Maximize Performance Using Read Committed Snapshot Isolation

Haripriya Naidu

She/Her

Database Administrator

SQL Sat Boston 9/27/25

Haripriya Naidu

- SQL Database Administrator
- 12 years
- AWS Certified Solutions Architect
- Speaker
- Blogger
- New Stars of Data 2024

gohigh.substack.com

linkedin.com/in/haripriya-naidu1215/

github.com/haripriyasb/RCSI

Problem I needed to solve

Agenda

- Read Committed and its problems
- Read Committed Snapshot Isolation
- How it works and its benefits
- Version Store
- Tradeoffs

Isolation

- Concurrently running transactions shouldn’t interfere with each other
- Example:
  - Two people editing the same document.
  - Without isolation, one person might see the other’s half-finished

Isolation Levels

Locking model

Read Uncommitted
Read Committed
Repeatable Read
Serializable

Locking & Row Versioning model

Read Committed Snapshot
Snapshot

What is Read Committed

- Default isolation level
- It makes two guarantees:
    1. No dirty reads – read only committed data
    2. No dirty writes – overwrite only on committed data

Problems with Read Committed

Read vs Write blocking

- Write operations are blocked when read is going on

  OR

- Read operations are blocked when write is going on

To resolve this

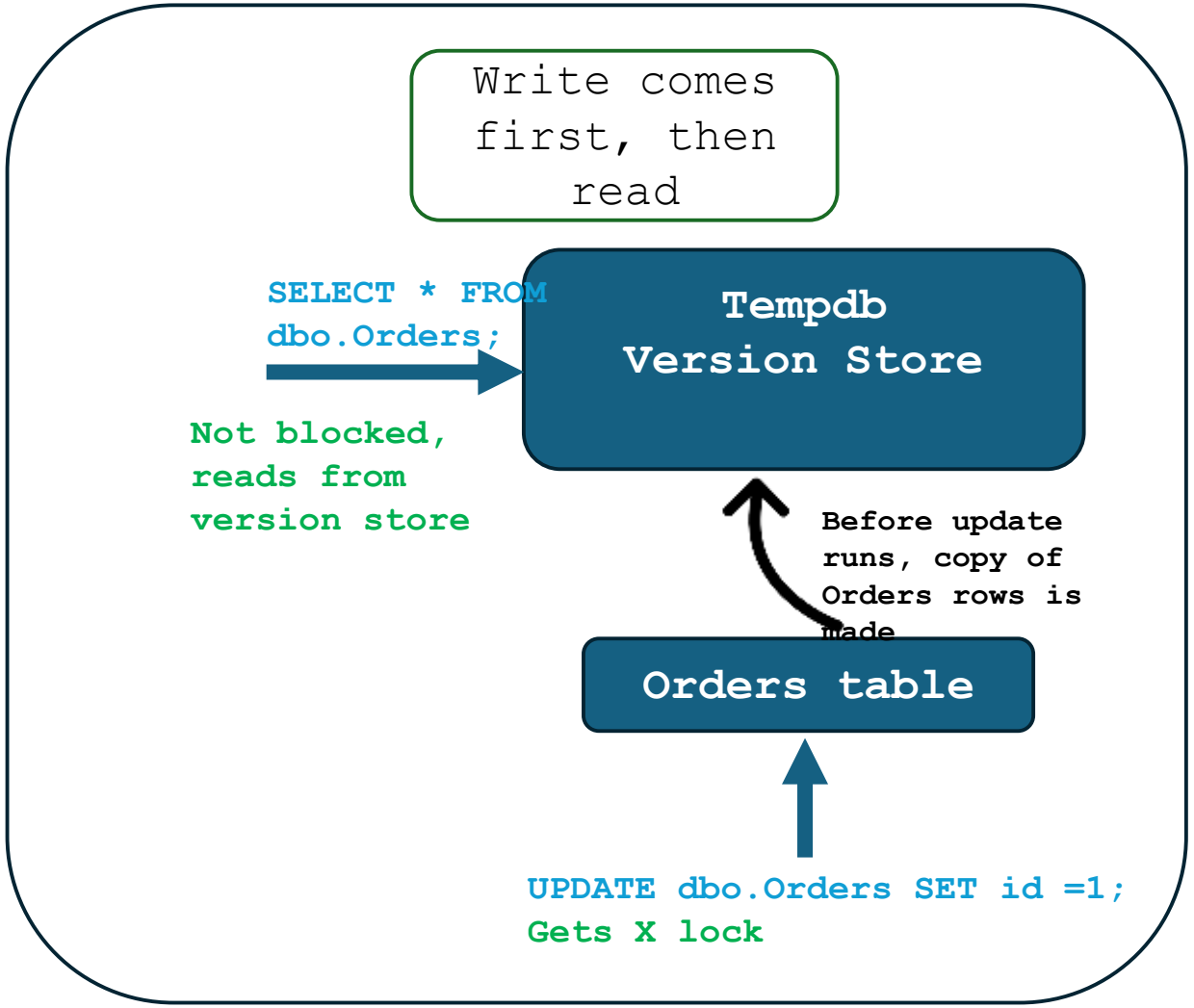
- Optimize queries
- Keep transactions short
- Add indexes but not cause too much overhead
- **Read Committed Snapshot Isolation level**

