

## Re: controlling lock order in transactions

**Source:**

<http://www.tech-archive.net/Archive/SQL-Server/microsoft.public.sqlserver.programming/2005-01/4076.html>

---

**From:** Andrew J. Kelly ([sqlmvpnoospam\\_at\\_shadhawk.com](mailto:sqlmvpnoospam_at_shadhawk.com))

**Date:** 01/20/05

Date: Thu, 20 Jan 2005 15:49:56 -0500

>(flagrant99) What's the differene If I check @@ERROR immediately or use  
>temp  
>var and then check that immediately after the INSERT? Is it that it will  
>prevent me from getting an @@ERROR from another session or something?

You had a block of code that looked like this:

```
IF (@@ERROR <> 0)
BEGIN
ROLLBACK TRANSACTION;
RETURN(@@ERROR)
END
```

The rollback command would reset @@ERROR (assuming it did not have an error) and the return value would be 0 even though there was an error above that to cause this block of code to execute in the first place.

>>>>

I am using sql ADO.NET driver. I performed a DBCC USEROPTIONS command and no isolation level appeared in the options. So I set it to READ UNCOMMITTED and it now does appear under DBCC USEROPTIONS.

>>>>

You can see the user options in profiler if you trace the Existing connections event. I believe the default isolation level for .net may be Serializable and not read committed? Have a look here:

<http://msdn.microsoft.com/msdnmag/issues/05/02/DataPoints/default.aspx>

>>>

Questions:

1. If I set isolation level in my stored procedure will it affect client stored procedure running in another process performing SELECT?

>>>

If you set the isolation level via the connection it only affects that connection.

>>

2. My transaction ONLY performs INSERTS. I thought for INSERTS Isolation Level doesn't make a difference? I thought it was always Exclusive lock.

>>

Yes it will put an exclusive lock on the row being inserted but Serializable puts range locks as well.

>>

3. If I require Exclusive lock to perform my INSERTS how does Isolation level even make a difference for the client? I mean If I need Exclusive lock to perform INSERT they will not be able to get any kind of lock right on my table anyway right? So If client does use READ COMMITED we will still have dead lock? right?

The only isolation level I can see removing the dead locks would be on the client side performing the select if they perform READ UNCOMMITTED so they don't get any lock at all. But then we have possibility for dirty reads.

>>

You don't need to specify anything special when you do an insert, update or delete if your isolation level is Read Committed. SQL Server decides what and how to do the locking and you should not get in it's way. The client doesn't really know anything about the isolation level and does not do anything differently per say. All the work is done on the server.

>>>

4. Is there any way to control lock order in a select JOIN. If I could force them to go in the same lock order this problem wouldn't happen.

<<<

Well there is a hint for the select called FORCE ORDER (See BOL for details) that may change the behavior but I would go there only as a last resort. I would ensure you are in Read Committed mode for both the inserts and the selects first.

>>>

Something like this a few times in a stored procedure :

```
BEGIN TRANSACTION
INSERT INTO SOME_TEMP_TABLE
SELECT DISTINCT TABLE3.SeqNum UniqueID, TABLE1.EventTime RecDateTime, @Arg1
FROM
TABLE3, TABLE1, SOME_OTHER_TABLE
WHERE
TABLE3.SeqNum > @LastSeqNum AND

TABLE1.EventTime <= @EndDate and
TABLE1.EventID > 0 and
TABLE3.SeqNum = TABLE1.SeqNum AND
SOME_OTHER_TABLE.SystemName = @Arg2 AND
SOME_OTHER_TABLE.SystemID = TABLE1.SystemID
```

## COMMIT TRANSACTION

How many rows is it returning and is it well indexed?  
1000's of rows. Most columns that they query against are indexed.  
<<<<

Just because the columns have indexes on them in no way guarantees they will be used. Especially if you are returning thousands of rows and have where clauses like that. Have you looked at the query plan to see if it is using seeks or scans? My guess is that at least some of the indexes are being scanned (either partially or fully) instead of seeks. That makes a big difference in the locking issues, especially if you are in Serializable isolation level.

Why are you wrapping the select in a transaction in the first place? SELECT's by them selves do not need an explicit transaction. Why the DISTINCT as well? Unless the table is denormalized there should be no need for a Distinct in a query such as that. What is the Some\_Temp\_Table? Is it a #table or a real table? Have you tried using a table variable for that?

