



# Exploring Optimized Locking

**John Morehouse**  
**Principal Consultant**

Lexington Data Technology Group  
Monday March 4, 2024

**DCAC** 



# John Morehouse

Principal Consultant


Denny Cherry & Associates

✉ [john@dcac.com](mailto:john@dcac.com)

 [/in/johnmorehouse](https://www.linkedin.com/in/johnmorehouse)

 [@SQLRUS](https://twitter.com/SQLRUS)

 [Sqlrus.com](http://Sqlrus.com)

 He/Him

Community Speaker

Blogger/Tweeter

Conference Organizer

---

MVP – Data Platform

Friend of Redgate

VMWare vExpert



Certified IT professionals to help achieve IT goals

Clients ranging from small business to Fortune 10  
corporations

Help save on costs while improving IT reliability and solving  
challenges





<https://bit.ly/mypresentationfiles>



# Objectives

---



- Current problems
- Lock refresher
- Optimized Locking Components
- How does it work
- Limitations
- Q&A



Slides & Demos

# Current Problems

---



- ACID Compliance
- Lock escalation
- Long-term locking
- Excessive blocking
- Excessive memory utilization





# Locking Refresher



Lock	Purpose
IX/IU – Intent Lock	Establishes a lock hierarchy,
U – Update Lock	Used on resources that can be updated.
X – Exclusive Lock	Used for data-modification operations, such as INSERT, UPDATE, or DELETE.
S – Shared Lock	Used for read operations that do not change or update data

# What can be locked?



Resource Lock	Description
Key	Lock on a row in an index
Object	Lock on table, procedure, view, etc
Page	Lock on an 8-KB page
RID	Lock on a single row in a heap
Xact	Lock on a transaction



# Lock Compatibility Matrix



Existing/Request Lock	IS	S	U	IX	X
Intent shared (IS)	Yes	Yes	Yes	Yes	No
Shared (S)	Yes	Yes	Yes	No	No
Update (U)	Yes	Yes	No	No	No
Intent exclusive (IX)	Yes	No	No	Yes	No
Exclusive (X)	No	No	No	No	No

# Traditional Locking

---



- Update lock taken on rows to allow for predicate evaluation
- If predicate is satisfied, then an exclusive lock is taken on the row
- Locks are held until the end of the transaction

# Traditional Locking



```
ALTER DATABASE [Locking] SET READ_COMMITTED_SNAPSHOT OFF;  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

```
CREATE TABLE t1 (a int, b int);  
INSERT INTO t1 VALUES (1,10), (2,20), (3,30);
```

```
-- TID2: Increase b by 10  
BEGIN TRAN UPDATE t1 SET b=b+10;
```



p1: Data Page for t1

r1: 1   20   TID2
r2: 2   30   TID2
r3: 3   40   TID2

Lock Manager

Lock Mode	Lock Type	Lock Resource
IX	OBJECT	t1
IX	PAGE	p1
X	RID	r1
X	RID	r2
X	RID	r3



Old  
School

Updating 1 million rows  
might require 1 million  
exclusive (X) row locks  
held until the end of the  
transaction.

# Optimized Locking Components

---



- Accelerated Database Recovery
- Transaction ID (TID)
- Lock After Qualification (LAQ)

# ADR Overview

---



- Persisted Version Store (PVS) lives in user databases
- In-row versions versus off-row versions (PVS)
- Facilitates much faster rollback operations
- Eliminates long-running transaction rollbacks

# Transaction ID (TID)

---



- A unique identifier
- Each row will contain the TID that last modified it
- Lock will be held on the transaction ID versus row key



# Lock After Qualification (LAQ)

---



- Predicate is applied to the row using the latest version of the row
- If the predicate is not satisfied, move to the next row
- If the predicate is satisfied, an exclusive (X) lock is placed on the row
- Can retry predicate evaluation as needed due to previous exclusive lock

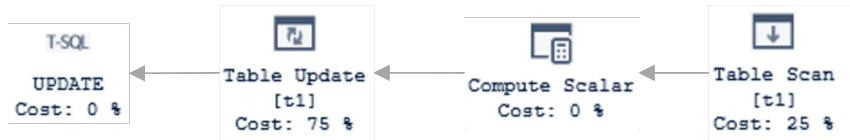
# Traditional Blocking



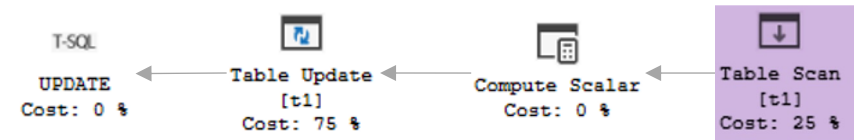
```
ALTER DATABASE [db1] SET READ_COMMITTED_SNAPSHOT ON;  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

```
CREATE TABLE t1 (a int, b int);  
INSERT INTO t1 VALUES (1,10), (2,20), (3,30);
```

```
-- TID2 [SESSION 1]: Increase b by 10 where a=1  
BEGIN TRAN UPDATE t1 SET b=b+10 where a=1;
```



```
-- TID3 [SESSION 2]: Increase b by 10 where a=2  
BEGIN TRAN UPDATE t1 SET b=b+10 where a=2;
```



