



#77 PARMA 2025

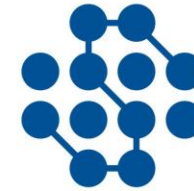
# SQL Server 2025 Optimized Locking in action!

Sergio Govoni



Slide and demos: [bit.ly/DataSatParma2025-sgovoni](https://bit.ly/DataSatParma2025-sgovoni)

# Sponsor & Org



**DATA SKILLS**  
UNDERSTANDING THE WORLD

M D  
massive dynamic holding H



dan|ela  
ma|visi  
COMMUNICATION

**Lucient<sup>1</sup>**  
ITALIA



# Sergio Govoni



Sergio Govoni



[linkedin.com/in/sgovoni](https://linkedin.com/in/sgovoni)



[github.com/segovoni](https://github.com/segovoni)



[x.com/segovoni](https://x.com/segovoni)



[bit.ly/sgovoni-MVP](https://bit.ly/sgovoni-MVP)


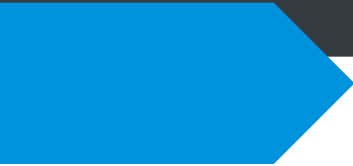
# Agenda

- SQL Server 2025 Optimized Locking
  - Key components
  - Underlying technologies
  - How it works
  - Demo




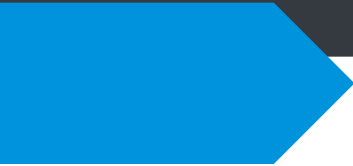
# Introduction to Optimized Locking



- In the landscape of modern applications, scalability and concurrency are crucial
  - Optimized Locking is a new technology available in SQL Server 2025
    - It redefines how SQL Server Engine handles locks, improving concurrency and efficiency
    - It helps to reduce lock memory and avoids lock escalations
- 
- 

# Introduction to Optimized Locking



- Optimized Locking is composed of two primary components
    - Transaction ID (TID) Locking
    - Lock After Qualification (LAQ)
  - Transaction ID Locking is designed to optimize memory usage in lock management
  - Lock After Qualification eliminates the risk of lock escalation and enhances concurrency in DML operations
- 
- 

# Introduction to Optimized Locking

- Optimized Locking is built on two existing technologies
  - Accelerated Database Recovery (ADR)
  - Read Committed Snapshot Isolation level (RCSI)
- [Accelerated Database Recovery](#) is mandatory, it must be enabled at the database level
- [Read Committed Snapshot Isolation](#) level is not a strict requirement; it significantly enhances because LAQ is active only when READ\_COMMITTED\_SNAPSHOT option is enabled

# Accelerated Database Recovery (ADR)



- It improves database availability, especially in the presence of long-running transactions, by redesigning the database engine recovery process
- When ADR is enabled, every row in the database internally contains a transaction ID (TID) that is persisted on disk



# Read Committed Snapshot Isolation (RCSI)



- Read Committed Snapshot is not a separate isolation level; it is a modification of the read committed isolation level when the `READ_COMMITTED_SNAPSHOT` option is enabled
- When it is enabled, locks are not used to protect data from updates by other transactions, it allows reading the last committed version from the snapshot, reducing contention between reads and writes


