北京邮电大学

实验报告

课程名称：数据库系统原理

实验名称：事务及其并发控制

计算机系2015211312班 姓名刘佳鑫

计算机系2015211312班 姓名万诗婕

教师吴起凡 成绩\_\_\_\_\_\_\_\_\_

2018年6月7日

### 实验目的

通过单事务、串行事务、并发事务实验，了解SQL SERVER数据库系统中

1）事务组成方式和执行模式；

2）对单事务和串行事务的原子性保障机制；

3）基于锁和隔离级别的事务并发控制和对并发事务的一致性、独立性保障机制。

### 实验环境

采用SQL SERVER数据库管理系统作为实验平台。其中，SQL SERVER 采用2012版本。

### 实验内容

1、单事务/串行事务提交与回滚

2、事务并发控制机制

### 实验步骤

### 一、单事务/串行事务提交与回滚

### 1. 违反check约束的update操作

### 实验要求：

根据现网实际情况，小区/基站工参表tbCell中的小区天线高度不能小于0。 在tbCell的备份表tbCellnew上，用Alter table add check添加约束，并在该备份表上完成以下实验内容：

Step1. 查询小区/基站工参表的小区天线高度（HEIGHT）小于20的SECTOR\_ID、SECTOR\_NAME和HEIGHT；

Step2. 更新小区/基站工参表将step1中的HEIGHT设置为当前值减去15（注意此时有可能违反check约束）

Step3. 查询小区/基站工参表的小区天线高度（HEIGHT）小于20的SECTOR\_ID、SECTOR\_NAME和HEIGHT；

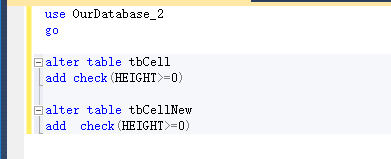
针对以上操作分别进行如下的操作：

（1）将以上操作组织成普通的SQL语句，顺序执行。

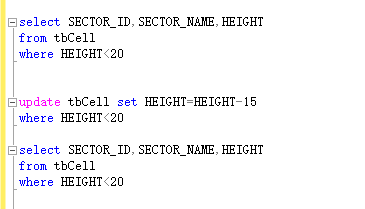
（2）将以上操作组织成事务执行（**以begin tran开始，以commit tran结束**）。

查看数据库，观察两次的执行结果有何异同。

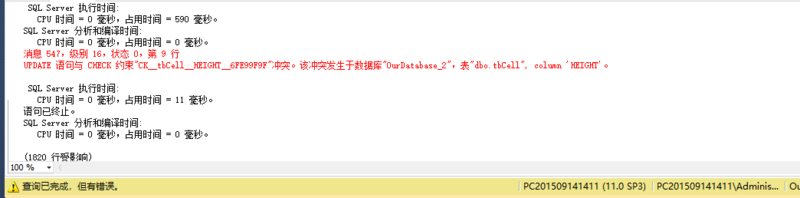
1.为tbCell和tbCellNew加上check约束



2.顺序执行



此时提示违反了check约束

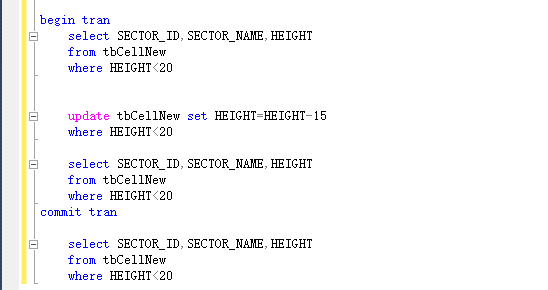


此时的查询结果：

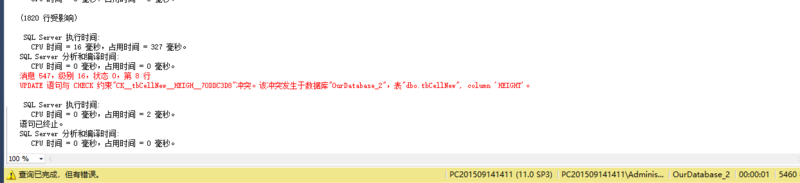


发现更新后和更新前的select结果是一样的，表示没有更新成功。

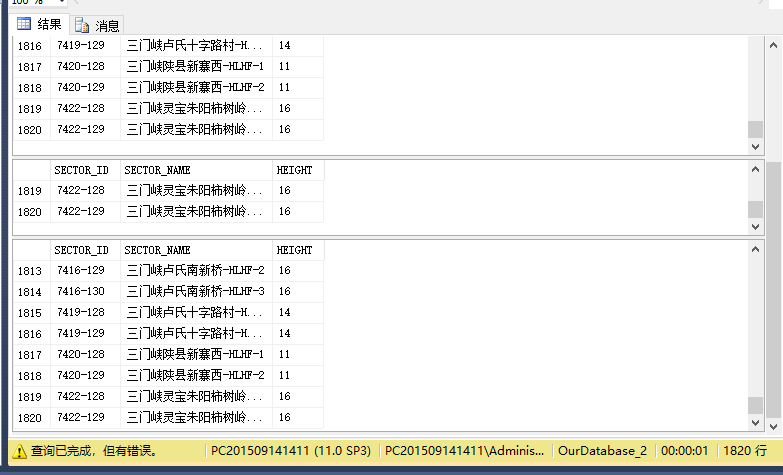
2.使用事务执行



此时发现违反了check约束



查询结果



更新后和更新前是相同的。

在此种情况下使用普通SQL语句执行和事务执行的结果是相同的。

事务的方式要么全部执行，要么全部不执行，所以在此处，更新失败。

**2. 数据表更新**

**实验要求：**

**分别以三种事务执行方式，在tbCell的备份表tbCellnew上执行以下操作，并观察、分析、解释执行结果**

Step1. 查看小区ID在'122880-0'和'122882-2'之间的小区配置的频点编号；

Step2. 将小区ID在'122880-0'和'122882-2''的小区配置的频点编号更新为37900;

Step3. 再次查看小区ID在'122880-0'和'122882-2'之间的小区配置的频点编号。

将step1、step2和step3的数据库访问组织成1个单一事务T1，再将step3作为1个独立事务，提交DBMS，串行执行这2个事务，观察T1中的rollback、commit对事务执行结果的影响。

