Ready, Steady, Catch!

- Julie Koesmarno

www.MsSQLGirl.com

SQL Lunch – 22 Oct 2012

@MsSQLGirl

About Me

- 8+ years experience with SQL Server (since SQL Server 2000)
- MCITP 2008, MCSA 2012
- Past projects: large scale 24/7 Data Integration, Data Management and Data Cleansing system
- Current project: Data Warehouse and Analysis Services Tabular Model project

Abstract

SQL Server 2005 introduced TRY - CATCH construct to handle errors, a significant improvement to its predecessor manual error handling.

In SQL Server 2012 , the error handling feature has been further enriched, to be a step closer to error handling that you can commonly see in other programming languages such as C#.

This session will go through fundamental steps to error handling in SQL Server 2012 and how to incorporate this as part of your coding standard.

Agenda

- Error Handling
- Techniques
- Tips & Tricks
- Summary

Error Handling

Error Handling

- Anticipate the known
- Trap the unknown

- Gracefully terminate execution
- Log the errors for later reporting/inspection
- Provide alternate path

Existing Technique

- Pre – SQL Server 2005
- Using @@ERROR
- Checks every single statement to be handled
- Using GOTO

```
DECLARE @Err        INT
DECLARE @AuditID    INT
DECLARE @Msg        NVARCHAR(255)

SET @AuditID = 1209

-- Statement 1
INSERT INTO [Dim].[Country] (CountryName, CountryCode, AuditID)
    SELECT 'New Zealand', 'NZ', @AuditID;
SELECT @Err = @@ERROR
IF (@Err != 0)
BEGIN
    SET @Msg = 'Error in Statement 1'
    IF @Err = 2601
        GOTO Statement2
    ELSE
        GOTO ExitOK
END

Statement2:
    -- Statement 2
    INSERT INTO [Dim].[Country] (CountryName, CountryCode, AuditID)
        SELECT 'Sweden', 'SE', @AuditID;
    SELECT @Err = @@ERROR
    IF (@Err != 0)
    BEGIN
        SET @Msg = 'Error in Statement 2'
        GOTO ExitOK
    END

GOTO ExitNormal

ExitOK:
    SELECT @Err ErrorNumber, @Msg + '. ExitOK executed' AS GoToMessage

ExitNormal:

GO
```

```
-- <statement to handle>
SELECT @Err = @@ERROR
IF (@Err != 0)
…
…
GOTO ExitError
…
```

```
-- <statement to handle>
SELECT @Err = @@ERROR
IF (@Err != 0)
…
…
GOTO ExitNormal
…
```

```
-- <statement to handle>
SELECT @Err = @@ERROR
IF (@Err != 0)
…
…
GOTO ExitWarning
…
```

```
-- <statement to handle>
SELECT @Err = @@ERROR
IF (@Err != 0)
…
…
GOTO ExitWarning
…
```

… spaghetti code …

New Technique with TRY…CATCH

- SQL Server 2005+

```
BEGIN TRY
   { sql_statement | statement_block }
END TRY
BEGIN CATCH
   [ { sql_statement | statement_block } ]
END CATCH [ ; ]
```

Throw (new)

- SQL Server 2012

```
THROW
[ { error_number | @local_variable },
  { message | @local_variable },
  { state | @local_variable } ]
[ ; ]
```

Useful Functions

- SQL Server 2000+

  `@@ERROR`
  `@@PROCID`

- SQL Server 2005+

  `ERROR_LINE()`
  `ERROR_SEVERITY()`
  `ERROR_MESSAGE()`
  `ERROR_STATE()`
  `ERROR_NUMBER()`
  `FORMATMESSAGE()`
  `ERROR_PROCEDURE()`

Examples

- Re-THROW

```
1  BEGIN TRY
2      SELECT 1/0;
3  END TRY
4  BEGIN CATCH
5      PRINT 'I''m Caught!!';
6      THROW;
7  END CATCH
```

0 %

Results | Messages

```
(0 row(s) affected)
I'm Caught!!
Msg 8134, Level 16, State 1, Line 2
Divide by zero error encountered.
```

- Customized THROW

```
14 BEGIN TRY
15     SELECT 1/0;
16 END TRY
17 BEGIN CATCH
18     THROW 60000, 'I''m Caught!', 5;
19 END CATCH
20
21
```

0 %

Results | Messages

```
(0 row(s) affected)
Msg 60000, Level 16, State 5, Line 6
I'm Caught!
```

Tips & Tricks

Try and Catch Me!

