Concurrency in SQL Server 2005

Kalen Delaney
Solid Quality Learning
www.SolidQualityLearning.com

Solid Quality Learning

Kalen Delaney

- Background:
  - MS in Computer Science from UC Berkeley
  - Working exclusively with SQL Server for 18 years
  - Contracted by both Sybase and Microsoft to develop and teach internals courses to Tech Support staff
- Author: *Inside SQL Server 2000* (MS Press, 2000)
- Author: Troubleshooting Locking and Blocking (E-book)
- *SQL Server Magazine* columnist and contributing editor

kalen@SolidQualityLearning.com

Abstract

In this seminar we will look at the methods of concurrency control in SQL Server, with an emphasis given to concurrency enhancements made in SQL Server 2005 based on a technology called row-level versioning (RLV).

RLV provides a new isolation level called Snapshot Isolation that allows readers of data to not be blocked by writers. The seminar will compare the new isolation level with the previous ones and discuss what application changes might be necessary to achieve the level desired.

In addition to new methods of concurrency control, we will discuss the ways that RLV increases the concurrency potential of other features of SQL Server 2005, including online index rebuilds, triggers and multiple active results sets.

Why Study Concurrency?

- Contention Degrades Performance
- Most Tuning Tools Don't Consider Locking Issues
- You Can Write Better Applications

Topics

- Concurrency Models
- Transactions in SQL Server
- Isolation Levels
- Pessimistic Concurrency Control
- Optimistic Concurrency Control
- Features using Row Level Versioning

Concurrency Models

- Pessimistic Concurrency
  - Preventative approach
  - Limited concurrent access
  - Uses locking to avoid conflicts
- Optimistic Concurrency
  - Allows full concurrent access
  - Row versioning allows access to consistent data
  - Conflict detection avoids inconsistent updates

Transactions in SQL Server

- Transaction Basics
- Transaction Control
- Nesting Transactions
- Preventable Phenomena

Transaction Basics

- Allow Correctness of Operations To Be Verified
- Exhibit Four Properties (ACID):
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Transaction Control in TSQL

- Explicit Transaction
  - BEGIN TRANSACTION
  - COMMIT / ROLLBACK TRANSACTION
- Autocommit Transaction
  - Statement Level Implicit Transaction
- Batch-scoped Transaction
  - Batch or Procedure initiated through a client connection with MARS enabled
  - Automatically rolled back if batch or proc ends without COMMIT
- Implicit Transaction
  - SET IMPLICIT_TRANSACTIONS ON
  - sp_configure ‘user options’, 2

Nesting Transactions

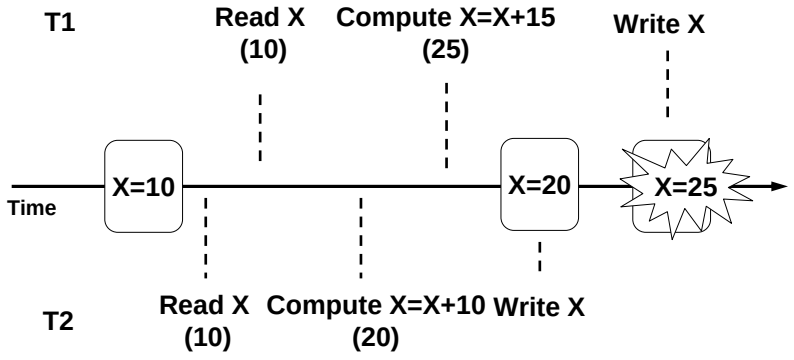
- Nesting is Only Possible Syntactically
  - There is at most ONE open transaction
- Successive BEGIN TRAN Statements Increment @@trancount
- Each COMMIT TRAN Decrements @@trancount
  - When @@trancount reaches 0, COMMIT occurs
- ROLLBACK Sets @@trancount to 0
- Useful for Transaction Control in Stored Procs

