**An Experiment with Deadlocks**

by [Jason Brimhall](http://jasonbrimhall.info/author/admin/)

FROM: <http://jasonbrimhall.info/2019/03/25/an-experiment-with-deadlocks/>

Categories: [Extended Events](http://jasonbrimhall.info/category/xevents/), [News](http://jasonbrimhall.info/category/news/), [Professional](http://jasonbrimhall.info/), [SQLServerPedia Syndication](http://jasonbrimhall.info/category/sqlserverpedia-syndication/), [SSC](http://jasonbrimhall.info/category/ssc/)

Tags: [Extended Events](http://jasonbrimhall.info/tag/extended-events/), [SQL Administration](http://jasonbrimhall.info/tag/sql-administration/), [SQL Internals](http://jasonbrimhall.info/tag/sql-internals/), [SQL Server](http://jasonbrimhall.info/tag/sql-server/), [XE](http://jasonbrimhall.info/tag/xe/), [XEvents](http://jasonbrimhall.info/tag/xevents/)

Comments: [No Comments](http://jasonbrimhall.info/2019/03/25/an-experiment-with-deadlocks/#comments)

Published on: **March 25, 2019**

Everything can be fixed with a query hint (\*cough\* directive), right? If a certain process is consistently causing deadlocks, a simple ROWLOCK hint can be added to prevent it, right?

[](http://bit.ly/XE60Days)Well, for whatever reason, there seems to be a myth out there that when deadlocks come a-knocking, then just throw this little directive at it and all will be well. Today, we get to test that and show what will continue to happen.

First, lets look at what the ROWLOCK actually means:

Specifies that row locks are taken when page or table locks are ordinarily taken.

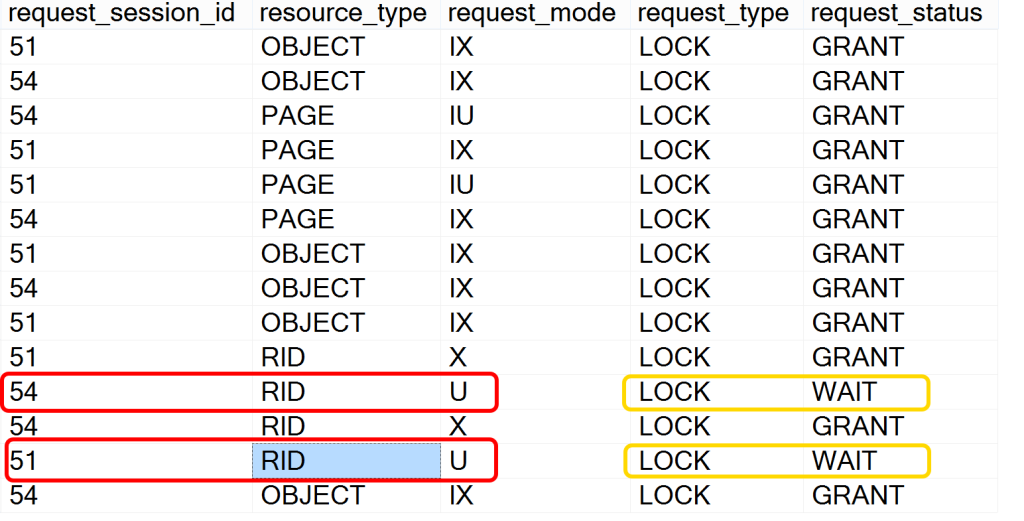
This seems like a fabulous idea if the deadlocks are occurring against a table involving a massive update. Let’s take a look at a small update scenario involving just a handful of records. For the setup, we will use the same setup used in a [previous article by Wayne](https://blog.waynesheffield.com/wayne/archive/2017/05/weaning-yourself-off-of-sql-profiler-part-2/).