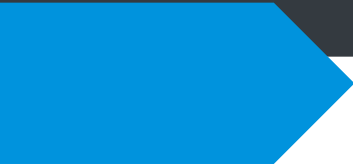




# How Optimized Locking works

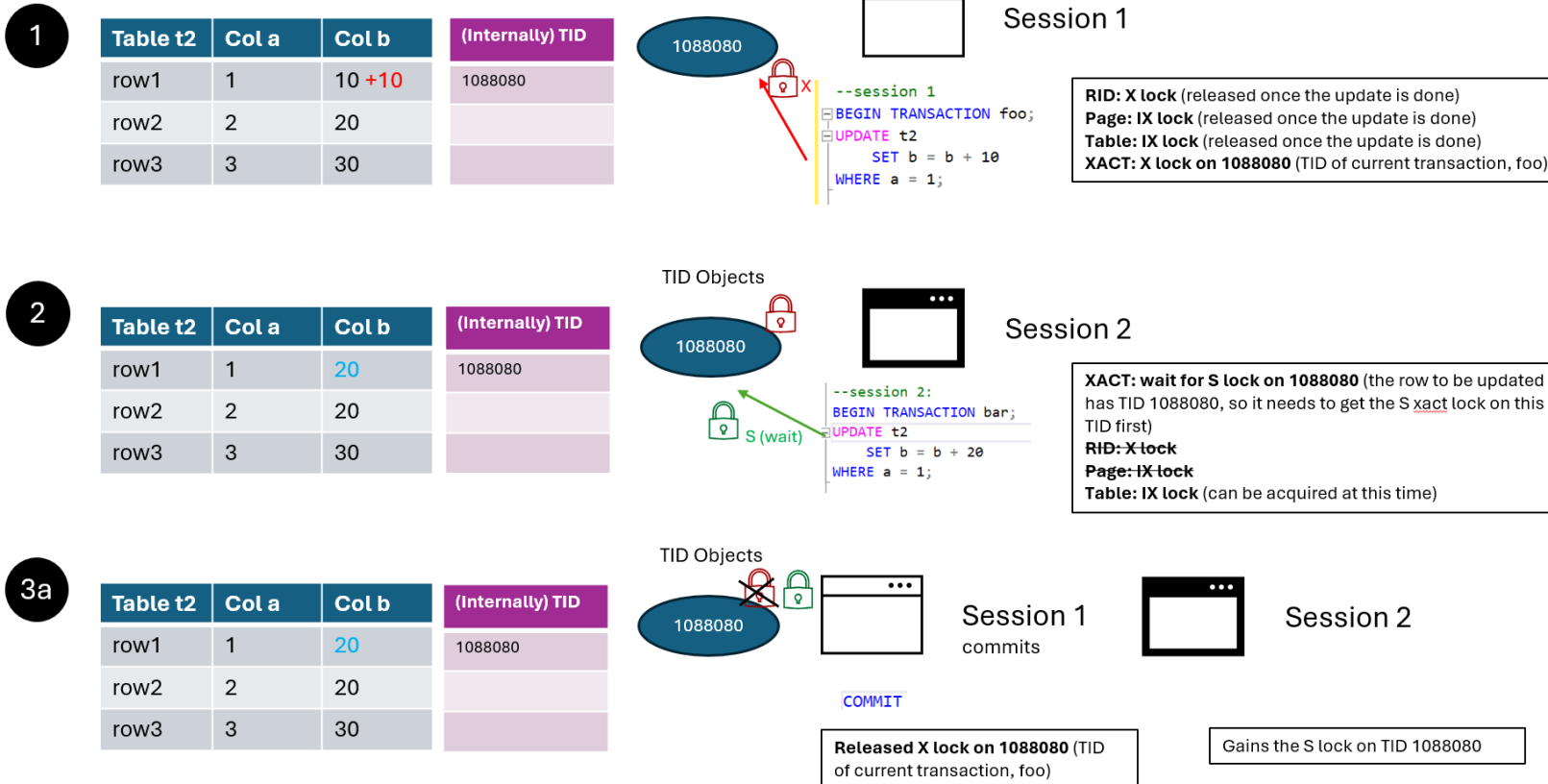
# Transaction ID (TID) locking in action



- With TID locking
    - Each row in the database internally contains a TID
    - TID is persisted on disk, and every transaction modifying a row assigns its own TID to that row
    - Instead of acquiring a lock on the row's key, a lock is taken on the row's TID
- 
- 

