



*PASS

SQLSATURDAY

EDINBURGH | 01 FEB 2020

When Things Go Wrong

Error Handling in SQL Server

Erland Sommarskog, SQL Server MVP



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

Thanks to our Sponsors



..and special THANK YOU to our **Volunteers** (light blue #SQLSat927 polo shirt)

Agenda

Focus: How to handle unexpected errors.

- What action does SQL Server take in case of error?
- SET XACT_ABORT ON.
- TRY-CATCH.
- How to write CATCH blocks.
- Client-Side Error Handling.
- How to handle nested procedures and transactions.
- A quick look at natively compiled stored procedures.

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)

Appeared first
in SQL 2012

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)

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 transaction.~~
- ~~4. Abort the scope and continue in the next scope.~~
- ~~5. Roll back statement and continue on next statement.~~
6. Ignores 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**.

Recommendation

Always have this statement on top of your stored procedures:

`SET XACT_ABORT, NOCOUNT ON`

More consistent error behaviour.

Reduces the risk for orphaned transactions.

Takeaway So Far

You cannot rely on that you will be able to continue execution on error.

You cannot rely on that execution will be aborted.

Therefore:

- Keep it simple.
- Use TRY-CATCH.

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 scope.

Various odd errors cannot be caught, more common with linked servers or CLR.

Attention signals.

CATCH Block Recipe

Important principle: Since this is code that is in **every** procedure it should be short and non-intrusive. Two lines, that's all:

1. Roll back any open transaction.
2. Re-raise the error.

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.

Re-raising the Error

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.

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 outer procedure starts a transaction...

...and calls an inner procedure that also starts a transaction?

Should CATCH handler of inner 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 Throw  
-- Second part  
ROLLBACK TRANSACTION Throw
```

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 savepoint when transaction is doomed.
- Always doomed with `XACT_ABORT ON`.
- And even with `OFF`, rollback is only possible for some errors.
- Not supported in distributed transactions.
- Not supported with mem-optimised tables.

Natively Compiled Procedures

```
CREATE PROCEDURE hekaton_sp  
WITH NATIVE_COMPILATION, SCHEMABINDING AS  
BEGIN ATOMIC WITH  
    (TRANSACTION ISOLATION LEVEL=SNAPSHOT,  
     LANGUAGE='us_english')  
    -- SQL code here.  
END
```


Natively Compiled Procedures, cont'd

Procedure == Transaction.

Only need TRY-CATCH for anticipated errors.

BEGIN/COMMIT/ROLLBACK TRAN not permitted.

A nested SP call defines an implicit savepoint.

An uncaught error aborts the procedure and rolls back to the savepoint.

Transactions only doomed for a reason – concurrency and resource errors. XACT_ABORT has no effect.

In short: how error handling should be!

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.
- Watch out for the semicolon trap!

Thanks to our Sponsors

Global Partner  Microsoft Azure  PASS

SentryOne



GETHYNELLIS.COM

 quorum

GOLD



**ADVANCING
ANALYTICS**


SIOS

Silver

 **DLM**
Consultants
Applying DevOps to Databases

Bronze  redgate

 **POWER BI
SENTINEL**
Governance · Auditing
Disaster Recovery

That's All Folks!

Erland Sommarskog
esquel@sommarskog.se

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

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