

Erland Sommarskog

When Things Go Wrong – Error Handling in SQL Server



Erland Sommarskog

Independent consultant based in Stockholm

SQL Server MVP since 2001

<http://www.sommarskog.se>

esquel@sommarskog.se

Slides and scripts are available on
<http://www.sommarskog.se/present>
and the SQL Saturday web site

Thank you to our AWESOME sponsors!



Agenda

Focus: How to handle unexpected errors.

- What action does SQL Server take in case of error?
- SET XACT_ABORT ON.
- How your code should react to an unexpected error.
- TRY-CATCH in SQL Server.
- How to write CATCH blocks and why.
- Client-Side Error Handling.
- How to handle nested procedures and transactions.

What Actions Can SQL Server Take?

[PlainTest.sql](#)

Default (when XACT_ABORT OFF)

Internal SQL
Server Errors

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.

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)

Many user errors willy-nilly

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.

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 scope.~~
- ~~5. Roll back statement and continue on next statement.~~
6. Ignore XACT_ABORT ON.

RAISERROR,
Error 266,
syntax errors.

Attention Signals

- Tells SQL Server to abandon execution.
- Occurs 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

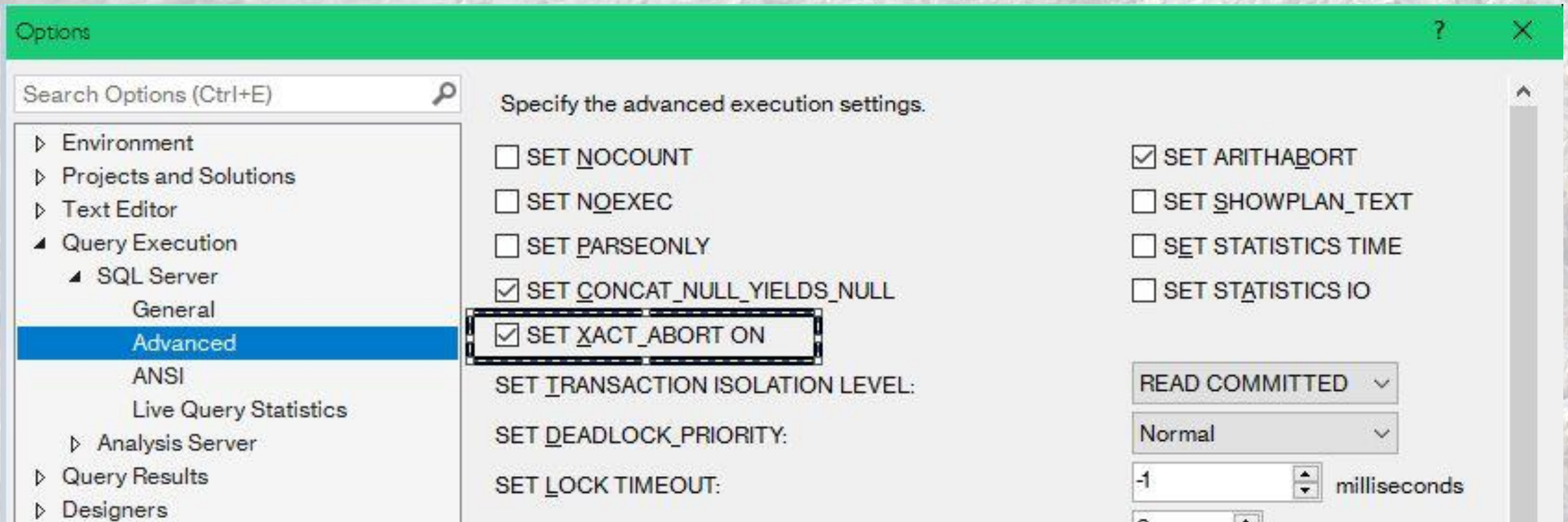
- With XACT_ABORT OFF (the default), there is no guarantee that execution will be aborted on error.
- Therefore, always have this statement on top of your stored procedures:

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.

A Tip for SSMS

You can have XACT_ABORT ON by default.



TRY-CATCH

- Error in TRY block transfers execution to CATCH block. [TryCatch.sql](#)
- 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.

Not All Errors Are Catchable

- Internal errors that close the connection.
- Compilation errors in the scope they occur – can be caught in outer scopes.
 - Vote here: <http://feedback.azure.com/forums/908035-sql-server/suggestions/33673684-try-catch-should-always-work>.
- Various odd errors cannot be caught, more common with linked servers or CLR.
- Attention signals.

How to use TRY-CATCH

- You should have TRY-CATCH in (almost) all your stored procedures.
- The TRY block is the main meat – this is where you have your business logic.
- The CATCH block is (almost) always the same in every procedure.
- It should be short and non-intrusive. Two lines only:
 1. Roll back any open transaction.
 2. Re-raise the error.

CATCH Block Rationale

- Roll back any open transaction:
 - To make sure that you don't persist incorrect data.
 - To avoid orphaned transactions.
- Re-raise the error:
 - 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.

Roll Back Open Transactions

- `IF @@trancount > 0 ROLLBACK TRANSACTION`
Always have this line.
- 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.

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”.

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.
- **DataAdapter.Fill(DataSet)** – Ditto.
- **DataAdapter.Fill(DataTable)** – Only gets first result set – errors can be missed, avoid!

”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.
- 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.

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 – don't do fancy error handling.
- Watch out for the semicolon trap!

That's All Folks!

Erland Sommarskog, esquel@sommarskog.se

Slides and scripts on the SQL Saturday site as well as <http://www.sommarskog.se/present>.

Three parts and three appendixes? Start here: http://www.sommarskog.se/error_handling/Part1.html