实验十五 SQLServer2008 管理事务和锁

实验目的：

- ●了解事务的 ACID 属性和数据库的并发控制；
- ● 理解不同事务隔离级别的作用；
- ● 掌握事务的基本操作、隔离级别的设置。
- ● 了解死锁发生的情况
- ● 掌握死锁的一般处理过程

实验内容：

- ● 模拟因并发操作而产生的数据不一致问题
  （丢失修改、读脏数据、不可重复读、幻读）。
- ● 运用锁与事务隔离级别解决数据不一致问题。
- ● 查看进程信息和锁与对象信息

实验准备：

请阅读以下知识点：

●事务是用户定义的一个数据库操作序列。作为事务是要么全执行，要么都不执行。

四个特性为：（1）原子性（2）一致性（3）完整性（4）持久性

●并发控制机制就是要用正确的方式调度并发操作，使一个用户事务的执行不受其他事务的干扰。确保一个用户所做的修改并不逆向地影响其它人的修改，这有两种类型：

- ■ 悲观的并发控制在读取数据以准备进行更新时对数据加锁，其它用户不能执行有可能改变基础数据的动作，直到对数据加锁的用户完成对数据进行的工作后为止。在数据高度争用的场合应该使用悲观的并发控制方法，这时候由于使用加锁技术对数据进行保护而付出的代价，比并发冲突发生后回滚事务付出的代价要小。
- ■ 乐观的并发控制在开始读取数据时不对数据加锁，当执行更新时，SQLSERVER 进行核对以确定基础数据从开始读取后是否发生了变化，如果发生了变化，用户会收到报错，事务回滚，用户重新开始。适合在数据争用不严重的场合中使用，这时偶乐回滚事务所付出的代价，比从开始读取数据就进行加锁付出的代价要小。SQL Server 支持范围广泛的悲观和乐观的并发控制机制，用户通过指定连接的事务隔离级别来指定并发控制的类型。

- ● SQL Server 2008 中定义的隔离级别：
  - ■ 1、Transaction Isolation Level
    - ◆ [1] READ UNCOMMITTED(未提交读,读脏 Dirty Read),相当于(NOLOCK)，命令 SQL Server 不发出共享锁且不给予独占锁的名义，这时可能产生脏读出。
    - ◆ [2] READ COMMITTED (Default) (已提交读,默认级别)命令 SQL Server 读出时使用共享锁，不会产生脏读出。
    - ◆ [3] REPEATABLE READ (可以重复读),相当于(HOLDLOCK)，不能发生脏读出和不能重复的读出，直到事务结束一直对事务加锁。
    - ◆ [4] SERIALIZABLE (可序列化)防止其它用户更新或插入与事务中 WHERE 子句中的规则相匹配的新行。
    - ◆ [5] SNAPSHOT (快照)
  - ■ 2、Statement Isolation Level
    - ◆ [6] READ COMMITTED SNAPSHOT (已经提交读隔离)

对于前四个隔离级别：**READ UNCOMMITTED < READ COMMITTED < REPEATABLE READ**

< SERIALIZABLE。

●事务的分类与基本操作

- **分类：**
  - **隐式的事务：用 SET IMPLICIT_TRANSACTIONS ON 语句将隐式事务的开关打开后，每个 SQL 语句，如 INSERT、UPDATE、DELETE 都作为一个事务执行。**
  - **显示的或用户自定义事务：事务的语句在 BEGIN TRANSACTION 和 COMMIT TRANSACTION 子句间组成一组，也就是说用 BEGIN TRANSACTION 明确指定开始的事务。**
- **操作：**
  - **开始事务：BEGIN TRANSACTION**
  - **提交事务：COMMIT TRANSACTION**
  - **回滚（撤消）事务：ROLLBACK TRANSACTION**

●事务的隔离级别设置

- 设置事务的隔离级别为程序员提供了在冒着增加完整性风险的前提下得到更高的数据并发访问能力，隔离级别越高，读操作的请求锁定就越严格，锁的持有时间久越长；所以隔离级别越高，一致性就越高，并发性就越低，同时性能也相对影响越大。
- 获取事务隔离级别(isolation level)
- DBCC USEROPTIONS