Read Committed Snapshot Isolation

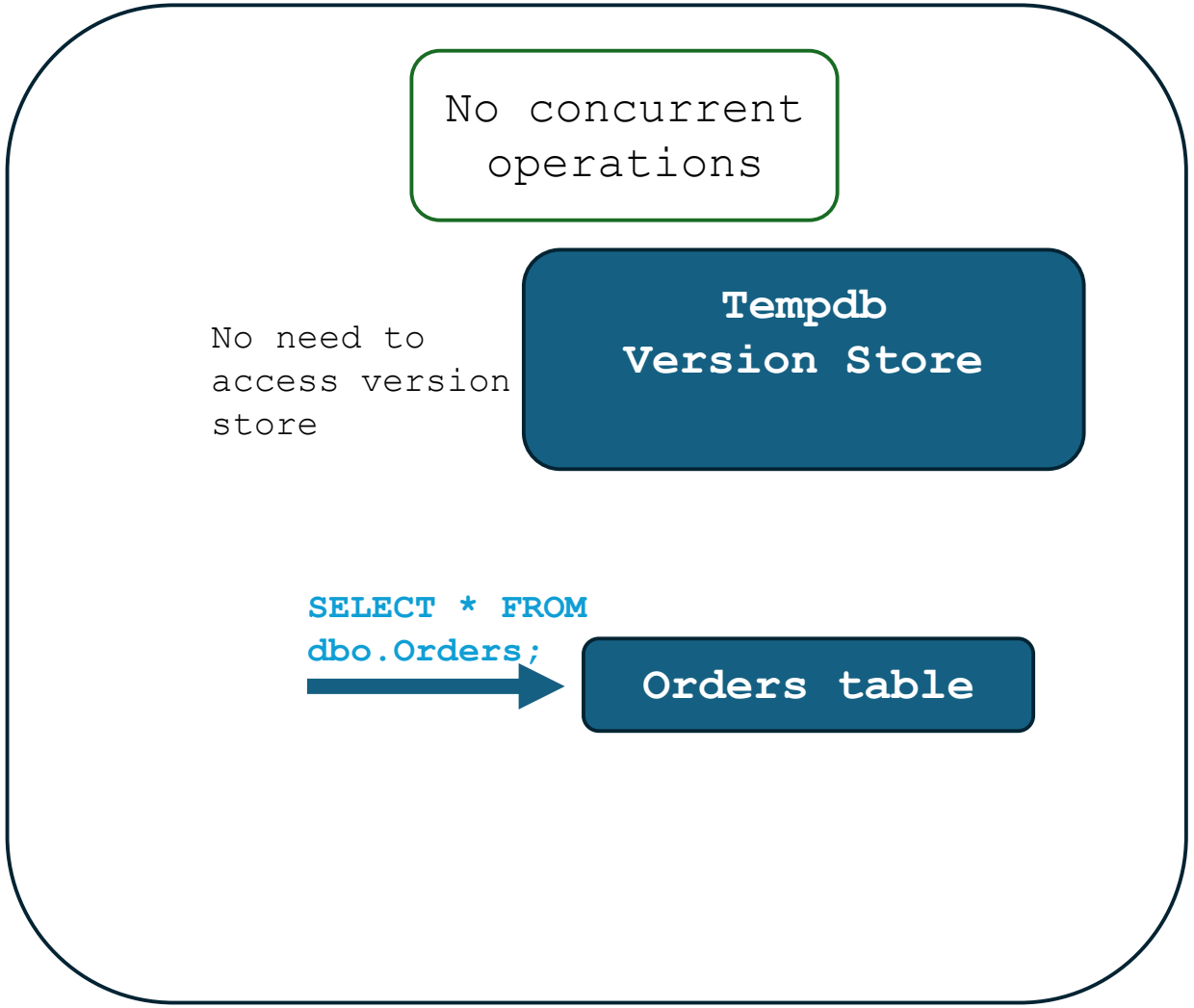
What is Read Committed Snapshot Isolation(RCSI)

- Row versioning-based isolation level
- UPDATE/DELETE operations:
  - Make a copy of **previously committed** data to tempdb
- Read operations
  - No Shared(S) lock
  - Read data from snapshot in TempDB
- Write happens on the original table
- No read/write blocking
- INSERT statements don’t need to be versioned

