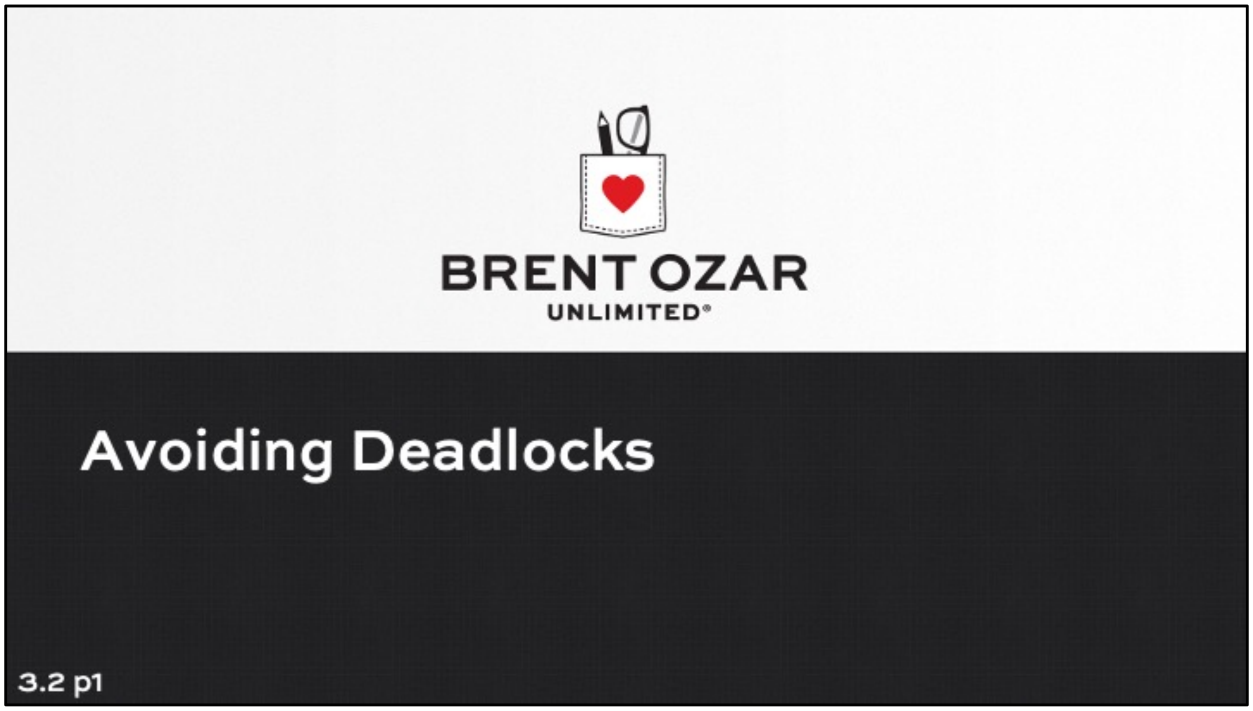
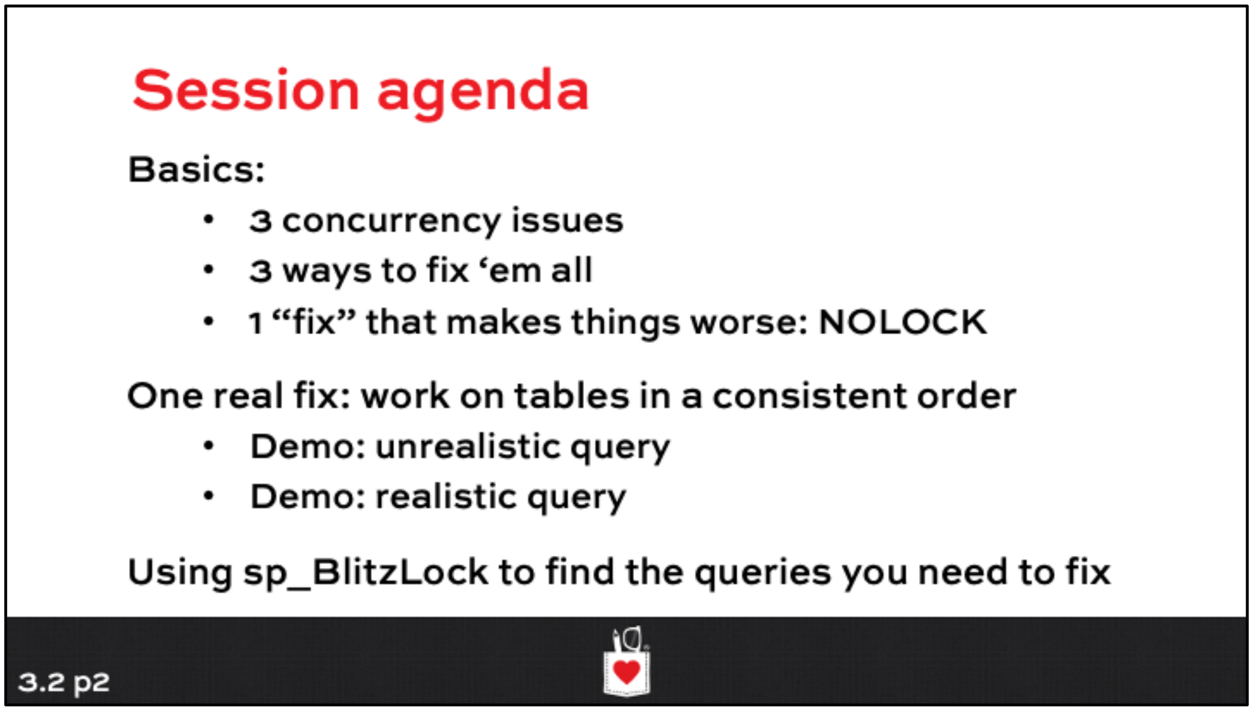


BRENT OZAR
UNLIMITED®

Avoiding Deadlocks

3.2 p1



Session agenda

Basics:

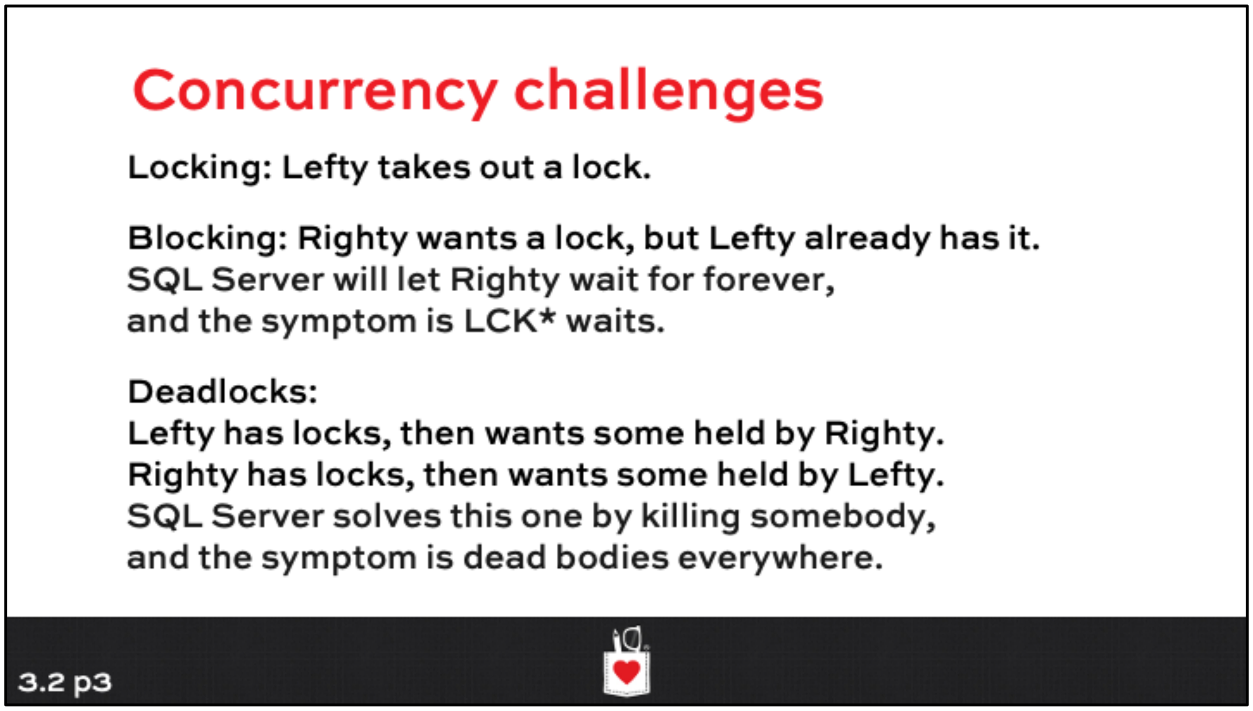
- 3 concurrency issues
- 3 ways to fix 'em all
- 1 "fix" that makes things worse: NOLOCK

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

3.2 p2



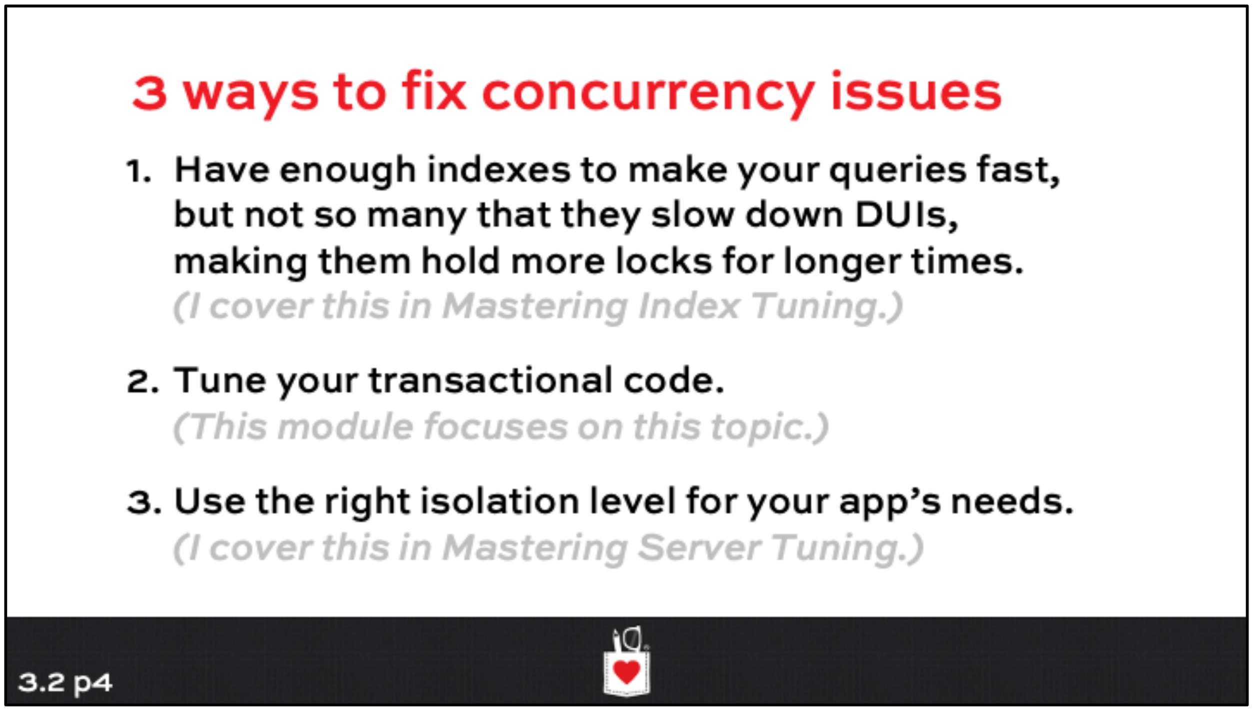
Concurrency challenges

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.

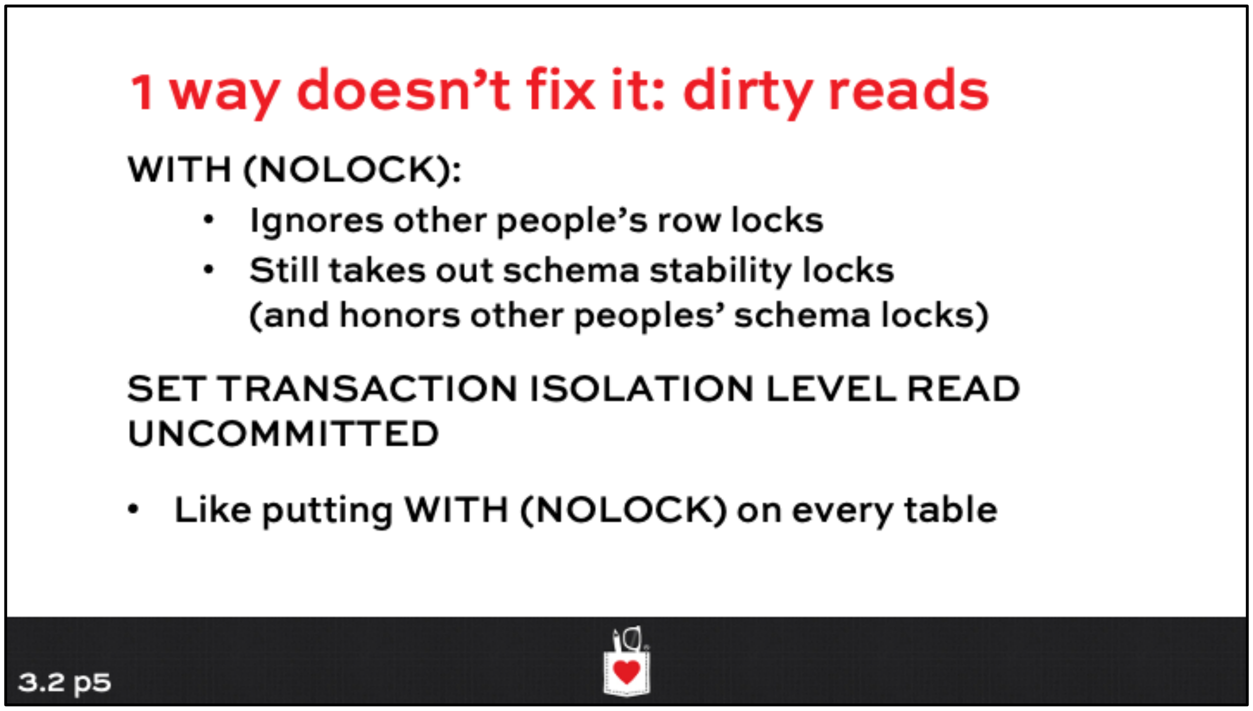
3.2 p3



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.)

3.2 p4



1 way doesn't fix it: dirty reads

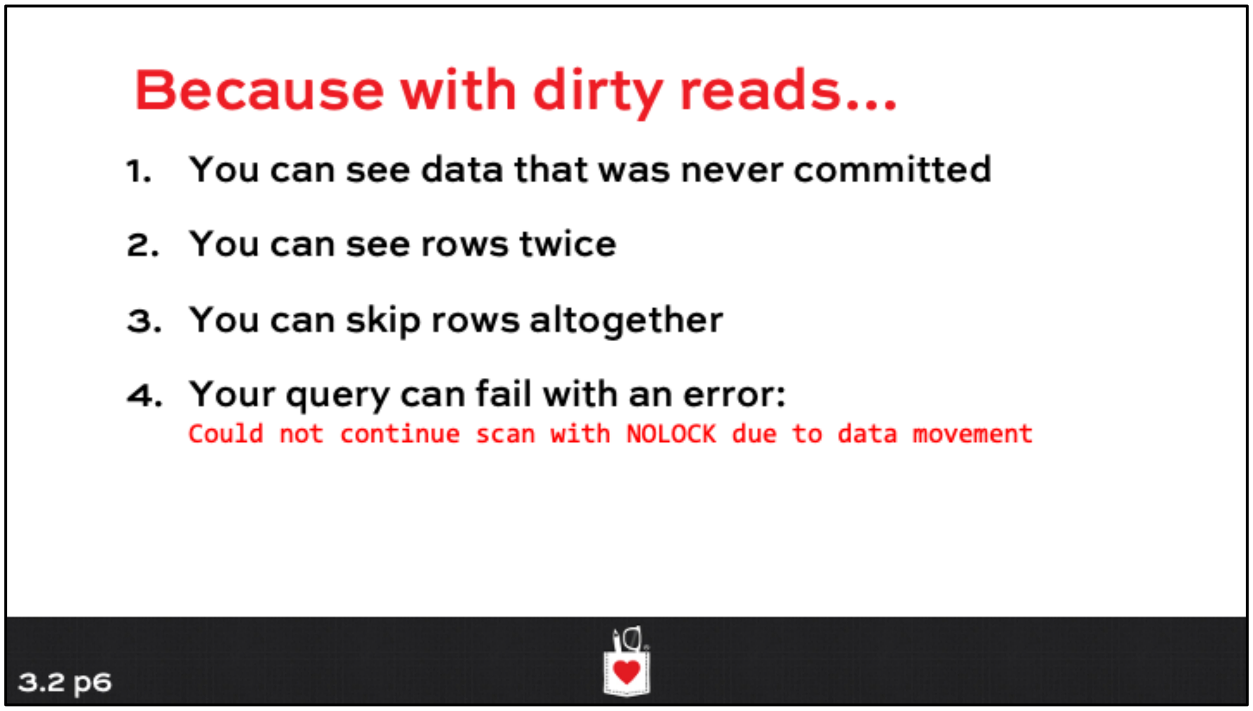
WITH (NOLOCK):

- 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

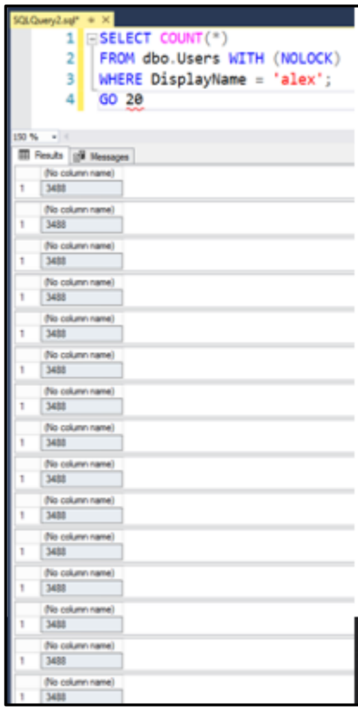
3.2 p5



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
   ```

3.2 p6



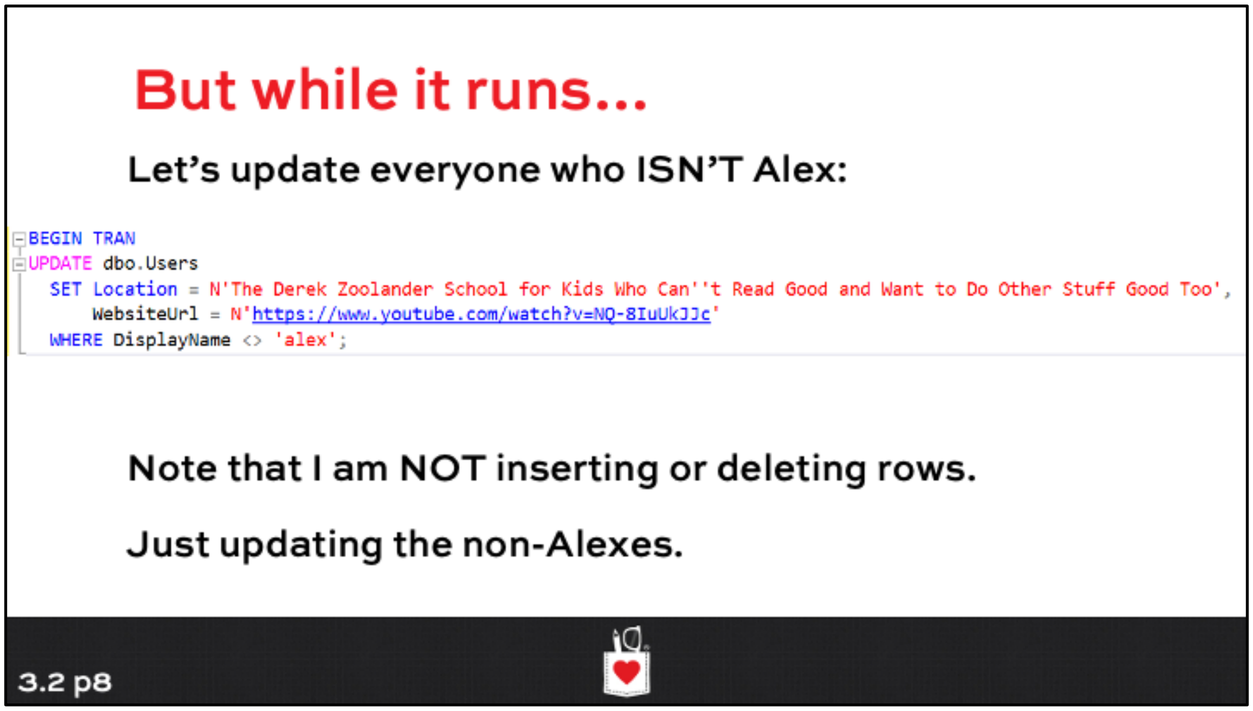
```sql
SELECT COUNT(*)
FROM dbo.Users WITH (NOLOCK)
WHERE DisplayName = 'alex';
GO 20
```

