All about locking in SQL Server

June 16, 2017 by Nikola Dimitrijevic

Locking is essential to successful SQL Server transactions processing and it is designed to allow SQL Server to work seamlessly in a multi-user environment. Locking is the way that SQL Server manages transaction concurrency. Essentially, locks are in-memory structures which have owners, types, and the hash of the resource that it should protect. A lock as an in-memory structure is 96 bytes in size.

To understand better the locking in SQL Server, it is important to understand that locking is designed to ensure the integrity of the data in the database, as it forces every SQL Server transaction to pass the ACID test.

ACID test consists of 4 requirements that every transaction have to pass successfully:

- **Atomicity** – requires that a transaction that involves two or more discrete parts of information must commit all parts or none
- **Consistency** – requires that a transaction must create a valid state of new data, or it must roll back all data to the state that existed before the transaction was executed
- **Isolation** – requires that a transaction that is still running and did not commit all data yet, must stay isolated from all other transactions
- **Durability** – requires that committed data must be stored using method that will preserve all data in correct state and available to a user, even in case of a failure

SQL Server locking is the essential part of the isolation requirement and it serves to lock the objects affected by a transaction. While objects are locked, SQL Server will prevent other transactions from making any change of data stored in objects affected by the imposed lock. Once the lock is released by committing the changes or by rolling back changes to initial state, other transactions will be allowed to make required data changes.

Translated into the SQL Server language, this means that when a transaction imposes the lock on an object, all other transactions that require the access to that object will be forced to wait until the lock

is released and that wait will be registered with the adequate wait type

SQL Server locks can be specified via the lock modes and lock granularity

Lock modes

Lock mode considers various lock types that can be applied to a resource that has to be locked:

- Exclusive (X)
- Shared (S)
- Update (U)
- Intent (I)
- Schema (Sch)
- Bulk update (BU)

Exclusive lock (X) – This lock type, when imposed, will ensure that a page or row will be reserved *exclusively* for the transaction that imposed the exclusive lock, as long as the transaction holds the lock.

The exclusive lock will be imposed by the transaction when it wants to modify the page or row data, which is in the case of DML statements DELETE, INSERT and UPDATE. An exclusive lock can be imposed to a page or row only if there is no other shared or exclusive lock imposed already on the target. This practically means that only one exclusive lock can be imposed to a page or row, and once imposed no other lock can be imposed on locked resources

Shared lock (S) – this lock type, when imposed, will reserve a page or row to be available only for reading, which means that any other transaction will be prevented to modify the locked record as long as the lock is active. However, a shared lock can be imposed by several transactions at the same time over the same page or row and in that way several transactions can *share* the ability for data reading since the reading process itself will not affect anyhow the actual page or row data. In addition, a shared lock will allow write operations, but no DDL changes will be allowed

Update lock (U) – this lock is similar to an exclusive lock but is designed to be more flexible in a way. An update lock can be imposed on a record that already has a shared lock. In such a case, the update lock will impose another shared lock on the target row. Once the transaction that holds the update lock is ready to change the data, the update lock (U) will be transformed to an exclusive lock (X). It is important to understand that update lock is asymmetrical in regards of shared locks. While the update lock can be imposed on a record that has the shared lock, the shared lock cannot be imposed on the record that already has the update lock

Intent locks (I) – this lock is a means used by a transaction to inform another transaction about its *intention* to acquire a lock. The purpose of such lock is to ensure data modification to be executed properly by preventing another transaction to acquire a lock on the next in hierarchy object. In practice, when a transaction wants to acquire a lock on the row, it will acquire an intent lock on a table,

which is a higher hierarchy object. By acquiring the intent lock, the transaction will not allow other transactions to acquire the exclusive lock on that table (otherwise, exclusive lock imposed by some other transaction would cancel the row lock).

