**学生1：**

**在学生-课程数据库上完成如下操作。**

**1.事务的定义**

**（1）了解事务运行模式**

**在查询编辑器中输入如下语句并执行，最后语句中包含语义错误，查看前面语句执行情况。了解在没有显示定义事务的情况下，DBMS默认每个SQL语句就是一个事务。发生错误后，只回滚一个SQL语句。**

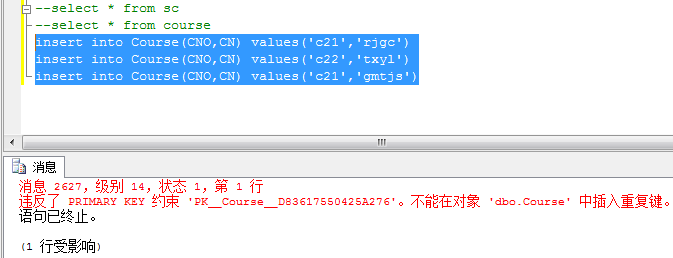
--select \* from sc

--select \* from c

insert into C(CNO,CN) values('c21','rjgc')

insert into C(CNO,CN) values('c22','txyl')

insert into C(CNO,CN) values('c21','gmtjs')



**（2）显示定义事务**

**①　定义一事务，包含前面的三条插入语句，最后语句中包含语义错误。执行该事务，与前面的执行结果进行比较，分析结果。**

BEGIN TRANSACTION

insert into Course(CNO,CN) values('c22','txyl')

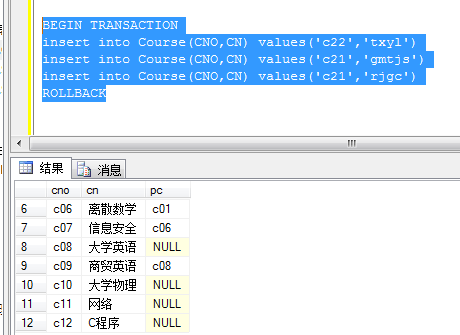
insert into Course(CNO,CN) values('c21','gmtjs')

insert into Course(CNO,CN) values('c21','rjgc')

ROLLBACK

**执行结果：**

select \* from course



创建一个事务，以ROLLBACK结尾时，只要事务中有一句不能执行，那么都不会执行。

**②　修改定义中的错误，以rollback和 commit两种方式结束事务。查看被更新表的数据，说明rollback的 commit不同。**

不同之处：

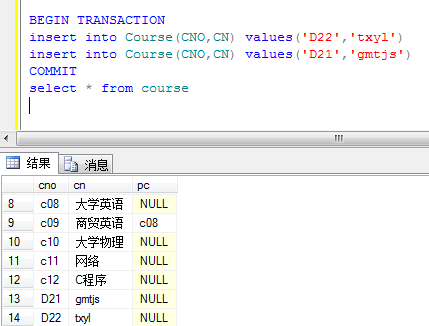
以rollback结尾时，不论成功与否，都会回滚；而以commit结尾时，只要能成功执行，就能进行更改。

BEGIN TRANSACTION

insert into Course(CNO,CN) values('D22','txyl')

insert into Course(CNO,CN) values('D21','gmtjs')

COMMIT



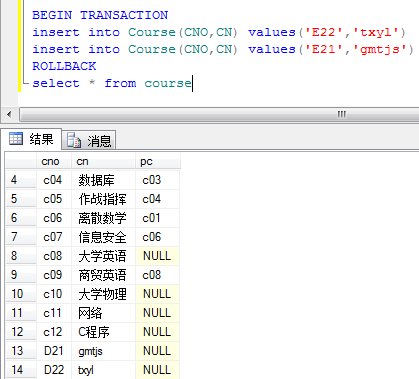
BEGIN TRANSACTION

insert into Course(CNO,CN) values('E22','txyl')

insert into Course(CNO,CN) values('E21','gmtjs')

