SQLskills Immersion Event

IEPTO1: Performance Tuning and Optimization

Module 3: Locking and Blocking

Kimberly L. Tripp

Kimberly@SQLskills.com

Overview

- **The anatomy of a data modification**
- **Locking and blocking**
  - Granularity
  - Escalation
  - Duration
- **Troubleshooting locking behavior**
  - Blocking situations
    - Detecting and avoiding
  - Deadlock situations
    - Detecting and avoiding

Anatomy of a Data Modification

- **User/application sends an UPDATE query**
  - The update is highly selective (only 5 rows)
- **Indexes exist to aid in finding these rows efficiently**
- **The update is a SINGLE statement batch therefore this is an IMPLICIT transaction**
  - Transactions can be ‘explicit’ or ‘implicit’
  - Explicit transactions are controlled by the user
    - Started with BEGIN TRAN
    - Ended with COMMIT TRAN or ROLLBACK TRAN
  - Implicit transactions are created internally by SQL Server and committed automatically when the operations complete
    - And obviously rolled-back if something goes wrong

Transaction Control
Do Your Developers Understand the Requirements?

- **Session Settings**
  - SET XACT_ABORT
- **Transaction Termination**
  - Resource Error
  - User Error
- **Transaction Mode**
  - Auto-commit transaction (default)
  - Explicit transaction (user-defined)
    - `BEGIN TRANSACTION`
    - `COMMIT TRANSACTION`
  - Implicit transaction MODE (same as Oracle's default)
    - Can be set with a session-level setting
    - `SET IMPLICIT_TRANSACTIONS ON`
  - Batch scoped transactions (2005+) with MARS enabled on client

And, what about Error Handling?

- None – they believe ALL transactions are ALWAYS atomic?
- Old-school if @@error…
- TRY … CATCH

Developers MUST:

- Understand session settings
- Understand transaction control
- Understand error handling

See extra resources at the end of module

Tips to Minimize Blocking

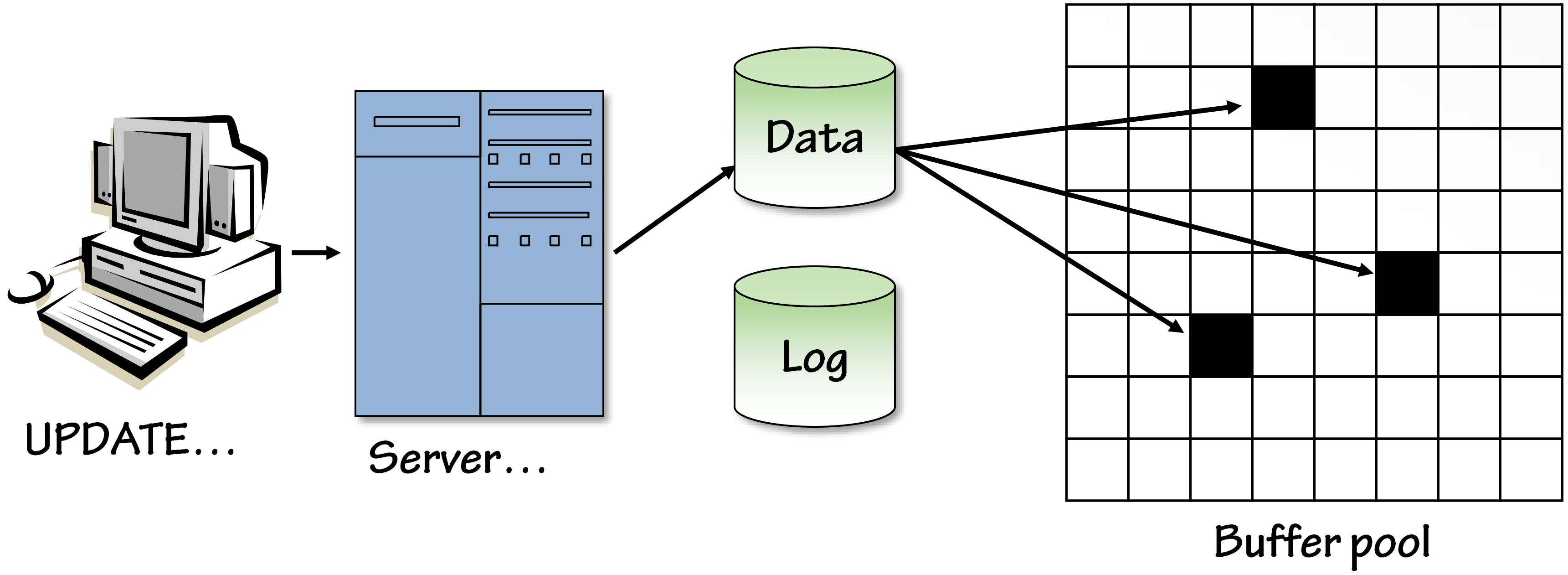
- **Write efficient transactions; keep them short and in one batch**
  - More details in this module's "side discussion" at the end of the deck
- **Do not allow interaction in the midst of the batch**
- **Use indexes to help SQL Server find, and lock, only the necessary data**
  - More details in modules on indexing
- **Use a flavor of versioning or maybe even NOLOCK**
  - More details in the versioning module
- **Consider estimates for long-running queries and/or migrating data to a secondary analysis server**

- **In summary: locking becomes problematic "blocking" when long-running [inefficient] and conflicting transactions execute**
  - Shorter transaction times mean less potential blocking…

Anatomy of a Data Modification

- **Server receives the request and locates the data in cache OR reads the data from disk into cache**
  - Since this is highly selective only the necessary pages are read into cache (maybe a few extra but that's not important here)
  - Let's use an example where the 5 rows being modified are located on 3 different data pages

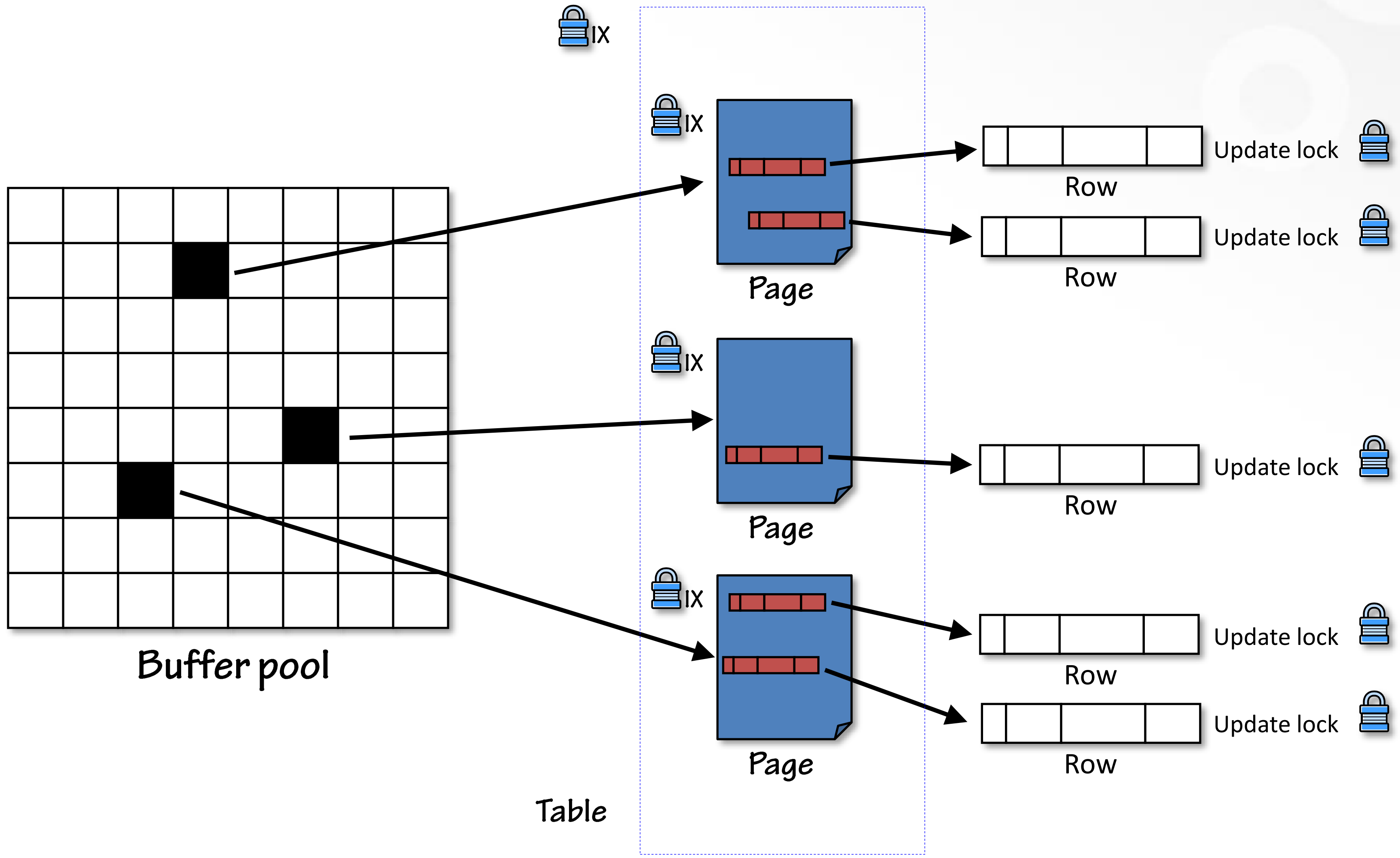
What it looks like: Data Reading From Disk



