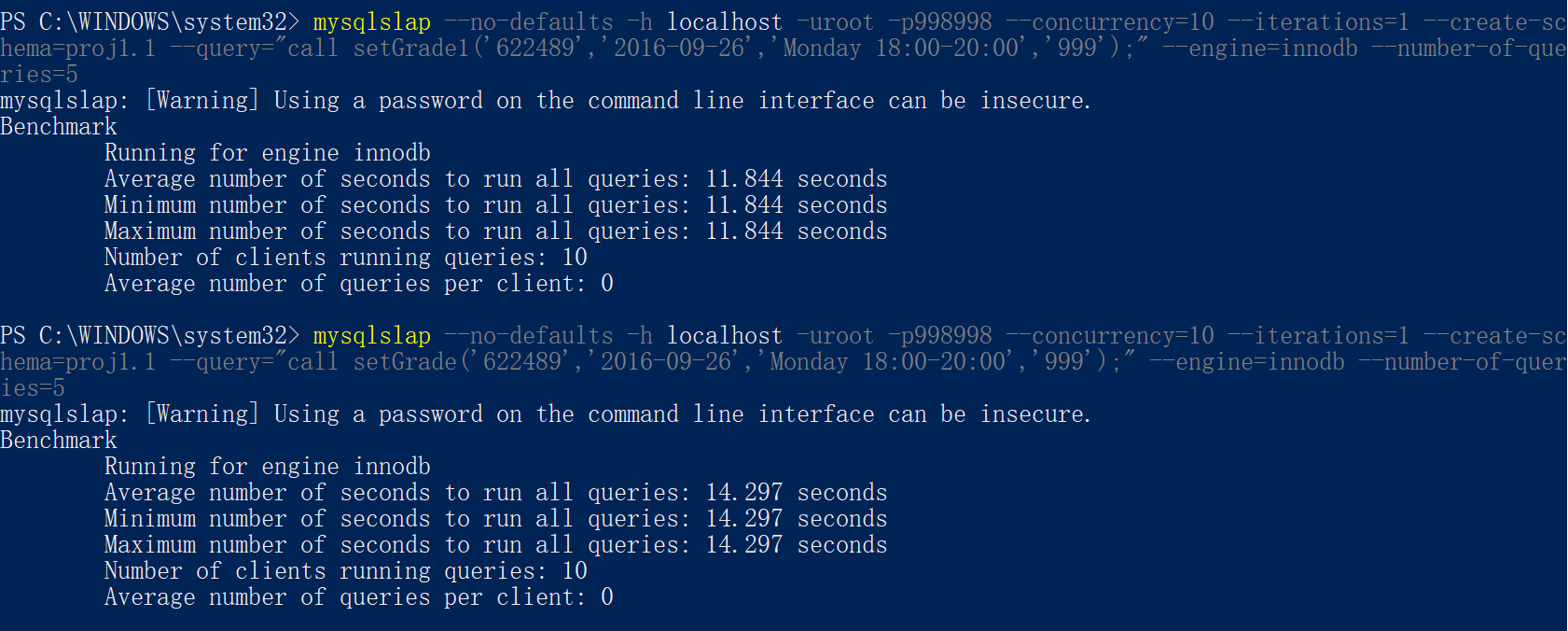
得到time\_slot\_id



调用了get\_timeslotid的增加一条appoint的业务sql。



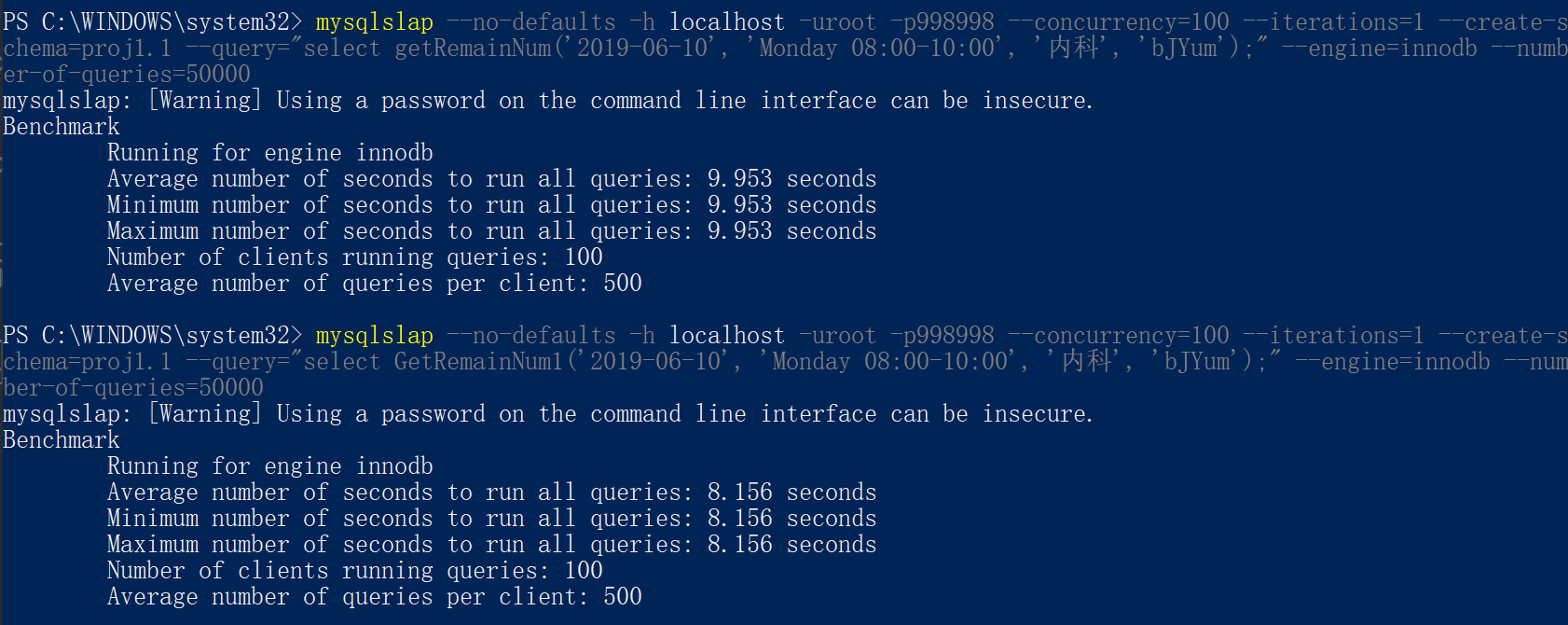
调用setGrade在病人完成就诊后给医生评分



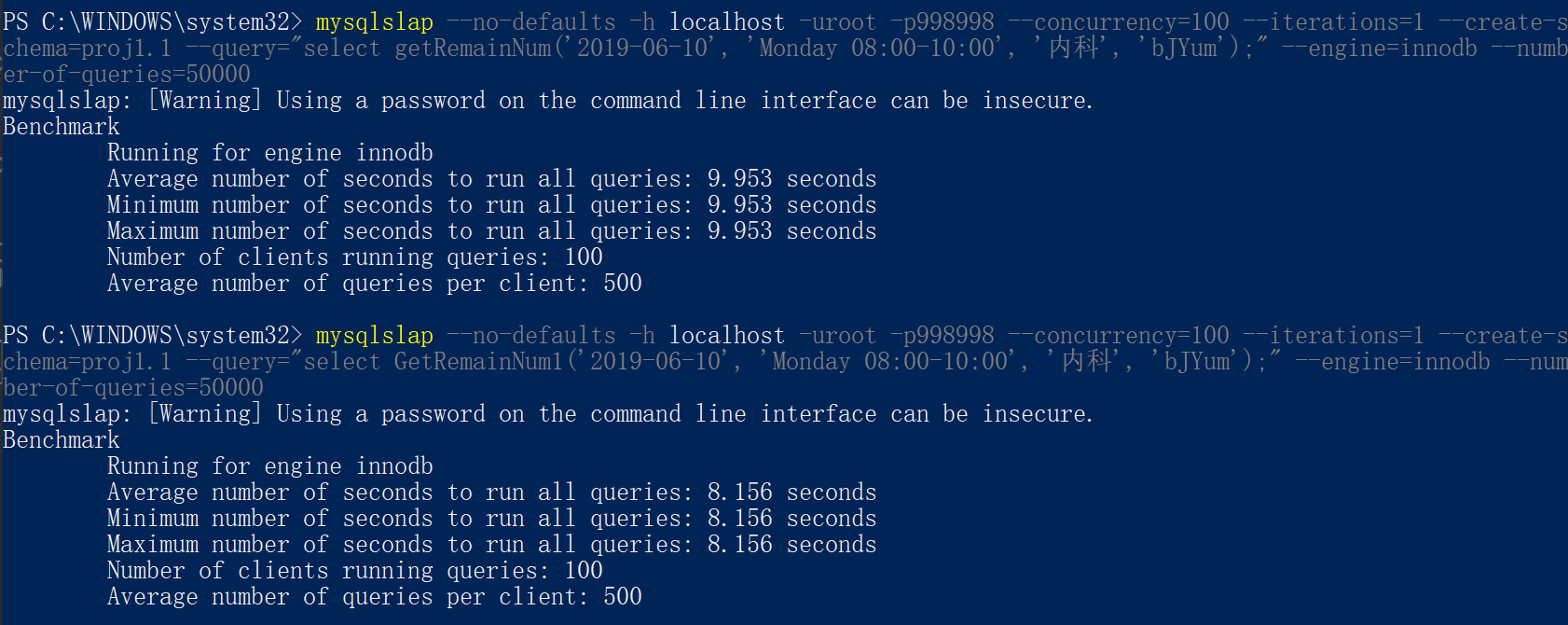
对这一过程的优化主要体现的是利用把where提前到子查询中减少natural join之前表的行数，进而减少了检索的次数。

3、用子查询的嵌套代替多次natural join

