**What is Read Committed Snapshot Isolation?**

The idea behind Read Committed Snapshot Isolation is the following: instead of locking a record during the reading phase with a Shared Lock, SQL Server will transparently return you the old committed record version from the Version Store. The Version Store is stored in TempDb. Therefore it is also very important that you have carefully designed and planned TempDb.

As soon as you are using Read Committed Snapshot Isolation your current record version points with 14 additional bytes to an older record version which is stored in TempDb. And this 14 additional bytes, which are used for the pointer into TempDb, can introduce now some serious problems. Why? Let’s have a look at it.

**14 bytes matter!**

Imagine you have an already created database with some tables. To demonstrate that scenario, let’s create a simple table and insert some records into it.

Transact-SQL



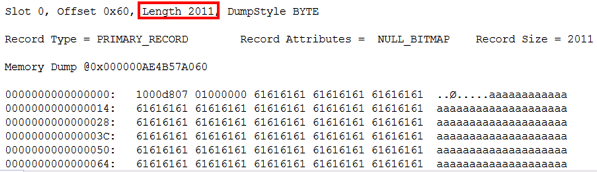
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | -- Let's create a new table without RCSI enabled  CREATE TABLE Test  (  Col1 INT IDENTITY(1, 1) PRIMARY KEY NOT NULL,  Col2 CHAR(1000),  Col3 CHAR(1000)  )  GO    -- Let's insert 100 records into the table  INSERT INTO Test VALUES (REPLICATE('a', 1000), REPLICATE('b', 1000))  GO 100 |

As you can see from the table definition, the record size is 2011 bytes long (7 bytes overhead + 4 bytes integer + 2x 1000 bytes character). You can also verify the record length by using the **DBCC PAGE** command in combination with a Data Page ID.

Transact-SQL



|  |  |
| --- | --- |
| 1  2  3  4 | -- Let's dump out a specific Data Page.  -- The record size is currently 2011 bytes (7 + 4 + 1000 + 1000).  DBCC PAGE(RCSI\_SideEffects, 1, 224, 1)  GO |



And now let’s enable Read Committed Snapshot Isolation. As I’m always saying it’s a simple database configuration change. It’s so easy!

Transact-SQL



|  |  |
| --- | --- |
| 1  2  3 | -- Let's enable now RCSI for the database  ALTER DATABASE RCSI\_SideEffects SET READ\_COMMITTED\_SNAPSHOT ON  GO |

When you now check the record length again, nothing has changed. It is still 2011 bytes. But what happens if we now update that record? In that case SQL Server has to store the old record version in TempDb, and has to add 14 additional bytes to the record on the data page. Let’s perform now a simple UPDATE statement against a record in the table.

Transact-SQL



|  |  |
| --- | --- |
| 1  2  3  4 | -- Update the specific record  UPDATE Test SET Col2 = REPLICATE('c', 1000)  WHERE Col1 = 1  GO |

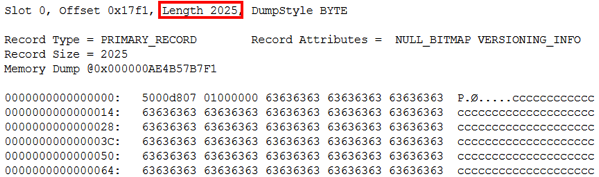
And now let’s check again the record length with the **DBCC PAGE** command.

Transact-SQL



|  |  |
| --- | --- |
| 1  2  3  4 | -- Let's checkout the record size again.  -- The record size is now 2025 bytes - 14 bytes longer!  DBCC PAGE(RCSI\_SideEffects, 1, 224, 1)  GO |

As you can see now, the record length has now changed to 2025 bytes – 14 additional bytes!



**The Problems**

And exactly this behavior introduces now a lot of performance problems. Imagine that the Data Page in your Heap Table or in your Clustered Index has no free space available anymore for this 14 additional bytes. In that case you introduce [Forwarding Records](https://www.sqlpassion.at/archive/2014/09/08/forwarding-records-on-heap-tables/) on Heap Tables, and [Page Splits](https://www.sqlpassion.at/archive/2016/03/29/clustered-indexes-advantages-disadvantages/) in your Clustered Indexes. And both are degrading the performance of your database, because additional I/O is involved.

What can you do about it? Rebuild your Indexes with a lower Fill Factor, or rebuild your Heap Tables as soon as you have Read Committed Snapshot Isolation enabled. Another option is to enable Read Committed Snapshot Isolation as soon as you have created your database. Because in that case SQL Server will add these 14 additional bytes immediately to your records.

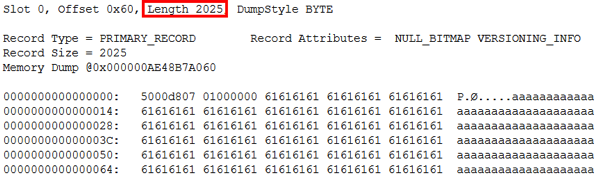
This is very easy to prove: when you have followed the example so far, you have now for your database Read Committed Snapshot Isolation enabled. Let’s try now to create another table and insert some records into it.

Transact-SQL



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | -- Let's create another table  CREATE TABLE Test1  (  Col1 INT IDENTITY(1, 1) PRIMARY KEY NOT NULL,  Col2 CHAR(1000),  Col3 CHAR(1000)  )  GO    -- Let's insert 100 records into the table  INSERT INTO Test1 VALUES (REPLICATE('a', 1000), REPLICATE('b', 1000))  GO 100 |

When you now check the record length with a **DBCC PAGE** command, you can see that the record size is immediately 2025 bytes.



**Readable Secondaries in Availability Groups**

And now let’s continue with the bad news: when you use **Readable Secondary Replicas** in an SQL Server Availability Group, SQL Server will also use internally Read Committed Snapshot Isolation. Otherwise the REDO thread on a Secondary Replica could be blocked a SELECT statement when it would acquire Shared Locks during the reading phase.

Imagine you are running your databases in your Availability Group without Readable Secondaries. In that case the record length is the traditional one without the 14 additional bytes. And when you now enable later Readable Secondaries, and you change/delete a record – SQL Server adds the 14 additional bytes. And you introduce the same problems as previously: Forwarding Records and Page Splits.

**Summary**

Read Committed Snapshot Isolation is really awesome (no kidding here!). It solves a lot of blocking situations between reads and writes, and is also a perfect candidate to eliminate Deadlocking Situations. But you are paying a high price for Read Committed Snapshot Isolation: the additional 14 bytes that I have covered today, and the introduced overhead in TempDb.

So please be very careful when you enable Read Committed Snapshot Isolation, and also monitor the number of Forwarding Records and Page Splits for the database in question.