Second Edition

Tuning SQL Server®

Boost Performance of your SQL Server® Instances!

Learn how to maintain indexes, handle log files, and more!

Artemakis Artemiou

Tuning SQL Server®

(Second Edition)

ARTEMAKIS ARTEMIOU

PUBLISHED BY
Artemakis Artemiou

Publication Date: May 2017.

About the Author

Artemakis Artemiou is a Senior SQL Server and Software Architect, Author, and a former Microsoft Data Platform MVP (2009-2018). He has over 15 years of experience in the IT industry in various roles. Artemakis is the founder of SQLNetHub and TechHowTos.com. Artemakis is the creator of the well-known software tools Snippets Generator, DBA Security Advisor and In-Memory OLTP Simulator. Moreover, he is the author of many eBooks on SQL Server. Artemakis currently serves as the President of the Cyprus .NET User Group (CDNUG) and the International .NET Association Country Leader for Cyprus (INETA). Artemakis's official website can be found at aartemiou.com. You can follow Artemakis on Twitter at https://twitter.com/artemakis.

Some of the online channels where you can find Artemakis are:

- SQLNetHub
- Official Website
- The SQL Server and .NET TV
- TechHowTos.com
- Twitter
- Cyprus .NET User Group

Contact the Author
You can contact the author using the contact methods on www.aartemiou.com and www.sqlnethub.com

SQLNETHUB

To all of those who never stop seeking knowledge,
who never stop sharing knowledge without
expecting anything in return, for those
who just want to help their fellow man.
-A.A.

Introduction

I started working with SQL Server and .NET (C#) 15 years ago. Since then it has been quite a journey! Each release came - and still comes - with exciting new features enabling us to do more and more! Every time waiting for that new built, in order to start testing it, exploring it, learning it, as soon as it becomes available. The possibilities are endless! The only limit is your creativity!

The massive interaction with the SQL Server community started at about seven years ago. Blogging, organizing user group events, speaking in user group meetings, conferences and other events, open-source projects related to SQL Server, guest articles, discussions on message boards/forums, and much more!

The love for technical writing and knowledge sharing urged me for adding another activity to my interaction with the community that is ***book authoring***. In order to be able to write, you first need to acquire and comprehend the specific technical knowledge. You need to explore, to experiment, to test. You need to test the limits of each new technology or feature in order to be able to fully understand its nature and capabilities.

SQL Server is a powerful data platform that includes several data management and analysis technologies that allow you to do just about anything. Since 2002 I have been exploring SQL Server in many areas. I have been deep diving into various topics of SQL server having to do mainly with: ***administration***, ***development/data access***, ***performance tuning,*** *and in the last few years****, In-Memory OLTP***. These are the four areas of SQL Server I mostly focus on and on which I base, but not limit, my interaction with SQL Server and acquisition of knowledge. To this end, I have decided to start writing eBooks mainly, but not limited to, on the above areas.

Who Should Read This Book?

This book is for database administrators and architects who monitor and tune SQL Server instances in order to keep them operating to the maximum possible performance and stability. The book suggests several techniques that can be used for ensuring a performant SQL Server instance. However, the book is not intended to be a step-by-step comprehensive guide. Additionally, it assumes at least intermediate-level experience with SQL Server administration and knowledge of basic database principles (i.e. indexing, locking, etc.).

Supported SQL Server Versions

Each article in this book has in the end an "***Applies to:***" note indicating the SQL Server versions against which the provided T-SQL scripts can be executed. There are also articles providing different T-SQL scripts for SQL Server 2000, SQL Server 2005 or later. Every time I write an article, I try to write code that can be executed in all versions of SQL Server unless stated otherwise. Even though SQL Server 2000 and

2005 are not officially supported anymore, I am quite sure that still there are many of you out there who still use these versions of SQL Server for various reasons (i.e. application compatibility issues, etc.). My advice is to consider upgrading to a newer version of SQL Server the soonest possible, in order to be able to use the benefits of the latest SQL Server releases, as well as being able to get official support and updates.

How Is This Book Organized?

This book is organized as follows. Chapter 1 discusses the usage of indexes and the derived benefits of using them along with techniques to maintain them healthy in order to ensure performance. Chapter 2 discusses ways of managing log files as well as how to ensure that their growth will not create issues during the execution of heavy set-based operations. Chapter 3 discusses how you can monitor locking and tracking blocking cases in SQL Server as well as other locking-related topics. Chapter 4 discusses topics having to do with maintaining the good health and performance of the tempdb database as well as retrieving information from tempdb's data structures.

Feedback

I am a proud member of the worldwide SQL Server community. Feedback from you, my fellow community members, is more than welcome. Please feel free to visit the following link and provide your feedback:

http://www.sqlebooks.com/feedback/

The survey is short. It will only take 5 minutes of your valuable time.

