

SQLskills Immersion Event

IETS: In-Depth SQL Server Programming for Developers

Module 5: Stored Procedures Part 2

Transactions and Error Handling

Bob Beauchemin
BobB@SQLskills.com



Overview

- Batches
- Transaction Mode
- Transaction Termination
- Creating/Controlling Transactions
- Transaction Guidelines
- Handling Errors



2

© SQLskills, All rights reserved.
<http://www.SQLskills.com>



Batches

- **Parsed as a unit**
 - If any statement fails syntax then NO statements (in that batch) are executed
 - **At Execution**
 - Some errors skip to next batch
 - Some errors continue execution
- ```
SELECT * FROM pubs.dbo.authors
SELECT * FROM pubs.dbo.authors
SELECT * FORM pubs.dbo.authors
GO
```
- Error: Server: Msg 170, Level 15, State 1, Line 3  
Line 3: Incorrect syntax near 'form'.

## Transactions

- Controlled by Transaction Mode of session
  - Defined explicitly
  - Defined at connection level
- Allow multiple statements to be treated as a single logical unit of work
- Handled automatically across databases on the same instance
- Handled through MSDTC inter-instance
- Require error handling logic

## Transaction Mode

- **Autocommit transaction (default)**
  - Statement level implicit transaction
  - Each statement commits as a single unit
- **Explicit transaction (user-defined)**
  - BEGIN TRANSACTION
  - COMMIT TRANSACTION
- **Implicit transaction**
  - Session Level Setting  
SET IMPLICIT\_TRANSACTIONS ON
- **Batch scoped transactions (SQL Server 2005+)**
  - Only when MARS enabled on client

## Distributed Transactions

- **Transactions are managed by MSDTC**
- **These are in effect**
  - Automatically when update across linked server
  - Manually in T-SQL
  - Distributed partitioned views
  - Automatically in programmatic APIs
    - System.Transactions
    - MTS/COM+
- **Distributed transactions run at serializable isolation**
- **Distributed transactions must have a timeout**
  - No multi-instance lock manager

## Hekaton and Isolation

- **Hekaton uses multi-version engine**
  - Multiple versions of rows in memory
  - Only one version is valid at a given time
  - Which one is read depends on transaction isolation level
- **Optimistic concurrency used**
  - No locks and no latches
  - Assumes transactions will commit (doesn't wait)
  - Three phases
    - Normal processing, preparation, postprocessing
- **Write-write conflicts cause rollback**
  - Immediately
- **Error-handling and retry logic required**

## Transaction Termination (1)

- **Resource Error – Automatically Handled**
  - If the statement fails due to a SQL Server resource error (e.g., the transaction log fills), SQL Server maintains data integrity automatically
- **User-defined Error – Programmatically Handled**
  - Conditionally, when your business rules (i.e. constraint violation or a lock timeout) are violated, YOU must programmatically determine the fate of the transaction.

## Transaction Termination (2)

- **SET ARITHABORT (set to off by default)**
  - ON – rollback statement/transaction when divide-by-zero or arithmetic overflow error occurs
  - OFF – return NULL
- **SET XACT\_ABORT (off by default)**
  - ON – rollback statement/transaction when run-time error occurs

## Understanding Transactions

- **BEGIN TRANSACTION**
  - Begins a new transaction
  - Increments @@TRANCOUNT
  - Supports special “marked transactions”
    - BEGIN TRANSACTION TransactionName  
WITH MARK ‘important transaction’
- **COMMIT TRANSACTION**
  - Decrements @@TRANCOUNT by 1
  - Does NOT commit the transaction UNLESS the @@TRANCOUNT is 0

## Understanding Transactions

- **ROLLBACK TRANSACTION**
  - Returns @@TRANCOUNT to 0
  - Cancels all levels of Nested Transactions – even when using ROLLBACK TRANSACTION to a named (i.e. marked transaction).
- **SAVE TRANSACTION (Named Savepoints)**
  - Do not affect @@TRANCOUNT
  - When you ROLLBACK to a savepoint you must still “commit” or “rollback” the pending transaction
  - MARS does not support named savepoints