Preventable Phenomena
Bad Dependencies

- Lost Updates
- Dirty Reads
- Non-Repeatable Reads
- Phantoms

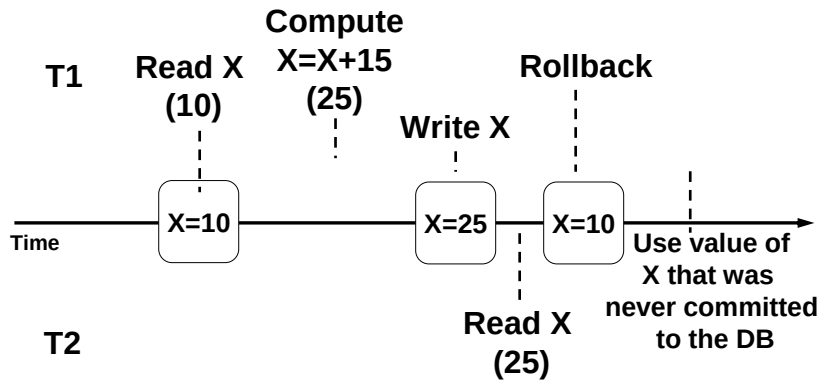
Lost Update

- Classic Unsynchronized Update



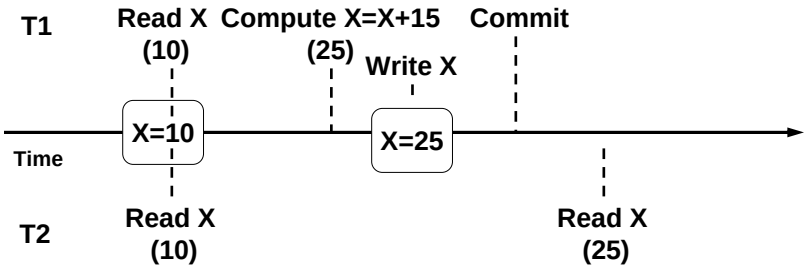
Dirty Read

- T2 Sees T1's Uncommitted Changes



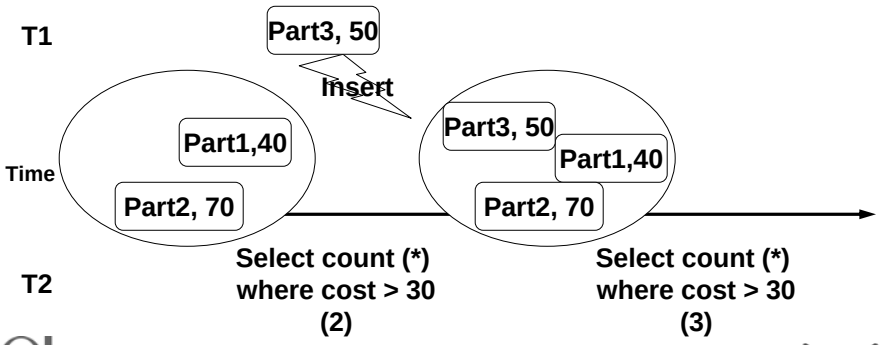
Non-Repeatable Read

- Transaction Sees Only Committed Data
- Data May Change on Subsequent Reads



Phantoms

- Transaction should not see changes in membership of a result set, based on a predicate
- Must prevent others from inserting and deleting



Isolation Levels

- True Isolation Is Expensive
  - Trade off between correctness and concurrency
- ANSI SQL Defines Four Isolation Levels based on which phenomena are allowed
  - No specification as to how to achieve isolation
- SQL Server 2000 implements all ANSI isolation levels using Pessimistic Concurrency Model
- SQL Server 2005 provides Optimistic Concurrency alternatives to 2 isolation levels

SQL 2005 Isolation Levels

Isolation Levels	Phenomena Allowed: Dirty Read	Non-Repeatable Read	Phantoms	Update Conflict	Concurrency Model
Read Uncommitted	Yes	Yes	Yes	No	
Read Committed 1 Locking *2 Snapshot*	 No No	 Yes Yes	 Yes Yes	 No No	**Pessimistic** **Optimistic**
Repeatable Read	No	No	Yes	No	**Pessimistic**
Snapshot	No	No	No	Yes	**Optimistic**
Serializable	No	No	No	No	**Pessimistic**

Pessimistic Concurrency Control

- Aspects of Locking
- Blocking
- Controlling Locking
- Tools for Troubleshooting
- Deadlock

Aspects of Locking

- Type of Lock
- Duration of Lock
- Granularity of Lock

Types of Locks

- Shared Lock
- Exclusive Lock
- Update Lock

Lock Duration

- Duration is Dependent on Owner
- Session Locks (Type = DB)
  - Block DROP, RESTORE
  - Prohibit some status changes
  - Held as long as connection is using DB context
- Transaction Locks
  - Shared locks held until done reading
  - Exclusive locks held until end of transaction
- Cursor Locks
  - Scroll locks held until next FETCH

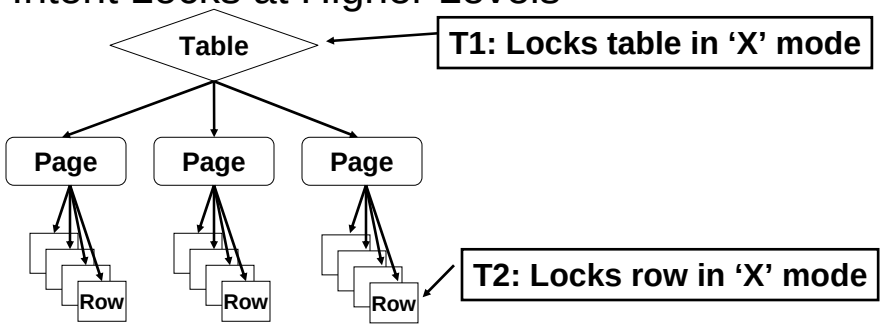
Granularity of Locks: Resources

- Row/Key
- Page
- Table
- Extent
- Database

Concurrency in SQL Server 2005
October 18, 2005

Multi-Granular Locking

- To Lock a Fine Level, SQL Server Places Intent Locks at Higher Levels



Both T1 and T2 update the same row thinking they have it covered by locks -- Result: Disaster

Key Range Locking

To Support Serializable Transactions:

- Lock sets of rows controlled by a predicate
  - WHERE salary between 25000 and 55000
- Need to lock data that doesn't exist!
  - If "where salary between 25000 and 55000" doesn't return any rows the first time, it shouldn't return any on subsequent scans
- Earlier version locked larger units to prevent phantoms
  - Prior to SQL Server 7.0 SQL Server used page and table locks

Key Range Locking Needs Index

- An Key Range Is:

  An half-open interval between values in the **leaf** level of an index
- Data shown below in Index Leaf consists of 7 ranges
  - A key-range lock on a key k_i indicates the range $(k_{i-1}, k_i]$ is locked
  - Nothing can be inserted > k_{i-1} or <= k_i

Index Key	Bookmark or other data
10000	…
20000	…
30000	…
40000	…
50000	…
60000	…

Blocking

- Occurs when one process requests a lock on the same resource help by another process in an incompatible mode
- **Blocking and Isolation Summary for Pessimistic Locking**
  - Writers block writers in all levels
  - Writers block readers in Read Committed and higher
  - Readers block writers in Repeatable Read and higher