ROLLBACK

select \* from course



**2．事务故障恢复**

**事务故障破坏事务的原子性。事务故障后，系统自动强行回滚（rollback）该事务。即利用日志撤销此事务已对数据库的更新，保持事务的原子性。对提供检测点的DBMS，事务的回滚与设置的检测点有关。**

**在“学生—课程”学数据库上，执行下面的事务，分析结果，阐述设置存储点的作用。**

begin tran t1

select \* from sc

insert into SC(SNO,CNO) values('s01','c05')

select \* from sc

save tran t1

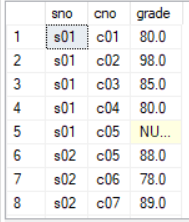
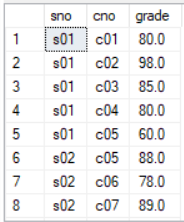
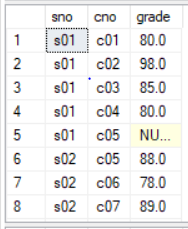
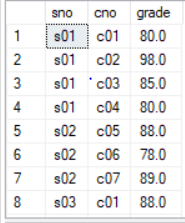
update sc set grade=60 where sno=’s01’ and cno=’c05’

select \* from sc

rollback tran t1

select \* from sc

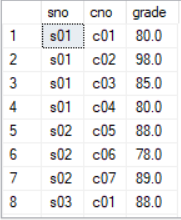
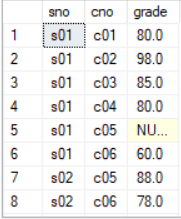
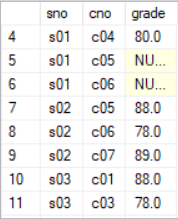
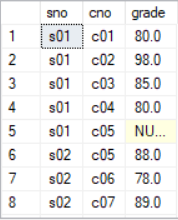
**比较每次查询的结果，说明 save tran t1的功能。把rollback tran t1改为rollback看一下，结果如何？**



第一次select 第二次select 第三次select 第四次select

Save tran t1的功能是将这条语句上面所执行的事务记录下来，执行rollback tran t1后，会回到刚才记录的那个状态。

**把rollback tran t1改为rollback，执行结果：**



第一次select 第二次select 第三次select 第四次select

若把rollback tran t1改为rollback，会发现回到最初事务没有执行的那个状态。

**3．事务的并发控制**

**大多数DBMS为并发事务提供封锁请求，有共享锁和排它锁。SQL SERVER中，为了模拟并发环境，打开多个查询窗口即可。**

**在“学生—课程”数据库上完成如下操作。**

**（1）在一个查询窗口中执行事务T1**

begin tran T1

SELECT \* from SC where SNO='s07'

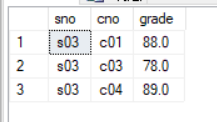
**（2）在另一个查询窗口中执行事务T2，此时事务T1还未结束。**

begin tran T2

SELECT \* from SC where SNO='s07'

**比较T2与T1的结果。**

T1和T2的执行结果一样

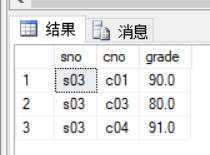


**（3）返回事务T1继续运行事务，进行数据修改并查询，例如：**

Update sc set grade=grade+2 where sno=’s07’

Select \* from where sno=’s07’

**观察结果。**



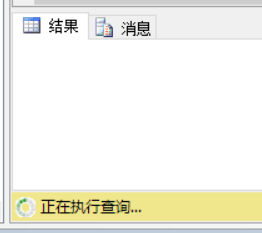
**（4）回到事务T2，事务T2进行同样操作，进行数据修改并查询。**

Update sc set grade=grade+2 where sno=’s07’

Select \* from sc where sno=’s07’

**说明此时的状态。**

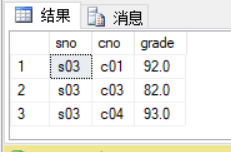
T2一直在执行中



**（5）回到事务T1，提交事务。即执行commit。**

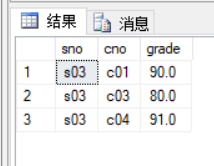
**（6）返回到事务T2，说明此时的状态和结果。**

T2成功执行，并且在T1加2的基础上又再加上2。



**（7）然后强行关闭两个查询，即T1已提交，T2不提交。**

**（8）再执行Select \* from sc where sno=’s07’，解释结果。对发生了故障的事务T2，系统是如何做的？保证了事务的什么特性？**



因为事务有ACID特性，事务T1提交，而T2没有提交，所以查看到的内容是T1执行后的内容。

保证了事务的ACID特性。

**系统是如何做的？**

事务的ACID特性是由关系数据库管理系统来实现的。数据库管理系统采用日志来保证事务的原子性、一致性和持久性。日志记录了事务对数据库所做的更新，如果某个事务在执行过程中发生错误，就可以根据日志，撤销事务对数据库已做的更新，使数据库退回到执行事务前的初始状态。

