[SQL Server 中WITH (NOLOCK)浅析](https://www.cnblogs.com/kerrycode/p/3946268.html)

2014-08-30 11:58  [潇湘隐者](https://www.cnblogs.com/kerrycode/)  阅读(135900)  评论(36)  [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=3946268)  [收藏](javascript:void(0))

**概念介绍**

开发人员喜欢在SQL脚本中使用WITH(NOLOCK), WITH(NOLOCK)其实是表提示（table\_hint）中的一种。它等同于 READUNCOMMITTED 。 具体的功能作用如下所示（摘自MSDN）：

   1： 指定允许脏读。不发布共享锁来阻止其他事务修改当前事务读取的数据，其他事务设置的排他锁不会阻碍当前事务读取锁定数据。允许脏读可能产生较多的并发操作，但其代价是读取以后会被其他事务回滚的数据修改。这可能会使您的事务出错，向用户显示从未提交过的数据，或者导致用户两次看到记录（或根本看不到记录）。有关脏读、不可重复读和幻读的详细信息，请参阅[并发影响](http://msdn.microsoft.com/zh-cn/library/ms190805(v=sql.105).aspx)。

   2： READUNCOMMITTED 和 NOLOCK 提示仅适用于数据锁。所有查询（包括那些带有 READUNCOMMITTED 和 NOLOCK 提示的查询）都会在编译和执行过程中获取 Sch-S（架构稳定性）锁。因此，当并发事务持有表的 Sch-M（架构修改）锁时，将阻塞查询。例如，数据定义语言 (DDL) 操作在修改表的架构信息之前获取 Sch-M 锁。所有并发查询（包括那些使用 READUNCOMMITTED 或 NOLOCK 提示运行的查询）都会在尝试获取 Sch-S 锁时被阻塞。相反，持有 Sch-S 锁的查询将阻塞尝试获取 Sch-M 锁的并发事务。有关锁行为的详细信息，请参阅[锁兼容性（数据库引擎）](http://msdn.microsoft.com/zh-cn/library/ms186396(v=sql.105).aspx)。

   3:  不能为通过插入、更新或删除操作修改过的表指定 READUNCOMMITTED 和 NOLOCK。SQL Server 查询优化器忽略 FROM 子句中应用于 UPDATE 或 DELETE 语句的目标表的 READUNCOMMITTED 和 NOLOCK 提示。

**功能与缺陷**

    使用WIHT(NOLOCK)有利也有弊，所以在决定使用之前，你一定需要了解清楚WITH(NOLOCK)的功能和缺陷，看其是否适合你的业务需求,不要觉得它能提升性能，稀里糊涂的就使用它。

    1：使用WITH(NOLOCK)时查询不受其它排他锁阻塞

    打开会话窗口1，执行下面脚本，不提交也不回滚事务，模拟事务真在执行过程当中

BEGIN TRAN

UPDATE TEST SET NAME='Timmy' WHERE OBJECT\_ID =1;

--ROLLBACK

   打开会话窗口2，执行下面脚本，你会发现执行结果一直查询不出来（其实才两条记录）。当前会话被阻塞了

SELECT \* FROM TEST;

    打开会话窗口3，执行下面脚本，查看阻塞情况，你会发现在会话2被会话1给阻塞了，会话2的等待类型为LCK\_M\_S：“当某任务正在等待获取共享锁时出现”

SELECT wt.blocking\_session\_id AS BlockingSessesionId

,sp.program\_name AS ProgramName

,COALESCE(sp.LOGINAME, sp.nt\_username) AS HostName

,ec1.client\_net\_address AS ClientIpAddress

,db.name AS DatabaseName

,wt.wait\_type AS WaitType

,ec1.connect\_time AS BlockingStartTime

,wt.WAIT\_DURATION\_MS/1000 AS WaitDuration

,ec1.session\_id AS BlockedSessionId

,h1.TEXT AS BlockedSQLText

,h2.TEXT AS BlockingSQLText

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.dm\_exec\_connections ec1

ON ec1.session\_id = tl.request\_session\_id

INNER JOIN sys.dm\_exec\_connections ec2

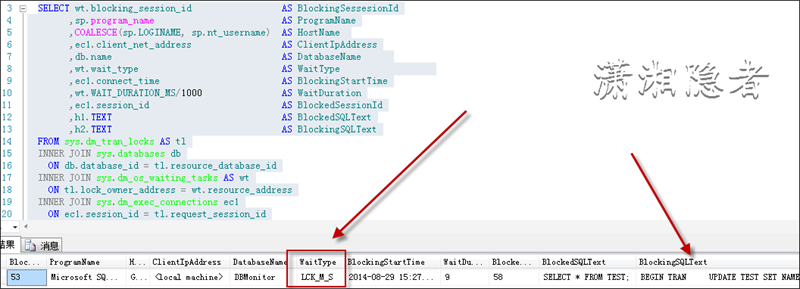
ON ec2.session\_id = wt.blocking\_session\_id