- 10 < Severity < 20
- DB connection not closed
- Can be nested
- Can be used with transaction
- Not available in User Defined Function

Can’t Catch Me!

- Compile errors
- Object name resolution errors
- Uncommittable transactions

Throw

- Preceded by semicolon (;) terminator
- Called within CATCH block:
  - To raise a new error
  - To rethrow error
- Called outside CATCH block:
  - Must specify parameters
  - Severity 16
  - Line number and procedure where it’s raised

Not using SQL Server 2012 yet?

- RAISERROR

```
RAISERROR ( { msg_id | msg_str | @local_variable }
{ ,severity ,state }
[ ,argument [ ,...n ] ] )
[ WITH option [ ,...n ] ]
```

- BOL:

  New application should use THROW instead

- Commonly used since SQL Server 2000

Inside CATCH

THROW	RAISERROR
• User exception in sys.messages • Custom error number >= 50000 • Original error number from “re-throw”	• User exception in sys.messages • Error number 50000 for adhoc
• Severity = 16 • Original severity from “re-throw”	• Any severity
• Can re-throw	• New exception generated
• Output is buffered	• Output is not buffered when using WITH NOWAIT
• Customize via FORMATMESSAGE() on user defined errors	• Customize via token substitutions

Useful Scenario

- Error Logging

```
133 BEGIN TRY
134     EXEC [ref].[uspNoErrorHandling]
135 END TRY
136 BEGIN CATCH
137     INSERT INTO [ref].[Audit]
138     (
139         [Username], [CreatedDate],  [ErrorNumber], [ErrorMessage],
140         [ErrorSeverity], [ErrorProcedure], [NestLevel]
141     )
142     SELECT
143         SUSER_NAME(), GETDATE(), ERROR_NUMBER(), ERROR_MESSAGE(),
144         ERROR_SEVERITY(), ERROR_PROCEDURE(), @@NESTLEVEL;
145 END CATCH
146 GO
147
148 SELECT
149     [ID], [Username], [CreatedDate],    [ErrorNumber], [ErrorMessage],
150     [ErrorSeverity], [ErrorProcedure], [NestLevel]
151 FROM [ref].[Audit]
```

00 %

Results | Messages

	ID	Username	CreatedDate	ErrorNumber	ErrorMessage	ErrorSeverity	ErrorProcedure	NestLevel
1	1	TWENTYFOUR\Julie	2012-10-20 14:22:34.693	8134	Divide by zero error encountered.	16	uspNoErrorHandling	0

Useful Scenario

- Use with Transaction

```
231  -- Insert Country and Human Development Index for New Zealand.
232  -- If unsuccessful, rollback all statements within the transaction.
233  BEGIN TRY
234      BEGIN TRANSACTION
235          INSERT INTO [Dim].[Country] ...;
245
246          SET @CountryID = SCOPE_IDENTITY();
247
248          INSERT INTO [Dim].[HumanDevelopmentIndex]...;
268
269      COMMIT TRANSACTION;
270
271  END TRY
272  BEGIN CATCH
273      ROLLBACK TRANSACTION;
274
275      SELECT
276          @ErrorNumber    = ERROR_NUMBER(),
277          @ErrorMessage   = ERROR_MESSAGE(),
278          @ErrorSeverity  = ERROR_SEVERITY(),
279          @ErrorProcedure = ERROR_PROCEDURE()
280
281      EXEC ref.uspLogError...
286
287  END CATCH;
288
```

Useful Scenario

- Retry for Deadlock

```
25  DECLARE @TryCount TINYINT
26
27  SET @TryCount = 1;
28
29  WHILE @TryCount <= 3
30  BEGIN
31    BEGIN TRANSACTION
32    BEGIN TRY
33      INSERT INTO [Dim].[Country] (CountryName, CountryCode, AuditID)
34          VALUES ('New Zealand', 'NZ', 1209);
35
36      WAITFOR DELAY '00:00:05';
37
38      SELECT CountryName, CountryCode, AuditID FROM [Dim].[Country] WHERE CountryName = 'New Zealand';
39
40      COMMIT;
41      BREAK;
42    END TRY
43    BEGIN CATCH
44      SELECT @TryCount AS TryCount, ERROR_NUMBER() AS ErrorNumber, ERROR_MESSAGE() AS ErrorMessage;
45      ROLLBACK;
46      SET @TryCount = @TryCount + 1;
47      CONTINUE;
48    END CATCH;
49  END
50
```

Useful Scenario

- Retry for Deadlock (output)

Results | Messages

	TryCount	ErrorNumber	ErrorMessage
1	1	1205	Transaction (Process ID 54) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.