●设置隔离

- 设置会话隔离

  SET TRANSACTION ISOLATIONLEVEL<ISOLATION NAME>

  --注意:在设置回话隔离时(REPEATABLE READ)两个单词需要用空格隔开,但是在表隔离中可以粘在一起(REPEATABLEREAD)
- 设置查询表隔离
  - SELECT ....FROM<TABLE>WITH (<ISOLATION NAME>)

●死锁

- 死锁是怎样发生的？
  - 当两个事务已经锁定了独立的对象，而且每个事务又申请对另一个事务的对象加锁时，就会发生死锁。
  - 当几个运行时间很长的事务在同一数据库中并发执行时，也可能发生死锁。死锁还可能作为查询优化器对复杂查询（比如连接）处理顺序的结果而发生，因为在这里不能对处理顺序进行必要的控制。
- Sqlserver 如何结束死锁？
  - 回滚被死锁的事务。
  - 以 1205 号消息通知被死锁的应用程序。
  - 取消死锁牺牲者的当前请求。
  - 允许其它事务继续处理。

实验步骤：

一、环境准备

需求描述：某学校要求学生把校园卡与银行卡绑定，学生数据库 STUDENT 中有两张基本表，其中校园卡号 cardid 是学生的学号：

Icbc_card(studcardid,icbcid,balance) –校园卡编号 ID，工行卡 ID,银行卡余额

Campus_card(studcardid,balance) --校园卡编号 ID，校园卡余额

- 创建数据库：

```
CREATE DATABASE [STUDENT] ON  PRIMARY
( NAME = N'STUDENT', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\STUDENT.mdf' , SIZE = 3072KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'STUDENT_log', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\STUDENT_log.ldf' , SIZE = 1024KB ,
MAXSIZE = 2048GB , FILEGROWTH = 10%)
       GO
```

- **创建基本表**

```
use STUDENT
GO
create table campus_card
(
  studcardid char(8),
  balance decimal(10,2)
  )
go
create table icbc_card
(
  studcardid char(8) ,
  icbcid char(10),
  balance decimal(10,2)
)
```

GO

- **插入数据：**

```
insert into campus_card values('20150031',30)
insert into campus_card values('20150032',50)
insert into campus_card values('20150033',70)
-----------------------------------------
insert into icbc_card values('20150031','2015003101',1000)
insert into icbc_card values('20150032','2015003201',1000)
```

```
insert into icbc_card values('20150033','2015003301',1000)
```

studcardid	balance
20150031	30.00
20150032	50.00
20150033	70.00
NULL	NULL

LKZJ.STUDENT - dbo.icbc_card | LKZJ.STUDENT - dbo.c

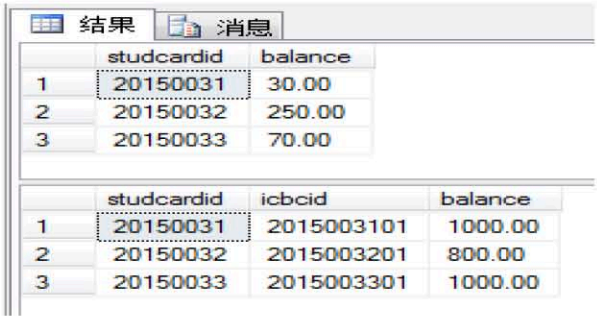
studcardid	icbcid	balance
20150031	2015003101	1000.00
20150032	2015003201	1000.00
20150033	2015003301	1000.00
NULL	NULL	NULL

二、针对数据库 STUDENT,按要求完成以下内容

1. 编写一个事务处理实现如下操作：学号为 20150032 的学生需要从银行卡中转帐 200 元到自己的校园卡中，若过程中出现错误则回滚。

```
set transaction isolation level repeatable read
Begin  transaction
use STUDENT
go

declare @x decimal(10,2)
select @x=balance
   from  icbc_card
where studcardid='20150032'
set @x=@x-200
if (@x>=0)
    begin
       update icbc_card set balance=@x where studcardid='20150032'
       update campus_card set balance=balance+200 where
studcardid='20150032'
       commit tran
    end
else
   begin
     print  '余额不足，不能转帐！'
     rollback tran
   end
   ----------------------------------
select * from campus_card
select * from icbc_card
```



