

PASS
Data Community
SUMMIT 2021

Error and Transaction Handling in SQL Server

Erland Sommarskog

Data Platform MVP



Erland Sommarskog

Independent consultant based in Stockholm.
SQL Server MVP since 2001.

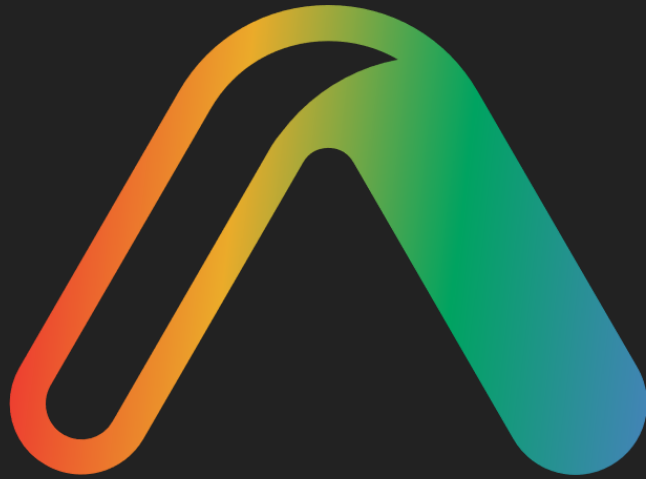
 esquel@sommarskog.se

 www.sommarskog.se



Slides and scripts:

<http://www.sommarskog.se/present.>



PASS
Data Community
SUMMIT 2021

Q&A for this session: Thu 11th at
15:15 to 15:45 UTC.
(10:15 to 10:45 EST)

Unexpected and Anticipated Errors

- Unexpected errors – You think that your code and your data is just fine, and you expect it to succeed. However, SQL Server thinks differently. This is the main focus of this presentation.
- Anticipated errors – You perform an action fully aware of that it can fail, and you have a plan B for this case.

Agenda


- What action does SQL Server take in case of error?
- How your code should react to an unexpected error.
- TRY-CATCH in SQL Server.
- Template for error handling in stored procedures.
- Client-side error handling.
- Handling anticipated errors.
- Nested procedures and transactions.
- Retry on deadlock?

What Actions Can SQL Server Take?

[PlainTest.sql](#)

Default (when XACT_ABORT OFF)

1. Close the connection.
2. Abort the batch and roll back transaction.
3. Abort the batch without rolling back.
4. Abort the scope and continue in caller.
5. Roll back statement and continue on next.



Internal SQL
Server Errors

What Actions Can SQL Server Take?

Default (when XACT_ABORT OFF)

1. Close the connection.
2. Abort the batch and roll back transaction.
3. Abort the batch without rolling back.
4. Abort the scope and continue in caller.
5. Roll back statement and continue on next.



Compilation errors at run-time

What Actions Can SQL Server Take?

Default (when XACT_ABORT OFF)

1. Close the connection.
2. Abort the batch and roll back transaction.
3. Abort the batch without rolling back.
4. Abort the scope and continue in caller.
5. Roll back statement and continue on next.

Appeared
first in SQL
2012

What Actions Can SQL Server Take?

Default (when XACT_ABORT OFF)

1. Close the connection.
2. Abort the batch and roll back transaction.
3. Abort the batch without rolling back.
4. Abort the scope and continue in call.
5. Roll back statement and continue on next.

