

Blocking and Locking Troubleshooting

By Pinal Dave

For

Embarcadero Technologies

August 2014



INTRODUCTION

How many times in your life have you had to stand in a long queue to get work done? Be it the railway station, waiting for a bus or anything else? Why do we all do this? There is a process and we need to follow the rules to get things done. If you look at it closely, there is a method to the madness and to avoid any chaos we need to follow these rules. If I had to compare this to SQL Server world there are tons of similarities. To maintain orderliness one has to stand in a queue, the longer the person in front takes to complete their task, the longer we will be waiting in the queue. If this were a railway station ticketing counter, then if the person takes a long time in the counter, the ticket master is unavailable to issue tickets for others. So where is the similarity? Well, waiting in queue while the previous person finishes their task is a typical blocking behavior and the state where the ticket master is not able to issue others ticket is a classic locking of resource problem. Hope you get the drift and this whitepaper is all about Blocking and Locking inside SQL Server.

ACID INTRO

Let me take you through the same from the basics. Blocking and Locking is inevitable in traditional relational databases. Like in our example above, it is a process and to ensure ACID properties of transactions one needs to have them.

Atomicity: Data modifications in a transaction is all-in or none behaviour.

Consistency: Once transaction is committed, data must be in consistent state i.e. data integrity needs to be met.

Isolation: This is isolating and protecting concurrent transactions in modifying data that have been changed by another concurrent transaction.

Durability: As the name suggests, once transaction completes the modifications are permanent and persisted even in event of system failures.

Now that we are aware of the basics, SQL Server uses locks on data to prevent data corruption and stop multiple users updating the same record at the same time.

LOCKING TYPE BASICS

Let me take a moment to talk only about the most commonly used Locking types in this section. Though this is a complex and heavy topic, it is worth to get a primer in this section. There are a number of lock types, but the most common ones are:

Shared: This is used to allow concurrent transactions to read source data. This lock type is released as soon as the data has been read, unless the Isolation level is repeatable read or higher.

Exclusive: This lock is used to make sure no other transactions can read or modify data locked with an exclusive (X) lock.

Intent: This lock type indicated that SQL Server wants to acquire a shared (S) lock or exclusive (X) lock on some of the resources lower down in the transaction process.

Update: This lock is used to prevent deadlock as exclusive is not used until modification is made. Let me explain you in detail. A typical update would acquire a Shared (S) lock on the resources and then modifying would require the locks to be converted to exclusive (X) locks. If two transactions try to perform an update data while one data tries to convert into exclusive lock, this transaction needs to wait as the shared lock from other transaction and this conversion to exclusive lock are not compatible. If the second transaction also tries to convert into Exclusive Lock, then this transaction is waiting for connection one to release its lock. This is a typical scenario of cyclic deadlock which we will explain in detail later. To avoid this scenario, Update locks are used by SQL Server because only one transaction can hold an Update lock on the same resource at a point in time. In the below table, have outlined few more for reference.

Lock Mode	Description
Schema-Stability (Sch-S)	Acquired when compiling queries
Schema Modification (Sch-M)	Acquired for DDL operations like ALTER or DROP on a table schema
Shared (S)	Acquired for reading data
Update (U)	Acquired before modification can get converted to exclusive
Exclusive (X)	Acquired for writing
Intent Shared (IS)	Requests for shared lock(s)
Intent Update (IU)	Requests for update lock(s)
Intent Exclusive (IX)	Requests for exclusive lock(s)
Bulk Update (BU)	Used when we do bulk copy operations into a table

Locks on a resource can be taken for a short or a long duration. Short locks are released before the transaction completes. These are like Shared lock in Read Committed Isolation wherein the lock is released as soon as the transaction completes. Long Locks are those where the locks are released only when the transaction completes. Typically these are like Exclusive locks taken for insert, update or delete of rows. It is these uncommitted transactions that hold onto locks and cause possible blocking behavior for other connections.

Note: Locks can also be held during sorting or hashing of rows as the query is waiting for memory resources.

