SINGLE STATEMENT, MANY CHANGES

How One Statement Can Modify Multiple Tables

Deborah Melkin
NESQL User Group
January 13, 2021

WHO AM I?

- 20 years as a DBA
- Mainly work with SQL Server
- Mainly work with OLTP but have worked with some data marts.
- NESQL Board Member
- SQL Saturday\User Group Speaker
- IDERA ACE Class of 2020
- Speaker Idol Winner 2019
- Microsoft MVP – Data Platform

Random facts:

- I'm the alto section leader in my choir.
- I go to bluegrass jams regularly.
- I've been learning guitar and now mandolin.
- I am a bit of a musical theater geek.

AGENDA

- EXPLICIT CHANGES
- IMPLICIT CHANGES
- CHANGES AFTER THE CHANGES

EXPLICIT CHANGES

CHANGES THAT ARE INCLUDED DIRECTLY IN THE STATEMENTS

INSERTED \ DELETED

- Internal System Tables that exist during a transaction to store the data as existed before the change and as it will exist after the transaction is committed.
  - The table structure matches the table.
  - You cannot modify the inserted or deleted tables.
  - They are stored in memory.

OUTPUT INTO

- Part of the DML statement
- Directs the output to variables, table variables, tables
- Limitations:
  - Cannot insert into a table that is any part of a foreign key constraint
  - Cannot insert into a table that has a trigger on it
  - Some restrictions with replication
  - Target cannot be a remote table, view or common table expression
  - Will always use a serial plan
  - Target table not eligible for parallelism

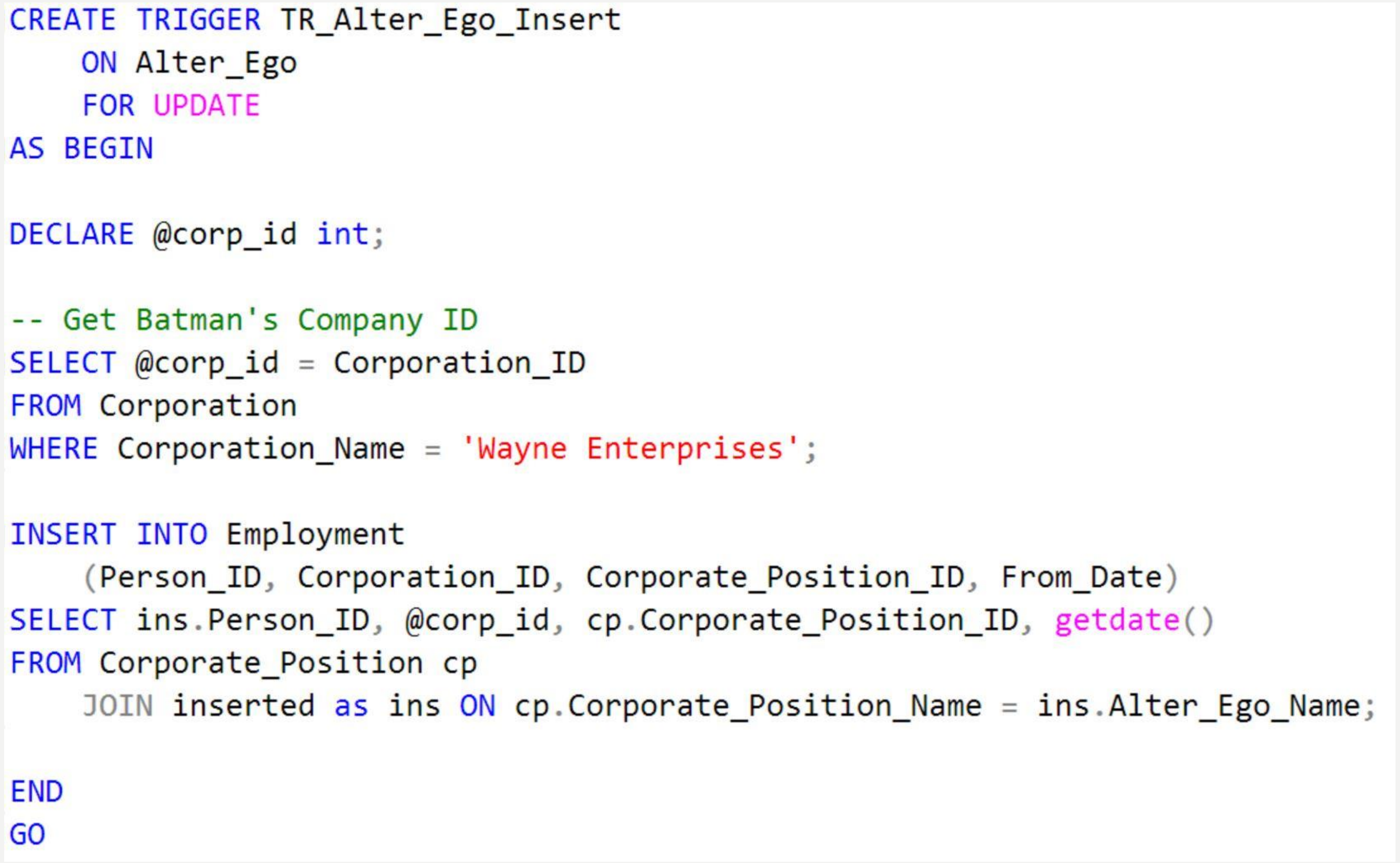
OUTPUT INTO

```
UPDATE Alter_Ego
SET Person_ID = Person.Person_ID
    OUTPUT  inserted.Person_ID, deleted.Person_ID
    INTO #Superhero_Changeover (New_Secret_ID, Old_Secret_ID)
FROM Person
WHERE First_Name = 'Deborah'
AND Last_Name = 'Melkin'
AND Alter_Ego.Alter_Ego_Name = 'Wonder Woman'
;
```