Anatomy of a Data Modification

- **SQL Server proceeds to lock the necessary data**
  - Locks are necessary to give a consistent point FOR ALL rows from which to start
  - If any other transaction(s) have ANY of these rows locked we will wait until ALL locks have been acquired before we can proceed
    - Locks are initially taken to stabilize the rows and then upgraded to exclusive locks
  - In the case of this update (because it's highly selective and because indexes exist to make this possible) SQL Server will use row level locking

What It Looks Like: Acquiring Locks

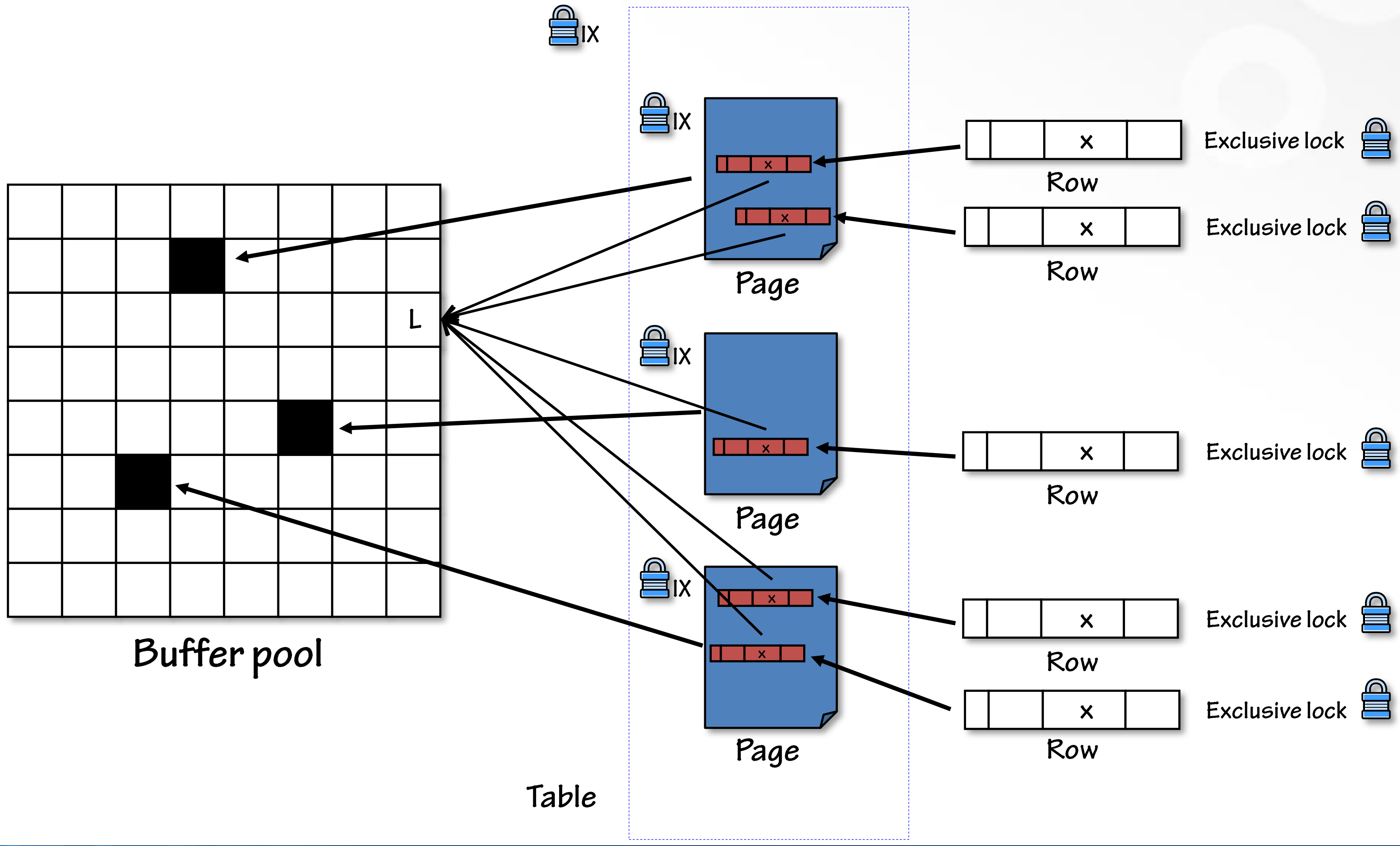


notes for prior slide

Anatomy of a Data Modification

- **The rows are locked but there are also 'intent' locks at higher levels to make sure other larger locks (like other potentially conflicting page or table level locks) are not attempted and then fail**
  - This transaction holds the following locks:
    - 5 update row-level locks
    - 3 intent-exclusive page-level locks
    - 1 intent-exclusive table-level lock
  - The connection also holds a shared database-level lock
- **And if indexes are accessed/used then there might be additional locks required – to read the data – they are not significant here**