原先函数



优化后函数



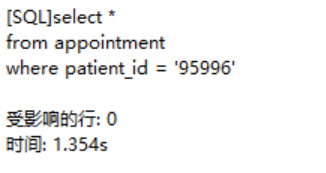
这一优化思想为：由于参与natural join的表数量太多，每次natural join时都很多列都在同时重复的被使用，同时，每进行一次natural join表的规模都会扩大，可能会造成性能上的问题，故考虑用子查询的嵌套取代natural join，在具体的尝试中发现，在某些时候使用嵌套查询代替多个表natural join可以一定程度上的提升效率，但这可能与表的规模与数据表的列之间的关系相关。同时使用嵌套查询的sql语句可读性不如使用natural join

1. **其他发现**
   1. **缓存**

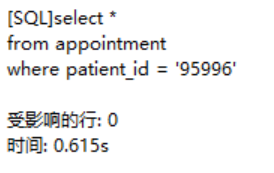
为什么第一次查询比第二次慢很多？

在我们进行实验的过程中，我们发现对于一条select语句，第一次查询要比第二次查询慢得多：

第一次查询

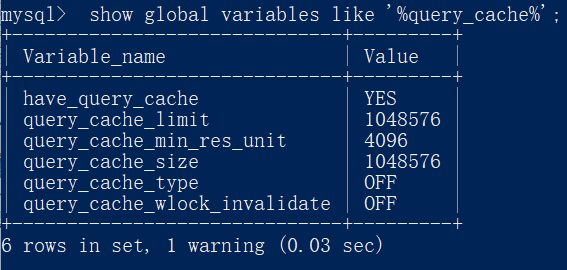


第二次查询



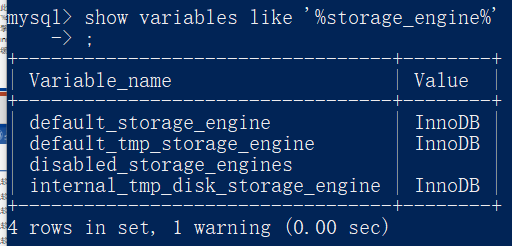
这是为什么呢？联系我们在别的课学到的知识，我们猜想是不是第一次查询是从磁盘读取数据而此后第二次是从类似的cache或者别的缓存区读取数据才会带来这么大的差距.

