Introduction to Locking in SQL Server

Written by **Mladen Prajdić** on **12 December 2007** | 4 Comments
http://www.sqlteam.com/article/introduction-to-locking-in-sql-server

Locking is a major part of every RDBMS and is important to know about. It is a database functionality which without a multi-user environment could not work. The main problem of locking is that in an essence it's a logical and not physical problem. This means that no amount of hardware will help you in the end. Yes you might cut execution times but this is only a virtual fix. In a heavy multi-user environment any logical problems will appear sooner or later.

Lock modes

All examples are run under the default READ COMMITED isolation level. Taken locks differ between isolation levels, however these examples are just to demonstrate the lock mode with an example. Here's a little explanation of the three columns from sys.dm_tran_locks used in the examples:

resource_type	This tells us what resource in the database the locks are being taken on. It can be one of these values: DATABASE, FILE, OBJECT, PAGE, KEY, EXTENT, RID, APPLICATION, METADATA, HOBT, ALLOCATION_UNIT.
request_mode	This tells us the mode of our lock.
resource_description	This shows a brief description of the resource. Usually holds the id of the page, object, file, row, etc. It isn't populated for every type of lock

The filter on resource_type <> 'DATABASE' just means that we don't want to see general shared locks taken on databases. These are always present. All shown outputs are from the sys.dm_tran_locks dynamic management view. In some examples it is truncated to display only locks relevant for the example. For full output you can run these yourself.

Shared locks (S)

Shared locks are held on data being read under the pessimistic concurrency model. While a shared lock is being held other transactions can read but can't modify locked data. After the locked data has been read the shared lock is released, unless the transaction is being run with the locking hint (READCOMMITTED, READCOMMITTEDLOCK) or under the isolation level equal or more restrictive than Repeatable Read. In the example you can't see the shared locks because they're taken for the duration of the select statement and are already released when we would select data from sys.dm_tran_locks. That is why an addition of WITH (HOLDLOCK) is needed to see the locks.

```
BEGIN TRAN
```

```
USE AdventureWorks

SELECT * FROM Person.Address WITH (HOLDLOCK)
WHERE AddressId = 2

SELECT resource_type, request_mode, resource_description
FROM   sys.dm_tran_locks
WHERE  resource_type <> 'DATABASE'

ROLLBACK
```

	resource_type	request_mode	resource_description
1	OBJECT	IS	
2	KEY	S	(020068e8b274)
3	PAGE	IS	1:9312

Update locks (U)

Update locks are a mix of shared and exclusive locks. When a DML statement is executed SQL Server has to find the data it wants to modify first, so to avoid lock conversion deadlocks an update lock is used. Only one update lock can be held on the data at one time, similar to an exclusive lock. But the difference here is that the update lock itself can't modify the underlying data. It has to be converted to an exclusive lock before the modification takes place. You can also force an update lock with the UPDLOCK hint:

```
BEGIN TRAN

USE AdventureWorks

SELECT * FROM Person.Address WITH (UPDLOCK)
WHERE AddressId < 2

SELECT resource_type, request_mode, resource_description
FROM   sys.dm_tran_locks
WHERE  resource_type <> 'DATABASE'

ROLLBACK
```

	resource_type	request_mode	resource_description
1	OBJECT	IX	
2	PAGE	IU	1:9312
3	KEY	U	(010086470766)

Exclusive locks (X)

Exclusive locks are used to lock data being modified by one transaction thus preventing modifications by other concurrent transactions. You can read data held by exclusive lock only by specifying a NOLOCK hint or using a read uncommitted isolation level. Because DML statements first need to read the data they want to modify you'll always find Exclusive locks accompanied by shared locks on that same data.

```
BEGIN TRAN
```

```
USE AdventureWorks

UPDATE Person.Address
SET AddressLine2 = 'Test Address 2'
WHERE AddressId = 5

SELECT resource_type, request_mode, resource_description
FROM   sys.dm_tran_locks
WHERE  resource_type <> 'DATABASE'

ROLLBACK
```

	resource_type	request_mode	resource_description
1	PAGE	IX	1:20680
2	OBJECT	IX	
3	KEY	X	(43040cc04e21)
4	PAGE	IX	1:9312
5	KEY	X	(0500d1d065e9)
6	KEY	X	(9106f07a8c69)

Intent locks (I)