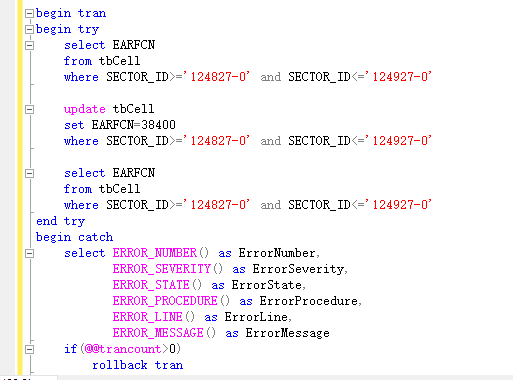
由step1、step2和step3组成的事务T1采用以下2种结束方式：

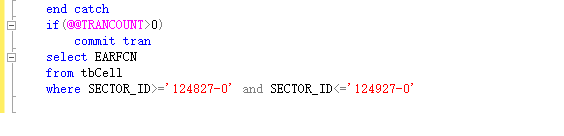
（1）以commit结束；

（2）以rollback结束。

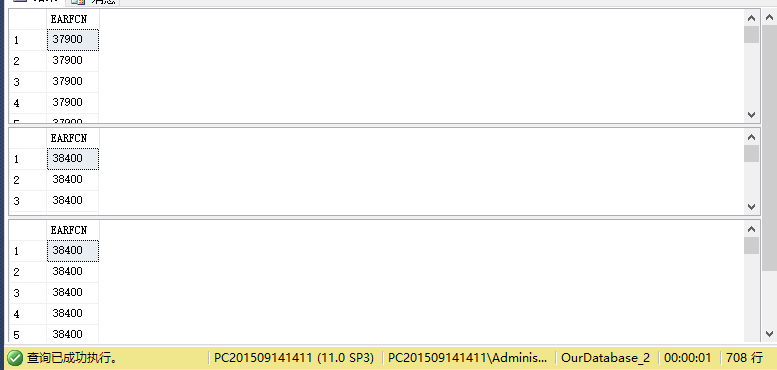
**2.1 显示执行模式：**

**完整过程：**





实验结果：



后两张表被修改了

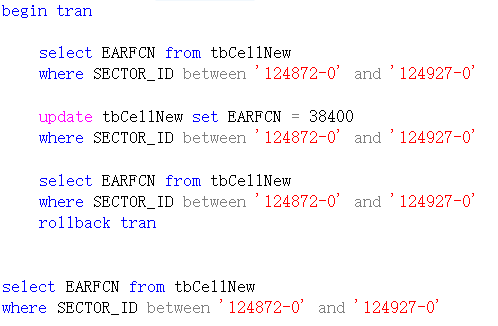
解释说明：

在SqlServer里，嵌套事务的层次是由@@TranCount全局变量反映出来的。

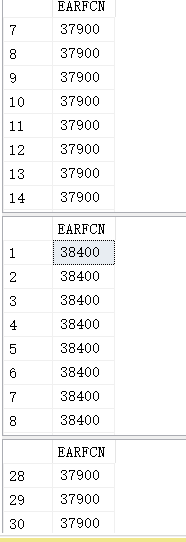
每一次Begin Transaction都会引起@@TranCount加1。而每一次Commit Transaction都会使@@TranCount减1，而RollBack Transaction会回滚所有的嵌套事务包括已经提交的事务和未提交的事务，而使@@TranCount置0。

所以在刚开始begin tran时，trancount+1，变为了1。因为没有出现异常，所以catch中间的部分没有被执行，即rollback部分并不会被执行。而对于commit，因为判断中trancount>0被符合，所以成功提交。

**只执行rollback过程：**



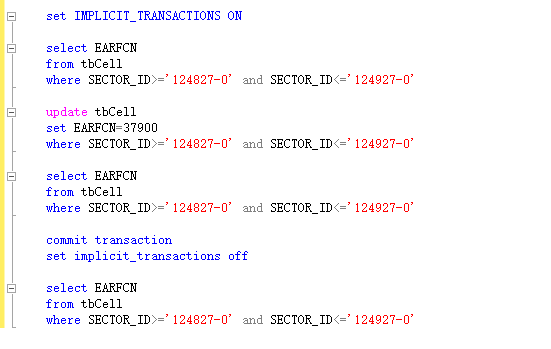
查看执行结果：



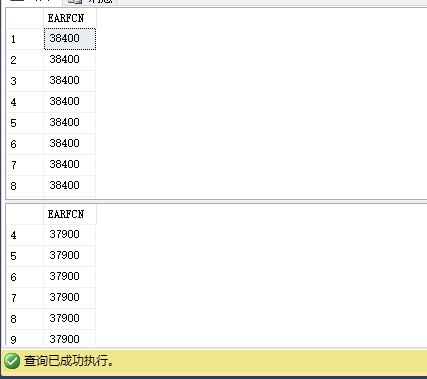
说明：只用rollback结束事务，在事务内部结果改变，但是事务结束后结果没有改变。因为更新部分被rollback了。

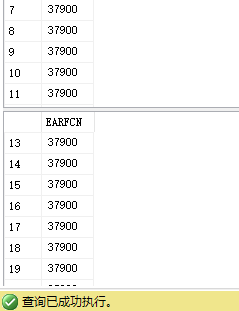
**2.2 隐式事务：**

**Commit提交事务：**



查看执行结果





后两张表成功更新。

如果设置为 ON，SET IMPLICIT\_TRANSACTIONS 将连接设置为隐式事务模式。如果设置为 OFF，则使连接恢复为自动提交事务模式。

事务是单个的工作单元。如果某一事务成功，则在该事务中进行的所有数据修改均会提交，成为数据库中的永久组成部分。如果事务遇到错误且必须取消或回滚，则所有数据库修改均被清除。

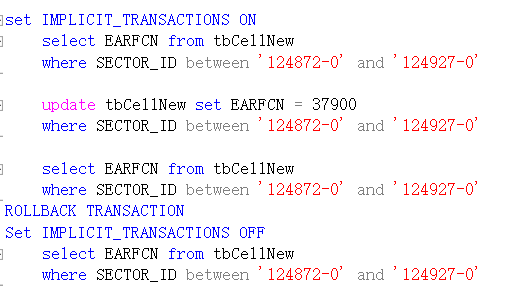
SQL Server中有一下几种事务运行模式。

1. 自动提交事务：每条单独的语句都是一个事务。

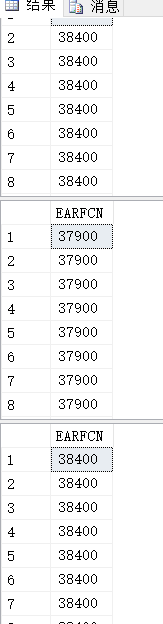
2. 显式事务：每个事务均以BEGIN TRANSACTION语句显式开始，以COMMIT或ROLLBACK语句显式结束。