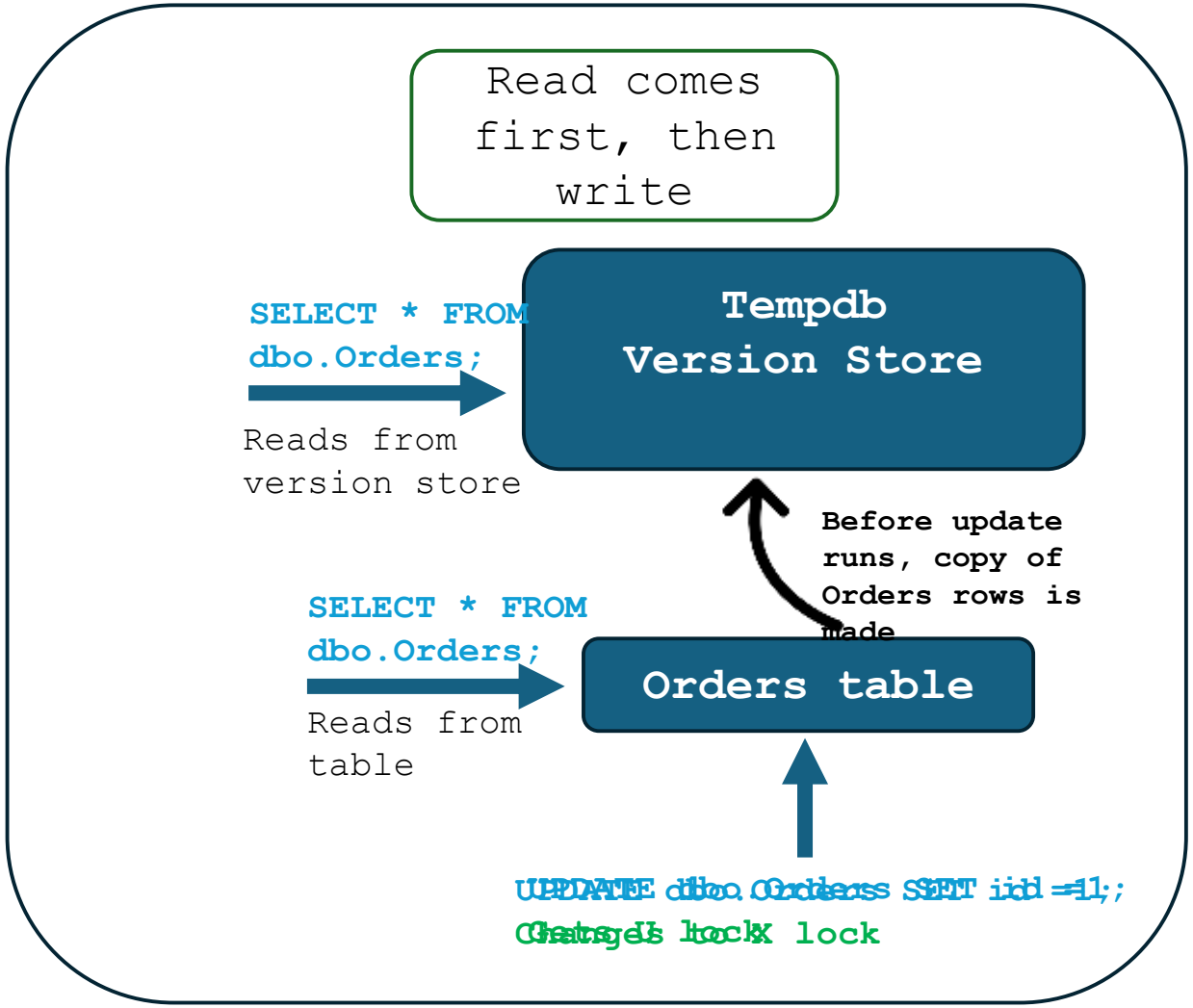
Scenario 1 – Write and Read operation



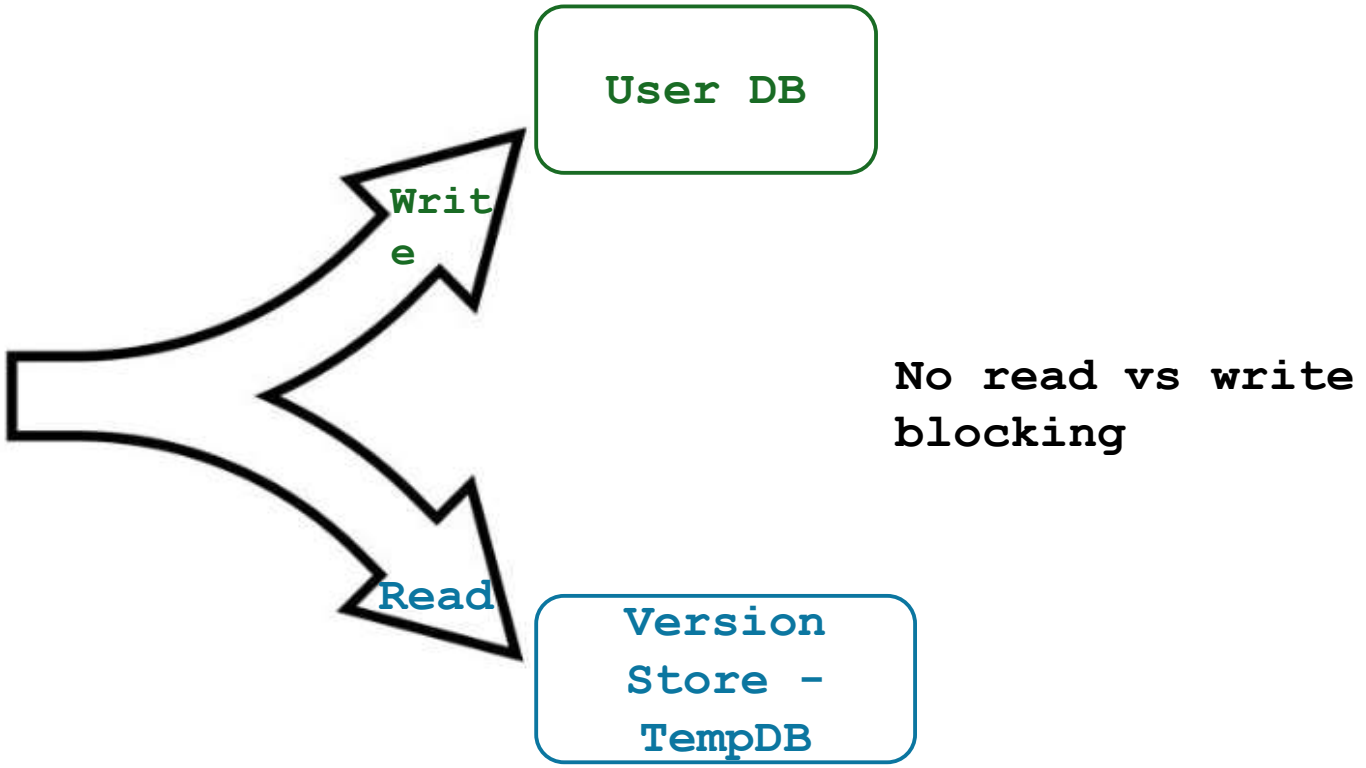