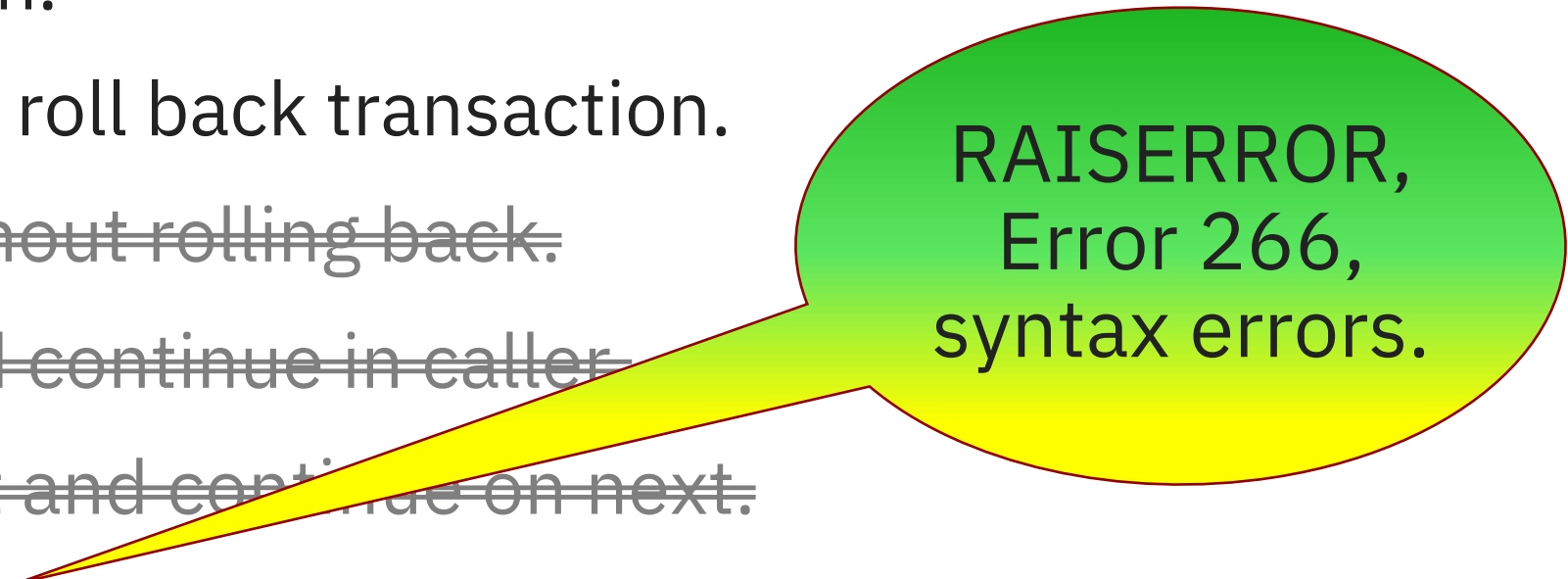
Many user errors
willy-nilly

Many user errors
willy-nilly

What Actions Can SQL Server Take?

SET XACT_ABORT ON changes this:

1. Close the connection.
2. Abort the batch and roll back transaction.
- ~~3. Abort the batch without rolling back.~~
- ~~4. Abort the scope and continue in caller.~~
- ~~5. Roll back statement and continue on next.~~
6. Ignore XACT_ABORT ON.



RAISERROR,
Error 266,
syntax errors.

Attention Signals

- Tell SQL Server to abandon execution.
- Occur with the client-side error "Timeout expired".
- Also generated by the red button in SSMS.
- Statement is always rolled back.
- Transaction rolled back **only** if XACT_ABORT is **on**.

How to Handle Unexpected Errors?

- Display an error message – the user must be informed that things went wrong.
- If you display a generic message to the user, log the original message somewhere – never lose it!
- Always rollback any open transaction.
 - Prevent incorrect/incomplete data from being persisted.
 - Avoid orphaned transactions.
- Abort execution – You have lost control, so you must not continue.

Handling Unexpected Errors, Cont'd

- With XACT_ABORT OFF (the default), there is no guarantee that execution will be aborted on error.
- Therefore, always have this statement on top of your stored procedures:
`SET XACT_ABORT, NOCOUNT ON.`
- Now execution is aborted for **most** errors.
- This can be sufficient for simple scripts.
- For application code, we also need TRY-CATCH.

TRY-CATCH

- Error in TRY block transfers execution to CATCH block.
- Transaction may be *doomed* – must be rolled back.
- Errors that roll back transaction => Dooming errors.
- Perfectly reasonable for deadlock, resource errors. Less so for conversion errors.
- Transaction always doomed with XACT_ABORT ON.

[TryCatch.sql](#)

Not All Errors Are Catchable

- Internal errors that close the connection.
- Compilation errors in the scope they occur – can be caught in outer scopes.
- Various odd errors cannot be caught, more common with linked servers or CLR.
- Attention signals.