```
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
  (No column name)
1 3488
```

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.



But while it runs…

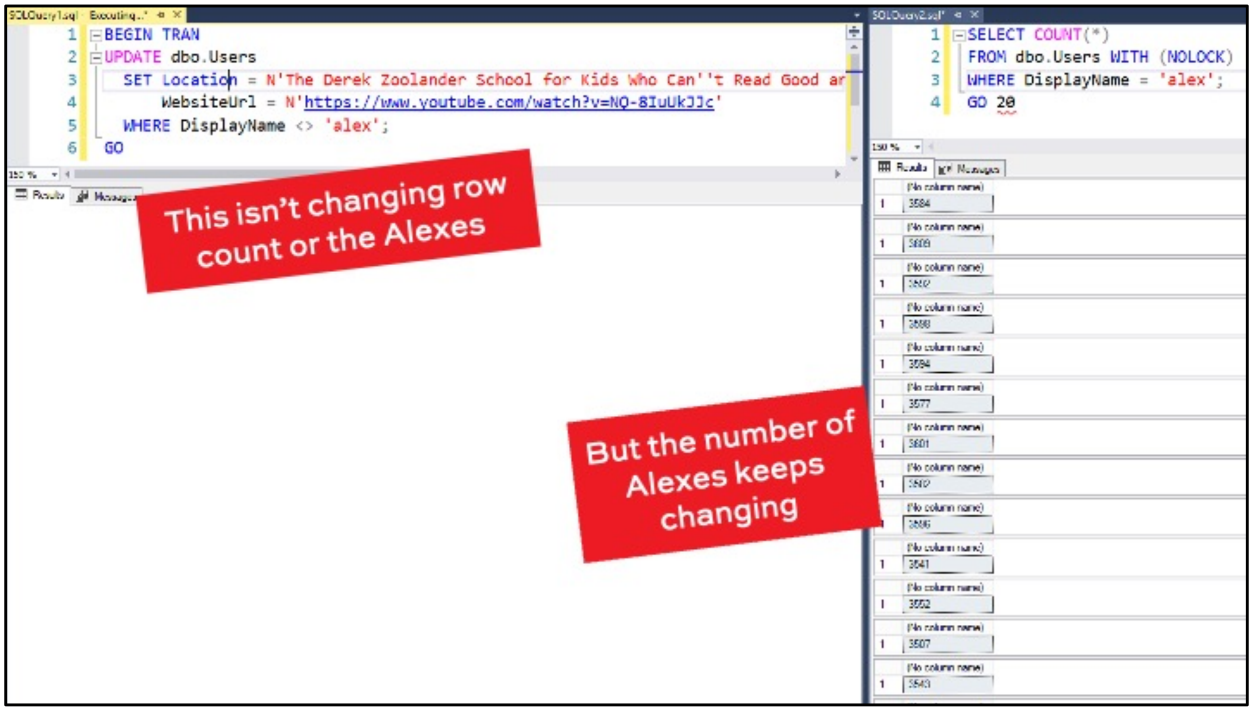
Let’s update everyone who ISN’T Alex:

```sql
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.

3.2 p8



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

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

```
(No column name)
1  3584
(No column name)
1  3609
(No column name)
1  3602
(No column name)
1  3598
(No column name)
1  3594
(No column name)
1  3577
(No column name)
1  3601
(No column name)
1  3602
(No column name)
1  3596
(No column name)
1  3541
(No column name)
1  3552
(No column name)
1  3507
(No column name)
1  3543
```

This isn't changing row count or the Alexes
But the number of Alexes keeps changing

That's NOLOCK.

And that's not a bug.

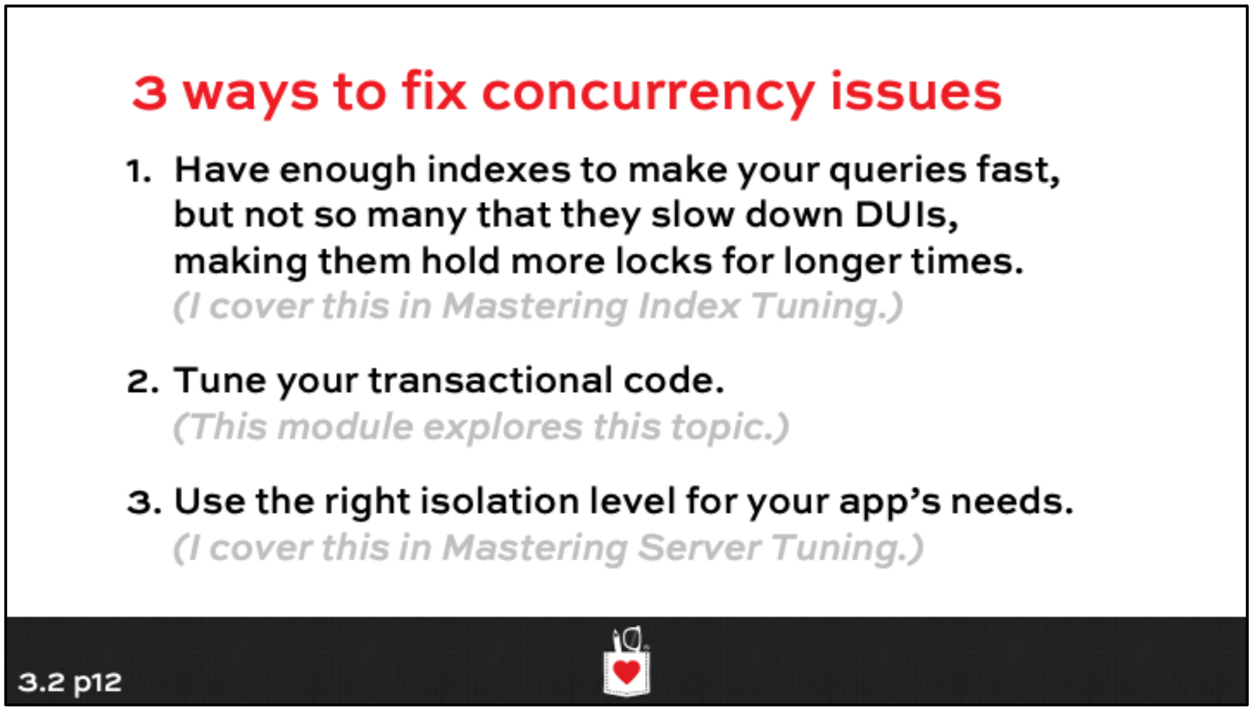
That's what it's designed to do.

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.

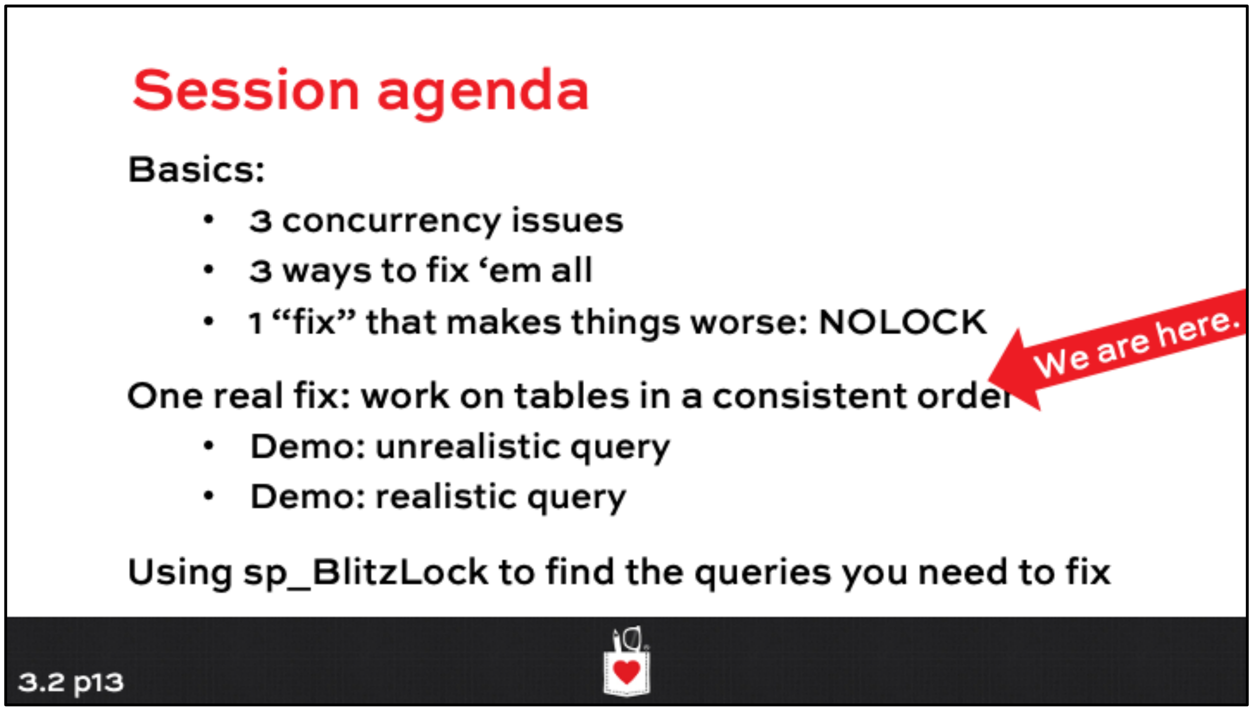
3.2 p11



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.)

3.2 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

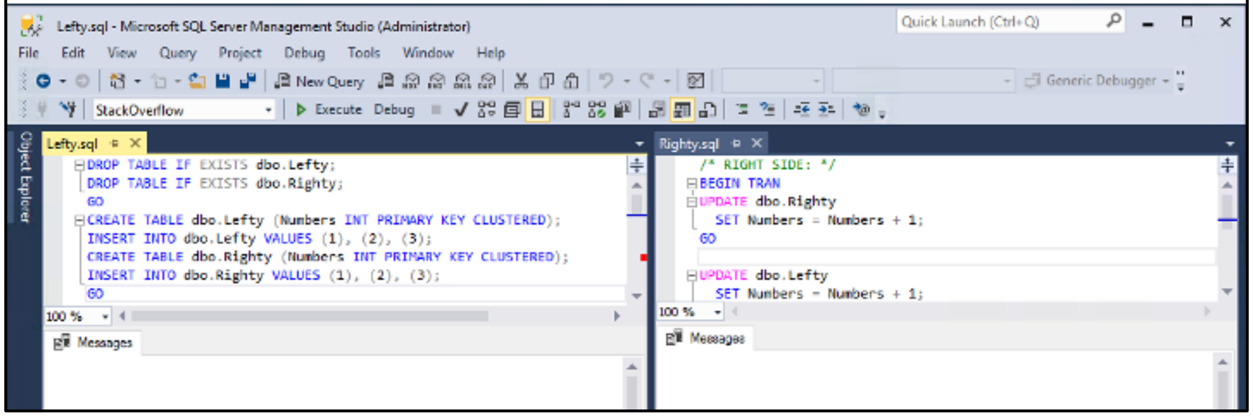
Using sp_BlitzLock to find the queries you need to fix

We are here.
3.2 p13

Start SSMS with two windows.

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

In Lefty, create & populate the tables.



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
```

Righty.sql:

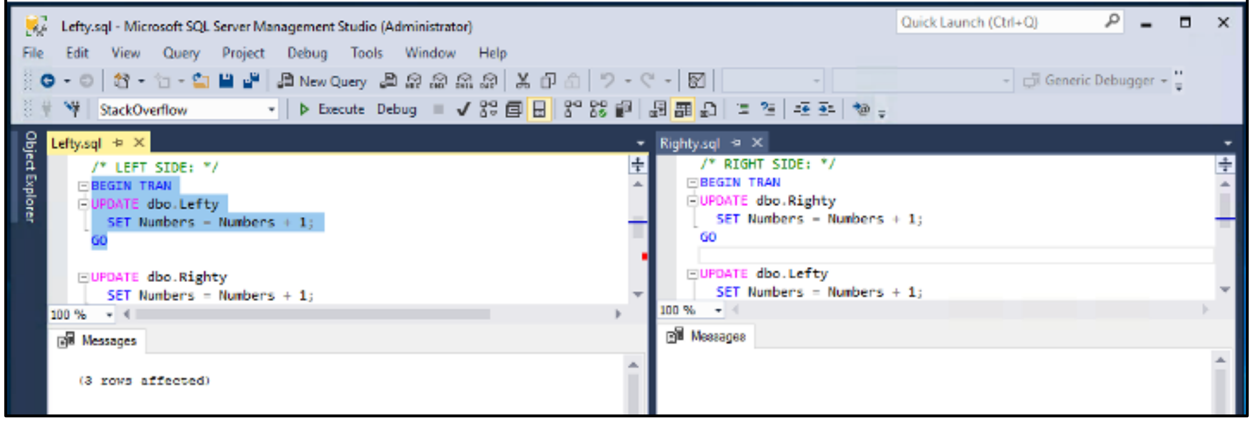
```
/* RIGHT SIDE: */
BEGIN TRAN
UPDATE dbo.Righty
  SET Numbers = Numbers + 1;
GO

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

Lefty.sql - Microsoft SQL Server Management Studio (Administrator)
Quick Launch (Ctrl+Q)
File Edit View Query Project Debug Tools Window Help
New Query
Generic Debugger
StackOverflow
Execute Debug
Object Explorer
100 %
Messages

In Lefty, start a transaction.

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

The left window is now locking dbo.Lefty.



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

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

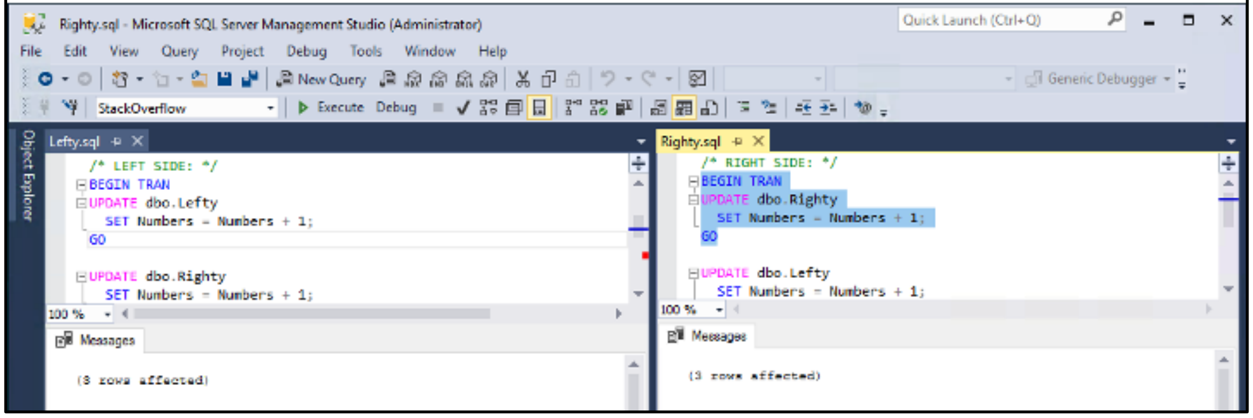
```
(3 rows affected)
```

```
/* 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.

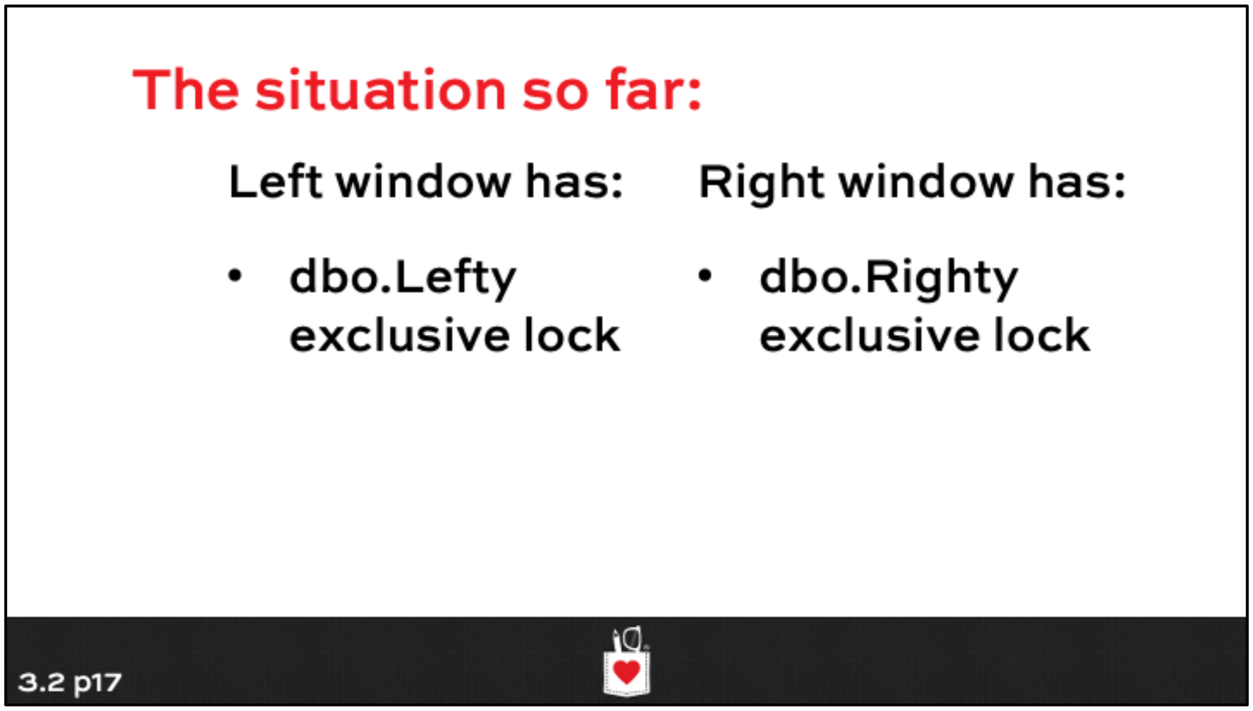

Righty.sql - Microsoft SQL Server Management Studio (Administrator)
Quick Launch (Ctrl+Q)
File Edit View Query Project Debug Tools Window Help
New Query
Generic Debugger
StackOverflow
Execute Debug
Object Explorer
Lefty.sql
Righty.sql
100 %
Messages

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

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

