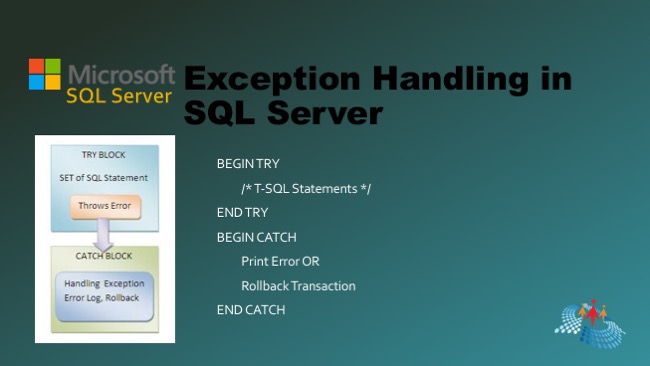
**Exception Handling in SQL Server**

Introduction

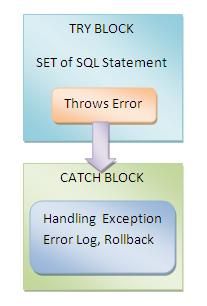
This article will teach us how to implement exception handling in SQL Server. [SQL Server](https://www.c-sharpcorner.com/topics/sql-server) provides TRY and CATCH blocks for exception handling.

Exception Handling in SQL Server

An error condition during program execution is called an exception, and the mechanism for resolving such an exception is known as exception handling. We can put all T-SQL statements into a TRY BLOCK, and the code for exception handling can be put into a CATCH block. We can also generate user-defined errors using a THROW block.



Syntax of Exception Handling



*BEGIN TRY*

*/\* T-SQL Statements \*/*

*END TRY*

*BEGIN CATCH*

*- Print Error OR*

*- Rollback Transaction*

*END CATCH*

In exception handling, all T-SQL statements are put into a try block. If all statements execute without any error, then everything is OK else. Control will go to the catch block.

Types of SQL Server Exceptions

SQL Server contains the following two types of exceptions:

1. System Defined
2. User Defined

System Defined Exception

In a System Defined Exception, the system generates exceptions (errors).

**Example**

Declare @val1 int;

Declare @val2 int;

BEGIN TRY

Set @val1=8;

Set @val2=@val1/0; /\* Error Occur Here \*/

END TRY

BEGIN CATCH

Print 'Error Occur that is:'

Print Error\_Message()

END CATCH

SQL

**Output**

* *Error Occur that is:*
* *Divide by zero error encountered*

User Defined Exception

This type of exception is user-generated, not system generated.

Declare @val1 int;

Declare @val2 int;

BEGIN TRY

Set @val1=8;

Set @val2=@val1%2;

if @val1=1

Print ' Error Not Occur'

else

Begin

Print 'Error Occur';

Throw 60000,'Number Is Even',5

End

END TRY

BEGIN CATCH

Print 'Error Occur that is:'

Print Error\_Message()

END CATCH

SQL

**Output**

* *Error Occur*
* *Error Occur that is:*
* *Number Is Even*

Here 60000 denotes the error number, and 5 denotes the state to associate with the message.

The following are system functions and the keyword used within a catch block:

@@ERROR

ERROR\_NUMBER()

ERROR\_STATE()

ERROR\_LINE()

ERROR\_MESSAGE()

ERROR\_PROCEDURE()

ERROR\_SEVERITY()

RAISERROR()

GOTO()

SQL

Now we will see some examples to help understand all these functions and keywords.

First, create a table and enter some value into the table as in the following:

Create TABLE Employee

(

Emp\_IId Int identity(1,1),

First\_Name Nvarchar(MAX) Not NUll,

Last\_Name Nvarchar(MAX) Not Null,

Salary Int Not Null check(Salary>20000),

City Nvarchar(Max) Not Null

)

SQL

Insert data into Employee.

Select 'Pankaj','Choudhary',25000,'Alwar' Union All

Select 'Rahul','Prajapat',23000,'Alwar' Union All

Select 'Sandeep','Jangid',27000,'Alwar' Union All

Select 'Sanjeev','Baldia',24000,'Alwar' Union All

Select 'Neeraj','Saini',22000,'Alwar' Union All

Select 'Narendra','Sharma',23000,'Alwar' Union All

Select 'Divyanshu','Gupta',25000,'Alwar'

SQL

Now execute a select command.