3. 隐式事务：在前一个事务完成时新事务隐式开始，但每个事务仍已COMMIT或ROLLBACK语句显式完成。隐式事务不用加begin。

Rollback回滚事务：



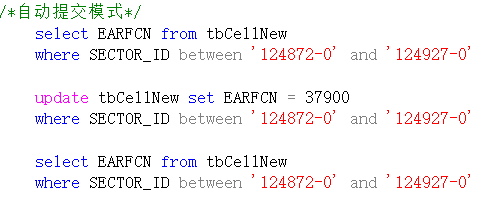
查看执行结果：



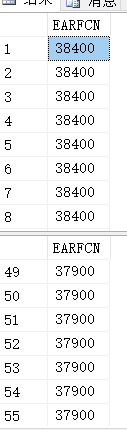
说明：结果与显式提交事务一致。结果未更新，rollback把update回滚了。

**2.3 自动提交模式：**

完整过程：



查看执行结果：



说明：自动提交模式下，每个SQL语句相当于1个事务，多条顺序提交的SQL语句对应于串行执行多个事务，未出错不会发生回滚。

**3. 数据库模式修改**

**实验要求：**

Step1. 修改TD-LTE数据库中的tbOptCell表的备份表tbOptCellnew的模式，删除列CELL\_TYPE（使用**alter table drop**）。

Step2. 修改tbOptCell表的备份表tbOptCellnew的模式，增加列CELL\_TYPE（使用**alter table add**）。

将step1、step2的数据库访问组织成1个单一事务（以显式事务的方式），采用以下2种结束方式：

（1）以commit结束；

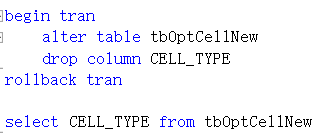
（2）以rollback结束。

查看数据库（用select语句查看被删除/增加的列），观察数据库模式修改语句（**alter table**），是否会受到**rollback**，**commit**语句的影响。

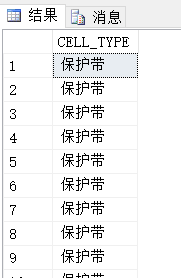
也可以自行创建表、删除表，重复以上两步，查看数据库，观察数据库模式定义语句（**create table**）模式修改语句（**drop table**）是否会受到**rollback**，**commit**语句的影响。

**3.1 删除列CELL\_TYPE**

**3.1.1 只有rollback的过程：**

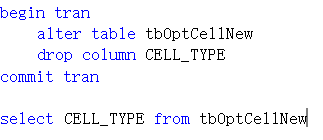


查看执行结果：

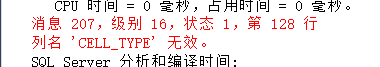


说明：只执行rollback语句时，事务结束后没有真正提交结果，没有真正删除CELL\_TYPE。

**3.1.2 执行commit的过程：**



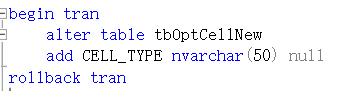
查看执行结果：



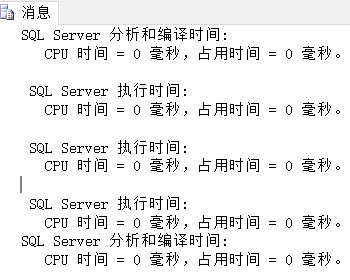
说明：执行commit语句后，成功删除CELL\_TYPE，事务结束后结果已提交，已不能再进行回滚。

**3.2 增加列CELL\_TYPE**

**3.2.1 只有rollback的过程**



执行结果：



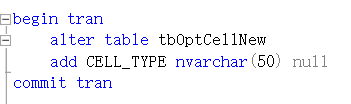
检查是否增加成功：

图片

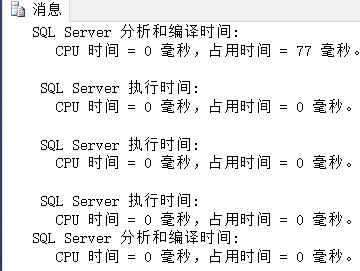
图片

说明：执行rollback语句后，事务结束后查找CELL\_TYPE显示无效，没有真正添加进去,添加语句被回滚。

**3.2.2 执行commit的过程**

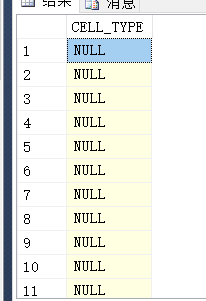


执行结果：



检查是否增加成功：

图片



说明：执行commit语句后，成功添加CELL\_TYPE，事务结束后结果已提交，已不能再进行回滚。

**4. 多条insert/delete操作执行比较**

**实验要求**

以备份表tbCellnew为访问对象，

Step1. 查询小区/基站工参表的小区天线高度（HEIGHT）小于7的SECTOR\_ID

SECTOR\_NAME和HEIGHT；

Step2. 在小区/基站工参表中，添加一条SECTOR\_ID为“211100-2”、HEIGHT为6的信息；

Step3. 删除step2所添加的信息；

Step4. 查询小区/基站工参表的小区天线高度（HEIGHT）小于7的SECTOR\_ID、SECTOR\_NAME和HEIGHT；

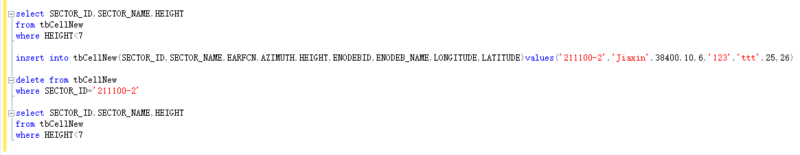
针对以上操作分别进行如下的操作：

（1）将以上操作组织成普通的SQL语句，顺序执行。

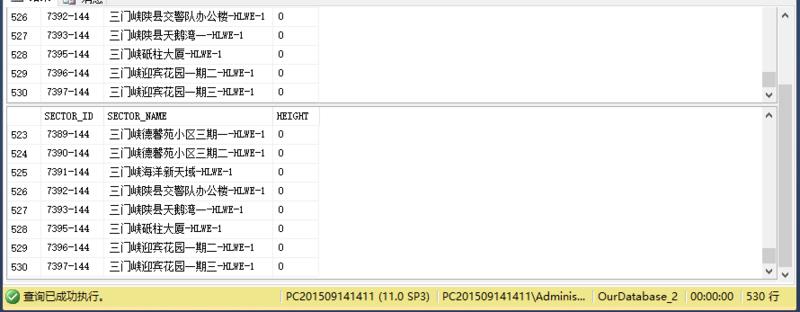
（2）将以上操作组织成事务执行（**以begin tran开始，以commit tran结束**）。

