Faster Transactions:

Query Tuning for Data Manipulation

Jeff Iannucci

Who in the world is Jeff Iannucci?

I live in Arizona, and I sell used cars!

Senior Database Administrator at DriveTime

SQL Server data professional since 1998

desertdba.com

jeff@desertdba.com

@desertdba

What is in this session?

Performance tuning for…

INSERT

UPDATE

DELETE

…especially for large amounts of data

What are the goals of this session?

Show WHAT commands help performance

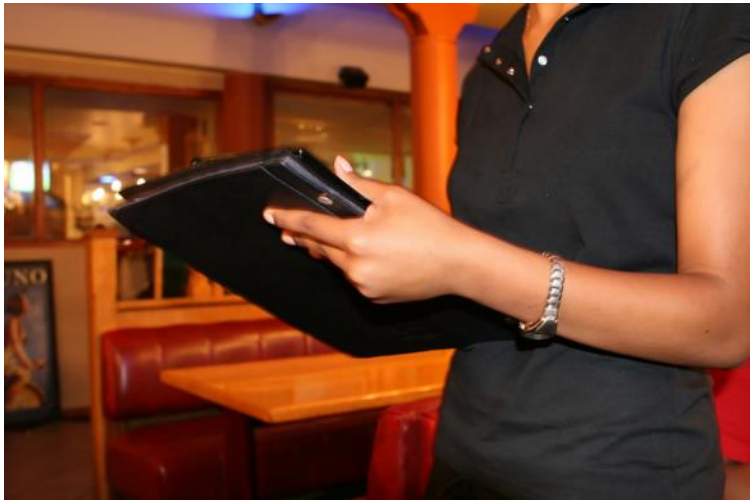
Discuss WHEN to use these commands

Explain WHY the performance improves

But first…let’s talk about food!

The waiter writes down the order…

…and orders go in the ticket queue



The chef assembles the ingredients…

…gets any missing ingredients from pantry

The chef holds items while working…

…until your order is ready!

“Le SQL Server”

Transaction orders at “Le SQL Server”

Log Buffer Cache

Transaction Log

Buffer Cache Pages

Pages on Disk

Lock

So…how can transactions go faster?

Read/write less data pages

Create less locks

Create less transaction log entries

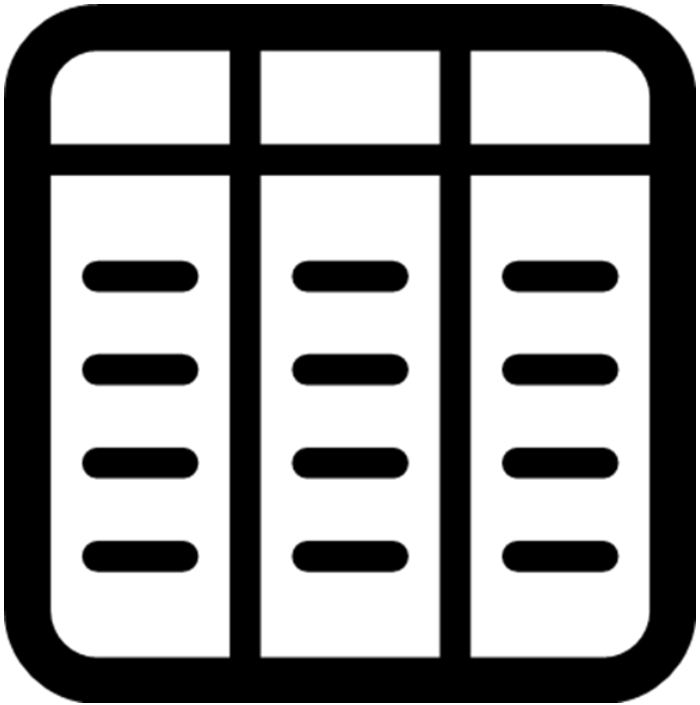
What's on the menu today?

Appetizers: Alter the target table

Entrees: Manipulate the transaction

Desserts: Modify the database settings

Appetizers: altering the target table



Let’s start with…

An INSERT

…of sorted data

…into an empty table

…that has no indexes

…(also known as a heap.)