Select \* From Employee

SQL



**Example 1. (@@ERROR)**

@@ERROR returns the error number for the last executed T-SQL statements. It returns 0 if the previous Transact-SQL statement encountered no errors; else returns an error number.

Update Employee set Salary=19000 Where Emp\_IID=5

IF @@ERROR = 547

PRINT 'A check constraint violation occurred.';

SQL

**Output**

* *Msg 547, Level 16, State 0, Line 1*
* *The UPDATE statement conflicted with the CHECK constraint "CK\_\_Employee\_\_Salary\_\_68487DD7". The conflict occurred in database "Home\_Management," table "dbo. Employee", column 'Salary.'*

The statement has been terminated.

**A check constraint violation occurred.**

**Example 2. (ERROR\_NUMBER)**

ERROR\_NUMBER() returns the error number that caused the error. It returns zero if called outside the catch block.

BEGIN TRY

Update Employee set Salary=19000 Where Emp\_IID=5

END TRY

BEGIN CATCH

SELECT ERROR\_NUMBER() AS ErrorNumber;

END CATCH;

GO

SQL

**Output**

catch block

Now a question develops of what is diff @@ERROR and ERROR\_NUMBER. Let me explain.

1. ERROR\_NUMBER can only be used in a catch block; outside a catch block, it returns Null, but @@ERROR can be used inside or outside the catch block (see Example 1).
2. ERROR\_NUMBER is a contrast to @@ERROR, that only returns the error number in the statement immediately after the one that causes an error or the first statement of a CATCH block.

Now we will see an example and observe the differences between them.

BEGIN TRY

Update Employee set Salary=19000 Where Emp\_IID=5

END TRY

BEGIN CATCH

SELECT ERROR\_NUMBER() AS ErrorNumber;

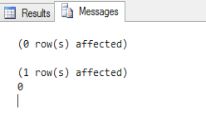
print @@ERROR

END CATCH;

GO

SQL

**Output**



BEGIN TRY

Update Employee set Salary=19000 Where Emp\_IID=5

END TRY

BEGIN CATCH

print @@ERROR

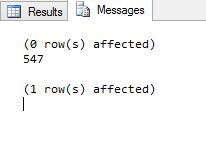
SELECT ERROR\_NUMBER() AS ErrorNumber;

END CATCH;

GO

SQL

**Output**



**Example 3. (ERROR\_MESSAGE)**

ERROR\_MESSAGE returns the message text of the error that caused the error. The return type of ERROR\_MESSAGE is **nvarchar(4000)**.

BEGIN TRY

Update Employee set Salary=19000 Where Emp\_IID=5

END TRY

BEGIN CATCH

SELECT ERROR\_MESSAGE() AS ErrorMsg;

END CATCH;

GO

SQL

**Output**

* *The UPDATE statement conflicted with the CHECK constraint "CK\_\_Employee\_\_Salary\_\_68487DD7". The conflict occurred in database "Home\_Management," table "dbo.Employee", column 'Salary.'*

**Example 4. ( ERROR\_STATE)**

ERROR\_STATE returns the state number of the error. The return type of ERROR\_STATE is **INT**.

BEGIN TRY

SELECT SALARY + First\_Name From Employee Where Emp\_IID=5

END TRY

BEGIN CATCH

SELECT ERROR\_STATE() AS ErrorState , ERROR\_MESSAGE() ErrorMsg ;

END CATCH;

GO

SQL

**Output**

INT

**Example 5. (ERROR\_LINE)**

ERROR\_LINE returns the line number at which an error occurred. The return type of ERROR\_LINE is **INT**.

BEGIN TRY

SELECT SALARY + First\_Name From Employee Where Emp\_IID=5

END TRY

BEGIN CATCH

SELECT ERROR\_STATE() AS ErrorLine;

END CATCH;

GO

SQL

**Output**

ERROR LINE

**Example 6. (ERROR\_PROCEDURE)**

ERROR\_PROCEDURE returns the name of the Stored Procedure or trigger where an error occurred. The return type of ERROR\_PROCEDURE is **nvarchar(128)**.

**Return value**

Return value returns the Stored Procedure Name if an error occurs in a Stored Procedure or trigger and the catch block is called.