	TryCount	ErrorNumber	ErrorMessage
1	2	1205	Transaction (Process ID 54) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.

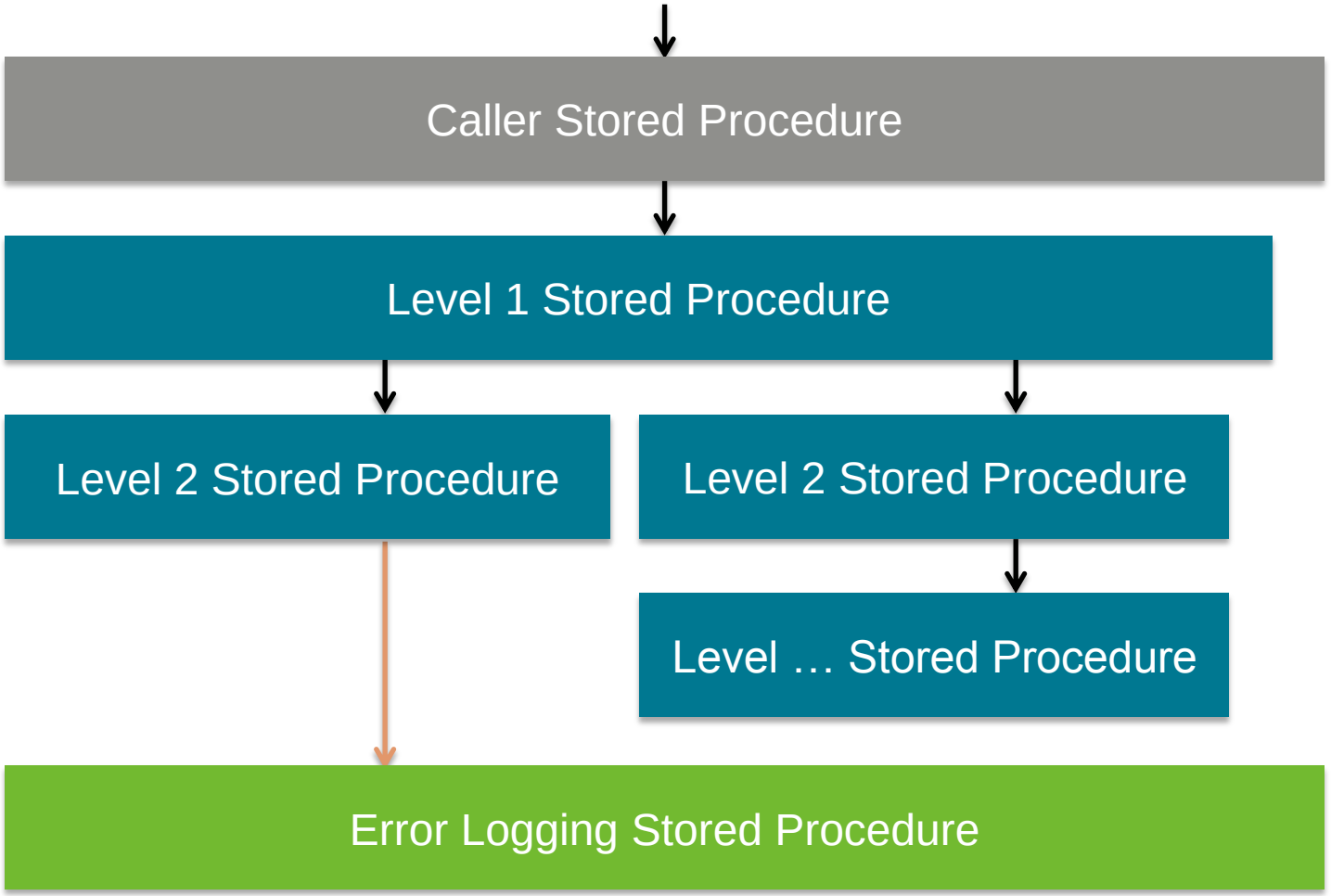
	TryCount	ErrorNumber	ErrorMessage
1	3	1205	Transaction (Process ID 54) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.