LEFT OUTER JOIN master.dbo.sysprocesses sp

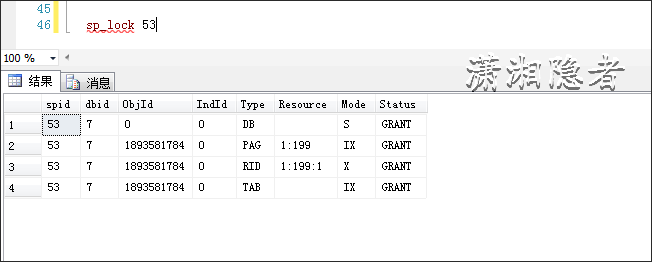
ON SP.spid = wt.blocking\_session\_id

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

[](https://images0.cnblogs.com/blog/73542/201408/301204003457660.png)

此时查看会话1（会话1的会话ID为53，执行脚本1前，可以用SELECT  @@spid查看会话ID）的锁信息情况，你会发现表TEST(ObjId=1893581784)持有的锁信息如下所示

[](https://images0.cnblogs.com/blog/73542/201408/301204032351645.png)

打开会话窗口4，执行下面脚本.你会发现查询结果很快就出来，会话4并不会被会话1阻塞。

    SELECT \* FROM TEST WITH(NOLOCK)

从上面模拟的这个小例子可以看出，正是由于加上WITH(NOLOCK)提示后，会话1中事务设置的排他锁不会阻碍当前事务读取锁定数据，所以会话4不会被阻塞，从而提升并发时查询性能。

2：WITH(NOLOCK) 不发布共享锁来阻止其他事务修改当前事务读取的数据，这个就不举例子了。

本质上WITH(NOLOCK)是通过减少锁和不受排它锁影响来减少阻塞，从而提高并发时的性能。所谓凡事有利也有弊，WITH(NOLOCK)在提升性能的同时，也会产生脏读现象。

如下所示，表TEST有两条记录，我准备更新OBJECT\_ID=1的记录，此时事务既没有提交也没有回滚

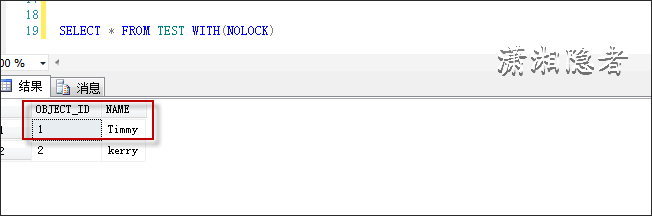
[](https://images0.cnblogs.com/blog/73542/201408/301204049076101.png)

BEGIN TRAN

UPDATE TEST SET NAME='Timmy' WHERE OBJECT\_ID =1;

--ROLLBACK

此时另外一个会话使用WITH(NOLOCK)查到的记录为未提交的记录值

[](https://images0.cnblogs.com/blog/73542/201408/301204060011901.png)

假如由于某种原因，该事务回滚了，那么我们读取到的OBJECT\_ID=1的记录就是一条脏数据。

脏读又称无效数据的读出，是指在数据库访问中，[事务](http://baike.baidu.com/view/121511.htm)T1将某一值修改，然后事务T2读取该值，此后T1因为某种原因撤销对该值的修改，这就导致了T2所读取到的数据是无效的。

**WITH(NOLOCK)使用场景**

什么时候可以使用WITH(NOLOCK)? 什么时候不能使用WITH(NOLOCK),这个要视你系统业务情况，综合考虑性能情况与业务要求来决定是否使用WITH(NOLOCK), 例如涉及到金融或会计成本之类的系统，出现脏读那是要产生严重问题的。关键业务系统也要慎重考虑。大体来说一般有下面一些场景可以使用WITH(NOLOCK)