Yet another reason can be because of IO intensive queries. I wrote a complete series on wait stats and refrain from expanding them here again. To understand the same, check [PAGEIOLATCH_SH](#) and [ASYNC_NETWORK_IO](#) for more details. As you can see, locks can be held for multiple reasons.

Some of the salient points to note is, locks are managed on a per connection basis.

If we are talking about Locking so much, what is blocking then? They seem to be a symptom of locking isn't it? Let us briefly understand the same.

BLOCKING BEHAVIOURS

Blocking is a scenario where two connections are fighting over an incompatibility lock on a resource, i.e., table, row, page, ranges of keys, indexes or database. This is a first come first serve basis scenario. The first connection that makes a request for a lock is granted access to the resource while all the subsequent requests are now blocked and cannot continue processing until the first connections locks are released. By default, there is no mandatory time-out period and no way to test if a resource is locked before locking it, except to attempt to access the data (and potentially get blocked indefinitely). Blocking as you can see is inevitable and is needed for data integrity (from ACID it is consistency and isolation). It is surely an extension to the Locking basics we discussed before in this whitepaper.

UNDERSTANDING BLOCKING AND WAITS

Since we are talking about waits at the row / page level, this information can be got from the DMV - `sys.dm_db_index_operational_stats` using the following query. The query finds the blocking and the wait times for the given database.

```
-- Calculate the blocking rates and wait times
SELECT SUM(row_lock_count) row_locks,
SUM(row_lock_wait_count) row_lock_waits,
SUM(row_lock_wait_in_ms) row_lock_wait_time_ms,
SUM(page_lock_count) page_locks,
SUM(page_lock_wait_count) page_lock_waits,
SUM(page_lock_wait_in_ms) page_lock_wait_time_ms
FROM sys.dm_db_index_operational_stats(DB_ID(),null,null,null)
```

These DMVs are so powerful that we can get tons of other information to how the access pattern has been on these resources. For example, we can be interested in understanding the number of inserts, updates and deletes happening on the database. This can be also queried from the same DMV using a query as shown below:


```

SELECT DB_NAME() as DB_NAME, obj.name as table_name,
ind.name as index_name, ind.type_desc,
leaf_allocation_count+nonleaf_allocation_count as splits,
range_scan_count, singleton_lookup_count,
leaf_insert_count+nonleaf_insert_count as inserts,
leaf_update_count+nonleaf_update_count as updates,
leaf_delete_count+nonleaf_delete_count as deletes
FROM sys.dm_db_index_operational_stats(DB_ID(),null,null,null) as os
INNER JOIN sys.indexes as ind
ON ind.object_id = os.object_id and ind.index_id = os.index_id
INNER JOIN sys.objects as obj
ON obj.object_id = os.object_id
WHERE obj.Type NOT LIKE 'S'

```

Let us query `sys.dm_exec_query_stats` and `sys.dm_exec_sql_text` to identify the top 20 blocked queries and their wait times using the below query.

```

SELECT TOP(20) sql_text.text AS "Batch text",
CASE qs.statement_end_offset
    WHEN -1
    THEN SUBSTRING(sql_text.text, qs.statement_start_offset/2, 64000)
    ELSE SUBSTRING(sql_text.text, qs.statement_start_offset/2,
        (qs.statement_end_offset-qs.statement_start_offset)/2)
    END AS "Statement text",
execution_count, total_elapsed_time, total_worker_time,
(total_elapsed_time - total_worker_time) AS total_wait_time
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(sql_handle) AS sql_text
ORDER BY (total_elapsed_time - total_worker_time) DESC

```

A typical output looks like below.