TRIGGERS

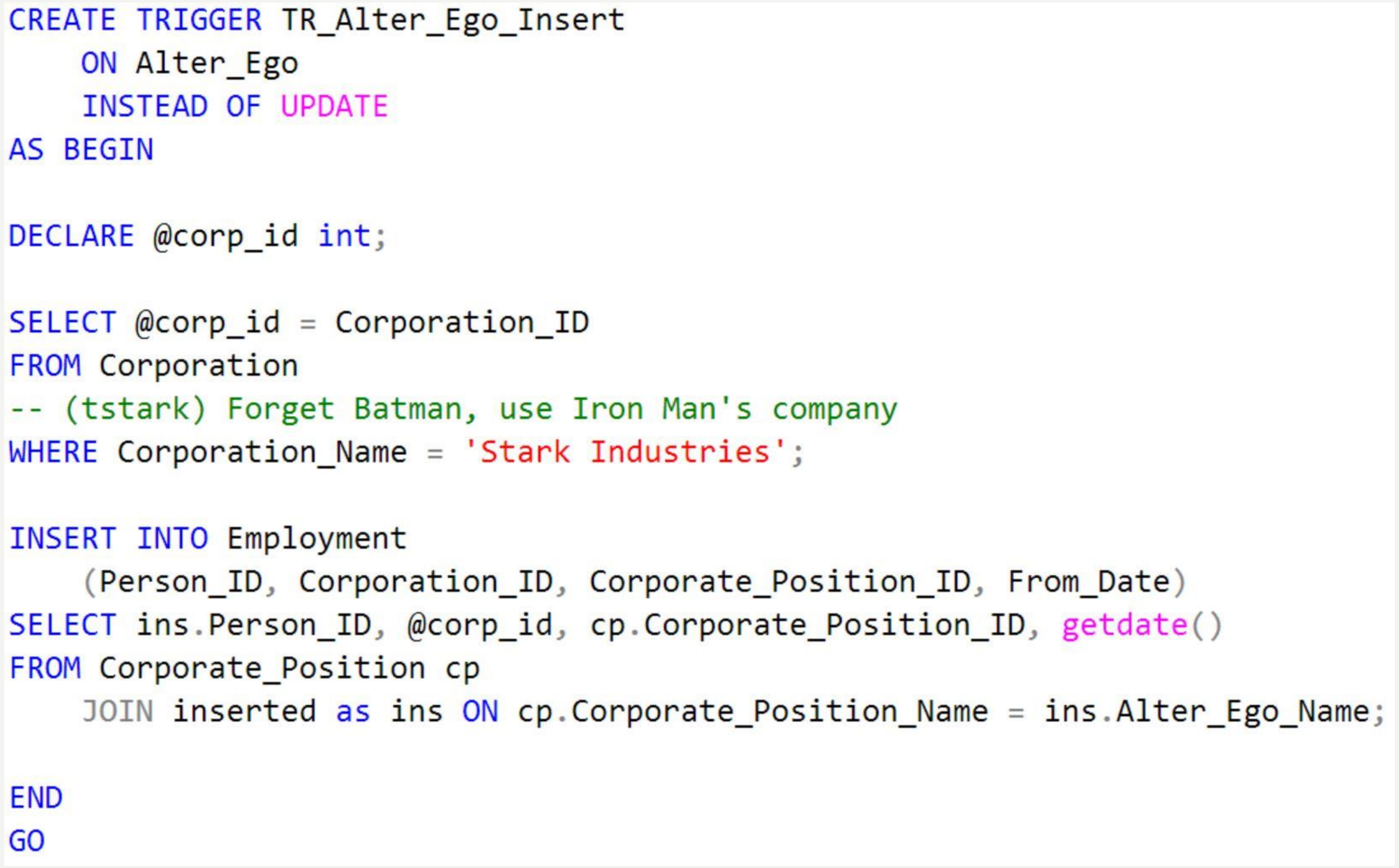
- Stored Procedures attached to a table or view executed as part of the data change
  - After the change
  - Instead of the change
  - Part of the same transaction as the data change statement
- Can be assigned to execute for INSERT, UPDATE or DELETE
- Can have multiple triggers for the same or different actions on a single table
  - Can assign trigger to be the first or last trigger fired but no control in between

TRIGGERS



```
CREATE TRIGGER TR_Alter_Ego_Insert
    ON Alter_Ego
    FOR UPDATE
AS BEGIN

DECLARE @corp_id int;

-- Get Batman's Company ID
SELECT @corp_id = Corporation_ID
FROM Corporation
WHERE Corporation_Name = 'Wayne Enterprises';

INSERT INTO Employment
    (Person_ID, Corporation_ID, Corporate_Position_ID, From_Date)
SELECT ins.Person_ID, @corp_id, cp.Corporate_Position_ID, getdate()
FROM Corporate_Position cp
    JOIN inserted as ins ON cp.Corporate_Position_Name = ins.Alter_Ego_Name;


END
GO
```

TRIGGERS



```
CREATE TRIGGER TR_Alter_Ego_Insert
    ON Alter_Ego
    INSTEAD OF UPDATE
AS BEGIN

DECLARE @corp_id int;

SELECT @corp_id = Corporation_ID
FROM Corporation
-- (tstark) Forget Batman, use Iron Man's company
WHERE Corporation_Name = 'Stark Industries';

INSERT INTO Employment
    (Person_ID, Corporation_ID, Corporate_Position_ID, From_Date)
SELECT ins.Person_ID, @corp_id, cp.Corporate_Position_ID, getdate()
FROM Corporate_Position cp
    JOIN inserted as ins ON cp.Corporate_Position_Name = ins.Alter_Ego_Name;

END
GO
```