Lock Compatibility Matrix

	Lock Mode Already Granted							
Mode Requested	**IS**	**S**	**U**	**IX**	**X**	**Sch S**	**Sch M**	**BU**
IS	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**No**	**No**
S	**Yes**	**Yes**	**Yes**	**No**	**No**	**Yes**	**No**	**No**
U	**Yes**	**Yes**	**No**	**No**	**No**	**Yes**	**No**	**No**
IX	**Yes**	**No**	**No**	**Yes**	**No**	**Yes**	**No**	**No**
X	**No**	**No**	**No**	**No**	**No**	**Yes**	**No**	**No**
SchS	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**Yes**
SchM	**No**	**No**	**No**	**No**	**No**	**No**	**No**	**No**
BU	**No**	**No**	**No**	**No**	**No**	**Yes**	**No**	**Yes**

Controlling Locking

- Lock Hints
- Lock Timeout
- Index Options
- Isolation Level
- Transaction Mode

Lock Hints

- Unit
- Duration
- Type
- READPAST
- Must use WITH keyword in SQL Server 2005

Lock Timeout

- Value Set in Milliseconds
- Transaction does NOT rollback
  - Except in SQL Server 2000, sp1
- Handle error
  - Use SET XACT_ABORT ON
  - Check for error 1222

Index Options

- sp_indexoption
  - AllowRowLocks
  - AllowPageLocks
  - DisAllowRowLocks
  - DisAllowPageLocks
- Cannot Disable TABLE Locks!
- Check Status With INDEXPROPERTY( )
  - INDEXPROPERTY (table_ID , index , property)

Isolation Level

- SET option for a connection
- Table hints override connection setting

- Tradeoffs:

  Higher Isolation Level provides more consistency and less throughput

Transaction Mode

- Implicit Transaction Mode is ANSI Compliant
- No BEGIN TRAN Used
- Must Always COMMIT or ROLLBACK

```
SET IMPLICIT_TRANSACTIONS ON
```


Tools for Troubleshooting

- SQL Server Profiler
- Performance Monitor
- SQL Server 2000 pseudo-system tables
  - Syslockinfo (interpret with sp_lock)
  - Sysprocesses (interpret with sp_who)
- SQL Server 2005 Dynamic Management Objects
  - sys.dm_tran_locks
- Summary Reports through Management Studio

SQL Server 2005 Profiler

- Deadlock Graph Event Class
- Lock:Acquired Event Class
- Lock:Cancel Event Class
- Lock:Deadlock Chain Event Class
- Lock:Deadlock Event Class
- Lock:Escalation Event Class
- Lock:Released Event Class
- Lock:Timeout (timeout > 0) Event Class
- Lock:Timeout Event Class

SQL Server 2005 Performance Monitor

- Counters
  - Number of Deadlocks/sec
  - Average Wait Time (ms)
  - Lock Wait Time (ms)
  - Lock Requests/sec
  - Lock Waits/sec
  - Lock Timeouts/sec
  - Lock Timeouts (timeout > 0)/sec
  - Average Wait Time Base
- Access programmatically using **sys.dm_os_performance_counters**
- Instances
  - Heap/BTree
  - Database
  - Object
  - AllocUnit
  - Metadata
  - RID
  - Page
  - Application
  - Key
  - File
  - Extent

Dynamic Management Objects

- Use sys.dm_tran_locks with sys.dm_os_waiting_tasks

```
-- This query will show blocking information.
SELECT resource_type, resource_database_id,
    resource_associated_entity_id, request_mode,
    request_session_id, blocking_session_id
FROM sys.dm_tran_locks as t1,
     sys.dm_os_waiting_tasks as t2
WHERE t1.lock_owner_address =
  t2.resource_address
```


Summary Reports

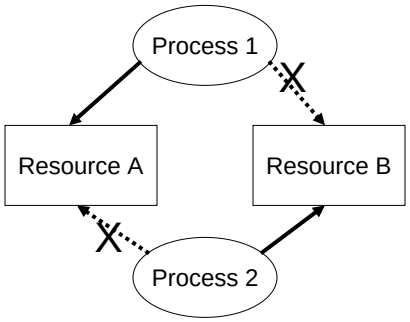
- Extensive set of reports available through Management Studio
- Based on Dynamic Management Objects and default lightweight trace
  - All transactions
  - All blocking transactions
  - Top transactions by age
  - Top transactions by blocked transactions count
  - Top transactions by locks count
  - Resource locking statistics by object

Deadlock

- What is deadlock?
- Handling deadlock
- Deadlock Avoidance
- Tracing Deadlock

What is Deadlock?

- Two Processes Mutually Blocking Each Other
- Resource is usually data
  - Table, page, row

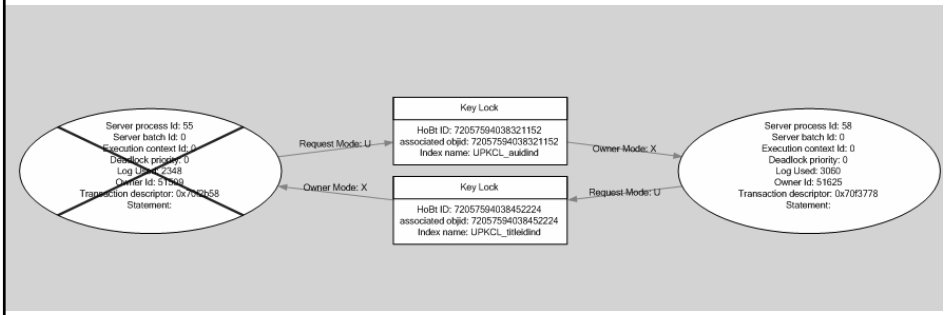


Handling Deadlock

- SQL Server automatically detects deadlock
  - Checks for cycles at regular intervals
  - Checks more often if there are frequent deadlocks
- Process with Cheapest Transaction is Chosen as Victim
  - Transaction rolled back
  - Error message 1205
- Developer Must Check For 1205
  - Pause briefly
  - Resubmit
  - Keep track of deadlock frequency
  - Occasional deadlocks are not a major problem
- Some Distributed Deadlocks Cannot Be Detected