   1: 基础数据表，这些表的数据很少变更。

   2：历史数据表，这些表的数据很少变更。

   3：业务允许脏读情况出现涉及的表。

   4：数据量超大的表，出于性能考虑，而允许脏读。

另外一点就是不要滥用WITH(NOLOCK),我发现有个奇怪现象，很多开发知道WITH(NOLOCK),但是有不了解脏读，习惯性的使用WITH(NOLOCK)。

**WITH(NOLOCK)与 NOLOCK区别**

为了搞清楚WITH(NOLOCK)与NOLOCK的区别，我查了大量的资料，我们先看看下面三个SQL语句有啥区别

    SELECT \* FROM TEST NOLOCK

    SELECT \* FROM TEST (NOLOCK);

    SELECT \* FROM TEST WITH(NOLOCK);

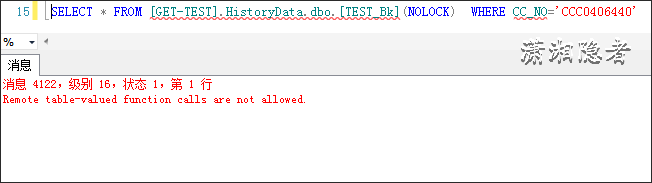
上面的问题概括起来也就是说NOLOCK、(NOLOCK)、 WITH(NOLOCK)的区别：

1： NOLOCK这样的写法，其实NOLOCK其实只是别名的作用，而没有任何实质作用。所以不要粗心将(NOLOCK)写成NOLOCK

2：（NOLOCK）与WITH(NOLOCK)其实功能上是一样的。(NOLOCK)只是WITH(NOLOCK)的别名,但是在SQL Server 2008及以后版本中，(NOLOCK)不推荐使用了，"不借助 WITH 关键字指定表提示”的写法已经过时了。 具体参见MSDN <http://msdn.microsoft.com/zh-cn/library/ms143729>%28SQL.100%29.aspx

    2.1  至于网上说WITH(NOLOCK)在SQL SERVER 2000不生效，我验证后发现完全是个谬论。

    2.2  在使用链接服务器的SQL当中，(NOLOCK)不会生效，WITH(NOLOCK)才会生效。如下所示

[](https://images0.cnblogs.com/blog/73542/201408/301204068133785.png)

    消息 4122，级别 16，状态 1，第 1 行

    Remote table-valued function calls are not allowed.

3.语法上有些许出入，如下所示

这种语法会报错

SELECT  \* FROM   sys.indexes  WITH(NOLOCK) AS i

-Msg 156, Level 15, State 1, Line 1

-Incorrect syntax near the keyword 'AS'.

这种语法正常

SELECT  \* FROM   sys.indexes  (NOLOCK) AS i

可以全部改写为下面语法

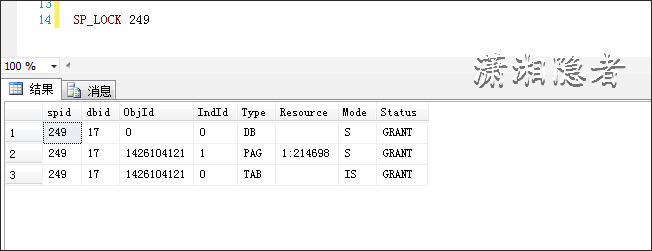
SELECT  \* FROM   sys.indexes   i WITH(NOLOCK)

SELECT  \* FROM   sys.indexes   i (NOLOCK)

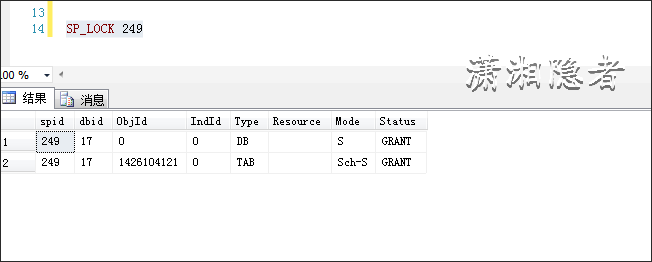
**WITH(NOLOCK)会不会产生锁**

    很多人误以为使用了WITH(NOLOCK)后，数据库库不会产生任何锁。实质上，使用了WITH(NOLOCK)后，数据库依然对该表对象生成Sch-S（架构稳定性）锁以及DB类型的共享锁， 如下所示，可以在一个会话中查询一个大表，然后在另外一个会话中查看锁信息(也可以使用SQL Profile查看会话锁信息)

    不使用WTIH(NOLOCK)

[](https://images0.cnblogs.com/blog/73542/201408/301204078603599.png)

  使用WITH(NOLOCK)

[](https://images0.cnblogs.com/blog/73542/201408/301204088769183.png)

  从上可以看出使用WITH(NOLOCK)后，数据库并不是不生成相关锁。  对比可以发现使用WITH(NOLOCK)后，数据库只会生成DB类型的共享锁、以及TAB类型的架构稳定性锁.

