Locking Mechanism in DataBases

Effects and Explanation

Kian Sahafi, Spring 2023

Why?

- 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

- 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

Why? (Again)

- 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.
- 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)

Lock Modes - 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.

Lock Modes - 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.

Lock Modes - 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.

Lock Modes - 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 which we have in the next slides.

Regular intent locks:

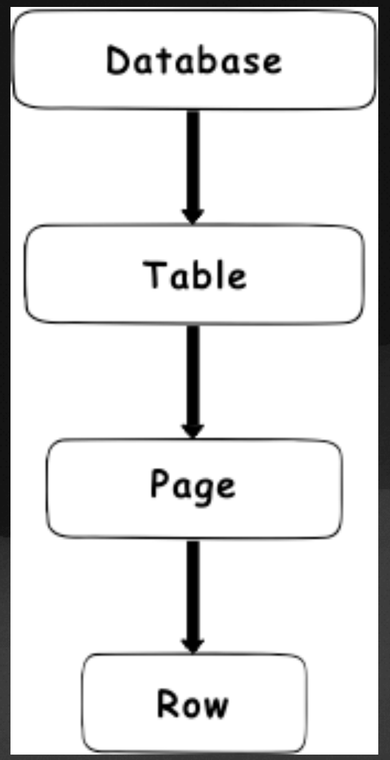
- **Intent exclusive (IX)** – when an intent exclusive lock (IX) is acquired it indicates to SQL Server that the transaction has the intention to modify some of lower hierarchy resources by acquiring exclusive (X) locks individually on those lower hierarchy resources
- **Intent shared (IS)** – when an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

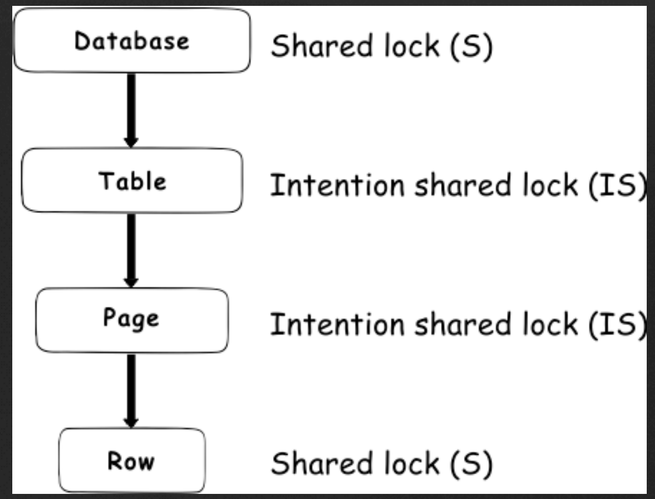
Conversion locks:

- **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 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
- **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.

Locking hierarchy

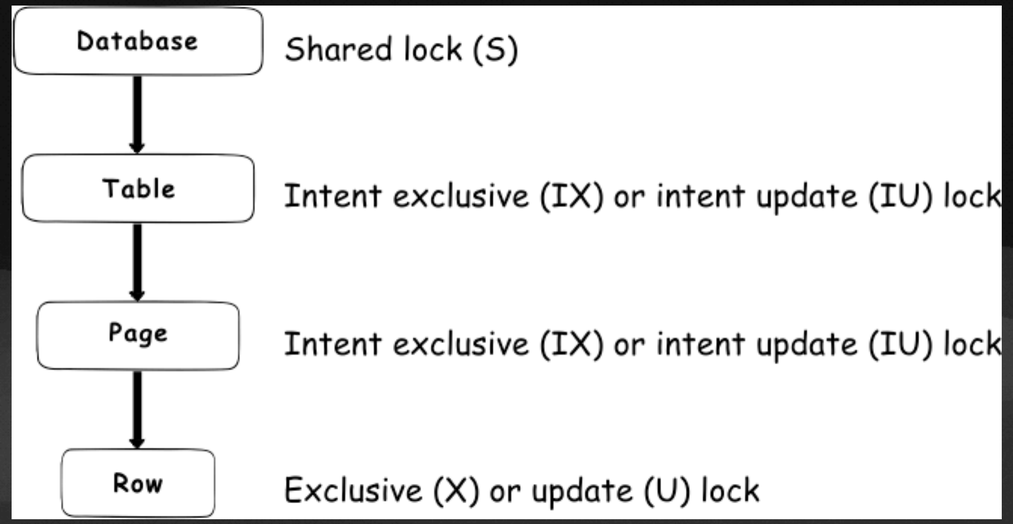
- SQL Server has introduced the locking hierarchy that is applied when reading or changing of data is performed. The lock hierarchy starts with the database at the highest hierarchy level and down via table and page to the row at the lowest level.
- Essentially, there is always a shared lock on the database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