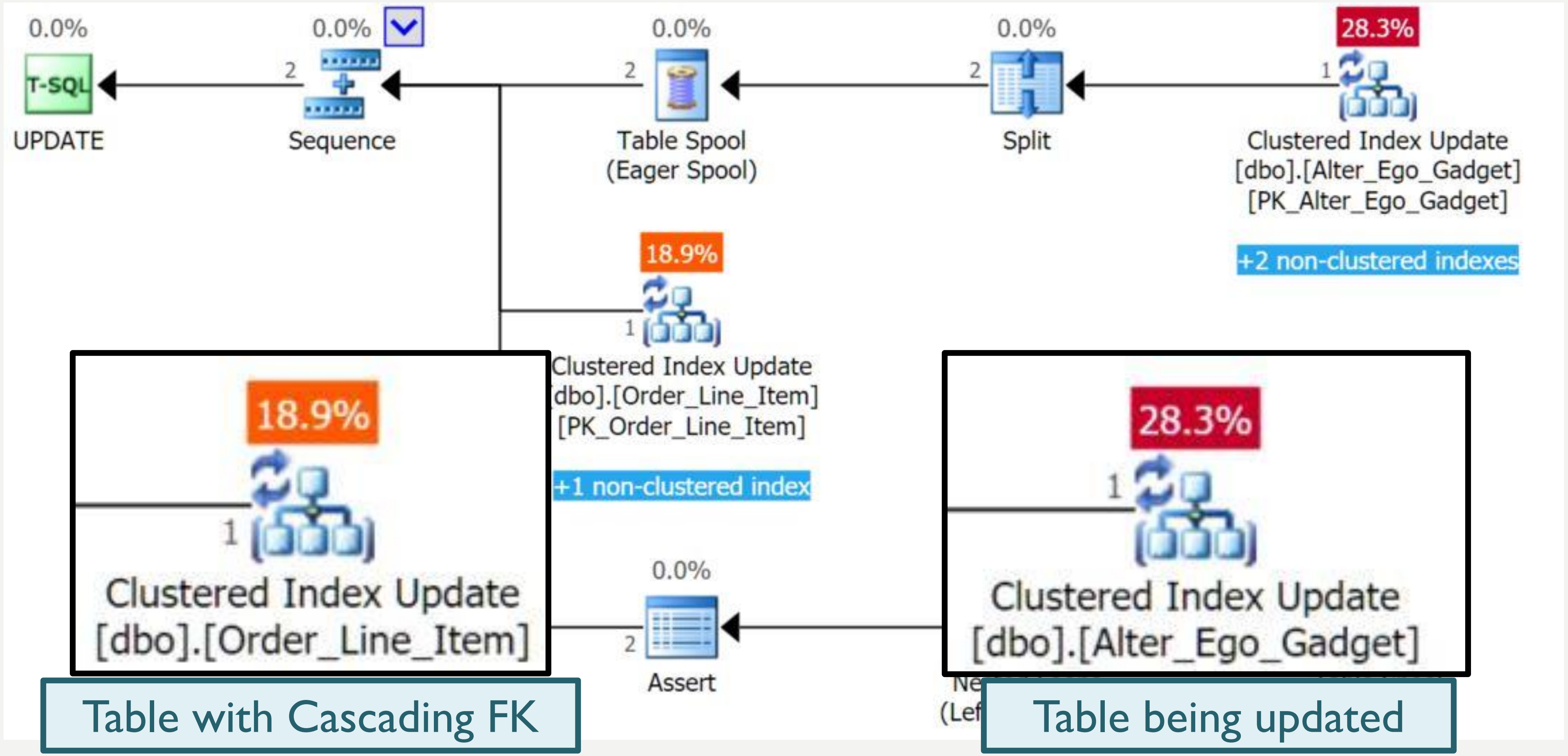
IMPLICIT CHANGES

CHANGES THAT ARE BUILT INTO THE DATABASE SCHEMA

CASCADING FOREIGN KEYS

- Defines actions that should be taken on the referencing columns when the referenced column is updated or referenced row is deleted
  - SET NULL
    - Sets the referencing column to NULL
  - SET DEFAULT
    - Sets the referencing column to a default value
  - CASCADE
    - Updates the referencing column to the same value as the referenced column
    - Deletes the referencing record when the referenced record is deleted
  - NO ACTION