Scenario 2 – Only Read operation



Scenario 3 – Read and Write Operation



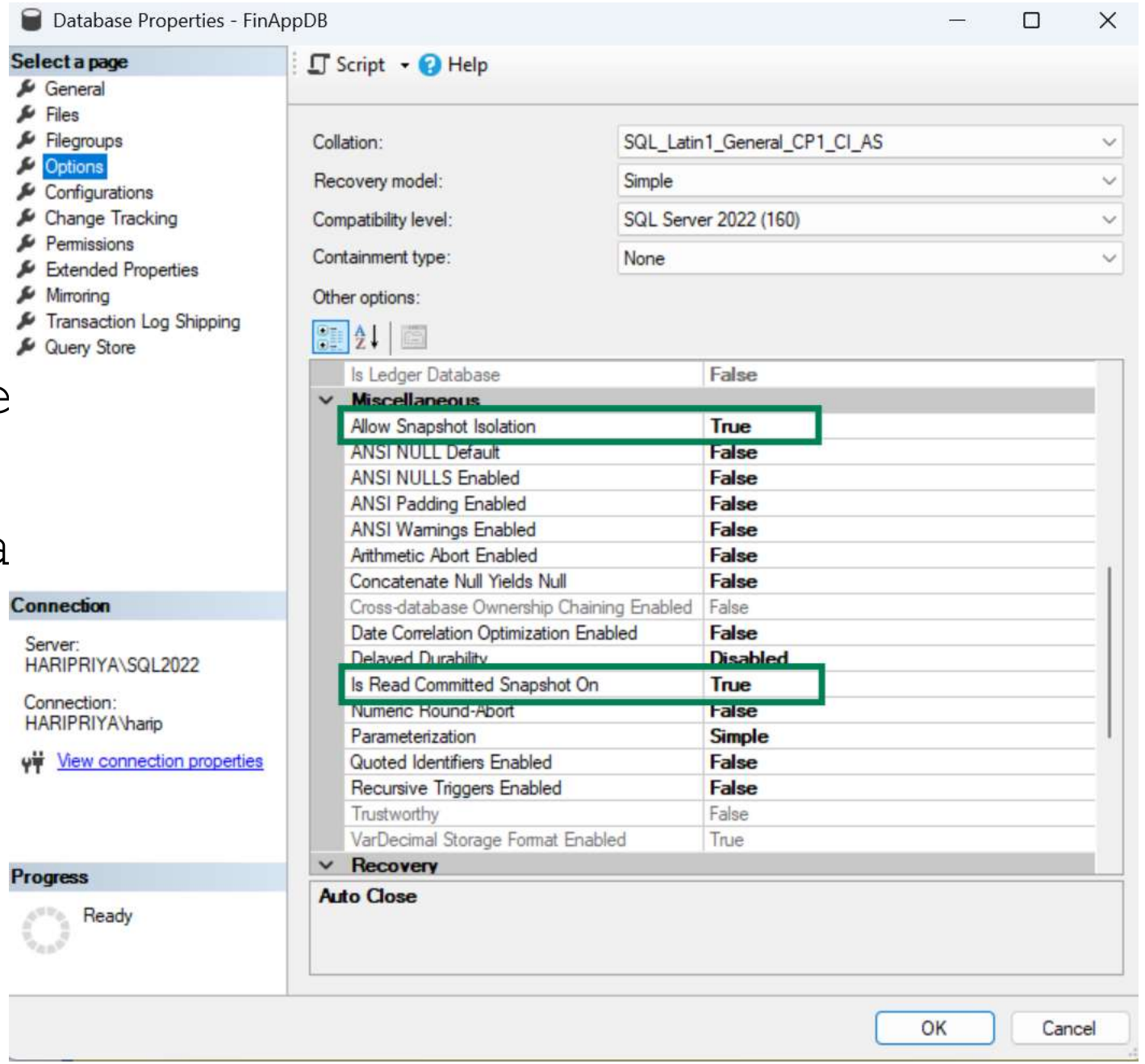
In short..