Demo: Tracing Deadlocks in SQL Server Profiler

- New Trace Event: Deadlock XML
  - Save results to file
  - One file per deadlock, or all in one file



Troubleshooting Suggestions

- Analyze Transaction Management
- Minimize Blocking
  - Consider Optimistic Concurrency
- Isolate and Tune Long Running Queries

How to Avoid Blocking

- Keep transactions short and in one batch
- No user interaction during transactions
- Rollback when canceling; Rollback on any error or timeout
- Proper indexing – Index Tuning Wizard or index analysis
- Beware of implicit transactions
- Process results quickly and completely
- Reduce isolation level to lowest possible
- Stress test at maximum projected user load before deployment
- Other possibilities to consider:
  - Locking hint, Index hint, Join hint

Deadlock Avoidance

- Minimize Blocking
- In-order Resource Access
- Bound Connections (will be deprecated)
- MARS
- Add and/or Remove Indexes to Provide or Remove Alternate Access Path to the Desired Resource
- SET DEADLOCK_PRIORITY LOW

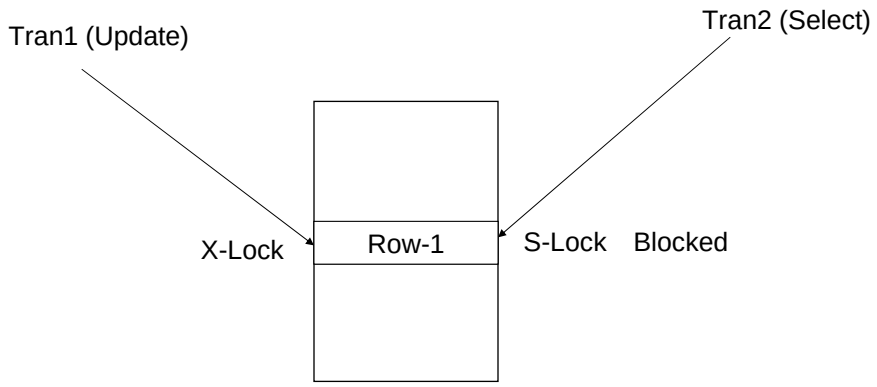
Summary of Pessimistic Concurrency

- Aspects of Locking
- Blocking
- Controlling Locking
- Tools for Troubleshooting
- Deadlock

Optimistic Concurrency Control

- Problems Addressed by Optimistic Concurrency
- Snapshot Based Isolation Levels
- Managing Snapshot Isolation
- Monitoring and Troubleshooting
- Migration to SQL2005 Snapshot
- Summary

Problems Addressed by Optimistic Concurrency: Reader Writer Blocking



Snapshot Based Isolations

- Two Varieties of Snapshot Isolation
- Snapshot Isolation
  - A new value for SET TRANSACTION ISOLATION LEVEL
  - Logically, another solution to achieving serializable isolation, allowing none of the bad dependencies
- Read Committed Snapshot
  - Implements read committed using optimistic concurrency
  - No code changes required
- Increased Concurrency for read/write applications
  - Allows non-blocking consistent reads in an online transaction processing (OLTP) environment
  - Writers do not block readers; readers do not block writers
  - Consistency of aggregates without using higher isolation levels
    - AVG, COUNT, SUM, etc.

Read Committed Snapshot

- Statement level snapshot isolation
- New “flavor” of read committed
- Enable/disable with ALTER DATABASE
- Readers see committed values as of start of statement
  - Writers do not block Readers
  - Readers do not block Writers
  - **Writers DO block writers**
- Can greatly reduce locking / deadlocking without changing applications

Example: Read Committed Snapshot

CREATE TABLE t1 (c1 int unique, c2 int)
INSERT INTO t1 VALUES (1, 5)

Transaction 1	Transaction 2 (Read Committed Snapshot)
BEGIN TRAN	
UPDATE t1 SET c2 = 9 WHERE c1 = 1	
	BEGIN TRAN
	SELECT c2 FROM t1 WHERE c1 = 1 -- SQL Server returns 5
	SELECT c2 FROM t1 WHERE c1 = 1 -- SQL Server returns **9**
COMMIT TRAN	
	COMMIT TRAN

Time

Read Committed Snapshot …

- No behavior change for Update, Delete or Inserts
  - Acquire locks
  - Data read based on locking read-committed
- New READCOMMITTEDLOCK hint
  - Directive for the query scan to run under the original flavor of locking-based read committed isolation

Snapshot Isolation

- Transaction level snapshot isolation
- New option to SET TRANSACTION ISOLATION LEVEL
  - First must enable with ALTER DATABASE
- Transactional consistent database as of the beginning of the transaction
- Readers do not lock data
- Higher concurrency, Fewer deadlocks

Example 1: Snapshot Isolation (Read)

```
CREATE TABLE t1 (c1 int, c2 int)
INSERT INTO t1 VALUES (1, 5)
```

Transaction 1

```
BEGIN TRAN
UPDATE t1
SET c2 = 9
   WHERE c1 =1

COMMIT TRAN
```

Transaction 2 (Snapshot Isolation)

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
BEGIN TRAN
SELECT c2 FROM t1 WHERE c1 = 1
    -- SQL Server returns 5
SELECT c2 FROM t1 WHERE c1 = 1
 -- SQL Server returns  5
COMMIT TRAN
 SELECT c2 FROM t1 WHERE c1 = 1
    -- SQL Server returns  9
```

Example 2: Snapshot Isolation (Update)