Clustered Index instead of a Heap

Ordered data INSERTs faster than random data

…if the Clustered Index (CI) matches sort order

This works for INSERTs only

Clustered Index instead of a Heap

```sql
-- Create a table a clustered index
CREATE TABLE SomeTable (
    ID int NOT NULL PRIMARY KEY CLUSTERED
    , Col1 varchar(100)
    , Col2 datetime
    ) ON [PRIMARY];

-- ...then an INSERT
INSERT SomeTable (ID, Col1, Col2)
SELECT ID, Col1, Col2
FROM SomeOtherTable;
```

Clustered Index instead of a Heap



Should every table have a clustered index?

- Unsorted INSERTs are slower with a CI
- Only faster when CI matches sort order
- …but, temporary tables can have a CI

Now let’s try a bit of…

An UPDATE

…to every record of a table

…that has a clustered index

…and non-clustered indexes.

Disabling indexes

Reduces data pages manipulated

Disable Non-Clustered Indexes (NCIs) only

Disable only NCIs with affected columns

This works for INSERTs, UPDATEs, & DELETEs

Disabling indexes

```sql
-- Disable an index
ALTER INDEX IX_SomeTable_ColX
 ON SomeSchema.SomeTable DISABLE;

UPDATE SomeTable
SET ColX = 1
WHERE ColX = 0;

-- Enable an index
ALTER INDEX IX_SomeTable_ColX
 ON SomeSchema.SomeTable REBUILD;
```

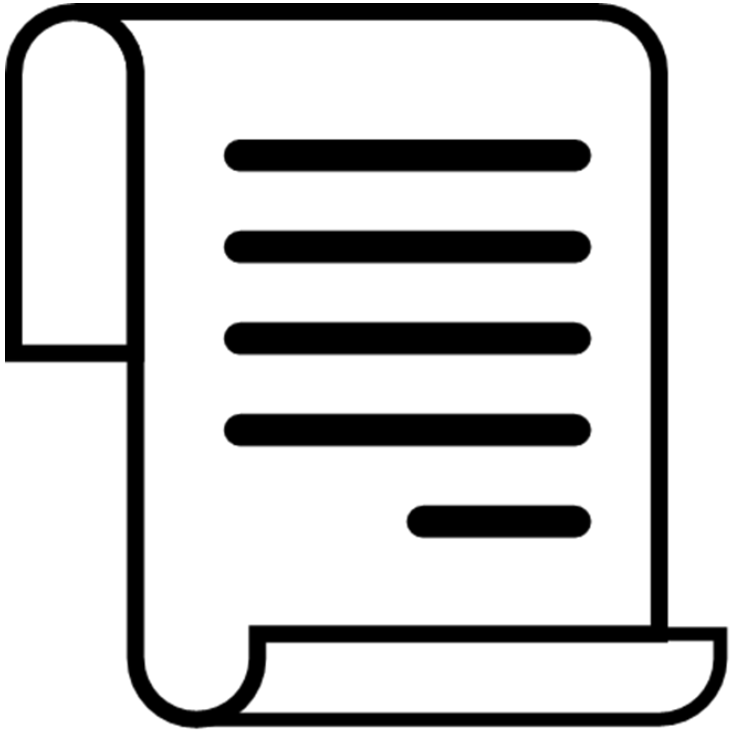
Disabling indexes



Yes…but…

- REBUILDs take additional time
- NCIs are not used until they are rebuilt
- Clustered Index scans (temporarily)

Entrees: manipulating the transaction



Today let’s have…

A DELETE

…of a majority of data

…in a table with lots of records

…with lots of user connections

…so don’t slow their orders.

Batch using WHILE with TOP

Separates one transaction to many smaller ones

Minimal disruption to concurrent users

Allows other processes to go faster (less slow)

This works for INSERTs, UPDATEs, & DELETEs

Batch using WHILE with TOP

```sql
-- Batch using WHILE with TOP
DECLARE
    @BatchIdMin int = 1
    , @BatchIdMax int
    , @RowCount int = 1

WHILE (@RowCount > 0) BEGIN

    SELECT TOP (50000) @BatchIdMax = Id
    FROM SomeTable
    WHERE Id > @BatchIdMin
    ORDER BY Id;

    DELETE
    FROM SomeTable
    WHERE Id > @BatchIdMin
     AND Id <= @BatchIdMax;

    SET @RowCount = @@ROWCOUNT;

    SET @BatchIdMin = @BatchIdMax

    END;
```

Batch using WHILE with TOP



What could go wrong?