No read vs write blocking

Enable RCSI

- Default is OFF
- Enable Read Committe Snapshot
- Allow Snapshot Isola optional



Verify RCSI is enabl

```
1  SELECT snapshot_isolation_state,is_read_committed_snapshot_on, name
2  FROM sys.databases
3  WHERE name ='DBATest'
```

100 % No issues found

Results Messages

	snapshot_isolation_state	is_read_committed_snapshot_on	name
1	1	1	DBATEST

```
1  USE DBATEST;
2  DBCC USEROPTIONS
```

100 % No issues found

Results Messages

	Set Option	Value
1	textsize	2147483647
2	language	us_english
3	dateformat	mdy
4	datefirst	7
5	lock_timeout	-1
6	quoted_identifier	SET
7	arithabort	SET
8	ansi_null_dflt_on	SET
9	ansi_warnings	SET
10	ansi_padding	SET
11	ansi_nulls	SET
12	concat_null_yields_null	SET
13	isolation level	read committed snapshot

When to Use RCSI

- **Use RCSI when read/write blocking is the main problem**
- Best for:
  - High-concurrency OLTP servers
  - A lot of read queries reading live data
  - Servers that have frequent read/write blocking
- Optimistic Locking Model or MVCC(Multi Version Concurrency Control)
- Default isolation level in Azure SQL

Who blocks who

- Locking behavior remains the same for write operations
- Blocking still exists between write operations
- Only the locking behavior of read operations change
- Reads don’t block writes
- Writes don’t block reads
- Writes block writes