Transact-SQL



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | USE tempdb;  GO  IF OBJECT\_ID('dbo.Test1') IS NOT NULL  DROP TABLE dbo.Test1;  IF OBJECT\_ID('dbo.Test2') IS NOT NULL  DROP TABLE dbo.Test2;  CREATE TABLE dbo.Test1 ( col1 INT );  CREATE TABLE dbo.Test2 ( col2 INT );  INSERT INTO dbo.Test1  VALUES ( 1 ),  ( 2 ),  ( 3 ),  ( 4 ),  ( 5 );  INSERT INTO dbo.Test2  VALUES ( 1 ),  ( 2 ),  ( 3 ),  ( 4 ),  ( 5 );  GO  BEGIN TRANSACTION;  UPDATE dbo.Test1  SET col1 = col1 \* 10  WHERE col1 = 3;    /\* query window 2\*/    USE tempdb;  BEGIN TRANSACTION;  UPDATE dbo.Test2  SET col2 = col2 \* 20  WHERE col2 = 4;  UPDATE dbo.Test1  SET col1 = col1 \* 20  WHERE col1 = 3;  COMMIT TRANSACTION;    /\* back to the first query window\*/  UPDATE dbo.Test2  SET col2 = col2 \* 10  WHERE col2 = 4;  COMMIT TRANSACTION; |

Looking at the code, we can see there are only five rows in each of the tables. In addition, an update will be performed to both col1 and col2 in each table for one specific row. So we are keeping this to a singleton type of update, and we are able to force a deadlock by using this setup. Not only do we see that a deadlock will occur consistently, we would see the following in the sys.dm\_tran\_locks DMV as well as in the deadlock graphs.

[](http://bit.ly/1Qytj9F)

In my environment, I used session 51 and 54 consistently for the deadlock repro. In each of the tests, each spid did obtain page locks as well as RID locks (row locks) that were waiting on each other in order to perform an Update. This is what we saw when I ran the setup without the ROWLOCK directive. What if I used the ROWLOCK directive (are you guessing already that there would be no change because the deadlock occurs on the lock held for the update that is waiting on the other update to complete?)? Let’s take a look at that too!

Transact-SQL



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | /\* what about using rowlock hint? let's try it \*/  USE tempdb;  GO  IF OBJECT\_ID('dbo.Test1') IS NOT NULL  DROP TABLE dbo.Test1;  IF OBJECT\_ID('dbo.Test2') IS NOT NULL  DROP TABLE dbo.Test2;  CREATE TABLE dbo.Test1 ( col1 INT );  CREATE TABLE dbo.Test2 ( col2 INT );  INSERT INTO dbo.Test1  VALUES ( 1 ),  ( 2 ),  ( 3 ),  ( 4 ),  ( 5 );  INSERT INTO dbo.Test2  VALUES ( 1 ),  ( 2 ),  ( 3 ),  ( 4 ),  ( 5 );  GO  BEGIN TRANSACTION;  UPDATE dbo.Test1 WITH (ROWLOCK)  SET col1 = col1 \* 10  WHERE col1 = 3;    /\* query window 2\*/    USE tempdb;  BEGIN TRANSACTION;  UPDATE dbo.Test2 WITH (ROWLOCK)  SET col2 = col2 \* 20  WHERE col2 = 4;  UPDATE dbo.Test1 WITH (ROWLOCK)  SET col1 = col1 \* 20  WHERE col1 = 3;  COMMIT TRANSACTION;    /\* back to the first query window\*/  UPDATE dbo.Test2 WITH (ROWLOCK)  SET col2 = col2 \* 10  WHERE col2 = 4;  COMMIT TRANSACTION; |

The only change to this setup is that the ROWLOCK directive has been added to the update statements. Examining the sys.dm\_tran\_locks DMV reveals the same locks being held as was seen without the directive. This shouldn’t be too big of a surprise since the updates are against a single row.