```
CREATE TABLE t1 ( c1 int, c2 int)
INSERT INTO t1 VALUES (1,5)
```

Transaction 2	Transaction 1 (Snapshot Isolation)
	SET TRANSACTION ISOLATION LEVEL SNAPSHOT
	BEGIN TRAN
BEGIN TRAN	SELECT c2 FROM t1 WHERE c1 = 1
UPDATE t1	-- SQL Server returns 5
SET c2 = 9	UPDATE t1 ← BLOCKED
WHERE c1 = 1	SET c2 = 15 WHERE c1 = 1
COMMIT TRAN	↑ Transaction rollback due to update Conflict

Conflict Detection

- Occurs only for Snapshot Isolation
  - Not for Read Committed Snapshot
- Occurs when the data to be changed by a snapshot transaction has been changed by a concurrent transaction
- Detection occurs automatically
- Automatic rollback of a snapshot transaction
- Conflict detection prevents the “lost update” problem

Minimizing Update Conflict

- Update lock hint in SELECT
  - UPDLOCK reduces chance of update conflict in a snapshot transaction
  - No other connections cannot change data locked with UPDLOCK
- Change isolation level
  - Evaluate if pessimistic concurrency will suffice
  - Consider Read Committed Snapshot instead

Example: Use locking hints

```
CREATE TABLE t1 ( c1 int, c2 int)
INSERT INTO t1 VALUES (1,5)
```

Transaction 1		Transaction 2 (Snapshot Isolation)
		SET TRANSACTION ISOLATION LEVEL SNAPSHOT
		BEGIN TRAN
		SELECT c2 FROM t1 with (UPDLOCK) WHERE c1 = 1
BEGIN TRAN		
UPDATE t1	BLOCKED	-- SQL Server returns 5
SET c2 = 9		
WHERE c1 = 1		UPDATE t1 SET c2 = 15 WHERE c1 = 1
		COMMIT

SQL 2005 Isolation Levels

Isolation Levels	Phenomena Allowed: Dirty Read	Non-Repeatable Read	Phantoms	Update Conflict	Concurrency Model
Read Uncommitted	Yes	Yes	Yes	No	
Read Committed 1 Locking *2 Snapshot*	No No	Yes Yes	Yes Yes	No No	**Pessimistic** **Optimistic**
Repeatable Read	No	No	Yes	No	**Pessimistic**
Snapshot	No	No	No	Yes	**Optimistic**
Serializable	No	No	No	No	**Pessimistic**

Snapshot Isolation: Not Serializable

SET TRANSACTION ISOLATION LEVEL SNAPSHOT

Transaction 1
BEGIN TRAN
Read (Row-1)

Transaction-2
BEGIN TRAN
Read (Row-2)

Write (Row-2)
COMMIT

Write (Row-1)
COMMIT

No Serialized execution possible

Snapshot Isolation and DDL

- Metadata is not versioned
- Two Cases
  - DDL inside Snapshot Isolation
  - Concurrent DDL outside Snapshot Isolation

DDL inside Snapshot Isolation

- Some DDL statements are disallowed
  - CREATE [XML] / ALTER / DROP INDEX
  - DBCC DBREINDEX
  - ALTER TABLE
  - ALTER PARTITION FUNCTION / SCHEME
- Some DDL statements are allowed
  - CREATE TABLE
  - CREATE TYPE
  - CREATE PROC

Example: DDL inside Snapshot Isolation

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
Transaction 1
BEGIN TRAN
SELECT count(*) FROM t1
..
CREATE TABLE t_new (c1 int)
..
                                        Transaction 2
                                        BEGIN TRAN
                                        INSERT t1 VALUES (<row>)
                                        COMMIT
CREATE CLUSTERED INDEX t1(c1)
```

← ERROR

Snapshot Isolation and Concurrent DDL

- Fails when concurrent DDL statement changes an object referenced in Snapshot Isolation
- Only concurrent DDL allowed is CREATE STATISTICS
- Querying catalog views under snapshot isolation may fail due to concurrent DDL

Example: DDL outside Snapshot Isolation

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
BEGIN TRAN
SELECT * FROM t1