What It Looks Like: Modifications



notes for prior slide

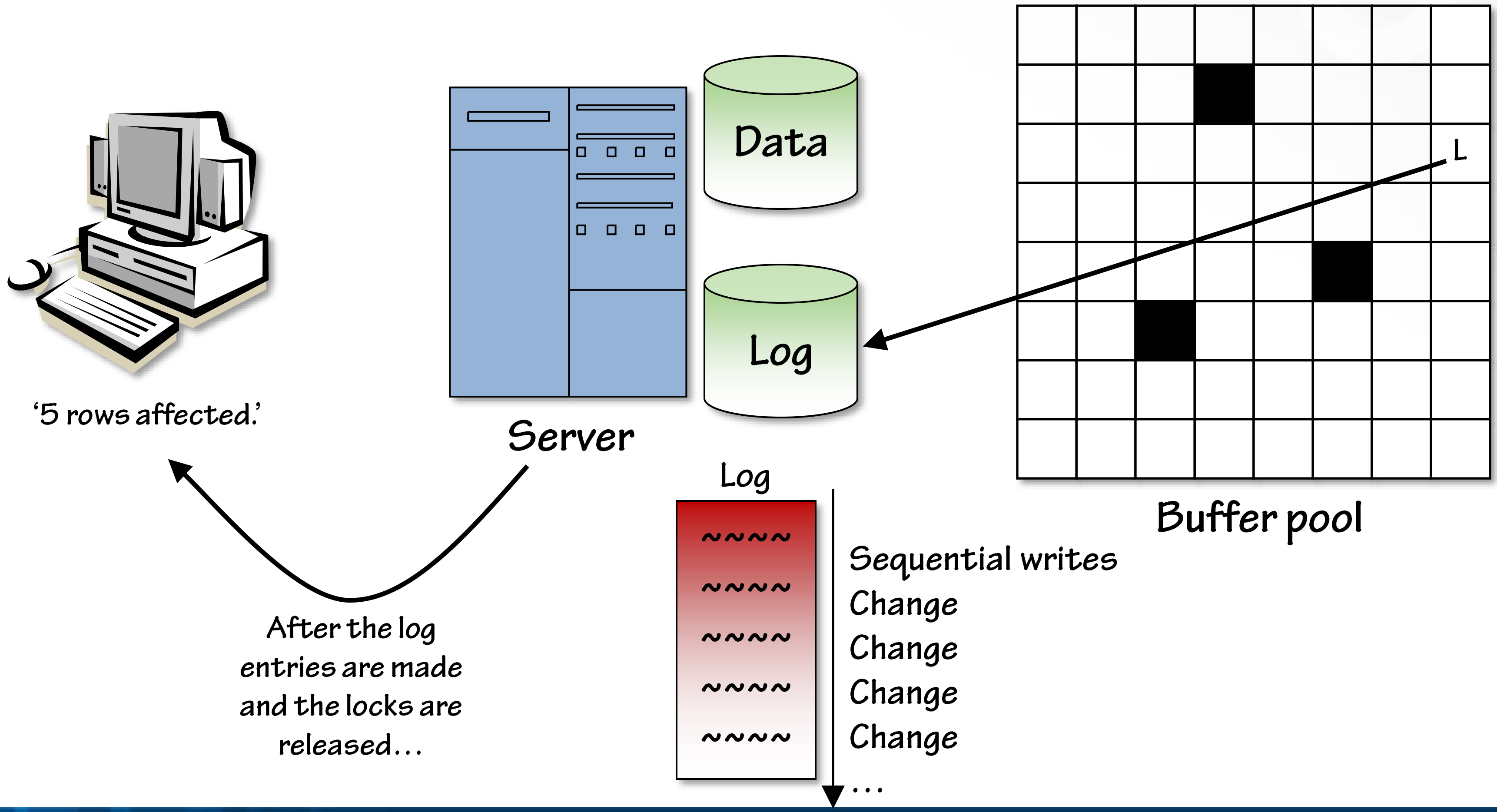
Anatomy of a Data Modification

- **SQL Server can now begin to make the modifications**
- **For EVERY row the process will include:**
  - Change to a stricter lock (eXclusive lock)
    - An update lock helps to allow better concurrency by being compatible with other shared locks (readers). Readers can read the pre-modified data as it is transactionally consistent
    - The eXclusive lock is required to make the change because once modified no other reads should be able to see this un-committed change
  - Make the modification (in cache)
  - Log the modification to the transaction log pages (also in cache)

Anatomy of a Data Modification

- **Finally, the transaction is complete (ONLY when @@trancount = 0)**
- **This is the MOST critical part (the key to Durability)**
  - All rows have been modified
  - There are no other statements in this transaction (and it's an auto-commit / implicit transaction)
- **Steps are:**
  - Write all log records for the transaction to the transaction log ON DISK (forced write-through to disk) = durable
    - This forces all of the transaction log up to the point of the COMMIT TRAN log record to be written to disk, regardless of which transaction it is for
  - Release all locks held by the transaction
  - Acknowledge the commit to the user/application:
    - A message may come back (e.g. 5 rows affected but not when NOCOUNT ON)
    - @@TRANCOUNT = 0 (this is what ***really*** defines the END of a transaction)

What It Looks Like: Committing


'5 rows affected.'
Server
Data
Log
L
Buffer pool
Log
~~~~
~~~~
~~~~
~~~~
~~~~
Sequential writes
Change
Change
Change
Change
...
After the log entries are made and the locks are released...
~~~~

So Now What?

- The transaction log ON DISK contains a record of the changes made to the database by the transaction
- The data pages in the buffer pool reflect the changes made to the database by the transaction
- When do the up-to-date data pages get written from buffer pool into the data files on disk?

Checkpoint

Anatomy of a Data Modification: Where Are We At?

- **Optimization**
  - Data is very random
  - Log is sequential
- **Locks**
  - Granularity
  - Duration
  - Escalation
- **Transactions**
  - Can make a mess of things if you don't know what you're doing…

- **NOTE: Paul will be talking more about logging, recovery, log records, and checkpoints in the logging module**

Locking

- **SQL Server pre-2005 accomplishes isolation semantics through locking only**
- **SQL Server 2005 introduced versioning**
  - 4 possible states of isolation for your database based on either:
    - Locking only
    - Versioning + locking
      - Locking is always used for writers
      - Versioning ***can*** be used for readers
  - With versioning enabled you still have locking, just more sparingly
    - Writers lock (for other writers)
    - Intent locks, metadata locks, other locks are same

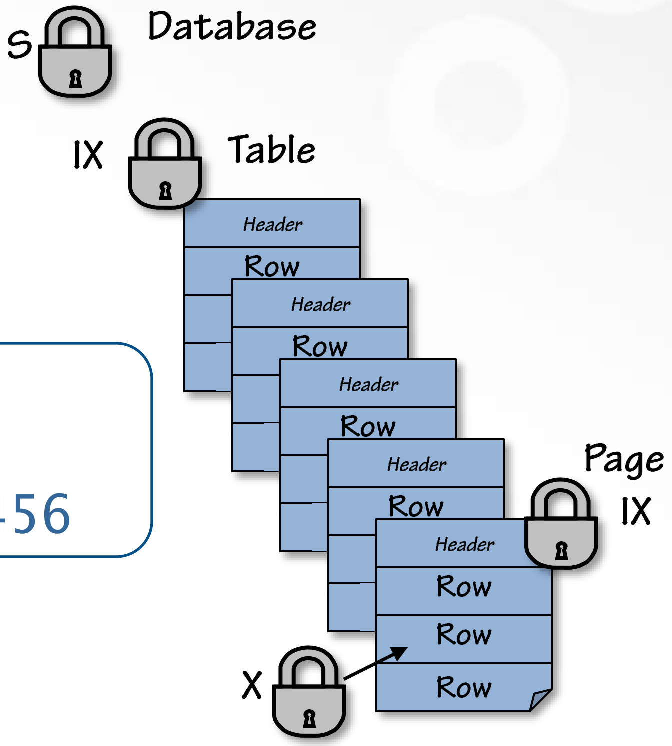
Lock Hierarchy

On connect, SQL Server allows access to the database and gives a Shared database lock to the connection.

Imagine the user submitting a query to modify a single row:

```
UPDATE dbo.Products
      SET Amount = 40.45
          WHERE ProductId = 112456
```

To perform the update SQL Server needs to first access the table and lock it with an IX (Intent eXclusive) lock.



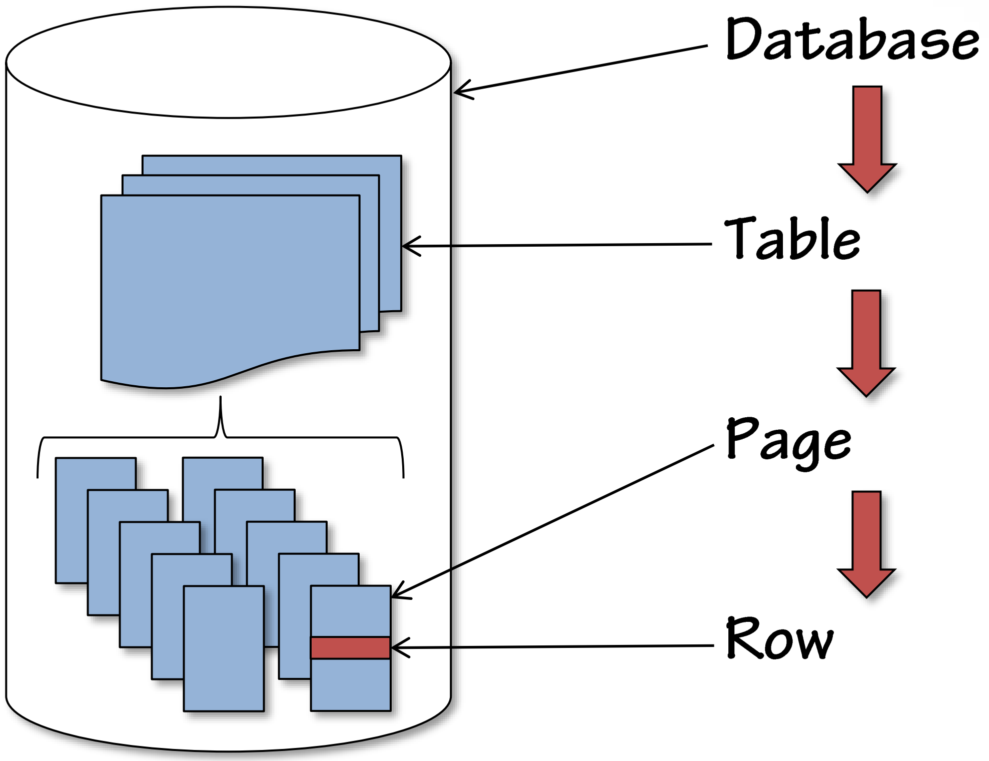
Second, SQL Server needs to find the page on which the row(s) exist and lock them with IX locks. If an index exists this can speed the request and limit the number of pages accessed.