	Batch text	Statement text	execution_count	total_elapsed_time	total_worker_time	total_wait_time
1	SELECT * FROM HumanResources....	SELECT * FROM Sales.SalesOrderData	4	12826225	798645	12027580
2	SELECT * FROM HumanResources....	SELECT * FROM Production.WorkOrderRouti	4	8947060	239713	8707347
3	SELECT * FROM HumanResources....	SELECT * FROM Production.TransactionHisto	4	8609484	308599	8300885
4	SELECT * FROM HumanResources....	SELECT * FROM Production.WorkOrd	4	7046589	250197	6796392
5	SELECT * FROM HumanResources....	SELECT * FROM Sales.SalesOrderHead	4	6999102	359077	6640025

KNOWING YOUR DEADLOCKS

Before we can get into a typical blocking behavior and troubleshooting, let us take a moment to recognize a unique behavior called as Deadlocks. A typical deadlock is a case when two connections are waiting to release resources locked by the other connection. A common form of deadlock is called as Cyclic deadlock. To outline how a typical cyclic deadlock happens, check the sequence of activity happening in two connections.

Time	Connection 1	Connection 2
T1	Begin Tran	Begin Tran
T2 Granted	Update Persons Set DOJ = '10/01/2014' Where name like 'Pinal'	
T3 Granted		Update Address Set active = 'N' Where person_name like 'Pinal'
T4 Request	Select * From Address Where person_name like 'Pinal'	
T5 Request		Select * From Persons Where name like 'Pinal'
T6	Deadlock Victim	(blocking removed)
T7		Commit

In the example, we can see that two connections are trying to take a lock on "Persons" and "Addresses" table within the same connection timespan. The sequence is made in such a way that now each of the connection is waiting for release of locks from other connection. We are sure, if you ever encountered this deadlock situation – then a typical error message is shown:

**Msg 1205, Level 13, State 56, Line 10
Transaction (Process ID 53) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.**

Though deadlocks happen, not many know that a deadlock need not happen only because of cyclic behavior as explained above. It can also occur in a single resource too. Let me show this in a simple timeline to achieve the same.

Time	Connection 1	Connection 2
T1	Begin Tran	Begin Tran
T2 Granted	Select * From Persons With (HOLDLOCK) Where name like 'Pinal'	
T3 Granted		Select * From Persons With (HOLDLOCK) Where name like 'Pinal'
T4 Blocked	Update Persons Set active = 'N' Where name like 'Pinal'	

Time	Connection 1	Connection 2
T5 Blocked		Update Persons Set active = 'Y' Where name like 'Pinal'
T6	Deadlock Victim	(blocking removed)
T7		Commit

TOOLS TO IDENTIFY BLOCKING

There are enough number of tools available with SQL Server that are out-of-box which we can use to identify blocking behavior. There is no one tool that will fit the bill for all blocking issues. We need to use the right tool for the right situation.

ACTIVITY MONITOR

One of the hidden gems inside SQL Server Management Studio is the Activity Monitor pane. The shortcut to invoke this window is "CTRL + ALT + A". From a blocking point of view, this can show us which are the processes blocked and who is blocking with the resource associated. In the below screenshot, I have shown a simple blocking behaviour.

Processes												
S...	U...	Login	Dat...	Task State	Comma...	Appl...	Wait Tim...	Wait...	Wait...	B...	H...	Memor...
52	1	sa	master			Microsoft...	0					24
53	1	sa	tempdb			Microsoft...	0				1	24
54	1	sa	tempdb	SUSPENDED	INSERT	Microsoft...	500001	LCK_M_IX	objectloc...	53		24
55	1	sa	master			Microsoft...	0					24

The way to read this is simple. Look for the value of 1 in the "Head Blocker" field, in our example it is SPID of 53. Next look for values in "Blocked By" - In our example above, the "Blocked By" column has 53, which means the SPID of 54 is being blocked by SPID of 53. Hence on current live systems it is worth to note that this is the fastest and quickest way to find blocking queries on a live system to start troubleshooting. This is very basic information to start, but more often we want a lot more information about the blocking behaviour which can be got by tons of DMVs available inside SQL Server. Next we will look at some of them.

