Data Management Using Microsoft SQL Server

Session: 15

Error Handling

SQL Server 2012

Objectives

- Explain the various types of errors
- Explain error handling and the implementation
- Describe the TRY-CATCH block
- Explain the procedure to display error information
- Describe the @@ERROR and RAISERROR statements
- Explain the use of ERROR_STATE, ERROT_SEVERITY, and ERROR_PROCEDURE
- Explain the use of ERROR_NUMBER, ERROR_MESSAGE, and ERROR_LINE
- Describe the THROW statement

Introduction

- Error handling in SQL Server has become easy through a number of different techniques such as:
  - SQL Server provides the `TRY…CATCH` statement that helps to handle errors effectively at the back end.
  - SQL Server also provides a number of system functions that print error related information, which can help fix errors easily.

SQL Server 2012

Types of Errors 1-2

A Transact-SQL programmer must identify the type of the error and then determine how to handle or overcome it.

Some of the types of errors are as follows:

- **Syntax Errors**

Are the errors that occur when code cannot be parsed by SQL Server.

Are detected by SQL Server before beginning the execution process of a Transact-SQL block or stored procedure.

Are easily identified as the code editor points them out and thus can be easily fixed.

SQL Server 2012

Types of Errors 2-2

- **Run-time Errors**

Are errors that occur when the application tries to perform an action that is supported neither by SQL Server nor by the operating system.

Are sometimes difficult to fix as they are not clearly identified or are external to the database.

- Instances of run-time errors are as follows:
  - Performing a calculation such as division by 0.
  - Trying to execute code that is not defined clearly.

Implementing Error Handling

Most important things that users need to take care of is error handling.

Users also have to take care of handling exception and errors while designing the database.

Various error handling mechanisms are as follows:

- When executing some DML statements such as `INSERT`, `DELETE`, and `UPDATE`, users can handle errors to ensure correct output.
- When a transaction fails and the user needs to rollback the transaction, an appropriate error message can be displayed.
- When working with cursors in SQL Server, users can handle errors to ensure correct results.

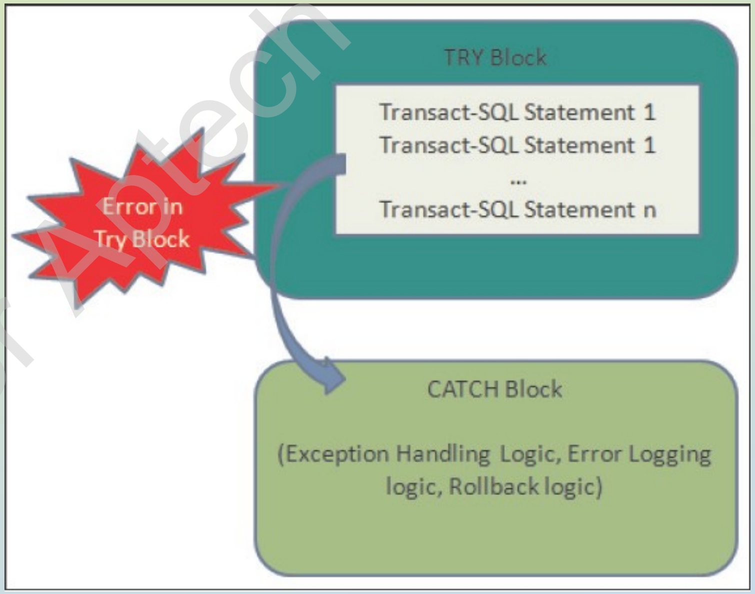
SQL Server 2012

TRY…CATCH 1-3

Are used to implement exception handling in Transact-SQL.

Can enclose one or more Transact-SQL statements within a `TRY` block.

Passes control to the `CATCH` block that may contain one or more statements, if an error occurs in the `TRY` block.



TRY…CATCH 2-3

Syntax:

```
BEGIN TRY

{ sql_statement | statement_block }
END TRY
BEGIN CATCH
[ { sql_statement | statement_block } ]
END CATCH
[ ; ]
```

where,

`sql_statement`: specifies any Transact-SQL statement.