Finally, an eXclusive lock is acquired on the row(s) that need to be modified.

hidden slide
w/extra details

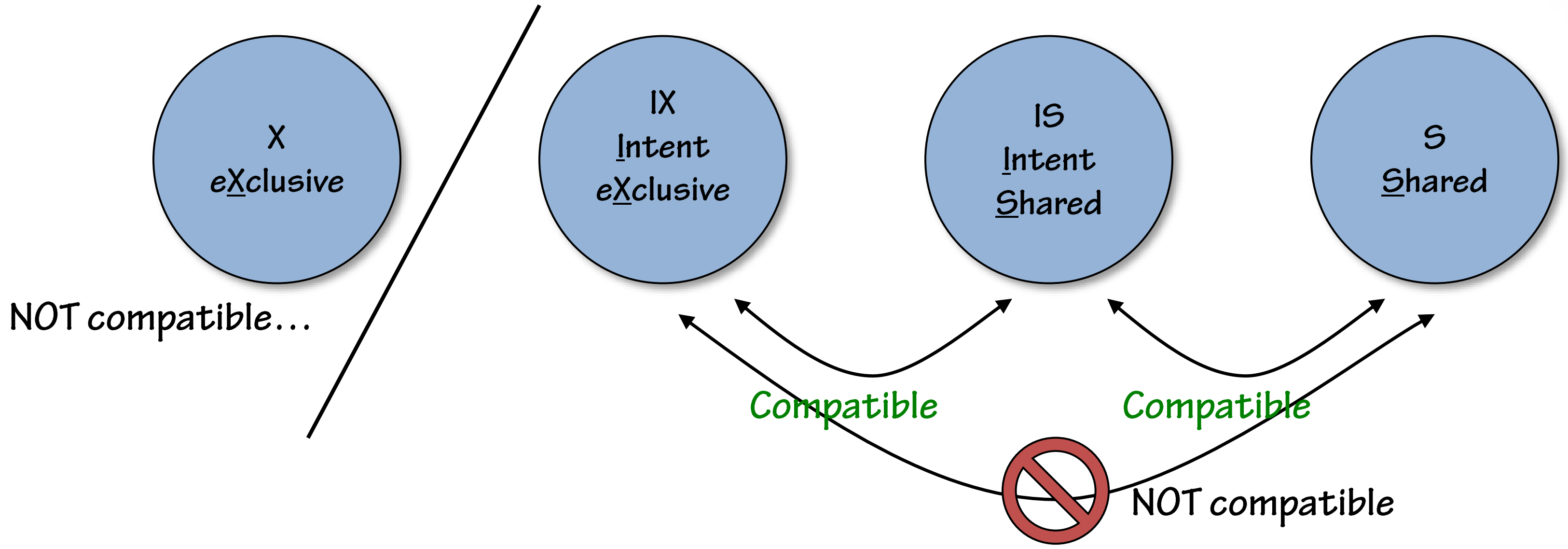
Lock Hierarchy

- Another look at basic lock hierarchy in SQL Server



Page and Table Level Locks
Intent Locks

- Shared: only readers on the table
- Intent Shared: reader with a shared page or row level lock at a lower granularity
- Intent eXclusive: writer with an exclusive page or row level lock at a lower granularity
- eXclusive: only ONE writer on the table



Row Locks
Shared, Update, and eXclusive

- **Shared: many readers**
- **Update: reading with the intent to modify but has NOT yet modified**
  - Allows better concurrency
- **eXclusive: has made a modification and the transaction is still pending**

Requested		Lock HELD		
		Shared	Update	eXclusive
	Shared	Granted	Granted	WAIT
	Update	Granted	WAIT	WAIT
	eXclusive	WAIT	WAIT	WAIT

Shared, Update, and eXclusive

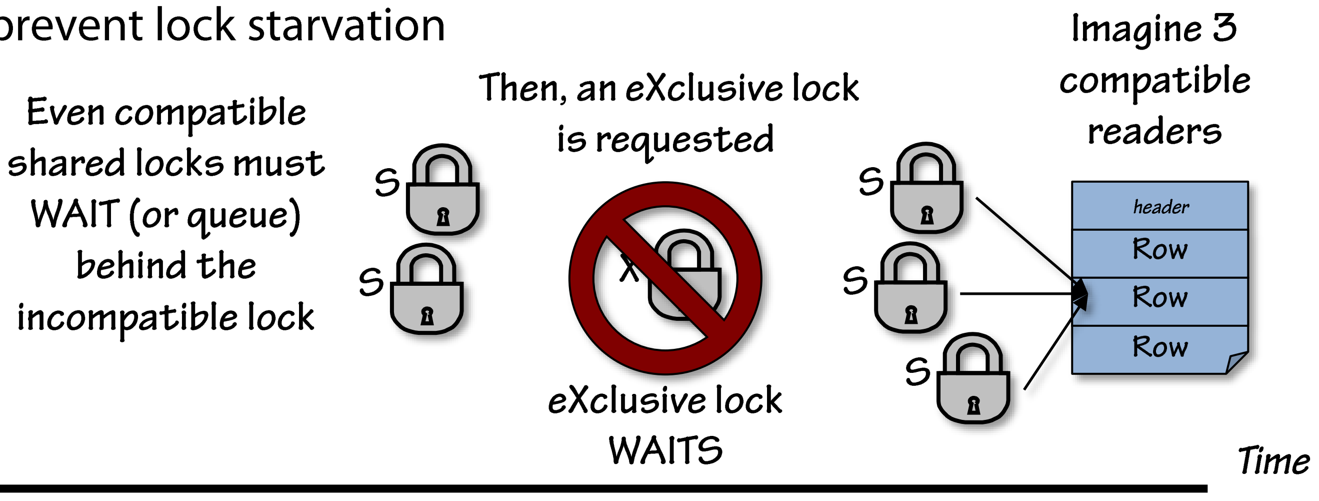
- **By DEFAULT (read committed [2] *locking*)**
  - Shared locks: read locks held until the resource has been read and processed
    - These locks are released almost immediately.
  - eXclusive locks: locks held for writers from just before the transaction modifies the data
    - These locks are not released until the owner (the transaction) has completed (i.e. committed).
  - Update locks: acquired at the beginning of the statement to isolate all of the needed rows for an update to guarantee a consistent starting point!
- **In other isolation levels, the behavior of locks can be different; we'll cover this in the versioning module**
  - Read uncommitted [1] – row locks are not used
  - Repeatable reads [3] – shared locks are used and held
  - Serializable [4] – key range locks and other resource (table) locks are used/held
  - Read committed [2] *versioning* and Snapshot Isolation [5] use versioning

Schema Locks

- **Two “types” of “schema” locks (confusing)**
  - Schema locks at the object’s schema level (meaning table-level)
  - Schema locks at the schema container level (meaning schema.object)
    - Transferring an object from one schema to another, locks the entire destination schema (see http://bit.ly/260rWol for an example)
- **Schema-stability (SCH-S)**
  - Prevents the schema changing or IAM pages being added to the IAM chain
  - Sometimes used when performing allocation order scan
- **Schema-modification (SCH-M)**
  - Used when the an object’s schema must change
    - Changing the definition of the object
    - Maintenance: creating an index / rebuilding an index / partition switching
    - sp_recompile *tablename* (requires a SCH-M on the table)
      - NOTE: sp_recompile *procedurename* does NOT
  - Not compatible with any other lock type – think of it as a super-exclusive lock
  - SQL Server 2014 has a low priority lock wait option for partition switching or index rebuilds (this can DRAMATICALLY reduce long blocking chains)
    - http://tinyurl.com/kc4g7dq