This is an important lock type from the performance aspect as the SQL Server database engine will inspect intent locks only at the table level to check if it is possible for transaction to acquire a lock in a safe manner in that table, and therefore intent lock eliminates need to inspect each row/page lock in a table to make sure that transaction can acquire lock on entire table

There are three regular intent locks and three so-called conversion locks:

Regular intent locks:

Intent exclusive (IX) – when an intent exclusive lock (IX) is acquired it indicates to SQL Server that the transaction has the intention to modify some of lower hierarchy resources by acquiring exclusive (X) locks individually on those lower hierarchy resources

Intent shared (IS) – when an intent shared lock (IS) is acquired it indicates to SQL Server that the transaction has the intention to read some lower hierarchy resources by acquiring shared locks (S) individually on those resources lower in the hierarchy

Intent update (IU) – when an intent shared lock (IS) is acquired it indicates to SQL Server that the transaction has the intention to read some of lower hierarchy resources by acquiring shared locks (S) individually on those resources lower in the hierarchy. The intent update lock (IU) can be acquired only at the page level and as soon as the update operation takes place, it converts to the intent exclusive lock (IX)

Conversion locks:

Shared with intent exclusive (SIX) – when acquired, this lock indicates that the transaction intends to read all resources at a lower hierarchy and thus acquire the shared lock on all resources that are lower in hierarchy, and in turn, to modify part of those, but not all. In doing so, it will acquire an intent exclusive (IX) lock on those lower hierarchy resources that should be modified. In practice, this means that once the transaction acquires a SIX lock on the table, it will acquire intent exclusive lock (IX) on the modified pages and exclusive lock (X) on the modified rows.

Only one shared with intent exclusive lock (SIX) can be acquired on a table at a time and it will block other transactions from making updates, but it will not prevent other transactions to read the lower hierarchy resources they can acquire the intent shared (IS) lock on the table

Shared with intent update (SIU) – this is a bit more specific lock as it is a combination of the shared (S) and intent update (IU) locks. A typical example of this lock is when a transaction is using a query

(S) and intent update (IU) locks. A typical example of this lock is when a transaction is using a query executed with the PAGELOCK hint and query, then the update query. After the transaction acquires an SIU lock on the table, the query with the PAGELOCK hint will acquire the shared (S) lock while the update query will acquire intent update (IU) lock

Update with intent exclusive (UIX) – when update lock (U) and intent exclusive (IX) locks are acquired at lower hierarchy resources in the table simultaneously, the update with intent exclusive lock will be acquired at the table level as a consequence

Schema locks (Sch) – The SQL Server database engine recognizes two types of the schema locks: **Schema modification lock (Sch-M)** and **Schema stability lock (Sch-S)**

- A **Schema modification lock (Sch-M)** will be acquired when a DDL statement is executed, and it will prevent access to the locked object data as the structure of the object is being changed. SQL Server allows a single schema modification lock (Sch-M) lock on any locked object. In order to modify a table, a transaction must wait to acquire a Sch-M lock on the target object. Once it acquires the schema modification lock (Sch-M), the transaction can modify the object and after the modification is completed and the lock will be released. A typical example of the Sch-M lock is an index rebuild, as an index rebuild is table modification process. Once the index rebuild ID is issued, a schema modification lock (Sch-M) will be acquired on that table and will be released only after the index rebuild process is completed (when used with ONLINE option, index rebuild will acquire Sch-M lock shortly at the end of the process)
- A **Schema stability lock (Sch-S)** will be acquired while a schema-dependent query is being compiled and executed and execution plan is generated. This particular lock will not block other transactions to access the object data and it is compatible with all lock modes except with the schema modification lock (Sch-M). Essentially, Schema stability locks will be acquired by every DML and select query to ensure the integrity of the table structure (ensure that table doesn't change while queries are running).

Bulk Update locks (BU) – this lock is designed to be used by bulk import operations when issued with a TABLOCK argument/hint. When a bulk update lock is acquired, other processes will not be able to access a table during the bulk load execution. However, a bulk update lock will not prevent another bulk load to be processed in parallel. But keep in mind that using TABLOCK on a clustered index table will not allow parallel bulk importing. More details about this is available in Guidelines for Optimizing Bulk Import

Locking hierarchy

SQL Server has introduced the locking hierarchy that is applied when reading or changing of data is performed. The lock hierarchy starts with the database at the highest hierarchy level and down via table and page to the row at the lowest level

Essentially, there is always a shared lock on the database level that is imposed whenever a transaction is connected to a database. The shared lock on a database level is imposed to prevent dropping of the database or restoring a database backup over the database in use. For example, when a SELECT statement is issued to read some data, a shared lock (S) will be imposed on the database level, an intent shared lock (IS) will be imposed on the table and on the page level, and a shared lock (S) on the row itself

In case of a DML statement (i.e. insert, update, delete) a shared lock (S) will be imposed on the database level, an intent exclusive lock (IX) or intent update lock (IU) will be imposed on the table and on the page level, and an exclusive or update lock (X or U) on the row

Locks will always be acquired from the top to the bottom as in that way SQL Server is preventing a so-

Locks will always be acquired from the top to the bottom as in that way SQL Server is preventing a so-called Race condition to occur.

