SqlHints.com

Tag Archives: RAISERROR Vs THROW

Differences Between RAISERROR and THROW in Sql Server

June 30, 2013 Differences, Exception Handling, Sql Server, Sql Server 2012 Difference Between RAISERROR and THROW, Difference Between THROW and RAISERROR, Exception Handling, Exception Handling Enhancements in Sql Server 2012, New Feature in Sql Server 2012, RAISEERROR, RAISERROR, RAISERROR Vs THROW, Sql Server, Sql Server 2005, SQL SERVER 2012, THROW, THROW Vs RAISERROR, TRY CATCH Basavaraj Biradar

Both RAISERROR and THROW statements are used to raise an error in Sql Server. The journey of **RAISERROR** started from **Sql Server 7.0**, where as the journey of **THROW** statement has just began with **Sql Server 2012**. obviously, Microsoft suggesting us to start using THROW statement instead of RAISERROR. THROW statement seems to be simple and easy to use than RAISERROR.

This is the third article in the series of articles on Exception Handling in Sql Server. Below is the complete list of articles in this series.

Part I: Exception Handling Basics – **MUST Read Article**
Part II: TRY…CATCH (Introduced in Sql Server 2005)
Part III: RAISERROR Vs THROW (Throw: Introduced in Sql Server 2012)
Part IV: Exception Handling Template

Raiserror Vs Throw

Below table lists-out 10 major difference between RAISERROR and THROW with examples:

RAISERROR	THROW
Version of the Sql Server in which it is introduced?	
Introduced in SQL SERVER 7.0. And as per BOL, Microsoft is suggesting to start using THROW statement instead of RAISERROR in New Applications. RAISERROR can't be used in the Sql Server 2014's Natively compiled Stored Procedures.	Introduced in SQL SERVER 2012. THROW statement seems to be simple and easy to use than RAISERROR. THROW statement can be used in the Sql Server 2014's Natively Compiled Stored Procedure.
SYNTAX	
`RAISERROR ( { error_number | message | @local_variable } { ,severity ,state } [ ,argument [ ,...n ] ] ) [ WITH option [ ,...n ] ]`	`THROW [ { error_number | @local_variable }, { message | @local_variable }, { state | @local_variable } ] [ ; ]`
Can re-throw the original exception that invoked the CATCH block?	
NO. It always generates new exception and results in the loss of the original exception details. Below example demonstrates this: `BEGIN TRY DECLARE @result INT --Generate divide-by-zero error SET @result = 55/0 END TRY`	**YES.** To Re-THROW the original exception caught in the TRY Block, we can just specify the THROW statement without any parameters in the CATCH block. Below example demonstrates this: `BEGIN TRY DECLARE @result INT --Generate divide-by-zero error`

```
BEGIN CATCH
--Get the details of the error
--that invoked the CATCH block
 DECLARE
   @ErMessage NVARCHAR(2048),
   @ErSeverity INT,
   @ErState INT

 SELECT
   @ErMessage = ERROR_MESSAGE(),
   @ErSeverity = ERROR_SEVERITY(),
   @ErState = ERROR_STATE()

 RAISERROR (@ErMessage,
            @ErSeverity,
            @ErState )
END CATCH
```

RESULT:

Msg **50000**, Level 16, State 1, **Line 19**

Divide by zero error encountered.

NOTE:*The actual line number of the code which generated Divided By Zero error here is 4, but the exception message returned by RAISERROR is showiung it as 19. Also the error number corresponding to divide by zero error is 8134 in the SYS.Messages table, but the one returned by RAISERROR is 50000.*

```
  SET @result = 55/0
END TRY
BEGIN CATCH
    THROW
END CATCH
```

RESULT:

Msg 8134, Level 16, State 1, Line 4

Divide by zero error encountered.

With above example it is clear that THROW statement is very simple for RE-THROWING the exception. And also it returns correct error number and line number.

Causes the statement batch to be ended?

Example 1: In the below Batch of statements the PRINT statement after RAISERROR statement will

Example 1: In the below Batch of statements the PRINT statement after THROW statement will not

be executed.

```
BEGIN
 PRINT 'BEFORE RAISERROR'
 RAISERROR('RAISERROR TEST',16,1)
 PRINT 'AFTER RAISERROR'
END
```

RESULT:

BEFORE RAISERROR

Msg 50000, Level 16, State 1, Line 3

RAISERROR TEST

AFTER RAISERROR

Example 2: In the below example all the statement's after RAISERROR statement are executed.

```
BEGIN TRY
 DECLARE @RESULT INT = 55/0
END TRY
BEGIN CATCH
 PRINT 'BEFORE RAISERROR';

--Get the details of the error
--that invoked the CATCH block
 DECLARE
  @ErMessage NVARCHAR(2048),
  @ErSeverity INT,
  @ErState INT

 SELECT
  @ErMessage = ERROR_MESSAGE(),
  @ErSeverity = ERROR_SEVERITY(),
  @ErState = ERROR_STATE()
```


executed.

```
BEGIN
    PRINT 'BEFORE THROW';
    THROW 50000,'THROW TEST',1
    PRINT 'AFTER THROW'
END
```

RESULT:

BEFORE THROW

Msg 50000, Level 16, State 1, Line 3

THROW TEST

Example 2: In the below example no PRINT statement's after THROW statement are executed.

```
BEGIN TRY
  DECLARE @RESULT INT = 55/0
END TRY
BEGIN CATCH
  PRINT 'BEFORE THROW';
  THROW;
  PRINT 'AFTER THROW'
END CATCH
  PRINT 'AFTER CATCH'
```

RESULT:

BEFORE THROW

Msg 8134, Level 16, State 1, Line 2

Divide by zero error encountered.

```
 RAISERROR (@ErMessage,
            @ErSeverity,
            @ErState )

 PRINT 'AFTER RAISERROR'
END CATCH
 PRINT 'AFTER CATCH'
```

RESULT:

BEFORE RAISERROR

Msg 50000, Level 16, State 1, Line 19

Divide by zero error encountered.

AFTER RAISERROR

AFTER CATCH

CAN SET SEVERITY LEVEL?	
YES. The severity parameter specifies the severity of the exception.	**NO.** There is no severity parameter. The exception severity is always set to 16. (unless re-throwing in a CATCH block)
Requires preceding statement to end with semicolon (;) statement terminator?	
NO.	**YES.** The statement before the THROW statement must be followed by the semicolon (;) statement terminator.
CAN RAISE SYSTEM ERROR MESSAGE? The SYS.MESSAGES Table will have both system-defined and user-defined messages. Message IDs less than 50000 are system messages.	

YES. With RAISERROR we can raise the System Exception. Example: RAISERROR (40655,16,1)**RESULT:** Msg 40655, Level 16, State 1, Line 1 Database 'master' cannot be restored.	**NO.** With THROW we can't raise the System Exception. But when it used in CATCH BLOCK it can Re-THROW the system exception.Example: Trying to raise system exception (i.e. exception with ErrorNumber less than 50000). THROW 40655, 'Database master cannot be restored.', 1 **RESULT:** Msg 35100, Level 16, State 10, Line 1 Error number 40655 in the THROW statement is outside the valid range. Specify an error number in the valid range of 50000 to 2147483647
CAN RAISE user-defined message with message_id greater than 50000 which is not defined in SYS.MESSAGES table?	
NO. If a msg_id is passed to RAISERROR, the ID must be defined in sys.messages.Example: RAISERROR (60000, 16, 1) **RESULT:** Msg 18054, Level 16, State 1, Line 1 Error 60000, severity 16, state 1 was raised, but no	**YES.** The error_number parameter does not have to be defined in sys.messages.Example: THROW 60000, 'Test User Defined Message', 1**RESULT:** Msg 60000, Level 16, State 1, Line 1 Test User Defined Message

message with that error number was found in sys.messages. If error is larger than 50000, make sure the user-defined message is added using sp_addmessage. Now add the Message to SYS.MESSAGES Table by using the below statement: EXEC sys.sp_addmessage 60000, 16, 'Test User Defined Message' Now try to Raise the Error: RAISERROR (60000, 16, 1) **RESULT:** Msg 60000, Level 16, State 1, Line 1 Test User Defined Message	
Allows substitution parameters in the message parameter? By using the below statement add a sample test message with parameteres to the SYS.Messages Table: **EXEC sp_addmessage 70000,16,'Message with Parameter 1: %d and Parameter 2:%s'**	
YES. The msg_str parameter can contain **printf** formatting styles.Example 1:	**NO.**The message parameter does not accept **printf** style formatting.Example 1:

```
RAISERROR (70000, 16, 1, 505,'Basavaraj' )
```

RESULT:

```
Msg 70000, Level 16, State 1, Line 1
Message with Parameter 1: 505 and Parameter 2:Basavaraj
```

```
THROW 70000, 'Message with Parameter 1: %d and Parameter 2:%s', 1, 505,'Basavaraj'
```

RESULT:

```
Msg 102, Level 15, State 1, Line 1
Incorrect syntax near ','.
```

Alternative Way of doing this is:

```
DECLARE @ErrorMsg NVARCHAR(2048) = FORMATMESSAGE(70000, 505, 'Basavaraj' );
THROW 70000, @ErrorMsg, 1
```

Example 2: Message manipulation is not allowed in the THROW statement

Below statement will fail

```
THROW 58000,'String1' + ' String2',1
```

RESULT:

```
Msg 102, Level 15, State 1, Line 1
Incorrect syntax near '+'.
```

We can solve such problems, we can prepare the message prior to the THROW statement and then

	pass it to throw statement as a variable. Below example illustrates this. DECLARE @message NVARCHAR(2048) SET @message = 'String1' + ' String2'; THROW 58000, @message, 1 RESULT: Msg 58000, Level 16, State 1, Line 3 String1 String2

RAISERROR WITH NOWAIT statement can also be used to flushes all the buffered PRINT/SELECT Statement Messages within a batch.

[ALSO READ] You may like to read below other popular articles on differences

1. Varchar vs NVarchar

2. Varchar vs Varchar(MAX)

3. Char vs Varchar

4. Text vs Varchar(Max)

5. Union vs Union All

6. DateTime vs DateTime2

7. SET QUOTED_IDENTIFIER ON vs SET QUOTED_IDENTIFIER OFF