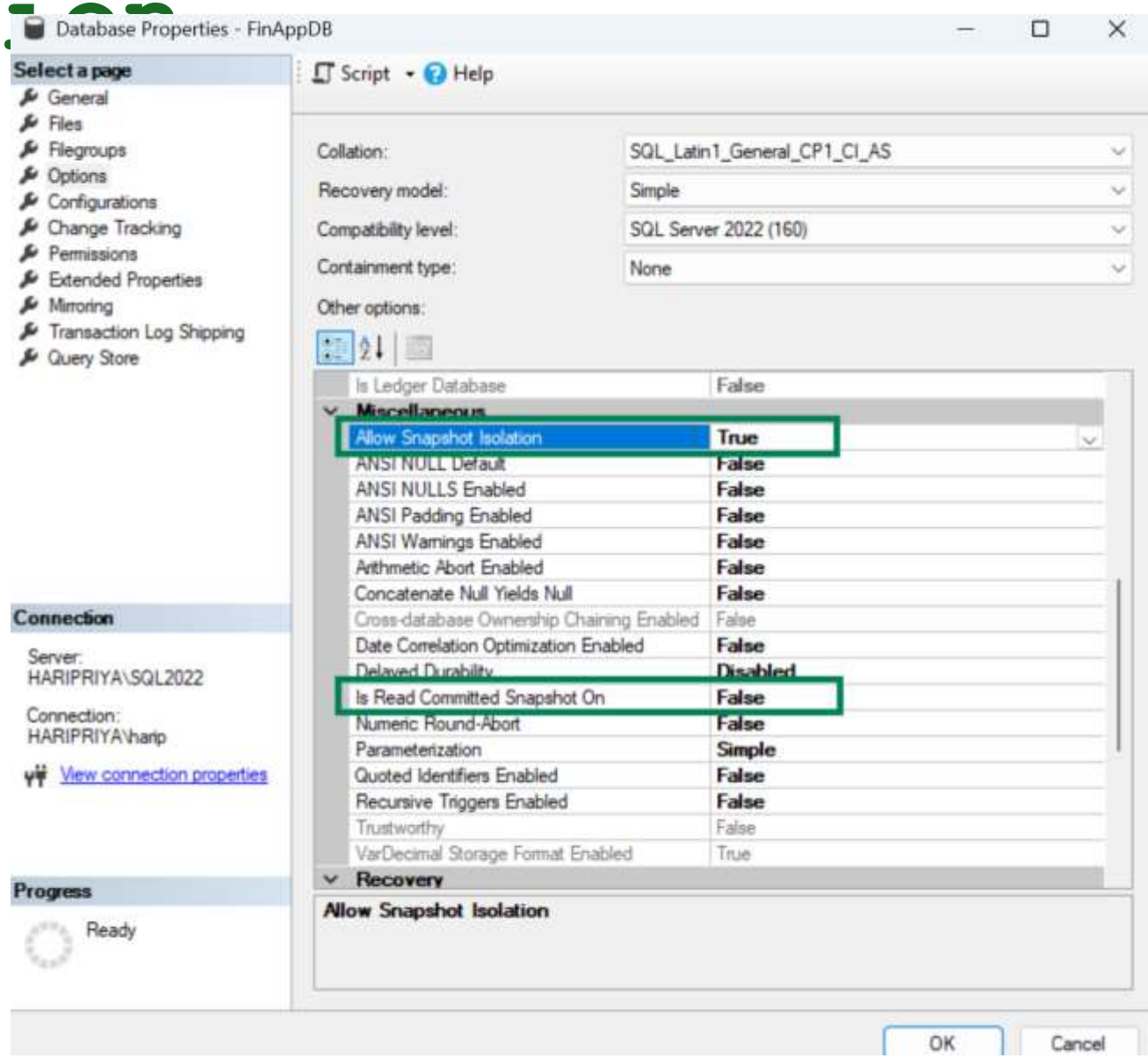
Who blocks who

Read Committed	Read	Write
Read	Doesn't block	Blocks
Write	Blocks	Blocks

Read Committed Snapshot	Read	Write
Read	Doesn't block	Doesn't block
Write	Doesn't block	Blocks

Snapshot Isolation

- Only queries with set transaction isolation level can access the version store
- SET TRANSACTION ISOLATION LEVEL SNAPSHOT;



RCSI vs SI

RCSI	SI
Enable Read Committed Snapshot Isolation option	Enable Allow Snapshot Isolation option
No code changes	Code changes
Automatically read from version store	Set transaction isolation level to read from version store
Statement consistent	Transaction consistent

In simple words

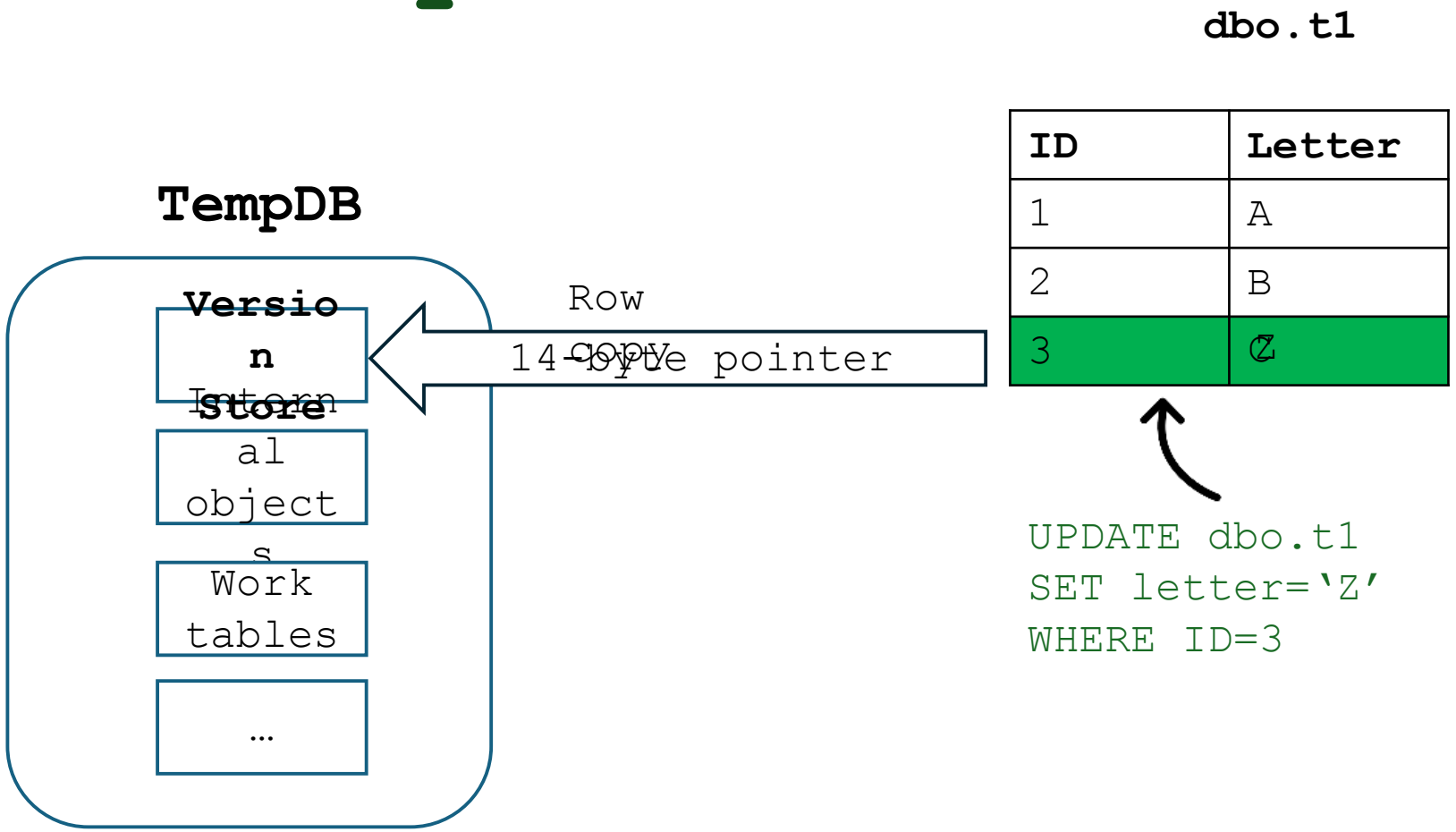
- Read Committed
  - read committed data from the *table*
- Read Committed Snapshot
  - read committed data from the *snapshot*
- Snapshot Isolation
  - read committed data from the *snapshot only when explicitly set*