## Transaction Flow (@@trancount)

```
BEGIN TRANSACTION
UPDATE1 ...
SELECT @@TRANCOUNT ←
SAVE TRANSACTION Bar
SELECT @@TRANCOUNT ←
 BEGIN TRANSACTION
 SELECT @@TRANCOUNT ←
 UPDATE2 ...
 ROLLBACK TRANSACTION Bar
 SAVE TRANSACTION Foo ← Probably the most interesting
 SELECT @@TRANCOUNT ←
 UPDATE3 ...
 SELECT ...
 SELECT @@TRANCOUNT ←
 COMMIT TRANSACTION
ROLLBACK TRANSACTION Foo
SELECT @@TRANCOUNT ←
SELECT ...
```

## Nesting Transactions

- **Issues in Nesting Transactions**
  - Use @@trancount to determine nesting level
  - Transactions aren't truly "nested" when named – rollback always rolls back to OUTER most transaction
  - Rollback to a savepoint does NOT decrement @@trancount – must also commit nested transaction to decrement @@trancount
- **Handle errors**
  - Try/Catch logic
  - Using session variables

## Transaction Guidelines

- **Keep transactions as short as possible**
  - Use multiple smaller transactions if possible
  - Consider messaging
- **Keep transactions to only one batch**
- **Avoid transactions that require user interaction**
- **Be aware of complications around nesting of transactions**
- **Consider logging**

## Types of Errors

- **Categories of Errors Raised in T-SQL code**

- Syntax
  - (SELEC \* FROM SomeTable)
- Object resolution
  - (SELECT \* FROM NonExistentTable)
- Statement termination errors
  - Execution resumes at next statement
- Scope/level termination errors
  - Execution resumes in calling procedure
- Batch termination errors
- Batch termination and rollback transaction errors
- Connection termination errors
- Server termination errors

## Error Severities

- **Error Severities Are Grouped**

- Severity 0 - Information (T-SQL PRINT)
- Severity 1-9 - Information (not used by SQL Server)
- Severity 10 - Converted to severity 0
- Severity 11-16 - Programming Errors
- Severity 17-25 - Resource Error
- Severity 19 and above - must be sysadmin to raise
- Severity 20-25 - Terminate the connection



## Programming Error Severity

- **Programming error severity depends on cause**
  - Not how severe the error is
- **Severity meanings**
  - 11 - Database object not found or other object problem
  - 12 - NOLOCK scan error
  - 13 - Transaction Deadlock
  - 14 - Insufficient Permission, login failure, other environment errors
  - 15 - Syntax error in SQL statement
  - 16 - Miscellaneous User Error

## Handling Errors – The Old Way

- After each statement you need to test the state of @@ERROR
- To centralize error checking code, use labels with GOTO and RETURN statements
- Errors such as data type conversion errors causes batch to terminate; not every error can be trapped
- Makes for a lot of additional code

```
IF @@ERROR <> 0
BEGIN
 RAISERROR...
 ROLLBACK
 RETURN
END
```

## Handling exceptions

- **SQL Server 2005+ added exception handling**
  - Error handling in previous SQL Server versions tedious
    - @@ERROR set on each statement
    - set variable with @@ERROR, then check value
  - BEGIN-END TRY BEGIN-END CATCH blocks
    - semantic equivalent of BEGIN-END blocks
    - additional functions return error info
    - can query transaction status
    - can save @@ERROR equivalent

## Error handling functions

- **New error information functions**
  - available inside catch block
  - ERROR\_NUMBER() - number of the error
  - ERROR\_SEVERITY() - severity
  - ERROR\_STATE() - error state number
  - ERROR\_MESSAGE() - message text
  - ERROR\_LINE(), ERROR\_PROCEDURE() - line & proc
- **New transaction information function**
  - Operation that forced logic into catch block may cause un-committable transaction
  - XACT\_STATE() – state of transaction
    - Returns 1, 0, -1