                                        BEGIN TRAN
                                        ALTER TABLE t2 ….
                                        COMMIT

SELECT * FROM t1
                                        ALTER TABLE t1 ADD COLUMN c3

SELECT * FROM  t1
```

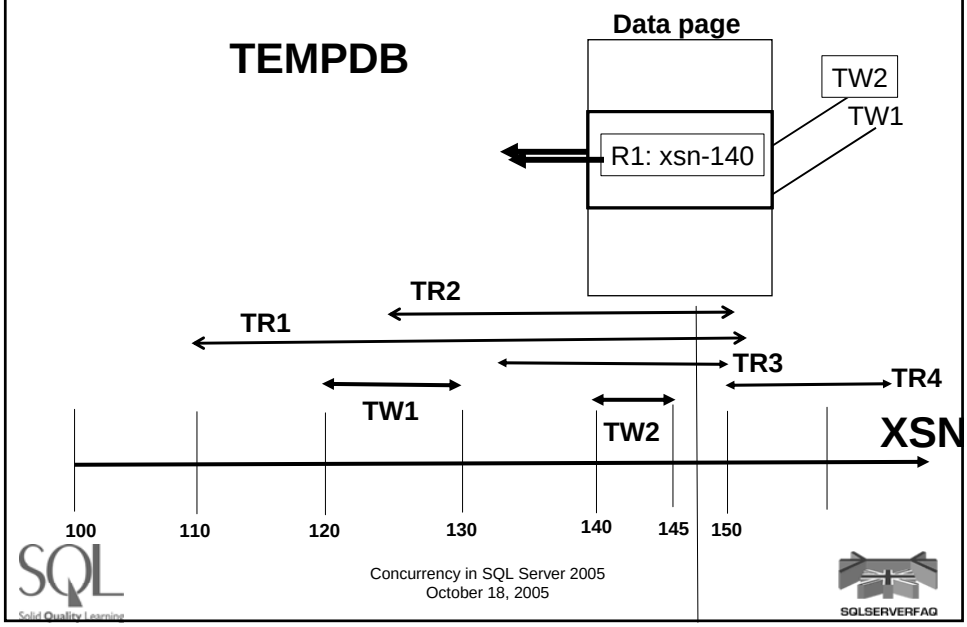
ERROR

Snapshot Isolation Details

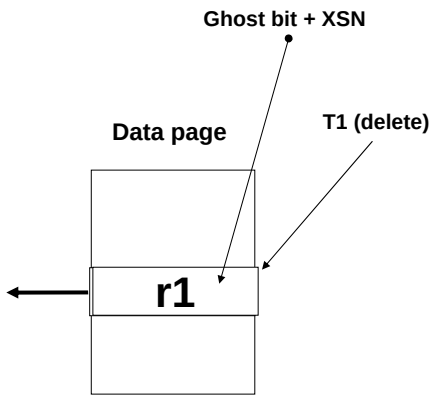
- Version Chains
- Enabling and Disabling Snapshot
- Setting up Tempdb

Concurrency in SQL Server 2005
October 18, 2005

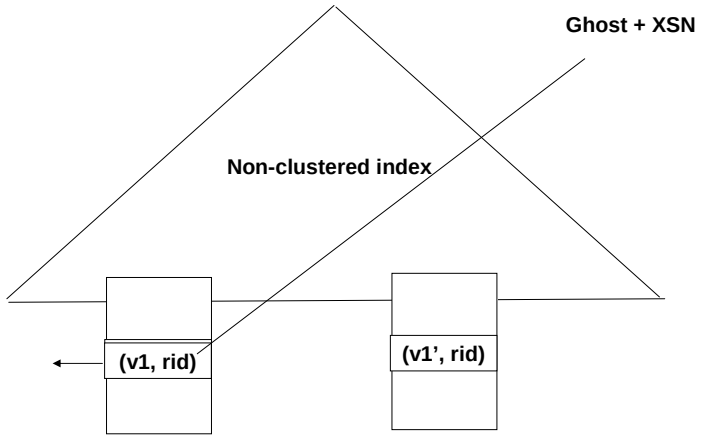
Version Chains



Delete Operation



Btree updates



Version Store

- Versions Maintenance
  - Versions kept in tempdb
  - Background thread (once every minute) does the garbage collection of stale versions
  - A long running transaction can cause tempdb to become full.
- Update/Delete operations generate versions
- Insert does not generate a version
- Snapshot based queries retrieve consistent data by traversing version chains

Enabling Snapshot Based Isolation Levels

- Enable both kinds at database level
- Enable snapshot isolation:

```
ALTER DATABASE mydatabase
    SET ALLOW_SNAPSHOT_ISOLATION ON
```

  - Also requires SET option for connection
  - Change may be deferred
- Enable RCSI

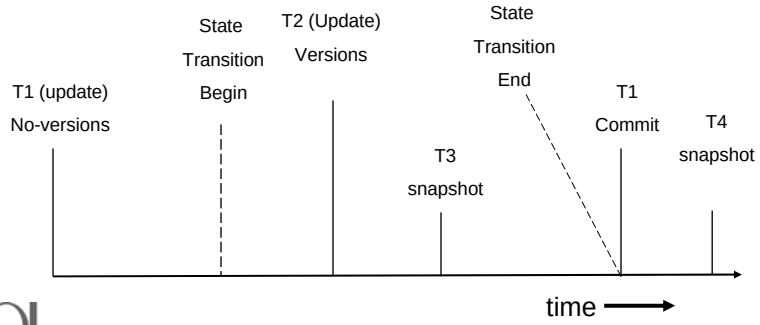
```
ALTER DATABASE mydatabase
    SET READ_COMMITTED_SNAPSHOT ON
```

  - All read committed operations will use RCSI
  - No other changes required
  - Will block if database is in use

Example: Enabling Snapshot Isolation

Transition Phase

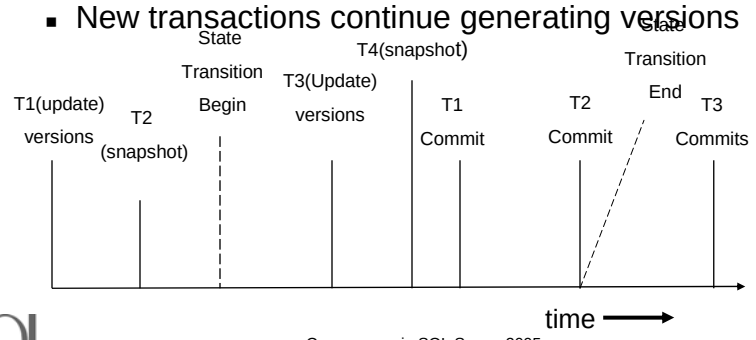
- Waits for the completion of all active transactions
- New update transactions start generating versions
- New Snapshot transactions cannot start



Example: Disabling Snapshot Isolation

Transition Phase

- New snapshot transactions cannot start
- Existing snapshot transactions still execute snapshot scans
- New transactions continue generating versions



Metadata for Snapshot Isolation State

```
SELECT snapshot_isolation_state_desc,
          is_read_committed_snapshot_on
FROM      sys.databases
WHERE    name= 'mydatabase'
```

Values for `snapshot_isolation_state_desc`:

OFF, PENDING_OFF, ON, PENDING_ON

Values for `is_read_committed_snapshot_on`:

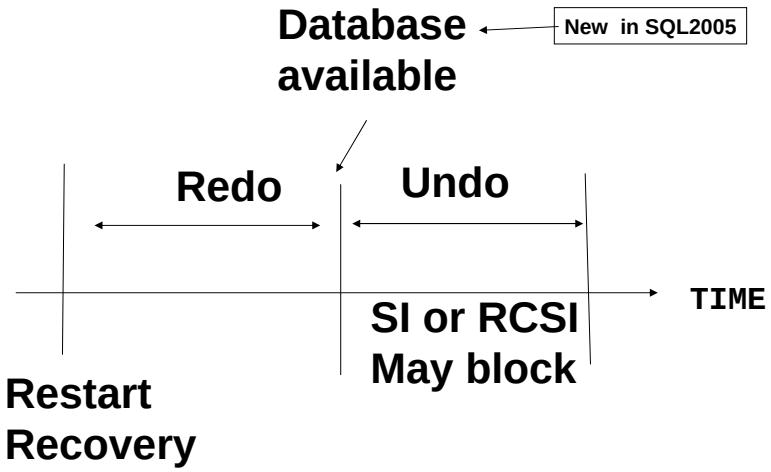
0, 1

Setting up TempDB

- Version store resides in tempdb
  - Cannot reserve space exclusively for version store
  - All databases share the version store
- Estimate Versioning Space

