

5 Things SQL Server does different from what many developers expect

Dr. Nico Jacobs

SQL Server Trainer & Consultant

U2U

Agenda

- Transactions
 - Isolation levels
 - Nested transactions
- Error handling
 - Things Try-Catch doesn't catch
 - Dealing with doomed transactions
- Stored procedures
 - Stored procedures versus ad-hoc SQL statements
 - Stored procedures and execution plans
- Indexes
 - Common mistakes in index design
- Statistics
 - Why automatically creating and updating statistics is not enough

Agenda

- Transactions
 - Isolation levels
 - Nested transactions
 - Error handling
 - Things Try-Catch doesn't catch
 - Dealing with doomed transactions
 - Stored procedures
 - Stored procedures versus ad-hoc SQL statements
 - Stored procedures and execution plans
 - Indexes
 - Common mistakes in index design
 - Statistics
 - Why automatically creating and updating statistics is not enough

Transactions

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability
- However, the default isolation is not what many people expect

Read committed isolation level

DEMO

What happened?

- Misconception 1a: *“Transactions protect me against operations that are happening in concurrent operations”*
- In the default isolation level (read committed) read locks are taken
 - When needed, and immediately released after use (not after transaction end)
 - Often only on the object that needs them (but the lock might escalate)

Select *
from
customers



How can we inspect?

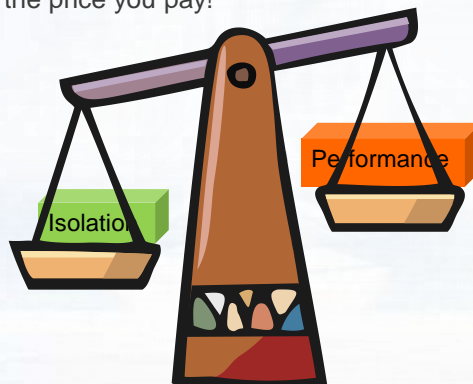
- With predefined report (database level)

Session ID	Host Name	Program Name	Login Name
52	SILVESTER	Microsoft SQL Server Management Studio - Query	Silvester@com
Transaction ID	Transaction Name	# Total Locks	Transaction Type
51699	user_transaction	6	Full Transaction
Resource Name			
Customers			
Lock Type	Request Mode	# Locks Granted	# Locking Requests Waiting
KEY	Shared	4	0
PAGE	Intent Shared	1	0
Other resources			

- With sys.dm_tran_locks view

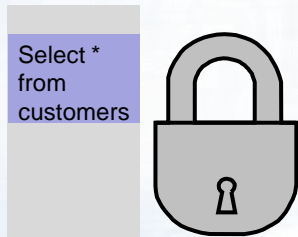
How can we solve it?

- Use stronger isolation level
 - **Repeatable read** to obtain multiple consistent reads
 - **Serializable** if you really need full isolation
 - **Snapshot** if you need full isolation but want to avoid blocking
- But beware of the price you pay!



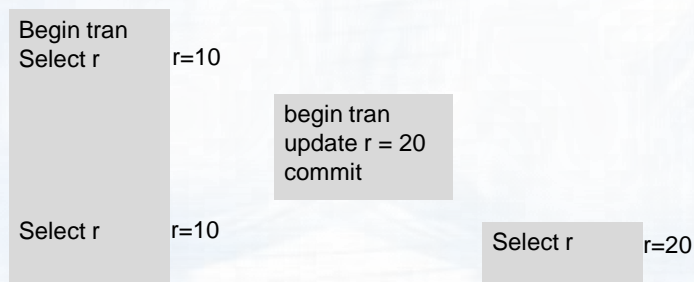
Serializable isolation level

- Serializable guarantees full isolation
 - Locks the range of rows used in the transaction
 - Keeps the lock until the transaction commits or rolls back
- It strongly reduces concurrency



Snapshot isolation level

- Snapshot isolation level uses versioning to solve isolation problems
 - Updates and deletes keep a copy of the original value
 - These copies are kept until all transactions that started before the modification was committed have finished: tempdb size ↑



