



BRENT OZAR
UNLIMITED®

Avoiding Deadlocks

3.3 p1

Session agenda

Basics:

- 3 concurrency issues
- 3 ways to fix ‘em all
- 1 “fix” that makes things worse: NOLOCK

One real fix: work on tables in a consistent order

- Demo: unrealistic query
- Demo: realistic query

Using sp_BlitzLock to find the queries you need to fix



3.3 p2

Concurrency challenges

Locking: Lefty takes out a lock.

Blocking: Righty wants a lock, but Lefty already has it.
SQL Server will let Righty wait for forever,
and the symptom is LCK* waits.

Deadlocks:

Lefty has locks, then wants some held by Righty.
Righty has locks, then wants some held by Lefty.
SQL Server solves this one by killing somebody,
and the symptom is dead bodies everywhere.



3.3 p3

3 ways to fix concurrency issues

1. Have enough indexes to make your queries fast, but not so many that they slow down DUIs, making them hold more locks for longer times.

(I cover this in Mastering Index Tuning.)

2. Tune your transactional code.

(This module focuses on this topic.)

3. Use the right isolation level for your app's needs.

(I cover this in Mastering Server Tuning.)



3.3 p4

1 way doesn't fix it: dirty reads

WITH (NOLOCK):

- Ignores other people's row locks
- Still takes out schema stability locks
(and honors other peoples' schema locks)

**SET TRANSACTION ISOLATION LEVEL READ
UNCOMMITTED**

- Like putting **WITH (NOLOCK)** on every table



3.3 p5

Because with dirty reads...

1. You can see data that was never committed
2. You can see rows twice
3. You can skip rows altogether
4. Your query can fail with an error:
`Could not continue scan with NOLOCK due to data movement`



3.3 p6



```
1 SELECT COUNT(*)  
2 FROM dbo.Users WITH (NOLOCK)  
3 WHERE DisplayName = 'alex';  
4 GO 28
```

Demoing it

If I count the number of Alexes,
and I use NOLOCK,
I get the same result every time.

As long as nothing is happening.



3.3 p7

But while it runs...

Let's update everyone who ISN'T Alex:

```
BEGIN TRAN  
UPDATE dbo.Users  
SET Location = N'The Derek Zoolander School for Kids Who Can''t Read Good and Want to Do Other Stuff Good Too',  
    WebsiteUrl = N'https://www.youtube.com/watch?v=NQ-8IuUkJJc'  
WHERE DisplayName <> 'alex';
```

Note that I am NOT inserting or deleting rows.

Just updating the non-Alexes.



3.3 p8

SQLOutLog1.sql Executing... > X

```
1 BEGIN TRAN
2 UPDATE dbo.Users
3     SET Location = N'The Derek Zoolander School for Kids Who Can''t Read Good ar
4     WebsiteUrl = N'https://www.youtube.com/watch?v=NQ-8IuUkJJc'
5     WHERE DisplayName <> 'alex';
6 GO
```

This isn't changing row count or the Alexes

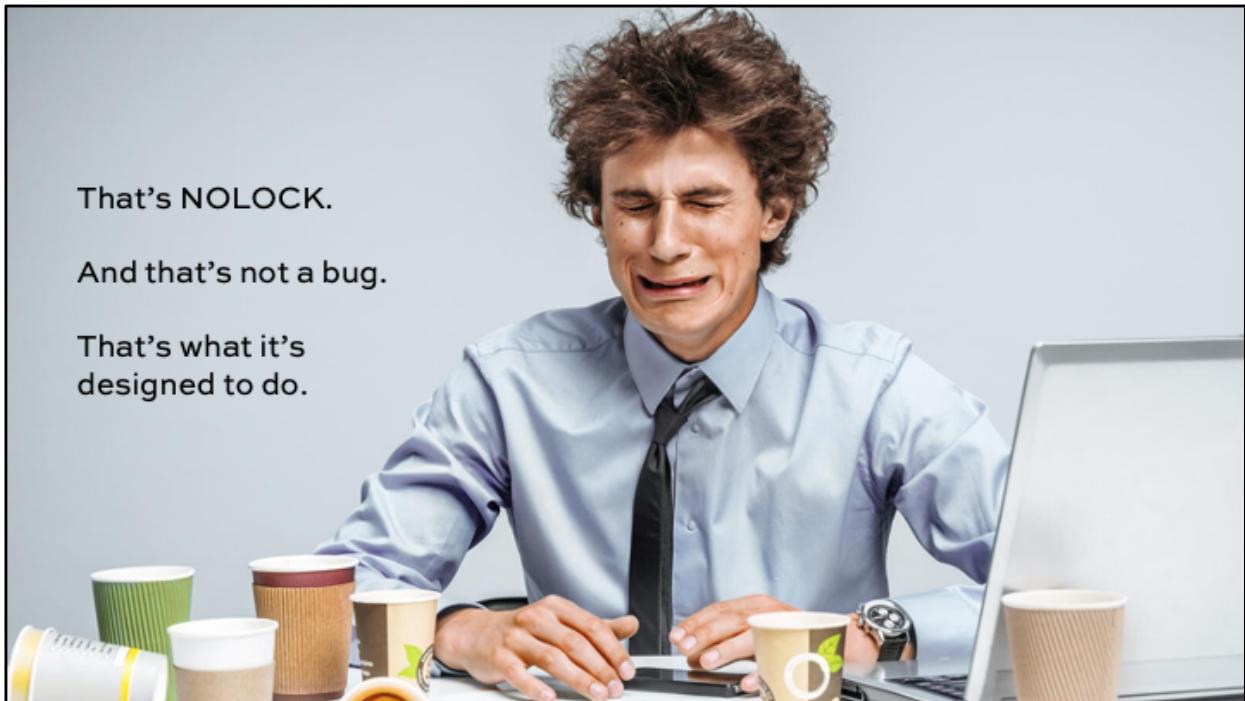
SQLOutLog2.sql > X

```
1 SELECT COUNT(*)
2 FROM dbo.Users WITH (NOLOCK)
3 WHERE DisplayName = 'alex';
4 GO 20
```

Results Messages

(No column name)
1 3584
(No column name)
1 3589
(No column name)
1 3592
(No column name)
1 3588
(No column name)
1 3584
(No column name)
1 3577
(No column name)
1 3501
(No column name)
1 3519
(No column name)
1 3546
(No column name)
1 3541
(No column name)
1 3552
(No column name)
1 3507
(No column name)
1 3543

But the number of Alexes keeps changing



That's NOLOCK.

And that's not a bug.

That's what it's
designed to do.