Blocking – Is it Really a Problem? It depends…

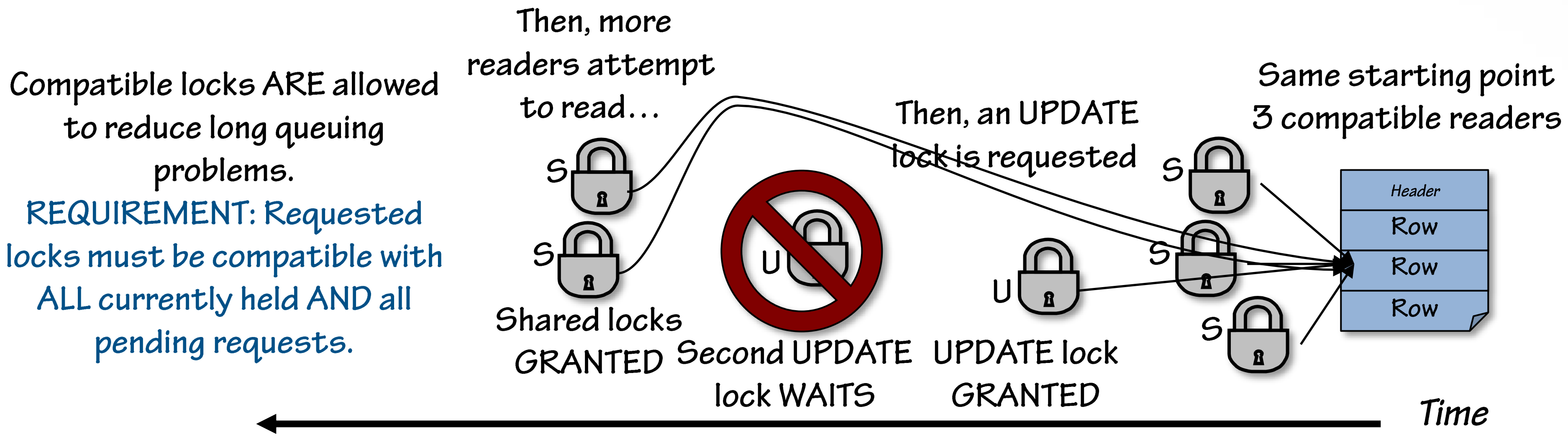
- Locks guarantee consistency
- First incompatible lock request waits
- Once an incompatible lock request is made then pending locks will wait even if they're compatible with locks currently held
  - Special case WHEN the requested lock IS compatible WITH ALL pending requests…
- Primary reason for the locks to WAIT
  - Why? To prevent lock starvation


Even compatible shared locks must WAIT (or queue) behind the incompatible lock
S
S
Then, an eXclusive lock is requested
X
eXclusive lock WAITS
S
S
S
Imagine 3 compatible readers
header
Row
Row
Row
Time

Relaxed FIFO
To Reduce Really Long Blocking Chains

- http://blogs.msdn.com/b/psssql/archive/2009/06/02/sql-server-lock-manager-and-relaxed-fifo.aspx (http://bit.ly/NWAyVC)
- An update is incompatible with another update
- The requested (second) update will wait behind the current update…
- What about shared locks if ONLY update locks? This is OK!

BUG / NOTE: In 2008 R2, a SCH_S lock request was granted despite it being incompatible with the union of granted and waiting requests (when a SCH_M was waiting), which might lead to lock starvation (of the SCH_M). In 2012, the SCH_S lock request is blocked.


Compatible locks ARE allowed to reduce long queuing problems.
REQUIREMENT: Requested locks must be compatible with ALL currently held AND all pending requests.
Then, more readers attempt to read…
S
S
Shared locks GRANTED
U
Second UPDATE lock WAITS
Then, an UPDATE lock is requested
U
UPDATE lock GRANTED
Same starting point 3 compatible readers
S
S
S
Header
Row
Row
Row
Time

Nasty Blocking CHAINS

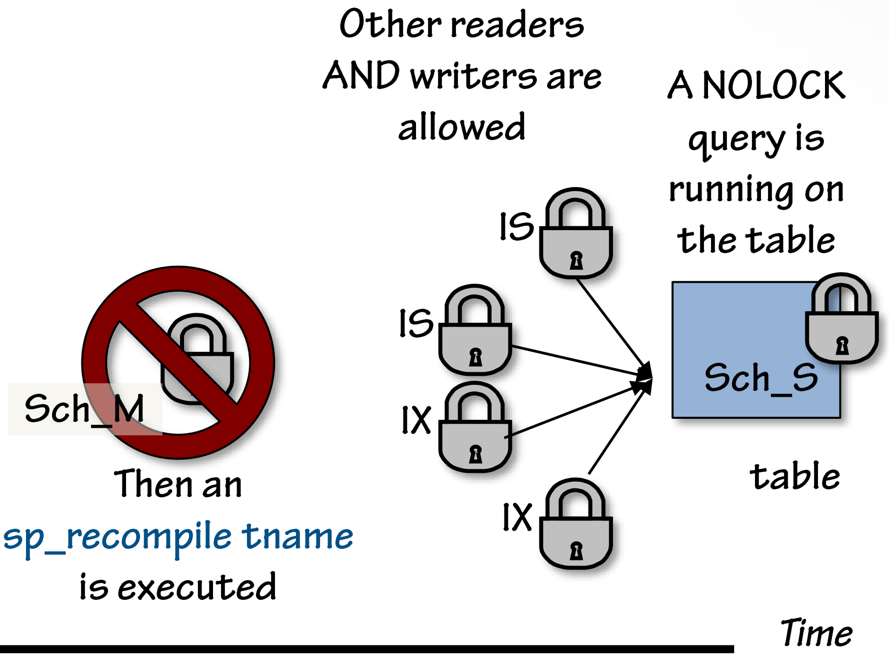
Relaxed FIFO isn't Always Enough

- Imagine someone's running a NOLOCK query against a very large table (the query takes 4 minutes to run [ok, not that nasty])
- This query's running off-hours but the standard is to use NOLOCK
- An `sp_recompile tname` is requested

Unfortunately now… everybody waits………

Relaxed FIFO only let's through the folks that are compatible with ALL pending locks (not just those that are currently held).

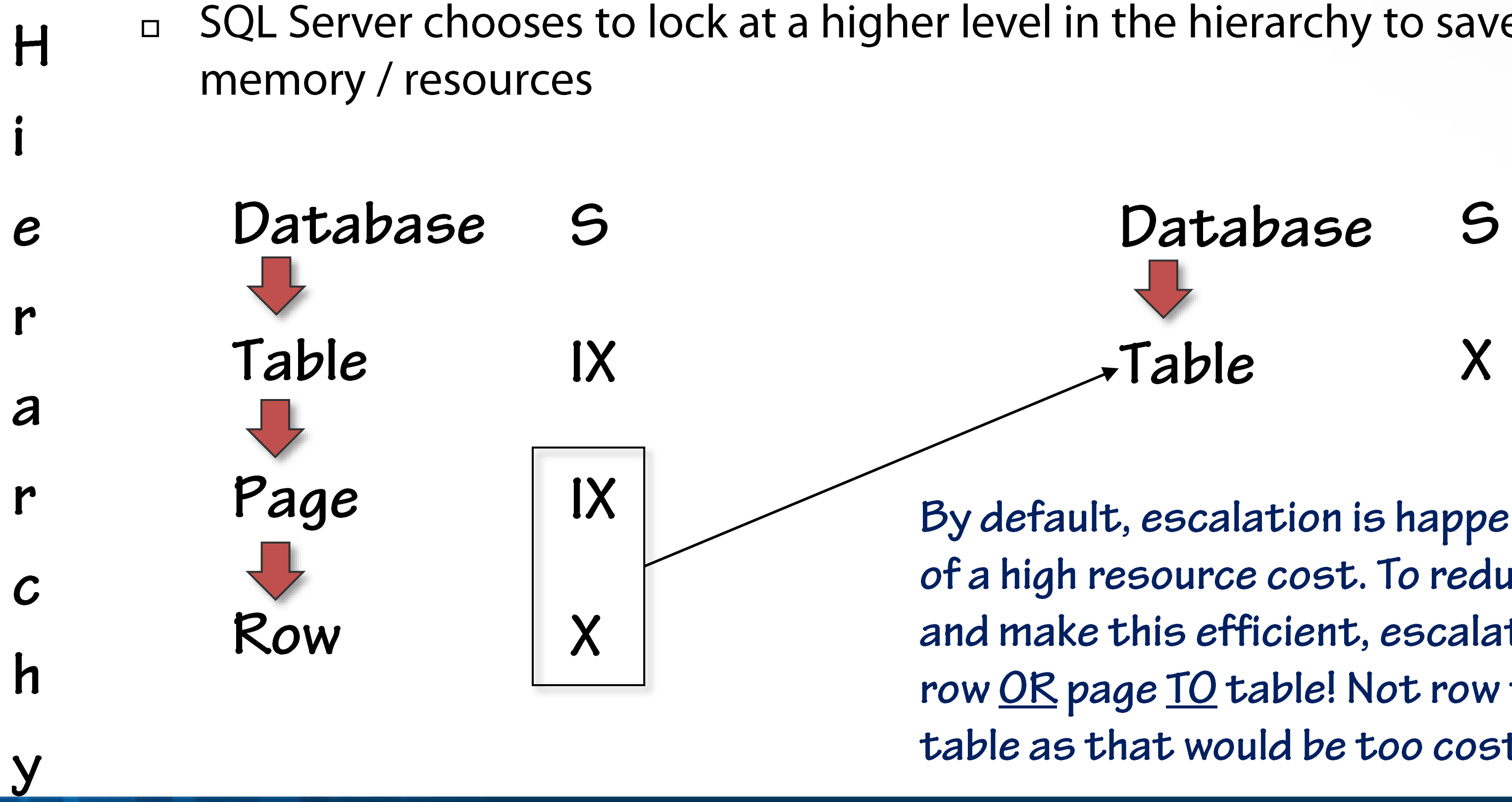
This can create terribly long blocking chains.