Now that lock modes and lock hierarchy have been explained, let's further elaborate on lock modes and how those translate to a lock hierarchy.

Not all lock modes can be applied at all levels.

At the row level, the following three lock modes can be applied:

- Exclusive (X)
- Shared (S)
- Update (U)

To understand the compatibility of those modes, please refer to the following table:

	Exclusive (X)	**Shared (S)**	**Update (U)**
Exclusive (X)	✗	✗	✗
Shared (S)	✗	✓	✓
Update (U)	✗	✓	✗

✓ – Compatible ✗ – Incompatible

At the table level, there are five different types of locks:

- Exclusive (X)
- Shared (S)
- Intent exclusive (IX)
- Intent shared (IS)
- Shared with intent exclusive (SIX)

Compatibility of these modes can be seen in the table below

	(X)	**(S)**	**(IX)**	**(IS)**	**(SIX)**
(X)	✗	✗	✗	✗	✗
(S)	✗	✓	✗	✓	✗
(IX)	✗	✗	✓	✓	✗
(IS)	✗	✓	✓	✓	✓

(SIX)	✗	✗	✗	✓	✗

✓ – Compatible ✗ – Incompatible

A Schema lock (Sch) is also a table level lock as well, but it is not a data related lock

To better understand the compatibility between these lock types, please refer to this table:

Lock escalation

In order to prevent a situation where locking is using too many resources, SQL Server has introduced the lock escalation feature.

Without escalation, locks could require a significant amount of memory resources. Let's take an example where a lock should be imposed on the 30,000 rows of data, where each row is 500 bytes in size, to perform the delete operation. Without escalation, a shared lock (S) will be imposed on the database, 1 intent exclusive lock (IX) on the table, 1,875 intent exclusive locks (IX) on the pages (8KB page hold 16 rows of 500 bytes, which makes 1,875 pages that hold 30,000 rows) and 30,000 exclusive locks (X) on the rows itself. As each lock is 96 bytes in size, 31,877 locks will take about 3 MB of memory for a single delete operation. Running large number of operations in parallel could require some significant resources just to ensure that locking manager can perform the operation smoothly

To prevent such a situation, SQL Server uses lock escalation. This means that in a situation where more than 5,000 locks are acquired on a single level, SQL Server will *escalate* those locks to a single table level lock. By default, SQL Server will always escalate to the table level directly, which mean that escalation to the page level never occurs. Instead of acquiring numerous rows and pages lock, SQL Server will escalate to the exclusive lock (X) on a table level

While this will reduce the need for resources, exclusive locks (X) in a table mean that no other transac-

tion will be able to access locked table and all queries trying to access that table will be blocked. Therefore, this will reduce system overhead but will increase the probability of concurrency contention

In order to provide control over the escalation, starting with SQL Server 2008 R2, the LOCK_EXCALATION option is introduced as part of the ALTER TABLE statement

```
USE AdventureWorks2014
GO
ALTER TABLE Table_name
SET (LOCK_ESCALATION = < TABLE | AUTO | DISABLE > -One of those options)
GO
```

Each of these options is defined to allow specific control over the lock escalation process:

Table – This is the default option for any newly created table, as by default SQL Server will always execute lock escalation to the table level, which also includes partitioned tables

Auto – This option allows the lock escalation to a partition level when a table is partitioned. When 5,000 locks are acquired in a single partition, lock escalation will acquire an exclusive lock (X) on that partition while the table will acquire intent exclusive lock (IX). In case that table is not partitioned, lock escalation will acquire the lock on the table level (equal to the **Table** option).