数据库管理系统采用锁机制来实现事务的隔离性。当多个事务同时更新数据库中相同的数据时，只允许持有锁的事务能更新该数据，其他事务必须等待，直到前一个事务释放了锁，其他事务才有机会更新该数据。

**1. 事务的回滚与设置检查点的关系。**

事务回滚：是数据库返回到事务开始的状态：事务在运行过程中发生某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的更新操作全部撤销，使数据库回滚到事务开始时的状态。

检查点：出于性能方面的考虑，数据库引擎对内存（缓冲区缓存）中的数据库页进行修改，但在每次更改后不将这些页写入磁盘。 相反，数据库引擎定期发出对每个数据库的检查点命令。 “检查点”将当前内存中已修改的页和事务日志信息从内存写入磁盘，并记录有关事务日志的信息。对于自动、手动和内部检查点，在数据库恢复期间只有在最新检查点后所做的修改需要前滚。 这将减少恢复数据库所需的时间。

**学生2：**

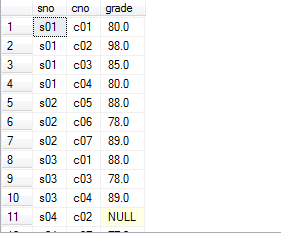
在学生-课程数据库上完成如下操作。

1.事务的定义

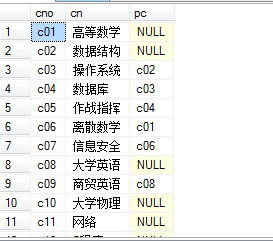
（1）了解事务运行模式

在查询编辑器中输入如下语句并执行，最后语句中包含语义错误，查看前面语句执行情况。了解在没有显示定义事务的情况下，DBMS默认每个SQL语句就是一个事务。发生错误后，只回滚一个SQL语句。

--select \* from sc



--select \* from Course



insert into Course(CNO,CN) values('c21','rjgc')

插入成功

insert into Course(CNO,CN) values('c22','txyl')

插入成功

insert into Course(CNO,CN) values('c21','gmtjs')

**由于存在主键约束，违背了主键约束，导致插入失败**

（2）显示定义事务

定义一事务，包含前面的三条插入语句，最后语句中包含语义错误。执行该事务，与前面的执行结果进行比较，分析结果。

修改定义中的错误，以rollback和 commit两种方式结束事务。查看被更新表的数据，说明rollback的 commit不同。

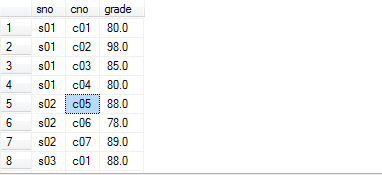
事务故障恢复

事务故障破坏事务的原子性。事务故障后，系统自动强行回滚（rollback）该事务。即利用日志撤销此事务已对数据库的更新，保持事务的原子性。对提供检测点的DBMS，事务的回滚与设置的检测点有关。

在“生—课程”学数据库上，执行下面的事务，分析结果，阐述设置存储点的作用。

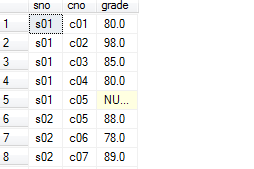
begin tran t1

select \* from sc



insert into SC(SNO,CNO) values('s01','c05')

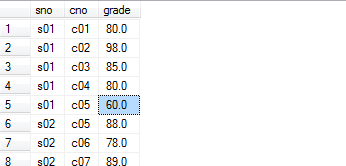
select \* from sc



save tran t1

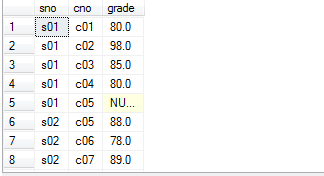
update sc set grade=60 where sno=’s01’ and cno=’c05’

select \* from sc



rollback tran t1

select \* from sc



比较每次查询的结果，说明 save tran t1的功能。把rollback tran t1改为rollback看一下，结果如何？

**第一次查询出所有表中的数据，第二次执行插入操作再次查询发现插入数据成功，插入成功后定义一个存储点，然后执行更新操作，在rollback之前查询表，表中数据已经更新，当执行到rollback tran t1时会发现表中数据回到更新操作之前若将rollback tran t1改为rollback则在最后一张表中的数据和第一次查询结果相同，插入和更新操作全部被取消**