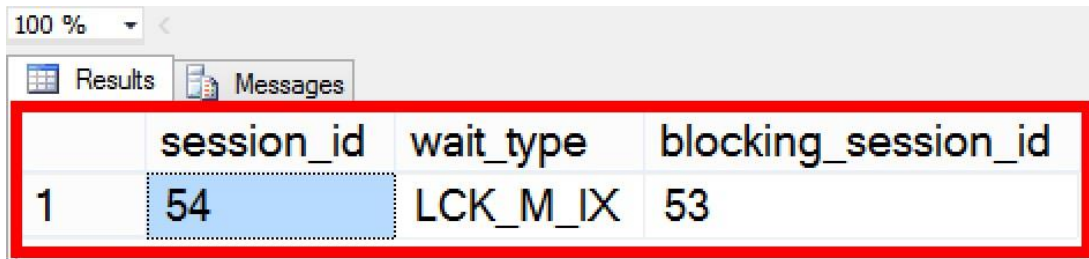
DMVs

The Dynamic Management Views (DMVs) can be defined as a set of predefined views given out-of-box by SQL Server to understand, monitor and troubleshoot activities happening inside SQL Server. With each release of SQL Server, the number of DMVs keep increasing because we have additional features to monitor. Getting back to blocking, we can easily get information about

```

USE MASTER
GO
SELECT session_id, wait_type, blocking_session_id
FROM sys.dm_os_waiting_tasks
WHERE blocking_session_id <> 0
GO

```



	session_id	wait_type	blocking_session_id
1	54	LCK_M_IX	53

A more complex query with tons of additional fields can be got from multiple DMVs. This is something I shared over my blog and thought is worth a note here for quick reference.

```

SELECT
    [Session ID]      = s.session_id,
    [Login]           = s.login_name,
    [Database]        = case when p.dbid=0 then N'' else
ISNULL(db_name(p.dbid),N'') end,
    [Task State]      = ISNULL(t.task_state, N''),
    [Command]         = ISNULL(r.command, N''),
    [Wait Time (ms)]  = ISNULL(w.wait_duration_ms, 0),
    [Wait Type]       = ISNULL(w.wait_type, N''),
    [Blocked By]      = ISNULL(CONVERT (varchar, w.blocking_session_id), ''),
    [Login Time]      = s.login_time
FROM sys.dm_exec_sessions s LEFT OUTER JOIN sys.dm_exec_connections c ON
(s.session_id = c.session_id)
LEFT OUTER JOIN sys.dm_exec_requests r ON (s.session_id = r.session_id)
LEFT OUTER JOIN sys.dm_os_tasks t ON (r.session_id = t.session_id AND
r.request_id = t.request_id)
LEFT OUTER JOIN sys.dm_os_waiting_tasks w
ON (t.task_address = w.waiting_task_address)
LEFT OUTER JOIN sys.dm_exec_requests r2 ON (s.session_id =
r2.blocking_session_id)
LEFT OUTER JOIN sys.sysprocesses p ON (s.session_id = p.spid)
WHERE s.is_user_process = 1
AND (r2.session_id IS NOT NULL
OR w.blocking_session_id IS NOT NULL)
ORDER BY s.session_id;

```

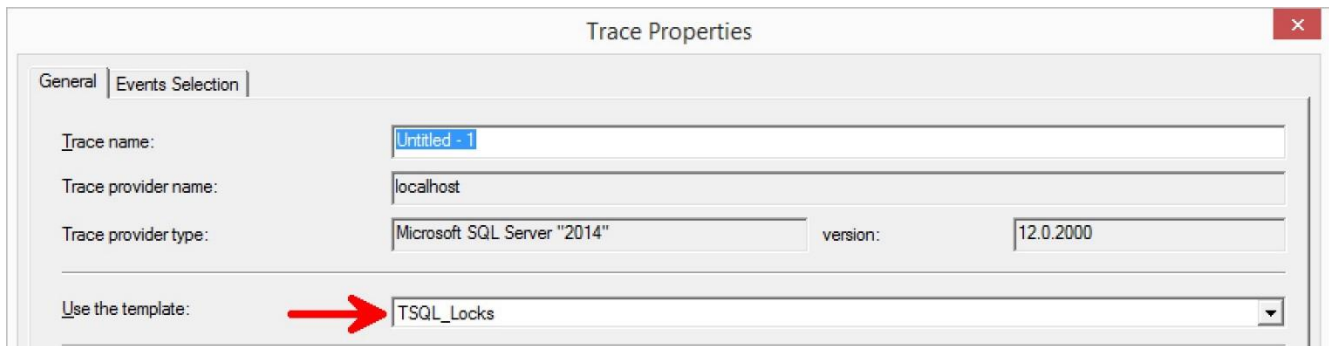
The same output is now shown with more details from multiple DMVs. This can give you a rough idea to how powerful the DMVs are inside SQL Server.

