Faster Transactions:

Query Tuning for Data Manipulation

Jeff Iannucci

What is this session?

Improving performance for:

INSERT

UPDATE

DELETE

Le SQL Server

Log Buffer Cache

Transaction Log

Buffer Cache

Pages on Disk

Locks

Appetizers: Alter the target table

Entrees: Manipulate the transaction

Desserts: Modify the database settings

Appetizers: Alter the target table

Entrees: Manipulate the transaction

Desserts: Modify the database settings

Let’s start with…

An INSERT

…of sorted data

…into an empty table

…that has no indexes

…(also known as a heap.)

Clustered Index on a Heap

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

-- ...and then an ordered INSERT
INSERT SomeTable (Id, Col1, Col2)
SELECT ID, Col1, Col2
FROM SourceTable
ORDER BY ID;
```

Clustered Index on a Heap

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

-- ...and then an ordered INSERT
INSERT SomeTable (Id, Col1, Col2)
SELECT ID, Col1, Col2
FROM SourceTable
ORDER BY ID;
```

Clustered Index (CI) on Heap

Unsorted INSERTs are slower with a CI

Only faster when CI order matches sort order

Temporary tables can have a CI

Now let's try a bit of…

An UPDATE

…to most or all rows of a table

…that has a clustered index

…and non-clustered indexes.

Disabling indexes

```
-- 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

```
-- 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

REBUILDs take additional time

NCIs are not used until they are rebuilt

Disable all NCIs for INSERTs and DELETEs

Appetizers: Alter the target table

Entrees: Manipulate the transaction

Desserts: Modify the database settings

Today let's have…

A DELETE

…of a majority of data

…in a table with lots of rows

…with lots of user connections

…so don't slow other queries.

Batch using WHILE with TOP

```
-- 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

```
-- 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

Will likely result in a slower query for you

Will certainly result in partial “transaction” if stopped

Experiment with different methods and batch sizes

For RBAR dieters, try…

An UPDATE

…of much of the data

…in a table with lots of rows

…but each row is updated

…in a separate transaction.

Explicit transaction

```
DECLARE @Id int = 1;

-- Explicit start
BEGIN TRANSACTION;

WHILE @id <= 100000 BEGIN

    EXEC usp_UpdateSomething @Id = @id;

    SET @id += 1;

    END;

-- Explicit end
COMMIT;
```

Explicit transaction

```
DECLARE @Id int = 1;

-- Explicit start
BEGIN TRANSACTION;

WHILE @id <= 100000 BEGIN

    EXEC usp_UpdateSomething @Id = @id;

    SET @id += 1;

    END;

-- Explicit end
COMMIT;
```