Isolation levels

DEMO

Agenda

- **Transactions**
 - Isolation levels
 - **Nested transactions**
- Error handling
 - Things Try-Catch doesn't catch
 - Dealing with doomed transactions
- Stored procedures
 - Stored procedures versus ad-hoc SQL statements
 - Stored procedures and execution plans
- Indexes
 - Common mistakes in index design
- Statistics
 - Why automatically creating and updating statistics is not enough

Nested transactions

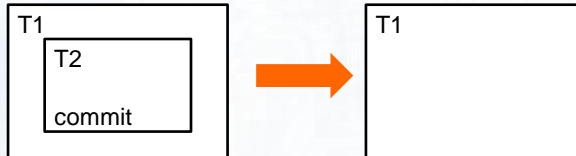
- SQL Server allows you to start a second transaction within a running transaction
- But these nested transactions behave different from a 'regular' transaction

Nested transactions

DEMO
DEMO

What happened?

- Misconception 1b: program nested transactions just as you would program a regular transaction
- When you have nested transactions
 - A commit commits the innermost transaction

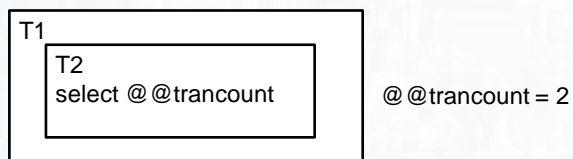


- A rollback rolls back the outer transaction (and hence all nested transactions)



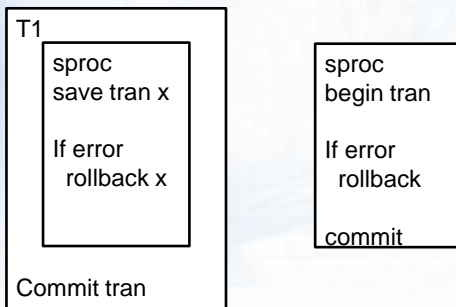
How can we inspect?

- @@trancount returns the number of nested transactions
 - @@trancount = 0 means we are not in any transaction



How can we solve it?

- If you want to locally rollback changes, use savepoints in the original transaction
- Only if there is no transaction we have to explicitly create one



Savepoints

DEMO

Agenda

- Transactions
 - Isolation levels
 - Nested transactions
- Error handling
 - Things Try-Catch doesn't catch
 - Dealing with doomed transactions
- Stored procedures
 - Stored procedures versus ad-hoc SQL statements
 - Stored procedures and execution plans
- Indexes
 - Common mistakes in index design
- Statistics
 - Why automatically creating and updating statistics is not enough

Structured error handling

- Try-catch blocks simplify coding and maintenance
- But they do not always behave as developers expect...

Try-catch

DEMO

What happened?

- Misconception 2: Try-catch will catch any problem that may occur
- Catch block cannot catch all errors
 - Severity < 11
 - Are warnings, and warnings are not caught
 - Severity > 19
 - Are fatal errors which can disconnect from the server before the catch block executes
 - Syntax errors
 - Cause the whole batch to fail, hence nothing in the batch executes
 - Late binding errors
 - Cause the batch from that point on to fail, hence none of the following code executes
- Remember that your catch code can cause errors as well

How can we solve it?

- If we detect a warning, raise it into an error, which can be caught
- Define alerts in SQL Server agent to respond to very severe errors
- Late-binding errors can be caught in a surrounding catch block
- Also errors in the catch-code can be caught in a surrounding catch block
- When working with transactions, verify with XACT_STATE if your transaction is doomed, and rollback or commit
- Consider using XACT_ABORT to influence the above state

Try-catch

DEMO

Agenda

- Transactions
 - Isolation levels
 - Nested transactions
- Error handling
 - Things Try-Catch doesn't catch
 - Dealing with doomed transactions
- **Stored procedures**
 - **Stored procedures versus ad-hoc SQL statements**
 - **Stored procedures and execution plans**
- Indexes
 - Common mistakes in index design
- Statistics
 - Why automatically creating and updating statistics is not enough

Stored Procedures versus ad-hoc SQL

- It is best to have an abstraction layer between the database and the application
- Just as views, stored procedures can be an abstraction layer...
- ... but stored procedures don't always do what people think they do...