- Could result in a slower query for you
- Will result in partial “transaction” if stopped
- Lock escalation occurs at 5000 locks

For RBAR dieters, try…

An UPDATE

…of much of the data

…in a table with lots of records

…but each row is updated

…in a separate transaction.

Explicit transaction

Reduces the pain of Row By Agonizing Row

Consolidates many transaction into one

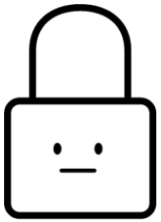
Use when concurrency is not a factor

This works for INSERTs, UPDATEs, & DELETEs

Explicit transaction

```sql
DECLARE @Counter int = 1

-- Explicit start
BEGIN TRAN

WHILE @Counter <= 1000000 BEGIN
    UPDATE SomeTable
    SET ColX = 1
    WHERE ColPK = @Counter;

    SET @Counter += 1
    END

-- Explicit end
COMMIT
```

Explicit transaction


The fine print:

- Locked resources during transaction
- Concurrent users wait for resources
- A bit like TABLOCK

What you really want is…

An INSERT

…of sorted or unsorted data

…into an empty table

…and make it fast as you can!

Minimally logged INSERT

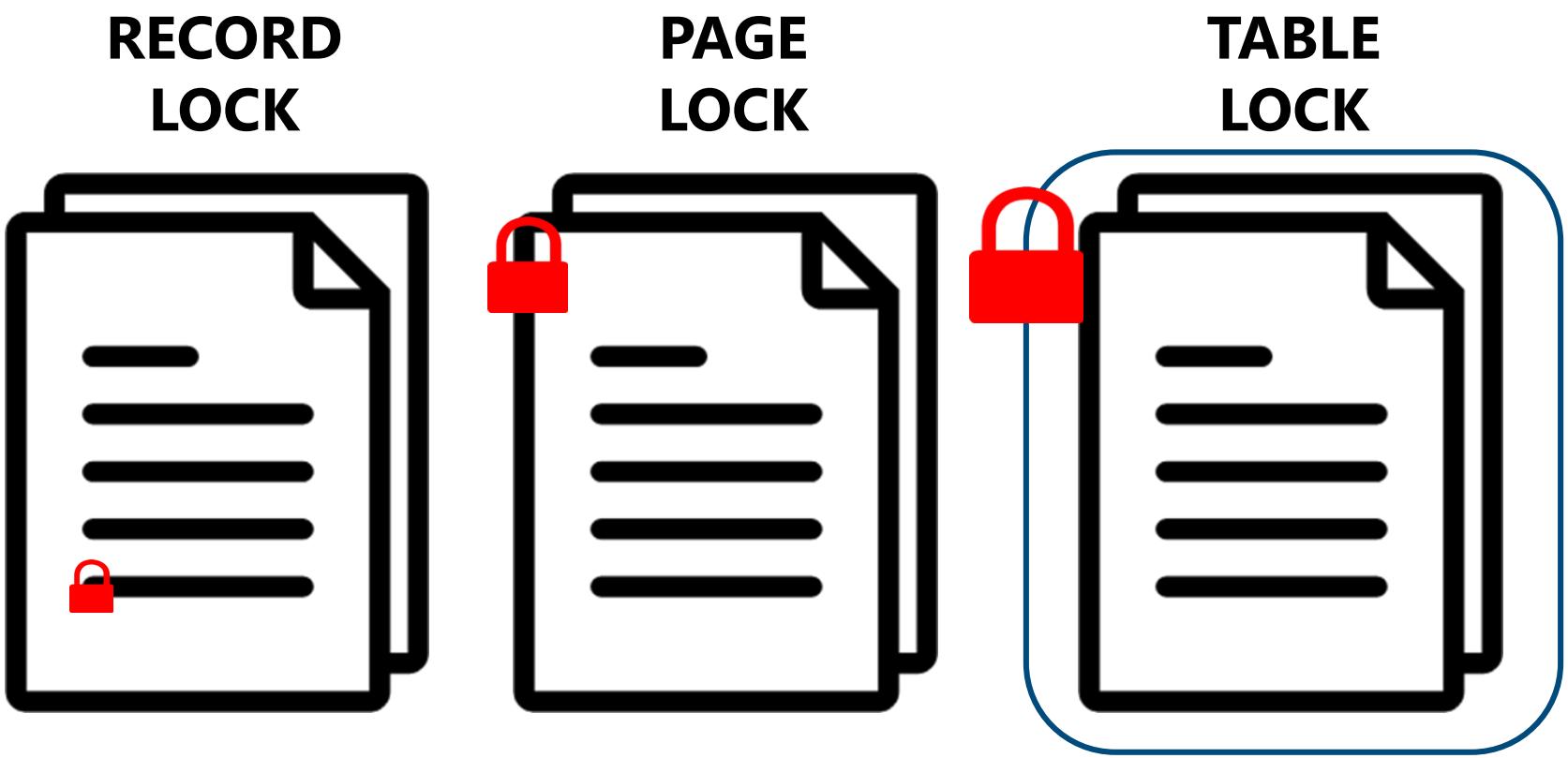
Less records in the Transaction Log

Logging of page allocations, not record inserts

Use when no concurrent usage

…because this locks the entire INSERT table

What do you mean by “table lock”?



What do you mean by “page allocation”?

RECORD logging (default)

Current LSN	Operation	Context
00000276:00008c68:00e7	LOP_INSERT_ROWS	LCX_HEAP
00000276:00008c68:00e8	LOP_INSERT_ROWS	LCX_HEAP
00000276:00008c68:00e9	LOP_INSERT_ROWS	LCX_HEAP
00000276:00008c68:00ea	LOP_INSERT_ROWS	LCX_HEAP
00000276:00008c68:00eb	LOP_INSERT_ROWS	LCX_HEAP
00000276:00008c68:00ec	LOP_INSERT_ROWS	LCX_HEAP