```
(3 rows affected)
```

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

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

```
(3 rows affected)
```



The situation so far:

Left window has:

- dbo.Lefty exclusive lock

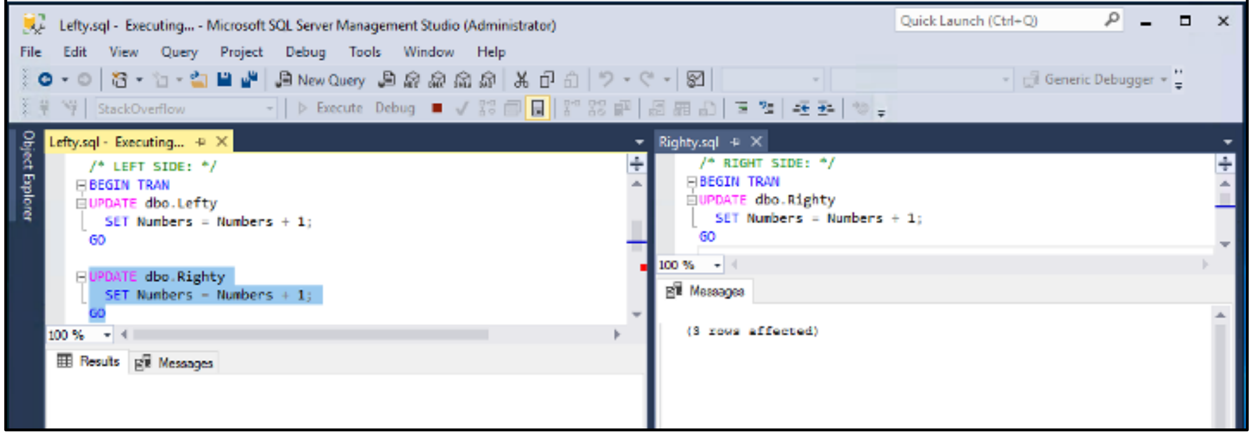
Right window has:

- dbo.Righty exclusive lock

3.2 p17

In Lefty, update dbo.Righty.

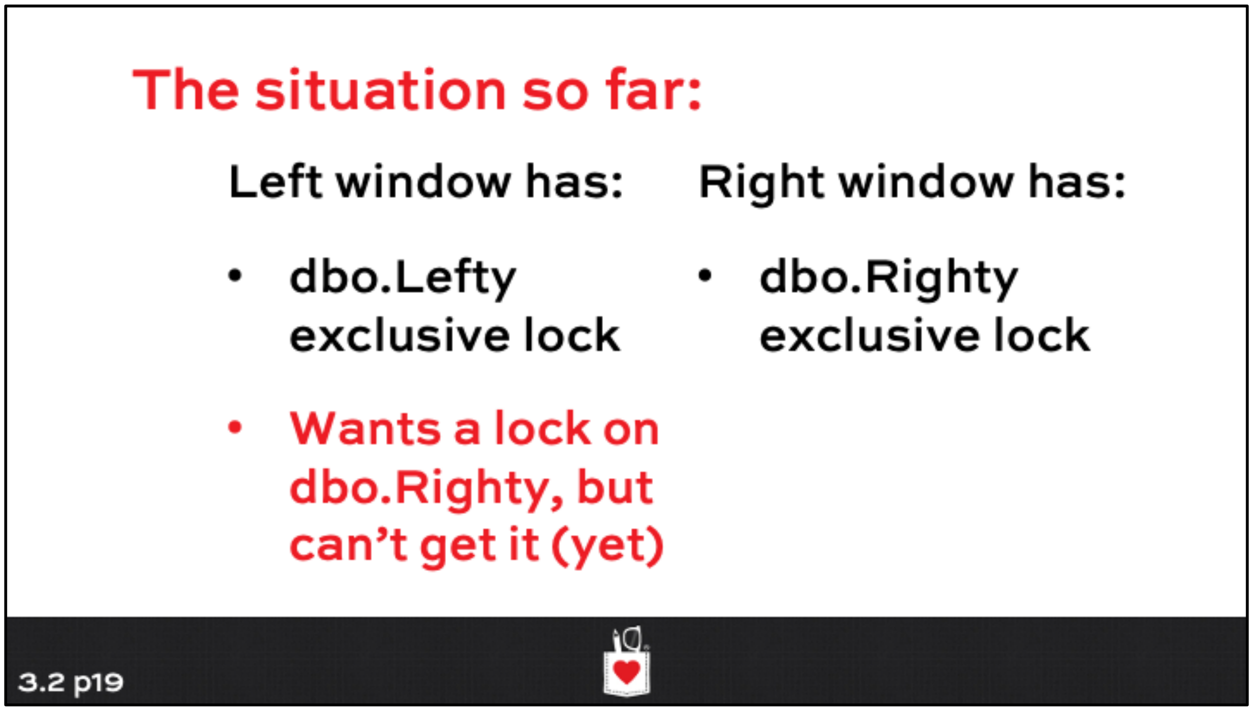
The update starts running, but is blocked, and sits there waiting for dbo.Righty to commit or roll back.


Lefty.sql - Executing... - Microsoft SQL Server Management Studio (Administrator)
File Edit View Query Project Debug Tools Window Help
New Query
StackOverflow
Execute Debug
Generic Debugger
Object Explorer
Lefty.sql - Executing...
Righty.sql
100 %
Results
Messages

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

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

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

```
(8 rows affected)
```



The situation so far:

Left window has:

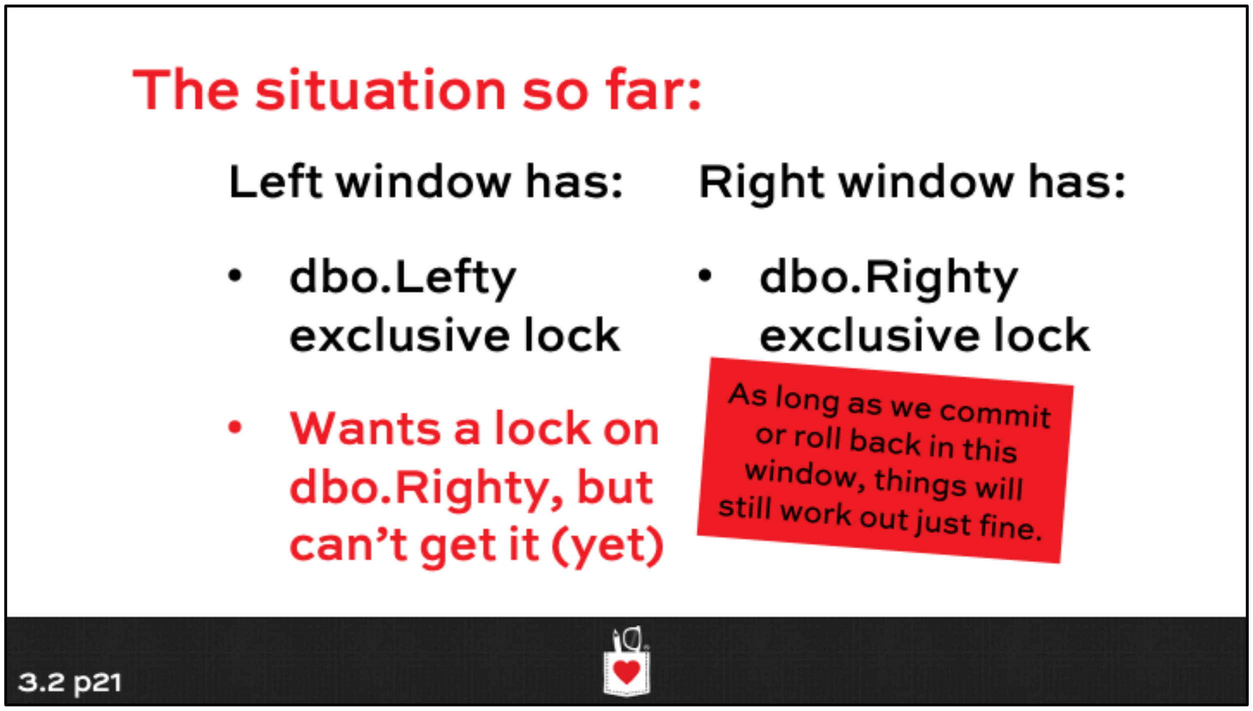
- dbo.Lefty exclusive lock
- Wants a lock on dbo.Righty, but can't get it (yet)

Right window has:

- dbo.Righty exclusive lock

3.2 p19

This is blocking.
One query is sitting around bored.



The situation so far:

Left window has:

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

Right window has:

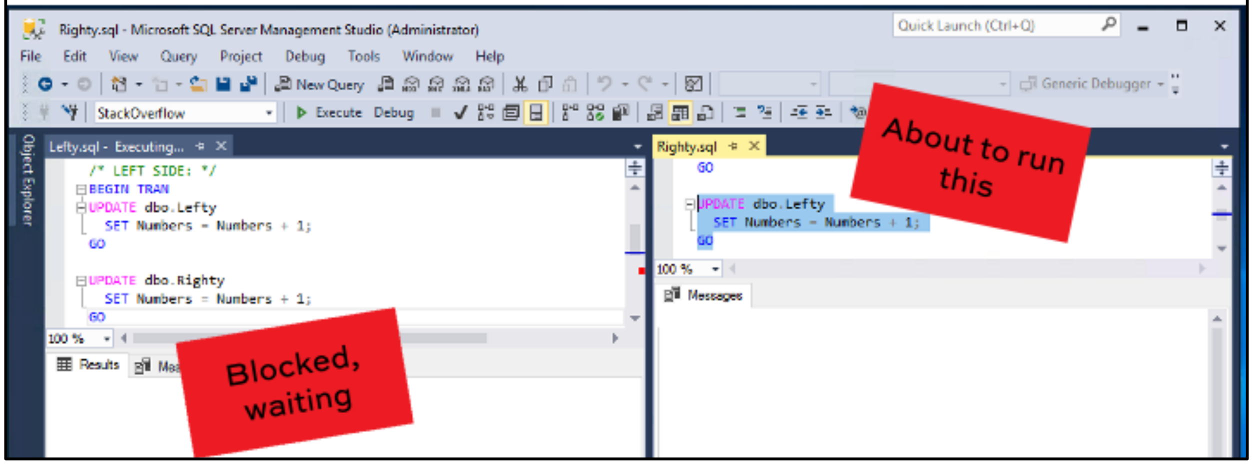
- dbo.Righty exclusive lock

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

3.2 p21

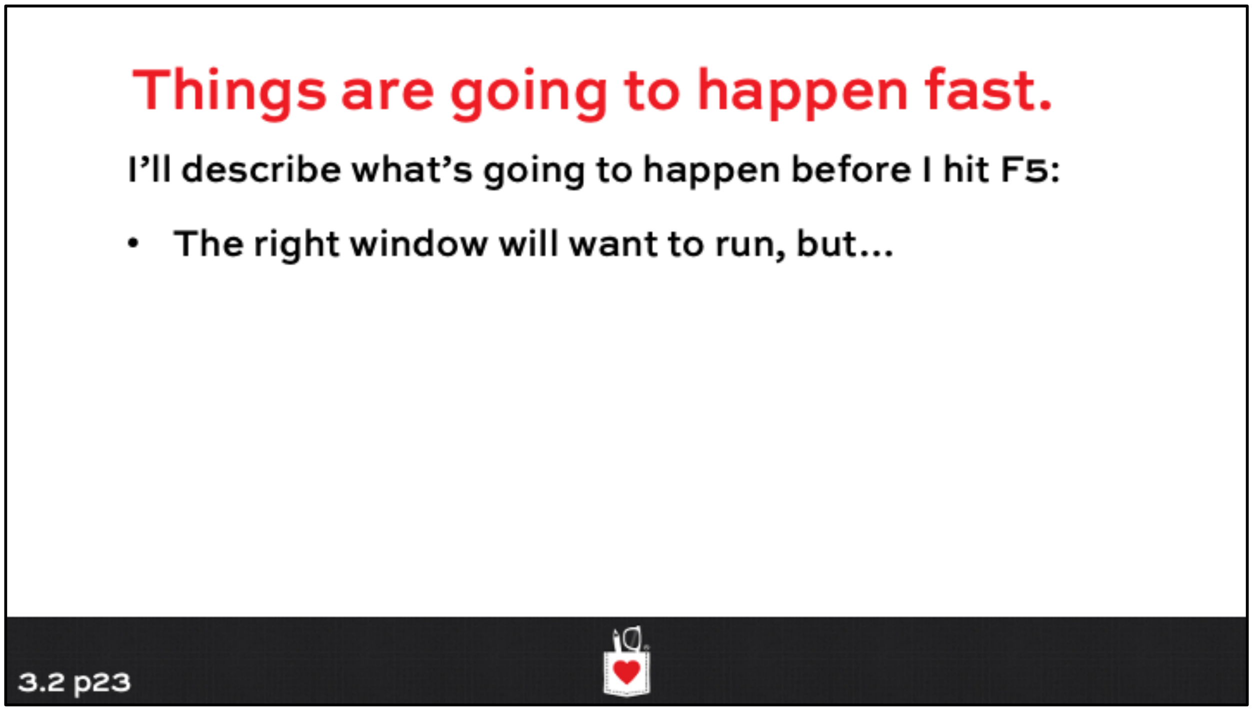
But let’s do something terrible.

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


Righty.sql - Microsoft SQL Server Management Studio (Administrator)
File Edit View Query Project Debug Tools Window Help
New Query
StackOverflow
Execute Debug
Lefty.sql - Executing...
Righty.sql
Blocked, waiting
About to run this
Results
Messages

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

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

```
GO

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

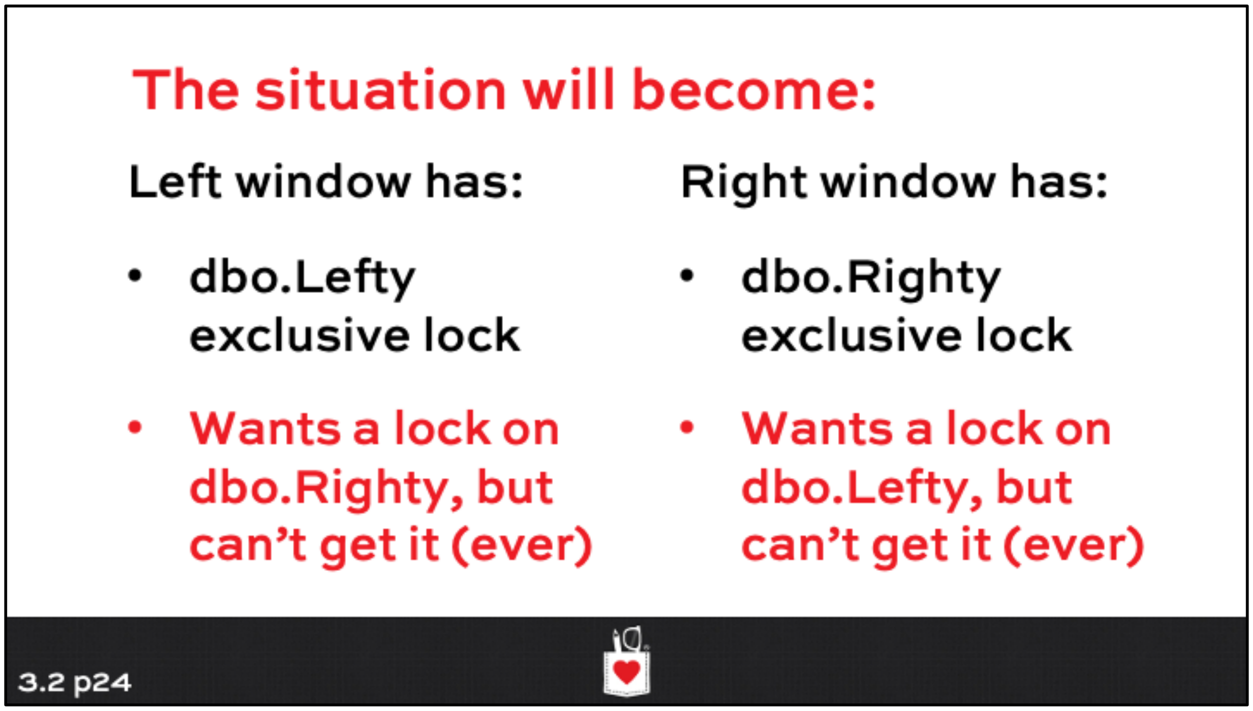


Things are going to happen fast.

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

- The right window will want to run, but…

3.2 p23



The situation will become:

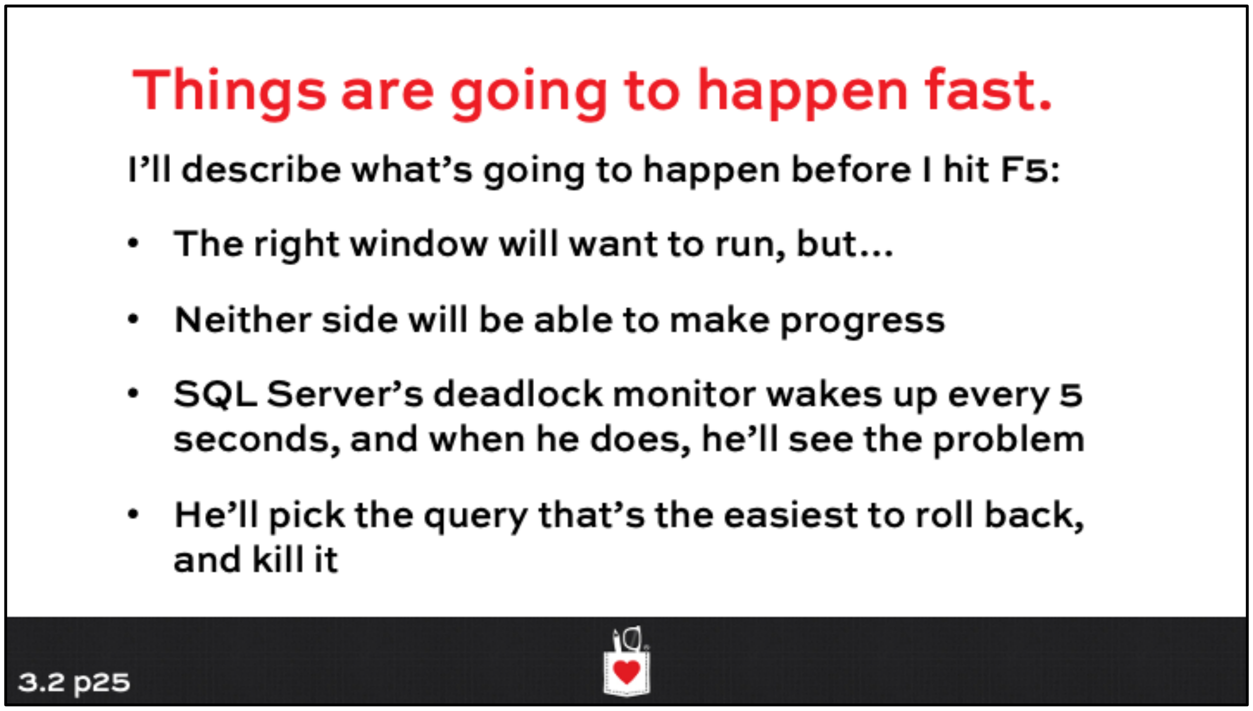
Left window has:

- 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)

3.2 p24



Things are going to happen fast.