	Session ID	Login	Database	Task State	Command	Wait Time (ms)	Wait Type	Blocked By	Login Time
1	53	sa	tempdb			0			2014-10-02 10:26:27.607
2	54	sa	tempdb	SUSPENDED	INSERT	6830605	LCK_M_IX	53	2014-10-02 10:26:26.387

PROFILER

Profiler has been with SQL Server for close to 1.5 decades and it is one of the most users (DBA or Developers alike) rely on this tool heavily. As a DBA, this can be an awesome tool to troubleshoot activities happening inside SQL Server.

Profiler can also be powerful in troubleshooting blocking, deadlock, waiting and more. One of the lesser known fact is the way in which we use trace templates. One of the default templates that come with profiler is called as TSQL_Locks. As the name suggests, it gives us vital information about locks that have happened inside our SQL Server instance.



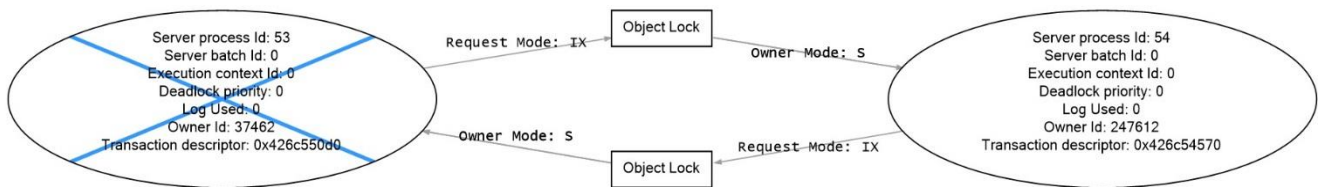
Since profiler collects specific events, the TSQL_Locks template collects the following events.

- Blocked Process Report
- SP: StmtCompleted
- SP: StmtStarting
- SQL: StmtCompleted
- SQL: StmtStarting
- Deadlock Graph
- Lock: Cancel
- Lock: Deadlock
- Lock: Deadlock Chain
- Lock: Escalation
- Lock: Timeout (timeout>0)

If you haven't used this before with your SQL Server environments, then I highly recommend in using the same and giving it a try as part of your troubleshooting. Having said that, from SQL Server 2014, profiler tool has been deprecated as we seem to move slowly but surely towards XEvents. We will discuss them later in this paper.

PROFILER – DEADLOCK GRAPHS

There are a number of events mentioned above worth a look, let me take one of the most interesting event called as “Deadlock Graph”. As the name suggests, it is a visual representation of how deadlock has occurred and what are the connections involved.



We just simulated a single resource deadlock and we can see how the Deadlock graph looks like.

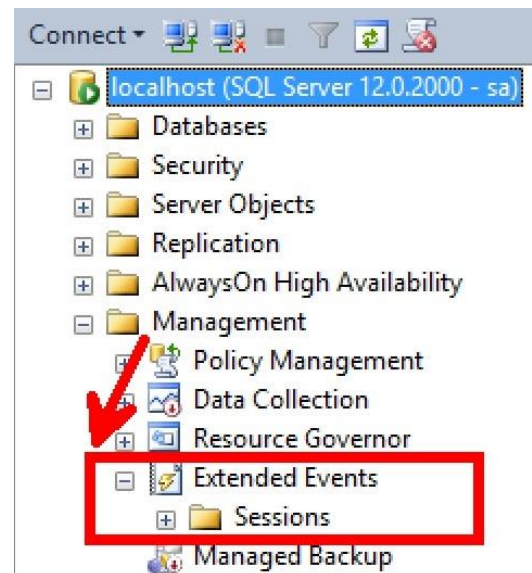
As we said, deadlocks are special case scenario of locking. It is important to mention that from SQL Server 2005 we also had another capability called as “Blocked Process Report”. This is also available in the above template.