## Try/Catch Block Programming

```
BEGIN TRANSACTION
BEGIN TRY
 -- Generate a constraint violation error.
 DELETE FROM Production.Product
 WHERE ProductID = 980;
END TRY
BEGIN CATCH
 SELECT ERROR_NUMBER() AS ErrorNumber,
 ERROR_SEVERITY() AS ErrorSeverity,
 ERROR_STATE() as ErrorState,
 ERROR_MESSAGE() as ErrorMessage;
END CATCH
IF ...
 ROLLBACK TRANSACTION
ELSE
 COMMIT TRAN...
GO
```

## Exception Handling: TRY/CATCH Semantics

- TRY/CATCH constructs may be nested
- Thrown exceptions are caught by the CATCH block associated with the closest compiled TRY/CATCH construct
- Error details are available at any time in catch block
  - Error functions cannot be used outside catch

## Exception Handling: Errors

- All errors that set @@error throw an exception
  - Statement Abort Errors
  - Level Abort Errors
  - Batch Abort Errors
  - Tran Abort Errors
    - Transaction Doomed
- TRY-CATCH blocks available in batches, procedures, triggers
  - Not available in T-SQL UDFs
  - UDF errors caught in caller's TRY-CATCH

## What TRY/CATCH Doesn't Catch

- TRY/CATCH doesn't always catch
  - Compile errors
  - Object resolution errors
    - Can be caught in outer catch block, calling procedure
  - Recompile errors
    - Not caught for backward compatibility
  - Remote errors
    - Remote is not SQL Server - generic error in catch
    - Remote is SQL Server - caught if batch termination error
  - Attention (i.e. Command.Cancel)

## Exception Handling: (Re)throwing an Error

- **RAISERROR** raises real errors within the scope of a TRY or CATCH block
  - "Throws" within the scope of a TRY block
- **To "Rethrow"**
  - Capture error details within the catch block
  - Log error if desired
  - Pass error details to RAISERROR
  - Get procedure and line number from message
  - Can't throw a "system" error with RAISERROR
    - one workaround is to add 50000 to original error, then subtract 50000 on caller to get real number
    - Caller/client code that depends on error numbers will need to be rewritten
- **THROW** statement in SQL 2012 accomplishes this

## THROW and FORMATMESSAGE

- **Adds required functionality to TRY/CATCH**
  - You can (re)throw a system error
- **Use FORMATMESSAGE in conjunction**
  - To format user-defined messages
  - Does not completely replace RAISERROR
    - Can't write to error log
    - Can't write error WITH NOWAIT

## Client Error Handling vs T-SQL

- **T-SQL Error Message is flat**
  - Multiple errors combined into one message
  - ERROR\_NUMBER(), @@ERROR come from the last error message
- **ADO.NET SqlException is like T-SQL error**
  - One flat error
- **SqlErrors collection = one message per error**
  - SqlError.Number, other info is correct for each error
  - Can be processed separately
- **Can parse aggregated messages (eg. from T-SQL TRY/CATCH) using Regex**
  - Regex parsing function can be exposed as server-side UDF too
  - Can expose multiple errors in XML format
  - Decompose on client or server

## Review

- **Batches v. Transactions**
- **Transaction Mode**
- **Transaction Termination**
- **Creating/Controlling Transactions**
- **Transaction Guidelines**
- **Tips for Minimizing Transactional Code**
- **Handling Errors**

## Resources

- MSDN Webcast: T-SQL Exception Handling in SQL Server 2005, Sarah Tahir
- A Developer's Guide to SQL Server 2005 - Ch 7-8, Addison-Wesley, *Bob Beauchemin and Dan Sullivan*
- Erland Sommarskog writings on error handling
  - <http://www.sommarskog.se>
- **Books (SQL Server Specific):**
  - The Guru's Guide to SQL Server Architecture and Internals
- **Books (General)**
  - Principles of Transaction Processing, Gray and Reuter



29

© SQLskills, All rights reserved.  
<http://www.SQLskills.com>

# Questions?