## p1: Data Page for t1

## Row version store

Row ~~does not~~ qualify

r1: 1   20   TID2	✓
r2: 2   20   TID1	✗
r3: 3   30   TID1	✗

r1: 1   10   TID1
-------------------

## Lock Manager

Lock Mode	Lock Type	Resource	Owner	Status
IX	OBJECT	t1	TID2, TID3	GRANT
IX	PAGE	p1	TID2	GRANT
X	RID	r1	TID2	GRANT
IU	PAGE	p1	TID3	GRANT
U	RID	r1	TID3	WAIT

Slide Credit: Perry Skountrianos/Prashanth Purnananda & the Microsoft team who brought you optimized locking

⛔ Session 2 is blocked waiting for Session 1 to commit

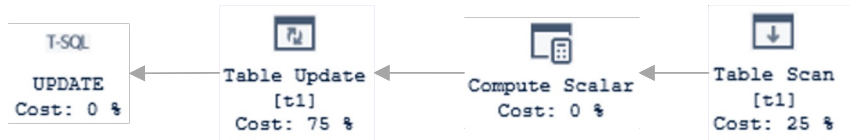
# Optimized Locking



```
ALTER DATABASE [db1] SET READ_COMMITTED_SNAPSHOT ON;  
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

```
CREATE TABLE t1 (a int, b int);  
INSERT INTO t1 VALUES (1,10), (2,20), (3,30);
```

```
-- TID2 [SESSION 1]: Increase b by 10 where a=1  
BEGIN TRAN UPDATE t1 SET b=b+10 where a=1;
```

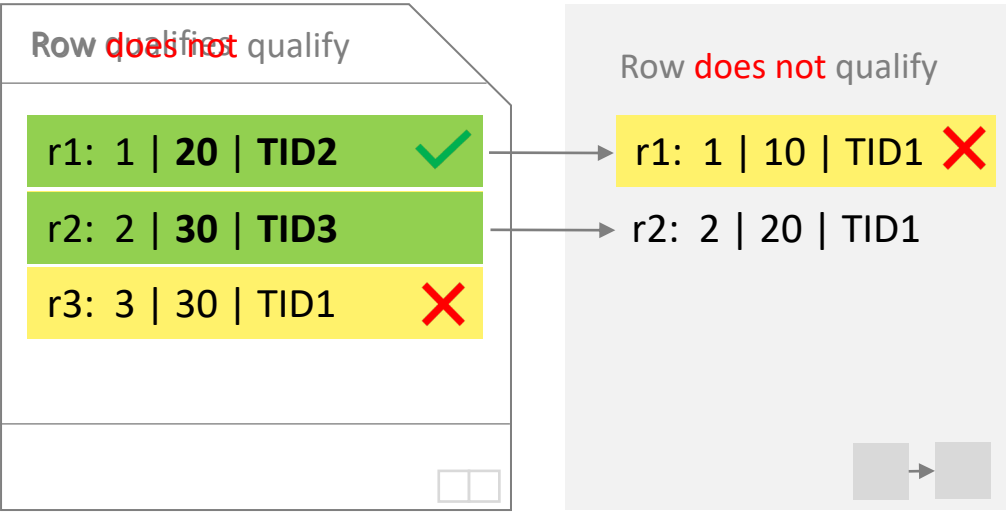