事务的并发控制

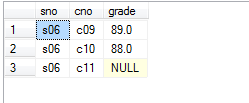
大多数DBMS为并发事务提供封锁请求，有共享锁和排它锁。SQL SERVER中，为了模拟并发环境，打开多个查询窗口即可。

在“学生—课程”数据库上完成如下操作。

（1）在一个查询窗口中执行事务T1

begin tran T1

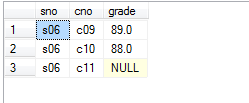
SELECT \* from SC where SNO='s07'



（2）在另一个查询窗口中执行事务T2，此时事务T1还未结束。

begin tran T2

SELECT \* from SC where SNO='s07'



比较T2与T1的结果。

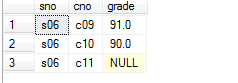
T2和T1的结果相同

（3）返回事务T1继续运行事务，进行数据修改并查询，例如：

Update sc set grade=grade+2 where sno=’s07’

Select \* from SC where sno=’s07’

观察结果。

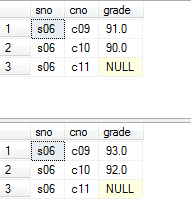


（4）回到事务T2，事务T2进行同样操作，进行数据修改并查询。

Update sc set grade=grade+2 where sno=’s06’

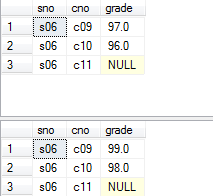
Select \* from sc where sno=’s06’

说明此时的状态。



数据再次被更新，此时事务默认提交但并未结束

1. 回到事务T1，提交事务。即执行commit。



1. 返回到事务T2，说明此时的状态和结果。

事务T2仍处于执行状态并未结束

1. 然后强行关闭两个查询，即T1已提交，T2不提交。
2. 再执行Select \* from sc where sno=’s06’，解释结果。对发生了故障的事务T2，系统是如何做的？保证了事务的什么特性？

系统只处理事务T1中对数据的操作，但是不执行事务T2，保证了事务的隔离性和一致性

\*4.多粒度封锁

SQL SERVER的封锁操作是在增、删、改、查等语句中添加WITH子句来完成的，封锁方式和粒度有以下几种。

TABLOCK:对表实施封锁，

HOLDLOCK,TABLOCK

TABLOCKX

UPDLOCK

NOLOCK

**学生3：**

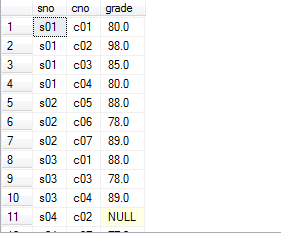
学生-课程数据库上完成如下操作。

1.事务的定义

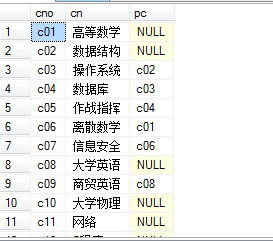
（1）了解事务运行模式

在查询编辑器中输入如下语句并执行，最后语句中包含语义错误，查看前面语句执行情况。了解在没有显示定义事务的情况下，DBMS默认每个SQL语句就是一个事务。发生错误后，只回滚一个SQL语句。

--select \* from sc



--select \* from Course



insert into Course(CNO,CN) values('c21','rjgc')

插入成功

insert into Course(CNO,CN) values('c22','txyl')

插入成功

insert into Course(CNO,CN) values('c21','gmtjs')