CASCADING FOREIGN KEYS



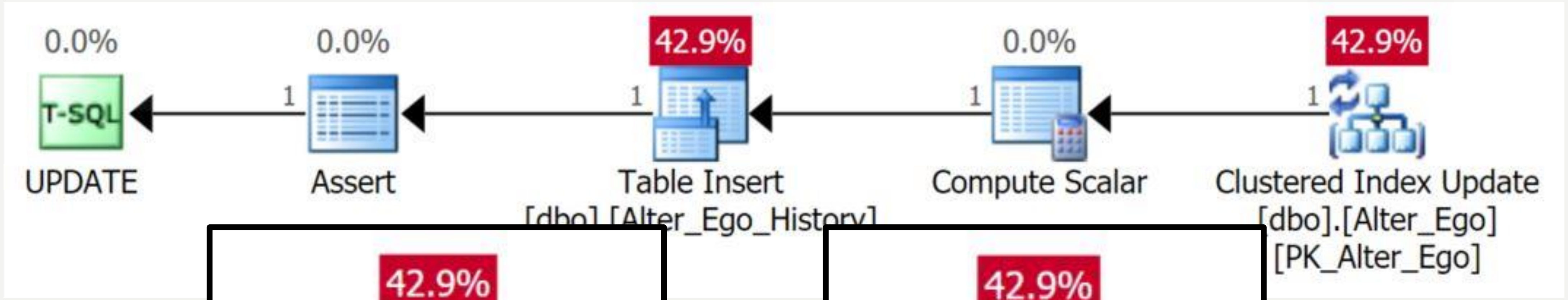
TEMPORAL TABLES

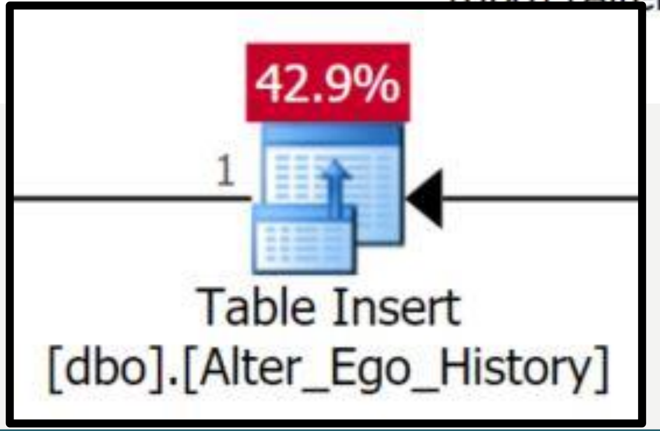
- Creates system version of the changes to a table
  - Only records UPDATEs and DELETEs
- Versioning is managed by the system
- History table can be created automatically by SQL Server or it can be specified
- Retention for data in the history table can be specified start with SQL 2017
- Limitations:
  - The history table must exist in the same database
  - Cannot manually modify the system date columns on the table being versioned.
  - No direct data modification of the history table
  - Some restrictions with Replication

TEMPORAL TABLES

- Can query the table to return the data as it existed at a point in time
- Options for querying data:
  - AS OF
  - FROM
  - BETWEEN
  - CONTAINED IN
  - ALL