Other readers AND writers are allowed
A NOLOCK query is running on the table
IS
IS
IX
IX
Sch_S
table
Sch_M
Then an sp_recompile tname is executed
Time

Lock Escalation

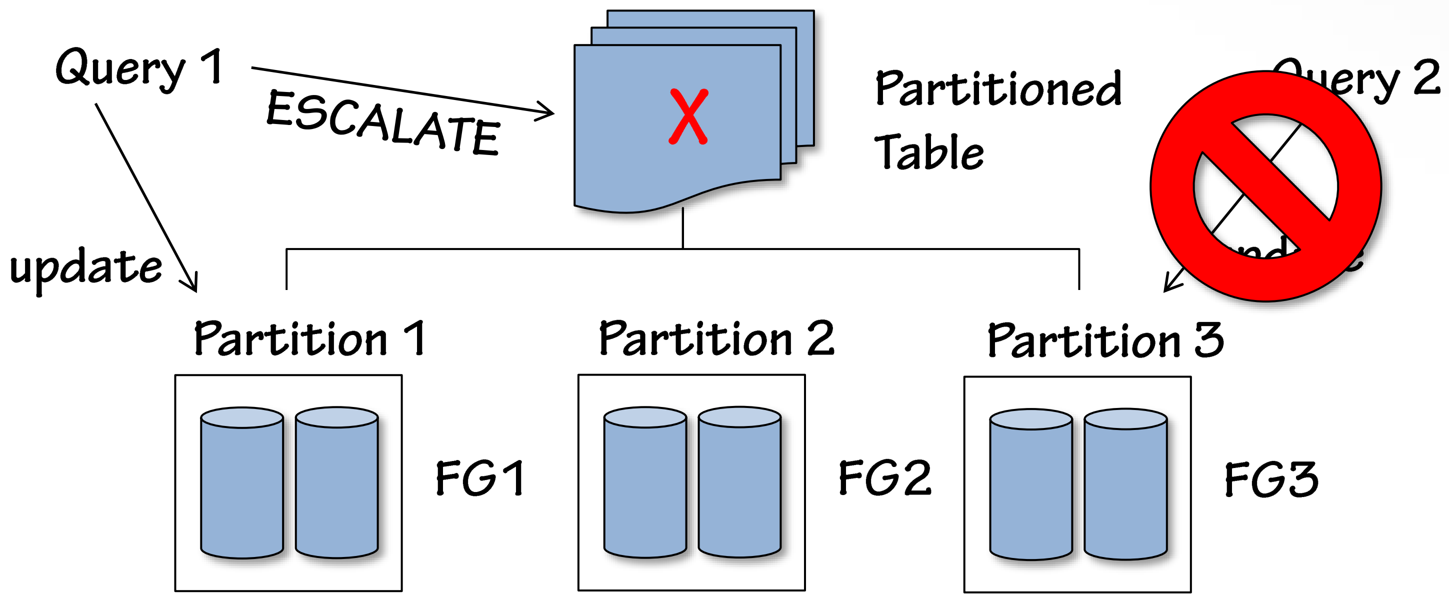
- **What is 'lock escalation'?**
  - When the cost to create / maintain the locks is too high (roughly 5000 locks for a single statement / request)
  - SQL Server chooses to lock at a higher level in the hierarchy to save lock memory / resources



By default, escalation is happening because of a high resource cost. To reduce costs and make this efficient, escalation is from row OR page TO table! Not row to page to table as that would be too costly!

Lock Escalation: The Problem

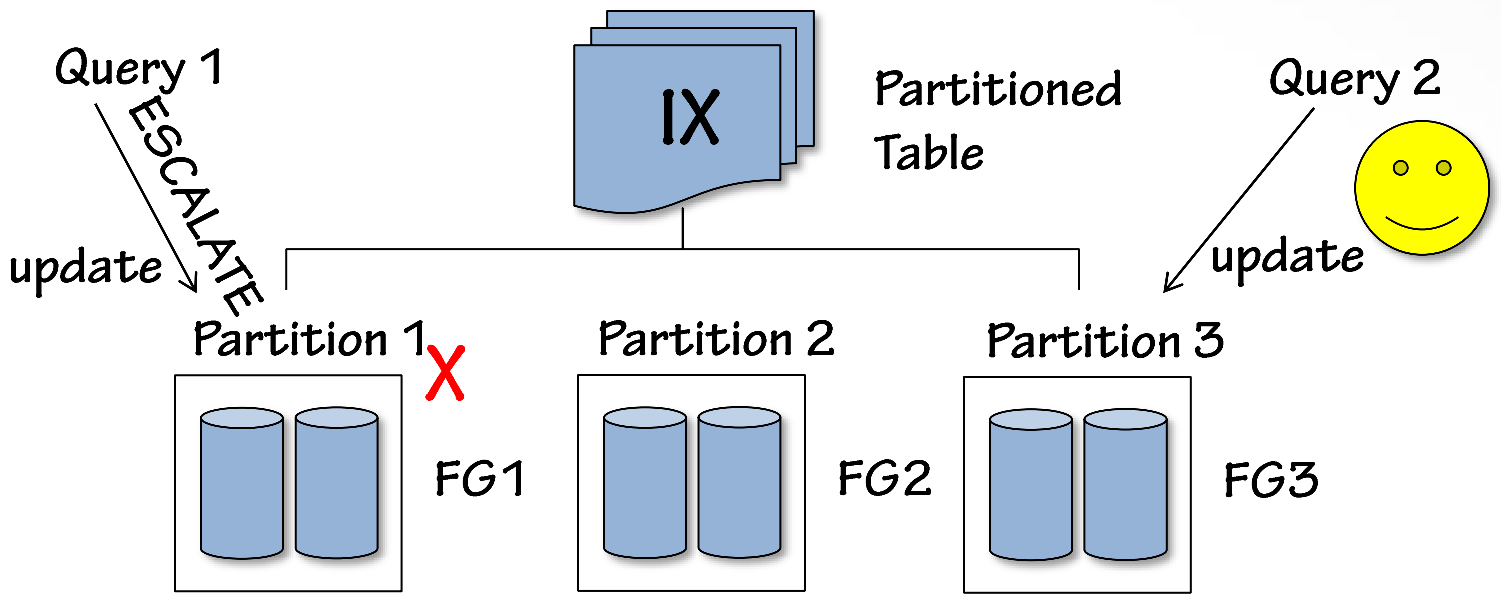
- Lock escalation on partitioned tables reduces concurrency as the table lock locks ALL partitions



- Only way to solve this before 2008 is to disable escalation

Lock Escalation: The Solution

- SQL Server 2008+ allows lock escalation to the partition level, allowing concurrent access to other partitions



- Escalation to partition level does not block queries on other partitions

Options and Syntax

- Escalation setting is per-table, set with ALTER TABLE:
  - `ALTER TABLE mytable SET (LOCK_ESCALATION = {AUTO | TABLE | DISABLE})`
- AUTO: Partition-level escalation if the table is partitioned
- TABLE: Always table-level escalation
- DISABLE: Don't escalate unless absolutely necessary

- How to tell what setting a table already has?
  - `SELECT lock_escalation_desc FROM sys.tables WHERE name = 'mytablename'`
- There is no tool (SSMS) support for ALTER TABLE

hidden slide w/extra details

Lock Escalation Summary

- **What is 'lock escalation'?**
  - When there are too many locks and SQL Server takes a lock higher in the hierarchy to save lock memory

Hierarchy		SQL Server 2005	SQL Server 2008+
Database	S	At compilation/optimization chooses row or page	At compilation/optimization chooses row or page
↓		At execution escalates:	At execution escalates:
Table	IX	Row -> table	Row -> partition/table
↓		Page -> table	Page -> partition/table
Page	IX	Never Row->page->table	By default, partition-level lock escalation is off (principle of least surprise)
↓		Why? Lock conversion too expensive and they're already out of resources…	
Row	X		