Intent locks are a means in which a transaction notifies other transaction that it is intending to lock the data. Thus the name. Their purpose is to assure proper data modification by preventing other transactions to acquire a lock on the object higher in lock hierarchy. What this means is that before you obtain a lock on the page or the row level an intent lock is set on the table. This prevents other transactions from putting exclusive locks on the table that would try to cancel the row/page lock. In the example we can see the intent exclusive locks being placed on the page and the table where the key is to protect the data from being locked by other transactions.

```
BEGIN TRAN

USE AdventureWorks

UPDATE TOP(5) Person.Address
SET AddressLine2 = 'Test Address 2'
WHERE PostalCode = '98011'

SELECT resource_type, request_mode, resource_description
FROM   sys.dm_tran_locks
WHERE  resource_type <> 'DATABASE'

ROLLBACK
```

	resource_type	request_mode	resource_description
1	PAGE	UIX	1:20635
2	PAGE	UIX	1:20636
3	PAGE	UIX	1:20608
4	PAGE	IX	1:20696
5	PAGE	UIX	1:20680
6	PAGE	UIX	1:2153
7	OBJECT	IX	

Schema locks (Sch)

There are two types of schema locks:

- Schema stability lock (Sch-S): Used while generating execution plans. These locks don't block access to the object data.
- Schema modification lock (Sch-M): Used while executing a DDL statement. Blocks access to the object data since its structure is being changed.

In the example we can see the Sch-S and Sch-M locks being taken on the system tables and the TestTable plus a lot of other locks on the system tables.

```
BEGIN TRAN

USE AdventureWorks

CREATE TABLE TestTable (TestColumn INT)

SELECT resource_type, request_mode, resource_description
FROM   sys.dm_tran_locks
WHERE  resource_type <> 'DATABASE'

ROLLBACK
```

	resource_type	request_mode	resource_description
1	METADATA	Sch-S	data_space_id = 1
2	HOBT	Sch-M	
3	OBJECT	Sch-M	

Bulk Update locks (BU)

Bulk Update locks are used by bulk operations when TABLOCK hint is used by the import. This allows for multiple fast concurrent inserts by disallowing data reading to other transactions.

Conversion locks

Conversion locks are locks resulting from converting one type of lock to another. There are 3 types of conversion locks:

- Shared with Intent Exclusive (SIX). A transaction that holds a Shared lock also has some pages/rows locked with an Exclusive lock
- Shared with Intent Update (SIU). A transaction that holds a Shared lock also has some pages/rows locked with an Update lock.
- Update with Intent Exclusive (UIX). A transaction that holds an Update lock also has some pages/rows locked with an Exclusive lock.

In the example you can see the UIX conversion lock being taken on the page:

```
BEGIN TRAN

USE AdventureWorks
```

```
UPDATE TOP(5) Person.Address
SET AddressLine2 = 'Test Address 2'
WHERE PostalCode = '98011'

SELECT resource_type, request_mode, resource_description
FROM   sys.dm_tran_locks
WHERE  resource_type <> 'DATABASE'

ROLLBACK
```

	resource_type	request_mode	resource_description
1	PAGE	UIX	1:20635
2	PAGE	UIX	1:20636
3	PAGE	UIX	1:20608
4	PAGE	IX	1:20696
5	PAGE	UIX	1:20680
6	PAGE	UIX	1:2153
7	OBJECT	IX	
8	KEY	X	(06003f7fd0fb)

Key - Range locks

Key-range locks protect a range of rows implicitly included in a record set being read by a Transact-SQL statement while using the serializable transaction isolation level. Key-range locking prevents phantom reads. By protecting the ranges of keys between rows, it also prevents phantom insertions or deletions into a record set accessed by a transaction. In the example we can see that there are two types of key-range locks taken:

- RangeX-X - exclusive lock on the interval between the keys and exclusive lock on the last key in the range
- RangeS-U – shared lock on the interval between the keys and update lock on the last key in the range

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRAN

USE AdventureWorks

UPDATE Person.Address
SET AddressLine2 = 'Test Address 2'
WHERE AddressLine1 LIKE '987 %'

SELECT resource_type, request_mode, resource_description
FROM   sys.dm_tran_locks
WHERE  resource_type <> 'DATABASE'

