RAISERROR (Transact-SQL)

* Generates an error message and initiates error processing for the session.
* RAISERROR can either reference a user-defined message stored in the sys.messages catalog view or build a message dynamically.
* The message is returned as a server error message to the calling application or to an associated CATCH block of a TRY…CATCH construct.

Topic link iconRAISERROR ( { msg\_id | msg\_str | @local\_variable }

    { ,severity ,state })

    [ WITH option [ ,...n ] ]

Arguments

msg\_id

* Is a user-defined error message number stored in the sys.messages catalog view using sp\_addmessage. Error numbers for user-defined error messages should be greater than 50000.
* When msg\_id is not specified, RAISERROR raises an error message with an error number of 50000.

msg\_str

* Is a user-defined message with formatting similar to the printf function in the C standard library.
* The error message can have a maximum of 2,047 characters.
* When msg\_str is specified, RAISERROR raises an error message with an error number of 50000.

@local\_variable

* Is a variable of any valid character data type that contains a string formatted in the same manner as msg\_str.
* @local\_variable must be char or varchar, or be able to be implicitly converted to these data types.

severity

* Is the user-defined severity level associated with this message.
* When using msg\_id to raise a user-defined message created using sp\_addmessage, the severity specified on RAISERROR overrides the severity specified in sp\_addmessage.
* Severity levels from 0 through 18 can be specified by any user.
* Severity levels from 19 through 25 can only be specified by members of the sysadmin fixed server role.
* For severity levels from 19 through 25, the WITH LOG option is required.
* Severity levels from 20 through 25 are considered fatal. If a fatal severity level is encountered, the client connection is terminated after receiving the message, and the error is logged

in the error and application logs.

* Severity levels less than 0 are interpreted as 0. Severity levels greater than 25 are interpreted as 25

state

* Is an integer from 0 through 255. Negative values or values larger than 255 generate an error.
* If the same user-defined error is raised at multiple locations, using a unique state number for each location can help find which section of code is raising the errors.

option

Is a custom option for the error and can be one of the values in the following table.

|  |  |
| --- | --- |
| Value | Description |
| LOG | * Logs the error in the error log and the application log for the instance of the Microsoft SQL Server Database Engine. * Errors logged in the error log are currently limited to a maximum of 440 bytes. * Only a member of the sysadmin fixed server role or a user with ALTER TRACE permissions can specify WITH LOG. |
| NOWAIT | Sends messages immediately to the client. |
| SETERROR | Sets the @@ERROR and ERROR\_NUMBER values to msg\_id or 50000, regardless of the severity level. |

Additional Info

* The errors generated by RAISERROR operate the same as errors generated by the Database Engine code.
* The values specified by RAISERROR are reported by the ERROR\_LINE, ERROR\_MESSAGE, ERROR\_NUMBER, ERROR\_PROCEDURE, ERROR\_SEVERITY, ERROR\_STATE, and @@ERROR system functions.
* When RAISERROR is run with a severity of 11 or higher in a TRY block, it transfers control to the associated CATCH block. The error is returned to the caller if RAISERROR is run:

1. Outside the scope of any TRY block.
2. With a severity of 10 or lower in a TRY block.
3. With a severity of 20 or higher that terminates the database connection.

**Various severity levels.**

|  |  |
| --- | --- |
| Severity level | Description |
| 0-9 | Informational messages that return status information or report errors that are not severe. The Database Engine does not raise system errors with severities of 0 through 9. |
| 10 | Informational messages that return status information or report errors that are not severe. For compatibility reasons, the Database Engine converts severity 10 to severity 0 before returning the error information to the calling application. |
| 11-16 | Indicate errors that can be corrected by the user. |
| 11 | Indicates that the given object or entity does not exist. |
| 12 | A special severity for queries that do not use locking because of special query hints. In some cases, read operations performed by these statements could result in inconsistent data, since locks are not taken to guarantee consistency. |
| 13 | Indicates transaction deadlock errors. |
| 14 | Indicates security-related errors, such as permission denied. |
| 15 | Indicates syntax errors in the Transact-SQL command. |
| 16 | Indicates general errors that can be corrected by the user. |
| 17-19 | Indicate software errors that cannot be corrected by the user. Inform your system administrator of the problem. |
| 17 | Indicates that the statement caused SQL Server to run out of resources (such as memory, locks, or disk space for the database) or to exceed some limit set by the system administrator. |
| 18 | Indicates a problem in the Database Engine software, but the statement completes execution, and the connection to the instance of the Database Engine is maintained. The system administrator should be informed every time a message with a severity level of 18 occurs. |
| 19 | Indicates that a nonconfigurable Database Engine limit has been exceeded and the current batch process has been terminated. Error messages with a severity level of 19 or higher stop the execution of the current batch. Severity level 19 errors are rare and must be corrected by the system administrator or your primary support provider. Contact your system administrator when a message with a severity level 19 is raised. Error messages with a severity level from 19 through 25 are written to the error log. |
| 20-24 | Indicate system problems and are fatal errors, which means that the Database Engine task that is executing a statement or batch is no longer running. The task records information about what occurred and then terminates. In most cases, the application connection to the instance of the Database Engine may also terminate. If this happens, depending on the problem, the application might not be able to reconnect.  Error messages in this range can affect all of the processes accessing data in the same database and may indicate that a database or object is damaged. Error messages with a severity level from 19 through 24 are written to the error log. |
| 20 | Indicates that a statement has encountered a problem. Because the problem has affected only the current task, it is unlikely that the database itself has been damaged. |
| 21 | Indicates that a problem has been encountered that affects all tasks in the current database, but it is unlikely that the database itself has been damaged. |
| 22 | Indicates that the table or index specified in the message has been damaged by a software or hardware problem.  Severity level 22 errors occur rarely. If one occurs, run DBCC CHECKDB to determine whether other objects in the database are also damaged. The problem might be in the buffer cache only and not on the disk itself. If so, restarting the instance of the Database Engine corrects the problem. To continue working, you must reconnect to the instance of the Database Engine; otherwise, use DBCC to repair the problem. In some cases, you may have to restore the database.  If restarting the instance of the Database Engine does not correct the problem, then the problem is on the disk. Sometimes destroying the object specified in the error message can solve the problem. For example, if the message reports that the instance of the Database Engine has found a row with a length of 0 in a nonclustered index, delete the index and rebuild it. |
| 23 | Indicates that the integrity of the entire database is in question because of a hardware or software problem.  Severity level 23 errors occur rarely. If one occurs, run DBCC CHECKDB to determine the extent of the damage. The problem might be in the cache only and not on the disk itself. If so, restarting the instance of the Database Engine corrects the problem. To continue working, you must reconnect to the instance of the Database Engine; otherwise, use DBCC to repair the problem. In some cases, you may have to restore the database. |
| 24 | Indicates a media failure. The system administrator may have to restore the database. You may also have to call your hardware vendor. |