查看数据库，观察两次的执行结果有何异同。

**4.1 顺序执行**

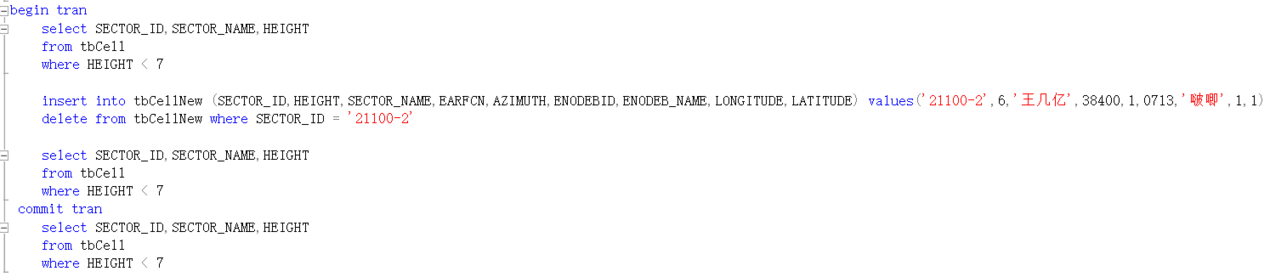


执行结果：



说明：先增加一行再删除同一行最后的结果不变，两次查询的数目相同，排列结果也相同。

**4.2 组织成事务执行**



执行结果：



说明：在事务提交后再添加一条select语句查看，先增加一行再删除同一行，表中的数据没有发生改变。

**5. 保存点Savepoint设置与回滚实验**

本实验要求在事务内部不同执行位置设置，例如添加之后、添加之前、删除之后等，使用 SAVE TRANSACTION savepoint\_name 语句创建保存点,使用ROLLBACK TRANSATCTION savepoint\_name语句将事务回滚，观察每次操作的结果。保存点提供了回滚部分事务的机制，而不是回滚到事务的开始。

**实验要求**

以小区PCI优化调整结果表tbPCIAssignment为访问对象，在创建的事务中insert插入语句后设置保存点，然后删除添加的信息，并回滚至保存点并提交事务；事务完成后再查询相应的行，观察执行结果是否插入成功，具体如下：

Step1. 查询小区PCI优化调整结果表的小区PCI为106的小区的SECTOR\_ID；

Step2. 在小区PCI优化调整结果表中，添加一条SECTOR\_ID为“220102-5”、PCI为106的信息；

Step3. 设置保存点；

Step4. 删除step2所添加的信息；

Step5. 回滚至保存点；

Step6. 事务提交结束；

Step7. 查询小区PCI优化调整结果表的小区PCI为106的小区的SECTOR\_ID；

事务结构如下：

begin tran

select语句（检查表中原始数据）

insert语句（向表中添加一行新的数据）

**save transaction ppp（设置保存点）**

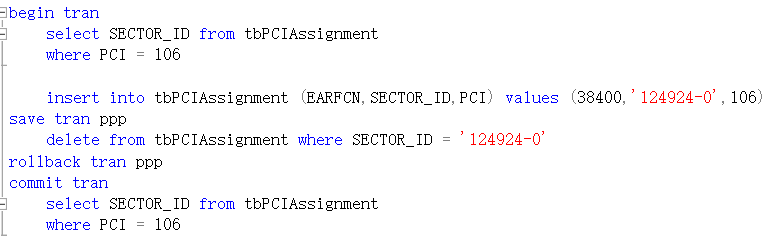
delete语句（删除insert语句添加的行）

rollback transaction ppp（回滚至保存点）

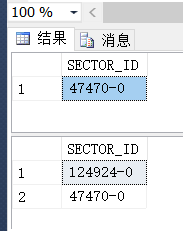
commit tran（提交事务）

select语句（检查插入数据是否成功）

事务语句：



执行结果：



说明：事务回滚到了保存点处，提交前delete语句被回滚，因此最后的结果中没有执行delete语句，SECTOR\_ID为124924-0的元组被插入数据库。

**6. 事务局部回滚/整体回滚**

**实验要求：**

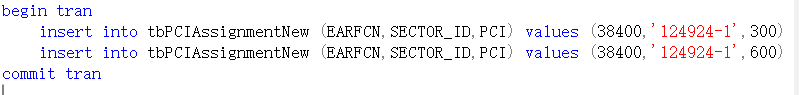
根据现网实际情况，小区PCI优化调整结果表tbPCIAssignment中的小区PCI在0到503之间。 在tbPCIAssignment的备份表tbPCIAssignmentnew上，用Alter table add check添加约束，并在该备份表上完成以下实验内容：

Step1. 在tbPCIAssignmentnew表上添加约束：加入约束check(PCI between 0 and 503)。

Step2. 以备份表tbPCIAssignmentnew表为访问对象，依次添加PCI为‘300’和‘600’的两条数据，将2条对备份表tbPCIAssignmentnew表进行顺序访问的insert语句组织成1个显示执行模式下的事务；

Step3. 利用SET XACT\_ABORT ON/OFF语句，控制事务执行过程中的进行整体回滚、局部回滚。观察并对比当事务执行违反约束时，事务结束后tbPCIAssignmentnew的内容，分析在整体回滚、局部回滚两种情况下，SQL Server提供的事务原子性保障机制。

**6.1 局部回滚**



**查看执行结果：**

图片

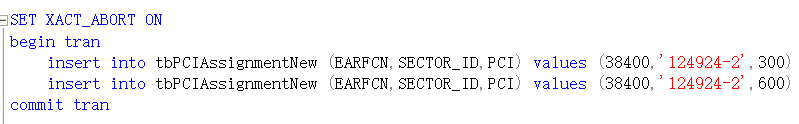
检查是否插入：

图片

图片

**说明：**不设置开关，然后插入两条数据，第二条数据导致PCI>503，违反约束，但是第一条插入数据执行成功。

**6.2 整体回滚**



**查看执行结果：**

图片

检查是否插入：

图片



**说明：设置整体回滚，插入两条数据，此时第二条数据违反了约束。事务进行整体回滚，select时发现两条数据都没有插入成功。**

当 SET XACT\_ABORT 为 ON 时，如果执行 Transact-SQL 语句产生运行时错误，则整个事务将终止并回滚。

当 SET XACT\_ABORT 为 OFF 时，有时只回滚产生错误的 Transact-SQL 语句，而事务将继续进行处理。如果错误很严重，那么即使 SET XACT\_ABORT 为 OFF，也可能回滚整个事务。OFF 是默认设置。