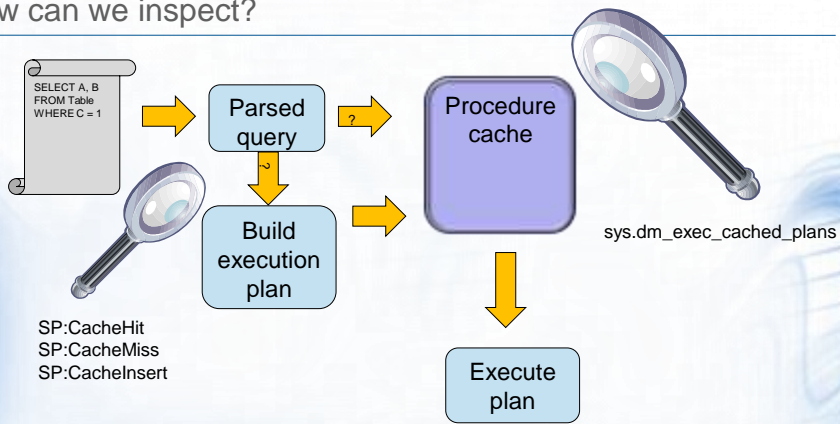
Timing stored procedures

DEMO

What happened?

- Stored procedures and ad-hoc code do not always execute in the same way
- Sometimes (as rumor has it) stored procedures can be beneficial...
- ... but this demo shows it can also be the other way round
- We need to know or inspect what is happening internally if performance is crucial

How can we inspect?



Inspecting stored procedures

DEMO

What happened?

- Misconception 3A: *“Stored procedures are compiled at creation time”*
 - Stored procedures are just ‘stored’ at creation time
 - They only get compiled at first use
 - The compiled plans are stored in memory, but not on disk
- Misconception 3B: *“Stored procedures have better performance than ad-hoc code”*
 - Stored procedures better reuse execution plans than ad-hoc code does...
 - ... unless you use `sp_executesql`
- This doesn't mean stored procedures are useless!

Execution plan reuse

- Stored procedures reuse execution plans when possible
 - Usually, this saves time and memory
 - Sometimes, this is bad

How can we solve it?

- Sometimes we want to avoid reuse of certain execution plans
 - EXECUTE ... WITH RECOMPILE
 - CREATE PROC ... WITH RECOMPILE
 - sp_recompile
 - OPTIMIZE FOR query hint

Agenda

- Transactions
 - Isolation levels
 - Nested transactions
- Error handling
 - Things Try-Catch doesn't catch
 - Dealing with doomed transactions
- Stored procedures
 - Stored procedures versus ad-hoc SQL statements
 - Stored procedures and execution plans
- Indexes
 - Common mistakes in index design
- Statistics
 - Why automatically creating and updating statistics is not enough

Indexes

- Indexes help to speed up your queries... some of them at least...

Indexes

DEMO
DEMO

What happened?

- Misconception 4: *"If SQL has an index on my search criteria, it will use that index"*
- Non-clustered indexes are comparable with indexes at the back of technical books
 - They repeat all the keys ('keywords')
 - But they do not repeat all the data, they just point to the data ('page number')
- If you get more references to pages than there are pages, it is easier to read the whole book
 - This also holds for your SQL Server
- Hence, non-clustered indexes are only useful for
 - Highly-selective queries, or
 - Queries where all the columns can be found in a few non-clustered indexes

How can we inspect?

- Execution plans can be seen ad-hoc with management studio
 - The plans can be saved as XML files, which makes storage and exchange easy
- Profiler can store execution plans with the SHOWPLAN XML event
 - Better filter this trace because it stores a lot of data
- Execution plans in the procedure cache can be inspected using the `sys.dm_exec_query_plan` dynamic management function

How can we solve it?