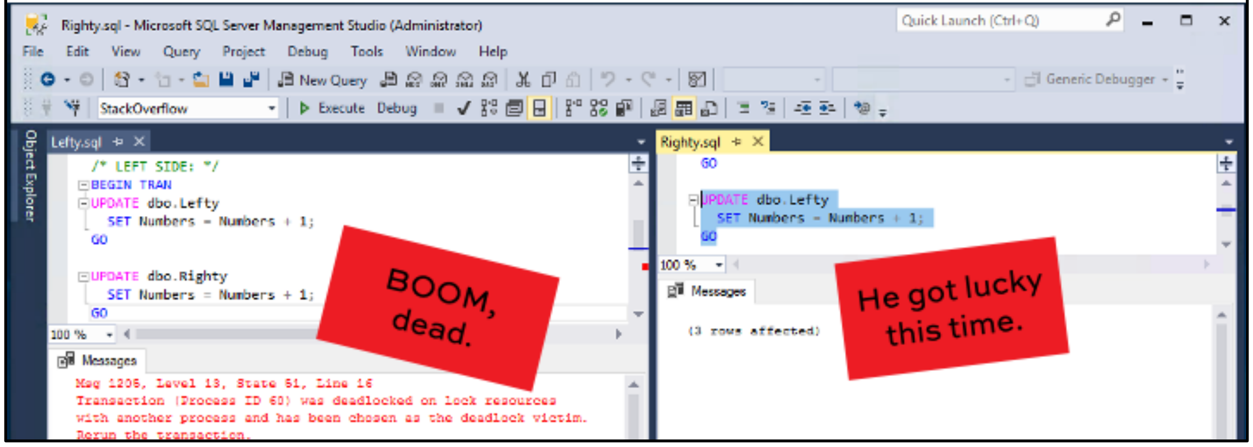
I'll describe what's going to happen before I hit F5:

- 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

3.2 p25

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

Within 5 seconds, SQL Server kills one.

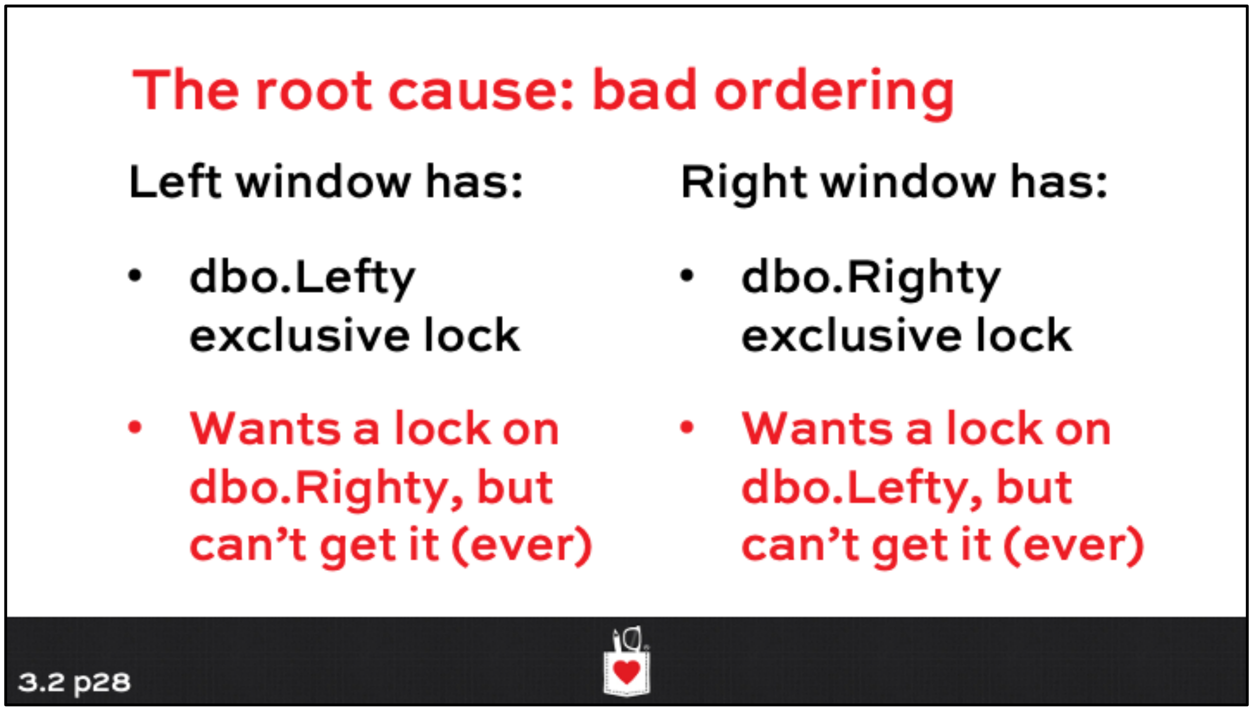

Righty.sql - Microsoft SQL Server Management Studio (Administrator)
File Edit View Query Project Debug Tools Window Help
New Query
Execute Debug
StackOverflow
Lefty.sql
Righty.sql
Messages
BOOM, dead.
He got lucky this time.

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

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

```
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.
```

```
GO

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

```
(3 rows affected)
```



The root cause: bad ordering

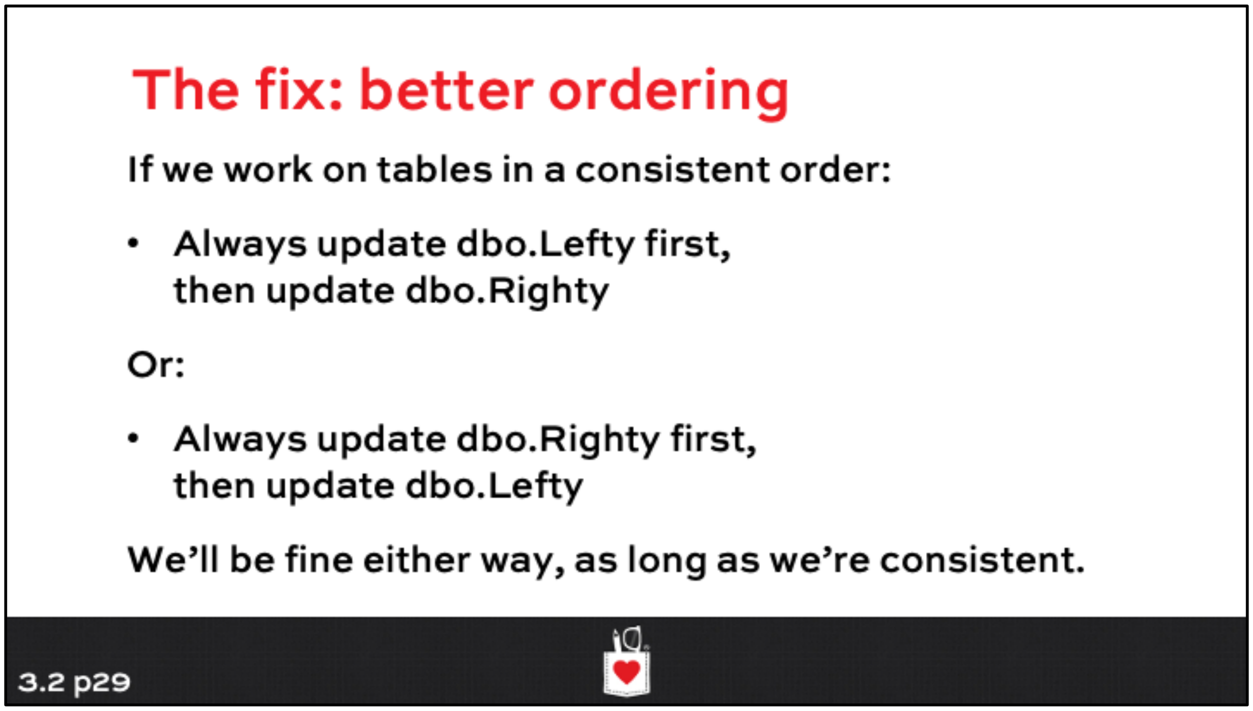
Left window has:

- 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)

3.2 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

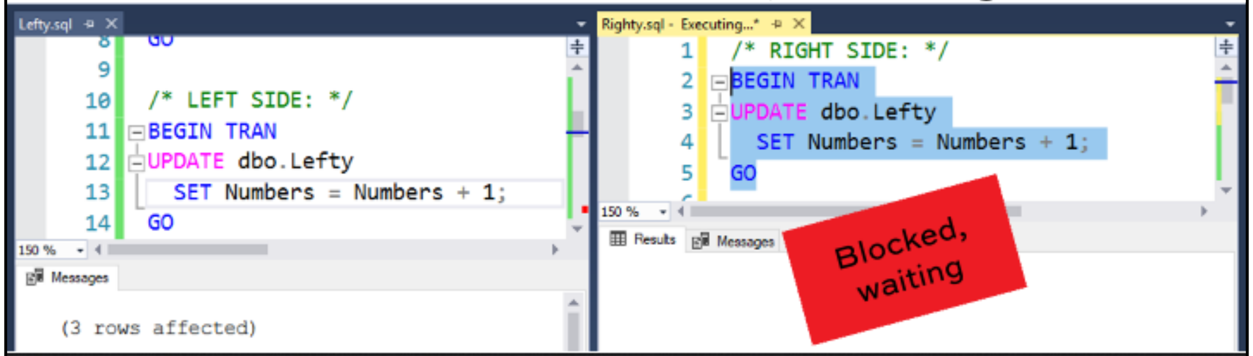
We'll be fine either way, as long as we're consistent.

3.2 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:



Lefty.sql

```
GO

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

Messages

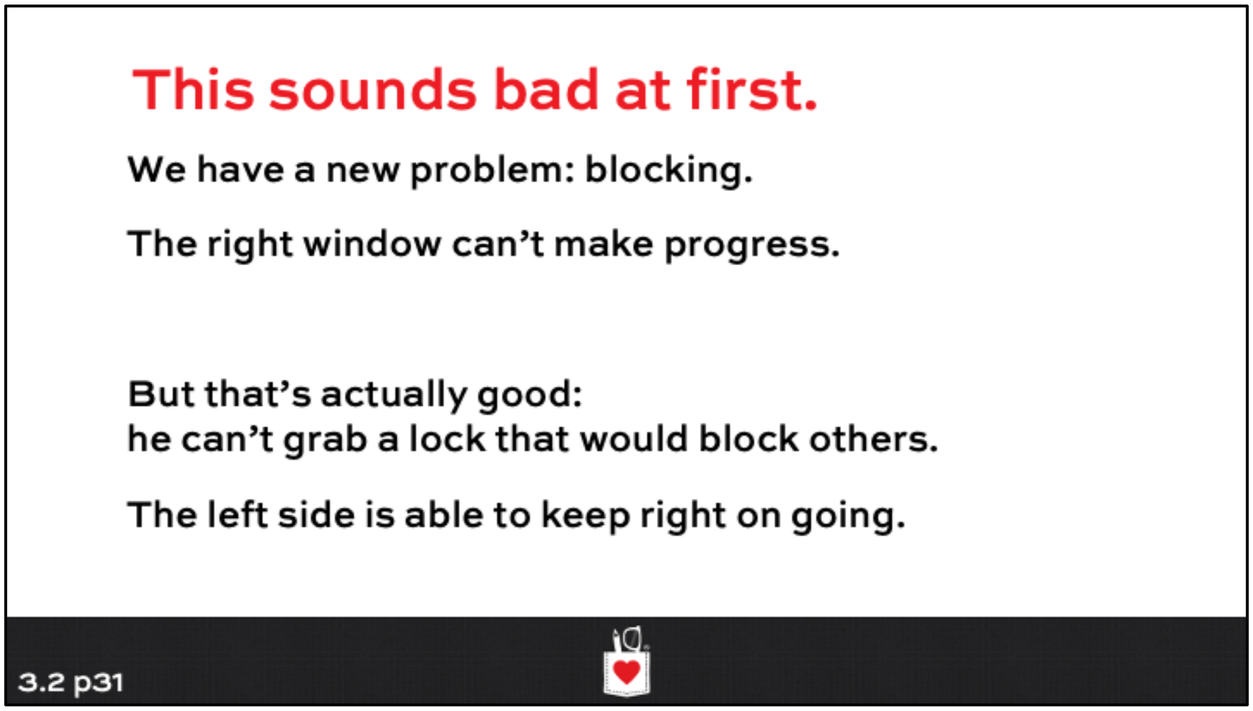
```
(3 rows affected)
```

Righty.sql - Executing...*

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

Results | Messages

3.2 p30



This sounds bad at first.

We have a new problem: blocking.

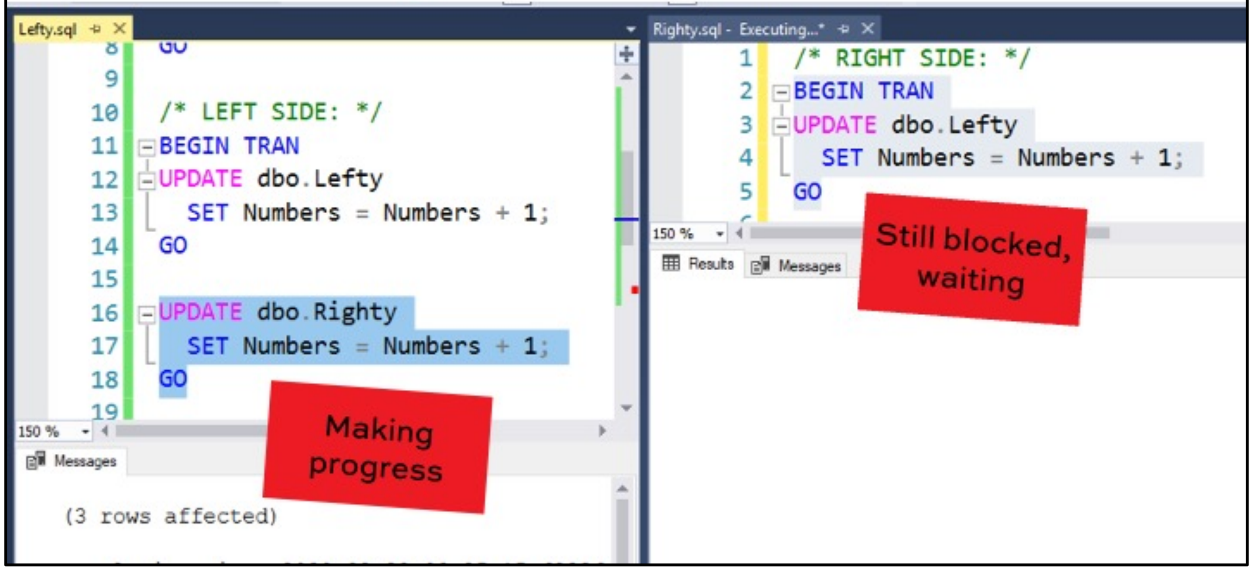
The right window can't make progress.

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.

3.2 p31

Continuing in the left window…



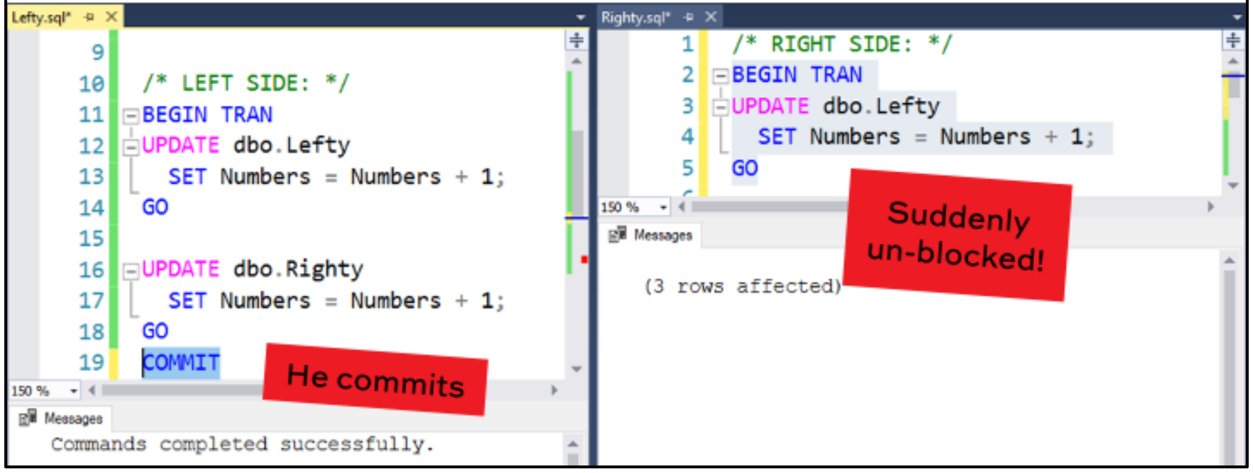
```
GO

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

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

```
(3 rows affected)
```

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

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.

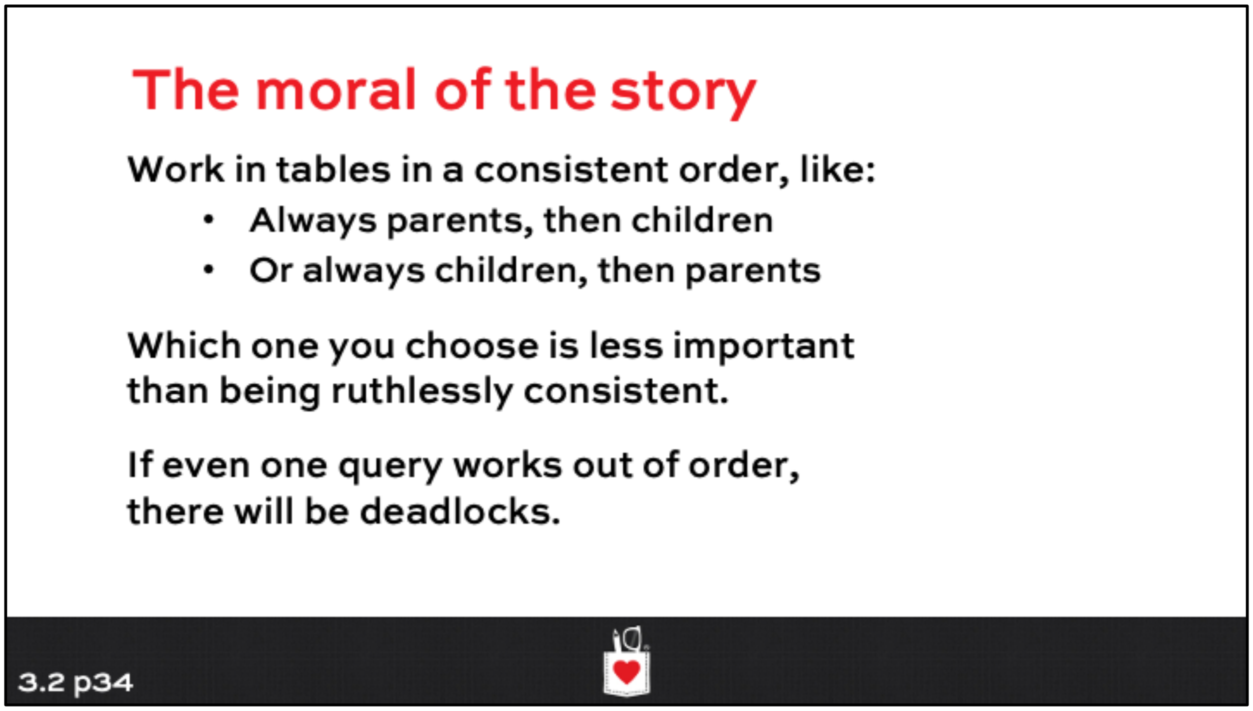