I think you have two main issues. One is I really think you are in Serializable level when you should be using Read Committed. The other is that I doubt very much that the Join is really optimized as fully as it can be.

```
--
Andrew J. Kelly  SQL MVP
"flagrant99" <flagrant99@discussions.microsoft.com> wrote in message
news:F7DD2354-88BA-4376-8777-929A677AC9D3@microsoft.com...
> Thanks for the tips
>
> "Andrew J. Kelly" wrote:
>
>> A couple of comments on this. First if you are using SQL2000 then you
>> should use SCOPE_IDENTITY instead of @@Identity.
>
> (flagrant99) Agreed. I will change this.
>
> Next the way you are doing
>> the error checking is not correct. As soon as you do any tsql commands
>> the
>> value of @@ERROR is reset to be the error code of that last statement.
>> You
>> should declare a var and set it then use that later if you need to.
>>
>> DECLARE @Error INT
>>
>> INSERT xxx
>>
>> SET @Error = @@ERROR
>>
>> IF @Error <> 0
>> ....
>
> (flagrant99) What's the differene If I check @@ERROR immediately or use
> temp
> var and then check that immediately after the INSERT? Is it that it will
```

## microsoft.public.sqlserver.programming: Re: controlling lock order in transactions

```
> prevent me from getting an @@ERROR from another session or something?
>
>
>>
>> You should get in the habit of checking the level of @@TRANCOUNT before
>> issuing a COMMIT or ROLLBACK. If you inserted into a table with a trigger
>> and the trigger issued a rollback your commit or rollback would error.
>>
>> IF @@TRANCOUNT > 0
>>     COMMIT TRAN
>
> (flagrant99) Agreed. I will change this.
>
>>
>> If you don't know what isolation level you are using then you don't know
>> if
>> it is serializable or not. The default for SQL Server is read committed
>> but
>> many of the drivers will use Serializable as their default. You can run
>> a
>> trace to see what it is.
>
> I am using sql ADO.NET driver. I performed a DBCC USEROPTIONS command and
> no
> isolation level appeared in the options. So I set it to READ UNCOMMITTED
> and
> it now does appear under DBCC USEROPTIONS.
>
> Questions:
> 1. If I set isolation level in my stored procedure will it affect client
> stored procedure running in another process performing SELECT?
>
> 2. My transaction ONLY performs INSERTS. I thought for INSERTS Isolation
> Level doesn't make a difference? I thought it was always Exclusive lock.
>
> 3. If I require Exclusive lock to perform my INSERTS how does Isolation
> level even make a difference for the client? I mean If I need Exclusive
> lock
> to perform INSERT they will not be able to get any kind of lock right on
> my
> table anyway right? So If client does use READ COMMITED we will still have
> dead lock? right?
>
> The only isolation level I can see removing the dead locks would be on the
> client side performing the select if the perform READ UNCOMMITTED so they
> don't get any lock at all. But then we have possiblity for dirty reads.
>
> 4. Is there any way to control lock order in a select JOIN. If I could
> force
> them to go in the same lock order this problem wouldn't happen.
>
>
>
> What exactly does the select with the join look
>> like?
>
> Something like this a few times in a stored procedure :
>
> BEGIN TRANSACTION
> INSERT INTO SOME_TEMP_TABLE
> SELECT DISTINCT TABLE3.SeqNum UniqueID, TABLE1.EventTime RecDateTime,
> @Arg1
```

## microsoft.public.sqlserver.programming: Re: controlling lock order in transactions

```
> FROM
> TABLE3, TABLE1, SOME_OTHER_TABLE
> WHERE
> TABLE3.SeqNum > @LastSeqNum AND
>
> TABLE1.EventTime <= @EndDate and
> TABLE1.EventID > 0 and
> TABLE3.SeqNum = TABLE1.SeqNum AND
> SOME_OTHER_TABLE.SystemName = @Arg2 AND
> SOME_OTHER_TABLE.SystemID = TABLE1.SystemID
>
> COMMIT TRANSACTION
>
> How many rows is it returning and is it well indexed?
> 1000's of rows. Most columns that they query against are indexed.
>
> Thanks for all of your feedback :)
>
>
> Have you
>> looked to see what type of locks these two processes generate? One more
>> comment. If you do this hundred's of times a second I would consider
>> batching the inserts if at all possible.
>>
>> --
>> Andrew J. Kelly SQL MVP
>>
>>
>> "flagrant99" <flagrant99@discussions.microsoft.com> wrote in message
>> news:ABE67713-0F16-4907-AAFC-B7DA4E953088@microsoft.com...
>> > Insert Code goes something like this:
>> >
>> > BEGIN TRANSACTION
>> > DECLARE @MySeqNum int
>> >
>> > --//The 1st Field in TABLE1 is actually a unique, index id
>> > autogenerated
>> > by
>> > SQL
>> >
>> > INSERT INTO [TABLE1]
>> > (
>> > [Field2],
>> > [Field3],
>> > [etc]
>> > VALUES
>> > (
>> > GETUTCDATE(),
>> > @Arg1,
>> > @etc
>> > )
>> >
>> >
>> > IF (@@ERROR <> 0)
>> > BEGIN
>> > ROLLBACK TRANSACTION;
>> > RETURN(@@ERROR)
>> > END
>> > SET @MySeqNum = @@Identity
>> >
>> > INSERT INTO [TABLE2]
>> > (
```