It returns **NULL** if the error did not occur within a Stored Procedure or trigger or if it is called outside the scope of a CATCH block.

First, we create a [Stored Procedure](https://www.c-sharpcorner.com/topics/sql-server).

Create Procedure My\_Proc

AS

begin

BEGIN TRY

SELECT SALARY + First\_Name From Employee Where Emp\_IID=5

END TRY

BEGIN CATCH

SELECT ERROR\_PROCEDURE() AS ProcName;

END CATCH;

END

SQL

Now execute this Stored Procedure.

Exec My\_Proc

SQL

**Output**

execute this Stored Procedure

**Example 7. (ERROR\_SEVERITY)**

ERROR\_SEVERITY returns the severity of the error. The return type of ERROR\_SEVERITY is **INT**.

BEGIN TRY

SELECT SALARY + First\_Name From Employee Where Emp\_IID=5

END TRY

BEGIN CATCH

SELECT ERROR\_SEVERITY() AS ErrorSeverity;

END CATCH;

SQL

**Output**



The severity level of an error message indicates the type of problem that Microsoft® SQL Server encountered. In the preceding example, the Severity Level is 16. That means that the user can remove the error.

Some important severity levels are:

|  |  |
| --- | --- |
| 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 the user can correct. |

**Example 8. (RAISERROR)**

RAISEERROR generates an error message and initiates error processing for the session.

BEGIN TRY

SELECT SALARY + First\_Name From Employee Where Emp\_IID=5

END TRY

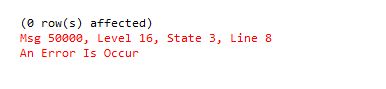
BEGIN CATCH

RAISERROR(N'An Error Is Occur',16,3);

END CATCH;

SQL

**Output**



In RAISERROR(N'An Error Is Occur',16,3), the first argument represents the error message, the second represents the Severity Level, and the last represents the Error State.

**Example 9. (GOTO)**

GOTO causes a jump to a specific step or statement. It alters the flow of execution to a label. We declare some labels in batches and alter we can move to a specific label. GOTO can exist within a conditional control-of-flow statement, statement blocks, or procedures but cannot go to a label outside the batch. GOTO cannot be used to jump into a TRY or CATCH scope.

Declare @Var Int;

Set @Var=1

Print 'Goto exercise'

If @Var%2=0

GOTO Label1;

else

GOTO Label2;

Set @Var=@Var+1;

Label1:

Print 'Var Is Odd'

Label2:

Print 'Var Is Even'

SQL

**Output**

* *Goto exercise*
* *Var Is Even*

**Example 10**

BEGIN TRY

SELECT SALARY + First\_Name From Employee Where Emp\_IID=5

END TRY

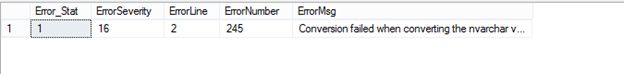
BEGIN CATCH

SELECT ERROR\_STATE() AS Error\_Stat,ERROR\_SEVERITY() AS ErrorSeverity, ERROR\_LINE() as ErrorLine, ERROR\_NUMBER() AS ErrorNumber, ERROR\_MESSAGE() AS ErrorMsg;

END CATCH;

SQL

**Output**



**Exercise 11. (Transaction Management)**

Exception handling is mainly used for Transaction Management. Let us see an example.

Begin Transaction Trans

Begin Try

Delete From Employee Where Employee.Emp\_IID<3

Update Employee Set Employee.First\_Name='Pankaj kumar' Where Employee.Emp\_IID='6th' /\* Error Will Occur Here \*/

If @@TranCount>0

begin Commit Transaction Trans

End

End Try

Begin Catch

if @@TranCount>0

Print 'Error Is Occur in Transaction'

begin Rollback Transaction Trans

End

End Catch

Select \* From Employee

SQL

**Output**



When to use Exception Handling?

1. In Transaction Management to Rollback the transaction.
2. While using cursors in SQL Server.
3. When implementing a DML Query (insert, update, or delete) for checking and handling the error.

Conclusion

This article taught us about Exception Handling in [SQL Server](https://www.c-sharpcorner.com/topics/sql-server)with different types of exceptions and handling these exceptions by using TRY-Catch blocks.