**Locks, Blocking, and Deadlocks**

**What is the purpose of locks?**

* Applications use database locks to control data integrity in multiuser concurrency situations and are part of the SQL Server internals. Locks prevent data from being modified by two simultaneous sessions. In a normal server environment, infrequent blocking locks are acceptable. Blocking is not the same thing as a deadlock.
* A certain amount of blocking is normal and unavoidable.

**What are locks?**

* A lock is a placed on an object (row, page, extent, table, database) by the SQL Server when any connection access the same piece of data concurrently

**What is blocking**

* Blocking occurs when one session has a lock on an object and thus causes another session to wait in a holding queue until the current session is entirely done with the resources

**What is a deadlock?**

* A deadlocks occur when two separate transactions (T#1 and T#2) have exclusive locks on objects and at the same time are trying to update or access each other’s objects.

**What causes locks, blocking and deadlocks?**

* Poor database design can cause crippling database lock contention
* Poor indexing strategy
* Query implementation problem
* Tables are not completely normalized
* Optimize Transact-SQL code

**Ways to monitor lock, deadlocks and blocking**

* sp\_lock
* sp\_who2 sproc
* Activity Monitor
* SQL Profiler
* Performance monitor

**Ways to avoid blocking and deadlocks**

* Use clustered indexes on high-usage tables
* Break long transactions up into many shorter transactions.
* Make sure that UPDATE and DELETE statements use an existing index
* If your application’s code allows a running query to be cancelled, it is important that the code also roll back the transaction.

The most common lock modes

* Exclusive locks (X)
* Shared locks (S)
* Update locks (U)
* Intent locks (I)
* Schema locks (two types, SCH-M and SCH-S)
* Bulk Update locks (BU)
* Exclusive lock (X) is placed on a database object whenever it is updated with an INSERT or UPDATE statement
* The shared locks (S) is put to a database object whenever it is being read (using the SELECT statement
* Update lock (U), which can be thought of as an in-between mode between shared and exclusive locks. The main purpose of the update lock is to prevent deadlocks where multiple users simultaneously try to update data
* Intent locks are used to indicate that a certain lock will be later placed on a certain database object. Intent locks are used because locks can form hierarchies
* Schema locks (SCH-M and SCH-S) are used to prevent changes to object structure, bulk update locks (BU) are used when updating or inserting multiple rows using a bulk update,
* Key-range locks (R) are used to lock ranges of keys in an index
* Because maintaining locks can be an expensive operation performance-wise, SQL Server supports a feature called multigranular locking. This means that locks can be placed on different levels, depending on the situation.

What is the sign that blocking and deadlocks are occurring: The USER will tell you!!!

Anytime a query of any type, whether it is a SELECT, INSERT, UPDATE, or DELETE, takes more than a few seconds to complete, blocking is likely.

Locks on SELECT statements are only held as long as it takes to read the data, not the entire length of the transaction.

Locks held by INSERT, UPDATE, and DELETE statements are held until the entire transaction is complete. This is done in order to allow easy rollback of a transaction, if necessary

**What do to if blocking is taking a long time?**

* Most blocking locks go away soon
* But if a blocking lock does not go away, and it is preventing one or more users from performing necessary tasks then find the culprit (SPID) and KILL the process
* Note: killing the blocking SPID will cause the current transaction to rollback and
* Appropriate indexes have a great deal of control on blocking because the quicker that SQL Server can find the data it is looking for, the less time locks have to be in place

**Demo of Blocking and deadlock via SQL Server SSMS**

**Script used to identify the blocking query**

SELECT  
db.name DBName,  
tl.request\_session\_id,  
wt.blocking\_session\_id,  
OBJECT\_NAME(p.OBJECT\_ID) BlockedObjectName,  
tl.resource\_type,  
h1.TEXT AS RequestingText,  
h2.TEXT AS BlockingTest,  
tl.request\_mode  
FROM sys.dm\_tran\_locks AS tl  
INNER JOIN sys.databases db ON db.database\_id = tl.resource\_database\_id  
INNER JOIN sys.dm\_os\_waiting\_tasks AS wt ON tl.lock\_owner\_address = wt.resource\_address  
INNER JOIN sys.partitions AS p ON p.hobt\_id = tl.resource\_associated\_entity\_id  
INNER JOIN sys.dm\_exec\_connections ec1 ON ec1.session\_id = tl.request\_session\_id  
INNER JOIN sys.dm\_exec\_connections ec2 ON ec2.session\_id = wt.blocking\_session\_id  
CROSS APPLY sys.dm\_exec\_sql\_text(ec1.most\_recent\_sql\_handle) AS h1  
CROSS APPLY sys.dm\_exec\_sql\_text(ec2.most\_recent\_sql\_handle) AS h2  
GO

Deadlock scripts:

T1:

Select \* from TableOne

Select \* from TableTwo

--Session - Transaction #1

--As we execute the update, T1 has an exclusive lock (X) on TableOne

BEGIN TRAN

UPDATE TableOne

SET FNAME = 'MARY'

WHERE ID = 1

--When executing this update, it is blocked cuz T2 has exclusive lock (X) on TableTwo

BEGIN TRAN

UPDATE TableTwo

SET FNAME = 'SAM'

WHERE ID = 1

COMMIT TRANSACTION

T2:

Select \* from TableOne

Select \* from TableTwo

--Session2 - Transaction #2

--As we execute the update, T2 has an exclusive lock (X) on TableTwo

BEGIN TRAN

UPDATE TableTwo

SET FNAME = 'RANDOLPH'

WHERE ID = 1

--When executing this update, it is blocked cuz T2 has exclusive lock (X) on TableOne

BEGIN TRAN

UPDATE TableOne

SET FNAME = 'BOB'

WHERE ID = 1

COMMIT TRANSACTION