Demo

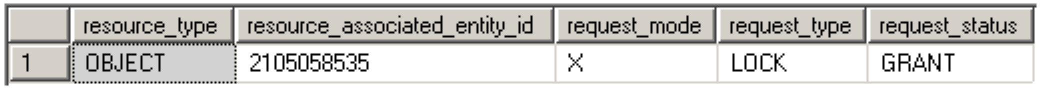
Examining locks and lock escalation

hidden slide w/extra details

Lock Escalation: Monitoring

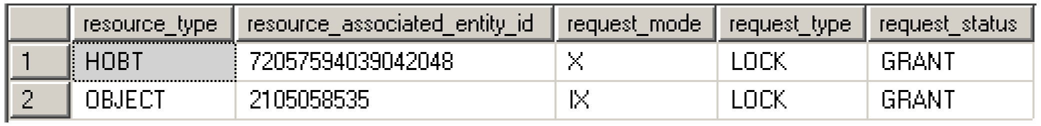
- Locking can be monitored using the DMV `sys.dm_tran_locks`

  -
    ```
    SELECT *
    FROM sys.dm_tran_locks
    WHERE [resource_type] <> 'DATABASE'
    ```
- Table level escalation will show:



	resource_type	resource_associated_entity_id	request_mode	request_type	request_status
1	OBJECT	2105058535	X	LOCK	GRANT

- Partition level escalation will show:



	resource_type	resource_associated_entity_id	request_mode	request_type	request_status
1	HOBT	72057594039042048	X	LOCK	GRANT
2	OBJECT	2105058535	IX	LOCK	GRANT

Locks – Granularity/Escalation

- **Granularity chosen at compilation**
  - Row/page or partition or table
- **Escalated at execution time if resources (to handle all of the smaller locks) are not available**
  - Row-to-table or page-to-table (or to partition if enabled on 2008+)
  - Locks do NOT escalate row-to-page-to-partition/table
  - Generally, lock escalation is desired as it results in less resources being needed to handle a query/transaction and often the query can complete faster, but at the expense of concurrency
  - SQL Server 2005+ has two trace flags, one allows lock escalation under memory pressure (1224), one does not (1211)
  - SQL Server 2008+ ONLY supports partition-level lock escalation when set at the table-level through ALTER TABLE

Controlling Lock Granularity

- **Be careful! Lock hints can negatively affect:**
  - Performance, as the query may take long to run
  - Concurrency, as the query might run faster but other users must wait… is that really faster?
- **ROWLOCK: use row level locking instead of table or page**
  - This may not be possible depending on resources and may force the query to take longer but give you better concurrency
- **PAGLOCK: similar to row level but page level instead, this may be a better choice depending on data distribution**
- **TABLOCK: better for off hour updates when concurrency might not be a problem**

Controlling Lock Type

- **UPDLOCK: requests that an update lock be used (instead of a shared lock – typically used when you plan to do modifications within the transaction)**
  - This can only be used at the row or page level
  - Update locks do not exist at any other level
- **XLOCK: requests an eXclusive lock at the row, page, or table level**
  - This can negatively affect concurrency while increasing performance for this statement/transaction
  - Used if you want to bypass update locks and disallow readers
- **For more information see "Locking Hints" in the BOL**

Controlling Wait Time

- **Check current value**
  - `SELECT @@lock_timeout`
- **Change current value**
  - `SET LOCK_TIMEOUT N`
    - Where N represents the number of milliseconds
- **Great for readers to give up waiting but what if a statement fails within a transaction?**
  - You must detect the runtime error (because this is user defined) and determine the fate of the transaction
  - The transaction will be rolled back IF you have `SET XACT_ABORT ON`
- **Skip over locked rows with READPAST**

Locking Prevents Conflicts

- **No other transactions can invalidate the data set through modifications**
- **Is this always what you want or need?**
  - Queuing applications typically want locks, if someone is modifying that row – we want to go to another not see last transactional state
  - Very volatile prices – don't want to give them the last price if it's currently being updated, so wait… (but the update's fast)
- **What about long running transactions or cases where you don't need absolute current…**

Tools for Troubleshooting Blocking

- Dynamic Management Views
- System stored procedures
  - `sp_lock [@@spid]`
- SQL Profiler
- Performance Monitor counters
- PSS Script
  - `sp_blocker_pss08` (KB#271509)
- Blocked Process Report
- Pssdiag/SQLdiag (http://diagmanager.codeplex.com/)
- SQL Server 2008+ adds
  - Performance Data Collection
  - Extended Events

Detecting Blocking

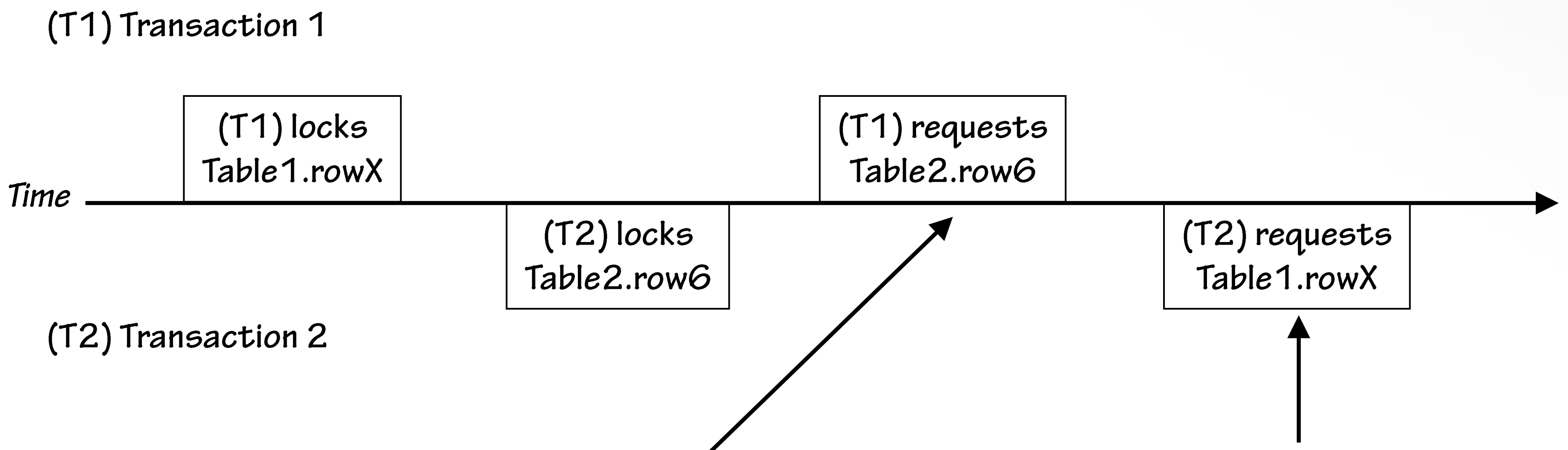
- **SSMS Reports – Top Transactions by Blocking**
- **Using DMVs**
  - `sys.dm_tran_locks` and `sys.dm_os_waiting_tasks`
  - `sys.dm_os_wait_stats`
  - `sys.dm_exec_requests`
    `CROSS APPLY sys.dm_exec_sql_text(sql_handle)`
    - Shows the last command submitted
- **Using system procedures**
  - `sp_lock/sp_lock @@spid`
  - `sp_who2` (DMV-based rewrite of `sp_who`)
    - Shows whether blocked and by who (BlkBy column)
    - LastBatch to show you when the last command was submitted from the client
    - CPUTime and DiskIO to show relative activity for transactions blocked/blocking
- **Adam Machanic's sp_whoisactive**
  - Very comprehensive and free monitoring tool

Detecting Blocking

- SQL Server Blocked Process Threshold
- Set at an instance level
  - `sp_configure 'blocked process threshold', n` (secs)
- Each spid blocked for n seconds triggers an event that can be tracked using:
  - 2005 and higher
    - Trace
    - Event Notification
    - WMI – SQL Agent Alert
    - See Jonathan's post: Using the Block Process Report in SQL Server 2005/2008
  - 2012 and higher – use Extended Events
    - See Erin's post: Capture Blocking Information with Extended Events and the Blocked Process Report

When Blocking Becomes Deadly ☺

- Two (or more) transactions request mutually desired resources in an undesirable order



This is just blocking… Which is not a problem as long as Transaction 2 completes in a timely manner…but it doesn't

This creates a <u>DEAD</u>lock (a.k.a. a "circular reference") which is infinite. SQL Server automatically detects and resolves a deadlock by choosing a victim.

Deadlocks

- **All resources have similar deadlock handling**
- **Deadlocks can be caused by**
  - Data locks: locks of rows, pages, partitions, tables, and database metadata
  - Worker threads: queued tasks waiting for worker threads can cause deadlock ('intra-query parallelism deadlock')
  - Memory: two tasks waiting for memory
  - Parallel query execution resources
  - MARS interleaving

Deadlock Monitor

- Background process in the lock manager
- Checks every 5 seconds usually, but time between checks throttles up and down based on how frequently deadlocks are being discovered
- If something enters the wait queue right after a deadlock is detected, a deadlock search will immediately begin
  - I.e. it is a pessimistic process
- If a deadlock is found, one of the participants is rolled back

Deadlock Resolution

- **By default, SQL Server chooses the least expensive victim to rollback (based on the amount of transaction log generated)**
- **If desired, you can impact this choice:**
  - `SET DEADLOCK_PRIORITY`
    - LOW (equivalent of -5)
    - NORMAL (equivalent of 0)
    - HIGH (equivalent of 5)
    - Things like shrink, reorganize are set to low automatically
  - SQL Server 2005+ allows numeric value (-10 through 10)
    - Can also set with a variable (lookup in a table)
- **The deadlock victim receives error 1205:**
  - Your transaction (<tran id>) was deadlocked with another process and has been chosen the deadlock victim. Rerun your transaction.

Detecting Deadlocks

- **Perfmon Counter: Deadlocks/Sec**
- **SQL Profiler events**
  - Select Locks/Deadlock Graph
  - Save all deadlock graphs to single file or one file per-deadlock
  - Saved in XML format with XDL extension
  - Displayed graphically in profiler
    - Or can be analyzed using XML APIs
- **Extended Events**
  - Writes deadlock graph
  - System health session tracks deadlocks
- **Trace flag 1222 (1204 and 1205)**
  - Writes deadlock info to SQL Server error log

Homework/Resources

Online Training:

- **See Jonathan Kehayias' course on Pluralsight:**
  SQL Server: Deadlock Analysis and Prevention

Online blog posts from Bart Duncan, Search: "bart duncan deadlocks"

- **Deadlock Troubleshooting, Parts 1, 2 and 3**
- http://tinyurl.com/clnwyyl (Part 1)

Avoiding Deadlocks

- **Minimize blocking**
  - More effective queries
  - More effective indexes
  - Add or remove lock hints
- **Access resources in the same order**
- `SET DEADLOCK_PRIORITY` **to lower value**
- **Add retry logic in application if error 1205 returned**
- **Consider versioning; either statement-level or transaction-level**

Review

- **The anatomy of a data modification**
- **Locking and blocking**
  - Granularity
  - Escalation
  - Duration
- **Troubleshooting locking behavior**
  - Blocking situations
    - Detecting and avoiding
  - Deadlock situations
    - Detecting and avoiding

hidden slide
w/extra details

Latches

- **Lightweight internal synchronization mechanism similar to locks but cannot be influenced externally**
- **A latch protects access to a structure in memory**
  - E.g. in memory copy of a page
- **Many types of latches**
  - Simple: e.g. update, exclusive, shared
  - More complex: e.g. keep, destroy
- **Possible to have latch waits (similar to blocking)**
- **Possible to have dead-latches (these are bugs)**

- **Covered in more detail in**
  - Paul's Pluralsight course **SQL Server: Performance Troubleshooting Using Wait Statistics** at http://pluralsight.com/training/Courses/TableOfContents/sqlserver-waits
  - Bob Ward's 2010 session at SQL PASS. Check out: http://tinyurl.com/d2bu2f6

Questions!

immerse yourself in sql server

SQLskills Immersion Event

IEPTO1: Performance Tuning and Optimization

Module 3: Locking and Blocking

Side discussion – for you / your developers!

Kimberly L. Tripp

Kimberly@SQLskills.com

SQLskills
immerse yourself in sql server

Batches

- **Parsed as a unit so if any statement fails syntax then NO statements (in that batch) are executed**
  - SQL Server proceeds to next batch
- **At execution, if a statement fails then that statement is rolled back and SQL Server exits the batch (or falls into catch block) and proceeds to the next batch**
  - **Any statement following the failed statement is NOT executed**