**二、事务并发控制机制**

本组实验涉及到事务的隔离级别、锁信息。相关知识介绍查看实验背景部分。

多个事务并发执行时，根据选择的事务隔离级别，DBMS自动使用不同粒度和不同类型的锁来保证隔离级别所要求的一致性等级，可以使用模式锁、行锁、表锁。，事务本身不需要显式请求对数据对象使用某个特定的加锁，而是由DBMS通过选择最符合要求的隔离级别来控制所维护的一致性级别。

了解各种锁类型有助于选择隔离级别和理解各个级别对性能的影响。

**1. 查看锁信息**

本实验要求观察SQL Server中有关当前活动的事务的锁管理器资源的信息。向锁管理器发出的已授予锁或正等待授予锁的每个当前活动请求分别对应一行。结果集中的列分为两组：资源组和请求组。资源组说明正在进行锁请求的资源，请求组说明锁请求。

**实验要求**

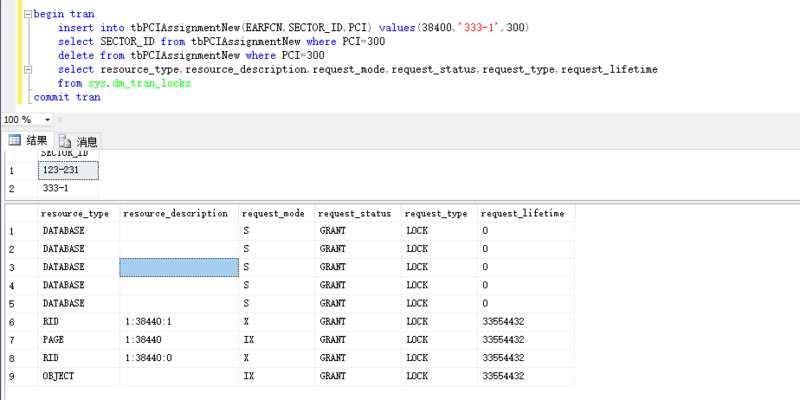
在SQL Server的默认隔离级别read committed下，从元数据表sys.dm\_tran\_locks中，查看由select/update/insert/delete等SQL访问语句组成的事务的锁信息。

可自行选择TD-LTE数据库中的一张表（tbCell/tbCellnew除外）作为事务的访问对象，要求事务中至少包含两种不同类型SQL访问语句。

可用前面实验中编写的事务语句。并根据相应的参数作简要分析。

**注意**：查看锁信息语句的作用域为当前连接，为了更好的查看当前事务的锁信息，可将相应的语句写在commit语句之前。

在tbPCIAssignmentNew上进行insert,select,delete操作



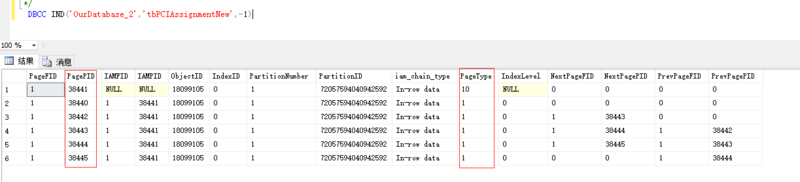
说明：

（1）从上图可看出不同粒度数据对象上的加锁模式。

DATABASE施加了S（共享）锁，页PAGE和OBJECT加了IX（意向互斥锁），表中的行RID施加了互斥锁X。

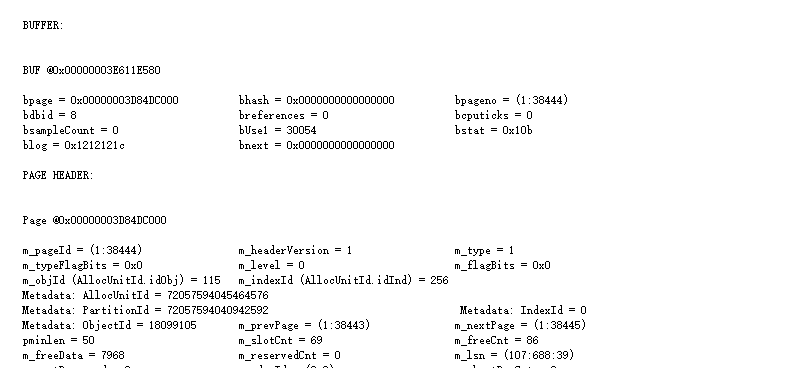
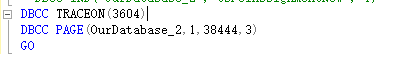
（2）在resource\_description部分，不是所有的锁粒度都支持，其中PAGE的取值表示：<file\_id>:<page\_in\_file>，RID的取值表示：<file\_id>:<page\_in\_file>:<row\_on\_page>，对应于此例中在第7行是加在file\_id为1，page\_id为38440d的意向互斥锁。

（3）查看存储结构

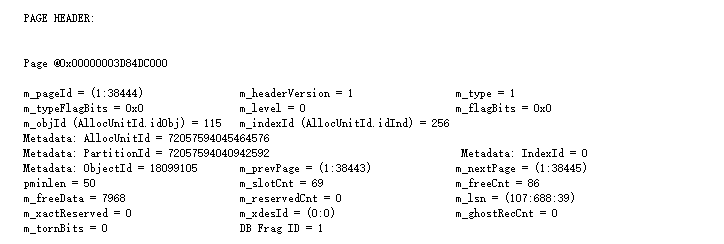


一共有六条记录。记录页面类型为10的页，是IAM页。页面类型为1的页是数据页。这六页的page\_id分别为38441,38440,38442,38443,38444,38445。

接下来看下page\_id为38444，类型为1的页里存放的数据。



关注页头信息：



Page @0x00000003D84DC000 同BUFFER中的bpage地址

m\_pageId = (1:38444)   数据页号

m\_headerVersion = 1   头文件版本号，一直为1

m\_typeFlagBits = 0x0   此页为0

m\_level = 0  该页在索引页（B树）中的级数

m\_flagBits = 0x0 页面标志

m\_objId (AllocUnitId.idObj) = 115  同Metadata：ObjectId

m\_indexId (AllocUnitId.idInd) = 256   同Metadata: IndexId

Metadata: AllocUnitId = 72057594045464576    存储单元的ID,sys.allocation\_units.allocation\_unit\_id

Metadata: PartitionId = 72057594040942592       数据页所在的分区号，sys.partitions.partition\_id