A Blocked Process Report is invoked once we configure the same. The idea here is to have a master switch which will trigger an event once a connection is waiting for a resource for more than the threshold time interval specified in the configuration. I wrote about this in detail over my blog, so make sure to [read it there](#).

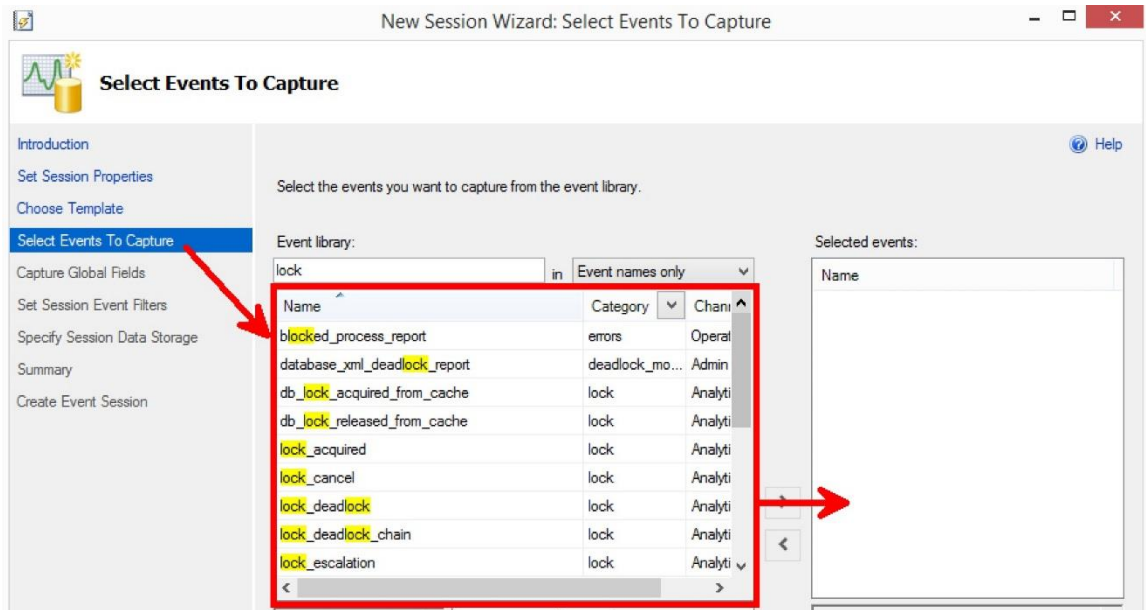
EXTENDED EVENTS

Extended Events or otherwise called as XEvents were available from SQL Server 2005 edition, but it took prominence from SQL Server 2008 R2 release for sure. From a mere TSQL syntax oriented to now with the latest release of SQL Server 2014, we have a decent UI to work with. This feature was introduced for lightweight logging and profiling – something similar to profiler but with much more capabilities. This is the main reason for profiler now being deprecated because this is the future to troubleshooting and logging inside SQL Server.

In this release of SQL Server, we can find a new node called Extended Events and we can start a new session wizard by right clicking the “Sessions” node.



The wizard is quite self-explanatory, there are tons of events to choose from and in the below example we have gone ahead and searched on “Lock” related events. As you can see, we have almost similar events as defined in Profiler. On closer look, actually there are a lot more than what Profiler can give too.