Locking hierarchy (continue)

- 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-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.



Lock Modes - Row Level

- At the row level, the following three lock modes can be applied:
  - Exclusive (X)
  - Shared (S)
  - Update (U)

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

✓ – Compatible ✗ – Incompatible

Lock Modes - Table Level

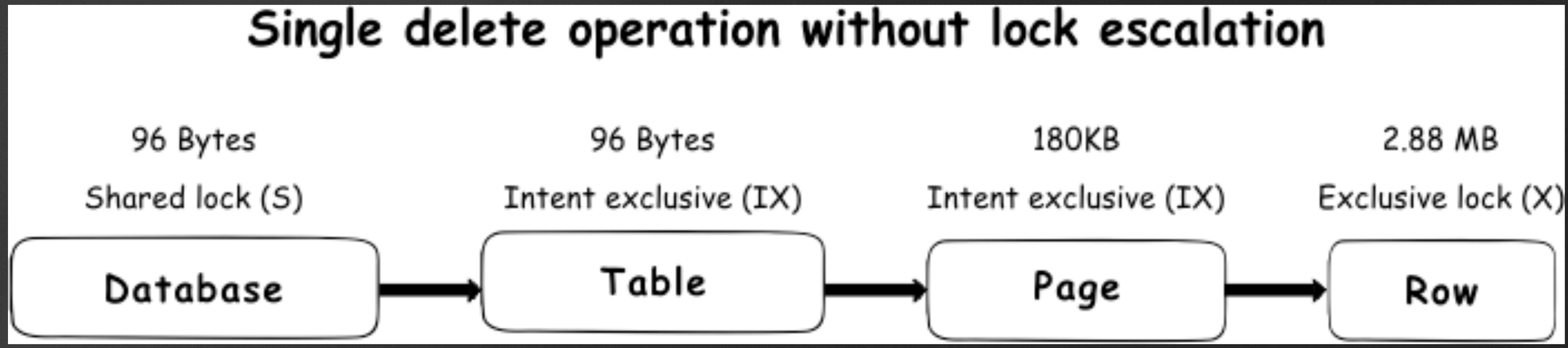
- At the table level, there are five different types of locks:
  - Exclusive (X)
  - Shared (S)
  - Intent exclusive (IX)
  - Intent shared (IS)
  - Shared with intent exclusive (SIX)
- Compatibility of these modes can be seen in the table below:

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

✓ – Compatible ✗ – Incompatible

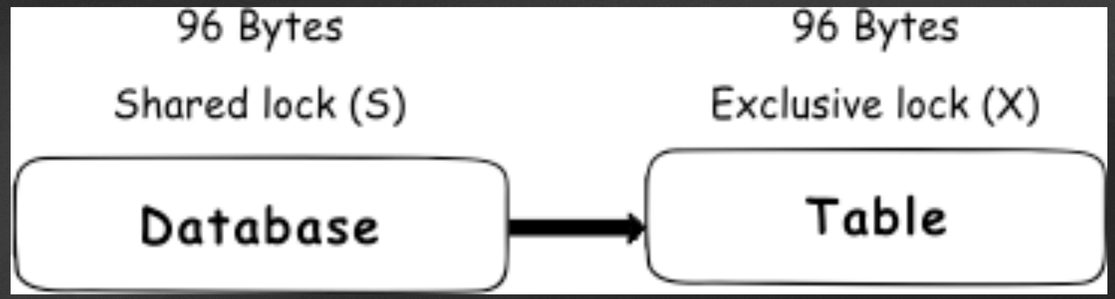
Lock escalation

- 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



Lock escalation (continue)