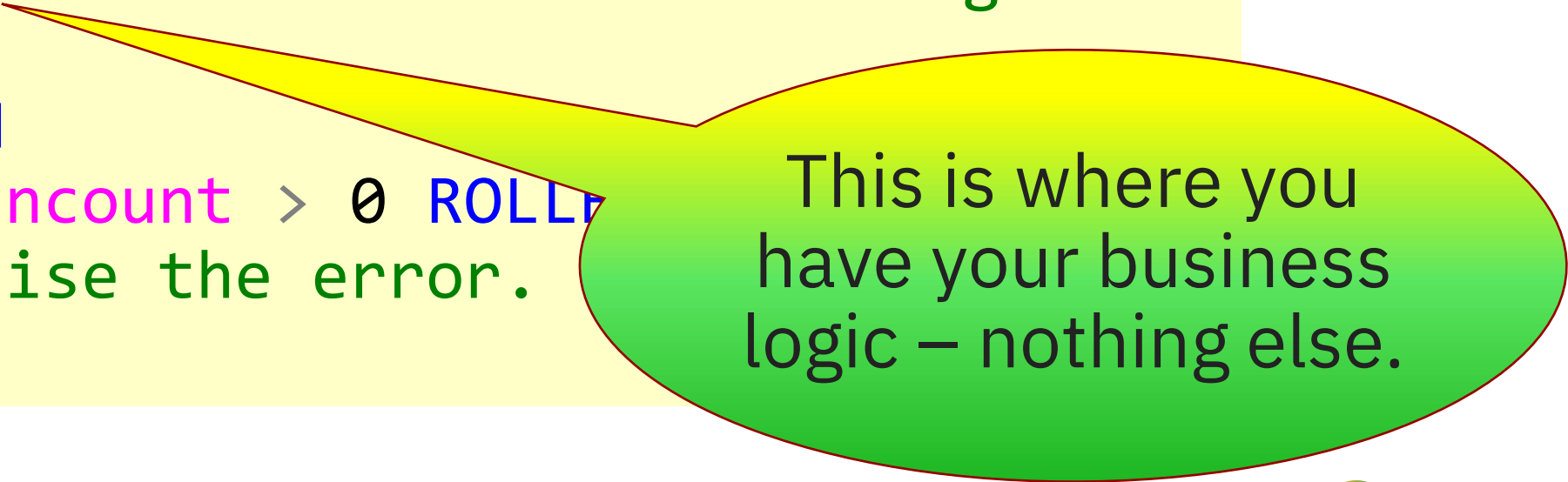
Template for Using TRY-CATCH

TRY-CATCH should be in (almost) all stored procedures.

```
CREATE PROCEDURE MySP @MyParameter int AS
SET XACT_ABORT, NOCOUNT ON
BEGIN TRY
    -- Here is the actual business logic.
END TRY
BEGIN CATCH
    IF @@trancount > 0 ROLLBACK TRANSACTION
    -- Re-raise the error.
END CATCH
```


The TRY Block Is Your Main Meat

```
CREATE PROCEDURE MySP @MyParameter int AS
SET XACT_ABORT, NOCOUNT ON
BEGIN TRY
    -- Here is the actual business logic.
END TRY
BEGIN CATCH
    IF @@trancount > 0 ROLLBACK
    -- Re-raise the error.
END CATCH
```



This is where you have your business logic – nothing else.

SET XACT_ABORT, NOCOUNT ON

```
CREATE PROCEDURE MySP @MyParameter int AS
SET XACT_ABORT, NOCOUNT ON
BEGIN TRY
    -- Here is the actual code
END TRY
BEGIN CATCH
    IF @@trancount > 0 RAISEERROR
    -- Re-raise the error.
END CATCH
```

This line is **before**
BEGIN TRY – only to be
seen at code review.

The CATCH Block – Two Lines Only

```
CREATE PROCEDURE MySP @MyParam INT
SET XACT_ABORT, NOCOUNT
BEGIN TRY
    -- Here is the actual code
END TRY
BEGIN CATCH
    IF @@trancount > 0 ROLLBACK TRANSACTION
    -- Re-raise the error.
END CATCH
```

The CATCH block is (almost) always the same – short and *non-intrusive*, just two lines.

IF @@trancount > 0 ROLLBACK TRANSACTION

- To make sure that you don't persist incorrect data and to prevent orphaned transactions.
- You may not have a BEGIN TRANSACTION today – but that could change tomorrow or two years later.
- IMPLICIT_TRANSACTIONS may be on.
- You may call a procedure which begins a transaction but fails to commit/roll back.
- What about caller's transaction? We'll talk about that later.

Re-raising the Error Is Essential

- An unexpected error must never be dropped on the floor!
- Always pass the bucket to the next guy on the call stack.
- Eventually the error message should be displayed and/or logged.

How to Re-raise Errors

Custom procedure

- Only choice on SQL 2005/2008.
- Want to log error or other custom behaviour.
- Avoids the semicolon trap.

;THROW

- Simple. :-)
- All error messages are preserved as-is.
- Makes sure that execution is aborted.

