Error Handling in SQL Server

Here's how error handling in SQL Server works. In SQL Server you can take advantage of TRY...CATCH statements to handle errors. When writing code that handles errors, you should have a TRY block and a CATCH block immediately after it. The TRY block starts with a BEGIN TRY statement and ends with an END TRY statement. Similarly, the CATCH block starts with a BEGIN CATCH statement and ends with an END CATCH statement.

Assume that there are a group of statements in the TRY block, i.e., enclosed between the BEGIN TRY...END TRY statements. As soon as an error occurs in this TRY block, the control moves to the CATCH block where you might have another group of statements for handling errors. In this regard, the following points should be noted:

A TRY block should be immediately be followed by a CATCH block where the error handling code resides. Here's an example code that illustrates this.

BEGIN TRY

-- Write statements here that may cause exception

END TRY

BEGIN CATCH

-- Write statements here to handle exception

END CATCH

When an error occurs inside the TRY block, the control moves to the first statement inside the CATCH block. On the contrary, if the statements inside a TRY block have completed execution successfully without an error, the control will not flow inside the CATCH block. Rather, the first statement immediately after the END CATCH statement will then be executed.

Retrieving detailed information on the error

You can take advantage of various functions inside the CATCH block to get detailed information about an error.

These functions include the following:

* ERROR\_MESSAGE() - you can take advantage of this function to get the complete error message.
* ERROR\_LINE() - this function can be used to get the line number on which the error occurred.
* ERROR\_NUMBER() - this function can be used to get the error number of the error.
* ERROR\_SEVERITY() - this function can be used to get the severity level of the error.
* ERROR\_STATE() - this function can be used to get the state number of the error.
* ERROR\_PROCEDURE() - this function can be used to know the name of the stored procedure or trigger that has caused the error.

Programming TRY…CATCH Blocks in SQL Server

Note that you cannot use TRY...CATCH blocks inside T-SQL UDFs. If you have to capture errors that occur inside a UDF, you can do that in the calling procedure or code.

Now consider the following code snippet that illustrates how an error generated inside a TRY block is handled in the CATCH block and the relevant error metadata displayed.

BEGIN TRY

Insert Into Categories(CategoryID, CategoryName, Description, Picture) Values (9, 'Test', 'Test Description', 'Test')

END TRY

BEGIN CATCH

SELECT ERROR\_MESSAGE() AS [Error Message]

,ERROR\_LINE() AS ErrorLine

,ERROR\_NUMBER() AS [Error Number]

,ERROR\_SEVERITY() AS [Error Severity]

,ERROR\_STATE() AS [Error State]

END CATCH

Once you run the code block shown above, here’s how the output looks in SQL Server Management Studio (SSMS).

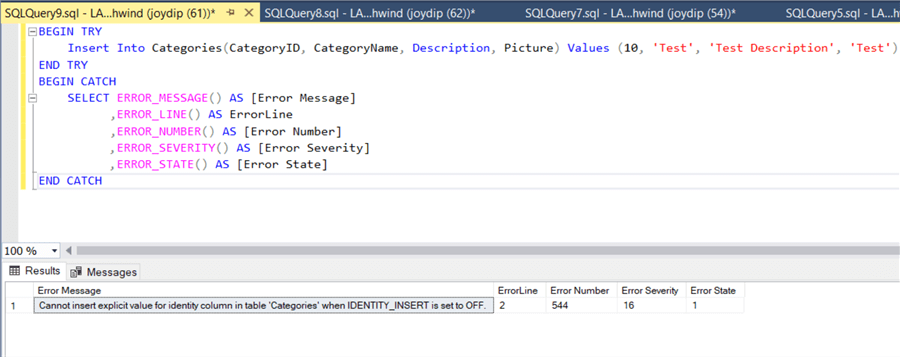


Figure 1

As evident from the error shown in Figure 1, you are not allowed to insert a value in an identity column whose IDENTITY\_INSERT is set to OFF.

Let's try to delete a record now.

We’ll attempt to delete a record in the Employees table. Since EmployeeID is a ForeignKey in the Orders table and the EmployeeID being deleted has already been used there, the Delete statement should fail. The following code snippet illustrates this.

BEGIN TRY

Delete FROM [northwind].[dbo].[Employees] Where [EmployeeID] = 9

END TRY

BEGIN CATCH

SELECT ERROR\_MESSAGE() AS [Error Message]

,ERROR\_LINE() AS ErrorLine

,ERROR\_NUMBER() AS [Error Number]

,ERROR\_SEVERITY() AS [Error Severity]

,ERROR\_STATE() AS [Error State]

END CATCH

When you execute the above code, the error metadata is captured as shown in Figure 2.