Although this looks like a very useful option, it has to be used very carefully as it can easily cause a deadlock. In a situation where we have two transactions on two partitions where the exclusive lock (X) is acquired, and transactions tries to access the date from partition used by other transaction, a deadlock will be encountered

So, it is very important to carefully control the data access pattern, if this option is enabled, which is not easy to achieve, and this is why this option is not the default settings in SQL Server

Disable – This option will completely disable lock escalation for a table. Again, this option must be used carefully to avoid the SQL Server lock manager to be forced to use an excessive amount of memory

As it can be seen, lock escalation could be a challenge for DBAs. If the application design requires deleting or updating more than 5,000 rows at once, a solution to avoid lock escalation, and the resulting effects, is splitting the single transaction into a two or more transaction where each will handle less than 5,000 rows, as in this way the lock escalation could be evaded

Get info about active SQL Server locks

SQL Server provides the Dynamics Management View (DMV) sys.dm_tran_locks that returns information about lock manager resources that are currently in use, which means that it will display all "live" locks acquired by transactions. More details about this DMV can be found in the sys.dm_tran_locks (Transact-SQL) article.

The most important column used for the identification of the lock are **resource_type, request_mode, and resource_description.** If needed, more columns as additional resource for information info can be included during troubleshooting

Here is the example of the query

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

The where clause in this query is used as the filter on the resource_type to eliminate. from the results, those generally shared locks acquired on the database since these are always present at the database level

A brief explanation of the three columns presented here:

resource_type – Displays a database resource where the locks are being acquired. The column can display one of the following values: ALLOCATION_UNIT, APPLICATION, DATABASE, EXTENT, FILE, HOBT, METADATA, OBJECT, PAGE, KEY, RID

request_mode – displays the lock mode that is acquired on resource

resource_description – displays a short resource description and is not populated for all lock modes. Most often the column contains the id of the row, page, object, file, etc

See more

Check out ApexSQL Plan, to view SQL execution plans, including comparing plans, stored procedure performance profiling, missing index details, lazy profiling, wait times, plan execution history and more

Nikola Dimitrijevic

Nikola is computer freak since 1981 and an SQL enthusiast with intention to became a freak. Specialized in SQL Server auditing, compliance and performance monitoring.

Military aviation devotee and hard core scale aircraft modeler. Extreme sports fan; parachutist and bungee jump instructor. Once serious, now just a free time photographer

View all posts by Nikola Dimitrijevic

Related Posts:

1. **An overview of sp_getapplock and sp_releaseapplock stored procedures**
2. **Top SQL Server Books**
3. **Protecting Azure SQL databases from accidental deletion**
4. **SQL Server Transaction Log Interview Questions**
5. **SQL Server table hints – WITH (NOLOCK) best practices**

Locking

475,524 Views

Pankaj Lala • 4 years ago

Very apt and detailed information about the said topic, just to confirm figure 3 under Locking hierarchy should it not be Intent exclusive (IX) instead of Intent exclusive (IS)?

10 • Reply • Share ›

Nikola Dimitrijevic ➚ Pankaj Lala • 4 years ago

Yes Pankaj, you are right. Good catch. The text is good but with the wrong acronym. It is corrected already

Thanks a lot

7 • Reply • Share ›

Sudhanshu Mishra • 4 years ago

Great Explanation! Thanks for the articulate write-up of this complex topic.

7 • Reply • Share ›

Garry Cotton • 4 years ago

Excellent article, extremely informative and concise - thank you :)

6 • Reply • Share ›

Thiago • 3 years ago

Nikola, thanks for the article.

Besides UPDATE operations, I've seen DELETEs invoking IU (Intent Update) locks also.

Isn't DELETEs supposed to only call IX (Intent Exclusive) locks instead?

Maybe it calls IU in order to evaluate if the rows are or aren't elegible to be deleted... but it seems awkward still.

Thanks.

4 ^ | v • Reply • Share ›

Nikola Dimitrijevic ➚ Thiago • 3 years ago • edited

Thiago,

I am not entirely sure about this as I don't know your actual scenario and frankly, I haven't seen such situations by myself, so my answers are based on pure logic. Honestly, I haven't seen something like this documented.

It may be possible that depending on the DELETE statement issued, SQL Server grants the IU lock at the page level before the actual update take place when it converts to IX lock, but again this is the long shot and I have not seen any documentation that can confirm that

However, there are situations where SQL Server can internally change the UPDATE operation to a DELETE/INSERT. It can occur in cases where updating a key column of the partitioned view or table, update multiple rows where the key of a nonunique clustered index changes to a nonconstant value or If changing column that is part of a unique constraint in transactional replication. It is a so-called "deferred update".