TRY-CATCH and XACT_ABORT ON

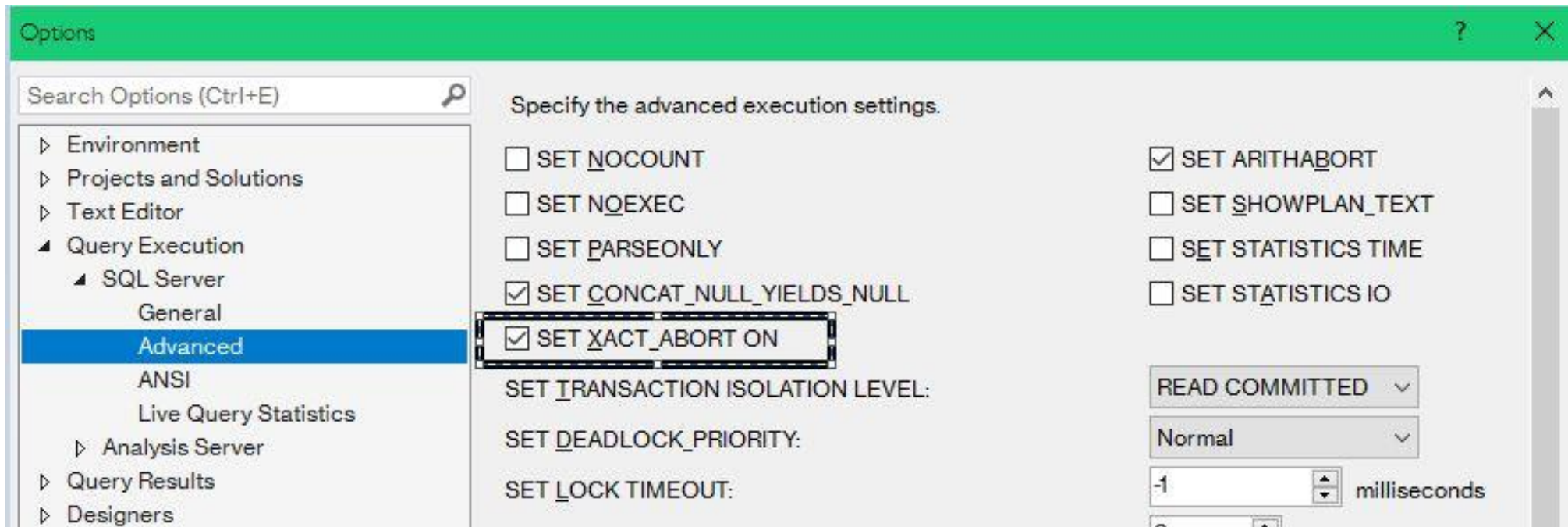
- If we have TRY-CATCH, do we still need SET XACT_ABORT ON?

Yes, because:

- TRY-CATCH does not catch all errors.
- Attention signals, i.e. “Timeout Expired”.

A Tip for SSMS

- You can have XACT_ABORT ON by default.



Client-side Error Handling

- All calls to SQL Server must be error-checked, and in case of error the client must always submit `IF @@trancount > 0 ROLLBACK TRANSACTION` (Or rollback through its own transaction object.)
- Don't rely on the SQL code having XACT_ABORT ON. Each component should do its job.

Make Sure You Get All Result Sets!

[AllResultSets.cs](#)

- With **ExecuteReader**, always use `NextResult` to get through all result sets.
- **ExecuteScalar** – only gets first result set, but since it's for a scalar value – not real issue.
- **ExecuteNonQuery** – gets all result sets and errors.
- **SqlDataAdapter.Fill(DataSet)** – Ditto.
- **SqlDataAdapter.Fill(DataTable)** – Only gets first result set – errors can be missed, avoid!

Handling Anticipated Errors

- Example: delete a customer if possible, but if there are FK references just set the isactive flag to 0.
- Too many FK references to check them all.
- Attempt a DELETE and in the CATCH block update the flag.
- Turn off XACT_ABORT around the DELETE – only.
- In CATCH handler, check for anticipated error number and procedure name.

[DeleteCustomer.sql](#)

”Nested” Transactions

BEGIN TRANSACTION

Transaction starts.

BEGIN TRANSACTION

Increments @@trancount.

COMMIT TRANSACTION

Commits nothing,
decrements @@trancount.

COMMIT TRANSACTION

@@trancount = 0 =>
Transaction commits.

ROLLBACK TRANSACTION

Rolls back it all.

Nested Procedures

- What if an outer procedure starts a transaction...
...and calls an inner procedure that also starts a transaction?
- Should CATCH handler of the inner SP really roll back it all?
- Ideally, no – caller may want to employ a plan B.
- In practice YES, because:
 - There is no better way in SQL Server.
 - The inner procedure has failed to fulfil its contract.