**由于存在主键约束，违背了主键约束，导致插入失败**

（2）显示定义事务

定义一事务，包含前面的三条插入语句，最后语句中包含语义错误。执行该事务，与前面的执行结果进行比较，分析结果。

修改定义中的错误，以rollback和 commit两种方式结束事务。查看被更新表的数据，说明rollback的 commit不同。

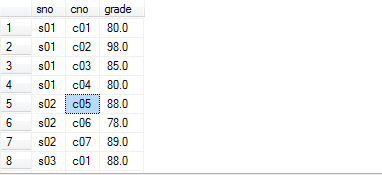
事务故障恢复

事务故障破坏事务的原子性。事务故障后，系统自动强行回滚（rollback）该事务。即利用日志撤销此事务已对数据库的更新，保持事务的原子性。对提供检测点的DBMS，事务的回滚与设置的检测点有关。

在“生—课程”学数据库上，执行下面的事务，分析结果，阐述设置存储点的作用。

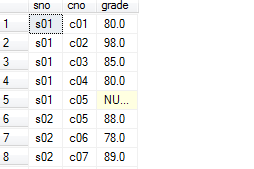
begin tran t1

select \* from sc



insert into SC(SNO,CNO) values('s01','c05')

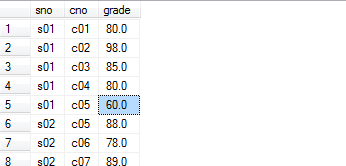
select \* from sc



save tran t1

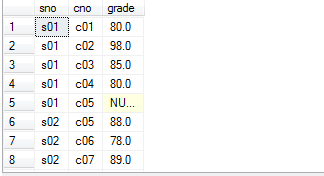
update sc set grade=60 where sno=’s01’ and cno=’c05’

select \* from sc



rollback tran t1

select \* from sc



比较每次查询的结果，说明 save tran t1的功能。把rollback tran t1改为rollback看一下，结果如何？

**第一次查询出所有表中的数据，第二次执行插入操作再次查询发现插入数据成功，插入成功后定义一个存储点，然后执行更新操作，在rollback之前查询表，表中数据已经更新，当执行到rollback tran t1时会发现表中数据回到更新操作之前若将rollback tran t1改为rollback则在最后一张表中的数据和第一次查询结果相同，插入和更新操作全部被取消**

事务的并发控制

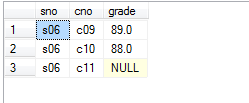
大多数DBMS为并发事务提供封锁请求，有共享锁和排它锁。SQL SERVER中，为了模拟并发环境，打开多个查询窗口即可。

在“学生—课程”数据库上完成如下操作。

（1）在一个查询窗口中执行事务T1

begin tran T1

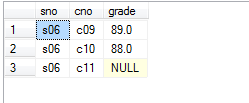
SELECT \* from SC where SNO='s07'



（2）在另一个查询窗口中执行事务T2，此时事务T1还未结束。

begin tran T2

SELECT \* from SC where SNO='s07'



比较T2与T1的结果。

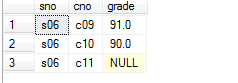
T2和T1的结果相同

（3）返回事务T1继续运行事务，进行数据修改并查询，例如：

Update sc set grade=grade+2 where sno=’s07’

Select \* from SC where sno=’s07’

观察结果。

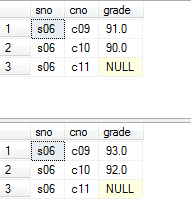


（4）回到事务T2，事务T2进行同样操作，进行数据修改并查询。

Update sc set grade=grade+2 where sno=’s06’

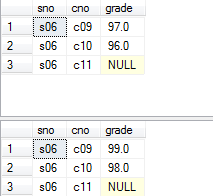
Select \* from sc where sno=’s06’

说明此时的状态。



数据再次被更新，此时事务默认提交但并未结束

1. 回到事务T1，提交事务。即执行commit。



1. 返回到事务T2，说明此时的状态和结果。

事务T2仍处于执行状态并未结束

1. 然后强行关闭两个查询，即T1已提交，T2不提交。
2. 再执行Select \* from sc where sno=’s06’，解释结果。对发生了故障的事务T2，系统是如何做的？保证了事务的什么特性？

系统只处理事务T1中对数据的操作，但是不执行事务T2，保证了事务的隔离性和一致性

\*4.多粒度封锁

SQL SERVER的封锁操作是在增、删、改、查等语句中添加WITH子句来完成的，封锁方式和粒度有以下几种。

TABLOCK:对表实施封锁，

HOLDLOCK,TABLOCK

TABLOCKX

UPDLOCK

NOLOCK