- 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 transaction 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

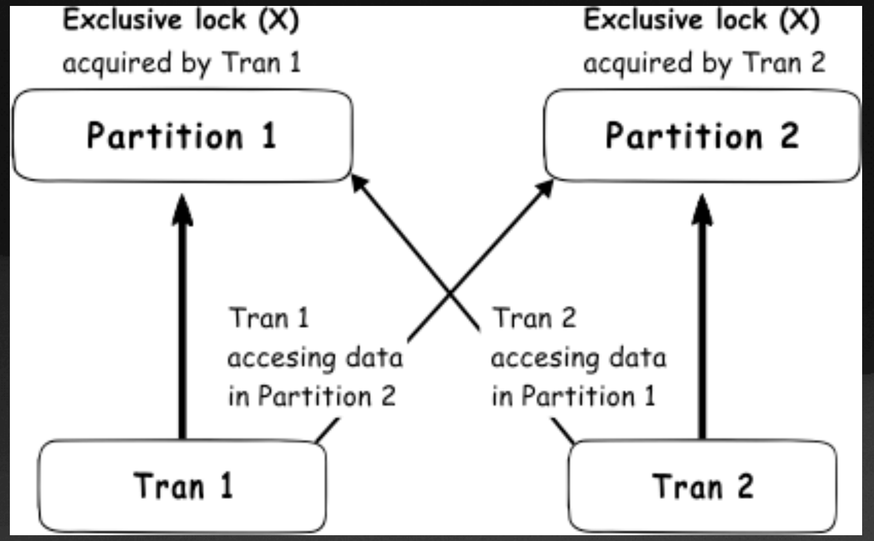
```
USE AdventureWorks2014
GO
ALTER TABLE Table_name
SET (LOCK_ESCALATION = < TABLE | AUTO | DISABLE > –One of those options)
GO
```

- 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

MySQL vs SQL Server vs MongoDB

- MySQL uses table-level locking for MyISAM(default storage engine for the MySQL), MEMORY, and MERGE tables, which means only one session can update those tables at a time. This reduces concurrency but also reduces overhead and memory usage. MySQL uses row-level locking for InnoDB tables(table type which allows you to define foreign key constraints to guard the integrity of your data), which means multiple sessions can update different rows of the same table concurrently. This increases concurrency but also increases overhead and memory usage.

- SQL Server uses row-level locking by default, but can escalate to page-level or table-level locking when necessary. This allows for a balance between concurrency and performance. SQL Server also supports optimistic locking with snapshot isolation and read committed snapshot isolation levels, which use row versions to avoid blocking readers.

- MongoDB uses multi-granularity locking that allows operations to lock at the global, database or collection level, and allows for individual storage engines to implement their own concurrency control below the collection level (e.g., at the document-level in WiredTiger). MongoDB also supports lock-free read operations with snapshot isolation and causal consistency levels.

The effect of locking mechanisms in database

- The effect of locking mechanisms in database is to ensure data consistency and concurrency control:
  - Data Consistency: Locking can help ensure data consistency by preventing multiple users from modifying the same data simultaneously. By controlling access to shared resources, locking can help prevent data conflicts and ensure that the database remains in a consistent state.
  - Concurrency Control: Locking can help manage concurrency control by allowing multiple users to access the same data concurrently without compromising data integrity. By using different levels and modes of locking, locking can balance the trade-offs between performance and isolation.
  - Lock Timeouts and Deadlocks: Locking can also cause some problems when the lock strategy prevents the progress of a thread or process. A lock timeout occurs when a thread or process waits for a lock to be released for too long and gives up. A deadlock occurs when two or more threads or processes wait for each other's locks to be released and none of them can proceed. Depending on the database system, lock timeouts and deadlocks can be handled by setting parameters, using exceptions, or using detection and resolution algorithms.

Thank you for listening…

If you have any questions now is the time.😉

Presented By Kian Sahafi

kiansahafi@gmail.com