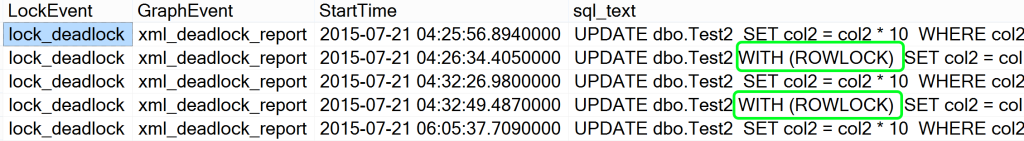
In addition to the same locks being held, we continue to experience the same deadlock problem. Using an Extended Events session to trap deadlock information (similar to the article previously discussed), we can pull out some pretty good info. Let’s examine some of the deadlock data trapped by an XE session.

Transact-SQL

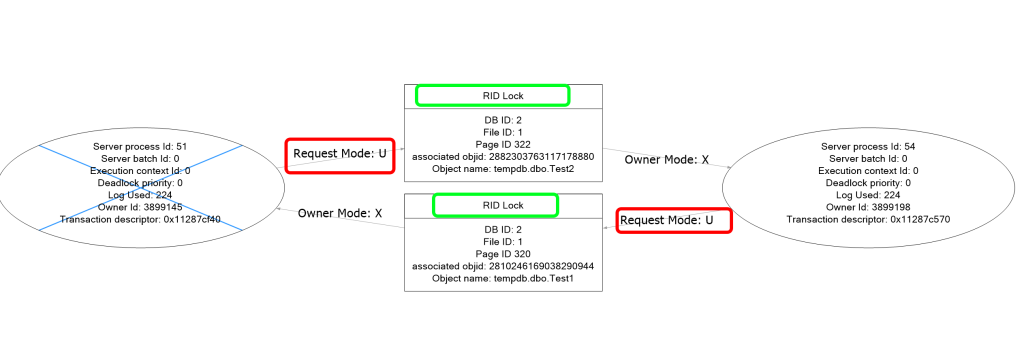


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68 | WITH cte AS ( SELECT t2.event\_data.value('(event/@name)[1]', 'varchar(50)') AS event\_name  , t2.event\_data.value('(event/@timestamp)[1]',  'datetime2') AS StartTime  , t2.event\_data.value('(event/data[@name="duration"]/value)[1]',  'bigint') AS duration  , t2.event\_data.value('(event/data[@name="database\_name"]/value)[1]',  'sysname') AS DBName  , t2.event\_data.value('(event/action[@name="sql\_text"]/value)[1]',  'varchar(max)') AS sql\_text  , t2.event\_data.value('(event/action[@name="nt\_username"]/value)[1]',  'varchar(500)') AS nt\_username  , t2.event\_data.value('(event/data[@name="mode"]/value)[1]',  'varchar(15)') + ' ('  + t2.event\_data.value('(event/data[@name="mode"]/text)[1]',  'varchar(50)') + ')' AS mode  , t2.event\_data.value('(event/data[@name="object\_id"]/value)[1]',  'integer') AS object\_id  , t2.event\_data.value('(event/data[@name="resource\_description"]/value)[1]',  'varchar(max)') AS resource\_description  , t2.event\_data.value('(event/data[@name="resource\_owner\_type"]/text)[1]',  'varchar(max)') AS resource\_owner\_type  , t2.event\_data.value('(event/data[@name="resource\_type"]/text)[1]',  'varchar(max)') + ' ('  + t2.event\_data.value('(event/data[@name="resource\_type"]/value)[1]',  'varchar(max)') + ')' AS resource\_type  , t2.event\_data.value('(event/action[@name="server\_principal\_name"]/value)[1]',  'varchar(max)') AS server\_principal\_name  , t2.event\_data.value('(event/action[@name="session\_id"]/value)[1]',  'varchar(max)') AS session\_id  , t2.event\_data.value('(event/action[@name="client\_pid"]/value)[1]',  'integer') AS client\_pid  , t2.event\_data.value('(event/action[@name="client\_app\_name"]/value)[1]',  'varchar(max)') AS client\_app\_name  , t2.event\_data  ,ROW\_NUMBER() OVER (ORDER BY t2.event\_data.value('(event/@timestamp)[1]',  'datetime2')  ,t2.event\_data.value('(event/@name)[1]', 'varchar(50)')) AS Event\_Seq  FROM sys.fn\_xe\_file\_target\_read\_file('C:\Database\XE\Deadlock\*.xel',  NULL, NULL, NULL) t1  CROSS APPLY ( SELECT CONVERT(XML, t1.event\_data)  ) t2 ( event\_data )  )  SELECT c1.event\_name AS LockEvent, c2.event\_name AS GraphEvent  , c1.StartTime  , DATEADD(MICROSECOND, c1.duration, CONVERT(DATETIME2, [c1].StartTime)) AS EndDate  , c1.duration  , c1.DBName  , c1.sql\_text  , c1.nt\_username  , c1.server\_principal\_name  , c1.mode  , c1.object\_id  , c1.resource\_description  , c1.resource\_owner\_type  , c1.resource\_type  , c1.session\_id  , c1.client\_pid  , c1.client\_app\_name  , c1.event\_data  , graph.query('.') AS EventDeadlockGraph  FROM cte c1  INNER JOIN cte c2  ON c1.Event\_Seq = c2.Event\_Seq - 1  CROSS APPLY c2.event\_data.nodes('(event/data[@name="xml\_report"]/value)[last()]/\*')  AS deadlock ( graph )  WHERE c1.event\_name = 'lock\_deadlock'  AND c2.event\_name = 'xml\_deadlock\_report'  ORDER BY c1.StartTime; |