Sometimes, these are OK.

1. You can see data that was never committed
2. You can see rows twice
3. You can skip rows altogether
4. Your query can fail with an error:
Could not continue scan with NOLOCK due to data movement

But when they're not OK, we have some fixes to do.



3.3 p11

3 ways to fix concurrency issues

1. Have enough indexes to make your queries fast, but not so many that they slow down DUIs, making them hold more locks for longer times.

(I cover this in Mastering Index Tuning.)

2. Tune your transactional code.

(This module explores this topic.)

3. Use the right isolation level for your app's needs.

(I cover this in Mastering Server Tuning.)



3.3 p12

Session agenda

Basics:

- 3 concurrency issues
- 3 ways to fix ‘em all
- 1 “fix” that makes things worse: NOLOCK

One real fix: work on tables in a consistent order

- Demo: unrealistic query
- Demo: realistic query

Using sp_BlitzLock to find the queries you need to fix

We are here.



3.3 p13

Start SSMS with two windows.

Use Lefty.sql and Righty.sql from your resources.

In Lefty, create & populate the tables.

```
Lefty.sql - Microsoft SQL Server Management Studio (Administrator)
File Edit View Query Project Debug Tools Window Help
New Query Execute Debug
StackOverflow
Object Explorer
Lefty.sql
DROP TABLE IF EXISTS dbo.Lefty;
DROP TABLE IF EXISTS dbo.Righty;
GO
CREATE TABLE dbo.Lefty (Numbers INT PRIMARY KEY CLUSTERED);
INSERT INTO dbo.Lefty VALUES (1), (2), (3);
CREATE TABLE dbo.Righty (Numbers INT PRIMARY KEY CLUSTERED);
INSERT INTO dbo.Righty VALUES (1), (2), (3);
GO
100 % Messages
```

```
Righty.sql
/*
RIGHT SIDE */
BEGIN TRAN
UPDATE dbo.Righty
SET Numbers = Numbers + 1;
GO
UPDATE dbo.Lefty
SET Numbers = Numbers + 1;
100 % Messages
```

In Lefty, start a transaction.

Begin tran, update dbo.Lefty, but don't commit.

The left window is now locking dbo.Lefty.

```
Lefty.sql - Microsoft SQL Server Management Studio (Administrator)
File Edit View Query Project Debug Tools Window Help
StackOverflow New Query Execute Debug Messages
Lefty.sql /* LEFT SIDE: */
BEGIN TRAN
UPDATE dbo.Lefty
SET Numbers = Numbers + 1;
GO

UPDATE dbo.Righty
SET Numbers = Numbers + 1;
100 % Messages
(8 rows affected)

Righty.sql /* RIGHT SIDE: */
BEGIN TRAN
UPDATE dbo.Righty
SET Numbers = Numbers + 1;
GO

UPDATE dbo.Lefty
SET Numbers = Numbers + 1;
```

In Righty, start another.

Begin tran, update dbo.Righty, but don't commit.

The right window is now locking dbo.Righty.

```
Lefty.sql - Microsoft SQL Server Management Studio (Administrator)
File Edit View Query Project Debug Tools Window Help
StackOverflow Execute Debug
Object Explorer
Lefty.sql + X
/* LEFT SIDE: */
BEGIN TRAN
UPDATE dbo.Lefty
SET Numbers = Numbers + 1;
GO

UPDATE dbo.Righty
SET Numbers = Numbers + 1;
100 % < > Messages
(8 rows affected)

Righty.sql + X
/* RIGHT SIDE: */
BEGIN TRAN
UPDATE dbo.Righty
SET Numbers = Numbers + 1;
GO

UPDATE dbo.Lefty
SET Numbers = Numbers + 1;
100 % < > Messages
(3 rows affected)
```

The situation so far:

Left window has: Right window has:

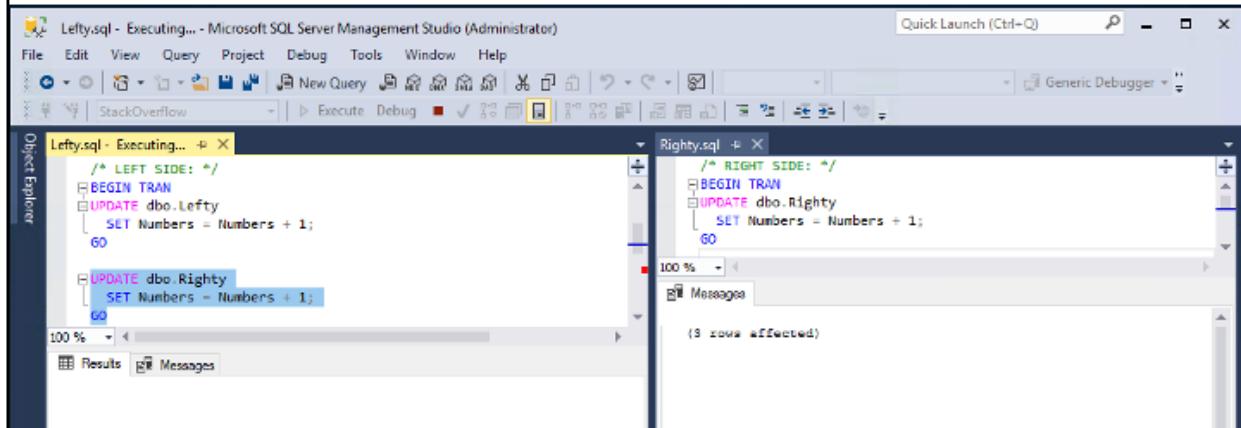
- dbo.Lefty
exclusive lock
- dbo.Righty
exclusive lock



3.3 p17

In Lefty, update dbo.Righty.

The update starts running, but is blocked, and sits there waiting for dbo.Righty to commit or roll back.



The screenshot shows the Microsoft SQL Server Management Studio interface with two query windows open:

- Lefty.sql - Executing...**: This window contains the following SQL code:

```
/* LEFT SIDE: */
BEGIN TRAN
UPDATE dbo.Lefty
SET Numbers = Numbers + 1;
GO

UPDATE dbo.Righty
SET Numbers = Numbers + 1;
GO
```
- Righty.sql - Executing...**: This window contains the following SQL code:

```
/* RIGHT SIDE: */
BEGIN TRAN
UPDATE dbo.Righty
SET Numbers = Numbers + 1;
GO
```

The status bar at the bottom of the interface indicates "100 %". The "Messages" pane in the Righty window shows the message "(3 rows affected)".