00000276:00008c68:00ed	LOP_INSERT_ROWS	LCX_HEAP
00000276:00008c68:00ee	LOP_INSERT_ROWS	LCX_HEAP
00000276:00008c68:00ef	LOP_INSERT_ROWS	LCX_HEAP

PAGE allocation logging

Current LSN	Operation	Context
00000276:00008ab8:004d	LOP_MODIFY_ROW	LCX_PFS
00000276:00008ab8:004e	LOP_SET_BITS	LCX_IAM
00000276:00008ab8:004f	LOP_SET_BITS	LCX_GAM

Some minimally-logged transaction rules

Database NOT in FULL recovery model

Table is not replicated

Table is not memory-optimized

Table has no indexes (is a heap), or…

…if the table has indexes then it must be empty

May require Trace Flag 610

https://docs.microsoft.com/en-us/sql/relational-databases/import-export/prerequisites-for-minimal-logging-in-bulk-import?view=sql-server-ver15

Minimally logged INSERT

```sql
-- Minimal Logging with TABLOCK
INSERT SomeTable
 WITH (TABLOCK) (Col1, Col2, Col3)
SELECT Col1, Col2, Col3
FROM SomeOtherTable;
```

Minimally logged INSERT



The bad news:

- Concurrent users wait for resources
- Only uncommitted (“dirty”) reads
- Only works for INSERT, although…

Next you might enjoy…

A DELETE

…of a majority of data

…in a table with lots of records

…and make it fast

…like minimal logging!

Minimally logged DELETE

Wait…what?!?

Utilize minimally logged INSERTs - Twice!

Let’s lock TWO tables

(Not simultaneously, though)

Minimally logged DELETE

```sql
-- 1st INSERT WITH (TABLOCK)
INSERT NewTable
 WITH (TABLOCK) (ID, Col1)
SELECT ID, Col1
FROM SomeTable
WHERE Col1 = 'X';

-- TRUNCATE
TRUNCATE TABLE SomeTable;

-- 2nd INSERT WITH (TABLOCK)
SET IDENTITY_INSERT SomeTable ON;

INSERT SomeTable
 WITH (TABLOCK) (ID, Col1)
SELECT ID, Col1
FROM NewTable;

SET IDENTITY_INSERT SomeTable OFF;
```