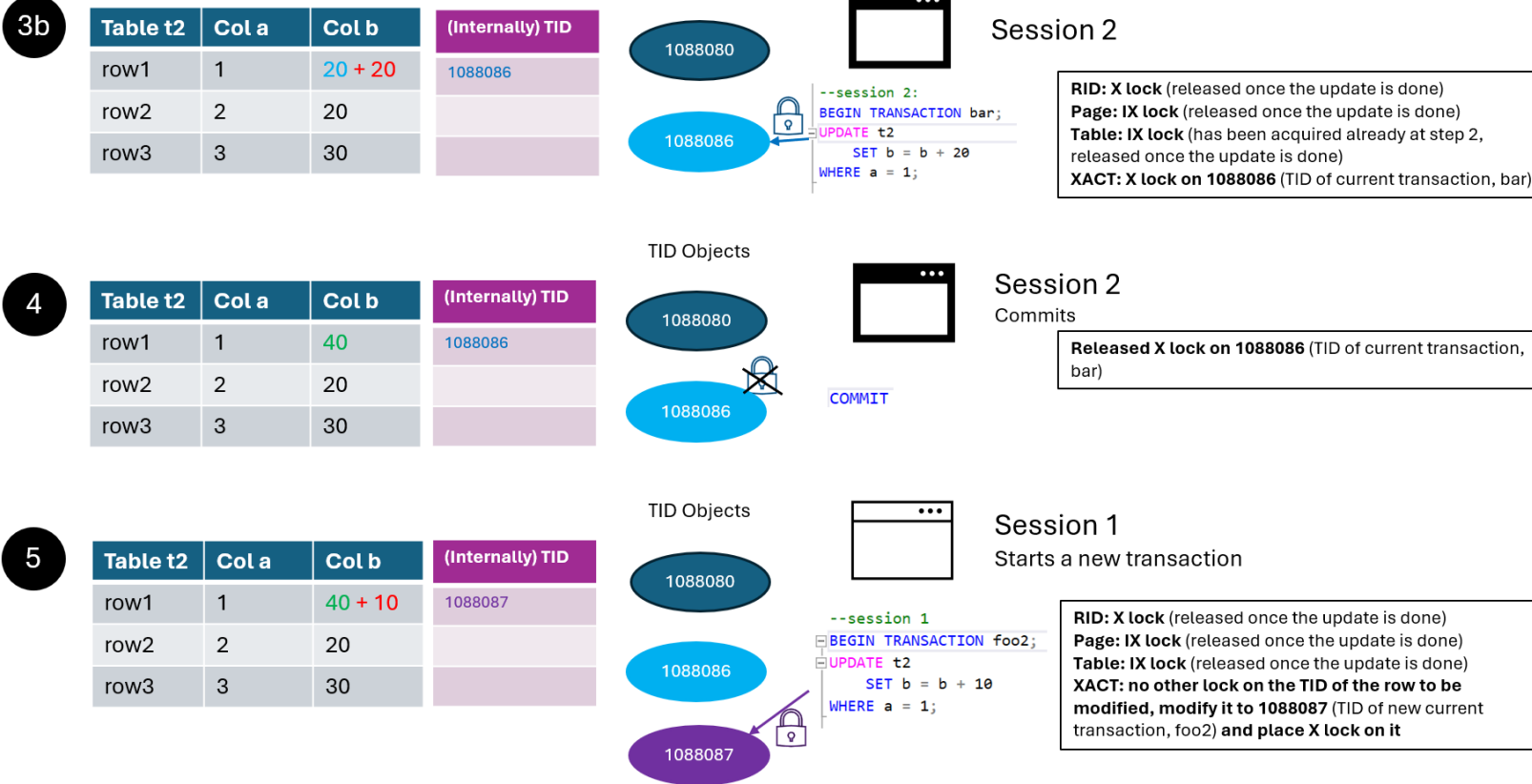
# Transaction ID (TID) locking in action

## With Optimized Locking – Scenario 1



# Transaction ID (TID) locking in action

## With Optimized Locking – Scenario 1



# Lock After Qualification (LAQ) in action

- One major cause of DML slowdowns is acquiring locks while searching for qualifying rows. LAQ modifies the way DML statements acquire locks
- Without optimized locking, queries evaluate predicates row by row, first acquiring a U lock, which is upgraded to an X lock if the row meets the condition. The X lock remains until the transaction ends