Metadata: ObjectId = 18099105   该页面所属的对象的id，sys.objects.object\_id

 m\_prevPage = (1:38443)     该数据页的前一页面；主要用在数据页、索引页和IAM页

m\_nextPage = (1:38445)  该数据页的后一页面；主要用在数据页、索引页和IAM页

pminlen = 50         定长数据所占的字节数

m\_slotCnt = 69      页面中的数据的行数

m\_freeCnt = 86 页面中剩余的空间

m\_freeData = 7968        从第一个字节到最后一个字节的空间字节数

m\_reservedCnt = 0         活动事务释放的字节数

m\_lsn = (107:688:39) 日志记录号

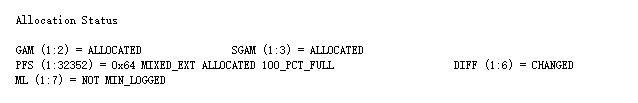
m\_xactReserved = 0        最新加入到m\_reservedCnt领域的字节数

m\_xdesId = (0:0)         添加到m\_reservedCnt的最近的事务id

m\_ghostRecCnt = 0 幻影数据的行数

m\_tornBits = 0           页的校验位或者被由数据库页面保护形式决定分页保护位取代

页面相关分配的情况：



GAM (1:2) = ALLOCATED    在GAM页上的分配情况

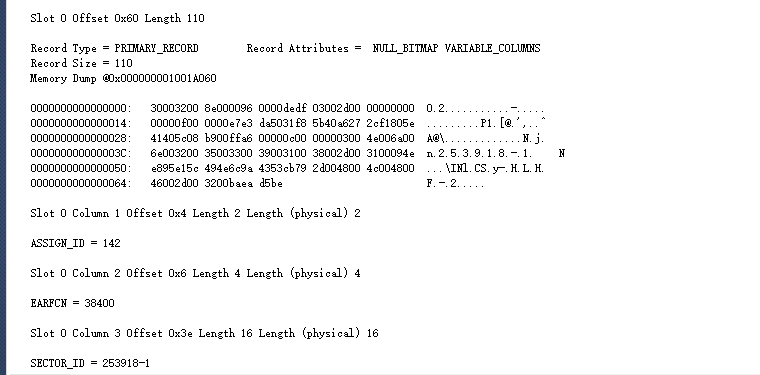
SGAM (1:3) = ALLOCATED     在SGAM页上的分配情况

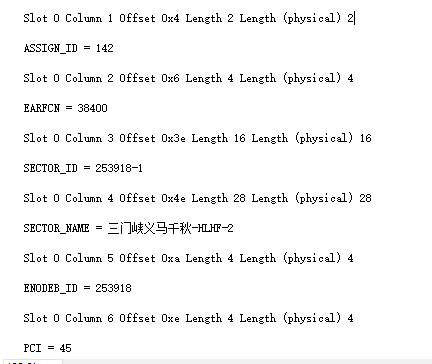
PFS (1:32352) = 0x64 MIXED\_EXT ALLOCATED 100\_PCT\_FULL   在PFS页上的分配情况，该页为100%满

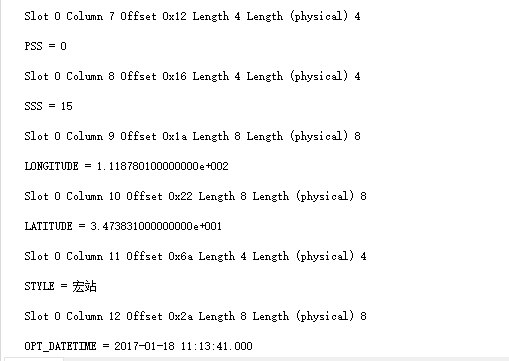
DIFF (1:6) = CHANGED

ML (1:7) = NOT MIN\_LOGGED

接下来就是用于存放实际数据的槽（slot），每条记录存放一个槽（slot）里。0号槽在页里拥有第1条数据，1号槽拥有第2条数据，以此类推。通过下面的图片，可以看到我们记录大小是110bytes，94bytes（2+4+16+28+4+4+4+4+8+8+4+8） 的定长和16 bytes 的系统行开销



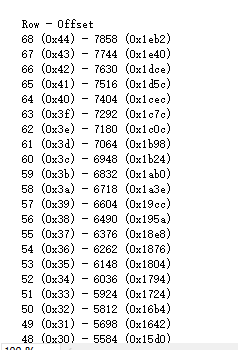


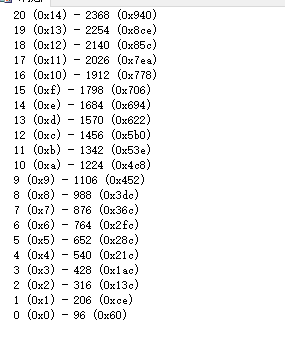


页的最后一部分是行偏移数组表，我们可以用参数为1的DBCC PAGE命令来，在输出信息的底部获得。

执行如下的命令：

图片

****



这个行偏移表，应该从下往上读。每条槽条目是一个2 bytes长的指针指向页里槽偏移量。这里我们插入了2条记录，所以表里有2个槽条目。第1条记录指向第96 bytes，刚好在页头后。这个行偏移表可以帮助我们管理页面的记录。在页里的行偏移表里，每条记录需要2 bytes的大小来存储。这个物理指针是[文件号：页号：槽号]（file:page:solt）的结构，因此在读取页的时候，可以找到堆表里的对应行，再通过行偏移表里槽号里的偏移量，就可以在页里读取到对应的行记录。如果我们要修改页中间的记录，我们并不一定需要重组整个页，我们只要修改偏移表里偏移量即可。