The situation so far:

Left window has: Right window has:

- dbo.Lefty
exclusive lock
- Wants a lock on
dbo.Righty, but
can't get it (yet)
- dbo.Righty
exclusive lock



3.3 p19



The situation so far:

Left window has: Right window has:

- dbo.Lefty exclusive lock
- Wants a lock on dbo.Righty, but can't get it (yet)

- dbo.Righty exclusive lock

As long as we commit or roll back in this window, things will still work out just fine.



3.3 p21

But let's do something terrible.

In the right hand window, don't commit or roll back:
instead, try to get a lock on dbo.Lefty.

```
Lefty.sql - Executing... ×
/* LEFT SIDE: */
BEGIN TRAN
UPDATE dbo.Lefty
SET Numbers = Numbers + 1;
GO

UPDATE dbo.Righty
SET Numbers = Numbers + 1;
GO

100 % < > Results Messages
```

```
Righty.sql - ×
60
UPDATE dbo.Lefty
SET Numbers = Numbers + 1;
GO
```

About to run this

Blocked, waiting

Things are going to happen fast.

I'll describe what's going to happen before I hit F5:

- The right window will want to run, but...



3.3 p23

The situation will become:

Left window has:

- dbo.Lefty exclusive lock
- Wants a lock on dbo.Righty, but can't get it (ever)

Right window has:

- dbo.Righty exclusive lock
- Wants a lock on dbo.Lefty, but can't get it (ever)



3.3 p24

Things are going to happen fast.

I'll describe what's going to happen before I hit F5:

- The right window will want to run, but...
- Neither side will be able to make progress
- SQL Server's deadlock monitor wakes up every 5 seconds, and when he does, he'll see the problem
- He'll pick the query that's the easiest to roll back, and kill it



3.3 p25

I hit F5 in the right window, and...

Within 5 seconds, SQL Server kills one.

```
/* LEFT SIDE: */
BEGIN TRAN
UPDATE dbo.Lefty
    SET Numbers = Numbers + 1;
GO

UPDATE dbo.Righty
    SET Numbers = Numbers + 1;
GO
```

```
/* RIGHT SIDE: */
GO
UPDATE dbo.Lefty
    SET Numbers = Numbers + 1;
GO
```

100 % 100 %

Messages Messages

Msg 1205, Level 13, State 51, Line 16
Transaction (Process ID 60) was deadlocked on lock resources
with another process and has been chosen as the deadlock victim.
Rerun the transaction.

(3 rows affected)

BOOM, dead.

He got lucky this time.



The root cause: bad ordering

Left window has:

- dbo.Lefty exclusive lock
- Wants a lock on dbo.Righty, but can't get it (ever)

Right window has:

- dbo.Righty exclusive lock
- Wants a lock on dbo.Lefty, but can't get it (ever)



3.3 p28

The fix: better ordering

If we work on tables in a consistent order:

- Always update dbo.Lefty first,
then update dbo.Righty

Or:

- Always update dbo.Righty first,
then update dbo.Lefty

We'll be fine either way, as long as we're consistent.



3.3 p29

Trying it:

I set up both windows to work on dbo.Lefty first.

I hit execute in the left window first, then the right:

The screenshot shows two SQL Editor windows in SSMS. The left window, titled 'Lefty.sql', contains the following script:

```
8 GO
9
10 /* LEFT SIDE: */
11 BEGIN TRAN
12 UPDATE dbo.Lefty
13     SET Numbers = Numbers + 1;
14 GO
```

The output pane shows '(3 rows affected)'. The right window, titled 'Righty.sql - Executing...', contains the following script:

```
1 /* RIGHT SIDE: */
2 BEGIN TRAN
3 UPDATE dbo.Lefty
4     SET Numbers = Numbers + 1;
5 GO
```

A red callout box with the text 'Blocked, waiting' is overlaid on the right window's status bar area. The status bar at the bottom right of the screen displays '3.3 p30'.

This sounds bad at first.

We have a new problem: blocking.

The right window can't make progress.

But that's actually good:
he can't grab a lock that would block others.

The left side is able to keep right on going.



3.3 p31

Continuing in the left window...

Lefty.sql - Executing... `GO`

/* LEFT SIDE: */
BEGIN TRAN
UPDATE dbo.Lefty
SET Numbers = Numbers + 1;
GO

UPDATE dbo.Righty
SET Numbers = Numbers + 1;
GO

(3 rows affected)

Righty.sql - Executing... `/* RIGHT SIDE: */`

BEGIN TRAN
UPDATE dbo.Lefty
SET Numbers = Numbers + 1;
GO

Still blocked, waiting

Continuing in the left window...

When the left finally commits, the right is free to start making progress, and can't be blocked by the left.

```
Lefty.sql*  Righty.sql*
9
10 /* LEFT SIDE: */
11 BEGIN TRAN
12 UPDATE dbo.Lefty
13     SET Numbers = Numbers + 1;
14 GO
15
16 UPDATE dbo.Righty
17     SET Numbers = Numbers + 1;
18 GO
19 COMMIT
```

He commits

```
1 /* RIGHT SIDE: */
2 BEGIN TRAN
3 UPDATE dbo.Lefty
4     SET Numbers = Numbers + 1;
5 GO
```

(3 rows affected)

Suddenly un-blocked!

The moral of the story

Work in tables in a consistent order, like:

- Always parents, then children
- Or always children, then parents

Which one you choose is less important than being ruthlessly consistent.

If even one query works out of order, there will be deadlocks.



3.3 p34

Session agenda

Basics:

- 3 concurrency issues
- 3 ways to fix ‘em all
- 1 “fix” that makes things worse: NOLOCK

One real fix: work on tables in a consistent order

- Demo: unrealistic query
- Demo: realistic query

We are here.

Using sp_BlitzLock to find the queries you need to fix



3.3 p35

Real-world scenario

At Stack Overflow, you can up/downvote questions.

The screenshot shows a Stack Overflow question titled "What do Clustered and Non clustered index actually mean?". The question was asked 10 years, 6 months ago and has been viewed 758k times. A user has voted on the question, indicated by a red arrow pointing to the "Voting" button next to the upvote (1076) and downvote (467) counts. The question text asks for an explanation of clustered and non-clustered indexes. A yellow callout box provides a detailed explanation of the difference between them. At the bottom, there are two comments asking for plain English explanations.

I have a limited exposure to DB and have only used DB as an application programmer. I want to know about Clustered and Non clustered indexes . I googled and what I found was :

A clustered index is a special type of index that reorders the way records in the table are physically stored. Therefore table can have only one clustered index. The leaf nodes of a clustered index contain the data pages. A nonclustered index is a special type of index in which the logical order of the index does not match the physical stored order of the rows on disk. The leaf node of a nonclustered index does not consist of the data pages. Instead, the leaf nodes contain index rows.

