**How to Raise Errors Explicitly in SQL Server?**

Generally, errors are raised in a program for predefined reasons like dividing a number by zero, violation of primary key, violation of check, violation of referential integrity, etc. But if you want then you can also raise an error in your programs in two different ways.

They are as follows.

1. **Raiserror Statement**
2. **throw Statement (new feature of SQL Server 2012)**

**Raiserror Syntax: Raiserror (errorid/errormsg, SEVERITY, state)[with log]**

**throw Syntax: Throw errorid, errormsg, state**

**Example: Raise Error using RAISERROR statement in SQL Server.**

In the following stored Procedure, we raise an error when the division is 1 by using the **RAISERROR** statement.

**CREATE** **PROCEDURE** spDivideBy1(@No1 **INT**, @No2 **INT**)

**AS**

**BEGIN**

**DECLARE** @Result **INT**

**SET** @Result = 0

**BEGIN** **TRY**

**IF** @No2 = 1

**RAISERROR** ('DIVISOR CANNOT BE ONE', 16, 1)

**SET** @Result = @No1 / @No2

**PRINT** 'THE RESULT IS: '+**CAST**(@Result **AS** **VARCHAR**)

**END** **TRY**

**BEGIN** **CATCH**

**PRINT** ERROR\_NUMBER()

**PRINT** ERROR\_MESSAGE()

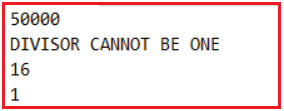
**PRINT** ERROR\_SEVERITY()

**PRINT** ERROR\_STATE()

**END** **CATCH**

**END**

**Example of execution: EXEC spDivideBy1 10, 1**



**Example: Raise Error using throw statement in SQL Server.**

The above procedure can also be rewritten with the help of a throw statement in place of Raiserror as following.

**ALTER** **PROCEDURE** spDivideBy2(@No1 **INT**, @No2 **INT**)

**AS**

**BEGIN**

**DECLARE** @Result **INT**

**SET** @Result = 0

**BEGIN** **TRY**

**IF** @No2 = 1

**THROW** 50001,'DIVISOR CANNOT BE ONE', 1

**SET** @Result = @No1 / @No2

**PRINT** 'THE RESULT IS: '+**CAST**(@Result **AS** **VARCHAR**)

**END** **TRY**

**BEGIN** **CATCH**

**PRINT** ERROR\_NUMBER()

**PRINT** ERROR\_MESSAGE()

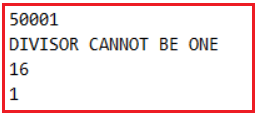
**PRINT** ERROR\_SEVERITY()

**PRINT** ERROR\_STATE()

**END** **CATCH**

**END**

**EXECUTION: EXEC spDivideBy2 10, 1**



**What is the difference between the RAISERROR function and the throw statement in SQL Server?**

If we use any of the two statements in a program for raising a custom error without try and catch blocks, the RAISERROR statement after raising the error will still continue the execution of the program whereas the throw statement will terminate the program abnormally on that line. But if they are used under try block both will behave in the same way that it will jump directly to catch block from where the error got raised.

The RAISERROR statement will give an option of specifying the ERROR SEVERITY Level of the error message whereas we don’t have this option in the case of the throw statement where all error messages will have a default  ERROR SEVERITY level as 16.

In the case of RAISERROR, there is a chance of recording the error message into the server log file by using the with log option whereas we cannot do this in case of a throw statement.

In the case of throw, we need to specify both the errorid and error message to raise the error whereas in the case of RAISERROR we can specify either id or message. If the id is not specified default error id is 50000 but if we want to specify only the error id first we need to add the error message in the sysmessage table by specifying a unique id to the table.

**OPTIONS WITH RAISERROR STATEMENT:**

**With Log:**By using this option in the RAISERROR statement we can record the error message in the SQL Server log file so that if the errors are fatal database administrator can take care of fixing those errors. If the severity of the error is greater than 20 specifying the With Log option is mandatory. To test this ALTER the procedure spDivideBy1 by changing the raiserror statement as following

**RAISERROR (‘DIVISOR CANNOT BE ONE’, 16, 1) WITH LOG**

**Below is the complete procedure**

**ALTER** **PROCEDURE** spDivideBy1(@No1 **INT**, @No2 **INT**)

**AS**

**BEGIN**

**DECLARE** @Result **INT**

**SET** @Result = 0

**BEGIN** **TRY**

**IF** @No2 = 1

**RAISERROR** ('DIVISOR CANNOT BE ONE', 16, 1) **WITH** **LOG**

**SET** @Result = @No1 / @No2

**PRINT** 'THE RESULT IS: '+**CAST**(@Result **AS** **VARCHAR**)

**END** **TRY**

**BEGIN** **CATCH**

**PRINT** ERROR\_NUMBER()

**PRINT** ERROR\_MESSAGE()

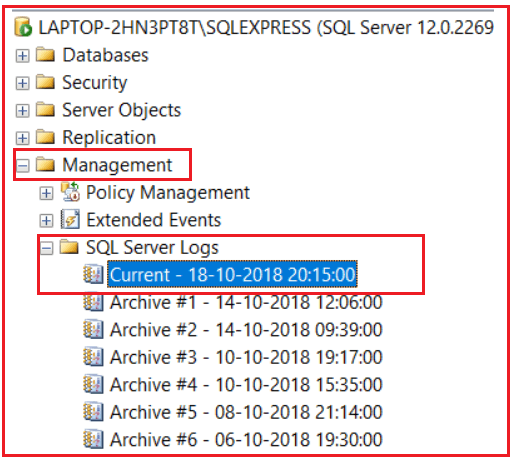
**PRINT** ERROR\_SEVERITY()

**PRINT** ERROR\_STATE()

**END** **CATCH**

**END**

Now execute the procedure and whenever the given error raises we can watch the error messages recorded under the SQL Server log file. To view the log file. In object explorer, go to the management node, then open SQL Server logs node and open the current log file by double-clicking on it as shown below.



**Using substitutional parameters in the error message of RAISERROR:**

Just like C language, we can also substitute values into the error message to make the error message as dynamic as following

**RAISERROR (‘THE NUMBER %d CANNOT BE DIVIDED BY %d’,16, 1, @No1, @No2)WITH LOG**

**Raising Errors By storing the Error message in the SysMessage table:**

We can raise an error without giving the error message in the RAISERROR statement but in place of the error message we need the specify the error id and to specify the error id we need to record that error id with the error message in the SysMessage table by using the system defined procedure “**SP\_ADDMESSAGE**”.

**Syntax: SP\_ADDMESSAGE <error id>, <severity>, <error message>**

**To test this first add a record to sysmessage table as following**

**EXEC sp\_Addmessage 51000, 16, ‘DIVIDE BY ONE ERROR ENCOUNTERED’**

**Now alter the procedure by changing the RAISERROR statement as following**

**RAISERROR (51000,16, 1)WITH LOG**

**Deleting the error messages from sysmessages table:**

**Syntax: SP\_DROPMESSAGE <error id>**

**Example: EXEC sp\_dropMessage 51000**