```
/* LEFT SIDE: */
BEGIN TRAN
UPDATE dbo.Lefty
  SET Numbers = Numbers + 1;
GO

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

```
Commands completed successfully.
```

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

```
(3 rows affected)
```



The moral of the story

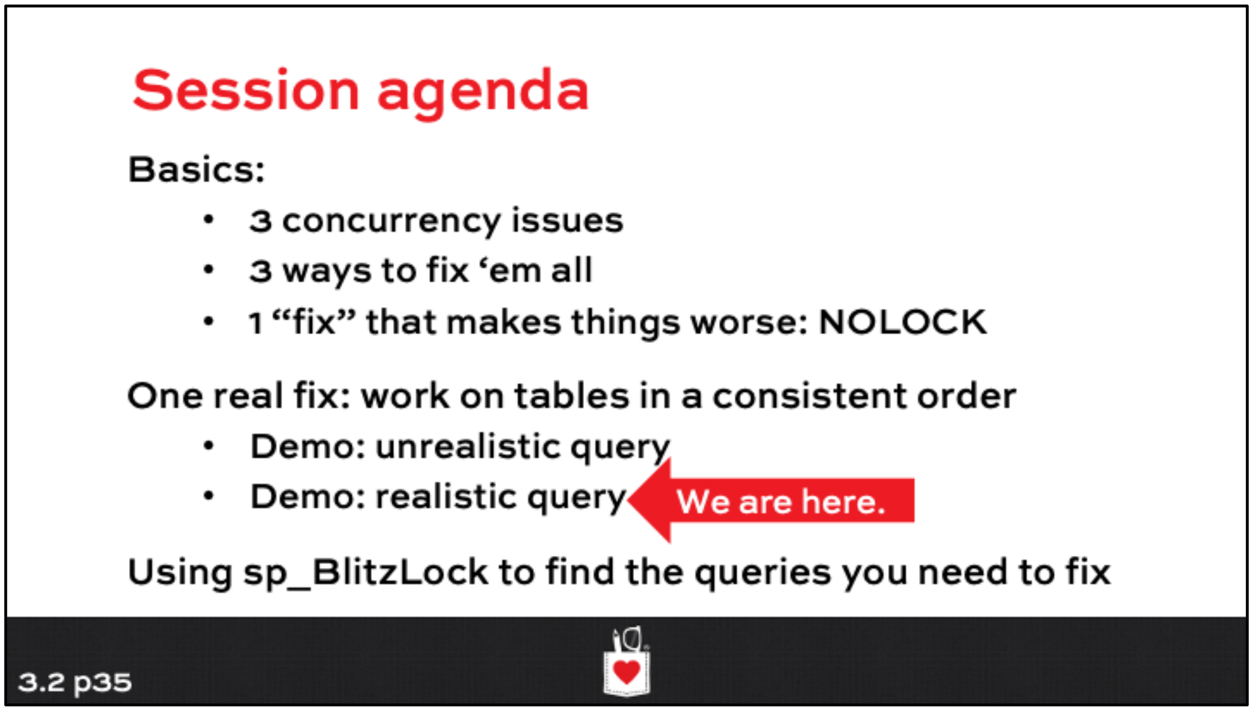
Work in tables in a consistent order, like:

- 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.

3.2 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

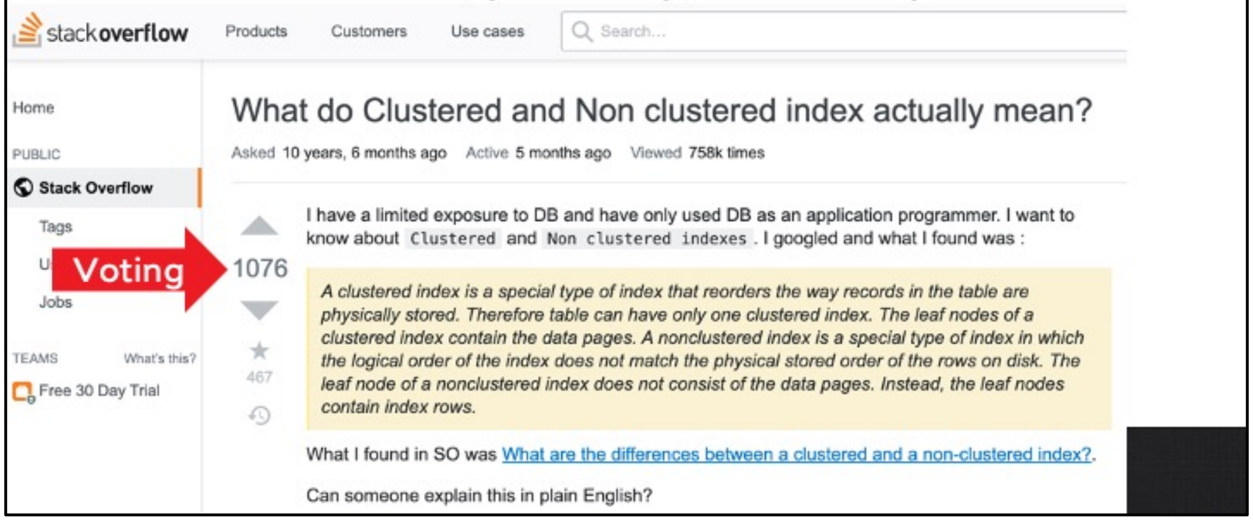
- Demo: unrealistic query
- Demo: realistic query

Using sp_BlitzLock to find the queries you need to fix

3.2 p35

Real-world scenario

At Stack Overflow, you can up/downvote questions.



What do Clustered and Non clustered index actually mean?

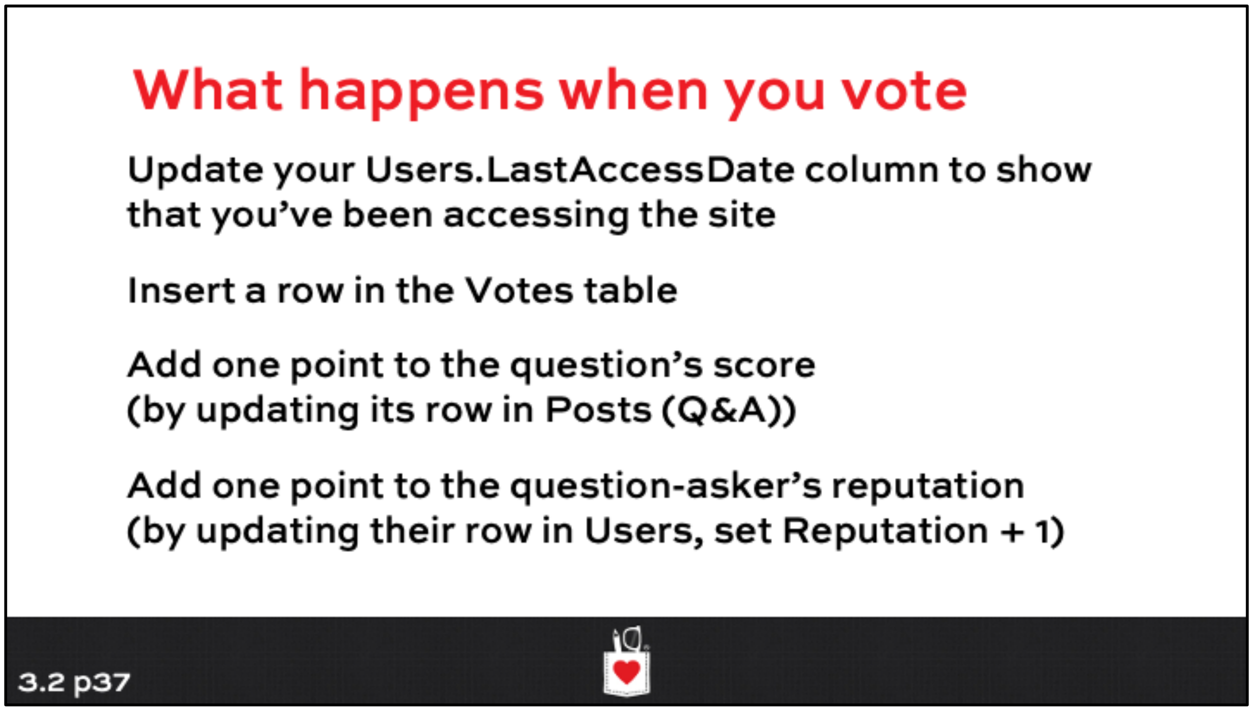
Asked 10 years, 6 months ago Active 5 months ago Viewed 758k times

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?



What happens when you vote

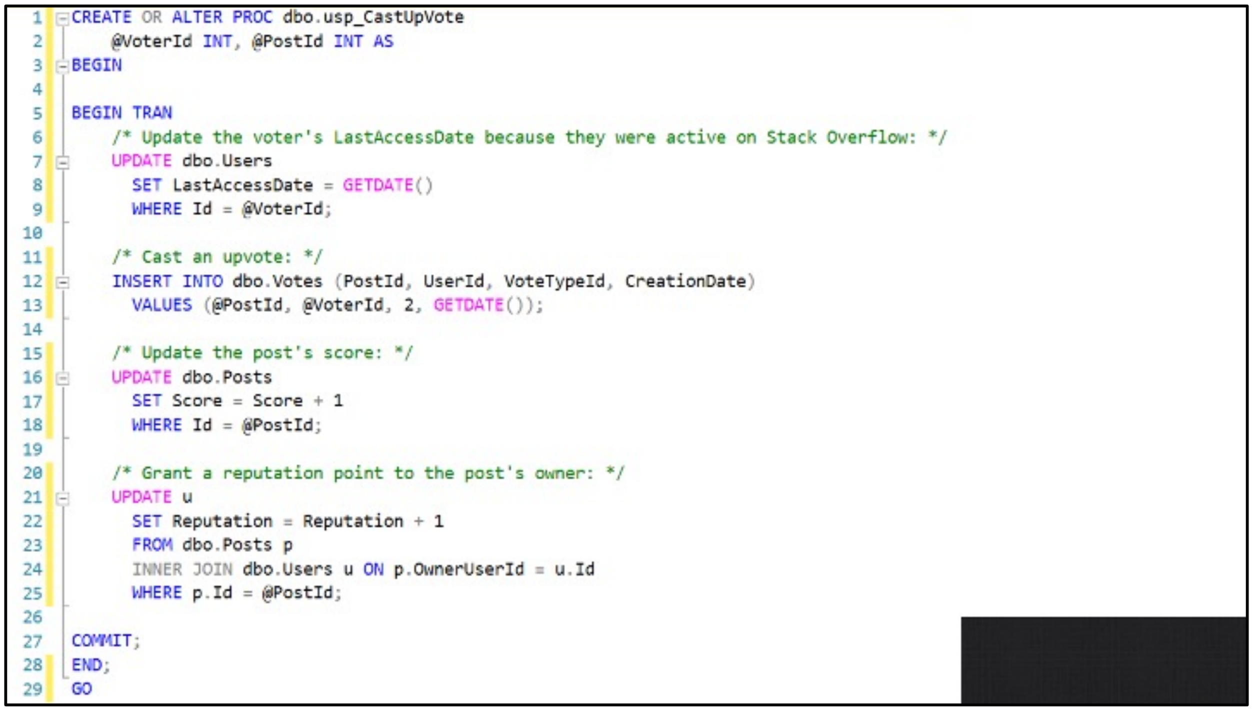
Update your Users.LastAccessDate column to show that you've been accessing the site

Insert a row in the Votes table

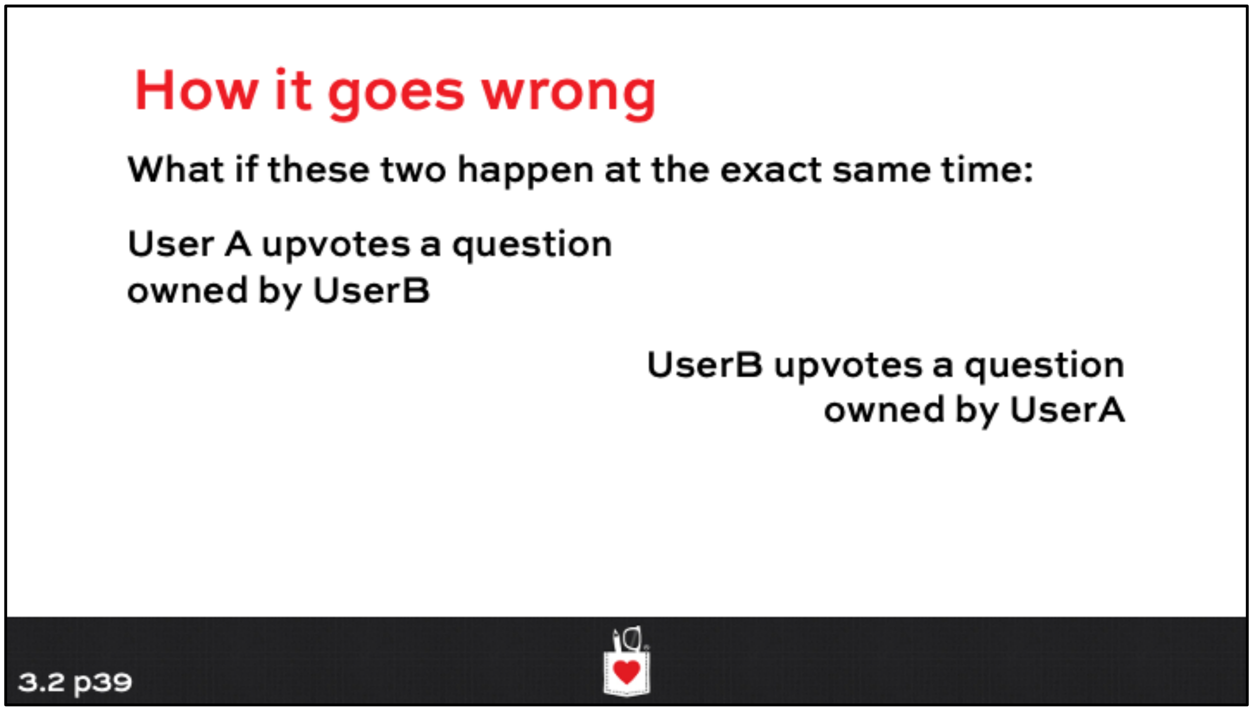
Add one point to the question's score (by updating its row in Posts (Q&A))

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

3.2 p37



```sql
CREATE OR ALTER PROC dbo.usp_CastUpVote
    @VoterId INT, @PostId INT AS
BEGIN

BEGIN TRAN
    /* Update the voter's LastAccessDate because they were active on Stack Overflow: */
    UPDATE dbo.Users
      SET LastAccessDate = GETDATE()
      WHERE Id = @VoterId;

    /* 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;

    /* Grant a reputation point to the post's owner: */
    UPDATE u
      SET Reputation = Reputation + 1
      FROM dbo.Posts p
      INNER JOIN dbo.Users u ON p.OwnerUserId = u.Id
      WHERE p.Id = @PostId;

COMMIT;
END;
GO
```



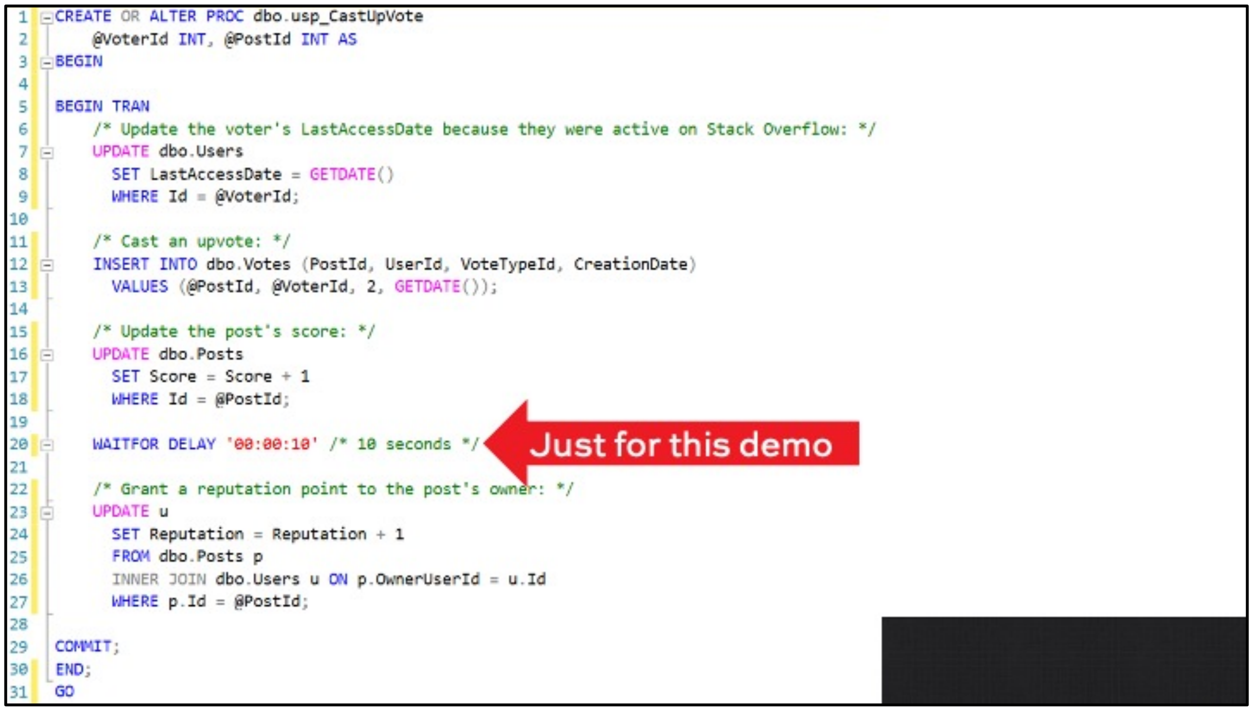
How it goes wrong

What if these two happen at the exact same time:

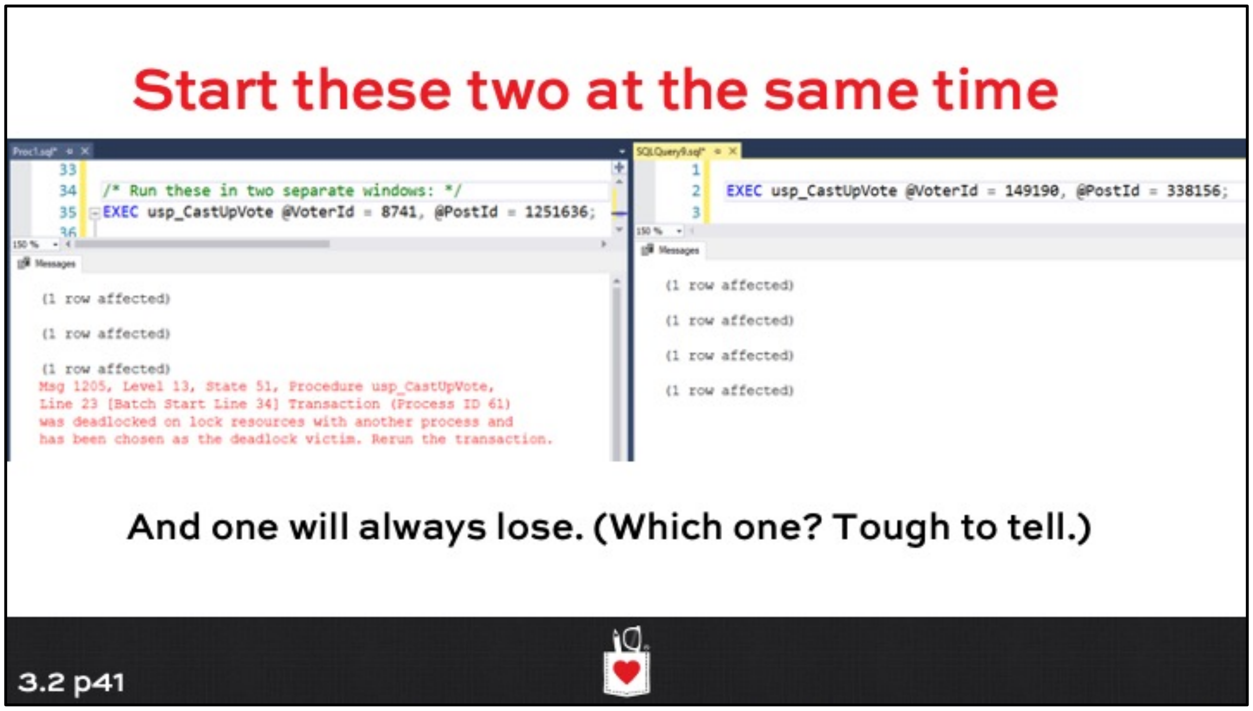
User A upvotes a question
owned by UserB