The results from this query will show us the deadlock graph, the event data, as well as several other pieces of data already parsed from the session data for you. And looking at the session data, one can see that the sql\_text from each of the queries will demonstrate both the ROWLOCK directive and the directive-free versions of the query. In this query you can also see that I did a little black magic to match up the two event types from the event session (lock\_deadlock and xml\_deadlock\_report). Then I was able to join the two together to produce one row per deadlock event and to see the sql\_text with the deadlock graph on one row. Otherwise, the sql\_text does not produce with the deadlock\_report event. I leave the rest of the query to the reader to discover and question.



From the EventDeadlockGraph column, we could click the cell and take a close look at the XML generated for the deadlock event. Further, if I choose to save the xml as an XDL file and then reopen it in SSMS, I can see the deadlock graphical report as shown in the following.



We see that row locks are still in effect for the update coming from both sides. This further supports that the directive really is just a waste of time in trying to combat this type of deadlock. This is one of those cases where the best option would be to optimize the code and work things out without trying to take a shortcut.

**Wrapping Up**

Look to optimize the code instead of trying to take a shortcut. In addition, take a look at the deadlocks, the locks held, and the code to get a better understanding of what is truly happening.

This article demonstrates briefly the power of Extended Events while diving into deadlocks. For more on using Extended Events, [start reading here](http://jasonbrimhall.info/2015/09/08/learning-extended-events-in-60-days/)! This [article](http://jasonbrimhall.info/2012/12/21/on-the-ninth-day/) may also be of interest.

**Related Posts:**

* [Events By Feature](http://jasonbrimhall.info/2016/08/30/events-by-feature/) August 30, 2016
* [Maintenance Plan Owner – Back to Basics](http://jasonbrimhall.info/2018/12/13/maintenance-plan-owner-back-to-basics/) December 13, 2018
* [Audit Database Offline Events](http://jasonbrimhall.info/2018/11/13/audit-database-offline-events/) November 13, 2018
* [An Introduction to Templates](http://jasonbrimhall.info/2017/12/30/an-introduction-to-templates/) December 30, 2017
* [Ghosts in your Database](http://jasonbrimhall.info/2018/11/26/ghosts-in-your-database/) November 26, 2018

No Comments - [Leave a comment](http://jasonbrimhall.info/2019/03/25/an-experiment-with-deadlocks/#respond)

This site uses Akismet to reduce spam. [Learn how your comment data is processed](https://akismet.com/privacy/).