Version Store

Version Store

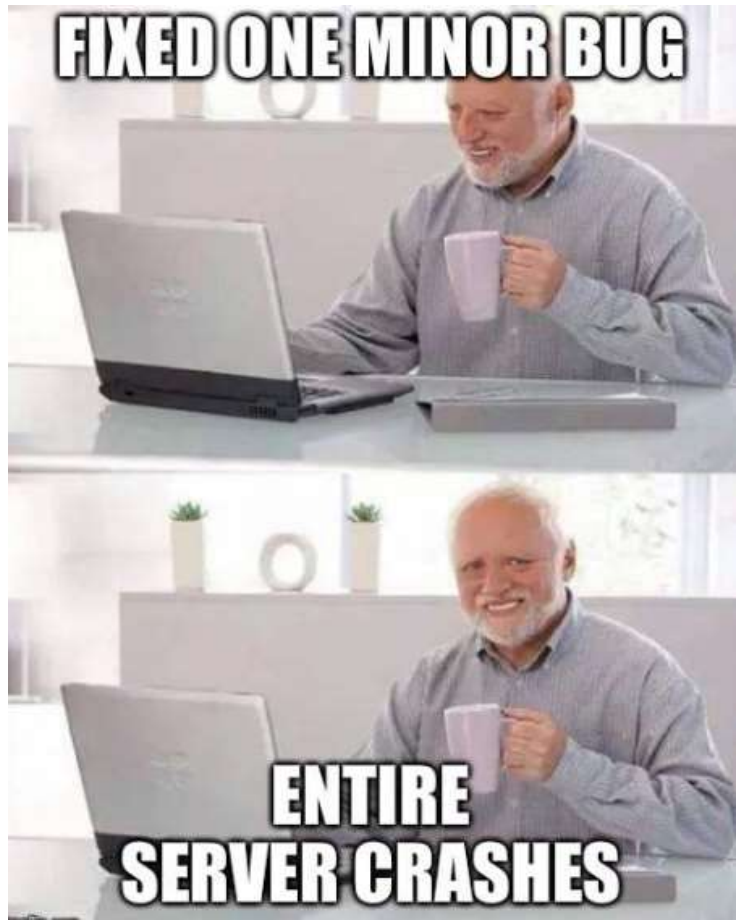
- Previously committed data in tempdb → row version
- Original row gets a 14-byte pointer to row version
- Update/Delete → copy data to version store
- Versions cleaned up when no active transactions need them

How Row Versioning Works Internally



Issue with Version Store

Uncommitted transaction → Tempdb **datafile** space fill up


FIXED ONE MINOR BUG
ENTIRE
SERVER CRASHES

Assume: One-way, single lane road, red car driver is sleeping.

First 3 cars left.

Cars behind red car can’t move.

They can’t be freed up.

