

Re: controlling lock order in transactions

Source:

<http://www.tech-archive.net/Archive/SQL-Server/microsoft.public.sqlserver.programming/2005-01/3932.html>

From: Andrew J. Kelly (*sqlmvpnoospam_at_shadhawk.com*)

Date: 01/20/05

Date: Thu, 20 Jan 2005 09:42:41 -0500

A couple of comments on this. First if you are using SQL2000 then you should use SCOPE_IDENTITY instead of @@Identity. 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
```

```
...
```

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
```

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. What exactly does the select with the join look like? How many rows is it returning and is it well indexed? 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
```

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

```
>
> --//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]
> (
> [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?
>
```

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

```
> 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
>> 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.
>> >
>> >
>> >
>> >
>> >
>> >
```