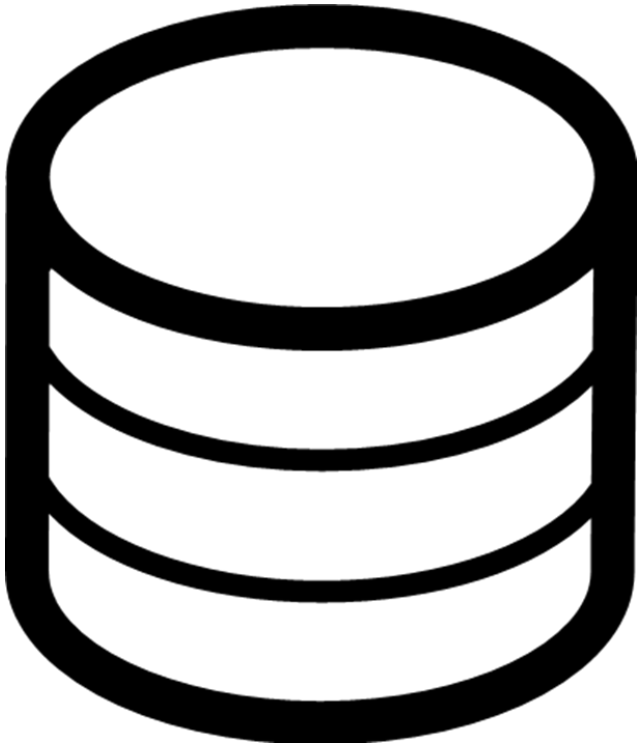
Minimally logged DELETE



So what's the catch?

- Best for removing a majority of the data
- DISABLE/re-ENABLE constraints
- All the rules for minimal logging apply

Desserts: modifying database settings



What sounds good is…

An INSERT

…of sorted or unsorted data

…into an empty table

…in a database

…with FULL recovery model

…and gimme minimal logging!

Bulk Logged Recovery Model

Change recovery model to BULK_LOGGED

Utilize minimally-logged transactions

Does NOT break the backup chain!

This works for INSERTs (and savvy DELETEs)

Bulk Logged Recovery Model

```sql
-- Change recovery model
ALTER DATABASE SomeDB
 SET RECOVERY BULK_LOGGED;

INSERT SomeTable
 WITH (TABLOCK) (Col1, Col2, Col3)
SELECT Col1, Col2, Col3
FROM SomeOtherTable;

ALTER DATABASE SomeDB
 SET RECOVERY FULL;
```

Bulk Logged Recovery Model



Warning:

- ALL database transactions affected
- No Point-In-Time recovery
- Consult with your Database Administrator

Let’s boldly finish with…

A DELETE

…of a bunch of data

…in a table with lots of records

…for those on a RBAR diet.

Delayed Durability

Delays the hardening of transactions to log

Log Buffer Cache

Work with data

Then write to T-Log

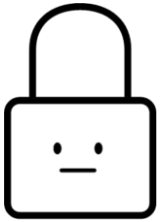
Use with lots of small transactions

Works for INSERTs, UPDATEs, and DELETEs

Delayed Durability

```sql
-- Allow Delayed Durability
ALTER DATABASE SomeDB
 SET DELAYED_DURABILITY = ALLOWED;

-- Start a transaction
BEGIN TRAN

    DELETE
    FROM SomeTable
    WHERE Id = @SomeVariable;

-- Use Delayed Durability
COMMIT
    WITH (DELAYED_DURABILITY = ON)

END
```

Delayed Durability



Why isn't everyone using this?

- **Lost transactions if SQL Server stops**
- Works best for lots of small transactions
- Consult with your Database Administrator

Let’s review the menu

Appetizers: Alter the target table

Query options	INSERT	UPDATE	DELETE
Clustered index on a heap			
Disable indexes			

Entrees: Transaction manipulation

Query options	INSERT	UPDATE	DELETE
Batching (WHILE with TOP)			
Explicit transactions			
Minimally logged INSERT			
Minimally logged DELETE			

Desserts: Database settings

Query options	INSERT	UPDATE	DELETE
Bulk Logged recovery model			
Delayed durability			

That’s the end. Thank you!

desertdba.com jeff@desertdba.com @desertdba