What I found in SO was [What are the differences between a clustered and a non-clustered index?](#).

Can someone explain this in plain English?

3.3 p36

What happens when you vote

Update your Users.LastAccessDate column to show that you've been accessing the site

Insert a row in the Votes table

Add one point to the question's score
(by updating its row in Posts (Q&A))

Add one point to the question-asker's reputation
(by updating their row in Users, set Reputation + 1)



3.3 p37

```
1  CREATE OR ALTER PROC dbo.usp_CastUpVote
2      @VoterId INT, @PostId INT AS
3  BEGIN
4
5      BEGIN TRAN
6          /* Update the voter's LastAccessDate because they were active on Stack Overflow: */
7          UPDATE dbo.Users
8              SET LastAccessDate = GETDATE()
9              WHERE Id = @VoterId;
10
11     /* Cast an upvote: */
12     INSERT INTO dbo.Votes (PostId, UserId, VoteTypeId, CreationDate)
13         VALUES (@PostId, @VoterId, 2, GETDATE());
14
15     /* Update the post's score: */
16     UPDATE dbo.Posts
17         SET Score = Score + 1
18         WHERE Id = @PostId;
19
20     /* Grant a reputation point to the post's owner: */
21     UPDATE u
22         SET Reputation = Reputation + 1
23         FROM dbo.Posts p
24             INNER JOIN dbo.Users u ON p.OwnerUserId = u.Id
25             WHERE p.Id = @PostId;
26
27     COMMIT;
28 END;
29 GO
```

3.3 p38

How it goes wrong

What if these two happen at the exact same time:

User A upvotes a question
owned by UserB

UserB upvotes a question
owned by UserA



3.3 p39

```
1 CREATE OR ALTER PROC dbo.usp_CastUpVote
2     @VoterId INT, @PostId INT AS
3 BEGIN
4
5     BEGIN TRAN
6         /* Update the voter's LastAccessDate because they were active on Stack Overflow: */
7         UPDATE dbo.Users
8             SET LastAccessDate = GETDATE()
9             WHERE Id = @VoterId;
10
11        /* Cast an upvote: */
12        INSERT INTO dbo.Votes (PostId, UserId, VoteTypeId, CreationDate)
13            VALUES (@PostId, @VoterId, 2, GETDATE());
14
15        /* Update the post's score: */
16        UPDATE dbo.Posts
17            SET Score = Score + 1
18            WHERE Id = @PostId;
19
20        WAITFOR DELAY '00:00:10' /* 10 seconds */
21
22        /* Grant a reputation point to the post's owner: */
23        UPDATE u
24            SET Reputation = Reputation + 1
25            FROM dbo.Posts p
26            INNER JOIN dbo.Users u ON p.OwnerUserId = u.Id
27            WHERE p.Id = @PostId;
28
29        COMMIT;
30    END;
31    GO
```

Just for this demo

3.3 p4O

Start these two at the same time

```
Proc1.sql* 33
34 /* Run these in two separate windows: */
35 EXEC usp_CastUpVote @VoterId = 8741, @PostId = 1251636;
36

SQLQuery9.sql* 1
2 EXEC usp_CastUpVote @VoterId = 149190, @PostId = 338156;
3

Msg 1205, Level 13, State 51, Procedure usp_CastUpVote,
Line 23 [Batch Start Line 34] transaction (process ID 61)
was deadlocked on lock resources with another process and
has been chosen as the deadlock victim. Rerun the transaction.
```

And one will always lose. (Which one? Tough to tell.)



3.3 p41

```

1 CREATE OR ALTER PROC dbo.usp_CastUpVote
2     @VoterId INT, @PostId INT AS
3 BEGIN
4
5     BEGIN TRAN
6         /* Update the voter's LastAccessDate because they were active on Stack Overflow: */
7         UPDATE dbo.Users
8             SET LastAccessDate = GETDATE()
9             WHERE Id = @VoterId;
10
11        /* Cast an upvote: */
12        INSERT INTO dbo.Votes (PostId, UserId, VoteTypeId, CreationDate)
13            VALUES (@PostId, @VoterId, 2, GETDATE());
14
15        /* Update the post's score: */
16        UPDATE dbo.Posts
17            SET Score = Score + 1
18            WHERE Id = @PostId;
19
20        WAITFOR DELAY '00:00:10' /* 10 seconds */
21
22        /* Grant a reputation point to the post's owner: */
23        UPDATE u
24            SET Reputation = Reputation + 1
25            FROM dbo.Posts p
26            INNER JOIN dbo.Users u ON p.OwnerUserId = u.Id
27            WHERE p.Id = @PostId;
28
29        COMMIT;
30    END;
31    GO

```

First, we lock our own row

Last, we try to lock someone else's

3.3 p42

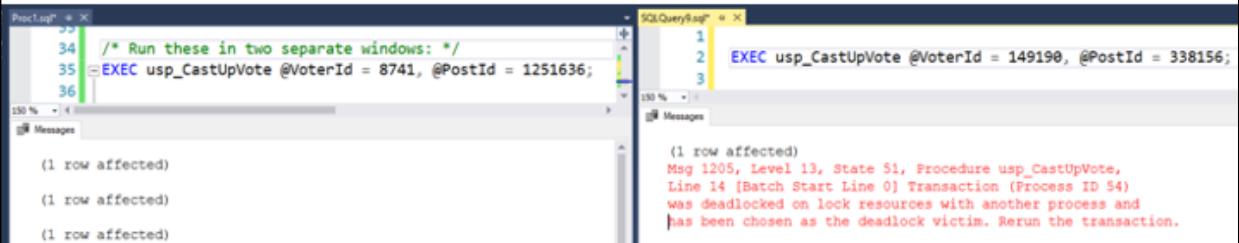
Will ordering help?

What if we move all the user updates to the top?

```
1 CREATE OR ALTER PROC dbo.usp_CastUpVote
2     @VoterId INT, @PostId INT AS
3 BEGIN
4
5     BEGIN TRAN
6     /* Update the voter's LastAccessDate because they were active on Stack Overflow: */
7     UPDATE dbo.Users
8         SET LastAccessDate = GETDATE()
9         WHERE Id = @VoterId;
10
11    WAITFOR DELAY '00:00:10' /* 10 seconds */ ← Just for this demo
12
13    /* Grant a reputation point to the post's owner: */
14    UPDATE u
15        SET Reputation = Reputation + 1
16        FROM dbo.Posts p
17        INNER JOIN dbo.Users u ON p.OwnerUserId = u.Id
18        WHERE p.Id = @PostId;
19
20    /* Cast an upvote: */
```

3.3 p43

Still screwed.