# Lock After Qualification (LAQ) in action



- With LAQ, predicates are evaluated on the latest committed row version without locks. If the condition is met, an X lock is acquired for the update and released immediately after
- This prevents blocking between concurrent queries modifying different rows



# Lock After Qualification (LAQ) in action

## With Optimized Locking – Scenario 2

1

Table t2	Col a	Col b	(Internally) TID
row1	1	10 +10	1088163
row2	2	20	
row3	3	30	

TID Objects



Session 1

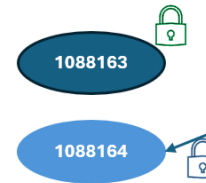
```
--session 1
BEGIN TRANSACTION foo;
UPDATE t2
  SET b = b + 10
  WHERE a = 1;
```

**RID:** X lock (51960,1) (released once the update is done)  
**Page:** IX lock (released once the update is done)  
**Table:** IX lock (released once the update is done)  
**XACT:** X lock on 1088163 (TID of current transaction, foo)

2

Table t2	Col a	Col b	(Internally) TID
row1	1	20	1088163
row2	2	20 +10	1088164
row3	3	30	

TID Objects



Session 2

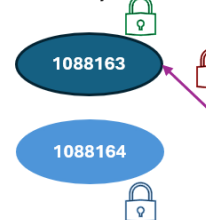
```
--session 2:
BEGIN TRANSACTION bar;
UPDATE t2
  SET b = b + 10
  WHERE a = 2;
```

**RID:** X lock (51960,65537) (released once the update is done)  
**Page:** IX lock (released once the update is done)  
**Table:** IX lock (released once the update is done)  
**XACT:** X lock on 1088164 (TID of current transaction, bar)

3

Table t2	Col a	Col b	(Internally) TID
row1	1	20	1088163
row2	2	30	1088164
row3	3	30	

TID Objects



Session 2

```
--session 2:
UPDATE t2
  SET b = b + 20
  WHERE a = 1;
```

**XACT:** wait for S lock on 1088163 (the row to be updated has TID 1088163, so it needs to get the S xact lock on this TID first)  
**RID:** X lock  
**Page:** IX lock  
**Table:** IX lock (already has it)

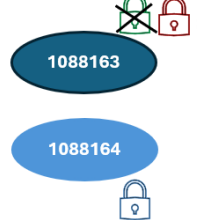
# Lock After Qualification (LAQ) in action

## With Optimized Locking – Scenario 2

4a

Table t2	Col a	Col b	(Internally) TID
row1	1	20	1088163
row2	2	30	1088164
row3	3	30	

TID Objects



Session 1  
commits



Session 2

COMMIT

Released X lock on 1088163 (TID of current transaction, foo)

Gains the S lock on TID 1088163

4b

Table t2	Col a	Col b	(Internally) TID
row1	1	20 + 20	1088164
row2	2	30	1088164
row3	3	30	



Session 2

```
--session 2:  
BEGIN TRANSACTION bar;  
UPDATE t2  
SET b = b + 20  
WHERE a = 1;
```

**RID:** X lock (51960,1) (released once the update is done)  
**Page:** IX lock (released once the update is done)  
**Table:** IX lock (already has it)  
**XACT:** X lock on 1088084 (already has it, update the TID for the row to be updated)



# DEMO

# Summary

- Optimized Locking represents a significant evolution in concurrency management; it redefines how SQL Server Engine handles locks
- By using TID locking and LAQ, optimized locking reduces memory consumption and eliminates the lock escalation
- SQL Server 2025 is now in GA
- In Azure SQL Database, optimized locking is enabled by default

# Resources

- [Download SQL Server 2025 today](#)
- [SQL Server 2025 documentation](#)
- [SQL Server 2025 Optimized Locking](#)
- [How Optimized Locking in SQL Server 2025 handles concurrent updates](#)
- [SQL Server 2025 Optimized Locking: TID internals](#)



Thanks!!!