ROLLBACK
```

	resource_type	request_mode	resource_description
1	OBJECT	IX	
2	PAGE	IX	1:2610
3	KEY	RangeX-X	(1605ead61088)
4	KEY	X	(22000dabf7be)
5	KEY	RangeS-U	(6308fb95e03f)
6	PAGE	IX	1:9538
7	KEY	X	(64079fd05627)

Lock Granularity

Lock granularity consists of TABLE, PAGE and ROW locks. If you have a clustered index on the table then instead of a ROW lock you have a KEY lock. Locking on the lower level increases concurrency, but if a lot of locks are taken consumes more memory and vice versa for the higher levels. So granularity simply means the level at which the SQL Server locks data. Also note that the more restricted isolation level we choose, the higher the locking level to keep data in correct state. You can override the locking level by using ROWLOCK, PAGLOCK or TABLOCK hints but the use of these hints is discouraged since SQL Server know what are the appropriate locks to take for each scenario. If you must use them you should be aware of the concurrency and data consistency issues you might cause.

Spinlocks

Spinlocks are a light-weight lock mechanism that doesn't lock data but it waits for a short period of time for a lock to be free if a lock already exists on the data a transaction is trying to lock. It's a mutual exclusion mechanism to reduce context switching between threads in SQL Server.

Lock Compatibility Matrix

This is taken from http://msdn2.microsoft.com/En-US/library/ms186396.aspx. Also a good resource to have is a Lock Compatibility Matrix which tells you how each lock plays nice with other lock modes. It is one of those things you don't think you need up until the moment you need it.

	NL	SCH-S	SCH-M	S	U	X	IS	IU	IX	SIU	SIX	UIX	BU	RS-S	RS-U	RI-N	RI-S	RI-U	RI-X	RX-S	RX-U	RX-X
NL	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SCH-S	N	N	C	N	N	N	N	N	N	N	N	N	N	I	I	I	I	I	I	I	I	I
SCH-M	N	C	C	C	C	C	C	C	C	C	C	C	C	I	I	I	I	I	I	I	I	I
S	N	N	C	N	N	C	N	N	C	N	C	C	C	N	N	N	N	N	C	N	N	C
U	N	N	C	N	C	C	N	C	C	C	C	C	C	N	C	N	N	C	C	N	C	C
X	N	N	C	C	C	C	C	C	C	C	C	C	C	C	C	N	C	C	C	C	C	C
IS	N	N	C	N	N	C	N	N	N	N	N	N	C	I	I	I	I	I	I	I	I	I
IU	N	N	C	N	C	C	N	N	N	N	N	C	C	I	I	I	I	I	I	I	I	I
IX	N	N	C	C	C	C	N	N	N	C	C	C	C	I	I	I	I	I	I	I	I	I
SIU	N	N	C	N	C	C	N	N	C	N	C	C	C	I	I	I	I	I	I	I	I	I
SIX	N	N	C	C	C	C	N	N	C	C	C	C	C	I	I	I	I	I	I	I	I	I
UIX	N	N	C	C	C	C	N	C	C	C	C	C	C	I	I	I	I	I	I	I	I	I
BU	N	N	C	C	C	C	C	C	C	C	C	C	N	I	I	I	I	I	I	I	I	I
RS-S	N	I	I	N	N	C	I	I	I	I	I	I	I	N	N	C	C	C	C	C	C	C
RS-U	N	I	I	N	C	C	I	I	I	I	I	I	I	N	C	C	C	C	C	C	C	C
RI-N	N	I	I	N	N	N	I	I	I	I	I	I	I	C	C	N	N	N	N	C	C	C
RI-S	N	I	I	N	N	C	I	I	I	I	I	I	I	C	C	N	N	N	C	C	C	C
RI-U	N	I	I	N	C	C	I	I	I	I	I	I	I	C	C	N	N	C	C	C	C	C
RI-X	N	I	I	C	C	C	I	I	I	I	I	I	I	C	C	N	C	C	C	C	C	C
RX-S	N	I	I	N	N	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C
RX-U	N	I	I	N	C	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C
RX-X	N	I	I	C	C	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C

Key

N	No Conflict	SIU	Share with Intent Update
I	Illegal	SIX	Shared with Intent Exclusive
C	Conflict	UIX	Update with Intent Exclusive
		BU	Bulk Update
NL	No Lock	RS-S	Shared Range-Shared
SCH-S	Schema Stability Locks	RS-U	Shared Range-Update
SCH-M	Schema Modification Locks	RI-N	Insert Range-Null
S	Shared	RI-S	Insert Range-Shared
U	Update	RI-U	Insert Range-Update
X	Exclusive	RI-X	Insert Range-Exclusive
IS	Intent Shared	RX-S	Exclusive Range-Shared
IU	Intent Update	RX-U	Exclusive Range-Update
IX	Intent Exclusive	RX-X	Exclusive Range-Exclusive

Conclusion

Hopefully this article has shed some light on how SQL Server operates with locks and why is locking of such importance to proper application and database design and operation. Remember that locking problems are of logical and not physical nature so they have to be well thought out. Locking goes hand in hand with transaction isolation levels so be familiar with those too. In the next article I'll show some ways to resolve locking problems.