The screenshot shows two separate SQL queries running in different windows. The left window, titled 'Proc1.sql', contains the following code:

```
/* Run these in two separate windows: */
EXEC usp_CastUpVote @VoterId = 8741, @PostId = 1251636;
EXEC usp_CastUpVote @VoterId = 149190, @PostId = 338156;
```

The right window, titled 'SQLQuery8.sql', contains:

```
EXEC usp_CastUpVote @VoterId = 149190, @PostId = 338156;
```

Both windows show the output '1 row affected' for each query. The right window also displays an error message in the 'Messages' pane:

(1 row affected)
Msg 1205, Level 13, State 51, Procedure usp_CastUpVote,
Line 14 [Batch Start Line 0] Transaction (Process ID 54)
was deadlocked on lock resources with another process and
has been chosen as the deadlock victim. Rerun the transaction.

**It's not enough to lock tables in the same order:
we also need to touch them as few times as practical.**



3.3 p44

Could we update both users at once?

How might we solve this problem?

```
1 CREATE OR ALTER PROC dbo.usp_CastUpVote
2     @VoterId INT, @PostId INT AS
3 BEGIN
4
5     BEGIN TRAN
6     /* Update the voter's LastAccessDate because they were active on Stack Overflow: */
7     UPDATE dbo.Users
8         SET LastAccessDate = GETDATE()
9         WHERE Id = @VoterId;
10
11    WAITFOR DELAY '00:00:10' /* 10 seconds */
12
13    /* Grant a reputation point to the post's owner: */
14    UPDATE u
15        SET Reputation = Reputation + 1
16        FROM dbo.Posts p
17        INNER JOIN dbo.Users u ON p.OwnerUserId = u.Id
18        WHERE p.Id = @PostId;
19
20    /* Cast an upvote: */
```

3.3 p45

Ways to fix it

“Just remove the waitfor” = “make it all faster”

- Get faster hardware
- Tune indexes on the underlying tables
- Don’t hold transactions open on the app side

Try merging both Users updates into a single query
(where it’s the Voter, OR it’s the question-owner)

Do the LastAccessDate update outside of the transaction (does it really matter?)



3.3 p46

```

CREATE OR ALTER PROC dbo.usp_CastUpVote
    @VoterId INT, @PostId INT AS
BEGIN
    BEGIN TRAN
        /* Update both the voter and the question-owner */
        UPDATE u
            SET LastAccessDate = CASE WHEN u.Id = @VoterId THEN GETDATE() ELSE u.LastAccessDate END,
                Reputation = CASE WHEN u.Id = p.OwnerUserId THEN u.Reputation + 1 ELSE u.Reputation END
        FROM dbo.Posts p
        INNER JOIN dbo.Users u ON (p.OwnerUserId = u.Id OR u.Id = @VoterId)
        WHERE p.Id = @PostId;

        WAITFOR DELAY '00:00:10' Still here
        /* Cast an upvote: */
        INSERT INTO dbo.Votes (PostId, UserId, VoteTypeId, CreationDate)
            VALUES (@PostId, @VoterId, 2, GETDATE());

        /* Update the post's score: */
        UPDATE dbo.Posts
            SET Score = Score + 1
        WHERE Id = @PostId;
    COMMIT;
END;
GO

```

Lock both at once

3.3 p47

```

CREATE OR ALTER PROC dbo.usp_CastUpVote
    @VoterId INT, @PostId INT AS
BEGIN
    BEGIN TRAN
        /* Update both the voter and the question-owner */
        UPDATE u
            SET LastAccessDate = CASE WHEN u.Id = @VoterId THEN GETDATE() ELSE u.LastAccessDate END,
                Reputation = CASE WHEN u.Id = p.OwnerUserId THEN u.Reputation + 1 ELSE u.Reputation END
        FROM dbo.Posts p
        INNER JOIN dbo.Users u ON (p.OwnerUserId = u.Id OR u.Id = @VoterId)
        WHERE p.Id = @PostId;

        WAITFOR DELAY '00:00:10' Still here
        /* Cast an upvote: */
        INSERT INTO dbo.Votes (PostId, UserId, VoteTypeId, CreationDate)
            VALUES (@PostId, @VoterId, 2, GETDATE());

        /* Update the post's score: */
        UPDATE dbo.Posts
            SET Score = Score + 1
        WHERE Id = @PostId;
    COMMIT;
END;
GO

```

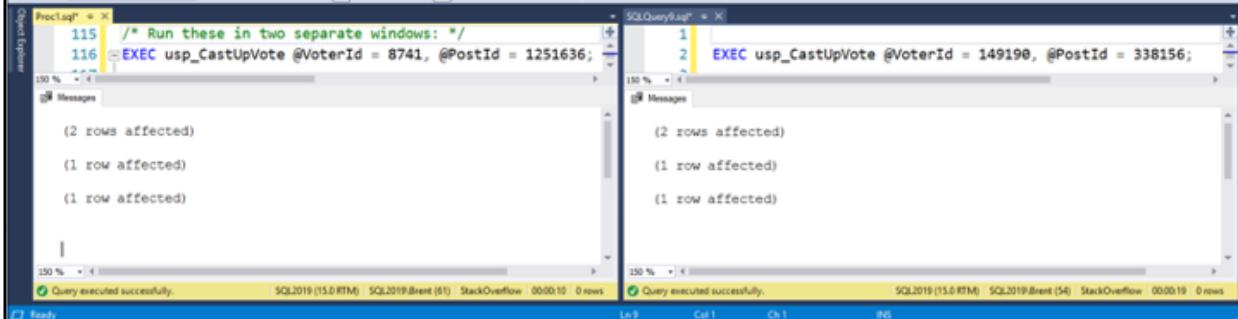
Lock both at once



3.3 p48



It works though! No deadlocks.



The screenshot shows two separate SSMS windows. The left window, titled 'ProcLog1', contains the following code and output:

```
115 /* Run these in two separate windows: */
116 EXEC usp_CastUpVote @VoterId = 8741, @PostId = 1251636;
```

Output:

```
(2 rows affected)
(1 row affected)
(1 row affected)
```

The right window, titled 'SQLQuery9.usp1', contains the following code and output:

```
1 EXEC usp_CastUpVote @VoterId = 149190, @PostId = 338156;
```

Output:

```
(2 rows affected)
(1 row affected)
(1 row affected)
```

Both windows show a green checkmark indicating 'Query executed successfully.' and the status bar at the bottom of each window shows '0 rows' affected.