另外，使用WITH(NOLOCK)并不是说就不会被其它会话阻塞，依然可能会产生Schema Change Blocking

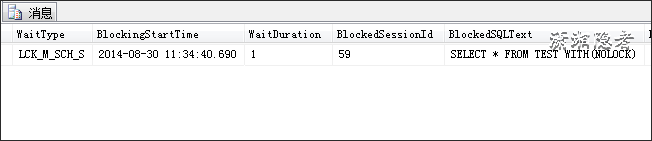
会话1：执行下面SQL语句，暂时不提交，模拟事务正在执行

BEGIN TRAN

ALTER TABLE TEST ADD Grade VARCHAR(10) ;

会话2：执行下面语句，你会发现会话被阻塞，截图如下所示。

SELECT \* FROM TEST WITH(NOLOCK)

[](https://images0.cnblogs.com/blog/73542/201408/301158081574024.png)

IIS当前请求队列查询：

进入CMD , 切换目录至： c:\windows\system32\inetsrv

常用命令： appcmd list request

异常时查看堆积的队列，判断当前请求堆积是由于web服务器或数据库效率问题导致。

若由于页面处理效率问题，则视情况回收IIS资源。

或者如果是由于数据库造成，则检查数据库服务器是否死锁。

sp\_who\_lock

或

SELECT TOP 200

[request\_id],[start\_time] AS '开始时间',[status] AS '状态',

[command] AS '命令',dest.[text] AS 'sql语句', DB\_NAME([database\_id]) AS '数据库名',

[blocking\_session\_id] AS '正在阻塞其他会话的会话ID',[wait\_type] AS '等待资源类型',

[wait\_time] AS '等待时间',[wait\_resource] AS '等待的资源',[reads] AS '物理读次数',

[writes] AS '写次数',[logical\_reads] AS '逻辑读次数',[row\_count] AS '返回结果行数'

FROM sys.[dm\_exec\_requests] AS der

CROSS APPLY

sys.[dm\_exec\_sql\_text](der.[sql\_handle]) AS dest

WHERE [session\_id]>50

ORDER BY [cpu\_time] DESC

SELECT TOP 200

[request\_id],[start\_time] AS '开始时间',[status] AS '状态',

[command] AS '命令',

REPLACE(REPLACE(dest.[text],'SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED',''),'SET NOCOUNT ON','') AS 'sql语句',

DB\_NAME([database\_id]) AS '数据库名',

[blocking\_session\_id] AS '正在阻塞其他会话的会话ID',[wait\_type] AS '等待资源类型',

[wait\_time] AS '等待时间',[wait\_resource] AS '等待的资源',[reads] AS '物理读次数',

[writes] AS '写次数',[logical\_reads] AS '逻辑读次数',[row\_count] AS '返回结果行数'

FROM sys.[dm\_exec\_requests] AS der

CROSS APPLY

sys.[dm\_exec\_sql\_text](der.[sql\_handle]) AS dest

WHERE [session\_id]>50

ORDER BY [cpu\_time] DESC

SELECT \* FROM master..sysdatabases WHERE name NOT IN ( 'master', 'model', 'msdb', 'tempdb', 'northwind','pubs' )

SELECT blocking\_session\_id, wait\_duration\_ms, session\_id FROM sys.dm\_os\_waiting\_tasks ORDER BY blocking\_session\_id

dbcc inputbuffer(166)

/\*Breaking “PAGE: 6:3:70133” down, we’ve got:

database\_id=6

data\_file\_id = 3

page\_number = 70133\*/

--PAGE: 7:1:30932824

DBCC TRACEON (3604);

GO

/\* DBCC PAGE (DatabaseName, FileNumber, PageNumber, DumpStyle)\*/

DBCC PAGE ('tlsoft',1,30932824,2);

GO

SELECT

sc.name as schema\_name,

so.name as object\_name,

si.name as index\_name

FROM sys.objects as so

JOIN sys.indexes as si on

so.object\_id=si.object\_id

JOIN sys.schemas AS sc on

so.schema\_id=sc.schema\_id

WHERE

so.object\_id = 2028234676

and si.index\_id = 117;

SELECT

sys.fn\_PhysLocFormatter (%%physloc%%),

\*

FROM YX\_T\_Ypdmb (NOLOCK)

WHERE sys.fn\_PhysLocFormatter (%%physloc%%) like '(1:31975334%'