```
SELECT * FROM AdventureWorks.person.person
SELECT * FROM AdventureWorks.person.address
SELECT * FORM AdventureWorks.person.emailaddress
go

Error: Server: Msg 170, Level 15, State 1, Line 3
Line 3: Incorrect syntax near 'form'.
```

Transactions

- Controlled by transaction mode of session
  - Defined explicitly
  - Defined at connection level
- Allows multiple statements to be treated as a single logical unit of work
- Handled automatically across databases on the same server (i.e. instance)
- Handled through MSDTC explicitly through
  - `BEGIN DISTRIBUTED TRANSACTION`
  - `COMMIT TRANSACTION`
- Requires error handling logic

Transaction Mode

- **Auto-commit transaction (default)**
  - Statement-level implicit transaction
  - Each statement commits as a single unit
- **Explicit transaction (user-defined)**
  - `BEGIN TRANSACTION`
  - `COMMIT TRANSACTION`
- **Implicit transaction**
  - Session Level Setting
  - `SET IMPLICIT_TRANSACTIONS ON`
- **Batch scoped transactions (2005+)**
  - Only when MARS enabled on client

Transaction Termination (1)

- **Resource error: automatically handled**
  - If the statement fails due to a SQL Server resource error (for example, the transaction log fills), SQL Server maintains data integrity automatically
- **User-defined error: programmatically handled**
  - Conditionally, when your business rules (i.e. constraint violation or a lock timeout (more on this coming up)) are violated, YOU must programmatically determine the fate of the transaction

Transaction Termination (2)

- `SET ARITHABORT` **(on by default in SSMS/sqlcmd)**
  - ON – rollback statement/transaction when divide-by-zero or arithmetic overflow error occurs
  - OFF – return NULL
- `SET XACT_ABORT` **(off by default)**
  - ON – rollback statement/transaction when runtime error occurs

- **Viewing session settings:**
  - `DBCC USEROPTIONS`

  -
    ```
    SELECT *
    FROM sys.dm_exec_sessions
    WHERE session_id = @@spid
    ```
    (optionally)

Understanding Transactions

- `BEGIN TRANSACTION`
  - Begins a new transaction
- **Supports a “name” but [VERY] limited uses**
- **Increments @@TRANCOUNT**
- **Supports special “marked transactions”**

```
BEGIN TRANSACTION TransactionName
WITH MARK ['important transaction']
```

  - This does NOT control rollback points
    (you’re always rolling back THE transaction)
  - Useful for restore/recovery operations
    - `STOPATMARK [AFTER` *`datetime`*`]`
      - Includes marked transaction
    - `STOPBEFOREMARK [AFTER` *`datetime`*`]`
      - Includes all operations up to – but not including the marked transaction

Understanding Transactions

- `COMMIT TRANSACTION`
  - Finalizes the transaction
  - Decrements @@TRANCOUNT by 1
- **Does NOT commit the transaction UNLESS the @@TRANCOUNT is 0**
- **Must correspond to a BEGIN TRANSACTION**

```
BEGIN TRANSACTION
SQL statements
COMMIT TRANSACTION
```

- **Or, session must have implicit_transactions turned on. This is not recommended.**

```
SET IMPLICIT_TRANSACTIONS ON
SQL statements…
COMMIT TRANSACTION
```

(open transaction until commit)

Understanding Transactions

- `ROLLBACK TRANSACTION`
  - Aborts the transaction
- **Returns @@TRANCOUNT to 0**
- **Cancels all levels of “nested” transactions**
- **You can rollback to a savepoint and still be within the bounds of the transaction (see next slide)**
- **However, you cannot rollback to a named transaction**

```
Msg 6401, Level 16, State 1, Line 5
Cannot roll back TransactionName. No transaction or
   savepoint of that name was found.
```

- **There is ALWAYS only one transaction in the context of rollback**
- **Only savepoints can be used to undo state within a transaction**

Understanding Transactions

- `SAVE TRANSACTION` (a.k.a. savepoints)
  - Are a “stack” (without any regard for “nesting” because that doesn’t exist) and you CAN reuse the same name
- Does not affect @@TRANCOUNT
- When you ROLLBACK to a savepoint you must still commit or rollback the pending transaction
- Commonly used inside of procedural code to limit rollback effect

```
BEGIN TRANSACTION
    SAVE TRANSACTION procname_tran
    Procedural code/logic
    If problem… ROLLBACK TRANSACTION procname_tran
COMMIT TRAN
RETURN [int]
```

- A rollback will always rollback to the most recent savepoint (not necessarily the scoped savepoint, even in procedures)

Check out the sample scripts

Transaction madness and save point coding

- You MUST work with your developers to understand this…
- It will become YOUR problem!