Acknowledgments

Writing a book is not an easy thing. It doesn't really matter the length of the book. Even if you just write a few pages, it takes a considerable amount of time and energy. When you write a book, "normal" time is not enough. That is why you also need to use valuable "free" time as well which is actually time from your family and beloved ones. You may use this time because you are just passionate in what you do. However, this is not enough, at least to me. Without the support of your family it just doesn't feel right. During this process, I had all the support I needed. I am grateful for all the support my family gave me during the period of writing this book and for all their support and love in everything I do. Without their support and encouragement none of this would have been possible.

Stay in Touch

Feel free to connect! You can find me on the following online channels:

- Official Website: https://www.aartemiou.com
- SQLArtBits: https://www.sqlartbits.com
- SQLEBooks.com: https://www.sqlebooks.com
- Twitter: https://twitter.com/artemakis
- The SQL Server and .NET Hub: http://www.sqlnethub.com
- The SQL Server and .NET TV: http://www.youtube.com/user/sqlserverdotnetblog
- Cyprus .NET User Group: http://www.cdnug.net
- CodePlex: http://www.codeplex.com/site/users/view/ArtSQL

CHAPTER 3

Locking and Blocking

IN THIS CHAPTER:

- Monitoring Locking in SQL Server
- Updating SQL Server Tables Using Hints
- Table-Level Locking Hints in SQL Server

The majority of modern relational database management systems (RDBMSs) make use of lock-based concurrency. Lock-based concurrency is the approach based on which the Database Engine of a RDBMS ensures that no actions of committed transactions are lost. This is actually what ***Locking*** is.

SQL Server locks resources using different lock modes that determine how the resources can be accessed by concurrent transactions. SQL Server uses the following lock modes:

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

In RDBMSs with lock-based concurrency, such as SQL Server, there are many cases where ***blocking*** can occur. Blocking takes place when one server process id (SPID) holds a lock on one resource and a second SPID tries to place a lock of a conflicting type on the same resource. Most of the times, each blocking incident does not take much time and it is a natural behavior of RDBMs that use lock-based concurrency in order to help ensure data integrity.

There are sometimes however undesirable “blocking” incidents. Such incidents are:

- **Starvation**: It is the case where a transaction does not release a lock on a table/page thus forcing another transaction to wait indefinitely. SQL Server handles such issues with timeouts.

- **Deadlock**: It is the case where two transactions wait for each other in order to release their respective locks. SQL Server handles this by automatically selecting one of the two transactions and aborting the other along with rolling it back and sending an error message.

This chapter discusses how you can monitor locking and track blocking cases in SQL Server as well as other locking-related topics.

Monitoring Locking in SQL Server

SQL Server 2005 (or later) has a specific dynamic management view (DMV) which provides detailed information regarding the active locks within the SQL Server instance, that is locks that have been already granted or they are waiting to be granted. The DMV is called: **sys.dm_tran_locks**.

The following T-SQL query uses sys.dm_tran_locks in order to return useful information:

```
SELECT
resource_type,
resource_database_id,
(select [name] from master.sys.databases where database_id =resource_database_id)
as [DBName],
(case resource_type when 'OBJECT' then
object_name(resource_associated_entity_id,resource_database_id) else NULL end) as
ObjectName,
resource_associated_entity_id,
request_status, request_mode,request_session_id,
resource_description
FROM sys.dm_tran_locks;
GO
```

Listing 3.1: Retrieving locking information for a SQL Server instance.

A sample output of the above query would look like the one below:

resource_type	resource_database_id	DBName	ObjectName	resource_associated_entity_id	request_status	request_mode	request_session_id	resource_description
DATABASE	5	InMemOLTPDBTemp		0	GRANT	S	78	
DATABASE	7	temp		0	GRANT	S	77	
OBJECT	1	master	sysobjvalues	60	GRANT	Sch-S	78	
OBJECT	1	master	syssingleobjrefs	74	GRANT	Sch-S	78	

Table 3.1: Sample Query Output for Locking Information

An alternative way of monitoring the active locks is to use the "Activity Monitor" module in SQL Server Management Studio.

By joining the records returned by **sys.dm_tran_locks** and **sys.dm_os_waiting_tasks** DMVs you can get blocking information:

```
SELECT
t1.resource_type,
t1.resource_database_id,
(select [name] from master.sys.databases where database_id =resource_database_id)
as [DBName],
t1.resource_associated_entity_id,
(case resource_type when 'OBJECT' then
object_name(resource_associated_entity_id,resource_database_id) else NULL end) as
ObjectName,
t1.request_mode,
t1.request_session_id,
t2.blocking_session_id
FROM sys.dm_tran_locks as t1, sys.dm_os_waiting_tasks as t2
WHERE t1.lock_owner_address = t2.resource_address;
GO
```

Listing 3.2: Retrieving blocking information for a SQL Server instance.

The above two queries provide you with information on active locks and blocking cases and can help you identify scenarios that might need manual intervention (i.e. in cases of a bad database design and when the database is being accessed concurrently).