Well, kinda: there's a catch. Notice the runtimes.



3.3 p50

```

CREATE OR ALTER PROC dbo.usp_CastUpVote
    @VoterId INT, @PostId INT AS
BEGIN
    BEGIN TRAN
        /* Update both the voter and the question-owner */
        UPDATE u
            SET LastAccessDate = CASE WHEN u.Id = @VoterId THEN GETDATE() ELSE u.LastAccessDate END,
                Reputation = CASE WHEN u.Id = p.OwnerUserId THEN u.Reputation + 1 ELSE u.Reputation END
        FROM dbo.Posts p
        INNER JOIN dbo.Users u ON (p.OwnerUserId = u.Id OR u.Id = @VoterId)
        WHERE p.Id = @PostId;

        WAITFOR DELAY '00:00:10';

        /* Cast an upvote: */
        INSERT INTO dbo.Votes (PostId, UserId, VoteTypeId, CreationDate)
            VALUES (@PostId, @VoterId, 2, GETDATE());

        /* Update the post's score: */
        UPDATE dbo.Posts
            SET Score = Score + 1
            WHERE Id = @PostId;
    COMMIT;
END;
GO

```

We're still locking both rows here.

So the longer the rest takes...

The longer it takes before others can start work.

3.3 p51

Another approach

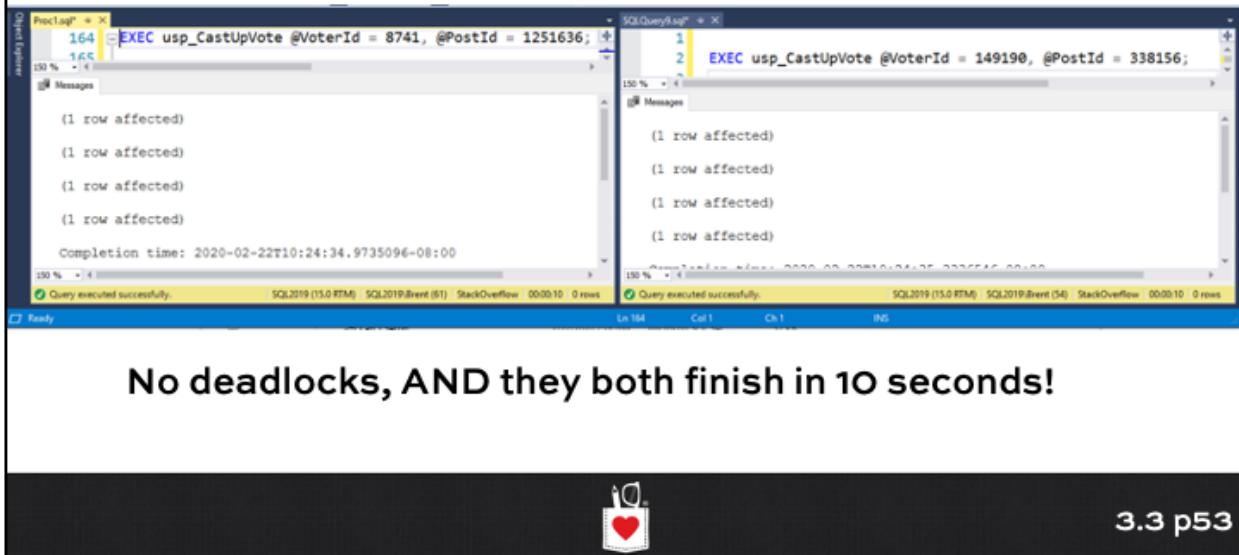
Do we really need
your Last Access
Date to be part of
the transaction?

```
127 /* Try doing one update outside of the transaction: */
128 CREATE OR ALTER PROC dbo.usp_CastUpVote
129   @VoterId INT, @PostId INT AS
130 BEGIN
131   /* Update the voter's LastAccessDate because they were active on Stack
132   UPDATE dbo.Users
133     SET LastAccessDate = GETDATE()
134   WHERE Id = @VoterId;
135
136
137 BEGIN TRAN
138
139 WAITFOR DELAY '00:00:10' /* 10 seconds */
140
141   /* Grant a reputation point to the post's owner: */
142   UPDATE u
143     SET Reputation = Reputation + 1
144   FROM dbo.Posts p
145   INNER JOIN dbo.Users u ON p.OwnerUserId = u.Id
146   WHERE p.Id = @PostId;
147
148   /* Cast an upvote: */
149   INSERT INTO dbo.Votes (PostId, UserId, VoteTypeId, CreationDate)
150     VALUES (@PostId, @VoterId, 2, GETDATE());
151
152   /* Update the post's score: */
153   UPDATE dbo.Posts
154     SET Score = Score + 1
155   WHERE Id = @PostId;
156
157 COMMIT;
158 END;
```

Before the TRAN

3.3 p52

Run 'em both at the same time...



The screenshot shows two separate SSMS sessions running simultaneously. Both sessions are executing the same stored procedure, `usp_CastUpVote`, with different parameters. Session 1 (ProcLog) is executing the query `EXEC usp_CastUpVote @VoterId = 8741, @PostId = 1251636;`. Session 2 (SQLQuery1) is executing the query `EXEC usp_CastUpVote @VoterId = 149190, @PostId = 338156;`. Both sessions show four rows affected and a completion time of approximately 10 seconds. The status bar at the bottom of each window indicates "Query executed successfully".

No deadlocks, AND they both finish in 10 seconds!



3.3 p53

The more you fix, the faster it goes

“Just remove the waitfor” = “make it all faster”

- Get faster hardware
- Tune indexes on the underlying tables
- Don’t hold transactions open on the app side

Try merging both Users updates into a single query
(where it’s the Voter, OR it’s the question-owner)

Do the LastAccessDate update outside of the transaction (does it really matter?)



3.3 p54

Session agenda

Basics:

- 3 concurrency issues
- 3 ways to fix ‘em all
- 1 “fix” that makes things worse: NOLOCK

One real fix: work on tables in a consistent order

- Demo: unrealistic query
- Demo: realistic query

Using sp_BlitzLock to find the queries you need to fix



3.3 p55

We are here.

sp_BlitzLock helps you spot 'em.