**Creating an ad hoc message in sys.messages**

The following example shows how to raise a message stored in the sys.messages catalog view. The message was added to the sys.messages catalog view by using the sp\_addmessage system stored procedure as message number 50005.

[-- add](javascript:CodeSnippet_CopyCode('CodeSnippetContainerCode4');" \o "Copy to clipboard.) message to sys.messages

sp\_addmessage @msgnum = 50005, @severity = 10, @msgtext = ‘sample error message';

GO

--Raise error

RAISERROR (50005, 10, 1,'abcde');

GO

--Drop error messahe from sys.messages

sp\_dropmessage @msgnum = 50005;

GO

**FORMATMESSAGE**

Constructs a message from an existing message in sys.messages. The functionality of FORMATMESSAGE resembles that of the RAISERROR statement. However, RAISERROR prints the message immediately, while FORMATMESSAGE returns the formatted message for further processing. The following example uses a replication message 20009 stored in sys.messages as, "The article '%s' could not be added to the publication '%s'." FORMATMESSAGE substitutes the values First Variable and Second Variable for the parameter placeholders. The resulting string, "The article 'First Variable' could not be added to the publication 'Second Variable'.", is stored in the local variable @var1.

SELECT text FROM sys.messages WHERE message\_id = 20009 AND language\_id = 1033;

DECLARE @var1 VARCHAR(200);

SELECT @var1 = FORMATMESSAGE(20009, 'First Variable', 'Second Variable');

SELECT @var1;

**THROW**

Raises an exception and transfers execution to a CATCH block of a TRY…CATCH construct in SQL Server 2012.

THROW [ { error\_number | @local\_variable },

{ message | @local\_variable },

{ state | @local\_variable } ]

[ ; ]

Remarks

1. The statement before the THROW statement must be followed by the semicolon (;) statement terminator.
2. If a TRY…CATCH construct is not available, the session is ended. The line number and procedure where the exception is raised are set. The severity is set to 16.
3. If the THROW statement is specified without parameters, it must appear inside a CATCH block. This causes the caught exception to be raised. Any error that occurs in a THROW statement causes the statement batch to be ended.

The following table lists differences between the RAISERROR and THROW statements.

|  |  |
| --- | --- |
| **RAISERROR statement** | **THROW statement** |
| If a *msg\_id*is passed to RAISERROR, the ID must be defined in sys.messages. | The *error\_number* parameter does not have to be defined in sys.messages. |
| The *msg\_str* parameter can contain **printf** formatting styles. | The *message* parameter does not accept **printf** style formatting. |
| The *severity* parameter specifies the severity of the exception. | There is no *severity* parameter. The exception severity is always set to 16. |

Using THROW to raise an exception

THROW 51000, 'The record does not exist.', 1;

Using THROW to raise an exception again

USE tempdb;

GO

CREATE TABLE dbo.TestRethrow

( ID INT PRIMARY KEY

);

BEGIN TRY

INSERT dbo.TestRethrow(ID) VALUES(1);

-- Force error 2627, Violation of PRIMARY KEY constraint to be raised.

INSERT dbo.TestRethrow(ID) VALUES(1);

END TRY

BEGIN CATCH

PRINT 'In catch block.';

THROW;

END CATCH;

Using FORMATMESSAGE with THROW

EXEC sys.sp\_addmessage

@msgnum = 60000

,@severity = 16

,@msgtext = N'This is a test message with one numeric parameter (%d), one string parameter (%s), and another string parameter (%s).'

,@lang = 'us\_english';

GO

DECLARE @msg NVARCHAR(2048) = FORMATMESSAGE(60000, 500, N'First string', N'second string');

THROW 60000, @msg, 1;

select

error\_number() as errornumber

,error\_severity() as errorseverity

,error\_state() as errorstate

,error\_procedure() as errorprocedure

,error\_line() as errorline

,error\_message() as errormessage;