UserB upvotes a question
owned by UserA

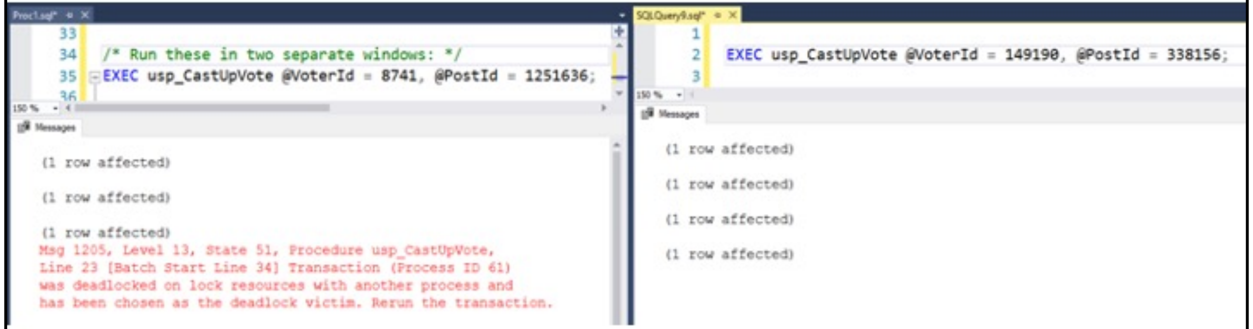
3.2 p39



```sql
CREATE OR ALTER PROC dbo.usp_CastUpVote
    @VoterId INT, @PostId INT AS
BEGIN

BEGIN TRAN
    /* Update the voter's LastAccessDate because they were active on Stack Overflow: */
    UPDATE dbo.Users
      SET LastAccessDate = GETDATE()
      WHERE Id = @VoterId;

    /* 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;

    WAITFOR DELAY '00:00:10' /* 10 seconds */

    /* Grant a reputation point to the post's owner: */
    UPDATE u
      SET Reputation = Reputation + 1
      FROM dbo.Posts p
      INNER JOIN dbo.Users u ON p.OwnerUserId = u.Id
      WHERE p.Id = @PostId;

COMMIT;
END;
GO
```



Start these two at the same time



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

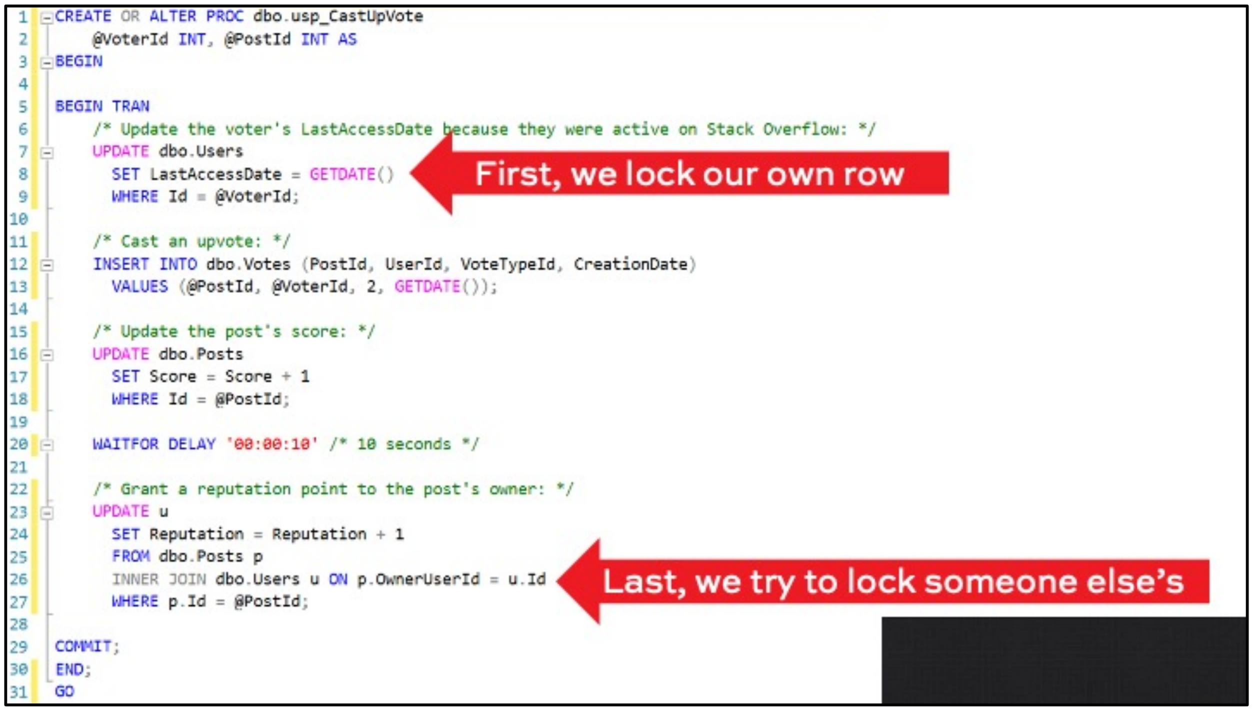
```
(1 row affected)

(1 row affected)

(1 row affected)
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.
```

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

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

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

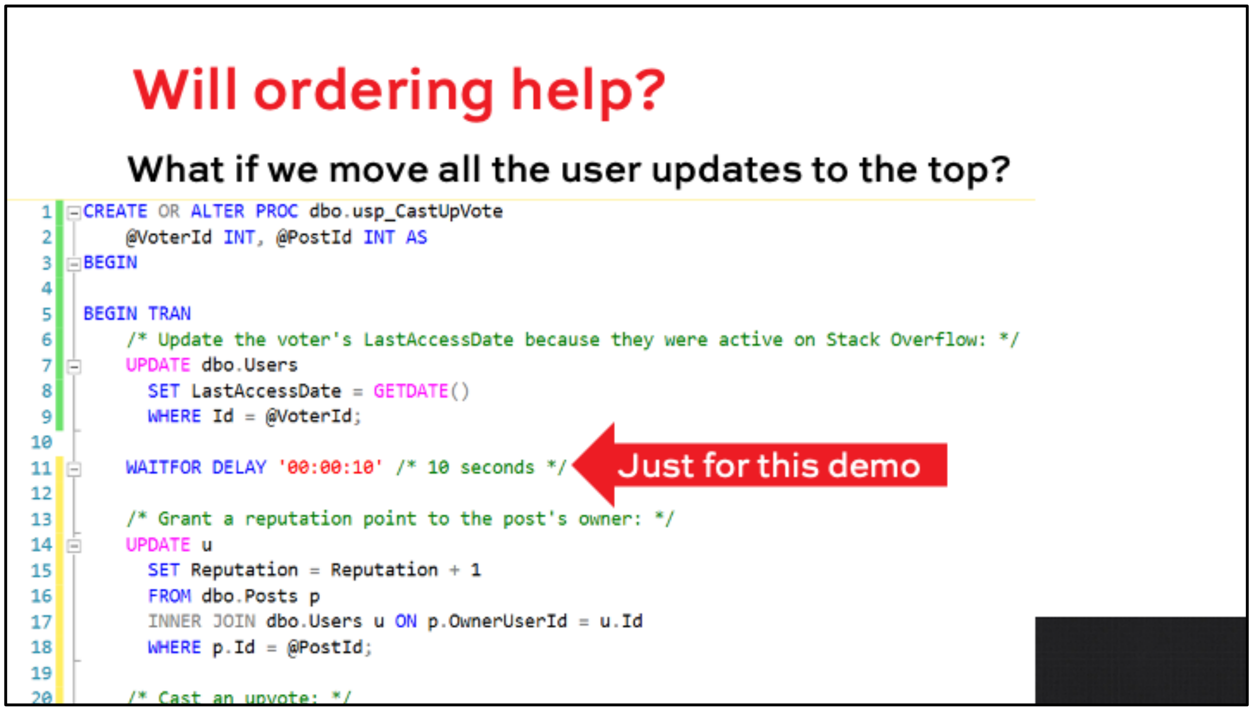
3.2 p41



```sql
CREATE OR ALTER PROC dbo.usp_CastUpVote
    @VoterId INT, @PostId INT AS
BEGIN

BEGIN TRAN
    /* Update the voter's LastAccessDate because they were active on Stack Overflow: */
    UPDATE dbo.Users
      SET LastAccessDate = GETDATE()
      WHERE Id = @VoterId;

    /* 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;

    WAITFOR DELAY '00:00:10' /* 10 seconds */

    /* Grant a reputation point to the post's owner: */
    UPDATE u
      SET Reputation = Reputation + 1
      FROM dbo.Posts p
      INNER JOIN dbo.Users u ON p.OwnerUserId = u.Id
      WHERE p.Id = @PostId;

COMMIT;
END;
GO
```

First, we lock our own row

Last, we try to lock someone else's

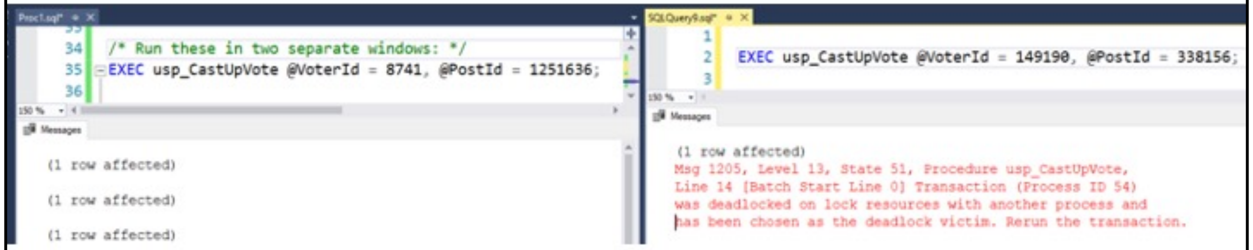


Will ordering help?

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

```
CREATE OR ALTER PROC dbo.usp_CastUpVote
    @VoterId INT, @PostId INT AS
BEGIN

BEGIN TRAN
    /* Update the voter's LastAccessDate because they were active on Stack Overflow: */
    UPDATE dbo.Users
      SET LastAccessDate = GETDATE()
      WHERE Id = @VoterId;

    WAITFOR DELAY '00:00:10' /* 10 seconds */

    /* Grant a reputation point to the post's owner: */
    UPDATE u
      SET Reputation = Reputation + 1
      FROM dbo.Posts p
      INNER JOIN dbo.Users u ON p.OwnerUserId = u.Id
      WHERE p.Id = @PostId;

    /* Cast an upvote: */
```

Still screwed.



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

```
(1 row affected)

(1 row affected)

(1 row affected)
```

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

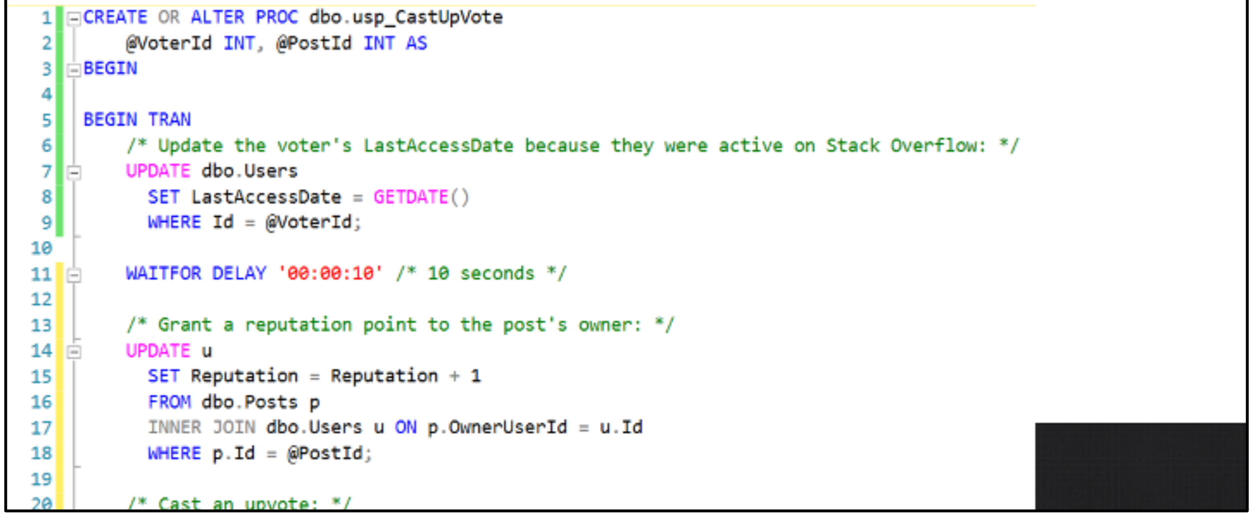
```
(1 row affected)
Msg 1205, Level 13, State 51, Procedure usp_CastUpVote,
Line 14 [Batch Start Line 0] Transaction (Process ID 54)
was deadlocked on lock resources with another process and
has been chosen as the deadlock victim. Rerun the transaction.
```

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

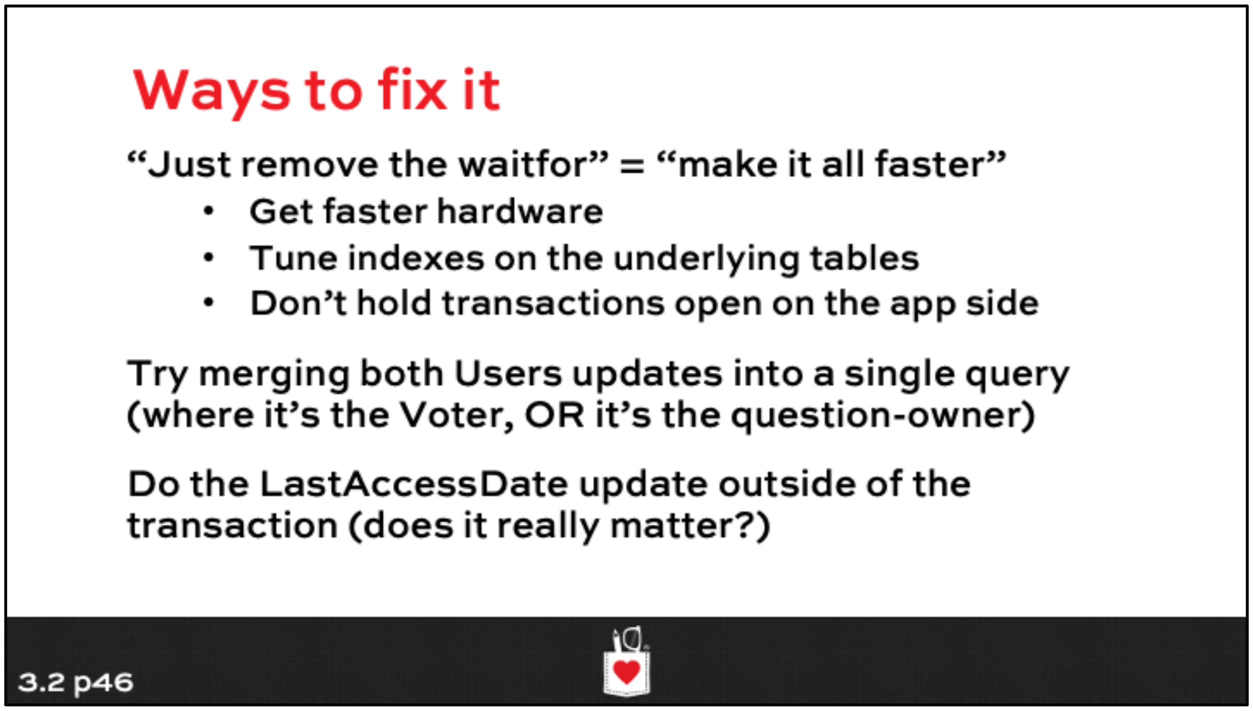
3.2 p44

Could we update both users at once?