TEMPORAL TABLES





Insert into History Table

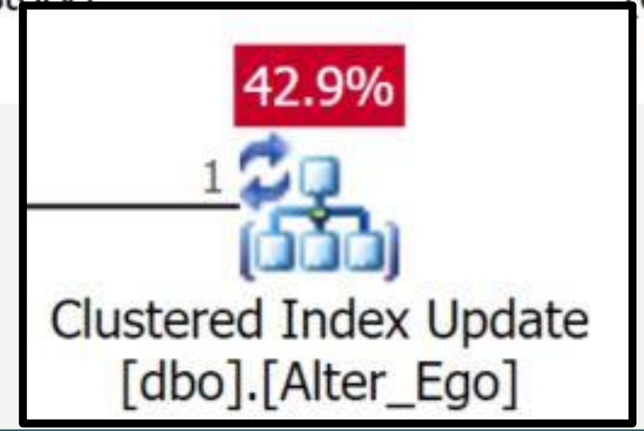
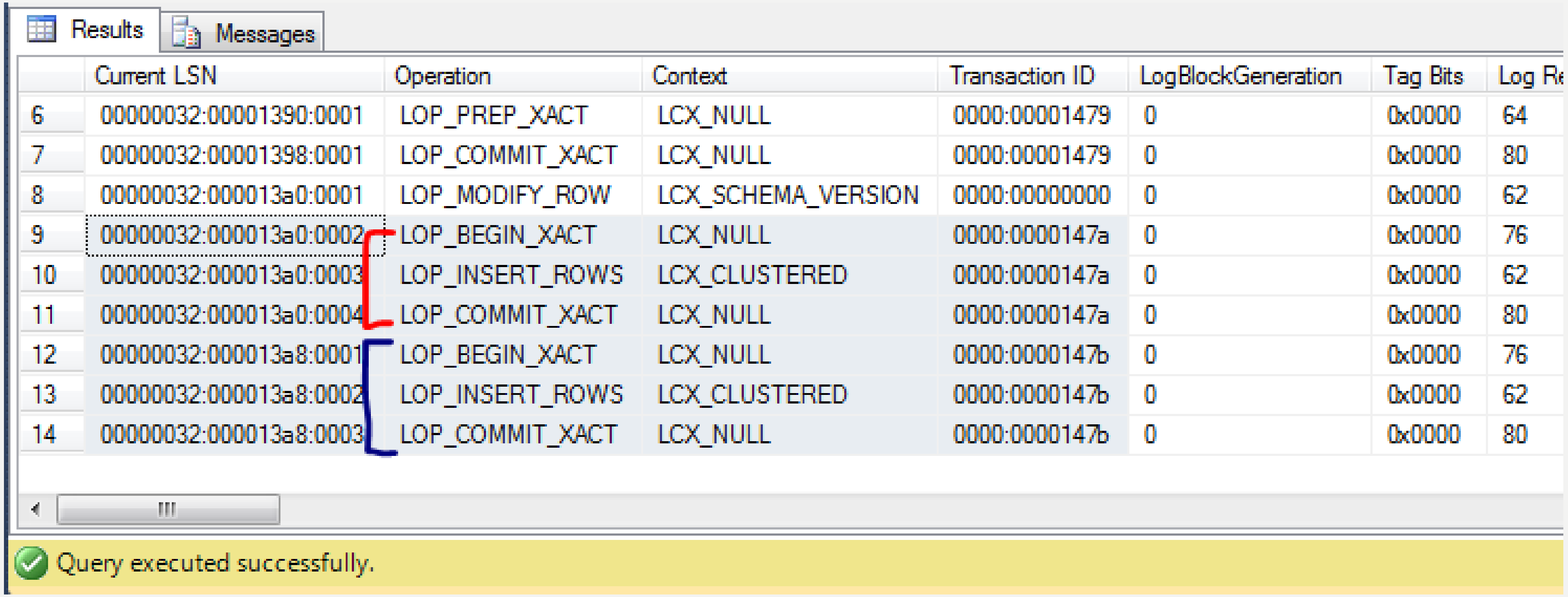