Explicit transaction

You are locking resources during your transaction

Concurrent queries will have to wait for resources

Could result in blocking or deadlocks

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

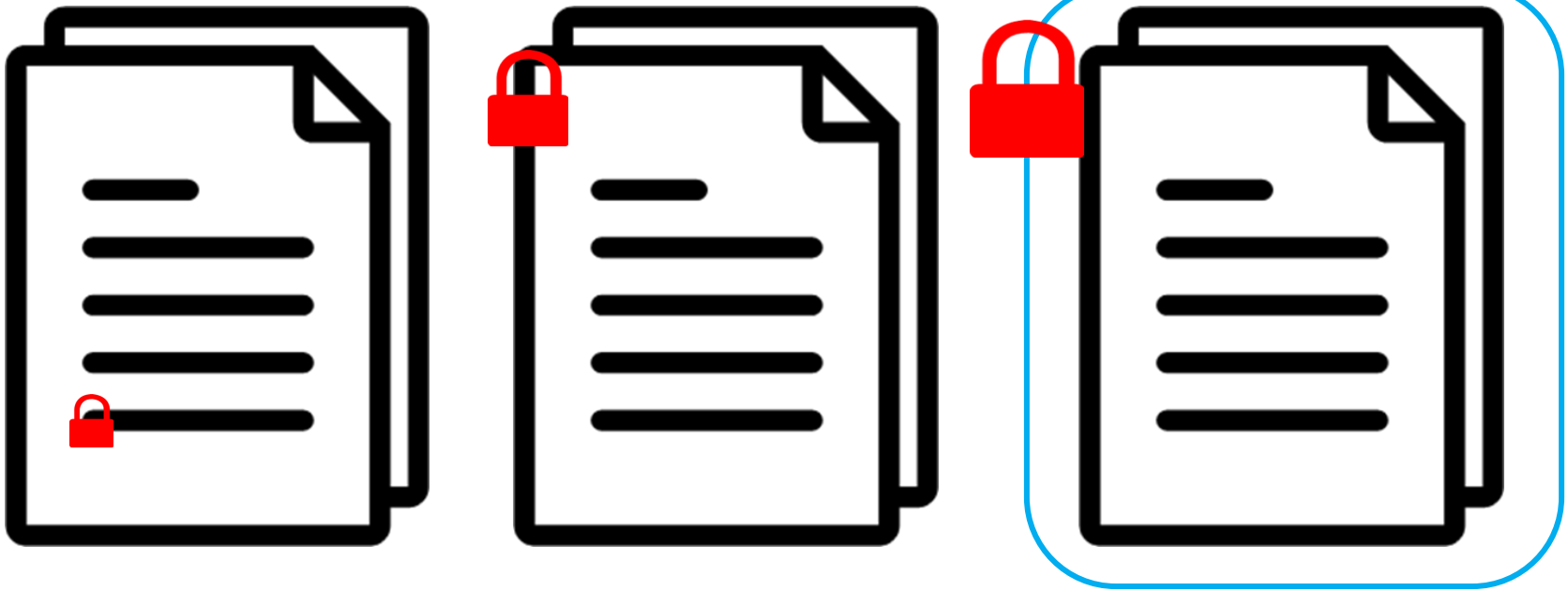
```
-- Minimal Logging with TABLOCK
INSERT SomeTable
 WITH (TABLOCK) (Col1, Col2, Col3)
SELECT Col1, Col2, Col3
FROM SomeOtherTable;
```

…“table lock”?

ROW LOCK

PAGE LOCK

TABLE LOCK



…“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

A few minimally-logged transaction rules

Database can NOT be 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://learn.microsoft.com/en-us/sql/relational-databases/import-export/prerequisites-for-minimal-logging-in-bulk-import?view=sql-server-ver16

Minimally logged INSERT

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

Minimally logged INSERT

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

Minimally logged INSERT

No guarantees of minimal logging with all those rules

Concurrent queries will likely have to wait to use the table

Minimal logging only works for INSERTs

Next you might enjoy…

A DELETE

…of a majority of data

…in a table with lots of rows

…and make it fast

…like minimal logging!

Minimally logged DELETE

```
-- 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

```
-- 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

All the rules for minimal logging apply

Best for removing a majority of the data

DISABLE/re-ENABLE constraints

Appetizers: Alter the target table

Entrees: Manipulate the transaction

Desserts: Modify the 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 give me minimal logging!

Bulk Logged Recovery Model

```
-- 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

```
-- 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

ALL database transactions affected

No Point-In-Time recovery while in Bulk Logged

Consult with your Database Administrator

Let’s boldly finish with…

A DELETE

…of a bunch of data

…in a table with lots of rows

…and then issue a ROLLBACK.

Accelerated Database Recovery (ADR)

```
ALTER DATABASE SomeDB
 SET ACCELERATED_DATABASE_RECOVERY = ON;

 -- Start a transaction
 BEGIN TRANSACTION

    DELETE
    FROM SomeTable
    WHERE Id = @SomeVariable

-- ROLLBACK
    ROLLBACK
```

Accelerated Database Recovery (ADR)

```
ALTER DATABASE SomeDB
 SET ACCELERATED_DATABASE_RECOVERY = ON;

 -- Start a transaction
 BEGIN TRANSACTION

    DELETE
    FROM SomeTable
    WHERE Id = @SomeVariable

-- ROLLBACK
    ROLLBACK
```

Accelerated Database Recovery

Writing more data during your transaction

Requires more space in data files

Database level setting


Menu
Antipasti
Secondi
Dolci

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			
Accelerated Database Recovery			

Transaction cost multipliers

Triggers

Replication

Change Data Capture

Availability Groups

desertdba.com

jeff@desertdba.com

@desertdba