Module 17

Implementing Error Handling

Module Overview

- Implementing T-SQL Error Handling
- Implementing Structured Exception Handling

Lesson 1: Implementing T-SQL Error Handling

- Errors and Error Messages
- Raising Errors Using RAISERROR
- Raising Errors Using THROW
- Using @@Error
- Creating Alerts When Errors Occur
- Demonstration: Handling Errors Using T-SQL

Errors and Error Messages

Elements of Database Engine Errors	
Error number	Unique number identifying the specific error
Error message	Text describing the error
Severity	Numeric indication of seriousness from 1 to 25
State	Internal state code for the database engine condition
Procedure	The name of the stored procedure or trigger in which the error occurred
Line number	Which statement in the batch or procedure generated the error

- System error messages are in **sys.messages**
- Add custom application errors using **sp_add_message**

Raising Errors Using RAISERROR

RAISERROR is used to:

- Help troubleshoot T-SQL code
- Check the values of data
- Return messages that contain variable text

```
RAISERROR (N'%s %d', -- Message text.
           10, -- Severity,
           1, -- State,
           N‘Custom error message number ',
           2);
```

```
Returns:
Custom error message number 2
```

Raising Errors Using THROW

- SQL Server provides the THROW statement
  - Successor to the RAISERROR statement
  - Does not require defining errors in the sys.messages table

```
THROW 50001, 'An Error Occurred', 0;
```

Using @@Error

- @@ERROR returns last error code
- Can be captured and stored in a variable

Creating Alerts When Errors Occur

- Alerts can be fired by messages that are stored in the Windows log
- If a message is not normally logged, it can be logged when it is raised with the addition of WITH LOG

Demonstration: Handling Errors Using T-SQL

In this demonstration, you will see how to:

- Handle errors
- Demonstration A

Answer

- Use RAISERROR

Lesson 2: Implementing Structured Exception Handling

- TRY/CATCH Block Programming
- Error Handling Functions
- Catchable vs. Noncatchable Errors
- Rethrowing Errors Using THROW
- Errors in Managed Code
- Demonstration: Using a TRY/CATCH Block

TRY/CATCH Block Programming

- TRY block defined by BEGIN TRY…END TRY statements
  - Place all code that might raise an error between them
  - No code may be placed between END TRY and BEGIN CATCH
  - TRY and CATCH blocks may be nested
- CATCH block defined by BEGIN CATCH…END CATCH
  - Execution moves to the CATCH block when catchable errors occur within the TRY block

Error Handling Functions

- CATCH blocks make the error-related information available throughout the duration of the CATCH block
- @@ERROR is reset when the next statement is run

Catchable vs. Noncatchable Errors

- TRY/CATCH blocks will only catch errors in the same block
- Common examples of noncatchable errors are:
  - Compile errors, such as syntax errors, that prevent a batch from compiling
  - Statement level recompilation issues that usually relate to deferred name resolution

Rethrowing Errors Using THROW

- Use THROW without parameters to re-raise a caught error
- Must be within a CATCH block

```
BEGIN TRY
        -- code to be executed
END TRY
BEGIN CATCH
        PRINT ERROR_MESSAGE();
        THROW;
END CATCH;
```

Errors in Managed Code

- Errors should be handled by managed code
- Unhandled errors will return error number 6522 to the calling T-SQL code

Demonstration: Using a TRY/CATCH Block

In this demonstration, you will see how to:

- Use a TRY/CATCH block
- Demonstration B

Module Review and Takeaways

- Review Question(s)
- **Question 1**

Which error types cannot by caught by structured exception handling?

- **Question 2**

Can TRY/CATCH blocks be nested?

- **Question 3**

How can you use THROW outside of a CATCH block?

- **Answer 1**

Compile/syntax errors, as well as some delayed name resolution errors.

- **Answer 2**

Yes.

- **Answer 3**

With arguments that raise a user-defined error.