How might we solve this problem?



```
CREATE OR ALTER PROC dbo.usp_CastUpVote
    @VoterId INT, @PostId INT AS
BEGIN

BEGIN TRAN
    /* Update the voter's LastAccessDate because they were active on Stack Overflow: */
    UPDATE dbo.Users
      SET LastAccessDate = GETDATE()
      WHERE Id = @VoterId;

    WAITFOR DELAY '00:00:10' /* 10 seconds */

    /* Grant a reputation point to the post's owner: */
    UPDATE u
      SET Reputation = Reputation + 1
      FROM dbo.Posts p
      INNER JOIN dbo.Users u ON p.OwnerUserId = u.Id
      WHERE p.Id = @PostId;

    /* Cast an upvote: */
```



Ways to fix it

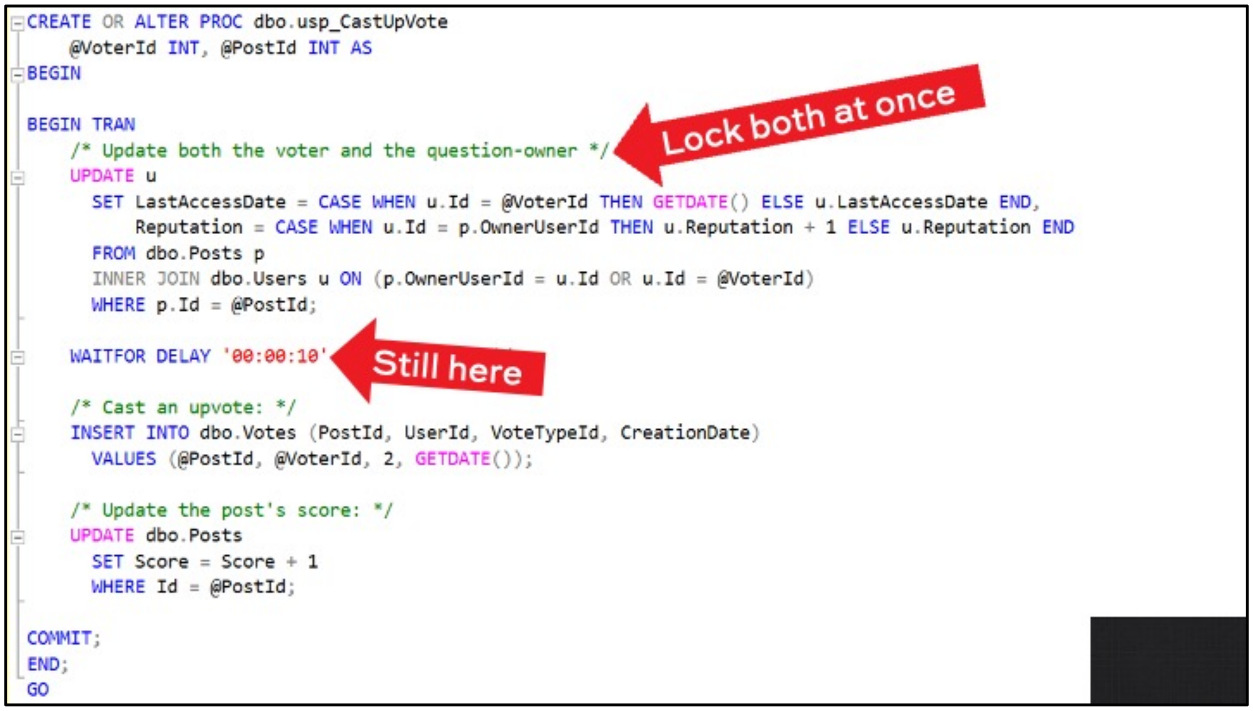
"Just remove the waitfor" = "make it all faster"

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

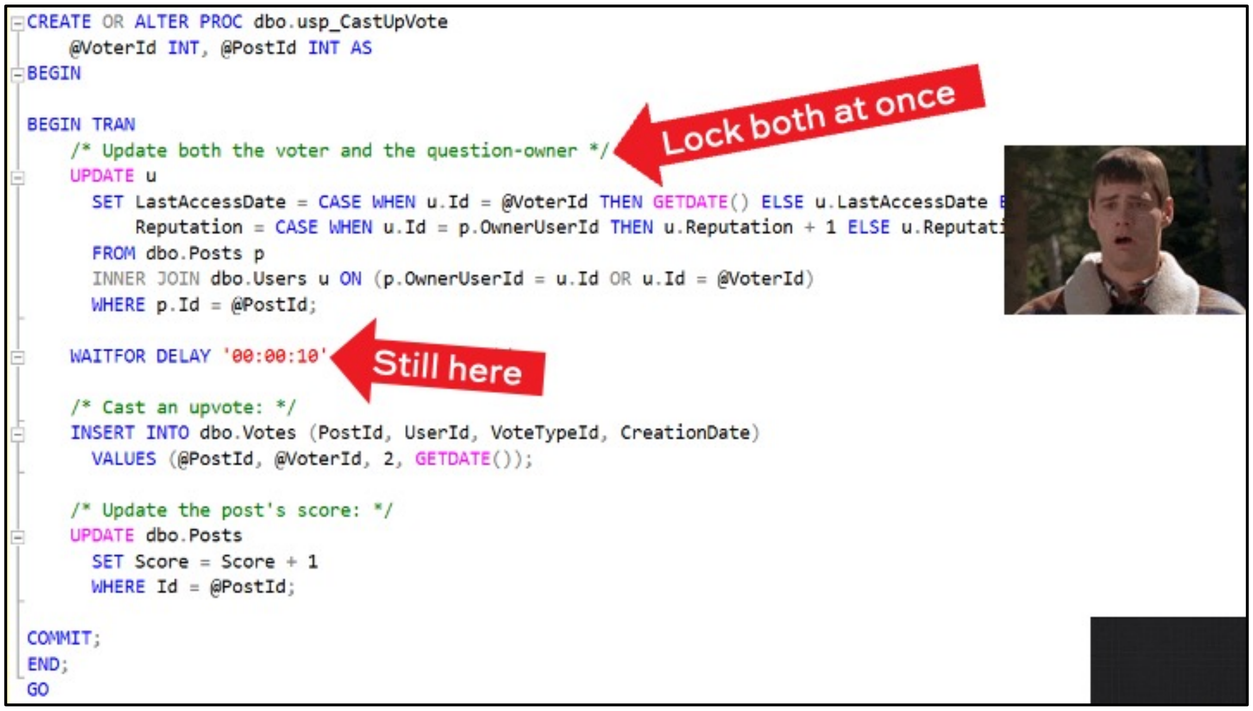
Try merging both Users updates into a single query (where it's the Voter, OR it's the question-owner)

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

3.2 p46

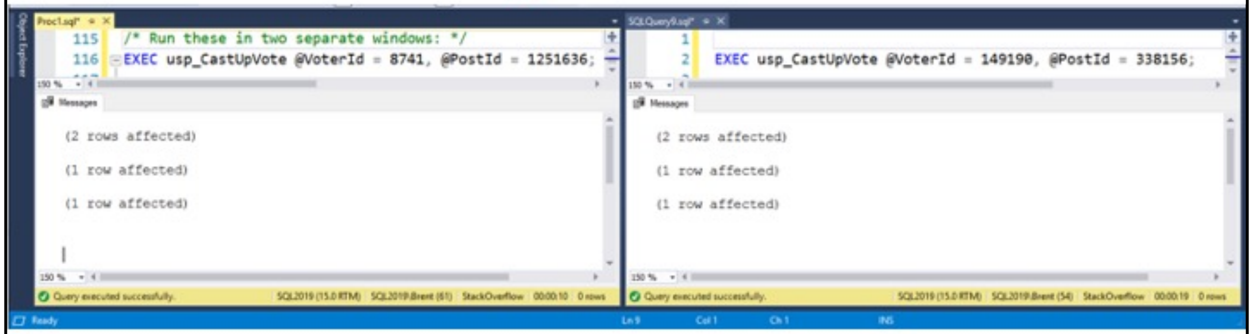


```
CREATE OR ALTER PROC dbo.usp_CastUpVote
    @VoterId INT, @PostId INT AS
BEGIN

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'

    /* 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
```



```
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 E
          Reputation = CASE WHEN u.Id = p.OwnerUserId THEN u.Reputation + 1 ELSE u.Reputati
      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
```

It works though! No deadlocks.



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

```
(2 rows affected)

(1 row affected)

(1 row affected)
```

Query executed successfully. | SQL2019 (15.0 RTM) | SQL2019\Brent (61) | StackOverflow | 00:00:10 | 0 rows

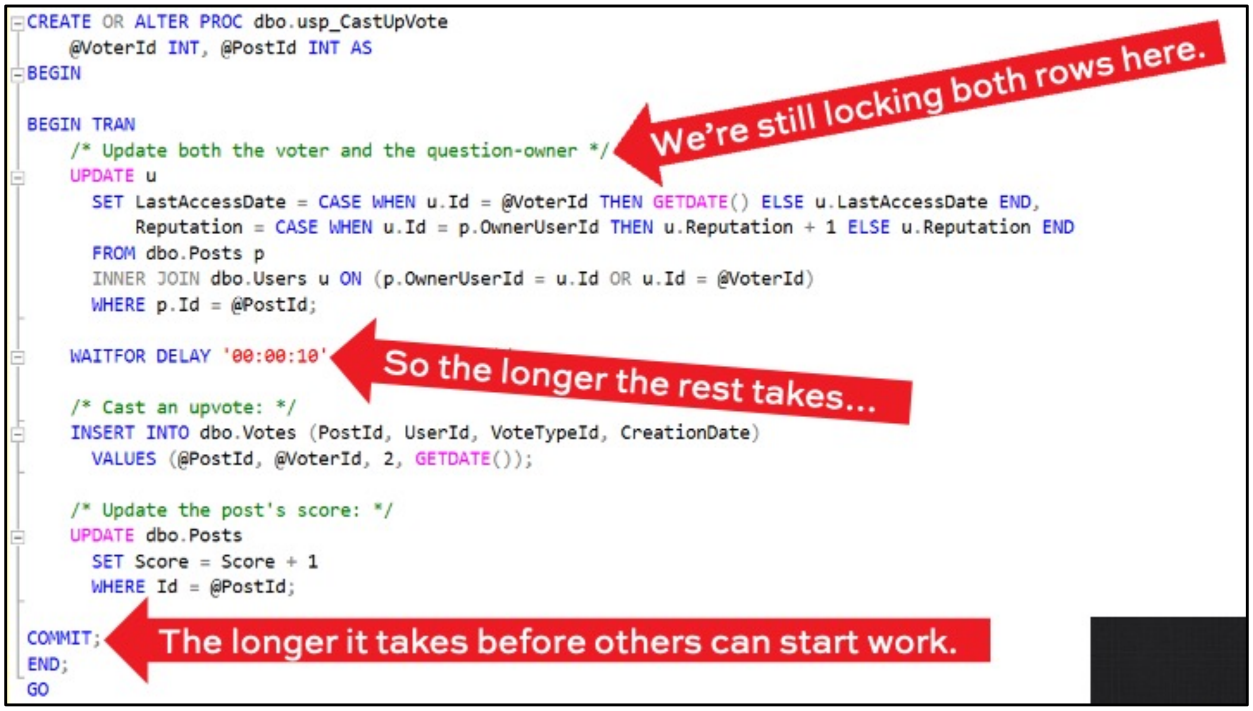
```
EXEC usp_CastUpVote @VoterId = 149190, @PostId = 338156;
```

```
(2 rows affected)

(1 row affected)

(1 row affected)
```

Query executed successfully. | SQL2019 (15.0 RTM) | SQL2019\Brent (54) | StackOverflow | 00:00:19 | 0 rows

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

3.2 p50

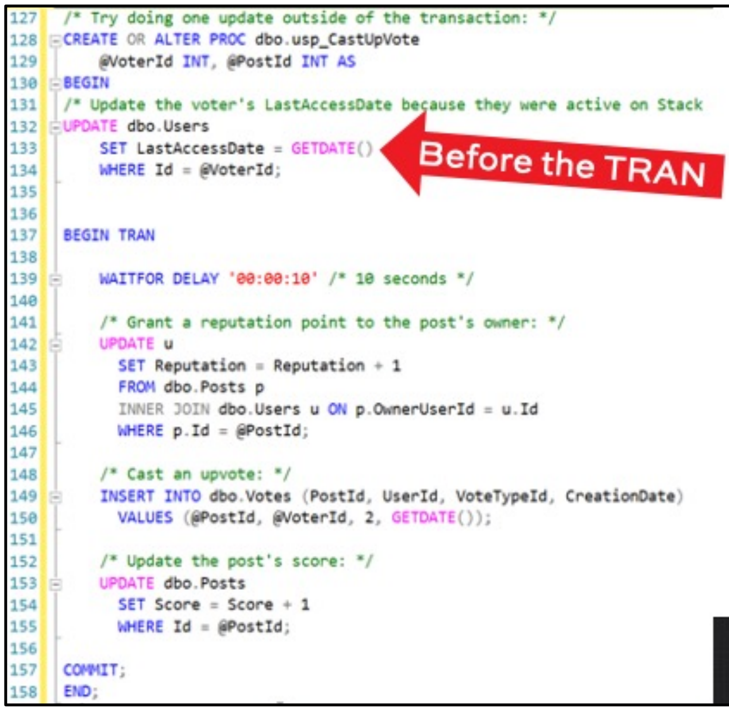


```
CREATE OR ALTER PROC dbo.usp_CastUpVote
    @VoterId INT, @PostId INT AS
BEGIN

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.

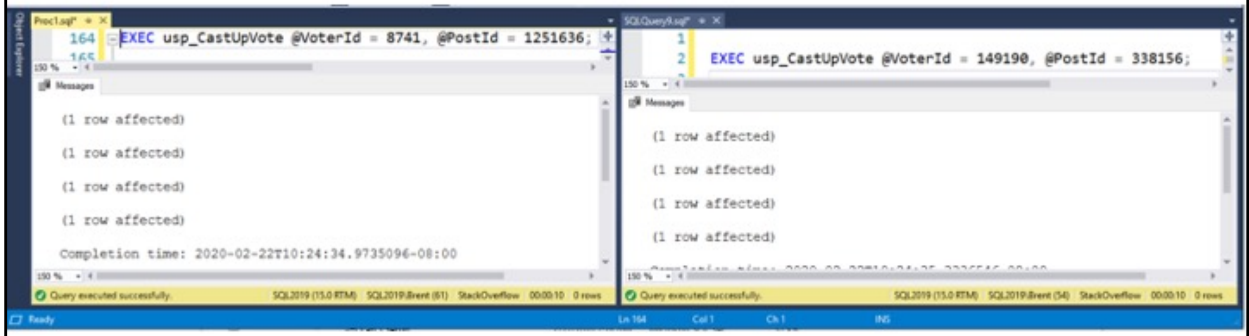
Another approach

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



```
/* Try doing one update outside of the transaction: */
CREATE OR ALTER PROC dbo.usp_CastUpVote
    @VoterId INT, @PostId INT AS
BEGIN
/* Update the voter's LastAccessDate because they were active on Stack
UPDATE dbo.Users
    SET LastAccessDate = GETDATE()
    WHERE Id = @VoterId;


BEGIN TRAN

    WAITFOR DELAY '00:00:10' /* 10 seconds */

    /* Grant a reputation point to the post's owner: */
    UPDATE u
      SET Reputation = Reputation + 1
      FROM dbo.Posts p
      INNER JOIN dbo.Users u ON p.OwnerUserId = u.Id
      WHERE p.Id = @PostId;

    /* 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;
```

Run 'em both at the same time...



Proc1.sql*

```
EXEC usp_CastUpVote @VoterId = 8741, @PostId = 1251636;
```

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2020-02-22T10:24:34.9735096-08:00
```

Query executed successfully. | SQL2019 (15.0 RTM) | SQL2019\Brent (61) | StackOverflow | 00:00:10 | 0 rows

SQLQuery9.sql*

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

Messages

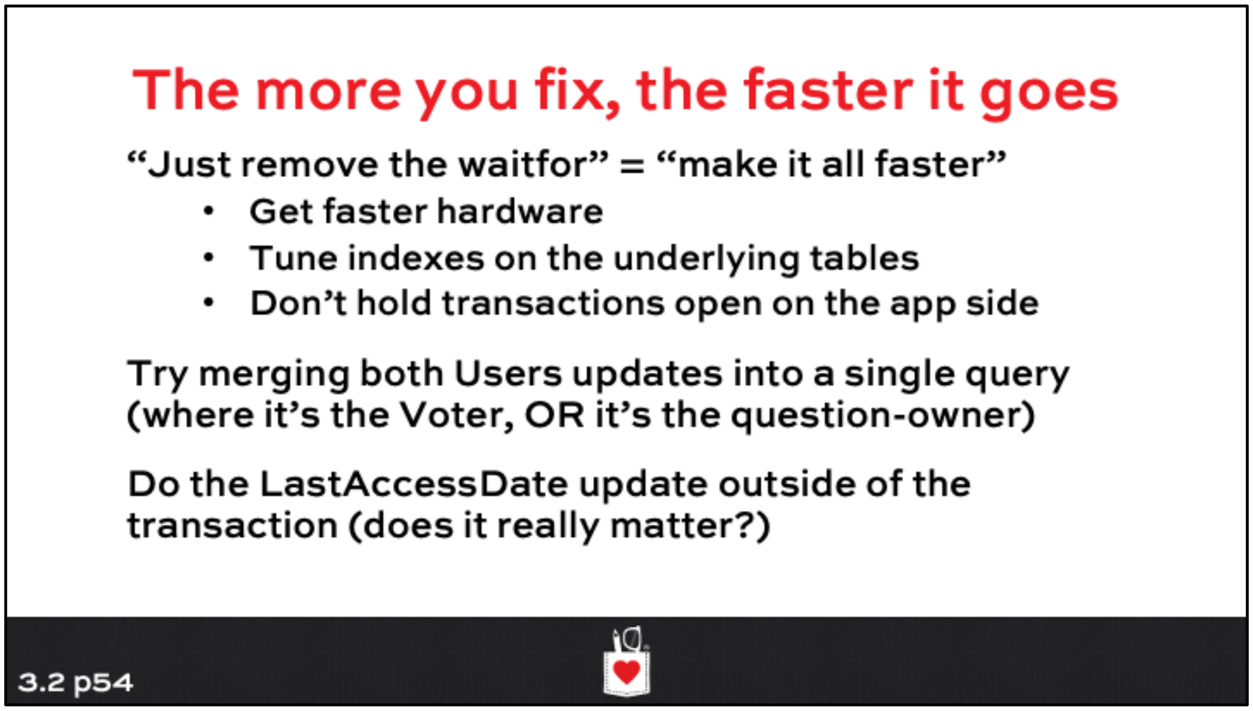
```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

Query executed successfully. | SQL2019 (15.0 RTM) | SQL2019\Brent (54) | StackOverflow | 00:00:10 | 0 rows

Ready | Ln 164 | Col 1 | Ch 1 | INS

No deadlocks, AND they both finish in 10 seconds!

3.2 p53



The more you fix, the faster it goes

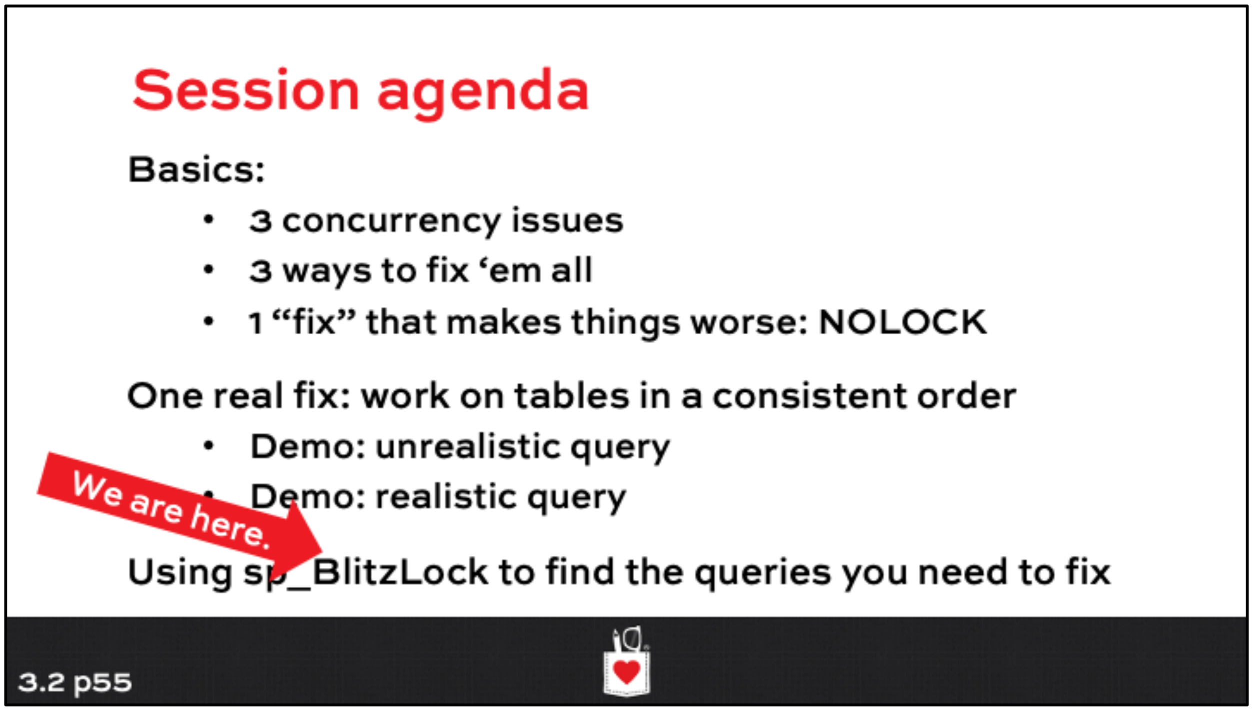
"Just remove the waitfor" = "make it all faster"

- 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?)

3.2 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

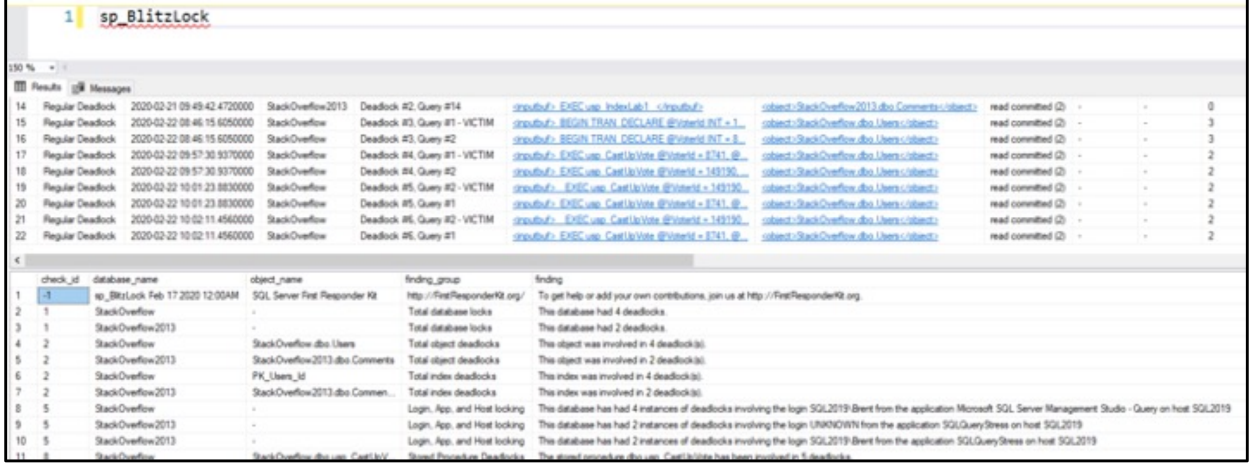
- Demo: unrealistic query
- Demo: realistic query

Using sp_BlitzLock to find the queries you need to fix

3.2 p55

sp_BlitzLock helps you spot ‘em.

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