➔ ***Applies to: SQL Server 2005 or later.***

Updating SQL Server Tables Using Hints

Even though the SQL Server Database Engine automatically sets the best possible locking hints on the underlying database objects of the various T-SQL operations that executes, there are cases where we might need to manually control locking as a part of the business logic in our T-SQL script.

A popular locking hint is the NOLOCK as it can be used in environments with high concurrency. By using the NOLOCK hint, the transaction isolation level for the SELECT statement is READ UNCOMMITTED. Of course, this means that the query may see inconsistent data (dirty reads), that is data not yet committed, etc. The NOLOCK hint can only be used in SELECT statements. *Even though the NOLOCK hint may seem handy in some cases, do not use it (or any other hint) unless you are sure that the issue cannot be resolved in any other way than using the hint and you deal with dirty reads.*

Now imagine the following scenario: You need to design a special UPDATE statement that will be updating an unknown number of records in a table which is being concurrently accessed by several other T-SQL statements (mostly UPDATE statements). *The above statement will be updating the table with non-critical information meaning that if it skips some records the first time, it can update them the second time and so on*.

If even one of the other UPDATE statements has locked a row that needs to be modified by your UPDATE statement, this will cause the latter to wait (blocking). However, in the case where waiting is not a much "desired" option what should you do? In that case you could use the **READPAST** locking hint.

The READPAST locking hint when used, instructs the SQL Server Database Engine to skip row-level locks. This means that the UPDATE statement using READPAST will only update the table rows that are not locked by another operation. In the opposite case, the Database Engine would block the UPDATE statement's execution until the rest of the target rows' locks are released. A typical UPDATE statement with the READPAST locking hint would look like this:

```
UPDATE [TABLE_NAME] WITH (READPAST)
SET ...
WHERE ...
```

Listing 3.3: Sample code for updating a table using hints.

***Important:** Even if the above locking hint can become quite handy, as well as the rest of the locking hints, you always need to have in mind that you should use them very carefully as you might cause locking issues in the database. SQL Server Query Optimizer typically selects the best execution plan for a query, so it is not recommended for inexperienced developers and administrators to make use of the locking hints.

As a last note, the READPAST hint can only be specified in transactions operating at the READ COMMITTED or REPEATABLE READ isolation levels.

➔ ***Applies to: SQL Server 2005 or later.***

Table-Level Locking Hints in SQL Server

I was once asked by a friend, how we can control the level of locking in SQL Server when executing SELECT, UPDATE, INSERT and DELETE statements. In the beginning I was trying to avoid answering to him by strongly supporting the position that *SQL Server users should avoid using locking hints as the Database Engine can efficiently handle locking*. When manually using hints, there is the danger of causing inconsistencies in your data. However my friend insisted and so I had to answer and thus talk about locking hints.

Locking hints direct SQL Server to the type of locks to be used. Even though the SQL Server Query Optimizer automatically determines the best locking option when executing a statement, there are cases where after some serious testing a DBA or Database Developer might want to explicitly control the level of locking.

The available locking hints in SQL Server are the following:

HOLDLOCK
Applies to: SELECT, UPDATE, INSERT, DELETE

NOLOCK

Applies to: SELECT

PAGLOCK
Applies to: SELECT, UPDATE, INSERT, DELETE

READCOMMITTED
Applies to: SELECT, UPDATE, INSERT, DELETE

READPAST
Applies to: SELECT, UPDATE, DELETE

READUNCOMMITTED
Applies to: SELECT

REPEATABLEREAD
Applies to: SELECT, UPDATE, INSERT, DELETE

ROWLOCK
Applies to: SELECT, UPDATE, INSERT, DELETE

SERIALIZABLE
Applies to: SELECT, UPDATE, INSERT, DELETE

TABLOCK
Applies to: SELECT, UPDATE, INSERT, DELETE

TABLOCKX
Applies to: SELECT, UPDATE, INSERT, DELETE

UPDLOCK
Applies to: SELECT, UPDATE, INSERT, DELETE

XLOCK
Applies to: SELECT, UPDATE, INSERT, DELETE

The syntax for using locking hints is very simple. You only have to add the expression ***WITH (LOCKING_HINT)*** just right after the database table name (or table alias) referenced in your query.

An example of using the **NOLOCK** locking hint within a **SELECT** statement is the following:

```
SELECT column_name
FROM table_name WITH (NOLOCK);
GO
```

Listing 3.4: Using the NOLOCK table hint.

The following MSDN Library article fully describes the above locking hints.

→ Applies to: SQL Server 2000 (partially), 2005 or later (fully).

▪ Summary

In this chapter, you learned what locking and blocking are, how you can update SQL Server tables with avoiding blocking, as well as how you can use table-level locking hints in SQL Server. The next chapter talks about one of the system databases that are shipped with SQL Server: **tempdb**.