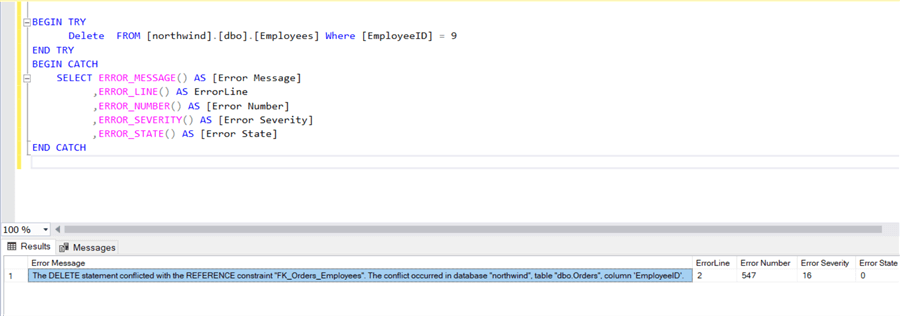


Figure 2

Error Propagation

When an error occurs inside a TRY block in SQL Server, the statements that immediately follow aren’t executed as the control enters the CATCH block and the first statement inside the CATCH block is executed. Consider the following code snippet:

BEGIN TRY

DECLARE @x int

SELECT @x = 1/0

PRINT 'This statement will not be executed'

END TRY

BEGIN CATCH

PRINT 'The error message is: ' + error\_message()

END CATCH

When you execute the preceding code snippet, you'll observe that the statement immediately after the SELECT statement is not executed. The reason is because an exception occurs in the previous statement, i.e., in the line having the SELECT statement. As soon as the exception occurs, the control moves inside the CATCH block and the PRINT statement inside it is executed. Figure 3 shows how the output looks when the code snippet is executed in SQL Server Management Studio.

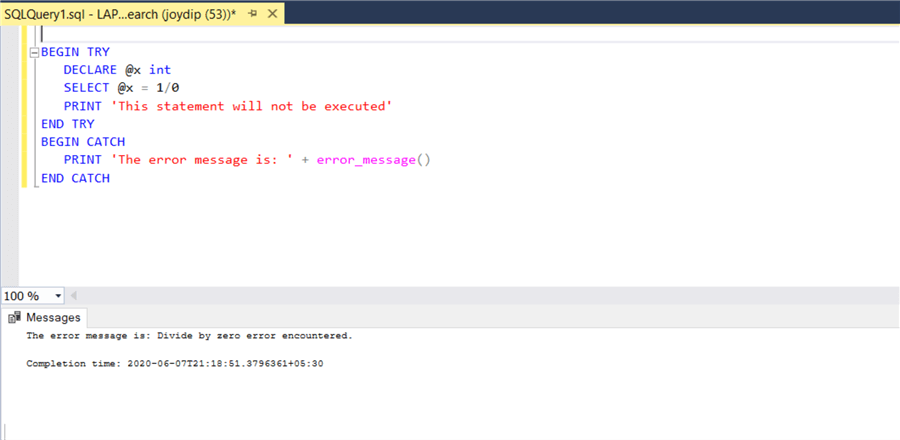


Figure 3

Using Nested TRY...CATCH Constructs

TRY...CATCH blocks can be nested as well. Nested TRY...CATCH blocks enable you to handle specific errors. As an example, assume that you have to insert multiple records in a database table and you want to know the failed insert statements. A nested TRY...CATCH block is one which resides inside another TRY...CATCH block, i.e., you have an outer TRY…CATCH block and an inner TRY…CATCH block.

The following code snippet illustrates how this can be accomplished.

BEGIN TRY

--- Write statements here that might cause exceptions

END TRY

BEGIN CATCH

-- Write statements here to handle the exception

BEGIN TRY

--- This is a nested TRY block. Write statements here that might cause exceptions

END TRY

BEGIN CATCH

-- This is a nested CATCH block. Write statements here to handle the exception

END CATCH

END CATCH

The THROW Statement

The THROW statement in SQL Server raises an exception and transfers the control to a CATCH block. The following code snippet shows the syntax of the THROW statement.

THROW [error\_number, message, state];

If you don't specify any parameters, then the THROW statement should be placed inside a CATCH block to raise the error that has been handled by the CATCH block. The following code snippet illustrates how you can work with the THROW statement.

BEGIN TRY

Delete FROM [northwind].[dbo].[Employees] Where [EmployeeID] = 9

END TRY

BEGIN CATCH

THROW 50000, N'Unable to delete record...', 1;

END CATCH

THROW vs RAISERROR

Note that both THROW and RAISERROR statements can be used to generate custom errors and re-throw exceptions. However, there are subtle differences between the two. While the THROW statement can re-throw the original exception that has been caught in the CATCH block, the RAISE ERROR statement re-throws an altogether new exception and the original exception is lost.