结果 消息

	studcardid	balance
1	20150031	30.00
2	20150032	250.00
3	20150033	70.00

	studcardid	icbcid	balance
1	20150031	2015003101	1000.00
2	20150032	2015003201	800.00
3	20150033	2015003301	1000.00

2. 基于当前数据库STUDENT,设计样例展示四种数据不一致问题：丢失修改、读脏数据、不可重复读和幻读（删除和插入）。

（1） 丢失修改

```
--事务1
Begin tran
    declare @balance decimal(10,2)
    select @balance=balance from campus_card where studcardid='20150033'
    waitfor delay '00:00:05'
    set @balance=@balance-10
    update campus_card set balance=@balance where studcardid='20150033'
    commit tran
    go
    select balance from campus_card where studcardid='20150033'

--事务2
Begin tran
    declare @balance1 decimal(10,2)
    select @balance1=balance from campus_card where
studcardid='20150033'
    waitfor delay '00:00:05'
    set @balance=@balance-20
    update campus_card set balance=@balance1 where studcardid='20150033'
    commit tran
    go
    select balance from campus_card where studcardid='20150033'
```

结果 消息

	balance
1	60.00

结果 消息

	balance
1	50.00

----事务1更改了数据，结果为60，但是没有被读到。最终事务2的结果覆盖了事务1的更改值，结果不是期望值40

（2）读脏数据

```
--（2）读脏数据
----事务1
set transaction isolation level read uncommitted
```

```
---read uncommitted 执行脏读，不发出共享锁，也不接受排他锁
begin tran
  declare @balance decimal(10,2)
  select  @balance=balance from campus_card
        where studcardid='20150032'
        update campus_card set balance =@balance+100
          where  studcardid='20150032'
          waitfor delay '00:00:05'
         rollback tran
         ----回滚
        go
        select balance from campus_card where studcardid='20150032'
--事务2
set transaction  isolation level read uncommitted
begin tran
    declare @balance decimal(10,2)
    select @balance=balance from campus_card where studcardid='20150032'
    update campus_card set @balance =@balance+50  where
studcardid='20150032'
commit tran
go
select balance from campus_card where studcardid='20150032'
```

结果 消息

	balance
1	400.00

-----事务1更改了数据，事务2读取了表中更改后的值再进行操作，事务1回滚，最终的表存储了错误结果。

--==========

（3）　不可重复读

```
--（3）不可重复读
--事务1

SET TRANSACTION ISOLATION LEVEL  READ COMMITTED
use STUDENT
GO
begin  tran
    select balance from  campus_card where studcardid='20150031'
    waitfor delay '00:00:10'
    select balance from campus_card where studcardid='20150031'
    commit tran
```

```
    --事务2
SET TRANSACTION ISOLATION LEVEL  READ COMMITTED
USE STUDENT
GO
begin  tran
    update campus_card set balance=balance+200 where
studcardid='20150031'
    commit tran
```

结果 消息

	balance
1	30.00

	balance
1	230.00

--事务1读取了数据，事务2更改了数据，事务1再读取数据，事务1两次读取都不一样！

```
--  （4）   幻读
--插入
--事务1
set transaction isolation level read committed
begin tran
    use STUDENT
    go
    select balance from campus_card where studcardid='20150031'
    waitfor delay '00:00:05'
    select balance from campus_card where studcardid='20150031'
commit tran
--事务2
set transaction isolation level serializable
begin tran
  insert into campus_card values('20150031',30)
  commit tran
```

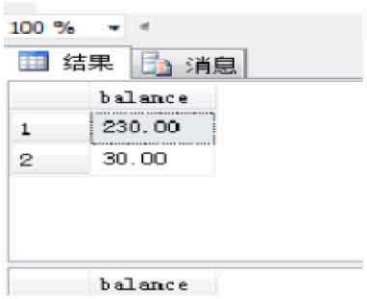
100 % 结果 消息

	balance
1	230.00

	balance
1	230.00
2	30.00

```
---删除
--事务1
set transaction isolation level read committed
begin tran
use STUDENT
go
    select balance from campus_card where studcardid='20150031'
    waitfor delay  '00:00:05'
    select balance from campus_card where studcardid='20150031'
    commit tran

---事务2
SET TRAN ISOLATION  LEVEL REPEATABLE READ
BEGIN  TRAN
    DELETE FROM campus_card where studcardid='20150031'
commit tran
    --insert into campus_card(studcardid,balance) values('20150031',30)
              --select * from campus_card
```