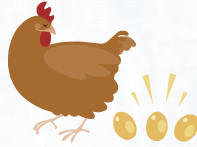
- For queries which return many rows and nearly all columns of a table, non-clustered indexes are of limited use to speed up data retrieval
 - This is why you should not write `SELECT *` queries or queries without a `WHERE` clause
 - They can still be useful to pre-sort your data to speed up `ORDER BY` and `JOIN` queries
- Queries that filter on a highly selective column or that only retrieve a small subset of the columns of a table should be able to benefit from well designed non-clustered indexes
- Remember that disjunctive queries (`OR`, `IN`) must have all their fields covered
 - `SELECT A, B FROM T WHERE C='v1' OR D<E`
- The database engine tuning advisor can give valuable advice
- Even with great indexes, sometimes wrong statistics can spoil the fun

Agenda

- Transactions
 - Isolation levels
 - Nested transactions
- Error handling
 - Things Try-Catch doesn't catch
 - Dealing with doomed transactions
- Stored procedures
 - Stored procedures versus ad-hoc SQL statements
 - Stored procedures and execution plans
- Indexes
 - Common mistakes in index design
- **Statistics**
 - **Why automatically creating and updating statistics is not enough**

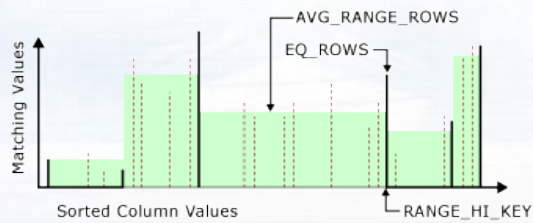
Statistics

- To determine how to optimally execute a query, SQL Server needs to know how many rows will be returned by each filter and join criterion
- How can SQL Server know these numbers without first executing the query?



- Statistic has histogram to summarize a column in at most 200 'steps'

----- Histogram is calculated from sampled column values



Statistics

DEMO

What happened?

- Misconception 5: *“SQL Server automatically creates and updates statistics, so I don’t need to worry about it”*
- Statistics are not updated on every insert, update or delete
 - They are only updated when a ‘statistically significant’ amount of data has been modified
- When multiple statistics are combined, they assume no correlation between the two columns
 - $P(\text{gender}='M') = P(\text{gender}='M' \mid \text{firstname} = \text{'Nico'})$
- Because of this, we sometimes get wrong estimates
 - And hence wrong index usage or execution plans

How can we inspect?

- DBCC SHOW_STATISTICS as well as the object explorer in Management Studio allow us to inspect the histogram used
- Sys.stats view and STATS_DATE function can be used to detect when a statistic was last updated
- Actual execution plans where the estimated number of rows is significantly different from the actual number of rows might indicate outdated statistics
 - Execution plans actually complain when they expect outdated statistics
- The profiler can capture the Missing Column Statistics event (in the errors and warnings group) if auto create statistics is turned off

How can we solve it?

- Frequently update statistics with job or script for columns on which the distribution changes quickly, such as identity columns, transaction dates, ...
- Create combined statistics for correlated columns which are queried together but don't occur in a combined index
- Some T-SQL language constructions can't create statistics, such as table valued functions or table variables. Storing this data in a (temporary) table can improve query performance due to the created statistics

Conclusions

- Transactions are of key importance in database development... but remember isolation levels, transaction nesting rules and XACT_ABORT
- TRY CATCH constructions make code much more readable... but they don't catch all problems
- Stored procedures are handy as an abstraction layer and can simplify security, but they do not necessarily provide better execution performance nor permanently store their execution plans
- Non-clustered indexes can be used to improve performance on SELECT queries, but not all SELECT queries will be better off when they use the index
- Statistics can be automatically created and maintained, but we still need to manually create and update some statistics to get optimal row estimates