Savepoints to the Rescue?

```
BEGIN TRANSACTION MyTran
-- First part
SAVE TRANSACTION MySave
-- Second part
ROLLBACK TRANSACTION MySave
```

This rolls back only Second Part, but transaction is alive and First Part can still be committed.

Useful?

Savepoints are Useless

- Cannot roll back to a savepoint when transaction is doomed.
- Always doomed with `XACT_ABORT ON`.
- And even with `OFF`, rollback is only possible for some errors.
- Also, trying to employ this as a general pattern would make your error handling too complex.
- Not supported in distributed transactions.

Deadlock Retry?

- On a deadlock, should you retry the operation?
- My answer: Only when deadlocks are a real problem!
- Retry code clutters the code and hides the actual business logic.
- Testing is difficult – how do you provoke a deadlock?
- Incorrectly written retry code can cause spurious data errors that are difficult to understand.

If You Encounter Deadlocks

- First investigate if you can prevent the deadlock.
 - Better indexing.
 - Reviewing access order.
 - Using hints.
 - Serialise with application locks.
 - SET DEADLOCK_PRIORITY LOW.
- But there certainly are situations where it is very difficult to prevent occasional deadlocks and the best solution is to implement deadlock retry.

Two Key Rules for Deadlock Retries

- Never redo only part of a transaction.
 - => Never do deadlock retry when called inside a transaction.
- Never redo a committed transaction. That could lead to data being doubled.
 - E.g.: after the transaction there is a SELECT that deadlocks.
- Design your deadlock retries to avoid these accidents.

[RetryLogic.sql](#)

Summary – Aims and Means

Always communicate unexpected errors – don't lure users to think they see correct data.

- Re-raise the error!
- Get all result sets!

Always abort execution on unexpected errors – don't persist incorrect data.

- SET XACT_ABORT ON
- IF @@trancount > 0 ROLLBACK TRANSACTION
- Re-raise the error!

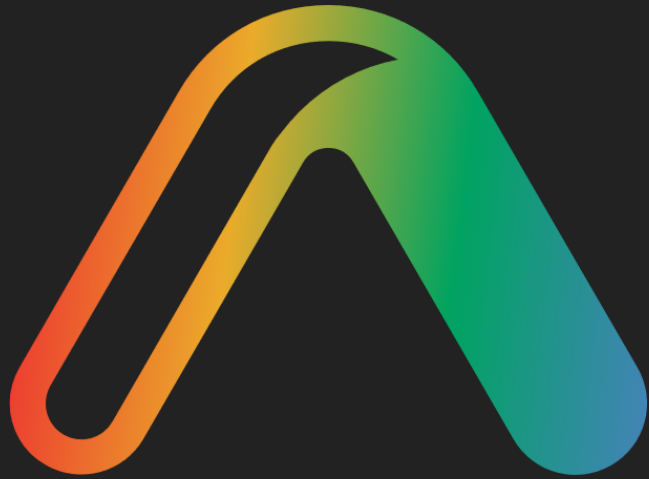
Summary – Aims and Means

Prevent orphaned transactions.

- SET XACT_ABORT ON
- IF @@trancount > 0 ROLLBACK TRANSACTION
- Also in client code!!

Don't lose the original error message – without it troubleshooting is very difficult.

- Keep things simple.
- Only do deadlock retry when needed, not as routine.
- Watch out for the semicolon trap!



PASS
Data Community
SUMMIT 2021

Q&A for this session: Thu 11th at
15:15 to 15:45 UTC.
(10:15 to 10:45 EST)

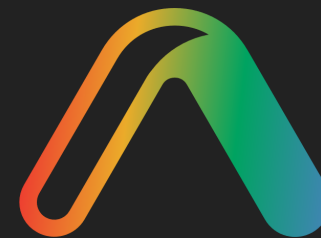
Session evaluation

Your feedback is important to us



Evaluate this session at:

www.PASSDataCommunitySummit.com/evaluation



PASS
Data Community
SUMMIT 2021

Thank you

Slides and scripts on

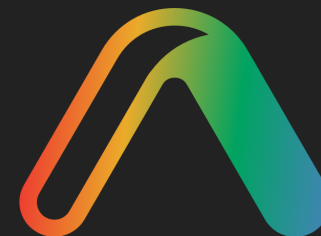
<http://www.sommarskog.se/present>

Three parts and three appendixes? Start here:

http://www.sommarskog.se/error_handling/Part1.html

Erland Sommarskog

esquel@sommarskog.se



PASS
Data Community
SUMMIT 2021