Coding Standard

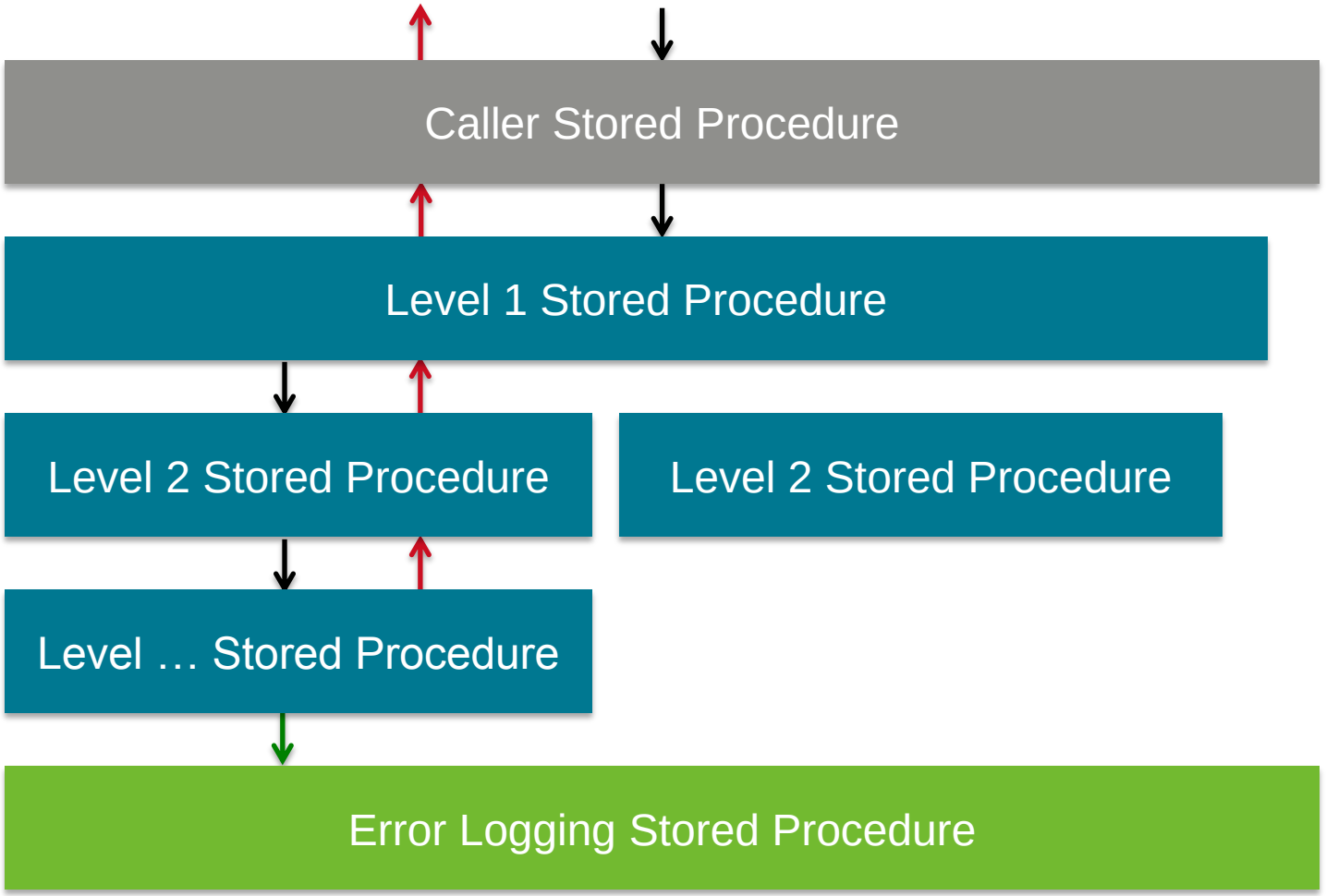
Best Practice – Error Handling

- Use TRY … CATCH (with THROW, if needed)
- Maintain complex data integrity
- Log errors
- Error recovery

