Why Is My DBA So Grumpy?

WHEN I USE NOLOCK AND OTHER HINTS

Rick Lowe

rick@data-flowe.com

DataFLowe

http://dataflowe.wordpress.com/

Microsoft CERTIFIED Master SQL Server® 2008

Microsoft CERTIFIED Solutions Master Charter - Data Platform

Beginning at the End

- Hints should be rare
- Rare is not the same as _never_. Most hints exist for a reason.
- Best added as a response to verified issues
- IMO, should always be accompanied by a comment

What Is NOLOCK

- Tells SQL Server we care about a fast response more than we care about a consistent response
- Specifically, suppresses the shared locks for queries
- Ignored by insert/update/delete statements (fortunately)
- NOT a “turbo button”
- More like “running with scissors” mode

Why NOLOCK Is Tempting

- READ COMMITTED is the default
- Readers take shared locks
- Writers take exclusive locks
- Shared and exclusive locks not compatible
- Many ways to block
  - Writers block other writers
  - Writers block readers
  - Readers block writers

Contributing Factors

- Past issues cleared up with NOLOCK
- Belief NOLOCK means … _NO_ lock
- Bad code samples / misinformation on internet
- Lock escalation concerns
- Belief dirty reads are usually OK

NOLOCK Issues

- Usually not necessary
- Possible to read uncommitted data
- Possible to miss records as they move
- Possible to double-count records as they move
- If persisted (e.g. ETL), corruption can spread
- Unpredictable results when joining

Demo 1: NOLOCK

```
CREATE TABLE dbo.Customer(
    CustomerID   INT IDENTITY  NOT NULL,
    SecretNumber CHAR(11)      NOT NULL,
    Salesperson  INT           NOT NULL,
    FullName     NVARCHAR(100) NOT NULL,
    MailAddress  NVARCHAR(200) NOT NULL,
    CONSTRAINT PK_Customer PRIMARY KEY CLUSTERED( CustomerID )
);
GO
```

```sql
WHILE 1 = 1
BEGIN
BEGIN TRANSACTION
    INSERT INTO dbo.Salesperson( SalespersonID, FullName )
    VALUES( 0, 'New Person' ); -- 0 means unassigned, don't know yet

    -- Do something to get new ID, then update
    DECLARE @newID INT = 1234;
    UPDATE dbo.Salesperson SET SalespersonID = @newID WHERE SalespersonID = 0;
ROLLBACK
END
```

```
USE KillerCode
GO

SELECT SalesPerson.SalesPersonID, SalesPerson.FullName, Customer.SecretNumber
FROM dbo.SalesPerson WITH(NOLOCK)
    INNER JOIN dbo.Customer ON SalesPerson.SalespersonID = Customer.Salesperson
WHERE SalesPerson.FullName = 'New Person';
```

200 %

Results | Messages

SalesPersonID	FullName	SecretNumber

```sql
SELECT SalesPerson.SalesPersonID, SalesPerson.FullName, Customer.SecretNumber
FROM dbo.SalesPerson WITH(NOLOCK)
    INNER JOIN dbo.Customer ON SalesPerson.SalespersonID = Customer.Salesperson
WHERE SalesPerson.FullName = 'New Person';
```

200 %

Results | Messages

	SalesPersonID	FullName	SecretNumber
1	0	New Person	123-45-0101
2	0	New Person	123-45-0102
3	0	New Person	123-45-0103
4	0	New Person	123-45-0104
5	0	New Person	123-45-0105
6	0	New Person	123-45-0106
7	0	New Person	123-45-0107
8	0	New Person	123-45-0108
9	0	New Person	123-45-0109
10	0	New Person	123-45-0110

```sql
CREATE TABLE dbo.OrderHistory(
    OrderID        INT IDENTITY NOT NULL,
    ClientID       INT          NOT NULL,
    SalesPersonID  INT          NOT NULL,
    TotalCost      DECIMAL(8,2) NOT NULL,
    LastAccess     DATETIME2(0) NOT NULL,
    Notes          VARCHAR(MAX)     NULL,
    CONSTRAINT PK_OrderHistory PRIMARY KEY CLUSTERED( OrderID )
);

CREATE INDEX IX_OrderHistory_LastAccess
    ON dbo.OrderHistory( LastAccess )
    INCLUDE( SalesPersonID, TotalCost );
```

```sql
DECLARE @max_id INT = (SELECT MAX( OrderID ) FROM dbo.OrderHistory );
DECLARE @x INT;

WHILE 1 = 1
BEGIN
    SET @x = 1;
    BEGIN TRANSACTION
        WHILE @x < @max_id
        BEGIN
            UPDATE dbo.OrderHistory SET LastAccess = SYSDATETIME() WHERE OrderID = @x;
            SET @x += 1;
        END
    ROLLBACK;
END
```

```sql
CREATE TABLE #accum(
    Trial          INT           NOT NULL,
    SalesPersonID INT           NOT NULL,
    Sales          DECIMAL(18,2) NOT NULL,
    PRIMARY KEY( Trial, SalesPersonID )
)

DECLARE @x INT = 0;

WHILE @x < 10
BEGIN
    SET @x += 1;

    INSERT INTO #accum( Trial, SalesPersonID, Sales )
    SELECT @x, SalesPersonID, SUM( TotalCost ) AS Sales
    FROM dbo.OrderHistory WITH( NOLOCK )
    GROUP BY SalesPersonID;
END
```

```sql
SELECT SalesPersonID, [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]
FROM
( SELECT SalesPersonID, Trial, FORMAT( Sales, 'N2' ) AS Sales FROM #accum ) p
PIVOT( MAX( Sales ) FOR Trial IN( [1], [2], [3], [4], [5], [6], [7], [8], [9], [10] )) AS pvt
ORDER BY SalesPersonID;
```