`statement_block`: specifies the group of Transact-SQL statements in a `BEGIN…END` block.

A `TRY…CATCH` construct will catch all run-time errors that have severity higher than 10 and that do not close the database connection.

A `TRY…CATCH` block cannot span multiple batches or multiple blocks of Transact-SQL statements.

TRY...CATCH 3-3

- Following code snippet shows a simple example of TRY...CATCH statements:

```
BEGIN TRY
  DECLARE @num int;
  SELECT @num=217/0;
END TRY
BEGIN CATCH
  PRINT 'Error occurred, unable to divide by 0'
END CATCH;
```

Both TRY and CATCH blocks can contain nested TRY...CATCH constructs

If the CATCH block encloses a nested TRY...CATCH construct, any error in the nested TRY block passes the control to the nested CATCH block

If there is no nested TRY...CATCH construct the error is passed back to the caller

TRY...CATCH constructs can also catch unhandled errors from triggers or stored procedures that execute through the code in TRY block

SQL Server 2012

Error Information 1-3

Good practice is to display error information along with the error, so that it can help to solve the error quickly and efficiently.

System functions need to be used in the `CATCH` block to find information about the error that initiated the `CATCH` block to execute.

- System functions are as follows:
  - `ERROR_NUMBER()`: returns the number of error.
  - `ERROR_SEVERITY()`: returns the severity.
  - `ERROR_STATE()`: returns state number of the error.
  - `ERROR_PROCEDURE()`: returns the name of the trigger or stored procedure where the error occurred.
  - `ERROR_LINE()`: returns the line number that caused the error.
  - `ERROR_MESSAGE()`: returns the complete text of the error. The text contains the value supplied for the parameters such as object names, length, or times.

Functions return `NULL` when they are called outside the scope of the `CATCH` block.

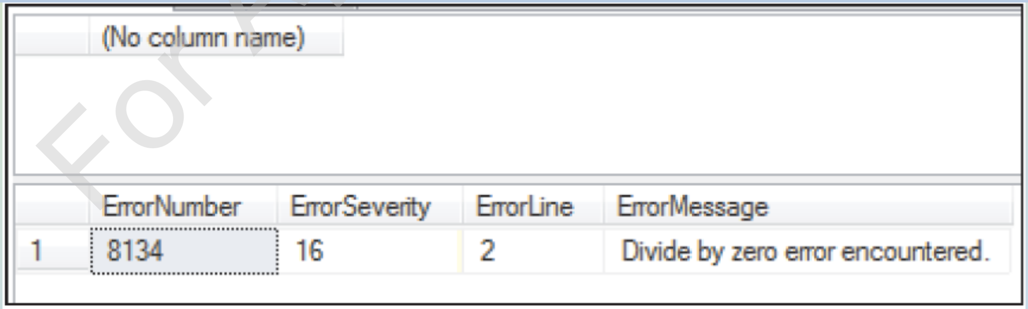
Error Information 2-3

Using TRY...CATCH with error information

- Following code snippet shows a simple example displaying error information:

```
USE AdventureWorks2012;
GO
BEGIN TRY
SELECT 217/0;
END TRY
BEGIN CATCH
SELECT
ERROR_NUMBER() AS ErrorNumber,
ERROR_SEVERITY() AS ErrorSeverity,
ERROR_LINE() AS ErrorLine,
ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

Output:



```
   (No column name)

   ErrorNumber  ErrorSeverity  ErrorLine  ErrorMessage
1  8134         16             2          Divide by zero error encountered.
```

Error Information 3-3

Using TRY…CATCH with Transaction

- Following code snippet shows a example that works inside a transaction:

```
USE AdventureWorks2012;
GO
BEGIN TRANSACTION;
BEGIN TRY
DELETE FROM Production.Product
WHERE ProductID = 980;
END TRY
BEGIN CATCH
SELECT
ERROR_SEVERITY() AS ErrorSeverity
,ERROR_NUMBER() AS ErrorNumber
,ERROR_PROCEDURE() AS ErrorProcedure
,ERROR_STATE() AS ErrorState
,ERROR_MESSAGE() AS ErrorMessage
,ERROR_LINE() AS ErrorLine;
IF @@TRANCOUNT > 0
ROLLBACK TRANSACTION;
END CATCH;
IF @@TRANCOUNT > 0
COMMIT TRANSACTION;
GO
```

SQL Server 2012

Need for Transactions

Uncommittable Transactions

An error generated in a `TRY` block, causes the state of the current transaction to be invalid and considers the transaction as an uncommitted transaction.

An uncommittable transaction performs only `ROLLBACK TRANSACTION` or read operations.

The transaction does not execute any Transact-SQL statement that performs a `COMMIT TRANSACTION` or a write operation.

`XACT_STATE` function returns -1 if the transaction has been classified as an uncommittable transaction.

SQL Server 2012

@@ERROR 1-2

@@ERROR function returns the error number for the last Transact-SQL statement executed.

Syntax:

```
@@ERROR
```

- function returns a value of the integer type
- function returns 0, if the previous Transact-SQL statement encountered no errors
- function returns an error number only if the previous statements encounter an error
- Users can view the text associated with an @@ERROR error number in the sys.messages catalog view

SQL Server 2012

@@ERROR 2-2

- Following code snippet shows how to use @@ERROR to check for a constraint violation:

```
USE AdventureWorks2012;
GO
BEGIN TRY
UPDATE HumanResources.EmployeePayHistory
  SET PayFrequency = 4
  WHERE BusinessEntityID = 1;
END TRY
BEGIN CATCH
IF @@ERROR = 547
PRINT N'Check constraint violation has occurred.';
END CATCH
```

Output:

```
Check constraint violation has occurred.
```

SQL Server 2012

RAISERROR 1-5

Starts the error processing for a session and displays an error message.

Can reference a user-defined message stored in the `sys.messages` catalog view or build dynamic error messages at run-time.

Returns the message as a server error message to the calling application or to the associated `CATCH` block of a `TRY…CATCH` construct.

SQL Server 2012

RAISERROR 2-5

Syntax:

```
RAISERROR ( { msg_id | msg_str | @local_variable }

{ ,severity ,state }
[ ,argument [ ,...n ] ] )
[ WITH option [ ,...n ] ]
```

where,

`msg_id`: specifies the user-defined error message number that is stored in the sys.messages catalog view using the `sp_addmessage`.

`msg_str`: Specifies the user-defined messages with formatting. `msg_str` is a string of characters with optional embedded conversion specifications. A conversion specification has the following format:

```
% [[flag] [width] [. precision] [{h | l}]] type
```

RAISERROR 3-5

- Parameters that can be used in `msg_str` are as follows:
  - `{h | l} type`: Specifies the use of character types d, i, o, s, x, X, or u, and creates shortint(h) or longint(l) values.

- Following are some of the type specifications:
  - `d or i`: Specifies the signed integer.
  - `o`: Specifies the unsigned octal.
  - `x or X`: Specifies the unsigned hexadecimal.
  - `flag`: Specifies the code that determines the spacing and justification of the substituted value. This can include symbols like – (minus) and + (plus) to specify left-justification or to indicate the value is a signed type respectively.
  - `precision`: Specifies the maximum number of characters taken from the argument value for string values.
  - `width`: Specifies an integer that defines the minimum width for the field in which the argument value is placed.

SQL Server 2012

RAISERROR 4-5

`@local_variable`: Specifies a variable of any valid character data type.

`severity`: Severity levels from 0 through 18 are specified by any user. Severity levels from 19 through 25 are specified by members of the sysadmin.

`option`: Specifies the custom option for the error.

- Following table list the values for the custom options:

Value	Description
LOG	Records the error in the error log and the application log for the instance of the Microsoft SQL Server Database Engine.
NOWAIT	Sends message directly to the client
SETERROR	Sets the ERROR_NUMBER and @@ERROR values to msg_id or 5000 irrespective of the severity level.

SQL Server 2012

RAISERROR 5-5

- Following code snippet shows how to build a RAISERROR statement to display a customized error statement:

```
RAISERROR (N'This is an error message %s %d.',
  10, 1, N'serial number', 23);
GO
```

Output:

```
This is error message serial number 23.
```

- Following code snippet shows how to use RAISERROR statement to return the same string:

```
RAISERROR (N'%*.*s', 10, 1, 7, 3, N'Hello world');
GO
RAISERROR (N'%7.3s', 10, 1, N'Hello world');
GO
```

ERROR _STATE

ERROR_STATE system function returns the state number of the error that causes the CATCH block of a TRY…CATCH construct to execute.

Syntax:

```
ERROR_STATE ( )
```

- ERROR_STATE is called from anywhere within the scope of a CATCH block.
- ERROR_STATE returns the error state regardless of how many times it is executed or whether it is executed within the scope of the CATCH block.

➢ Following code snippet shows how to use ERROR_STATE statement inside the TRY block:

```
BEGIN TRY
  SELECT 217/0;
END TRY
BEGIN CATCH
  SELECT ERROR_STATE() AS ErrorState;
END CATCH;
GO
```

SQL Server 2012

ERROR _SEVERITY

ERROR_SEVERITY function returns the severity of the error that causes the CATCH block of a TRY...CATCH construct to be executed.

Syntax:

```
ERROR_SEVERITY ( )
```

- ERROR_SEVERITY can be called anywhere within the scope of a CATCH block.
- ERROR_SEVERITY will return the error severity that is specific to the scope of the CATCH block where it is referenced in a nested CATCH blocks.

➢ Following code snippet shows how to display the severity of the error:

```
BEGIN TRY
  SELECT 217/0;
BEGIN CATCH
  SELECT ERROR_SEVERITY() AS ErrorSeverity;
END CATCH;
GO
END TRY
```

SQL Server 2012

ERROR _PROCEDURE

ERROR_PROCEDURE function returns the trigger or a stored procedure name where the error has occurred that has caused the CATCH block of a TRY…CATCH construct to be executed.

Syntax:

```
ERROR_PROCEDURE ( )
```

- ERROR_PROCEDURE can be called from anywhere in the scope of a CATCH block.
- ERROR_PROCEDURE returns the trigger or stored procedure name specific to the scope of the CATCH block where it is referenced in a nested CATCH blocks.

➢ Following code snippet shows how to use the ERROR_PROCEDURE function:

```
USE AdventureWorks2012;
GO
IF OBJECT_ID ( 'usp_Example', 'P' ) IS
   NOT NULL
DROP PROCEDURE usp_Example;
GO
CREATE PROCEDURE usp_Example
AS SELECT 217/0;
GO
```

```
BEGIN TRY
EXECUTE usp_Example;
END TRY
BEGIN CATCH
SELECT ERROR_PROCEDURE() AS
   ErrorProcedure;
END CATCH;
GO
```

ERROR _NUMBER

ERROR_NUMBER system function when called in a CATCH block returns the error number of the error that causes the CATCH block of a TRY…CATCH construct to be executed.

Syntax:

```
ERROR_NUMBER ( )
```

- ERROR_NUMBER returns the error number irrespective of how many times it executes or whether it executes within the scope of a CATCH block.

➢ Following code snippet shows how to use ERROR_NUMBER in a CATCH block:

```
BEGIN TRY
SELECT 217/0;
END TRY
BEGIN CATCH
SELECT ERROR_NUMBER() AS ErrorNumber;
END CATCH;
GO
```

SQL Server 2012

ERROR _MESSAGE

ERROR_MESSAGE function returns the text message of the error that causes the CATCH block of a TRY…CATCH construct to execute.

Syntax:

```
ERROR_MESSAGE ( )
```

- ERROR_MESSAGE function is called in the CATCH block, it returns the full text of the error message that causes the CATCH block to execute.
- ERROR_MESSAGE returns NULL if it is called outside the scope of a CATCH block.

➢ Following code snippet shows how to use ERROR_MESSAGE in a CATCH block:

```
BEGIN TRY
SELECT 217/0;
END TRY
BEGIN CATCH
SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

SQL Server 2012

ERROR _LINE

ERROR_LINE function returns the line number at which the error occurred in the TRY…CATCH block.