在页头看到当前页面还有86 bytes可以用，在此进行验证：

页的总大小-页头大小-记录长度-记录的总系统行开销-行偏移表槽占用总字节数

8\*1024 = 页的总大小,8K

96 = 页头大小 96 bytes

94 \* 69 = 每条记录的总长 \* 记录数

16 \* 69 = 每条记录的系统行开销 \* 记录数

2 \* 69 =行偏移表里每槽占用字节数 \* 记录数

8192-96-6624-1104-138=230

【但是此时计算结果错误，后来发现本数据库中槽中数据为变长，并不是定长，不能以此每条记录的总长\*记录数计算。但是方法大致一致。计算过程省略。】

**2. 单事务隔离级别及加锁信息和执行结果观察**

**实验要求**

比较不同隔离级别下，事务加锁的粒度、锁类型。

Step1. 以LTE数据库中某个关系表为对象，设计1个由多条SQL访问语句（增、删、改、查）组成的显示执行事务；

Step2. 使用DBCC USEROPTIONS查看当前的隔离级别和锁信息；然后设置不同的隔离级别，查看对比在不同的隔离级别的并发副作用；

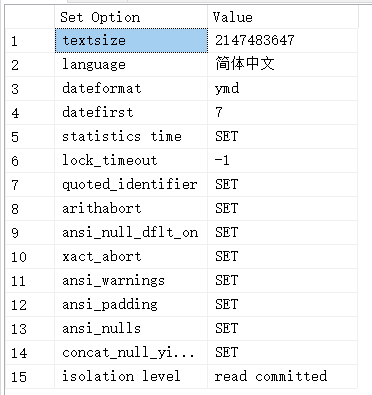
Step3. 分析对比不同隔离级别下，事务的执行结果差异。

1. 查看系统默认的当前隔离级别

在当前会话执行下面语句：

DBCC USEROPTIONS

查看执行结果：



说明：最后一行表示隔离级别，为“读提交”。

**2. 设置不同的隔离级别并查看并发副作用**

**2.1 未提交的相关性（脏读）**

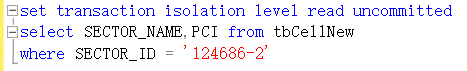
对2个并发事务，当第2个事务选择访问第1个事务正在更新的行时，会发生未提交的相关性问题。第2个事务正在读取的数据还没有确认提交，并且之后可能由正在更新此行的第1个事务所更改。

**实验要求**

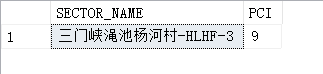
1. 开启两个连接，模拟两个并行的事务。简称会话一和会话二；
2. 在会话一中将隔离级别设置为未提交读read uncommitted，以tbCellnew为访问对象，查询SECTOR\_ID为‘124686-2’的小区的SECTOR\_NAME和PCI；
3. 在会话二中开始事务，更新SECTOR\_ID为‘124686-2’的小区的PCI为当前PCI值加20，保持未提交状态执行语句；
4. 在会话一中将隔离级别设置为未提交读，以tbCellnew为访问对象，查询SECTOR\_ID为‘124686-2’的小区的SECTOR\_NAME和PCI；
5. 消除并发副作用。

步骤1：会话1：

执行：

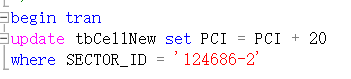


结果：

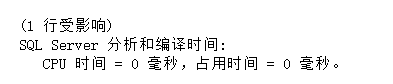


步骤2：会话2：

执行：（注意此时事务未提交）



结果：

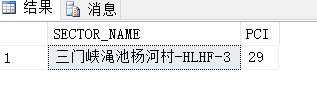


步骤3：再次回到会话1：

执行：

图片

结果：

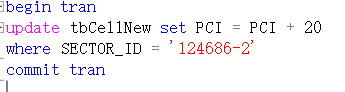


出现脏读，会话1读出了会话2未提交的数据。

步骤4：消除副作用

不能简单的把会话1的隔离级别设置为高于“未提交读”的隔离级别，这样会出现死锁，因为会话2的事务未提交，会话1要等提交完之后再读数据，就会一直等待。

为消除脏读，可以直接在会话2用commit或rollback结束事务即可。



**2.2 不一致的分析（不可重复读）**

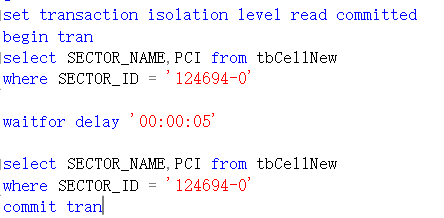
当第2个事务多次访问表中同一行数据而且每次读取不同内容时，会发生不一致的分析问题。不一致的分析与未提交的相关性类似，因为其它事务也是正在更改第二个事务正在读取的数据。然而，在不一致的分析中，第二个事务读取的数据是由已进行了更改的事务提交的。而且，不一致的分析涉及多次（两次或更多）读取同一行，而且每次信息都由其它事务更改；因而该行被非重复读取。

**实验要求**

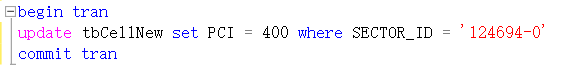
1. 开启两个连接，模拟两个并行的事务。简称会话一和会话二；
2. 在会话一中将隔离级别设置为提交读或者未提交读，以tbCellnew为访问对象，查询SECTOR\_ID为‘124694-0’的小区的SECTOR\_NAME和PCI；设置推迟等待5秒语句，然后再写一条查询语句，同样是查询SECTOR\_ID为‘124694-0’的小区的SECTOR\_NAME和PCI，执行语句后的五秒内开始第三步；
3. 第二步语句执行的五秒内在会话二中执行语句，开始事务，更新SECTOR\_ID为‘124694-0’的小区的PCI为400，并提交事务；
4. 在会话一中查看执行结果；
5. 消除并发副作用。

步骤1：会话1:

执行：（将隔离级别设置为READ UNCOMMITTED和READ COMMITTED都可以）

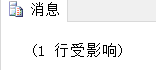


步骤2：步骤1执行后马上切换到会话2执行以下语句：



执行结果：

会话2：



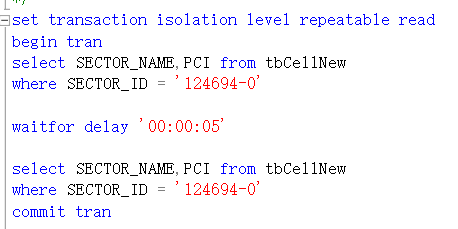
会话1：



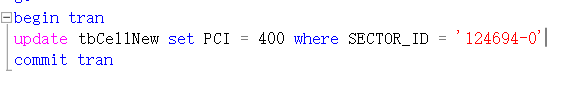
两次结果不一致，第二次读出的数据为更改后的数据，在会话1事务执行之中，会话2的事务也成功执行。出现“不可重复读”。

步骤3：消除副作用

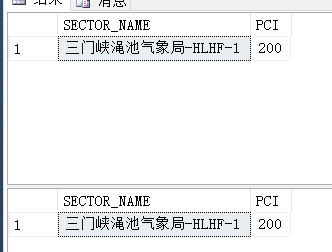
将会话1的隔离级别设置为REPEATABLE READ或SERIALIZABLE