	SalesPersonID	1	2	3	4	5	6	7	8	9	10
1	1	500,496,961.41	500,632,383.32	500,418,915.51	500,519,659.06	500,685,422.19	500,560,174.65	500,666,985.38	500,649,141.91	500,584,324.96	500,549,100.24
2	2	496,140,496.76	496,268,970.23	496,021,863.71	496,059,035.93	496,239,720.74	496,144,228.15	496,273,568.69	496,245,436.53	496,161,663.65	496,198,308.31
3	3	499,731,866.93	499,929,960.13	499,666,543.37	499,752,026.30	499,851,718.50	499,790,110.89	499,954,379.47	499,899,206.56	499,777,479.80	499,820,597.79
4	4	501,712,975.10	501,892,835.32	501,702,766.67	501,719,510.61	501,863,048.21	501,755,601.34	501,855,923.57	501,880,181.01	501,765,023.60	501,777,589.09
5	5	501,619,229.41	501,790,599.57	501,609,653.74	501,661,360.95	501,782,658.30	501,714,382.71	501,862,053.33	501,817,151.69	501,730,471.01	501,743,646.46
6	6	498,387,141.24	498,533,278.46	498,319,717.59	498,366,431.19	498,553,283.10	498,442,740.63	498,595,449.27	498,541,241.93	498,442,484.65	498,470,532.76
7	7	506,200,998.39	506,345,707.22	506,094,221.79	506,191,444.99	506,343,707.00	506,264,974.00	506,386,182.89	506,353,006.80	506,279,877.66	506,292,095.54
8	8	501,118,626.89	501,276,491.86	501,057,987.27	501,133,095.19	501,256,494.92	501,163,046.23	501,329,496.44	501,266,451.74	501,162,344.12	501,211,887.71
9	9	502,911,647.96	503,044,064.64	502,824,311.57	502,883,826.86	503,043,223.95	502,958,025.01	503,078,824.20	503,044,675.30	502,976,489.68	502,984,129.47
10	10	500,665,214.08	500,827,712.22	500,626,986.07	500,674,544.02	500,835,582.45	500,698,416.56	500,815,075.74	500,808,423.09	500,746,484.35	500,760,856.45

1	2	3	4	5
500,496,961.41	500,632,383.32	500,418,915.51	500,519,659.06	500,685,422.19
496,140,496.76	496,268,970.23	496,021,863.71	496,059,035.93	496,239,720.74
499,731,866.93	499,929,960.13	499,666,543.37	499,752,026.30	499,851,718.50
501,712,975.10	501,892,835.32	501,702,766.67	501,719,510.61	501,863,048.21
501,619,229.41	501,790,599.57	501,609,653.74	501,661,360.95	501,782,658.30

Eliminating NOLOCK

- If hints don't actually prevent blocking, simply remove
- If blocking exists, consider optimistic concurrency (row versioning)
  - Read Committed Snapshot Isolation (RCSI) makes row versioning the default behavior rather than blocking for the DB
  - If RCSI seems risky, consider enabling snapshot isolation level. If snapshot is active, row versioning can be requested as needed.

ROWLOCK

- Mostly harmless, more annoying than problematic
- Sometimes seen due to concern about lock escalation
  - In cases where lock escalation is driven by amount of memory used by locks, may delay lock escalation
  - With modern hardware, not likely to make a difference
- Can force use of row locks rather than page locks
  - If MSSQL is using page lock is there a more interesting problem?
  - Is it a heap? Adding clustered index may be good idea anyway

RECOMPILE

- Legitimate uses do exist
- "I think this is going to be hairy" isn't necessarily one of them
- When used correctly, great way to handle parameter sensitivity
- Side effects
  - Interferes with plan cache
  - Increases compilation (CPU use)
  - Did I mention … interferes with plan cache (query history)
- Less problematic at query level than stored procedure level

OPTIMIZE FOR

- Like RECOMPILE, helps with parameter sensitivity
- Plan will be cached (reused), but assumptions about parameters are changed
- Overuse is less problematic than overuse of RECOMPILE … but parameter sniffing is usually your friend
- Plan is like to be OK for most values, not optimal
- Possible no single plan will work for every set of parameters
- If optimizing for specific value, data distribution may change

Thank You