We have gone ahead and selected few events and clicked our way to close. At the finalize screen we have the option to script out the command that runs behind the scenes. The TSQL that gets generated for a sample XEvent I created is:

```
CREATE EVENT SESSION [Locks] ON SERVER
ADD EVENT sqlserver.database_xml_deadlock_report(
ACTION(sqlserver.client_pid,sqlserver.database_name,sqlserver.sql_text)),
ADD EVENT sqlserver.lock_deadlock(
ACTION(sqlserver.client_pid,sqlserver.database_name,sqlserver.sql_text)),
ADD EVENT sqlserver.lock_deadlock_chain(
ACTION(sqlserver.client_pid,sqlserver.database_name,sqlserver.sql_text)),
ADD EVENT sqlserver.xml_deadlock_report(
ACTION(sqlserver.client_pid,sqlserver.database_name,sqlserver.sql_text))
WITH (STARTUP_STATE=ON)
GO
```

Once the session event is started, it starts to collect data. If you are on SQL Server Management Studio then we can also do a "Live Preview" of the collection made. In my example, I have gone ahead and simulated a deadlock similar to the above. And if we watch our events collection for the session, we can get the XML_DeadLock_Report, which is similar to the graphical report we got from Profiler in XML format.

Displaying 5 Events. SQL server events session stopped or no longer available.

	name	timestamp
	lock_deadlock_chain	2014-08-03 13:03:08.8578336
	lock_deadlock_chain	2014-08-03 13:03:08.8578393
▶	xml_deadlock_report	2014-08-03 13:03:08.8584129
	lock_deadlock	2014-08-03 13:03:08.8584791
	database_xml_deadlock_report	2014-08-03 13:03:08.8585505

Event: xml_deadlock_report (2014-08-03 13:03:08.8584129)

Details Deadlock

Field	Value
client_pid	0
database_name	master
xml_report	<deadlock> <victim-list> <victimProcess id="process41d5b41..."

As you can see, the results and reports contain a number of interesting information which we will never be able to collect it using Profiler. Hence it makes sense, this is the future. So before it becomes late, I highly urge you to play and get accustomed with XEvents.

CONCLUSION

Blocking and Locking inside SQL Server is part of the system. It is inevitable because to maintain integrity of data and show consistent data to users accessing the system – Locking is important. As a special case scenario, keep an eye on Deadlocks and the best way to mitigate the same will be proper coding practices. This paper talked about how to troubleshoot and identify blocking behaviour. We have not explained in detail how to mitigate the situation yet, but these quick troubleshooting techniques will surely make you efficient in looking at locks and blocking situation better.

NEXT STEPS

For more information about Embarcadero award-winning database tools and to try them for free, visit www.embarcadero.com/products/database-tools.

ABOUT THE AUTHOR

Pinal Dave is a Developer Evangelist. He has authored 11 SQL Server database books, 14 Pluralsight courses and has written over 2900 articles on the database technology on his blog at <http://blog.sqlauthority.com>. Along with 10+ years of hands on experience he holds a Masters of Science degree and a number of certifications, including MCTS, MCDBA and MCAD (.NET). His past work experiences include Technology Evangelist at Microsoft and Sr. Consultant at SolidQ.



Embarcadero Technologies, Inc. is the leading provider of software tools that empower application developers and data management professionals to design, build, and run applications and databases more efficiently in heterogeneous IT environments. Over 90 of the Fortune 100 and an active community of more than three million users worldwide rely on Embarcadero's award-winning products to optimize costs, streamline compliance, and accelerate development and innovation. Founded in 1993, Embarcadero is headquartered in San Francisco with offices located around the world. Embarcadero is online at www.embarcadero.com.

Download a Free Trial at www.embarcadero.com

Corporate Headquarters | Embarcadero Technologies | 275 Battery Street, Suite 1000 | San Francisco, CA 94111 | www.embarcadero.com | sales@embarcadero.com

© 2014 Embarcadero Technologies, Inc. Embarcadero, the Embarcadero Technologies logos, and all other Embarcadero Technologies product or service names are trademarks or registered trademarks of Embarcadero Technologies, Inc.
All other trademarks are property of their respective owners 081414