执行完会话1马上执行会话2



会话1结果：



此时前后结果一致，为更新前的数据，因为REPEATABLE READ级别保证在一个事务中的两个读操作之间，其他的事务不能修改当前事务读取的数据。

**2.3 幻读**

当事务1对表中某行执行插入或删除操作时，如果该行属于某个事务2正在读取的行的范围内，会发生幻读问题。事务2第一次读的行范围显示出其中一行已不复存在于第二次读或后续读中，因为该行已被其它事务1删除。同样，由于其它事务1的插入操作，事务2的第二次或后续读显示有一行已不存在于原始读中。

**实验要求**

（1） 开启两个连接，模拟两个并行的事务。简称会话一和会话二；

（2） 在会话一中将隔离级别设置为READ UNCOMMITTED、READ COMMITTED或REPEATABLE READ，以tbCellnew为访问对象，查询LONGITUDE小于110.37644的小区的LONGITUDE并去重；设置推迟等待5秒语句，然后再写一条查询语句，同样是查询LONGITUDE小于110.37644的小区的LONGITUDE并去重；

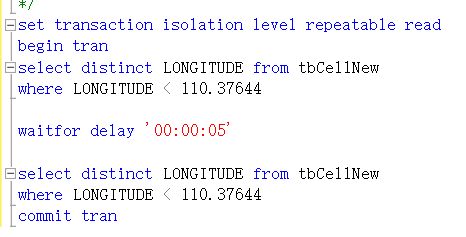
（3） 第二步语句执行的五秒内在会话二中执行语句，开始事务，添加LONGITUDE为110.366666的信息，并提交事务；

（4） 查看会话一执行结果；

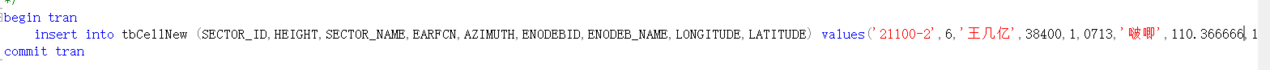
（5） 消除并发副作用。

步骤1：在会话1中执行：

（将隔离级别设置为READ UNCOMMITTED、READ COMMITTED和REPEATABLE READ都可以）

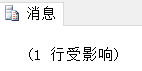


步骤2：步骤1执行后马上切换到会话2执行以下语句：

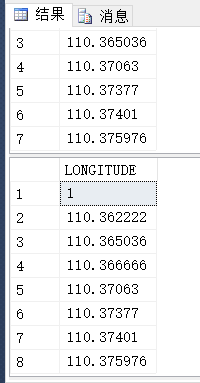


执行结果：

会话2：



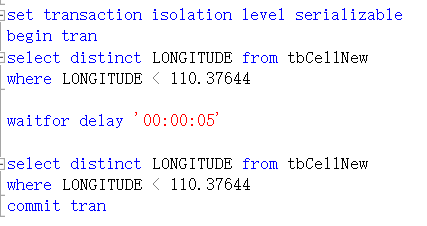
会话1：



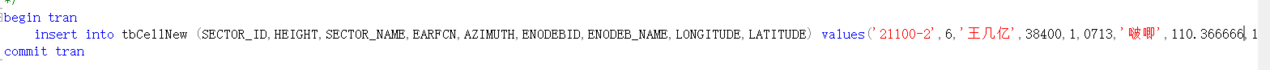
同一个事务中两次查询返回的结果记录数不同，出现了“幻读”。

步骤3：消除副作用

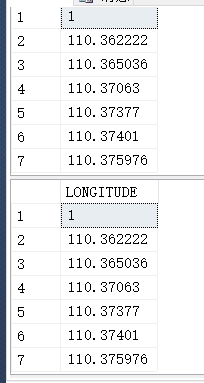
将会话1的隔离级别设置为SERIALIZABLE



执行完会话1之后马上执行会话2：



会话1结果：



同一事务中前后两次查询结果记录数相同，为插入之前的数据。

对于不同隔离级别的说明：

（1）**READ UNCOMMITTED** 事务隔离级别根本就没有提供事务间的隔离，它允许违反并发性原则的最基本形式之一 -- 脏读。READ UNCOMMITTED 常应用于：单用户系统；系统中两个事务同时访问同一资源的可能性为零或几乎为零；当使用Rowversion数据类型控制并发性时 。

（2）**READ COMMITTED** 已提交读。通过仅允许一个事务读取另一个事务中已经提交的数据，READ COMMITTED 事务隔离级别防止了“脏读”问题。这是SQL Server中默认的事务隔离级别。

（3）**REPEATABLE READ** 可重复读。该级别保证在一个事务中的两个读操作之间，其他的事务不能修改当前事务读取的数据。

（4）REPEATABLE READ能保证事务可重复读，但是事务只锁定查询第一次运行时获取的数据资源（数据行），而不能锁定查询结果之外的行，就是原本不存在于数据表中的数据，所以会出现幻读。而**SERIALIZABLE** 事务隔离级别防止的并发性问题，可以防止幻读。

**实验总结**

本实验中遇到了以下问题：

问题一：

“保存点Savepoint设置与回滚实验”中，没有按照预期结果将SECTOR\_ID为“220102-5”、PCI为106的数据插入表格中。

分析与解决：

单步执行插入语句后发现是违反了tbPCIAssignment的外键约束。我们一个人是选择了新建一个备份表格，另一个人选择了一个符合要求的SECTOR\_ID作为插入。

问题二：

"查看锁信息"实验中，在查看数据库内部的存储信息过程中，按照网页提示的方法验证了剩余空间，发现与实际显示的剩余空间值不同。

分析与解决：

网页中给的例子每条数据为定长，所以可以直接用每条数据长度\*条数计算数据所占的空间，但是因为我们的数据中有的类型为不定长，所以数据长度不是定长的，不能用此方法计算数据所占空间，可以将每条数据所占空间直接相加算出。

心得：

通过本次实验对于数据库中事务的运行和控制有了更深刻的理解。实验中我们学习到只用rollback结束事务时，事务内部的结果会发生改变，但是事务结束后结果没有改变，而执行commit后结果发生改变并且提交至数据库，不能再进行回滚。在自动提交模式下，每个SQL语句相当于1个事务，多条顺序提交的SQL语句对应于串行执行多个事务，未出错不会发生回滚。事务的并发处理中，对于脏读，不可重复读和幻读有了更为深刻的理解。所谓脏读，就是对2个并发事务，当第2个事务选择访问第1个事务正在更新的行时，会发生未提交的相关性问题。第2个事务正在读取的数据还没有确认提交，并且之后可能由正在更新此行的第1个事务所更改。不可重复读即当第2个事务多次访问表中同一行数据而且每次读取不同内容时，会发生不一致的分析问题。而幻读是当事务1对表中某行执行插入或删除操作时，如果该行属于某个事务2正在读取的行的范围内，会发生幻读问题。