Thanks to SQL 2012 or newer, the default system health Extended Events session, and Erik Darling:

check_id	database_name	object_name	finding_group	finding
1	sp_BlitzLock	Feb 17 2020 12:00AM	SQL Server First Responder Kit	http://FirstResponderKit.org/ To get help or add your own contributions, join us at http://FirstResponderKit.org .
2	1	StackOverflow	-	Total database locks This database had 4 deadlocks.
3	1	StackOverflow2013	-	Total database locks This database had 2 deadlocks.
4	2	StackOverflow	StackOverflow.dbo.Users	Total object deadlocks This object was involved in 4 deadlock(s).
5	2	StackOverflow2013	StackOverflow2013.dbo.Comments	Total object deadlocks This object was involved in 2 deadlock(s).
6	2	StackOverflow	PK_Users_Id	Total index deadlocks This index was involved in 4 deadlock(s).
7	2	StackOverflow2013	StackOverflow2013.dbo.Commen...	Total index deadlocks This index was involved in 2 deadlock(s).
8	5	StackOverflow	-	Login, App, and Host locking This database has had 4 instances of deadlocks involving the login SQL2019Brent from the application Microsoft SQL Server Management Studio - Query on host SQL2019
9	5	StackOverflow2013	-	Login, App, and Host locking This database has had 2 instances of deadlocks involving the login UNKNOWN from the application SQLQueryStress on host SQL2019
10	5	StackOverflow2013	-	Login, App, and Host locking This database has had 2 instances of deadlocks involving the login SQL2019Brent from the application SQLQueryStress on host SQL2019
11	8	StackOverflow	sysLocks.dbo.sys_Conditions	Global Processor Locks The global processor lock 'CardTable has been involved in 5 deadlock(s)'.

Top result set: list of deadlocks

1	sp_BlitzLock										
14	Regular Deadlock	2020-02-21 08:49:42.4720000	StackOverflow2013	Deadlock #2, Query #14	<code>sp_executesql N'EXEC sp_IndexLock @input1'</code>	<code>select [StackOverflow2013].[dbo].[Comments] as [object]</code>	read committed (2)	-	-	-	0
15	Regular Deadlock	2020-02-22 08:46:15.6500000	StackOverflow	Deadlock #3, Query #1 - VICTIM	<code>sp_executesql N'BEGIN TRAN; DECLARE @Victim INT = 1;</code>	<code>select [StackOverflow].[dbo].[Users] as [object]</code>	read committed (2)	-	-	-	3
16	Regular Deadlock	2020-02-22 08:46:15.6500000	StackOverflow	Deadlock #3, Query #1 - VICTIM	<code>sp_executesql N'BEGIN TRAN; DECLARE @Victim INT = 2;</code>	<code>select [StackOverflow].[dbo].[Users] as [object]</code>	read committed (2)	-	-	-	3
17	Regular Deadlock	2020-02-22 09:57:35.9700000	StackOverflow	Deadlock #4, Query #1 - VICTIM	<code>sp_executesql N'EXEC sp_CertVote @Victim = 3741, @P_</code>	<code>select [StackOverflow].[dbo].[Users] as [object]</code>	read committed (2)	-	-	-	2
18	Regular Deadlock	2020-02-22 09:57:35.9700000	StackOverflow	Deadlock #4, Query #2	<code>sp_executesql N'EXEC sp_CertVote @Victim = 149190,</code>	<code>select [StackOverflow].[dbo].[Users] as [object]</code>	read committed (2)	-	-	-	2
19	Regular Deadlock	2020-02-22 10:01:23.8300000	StackOverflow	Deadlock #5, Query #2 - VICTIM	<code>sp_executesql N'EXEC sp_CertVote @Victim = 149190,</code>	<code>select [StackOverflow].[dbo].[Users] as [object]</code>	read committed (2)	-	-	-	2
20	Regular Deadlock	2020-02-22 10:01:23.8300000	StackOverflow	Deadlock #5, Query #1	<code>sp_executesql N'EXEC sp_CertVote @Victim = 3741, @P_</code>	<code>select [StackOverflow].[dbo].[Users] as [object]</code>	read committed (2)	-	-	-	2
21	Regular Deadlock	2020-02-22 10:01:23.8300000	StackOverflow	Deadlock #6, Query #2 - VICTIM	<code>sp_executesql N'EXEC sp_CertVote @Victim = 345190,</code>	<code>select [StackOverflow].[dbo].[Users] as [object]</code>	read committed (2)	-	-	-	2
22	Regular Deadlock	2020-02-22 10:02:11.4560000	StackOverflow	Deadlock #6, Query #1	<code>sp_executesql N'EXEC sp_CertVote @Victim = 3741, @P_</code>	<code>select [StackOverflow].[dbo].[Users] as [object]</code>	read committed (2)	-	-	-	2

Scroll to the far right, and you also get the deadlock graph. Save it as an XDL file, then re-open it in SSMS or in SentryOne Plan Explorer.



3.3 p57

Bottom results: analytics

This is what really helps me get to the bottom of it:

check_id	database_name	object_name	finding_group	finding
1	-1	rp_BlkLock Feb 17 2020 12:00AM	SQL Server First Responder Kit	http://FirstResponderKit.org/ To get help or add your own contributions, join us at http://FirstResponderKit.org
2	1	StackOverflow	-	Total database locks This database had 4 deadlocks.
3	1	StackOverflow2013	-	Total database locks This database had 2 deadlocks.
4	2	StackOverflow	StackOverflow dbo.Users	Total object deadlocks The object was involved in 4 deadlock(s).
5	2	StackOverflow2013	StackOverflow2013dbo.Comments	Total object deadlocks The object was involved in 2 deadlock(s).
6	2	StackOverflow	PK_Users_Id	Total object deadlocks This index was involved in 4 deadlock(s).
7	2	StackOverflow2013	StackOverflow2013dbo.Commen...	Total index deadlocks The index was involved in 2 deadlock(s).
8	5	StackOverflow	-	Login, App, and Host locking This database has had 4 instances of deadlocks involving the login SQL2019Brent from the application Microsoft SQL Server Management.
9	5	StackOverflow2013	-	Login, App, and Host locking This database has had 2 instances of deadlocks involving the login UNKNOWN from the application SQLQueryStress on host SQL2019.
10	5	StackOverflow2013	-	Login, App, and Host locking This database has had 2 instances of deadlocks involving the login SQL2019Brent from the application Microsoft SQL Server Management.
11	8	StackOverflow	StackOverflow.dbo.usp_CastUpV...	Stored Procedure Deadlocks The stored procedure usp_CastUpVote has been involved in 5 deadlocks.
12	8	StackOverflow2013	StackOverflow2013dbo.usp_Co...	Stored Procedure Deadlocks The stored procedure usp_CCommentInsert_V1 has been involved in 14 deadlocks.
13	8	StackOverflow2013	StackOverflow2013dbo.usp_Ind...	Stored Procedure Deadlocks The stored procedure dbo.usp_IndexLab1 has been involved in 14 deadlocks.
14	9	StackOverflow	dbo.Users	More Info - Table EXEC sp_BlitzIndex @DatabaseName = 'StackOverflow', @SchemaName = 'dbo', @TableName = 'Users';
15	9	StackOverflow2013	dbo.Comments	More Info - Table EXEC sp_BlitzIndex @DatabaseName = 'StackOverflow2013', @SchemaName = 'dbo', @TableName = 'Comments';
16	11	StackOverflow	-	Total database deadlock wait time This database has had 0:00:00.45 [s/h/m/s] of deadlock wait time.
17	11	StackOverflow2013	-	Total database deadlock wait time This database has had 0:00:00.49 [s/h/m/s] of deadlock wait time.