```
sp_BlitzLock
```

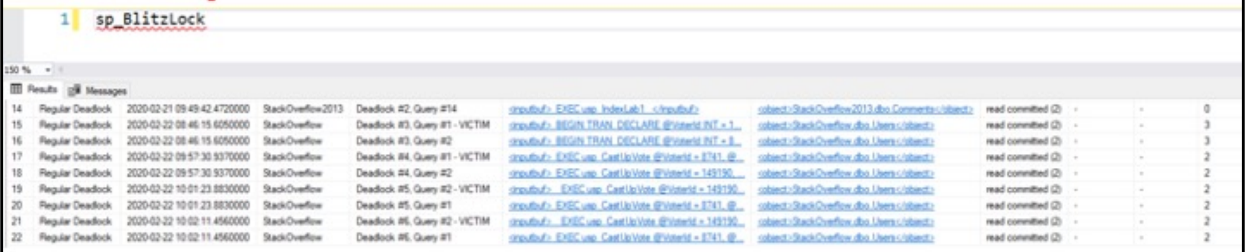
150 %

Results | Messages

14	Regular Deadlock	2020-02-21 09:49:42.4720000	StackOverflow2013	Deadlock #2, Query #14	<inputbuf> EXEC usp_IndexLab1 </inputbuf>	<object>StackOverflow2013.dbo.Comments</object>	read committed (2)	-	-	0
15	Regular Deadlock	2020-02-22 08:46:15.6050000	StackOverflow	Deadlock #3, Query #1 - VICTIM	<inputbuf> BEGIN TRAN DECLARE @VoterId INT = 1...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	3
16	Regular Deadlock	2020-02-22 08:46:15.6050000	StackOverflow	Deadlock #3, Query #2	<inputbuf> BEGIN TRAN DECLARE @VoterId INT = 8...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	3
17	Regular Deadlock	2020-02-22 09:57:30.9370000	StackOverflow	Deadlock #4, Query #1 - VICTIM	<inputbuf> EXEC usp_CastUpVote @VoterId = 8741, @...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	2
18	Regular Deadlock	2020-02-22 09:57:30.9370000	StackOverflow	Deadlock #4, Query #2	<inputbuf> EXEC usp_CastUpVote @VoterId = 149190, ...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	2
19	Regular Deadlock	2020-02-22 10:01:23.8830000	StackOverflow	Deadlock #5, Query #2 - VICTIM	<inputbuf> EXEC usp_CastUpVote @VoterId = 149190...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	2
20	Regular Deadlock	2020-02-22 10:01:23.8830000	StackOverflow	Deadlock #5, Query #1	<inputbuf> EXEC usp_CastUpVote @VoterId = 8741, @...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	2
21	Regular Deadlock	2020-02-22 10:02:11.4560000	StackOverflow	Deadlock #6, Query #2 - VICTIM	<inputbuf> EXEC usp_CastUpVote @VoterId = 149190...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	2
22	Regular Deadlock	2020-02-22 10:02:11.4560000	StackOverflow	Deadlock #6, Query #1	<inputbuf> EXEC usp_CastUpVote @VoterId = 8741, @...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	2

	check_id	database_name	object_name	finding_group	finding
1	-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 SQL2019\Brent 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 SQL2019\Brent from the application SQLQueryStress on host SQL2019
11	8	StackOverflow	StackOverflow.dbo.usp_CastUpV...	Stored Procedure Deadlocks	The stored procedure dbo.usp_CastUpVote has been involved in 5 deadlocks.

Top result set: list of deadlocks



```
sp_BlitzLock
```

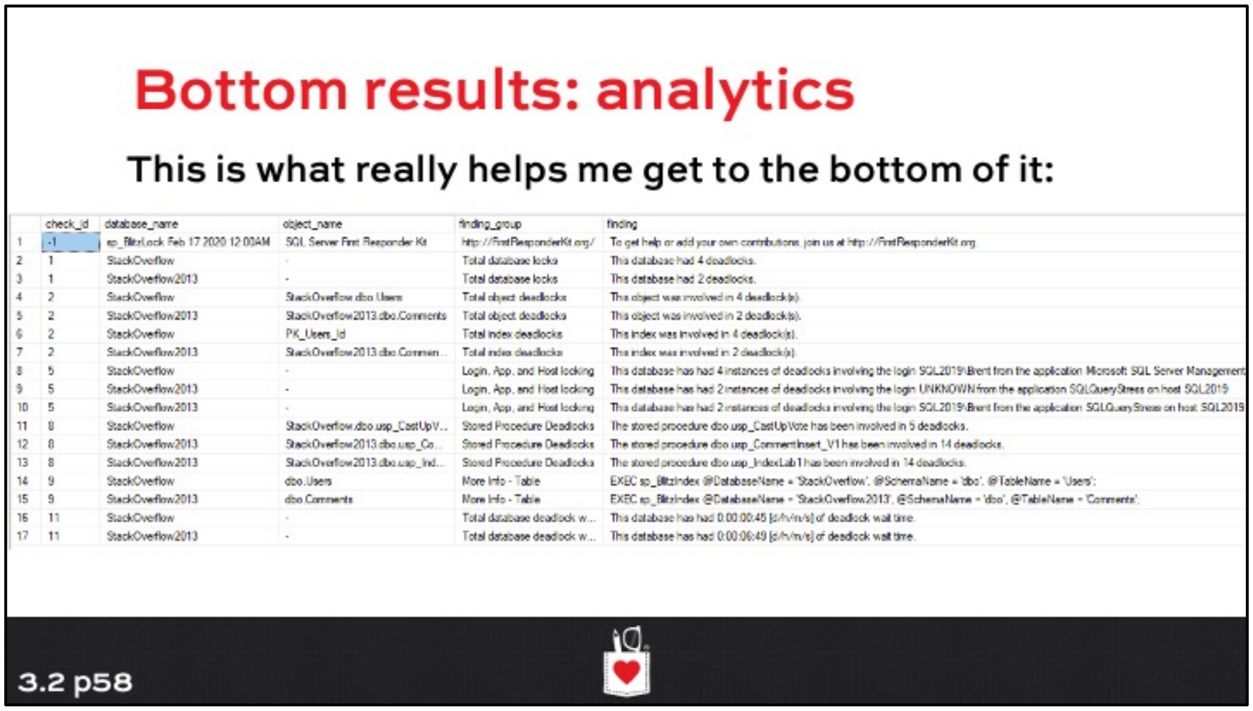
150 %

Results | Messages

14	Regular Deadlock	2020-02-21 09:49:42.4720000	StackOverflow2013	Deadlock #2, Query #14	<inputbuf> EXEC usp_IndexLab1 </inputbuf>	<object>StackOverflow2013.dbo.Comments</object>	read committed (2)	-	-	0
15	Regular Deadlock	2020-02-22 08:46:15.6050000	StackOverflow	Deadlock #3, Query #1 - VICTIM	<inputbuf> BEGIN TRAN DECLARE @VoterId INT = 1...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	3
16	Regular Deadlock	2020-02-22 08:46:15.6050000	StackOverflow	Deadlock #3, Query #2	<inputbuf> BEGIN TRAN DECLARE @VoterId INT = 8...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	3
17	Regular Deadlock	2020-02-22 09:57:30.9370000	StackOverflow	Deadlock #4, Query #1 - VICTIM	<inputbuf> EXEC usp_CastUpVote @VoterId = 8741, @...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	2
18	Regular Deadlock	2020-02-22 09:57:30.9370000	StackOverflow	Deadlock #4, Query #2	<inputbuf> EXEC usp_CastUpVote @VoterId = 149190,...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	2
19	Regular Deadlock	2020-02-22 10:01:23.8830000	StackOverflow	Deadlock #5, Query #2 - VICTIM	<inputbuf> EXEC usp_CastUpVote @VoterId = 149190,...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	2
20	Regular Deadlock	2020-02-22 10:01:23.8830000	StackOverflow	Deadlock #5, Query #1	<inputbuf> EXEC usp_CastUpVote @VoterId = 8741, @...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	2
21	Regular Deadlock	2020-02-22 10:02:11.4560000	StackOverflow	Deadlock #6, Query #2 - VICTIM	<inputbuf> EXEC usp_CastUpVote @VoterId = 149190,...	<object>StackOverflow.dbo.Users</object>	read committed (2)	-	-	2
22	Regular Deadlock	2020-02-22 10:02:11.4560000	StackOverflow	Deadlock #6, Query #1	<inputbuf> EXEC usp_CastUpVote @VoterId = 8741, @...	<object>StackOverflow.dbo.Users</object>	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.

3.2 p57

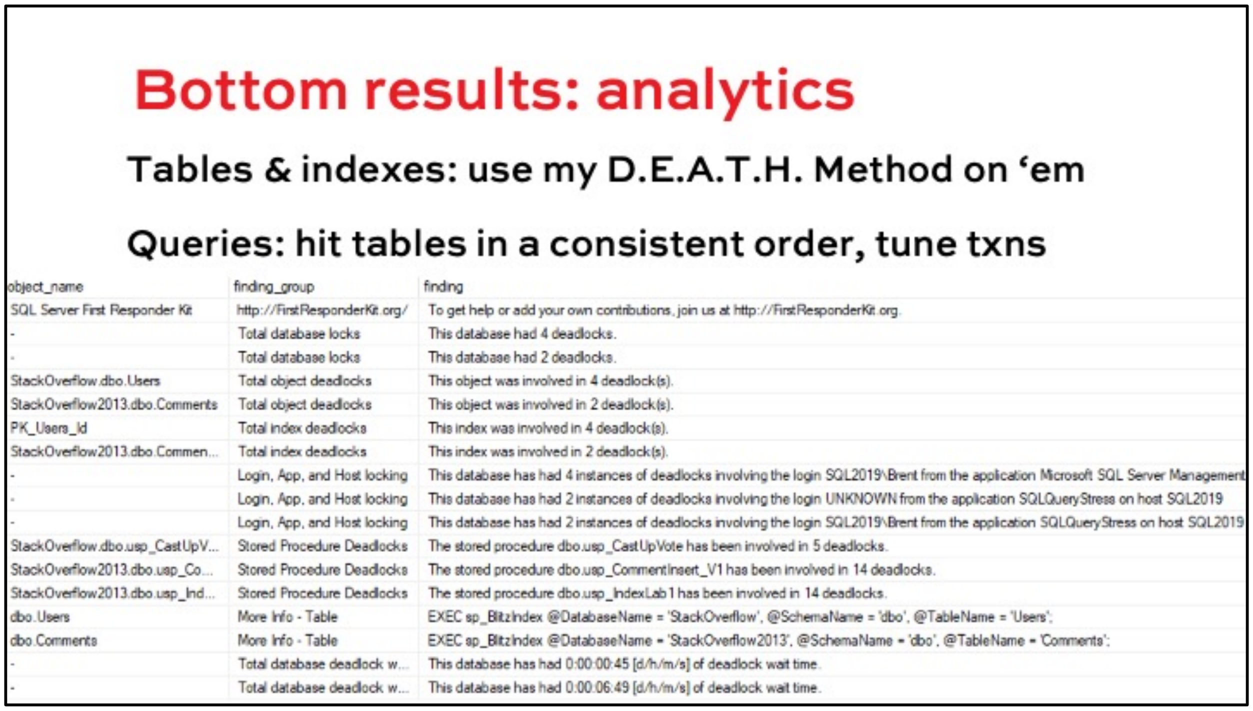


Bottom results: analytics

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

	check_id	database_name	object_name	finding_group	finding
1	-1	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 SQL2019\Brent 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 SQL2019\Brent from the application SQLQueryStress on host SQL2019
11	8	StackOverflow	StackOverflow.dbo.usp_CastUpV...	Stored Procedure Deadlocks	The stored procedure dbo.usp_CastUpVote has been involved in 5 deadlocks.
12	8	StackOverflow2013	StackOverflow2013.dbo.usp_Co...	Stored Procedure Deadlocks	The stored procedure dbo.usp_CommentInsert_V1 has been involved in 14 deadlocks.
13	8	StackOverflow2013	StackOverflow2013.dbo.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 w...	This database has had 0:00:00:45 [d/h/m/s] of deadlock wait time.
17	11	StackOverflow2013	-	Total database deadlock w...	This database has had 0:00:06:49 [d/h/m/s] of deadlock wait time.

3.2 p58

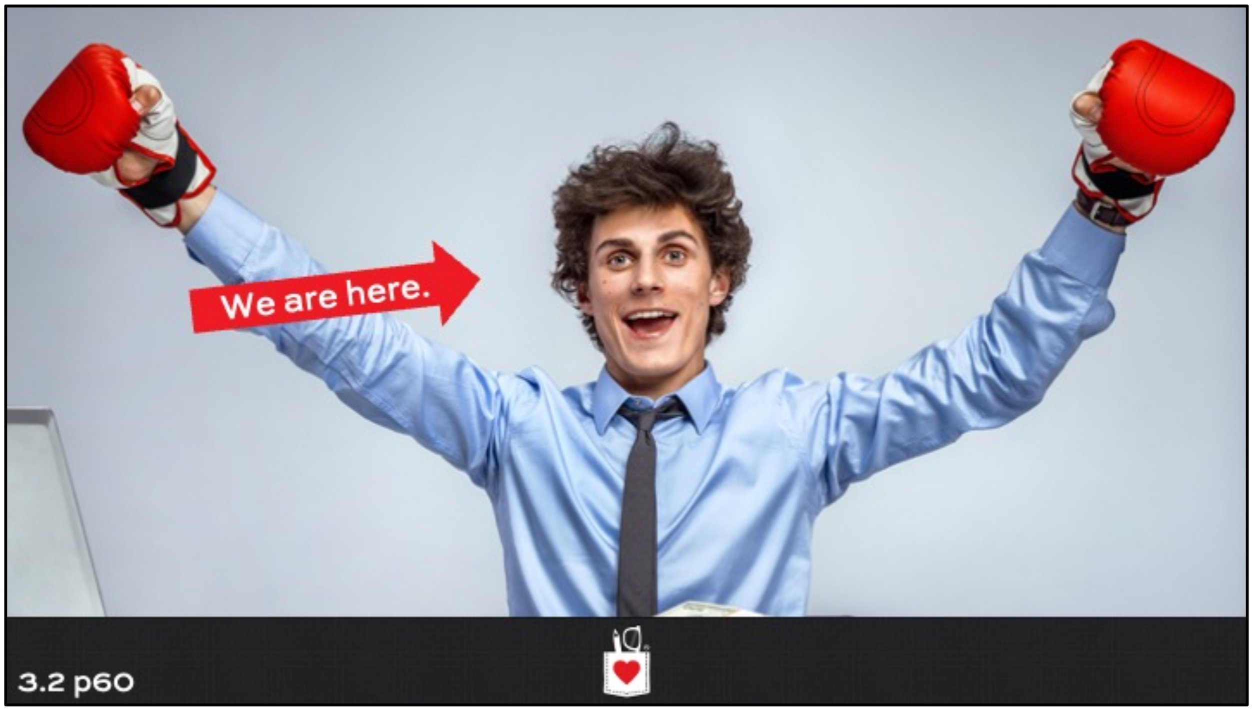


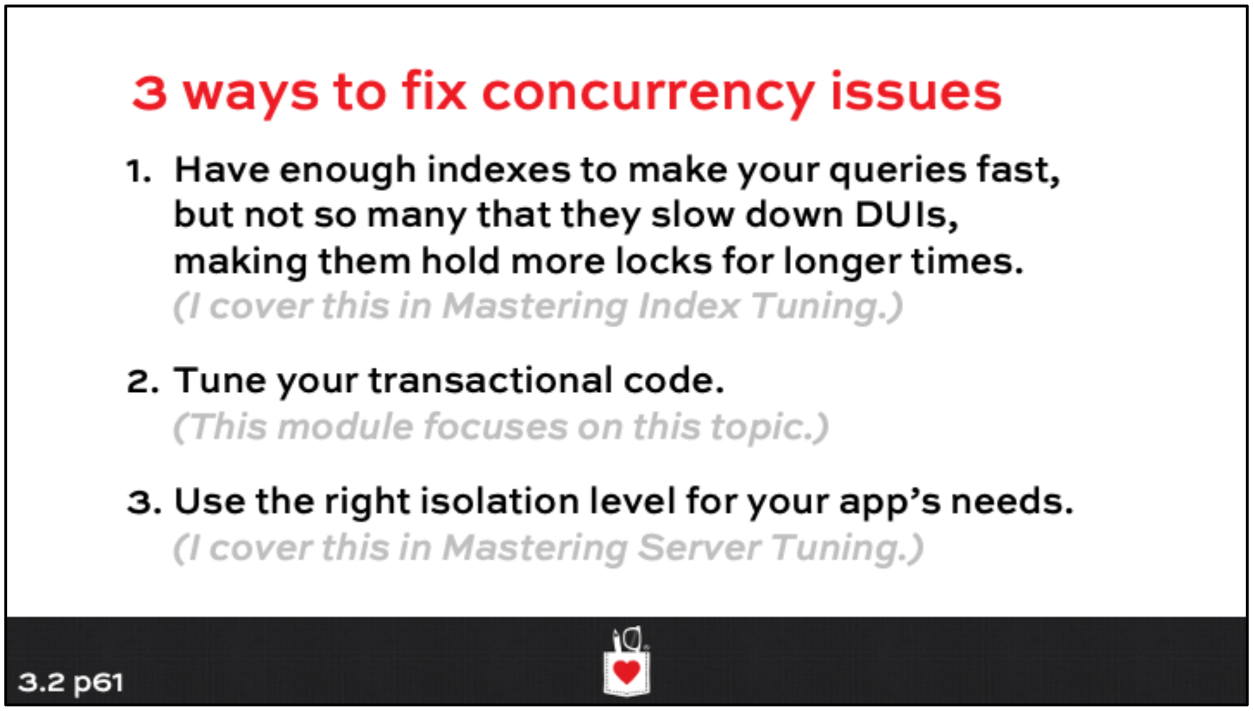
Bottom results: analytics

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.


We are here.
3.2 p60



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.)

3.2 p61