Version Store Cle

9:00 Transaction 1 – committed, 2GB - StackOverflow

9:01 Transaction 2 – committed, 2GB - AdventureWorks

9:02 Transaction 3 – committed, 3GB - WorldWideImporters

9:03 Transaction 4 – sleeping , not committed, 2 GB – WorldWideImporters

9:04 Transaction 5 – committed, 2GB - AdventureWorks

9:05 Transaction 6 – committed, 3GB - StackOverflow

9:06 Transaction 7 – committed, 2GB - AdventureWorks

9:07 Transaction 8 – committed, 1GB - WorldWideImporters

……Transaction 25 – committed, 20GB - StackOverflow

……Transaction 70 – committed, 80GB - AdventureWorks

……Transaction 95 – committed, 100GB - AdventureWorks

Cleanup task runs

3 transactions get cleaned up

7GB released

Can't be cleaned up

10GB is held

Still can't be cleaned up

TempDB gets filled up and server becomes inaccessible

Monitoring and Troubleshooting

sys.dm_tran_version_store_space_usage

sys.dm_tran_active_snapshot_database_transactions

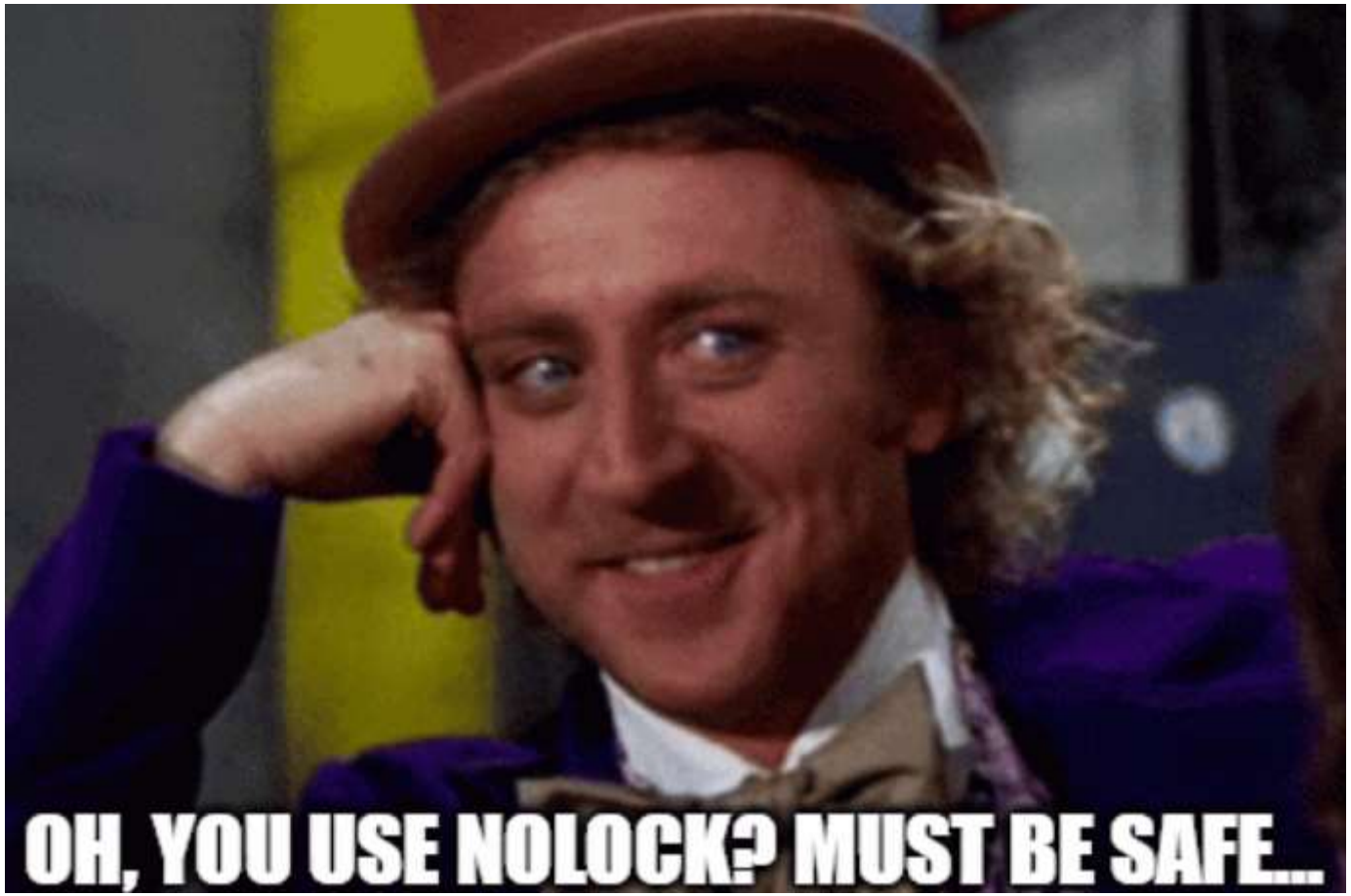
sys.dm_tran_version_store

- Check space used by version store in TempDB
- Setup a job to send notification when it exceeds 50% of TempDB size
- Link to GitHub - https://github.com/haripriyasb/RCSI

Tradeoffs

- Highly volatile systems:
  - Heavy tempdb usage → tempdb size
- Additional CPU and I/O load:
  - Overhead during data modification and retrieval
- Storage
  - 14 bytes increase in row size remains even after records are removed from version store

NOLOCK



Everyone's "Favorite" Hint

```
SELECT * FROM dbo.Table WITH (NOLOCK)
```

- No shared lock
- Dirty reads
- NOLOCK = Read Uncommitted Isolation Level
- NOLOCK table hint overrides RCSI or SI
- No performance benefit

Choose when to use NOLOCK


NOLOCK

- Archived data
- 100% accuracy isn’t required


NOLOCK

- Accuracy is critical
- Financial data
- Critical reports
- OLTP tables

Use RCSI

DEMO

Key Takeaways

- Use RCSI to solve read vs write blocking
- Locking behavior for writes remain the same
- TempDB size increases due to version store build up
- Watch for long open or uncommitted transaction
- Don’t use NOLOCK unless accuracy is not needed

Questions?

gohigh.substack.com

linkedin.com/in/haripriya-naidu1215/

https://github.com/haripriyasb/RCSI

Thank You!