```
Space Required =
   (Version store data generated per minute) *
   (Longest running time of any transaction)
```


Recovery and Snapshot Based Isolation Levels



Troubleshooting Snapshot Isolation

- Running out of tempdb space
- Too many update conflicts
- SQL Server's performance is impacted

Running Out of Tempdb space

- Check sizing of tempdb
- Monitor Perfmon Counters
  (SQL Server: Transactions Object)
  - Free Space in tempdb (KB)
  - Version Store Size (KB)
  - Version Generation rate (KB/sec)
  - Version Store clean up rate (KB/sec)
  - Longest Transaction Running Time
- Corrective Action
  - Identify badly behaving transactions
    - Sys.dm_tran_active_snapshot_database_transactions
    - Sys.dm_tran_transactions_snapshot
  - Kill transaction

Concurrency in SQL Server 2005
October 18, 2005

Too Many Update Conflicts

- Possible cause
  - Improper use of Snapshot Isolation
- Monitor PerfMon counters:
  Update Conflict Ratio
- Corrective Actions
  - Break Snapshot Transaction into smaller ones
  - Use locking hints

SQL Server Performance Problems

- Possible Causes
  - Your application has very little Reader/Writer blocking
  - I/O bottleneck in tempdb
- Use Perfmon counters and DMVs
  - Find if there is significant blocking (Reader/Writer)
  - Tempdb bottleneck by monitoring disk seconds/read and write, Read/write queue lengths
- Corrective Actions
  - Turn off Snapshot Isolation, if not needed
  - Move tempdb to faster disk
  - Break tempdb into multiple files

Troubleshooting Tools

- Lock Hints
- Performance Monitor
- Dynamic Management Objects

Locking Hints for RCSI

- READCOMMITTED

  Rows are not locked (default for RCSI)
- READCOMMITTEDLOCK

  Allows both locking and nonlocking behavior, this forces locks to be acquired
- UPDLOCK, XLOCK

  If the statement is reading from a table with an update lock hint, then regular locks are acquired

Locking Hints for Snapshot Isolation

- UPDLOCK
  - Guarantees that the rows returned will not have conflict problem when they are updated later
- READPAST
  - Disallowed inside SERIALIZABLE transactions and SNAPSHOT transactions
- READCOMMITTED and READCOMMITTEDLOCK are identical
  - Both will take locks as with pessimistic concurrency

Performance Monitor

- SQL Server: Transactions Object
- Look at SQL Server: Locks Object
  - Lock Wait Time (ms)
  - Average Lock Wait Time (ms)
- SQL Server: Wait Statistics
  - Lock Waits
- SQL Server: Databases (for tempdb)

Dynamic Management Objects

- Sys.dm_tran_active_snapshot_database_transactions (view)
  Returns virtual table for all active transactions in all snapshot-enabled databases under the SQL Server instance.
  - Find the top 10 earliest transactions

```
SELECT TOP 10 transaction_id, name FROM
  sys.dm_tran_active_snapshot_database_transactions
  ORDER BY transaction_id
```

  - Find the transaction that has traversed the longest version chain:

```
SELECT TOP 1 * FROM
  sys.dm_tran_active_snapshot_database_transactions
  ORDER BY max_version_chain_traversed
```

- sys.dm_tran_transactions_snapshot (function)
  Returns a virtual table for the sequence_number of transactions that are active when each snapshot transaction starts

Inspecting Version Store

- sys.dm_tran_version_store
  - Returns a virtual table that displays all version records in version store
  - Can be inefficient to run as it queries the entire version store
- Determine size of version store

```
SELECT count(*) AS NumRows,
  (sum(record_length_first_part_in_bytes)  +
  sum(record_length_second_part_in_bytes))/8060.
  AS Version_store_Pages
   FROM  sys.dm_tran_version_store
```

Migrating to SQL Server 2005

- From SQL2000 and earlier
- From Oracle

Migration from SQL Server

- Determine if your application will benefit from versioning
- Enable database for read-committed snapshot
- Determine your tempdb space requirements
- Application start benefiting with no further changes
- Storage considerations
  - 14 bytes overhead for each row to keep versioning information
- Test to determine if RCSI is sufficient
  - Application changes required if you need Snapshot Isolation

Migration from Oracle

- Oracle isolation levels map directly to Snapshot Based isolations.

Oracle	SQL Server
Read Committed (default)	Read-Committed-Snapshot Isolation
Serializable	Snapshot Isolation (Automatic Conflict Detection)
Read Only	Snapshot Isolation

Summary of Optimistic Concurrency

- Increased throughput by reducing Reader/Writer Blocking
- Point in time consistency
- Easy to Enable
  - No Application changes needed.
- New Perfmon counters/DMVs for easy supportability

Choosing a Concurrency Model

- Determine what consistency means for your apps
- Pessimistic concurrency control
  - Uses fewest resources
  - High contention for data
  - Appropriate when the cost of protecting data with locks is less than the cost of rolling back transactions if concurrency conflicts occur.
  - Well designed applications can manage short blocking times
- Optimistic concurrency control
  - Allowing for optimistic concurrency uses resources even if SI never invoked
  - RCSI is cheaper than SI
  - Appropriate in environments where there is low contention for data, and where the cost of occasionally rolling back a transaction outweighs the costs of locking data when read.

Features using Row Level Versioning

- Built on Versioning Infrastructure
  - Snapshot isolation does not need to be enabled
- Online Index Operations
- Triggers
- Multiple Active Result Sets (MARS)

Online Index Operations

- Indexes Can Be Created or Rebuilt Online
  - ONLINE cannot be set to ON if the underlying table contains a LOB data type
  - Disabled clustered indexes cannot be created, rebuilt, or dropped with the online option set to ON.
- Table and NC Index Data are Available During DDL

  For both modification and querying
- Normal Offline Index DDL Will Hold X Lock
  - No modifications or queries until the index operation is complete.

How Do Online Index Creates Work?

- Four Types of Structures Used
- Source Index and Pre-existing (NC) Indexes
  - All preexisting indexes are available for queries
  - Partitioning and parallelism are supported in online index operations
- Target Index
  - User insert, update, and delete operations to the source are applied to the target during the index operation.
  - The new index is not searched while processing SELECT statements until the index operation is committed.
  - Internally, the index is marked as write-only.
- Temporary Mapping Index
  - Used by concurrent transactions to determine which records to delete in the new indexes being built
  - Concurrent transactions also maintain the temporary mapping index in all insert, update, delete operations

Initiating Index Rebuild

Syntax:

```
ALTER INDEX [INDEX_NAME | ALL] ON
  <schema.object>
                REBUILD WITH (ONLINE = ON)
```

- Progress Report trace event monitors progress

Phase 1 of Online Index Create: Preparation

- Source
  - System meta data preparation to create the new index structure.
  - Version scan of original index starts
  - Concurrent write operations on the source are blocked for a very short period with Sch-S lock
- Target
  - New index is created and set to write-only.

Phase 2 of Online Index Create: Build

- Source
  - The data is scanned, sorted, merged, and inserted into the target in bulk load operations
  - Concurrent select, insert, update, and delete operations are applied to both the preexisting indexes as well as the new index(es) being built
- Target
  - Data is inserted from source
  - User modifications (inserts, updates, deletes) on source are applied
    - Note: this activity is transparent to the user.

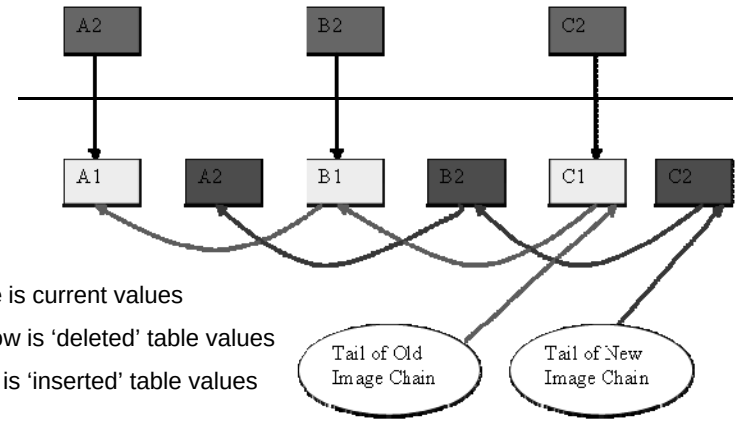
Phase 3 of Online Index Create: Final

- Source
  - All uncommitted update transactions must complete before this phase begins
  - All new user read and/or write transactions are blocked for a very short period until this phase completes
  - System meta data is updated to replace the source with the target
  - The source is dropped if required
    - For example, after rebuilding or dropping a clustered index
- Target
  - Index meta data is updated
  - Index is set to read-write status

Triggers and Row Level Versioning

- Inserted and Deleted tables are built using row versioning
- Version stored contains rows modified by triggering statement and in the trigger
- Not dependent on snapshot isolation

Use of Version Store by Triggers



- **Blue** is current values
- Yellow is 'deleted' table values
- Pink is 'inserted' table values

Multiple Active Result Sets

- One connection can have several concurrently executing batches
- MARS is off by default
  - Enabled through the connection string
- These statements can be interleaved with others even with results pending:
  - SELECT
  - FETCH
  - RECEIVE
  - All other statements will execute to completion

MARS and Transactions

- Connection can be set for auto commit
  - All batches run under their own transactions
  - Batch can start a manual transaction
  - Second concurrent batch to start a transaction will fail
- Connection can be set for manual commit
  - All batches run under the same transaction
- Uncommitted transactions will be rolled back when batch finishes
  - Batch-scoped transaction, new in SQL Server 2005

MARS and Row Level Versioning

Batch 1

BEGIN TRAN
SELECT …
FROM sometable

<return row1>
<return row2>
<return row3>

When Batch1 gets to ROW5, will it see updated value of Row5? NO

Will it see Row 6: YES

Batch 2

-- Must start after BEGIN TRAN

UPDATE ROW5 in sometable

-- There is no blocking because batches are
-- running under the same transaction

DELETE ROW6

If another transaction tries to see updated ROW5, the normal semantics will apply based on the isolation level of this transaction and the other one

Summary

- Concurrency Models
- Transactions in SQL Server
- Isolation Levels
- Pessimistic Concurrency Control
- Optimistic Concurrency Control
- Features using Row Level Versioning

Latest Information

- Check for updates and demo code
  http://www.solidqualitylearning.com/conferences.aspx

- Whitepapers
  Database Concurrency and Row Versioning in SQL Server 2005
  by Kalen Delaney and Fernando Guerrero
  www.microsoft.com/technet/prodtechnol/sql/2005/cncrrncy.mspx

  SQL Server 2005 Beta 2 Snapshot Isolation by Kim Tripp
  www.microsoft.com/technet/prodtechnol/sql/2005/SQL05B.mspx

Thank You!