```
-- TID3 [SESSION 2]: Increase b by 10 where a=2  
BEGIN TRAN UPDATE t1 SET b=b+10 where a=2;
```



p1: Data Page for t1

Row version store



Lock Manager

Lock Mode	Lock Type	Resource	Owner	Status
X	XACT	TID2	TID2	GRANT
IX	OBJECT	t1	TID2, TID3	GRANT
IX	PAGE	p1	TID2	GRANT
IX	PAGE	p1	TID3	GRANT
X	RID	r2	TID3	GRANT

Slide Credit: Perry Skountrianos/Prashanth Purnananda & the Microsoft team who brought you optimized locking

✓ Session 2 is not blocked by Session 1



New  
School

Updating 1 million rows might require 1 million X row locks but each lock is released as soon as each row is updated,

Only one TID lock will be held until the end of the transaction.

# Warning

---



“

*Even without LAQ, applications should not assume that SQL Server (under versioning isolation levels) will guarantee strict ordering, without using locking hints.*

<https://learn.microsoft.com/en-us/sql/relational-databases/performance/optimized-locking?view=azuresqlldb-current>

# Best Practices & Troubleshooting



- Avoid locking hints
- Make sure RCSI is enabled
- New entries for Deadlock Graphs
- New Waits introduced:
  - LCK\_M\_S\_XACT\_READ
  - LCK\_M\_S\_XACT\_MODIFY
  - LCK\_M\_S\_XACT

LCK\_M\_S\_XACT

Occurs when a task is waiting for a shared lock on an XACT `wait_resource` type, where the intent cannot be inferred. Rare. Related to [optimized locking](#).

LCK\_M\_S\_XACT\_READ

Occurs when a task is waiting for a shared lock on an XACT `wait_resource` type, with an intent to read. Related to [optimized locking](#).

LCK\_M\_S\_XACT\_MODIFY

Occurs when a task is waiting for a shared lock on an XACT `wait_resource` type, with an intent to modify. Related to [optimized locking](#).

# Limitations

---



- Azure SQL Database Only (for now)
  - DTU service tier
  - vCore Provisioned tier (including serverless)
- Accelerated Database Recovery Required
- Repeatable Read & Serializable Isolation forces the lock to be held on the row or page until the end of the transaction



# Summary

---



- Better concurrency
- Significantly reduced locking and lock memory
- On by default in Azure SQL Database
  - ADR/RCSI is enabled by default as well
- Every SQL Server DBA should be watching this feature arrive to the box product

# Resources

---



- Optimized Locking - <https://learn.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-locking-and-row-versioning-guide?view=sql-server-ver16&source=recommendations>
- Article - <https://www.red-gate.com/simple-talk/databases/sql-server/database-administration-sql-server/optimized-locking-in-azure-sql-database/> (Simple Talk - Aaron Bertrand)



**Questions?**  
**Answers!**

**Got  
Questions?**

**Follow Me on  
Twitter X!**

**Check out  
my blog!**

# John Morehouse

**Denny Cherry & Associates Consulting**

**DCAC\***



[john@dcac.com](mailto:john@dcac.com)



**Sqlrus.com**



**@SQLRUS**



**/in/johnmorehouse**



Slides & Demos