So it is possible that IU is acquired at the page level, and when the operation takes place it executes it as DELETE and INSERT, and therefore it might impose the false image that IU lock is acquired for DELETE rather than for UPDATE

3 ^ | v • Reply • Share ›

Jim Gerrard • 3 years ago

"Exclusive lock (X) – This lock type, when imposed, will ensure that a page or row will be reserved exclusively for the transaction that imposed the SHARED lock, as long as the transaction holds the lock."

I'm pretty sure you meant "exclusive" rather than "shared"...?

4 ^ | v • Reply • Share ›

Nikola Dimitrijevic ➚ Jim Gerrard • 3 years ago

Yes, you are right Jim.

That was an honest mistake that is already corrected. Thanks a lot for being such a careful reader and for

taking the time to let us know

4 ^ | v • Reply • Share ›

Agus • 3 years ago

Thank you very much! This explains things in a simple language that msdn can't. By the way would you share what tool do you use to draw the elegant lock hierarchy image and the cross (x) and check box (v) characters?

4 ^ | v • Reply • Share ›

Nikola Dimitrijevic ➚ Agus • 3 years ago

That means a lot, Agus! I'm glad to hear that you find the article useful.

As for the diagrams, I'm using "Balsamiq Mockups"

5 ^ | v • Reply • Share ›

Keshav Aggarwal • 3 years ago • edited

Share Lock(S) - In addition, a shared lock will allow write operations, but no DDL changes will be allowed how can a share lock provide a write operations can you explain
?

3 ^ | v • Reply • Share ›

Kevin ➚ Keshav Aggarwal • 3 years ago

I think it means you can write to the table (ie: insert new rows) as long as you don't touch the locked rows/pages.

1 ^ | v • Reply • Share ›

Jochen Ahleff • 3 years ago

Hi Nikola, very resourceful, very helpful for me. Thank you very much.
Just one question: in the first table, where you display the compatibility of Exclusive, Shared, and Update locks against each other, the legend says, "✗ – Compatible ✓ – Incompatible", while I think you meant the opposite, "✓ – Compatible ✗ – Incompatible". This looks more correct, more logical, and would conform with the second table's legend.

3 ^ | v • Reply • Share ›

Nikola Dimitrijevic ➚ Jochen Ahleff • 3 years ago

Hi Jochen,

Good catch and sharp eye. Already corrected

Thanks a lot!

3 ^ | v • Reply • Share ›

Abdul Taher Iqbal • 2 years ago

very Nice explanation ... Thanks

2 ^ | v • Reply • Share ›

Robert Kubarych • 2 years ago

I'm confused about what you said on shared locks. You say that "this lock type, when imposed, will reserve a page or row to be available only for reading, which means that any other transaction will be prevented to modify the locked record as long as the lock is active", but then you go on to say that "In addition, a shared lock will allow write operations, but no DDL changes will be allowed." These two statements appear to be in conflict to me. Can you please clarify for me? Thank you for the detailed and excellent article.

2 ^ | v • Reply • Share ›

Nikola Dimitrijević ➚ Robert Kubarych • 2 months ago

As it states, it will prevent "modifying" the locked records (delete/update), but the write operation (insert) will be allowed as it doesn't affect the data acquired by the shared lock

^ | v • Reply • Share ›

Christian McGhee ➚ Robert Kubarych
• 2 years ago • edited

I am confused about that as well. How can you be writing to page that has a shared lock on it? Seems contradictory.
Shared locks are acquired when reading data. The duration for which a shared lock is held depends on transaction isolation level or locking hints. Many concurrent transactions may hold shared locks on the same data. N**o other transaction can modify the data until the shared lock is released.**

^ | v • Reply • Share ›

John Doe ➚ Robert Kubarych • 2 years ago

Same here!

^ | v • Reply • Share ›

Gyula Kulifai • a year ago

How can I check a finished update transaction (from SSMS) why block a software database job.

And how can software test if his job will block by a lock? may it

can pop up a warning "Something block me!!!"

Gold question: This worked before. I don't know what changed...
But server installed updates...