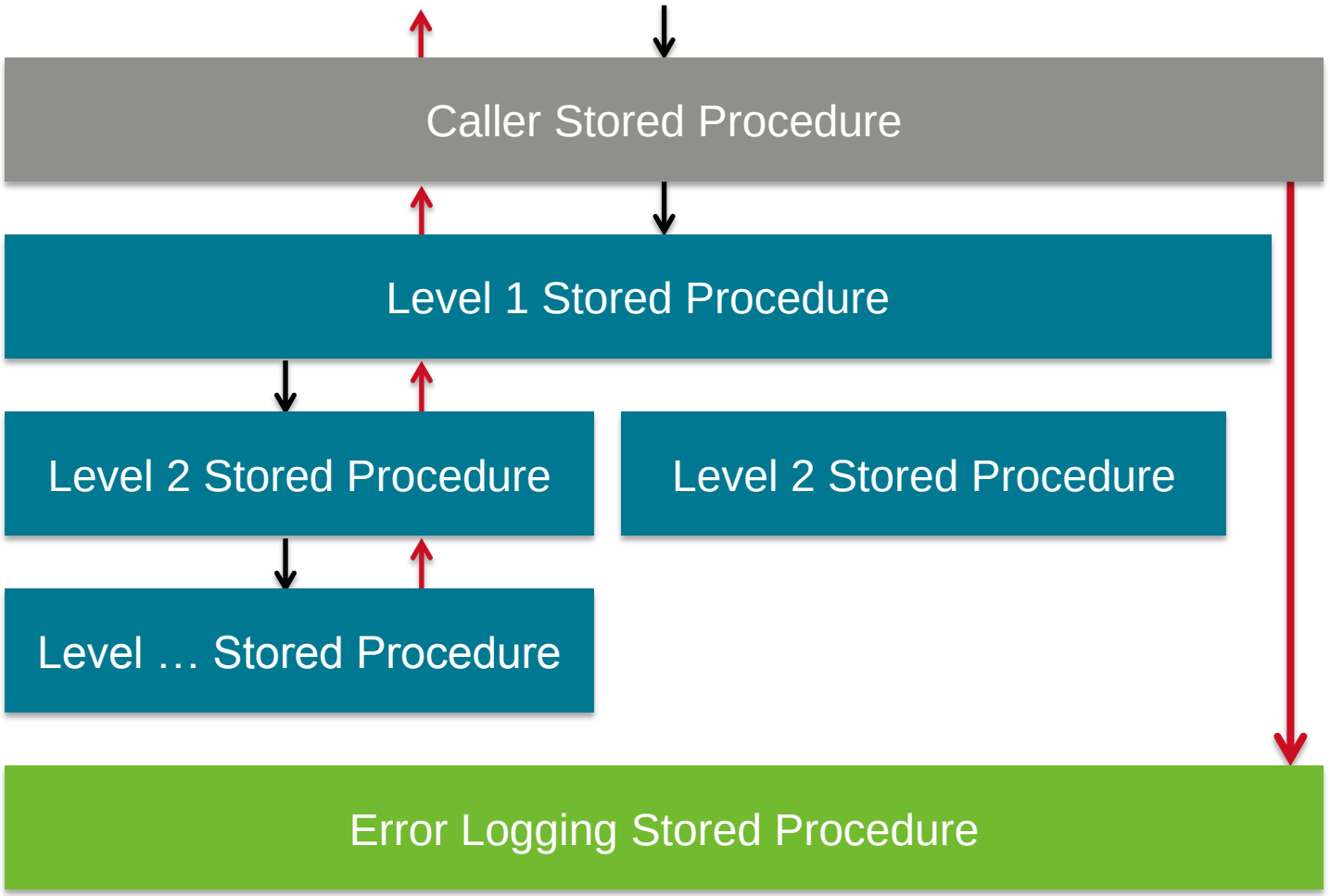
Contained


Caller Stored Procedure
Level 1 Stored Procedure
Level 2 Stored Procedure
Level 2 Stored Procedure
Level … Stored Procedure
Error Logging Stored Procedure

Bubble Up – Callee's Responsibility



Bubble Up – Caller's Responsibility



Best Practice – Error Handling

- Do it wisely
- Use re-throw over your own custom error if possible
- Avoid code obfuscation

Wrap Up

Summary

- TRY … CATCH construct
- Use THROW instead of RAISERROR
- Error Handling in Coding Standard

Further Reading

- Books Online:
  - TRY...CATCH (Transact-SQL) http://bit.ly/FBCsY
  - THROW http://bit.ly/WOczgm
  - RAISERROR http://bit.ly/49A9cn
  - Exception Handling (.Net 4) http://bit.ly/WOd0ae
- Throwing Errors in SQL Server 2012 – Leonard Lobel http://bit.ly/thumSq

Q & A


signal@mssqlgirl.com


http://www.mssqlgirl.com


http://au.linkedin.com/in/juliekoesmarno


@mssqlgirl

Thank you!