From Locks to No Locks – Concurrency in SQL Server
Concurrency with focus on read-operations

Andreas Wolter
SARPEDON Quality Lab
MCSM, MCM, MVP
The Audience, you 😊

1) SQL-Developer?
2) App-Developer?
3) Administrator?

• Assumed knowledge for this session: Basic T-SQL and Database-Design
Agenda

• Locking & Blocking Basics
  – Lock-Starvation prevention
  – Concurrency problems
• Lock-Escalation
  – What happens and when?
• How (B)Locking can be circumvented or avoided
  – NOLOCK
    • Good idea or not?
  – Why database/index design matters
• 1205!
  – When do deadlocks occur and how to prevent them
  – The SQL Deadlock Collector & Parser
• How Snapshot Isolation changes the game

Concurrency

• The ability to execute multiple transactions simultaneously
• ACID
  • Isolation
    • One transaction cannot see or modify the data that another transaction has modified and not committed
    • Isolation is maximized when Concurrency is minimized, and vice versa
Concurrency Problems, solved by Locking

- Dirty Reads (Uncommitted Dependency)
  - Retrieving uncommitted changes that might be rolled back or modified again before being committed
- Missing Reads & Double Reads
  - Row that has yet to be read moved to a location already scanned / Same row, moved to a location not yet scanned
- Lost Updates
  - Two sessions change the same row at about the same time, with a winner and a loser, and the loser isn’t warned
- Non-Repeatable Reads (Inconsistent Analysis)
  - Previous rows updated or deleted on a repeated query
- Phantom Reads
  - New rows showing up on a repeated query in a transaction
### Isolation Levels

<table>
<thead>
<tr>
<th></th>
<th>Dirty Reads</th>
<th>Lost Updates</th>
<th>Nonrepeatable reads</th>
<th>Phantom reads</th>
<th>Concurrency model</th>
<th>Conflict Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Uncommitted</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Pessimistic</td>
<td>No</td>
</tr>
<tr>
<td>Read Committed</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Pessimistic</td>
<td>No</td>
</tr>
<tr>
<td>Repeatable Read</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Pessimistic</td>
<td>No</td>
</tr>
<tr>
<td>Serializable</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Pessimistic</td>
<td>No</td>
</tr>
<tr>
<td>Snapshot Read</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Optimistic</td>
<td>Yes</td>
</tr>
<tr>
<td>Read Committed Snap</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Optimistic</td>
<td>-</td>
</tr>
</tbody>
</table>

### Isolation and Pessimistic Concurrency Control

- Isolation is maximized by serializing access to each set of data
- Locks are used to maintain isolation
  - When locks are incompatible, the session requesting the incompatible lock must wait -> is being blocked
  - Readers block writers
  - Writers block readers
### Complete Lock Compatibility Matrix

<table>
<thead>
<tr>
<th>NL</th>
<th>SCCH-S</th>
<th>SCCH-M</th>
<th>S</th>
<th>U</th>
<th>X</th>
<th>IS</th>
<th>S</th>
<th>U</th>
<th>IX</th>
<th>SIX</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>VL</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SCCH</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SCCH-M</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>S</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>U</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>X</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IX</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IXU</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IXS</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IXCU</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IXCS</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IXCSU</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IXCSM</td>
<td>N</td>
<td>C</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Source: Books Online
Better readable version by Mark Broadbent:

### Abbreviated Lock Compatibility Matrix

<table>
<thead>
<tr>
<th>Requested mode</th>
<th>Existing granted mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intent shared (IS)</td>
<td>Yes</td>
</tr>
<tr>
<td>Shared (S)</td>
<td>Yes</td>
</tr>
<tr>
<td>Update (U)</td>
<td>Yes</td>
</tr>
<tr>
<td>Intent exclusive (IX)</td>
<td>Yes</td>
</tr>
<tr>
<td>Shared with intent exclusive (SIX)</td>
<td>Yes</td>
</tr>
<tr>
<td>Exclusive (X)</td>
<td>No</td>
</tr>
</tbody>
</table>
Resource Types that can be Locked

- DATABASE
- OBJECT
- PAGE
- KEY
- RID
- HOBT
- METADATA
- EXTENT
- FILE
- ALLOCATION_UNIT
- APPLICATION (Tip: Can be used to serialize access to code instead of data!)
- Plus several subtypes depending on actual type

HOW (B)LOCKING CAN BE CIRCUMVENTED OR AVOIDED
How Blocking can be circumvented

- **NOLOCK Table hint**
  - Equivalent to READ UNCOMMITTED
  - Uncommitted data problems
    - False or missing Updates
    - Missing data
    - Double data
  - Check for error 601
  - **Could not continue scan with NOLOCK due to data movement.**

- Timeout
  - By default no timeout by SQL Server
  - Check for error 1222
  - No Auto-Rollback!

- Readpast
  - Blocked rows are skipped
  - Best used for queues
Circumventing blocking

DEMO

NOLOCK reads just uncommitted data
Is that really all there is to it? Demo
NOLOCK does not honor or create locks? really?

Demo

But the best option is: Picking the right indexes + writing good queries.

Demo

Circumventing blocking with an index
LOCK ESCALATION

The cost of Concurrency vs Locking

Lock Escalation

- Kicks in, when a statement holds more than 5000 Locks on an object.
  - If this does not work, it will be tried again all other 1250 locks

- Max 60% Memory will be used for locks.
- At 40 % Memory usage escalation occurs as well. (can be controlled by “locks” configuration, but deprecated)

- Can also be influenced by Traceflags 1211 and 1224
    - 1224: Disables lock escalation based on the number of locks. However, memory pressure can still activate lock escalation.
    - 1211: trace flag 1211 prevents escalation in every case

- Escalation always occurs from
  - Row/Key or Page -> Table or from Row/Key/Page -> Partition
  - Not from Key -> Page or from Partition -> Table
  - That’s a myth!