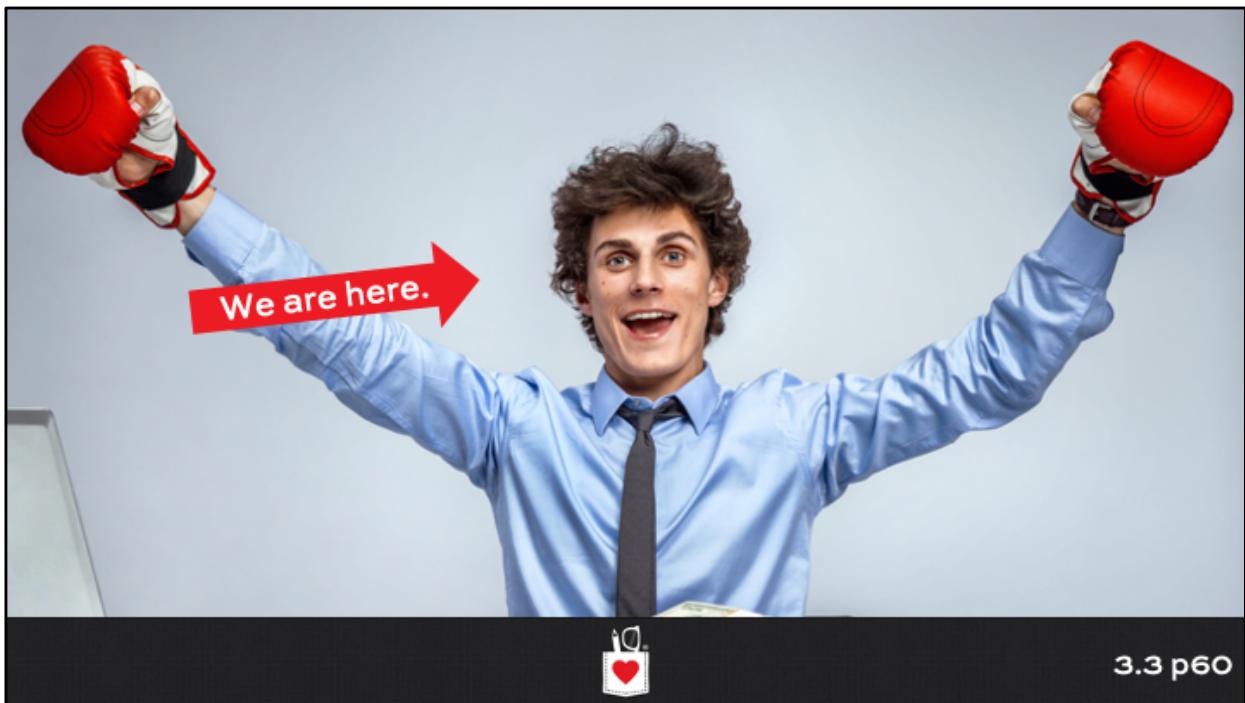
3.3 p58

Bottom results: analytics

Tables & indexes: use my D.E.A.T.H. Method on 'em

Queries: hit tables in a consistent order, tune txns

object_name	finding_group	finding
SQL Server First Responder Kit	http://FirstResponderKit.org/	To get help or add your own contributions, join us at http://FirstResponderKit.org .
-	Total database locks	This database had 4 deadlocks.
-	Total database locks	This database had 2 deadlocks.
StackOverflow.dbo.Users	Total object deadlocks	This object was involved in 4 deadlock(s).
StackOverflow2013 dbo.Comments	Total object deadlocks	This object was involved in 2 deadlock(s).
PK_Users_Id	Total Index deadlocks	This index was involved in 4 deadlock(s).
StackOverflow2013 dbo.Commen...	Total Index deadlocks	This index was involved in 2 deadlock(s).
-	Login, App, and Host locking	This database has had 4 instances of deadlocks involving the login SQL2019\Brent from the application Microsoft SQL Server Management
-	Login, App, and Host locking	This database has had 2 instances of deadlocks involving the login UNKNOWN from the application SQLQueryStress on host SQL2019
-	Login, App, and Host locking	This database has had 2 instances of deadlocks involving the login SQL2019\Brent from the application SQLQueryStress on host SQL2019
StackOverflow dbo.usp_CastUpV...	Stored Procedure Deadlocks	The stored procedure dbo.usp_CastUpVote has been involved in 5 deadlocks.
StackOverflow2013 dbo.usp_Co...	Stored Procedure Deadlocks	The stored procedure dbo.usp_CommentInsert_V1 has been involved in 14 deadlocks.
StackOverflow2013 dbo.usp_Ind...	Stored Procedure Deadlocks	The stored procedure dbo.usp_IndexLab1 has been involved in 14 deadlocks.
dbo.Users	More Info - Table	EXEC sp_BlitzIndex @DatabaseName = 'StackOverflow', @SchemaName = 'dbo', @TableName = 'Users';
dbo.Comments	More Info - Table	EXEC sp_BlitzIndex @DatabaseName = 'StackOverflow2013', @SchemaName = 'dbo', @TableName = 'Comments';
-	Total database deadlock w...	This database has had 0:00:00:45 [d/h/m/s] of deadlock wait time.
-	Total database deadlock w...	This database has had 0:00:06:49 [d/h/m/s] of deadlock wait time.



3 ways to fix concurrency issues

1. Have enough indexes to make your queries fast, but not so many that they slow down DUIs, making them hold more locks for longer times.

(I cover this in Mastering Index Tuning.)

2. Tune your transactional code.

(This module focuses on this topic.)

3. Use the right isolation level for your app's needs.

(I cover this in Mastering Server Tuning.)



3.3 p61