打开mysql,输入show global variables like '%query\_cache%’;



我们发现have\_query\_cache这一行确实是YES，证明cache确实是存在的，印证了我们之前的猜测。

此外，输入show variables like ‘%storage\_engine%’;，发现表的引擎是innodb，查询资料得知innodb有buffer\_pool，也起到了缓冲区的作用.



对于一次查询，只要经过之后，就会将其缓冲在缓冲区中，第二次只需要cache hit就可以从缓冲区中取出内容，而非再进行一次IO读取，因此读取的时间大大减少。

* 1. **数据库引擎**

我们对MyISAM和InnoDB两个数据库引擎进行了调查和实验比较，得出以下结果。

MyISAM基于ISAM存储引擎，并对其进行扩展。它是在Web、数据仓储和其他应用环境下最常使用的存储引擎之一。MyISAM拥有较高的插入、查询速度，但不支持事物。主要特性有：

1、myisam只支持表级锁，用户在操作myisam表时，select，update，delete，insert语句都会给表自动加锁，如果加锁以后的表满足insert并发的情况下，可以在表的尾部插入新的数据。也可以通过lock table命令来锁表，这样操作主要是可以模仿事务，但是消耗非常大，一般只在实验演示中使用。

2、myisam属于堆表

3、当把删除和更新及插入操作混合使用的时候，动态尺寸的行产生更少碎片。这要通过合并相邻被删除的块，以及若下一个块被删除，就扩展到下一块自动完成

InnoDB是事务型数据库的首选引擎，支持事务安全表，支持行锁定和外键。主要特性有：

1、Innodb支持事务和行级锁，InnoDB给MySQL提供了具有提交、回滚和崩溃恢复能力的事物安全（ACID兼容）存储引擎。

2、Innodb属于索引组织表。如果使用共享表空间，那么所有表的数据文件和索引文件都保存在一个表空间里，一个表空间可以有多个文件，通过innodb\_data\_file\_path和innodb\_data\_home\_dir参数设置共享表空间的位置和名字。

3、InnoDB是为处理巨[大数据](http://lib.csdn.net/base/hadoop" \t "_blank)量的最大性能设计。它的CPU效率可能是任何其他基于磁盘的关系型数据库引擎锁不能匹敌的

4、InnoDB存储引擎完全与MySQL服务器整合，InnoDB存储引擎为在主内存中缓存数据和索引而维持它自己的缓冲池。InnoDB将它的表和索引在一个逻辑表空间中，表空间可以包含数个文件（或原始磁盘文件）。这与MyISAM表不同，比如在MyISAM表中每个表被存放在分离的文件中。InnoDB表可以是任何尺寸，即使在文件尺寸被限制为2GB的[操作系统](http://lib.csdn.net/base/operatingsystem" \t "_blank)上

5、InnoDB支持外键完整性约束，存储表中的数据时，每张表的存储都按主键顺序存放，如果没有显示在表定义时指定主键，InnoDB会为每一行生成一个6字节的ROWID，并以此作为主键

据此，我们得出结论：由于MyISAM不是事务安全的，而且不支持外键，如果执行大量的select，insert操作MyISAM比较适合；而InnoDB是支持事务安全的引擎，支持外键、行锁、事务是他的最大特点。如果有大量的update和insert操作，建议使用InnoDB，特别是针对多个并发和QPS较高的情况。