Table being updated

AFTER THE TRANSACTION

CHANGES THAT HAPPEN BECAUSE OF THE CHANGE

LOG READERS



	Current LSN	Operation	Context	Transaction ID	LogBlockGeneration	Tag Bits	Log Re
6	00000032:00001390:0001	LOP_PREP_XACT	LCX_NULL	0000:00001479	0	0x0000	64
7	00000032:00001398:0001	LOP_COMMIT_XACT	LCX_NULL	0000:00001479	0	0x0000	80
8	00000032:000013a0:0001	LOP_MODIFY_ROW	LCX_SCHEMA_VERSION	0000:00000000	0	0x0000	62
9	00000032:000013a0:0002	LOP_BEGIN_XACT	LCX_NULL	0000:0000147a	0	0x0000	76
10	00000032:000013a0:0003	LOP_INSERT_ROWS	LCX_CLUSTERED	0000:0000147a	0	0x0000	62
11	00000032:000013a0:0004	LOP_COMMIT_XACT	LCX_NULL	0000:0000147a	0	0x0000	80
12	00000032:000013a8:0001	LOP_BEGIN_XACT	LCX_NULL	0000:0000147b	0	0x0000	76
13	00000032:000013a8:0002	LOP_INSERT_ROWS	LCX_CLUSTERED	0000:0000147b	0	0x0000	62
14	00000032:000013a8:0003	LOP_COMMIT_XACT	LCX_NULL	0000:0000147b	0	0x0000	80

Query executed successfully.

REPLICATION

- Data changes in the publication database are sent to one or more subscription databases
- The distribution database stores metadata and history data for all types of replication, and transactions for transactional replication
- Tables can be part of multiple subscriptions
- Uses SQL Server Agent jobs to run the various tasks needed for Replication

REPLICATION

- Multiple types of replication
  - Snapshot
  - Transactional
  - Peer-to-Peer
  - Merge
- Subscriptions can be Push or Pull

CHANGE DATA CAPTURE (CDC)

- "Before" and "After" data are written to system created tables for future use
- Limitations
  - Cannot rename the objects or schemas
  - Default retention policy

OTHER TRANSACTION LOG READERS

- Database Change Tracking
  - “lightweight” version of CDC
- Availability Groups
  - Full replicas of the database
  - Synchronous and Asynchronous
  - The transaction log records of the primary database are sent to the secondary replicas

DEMOS

LET’S REVIEW

TAKE-AWAYS

- Not all data modifications can be seen by traces
- Not all data modifications occur in the same database
- The amount of data that changes affects more than just the table but everything having to do with the transaction logs
- Troubleshooting data modifications may not have to do with the initial update but changes that occur because of the updates

ADDITIONAL RESOURCES

- OUTPUT
  - https://tinyurl.com/output-clause-transact-sql
  - https://tinyurl.com/insert-data-returned-output
  - https://www.erikdarlingdata.com/2020/01/an-unfortunate-side-effect-of-output/
  - https://www.sql.kiwi/2020/07/a-bug-with-halloween-protection-and.html
- Temporal Tables
  - https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-ver15
  - https://www.erikdarlingdata.com/sql-server/tracking-row-changes-with-temporal-columns/

ADDITIONAL RESOURCE (CONT'D)

- CDC
  - https://tinyurl.com/about-change-data-capture
  - https://www.sqlshack.com/change-data-capture-for-auditing-sql-server/
- Database Change Tracking:
  - https://www.sqlshack.com/creating-a-sql-server-audit-using-sql-server-change-tracking/
  - https://tinyurl.com/about-change-tracking

MORE QUESTIONS? LET ME KNOW!

- **Email:** dgmelkin@gmail.com
- **Twitter:** @dgmelkin
- **Blog:** DebtheDBA.wordpress.com
- **GitHub:** https://tinyurl.com/SingleStatmentManyChanges

Thanks for coming!