3. 利用锁机、数据库的隔离级别等知识，设计方案分别解决上述丢失修改、读脏数据和不可重复读（幻读）的数据不一致问题，可以用 sp_lock 过程查看当前锁的状态。

   修改隔离级别以确保数据的正确性：

   丢失修改，在 SQL 语句前加未提交读：

   SET TRAN ISOLATION LEVEL READ UNCOMMITTED

   读脏数据，在 SQL 语句前加已提交读：

   Set tran isolation level read committed

   不可重复读，在 SQL 语句前加可重复读：

   Set tran isolation level repeatable read

   幻读，在 SQL 语句前加可串行读：

   Set tran isolation level read committed

4. 构造一个出现死锁的情况

   将锁的级别改为提交可读

   Set tran isolation level read committed

   --事务 1

   Begin tran

```
Declare @read int
Select @read=grade
From sc
Where sno='95003'
Waitfor delay '00:00:10'
Update sc
Set grade =@read-1
Where sno='95003'
--事务 2
Begin tran
Declare @read int
Select @read=grade
From sc
Where sno='95003'
Update sc
Set grade=@read-1
Where sno='95003'
```


5. 利用 DBCC log 命令查看 Student 数据库的事务日志

DBCC LOG('Student',TYPE=2)

结果 消息

	Current LSN	Operation	Context	Transaction ID	LogBlockGeneration	Tag Bits	Log Record Fixed Length	Log Record Length	Previous
583	00000049:00000116:0006	LOP_MODIFY_ROW	LCX_HEAP	0000:000007f9	0	0x0000	62	100	000000
584	00000049:00000116:0007	LOP_MODIFY_ROW	LCX_HEAP	0000:000007f9	0	0x0000	62	76	000000
585	00000049:00000116:0008	LOP_ABORT_XACT	LCX_NULL	0000:000007f9	0	0x0000	48	52	000000
586	00000049:00000116:0009	LOP_BEGIN_XACT	LCX_NULL	0000:000007fa	0	0x0000	64	132	000000
587	00000049:00000116:000a	LOP_MODIFY_ROW	LCX_HEAP	0000:000007fa	0	0x0000	62	100	000000
588	00000049:00000116:000b	LOP_MODIFY_ROW	LCX_HEAP	0000:000007fa	0	0x0000	62	76	000000
589	00000049:00000116:000c	LOP_ABORT_XACT	LCX_NULL	0000:000007fa	0	0x0000	48	52	000000
590	00000049:00000116:000d	LOP_BEGIN_XACT	LCX_NULL	0000:000007fb	0	0x0000	64	132	000000
591	00000049:00000116:000e	LOP_MODIFY_ROW	LCX_HEAP	0000:000007fb	0	0x0000	62	100	000000
592	00000049:00000116:000f	LOP_MODIFY_ROW	LCX_HEAP	0000:000007fb	0	0x0000	62	76	000000
593	00000049:00000116:0010	LOP_ABORT_XACT	LCX_NULL	0000:000007fb	0	0x0000	48	52	000000
594	00000049:00000116:0011	LOP_BEGIN_XACT	LCX_NULL	0000:000007fc	0	0x0000	64	132	000000
595	00000049:00000116:0012	LOP_MODIFY_ROW	LCX_HEAP	0000:000007fc	0	0x0000	62	100	000000
596	00000049:00000116:0013	LOP_COMMIT_XACT	LCX_NULL	0000:000007fc	0	0x0000	48	52	000000
597	00000049:0000011a:0001	LOP_BEGIN_XACT	LCX_NULL	0000:000007fd	0	0x0000	64	132	000000
598	00000049:0000011a:0002	LOP_MODIFY_ROW	LCX_HEAP	0000:000007fd	0	0x0000	62	100	000000
599	00000049:0000011a:0003	LOP_COMMIT_XACT	LCX_NULL	0000:000007fd	0	0x0000	48	52	000000