Syntax:

```
ERROR_LINE ( )
```

- ERROR_LINE function is called in the CATCH block, it returns the line number where the error has occurred.
- ERROR_LINE returns the line number in that trigger or stored procedure where the error has occurred.

➢ Following code snippet shows how to use ERROR_LINE in a CATCH block:

```
BEGIN TRY
SELECT 217/0;
END TRY
BEGIN CATCH
SELECT ERROR_LINE() AS ErrorLine;
END CATCH;
GO
```

Errors Unaffected by the TRY…CATCH Construct 1-3

- TRY…CATCH construct does not trap the following conditions:
  - Informational messages or Warnings having a severity of 10 or lower.
  - An error that has a severity of 20 or higher that stops the SQL Server Database Engine task processing for the session.
  - Attentions such as broken client connection or client-interrupted requests.
  - When the session ends because of the KILL statements used by the system administrator.

- Following types of errors are not handled by a CATCH block that occur at the same execution level as that of the TRY…CATCH construct:
  - Compile errors such as syntax errors that restrict a batch from running.
  - Errors that arise in the statement-level recompilation such as object name resolution errors occurring after compiling due to deferred name resolution.

Errors Unaffected by the TRY...CATCH Construct 2-3

- Following code snippet shows how an object name resolution error is generated by the `SELECT` statement:

```
USE AdventureWorks2012;
GO
BEGIN TRY
  SELECT * FROM Nonexistent;
END TRY
BEGIN CATCH
SELECT
ERROR_NUMBER() AS ErrorNumber,
ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

SQL Server 2012

Errors Unaffected by the TRY...CATCH Construct 3-3

- Following code snippet shows how the error message is displayed in such a case:

```
IF OBJECT_ID ( N'sp_Example', N'P' ) IS NOT NULL
DROP PROCEDURE sp_Example;
GO
CREATE PROCEDURE sp_Example
AS
SELECT * FROM Nonexistent;
GO
BEGIN TRY
EXECUTE sp_Example;
END TRY
BEGIN CATCH
SELECT
ERROR_NUMBER() AS ErrorNumber,
ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
```

SQL Server 2012

THROW 1-2

THROW statement raises an exception and transfers control of the execution to a CATCH block of a TRY...CATCH construct.

Syntax:

```
THROW [ { error_number | @local_variable },
{ message | @local_variable },

{ state | @local_variable }
] [ ; ]
```

where,

error_number: specifies a constant or variable that represents the error_number as int.

message: specifies a variable or string that defines the exception message as nvarchar(2048).

state: specifies a variable or a constant between 0 and 255 that specifies the state to associate with state of message as tinyint.

SQL Server 2012

THROW 2-2

➢ Following code snippet shows the use of THROW statement to raise an exception again:

```
USE tempdb;
GO
CREATE TABLE dbo.TestRethrow
( ID INT PRIMARY KEY
);
BEGIN TRY
INSERT dbo.TestRethrow(ID) VALUES(1);
INSERT dbo.TestRethrow(ID) VALUES(1);
END TRY
BEGIN CATCH
PRINT 'In catch block.';
THROW;
END CATCH;
```

Output:

```
(1 row(s) affected)
(0 row(s) affected)
In catch block.
Msg 2627, Level 14, State 1, Line 6
Violation of PRIMARY KEY constraint 'PK__TestReth__3214EC27AAB15FEE'.
Cannot insert duplicate key in object 'dbo.TestRethrow'. The duplicate
key value is (1).
```

SQL Server 2012

Summary

- Syntax errors are the errors that occur when code cannot be parsed by SQL Server.
- Run-time errors occur when the application tries to perform an action that is neither supported by Microsoft SQL Server nor by the operating system.
- TRY…CATCH statements are used to handle exceptions in Transact-SQL.
- TRY…CATCH constructs can also catch unhandled errors from triggers or stored procedures that execute through the code in a TRY block.
- GOTO statements can be used to jump to a label inside the same TRY…CATCH block or to leave a TRY…CATCH block.
- Various system functions are available in Transact-SQL to print error information about the error that occurred.
- The RAISERROR statement is used to start the error processing for a session and displays an error message.