Lock Escalation & Partitioning Demo
Deadlocks

- Variances:
  - cycle deadlock
  - conversion deadlock
This is what it looks like in Profiler/Xevents

Deadlock discovery

- **> SQL 2005** with Traceflags – trashing the error log
  - 1222 - Returns information in an XML-like format
  - 1204 - Focused on the nodes involved in the deadlock.
  - 1205 - very verbose
- **>= SQL Server 2008** via „system-health“ Extended Event session – for free.
Traceflag 1222 Dump in Errorlog

SQL Deadlock Collector & Parser

- Free download
  - sqldeadlockcollector.codeplex.com
  - SQL Server 2008 and => 2012
- Zero extra tracing overhead
- Parses first 2 involved Processes into respective columns
- Views and Reports for Quick Overview
- Includes query Plans
Demo

Discovery with Xevents & Analysis with SQL Deadlock Collector & Parser

**DEADLOCK**

---

**Deadlock Monitor**

- standard interval is 5 seconds.
- When the lock monitor thread finds a deadlock, detection interval drops down to 100 milliseconds, depending on the occurring deadlocks.
- When a deadlock was detected, another deadlock detection will be executed at the next lock wait.
Deadlock-avoidance/minimization

- Recommended order (the later the more critical)

  (1) Consider Isolation Level
  (2) Uniform sequence for accessing objects in Dev-guidelines
  (3) Covering Indexes
  (4) Early Locking (already at Select)
  (5) Changing Isolation Level – more in next topic

- If this is not sufficient:
  (6) Define victim/winner (DEADLOCK_PRIORITY)

- In complex environments with various access paths deadlocks cannot be guaranteed to be avoided!
  Therefore, dear developers: develop defensively, check for errors.

OPTIMISTIC CONCURRENCY CONTROL
Optimistic Concurrency Control using Snapshot isolation and read committed snapshot isolation (RCSI)

- “readers do not block writers, and writers do not block readers”
- Based on row versioning
  - version store in Tempdb
- SI avoids
  - Dirty reads
  - non-repeatable reads
  - phantom reads
  - , without the need of L0cks: readers receive a snapshot of the data, as it existed at the start of the transaktion

- RCSI avoids:
  - Dirty reads
  - but not
  - non-repeatable reads or phantoms

The difference between SI and RCSI

- read committed snapshot isolation (RCSI)
  - queries return committed data as of the beginning of the current statement
- snapshot isolation (SI)
  - queries return committed data as of the beginning of the current transaction.
Attention

- SELECTS can be slowed down due to version-chains
- Tempdb is used heavily (depending on data change rate)
  - for all UPDATE, DELETE, and some Insert statements

- Row versioning enlarges every row by 14 bytes.

- In case of conflicts, UPDATE(!)-operations could error/be rolled back

- If Tempdb runs out of space, Select-statements fail (not the Updates)!
In-Memory OLTP using MVCC

- MVCC: Multi-versioned concurrency control
- Optimistic concurrency control
- Versioning using timestamps
- Combination of snapshot-based isolation and conflict detection
- NO “READ COMMITTED”
- Memory-optimized tables use snapshots for all transaction isolation levels
  - SNAPSHOT
  - REPEATABLE READ
  - SERIALIZABLE
Conclusion and further thoughts

- Developers should have a good understanding for transactional processing of SQL Server
  - Difference between Locking and Blocking
  - Blocking and Deadlocking

- Design matters!
- Also consider asynchronous processing (via Service Broker)

- Monitor systems (preferably without Deadlock Traceflag 1222 – using system_health) and draw correct conclusions

Ressources Deadlocking

- SQL Deadlock Collector & Parser
  - https://sqldeadlockcollector.codeplex.com/

- How to import Extended Events session file target and parse deadlock-graph

- Free Deadlock-Collector & -Parser, based on Extended Events session system_health

- Detecting and Ending Deadlocks

- Deadlock Troubleshooting, Part 1-3
Ressources

- Lock Compatibility

- Lock Modes

- SQL Server Transaction Locking and Row Versioning Guide

- Application Locks / sp_getapplock

- Lock Escalation

- Working with tempdb in SQL Server 2005

- SQL Server 2005 Row Versioning-Based Transaction Isolation

Contact:
andreas.wolter@SarpedonQualityLab.com
Blog: www.insidesql.org/blogs/andreaswolter/
Linkedin: www.linkedin.com/in/andreaswolter
Twitter: @AndreasWolter

Thank You

Andreas Wolter
Sarpedon Quality Lab: Your Specialist for Database-Systems based on SQL Server Technologies

We are **one of only 2 companies worldwide** who have reached the **highest technical certifications** from Microsoft for **SQL Server 2008** as well as **SQL Server 2012**!

We love to support you and use our know-how to your advantage.

our **Services** cover:

- SQL Server Health checks
- Performance Analysis & Tuning
- Disaster-Recovery & SLA-Compliance-Checks
- Security-Checks
- Data Rescue in case of corruption
- Architecture-Planning, Consulting and Implementation

**Training:** [SQL Server Master-Class](#)

Ask us: [info@Sarpedon.de](mailto:info@Sarpedon.de), [en.SarpedonQualityLab.com](http://en.SarpedonQualityLab.com)