## microsoft.public.sqlserver.programming: Re: controlling lock order in transactions

```
>> > [SeqNum],
>> > [Field2],
>> > [Field3],
>> > )
>> >
>> > VALUES
>> > (
>> > @MySeqNum,
>> > @Arg2,
>> > @etc
>> > )
>> >
>> > IF (@@ERROR <> 0)
>> > BEGIN
>> > ROLLBACK TRANSACTION;
>> > RETURN(@@ERROR)
>> > END
>> >
>> >
>> > INSERT INTO TABLE3
>> > ( [ASequenceNum], [BSequenceNum])
>> >
>> > VALUES
>> > ( 0, @MySeqNum)
>> >
>> > IF (@@ERROR = 0)
>> > COMMIT TRANSACTION;
>> > ELSE
>> > ROLLBACK TRANSACTION;
>> >
>> > The other client does a Join on TABLE1 and TABLE 3
>> > We never set any specific isolation level so I am assuming it is
>> > default.
>> > Many of the columns are indexed... How do indexes affect deadlocks?
>> >
>> > From my reading INSERTS always take out an exclusive lock that is held
>> > (once
>> > taken) for the duration of the transaction. The only possible thing I
>> > can
>> > find in the clients sp that could be causing the dead lock is the join
>> > on
>> > TABLE1 and TABLE3. I am guessing for some reason SQL is getting the
>> > lock
>> > on
>> > TABLE3 1st, which the transaction above requires at the end of the
>> > transaction.
>> >
>> > The above transaction is run 100's of time per second. The Select join
>> > on
>> > TABLE1 and TABLE3 occurs every five minutes. Dead Locks occur randomly,
>> > mabe
>> > once every 1/2 hour or so.
>> >
>> >
>> >
>> > "Andrew J. Kelly" wrote:
>> >
>> >> It would help to see the actual code and maybe even the DDL for the
>> >> tables
```

## microsoft.public.sqlserver.programming: Re: controlling lock order in transactions

```
>> >> involved. But it sounds like either you are using the Serializable
>> >> isolation level or you do not have proper indexes on the tables
>> >> involved.
>> >>
>> >> --
>> >> Andrew J. Kelly SQL MVP
>> >>
>> >>
>> >> "flagrant99" <flagrant99@discussions.microsoft.com> wrote in message
>> >> news:A0D49F72-7B57-4EB7-B0F2-CFE8250A24B1@microsoft.com...
>> >> >I have the following scenario which causes deadlocks in stored
>> >> >procedures:
>> >> >
>> >> > BEGIN TRANSACTION
>> >> > Insert to Table 1
>> >> > Used Id From Table1 to Insert to Table 2
>> >> > COMMIT
>> >> >
>> >> > An Insert gets an Exclusive Lock which is held for the whole
>> >> > transaction.
>> >> >
>> >> > Another client needs access to this data and performs Join on Both
>> >> > Tables
>> >> > So it gets the Shared read lock On Table 2 and when it tries to get
>> >> > the
>> >> > lock
>> >> > in the join on Table 1 we have a dead lock.
>> >> >
>> >> > Is there any way to force SQL to get the lock on table 1 first? To
>> >> > prevent
>> >> > deadlock?
>> >> > Will changing orders in the FROM clause cause locks to be taken in
>> >> > different
>> >> > orders?
>> >> >
>> >> > TIA.
>> >> >
>> >> >
>> >> >
>> >>
>> >>
>> >>
>>
>>
>>
```