Session - 15

Error Handling

Welcome to the Session, **Error Handling**.

This session introduces error-handling techniques in SQL Server and describes the use of TRY-CATCH blocks. Various system functions and statements that can help display error information are also covered in the session.

At the end of this session, you will be able to:

- 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, ERROR_SEVERITY, and ERROR_PROCEDURE
- Explain the use of ERROR_NUMBER, ERROR_MESSAGE, and ERROR_LINE
- Describe the THROW statement

15.1 Giới thiệu

Xử lý lỗi trong SQL Server đã trở nên dễ dàng thông qua một số kỹ thuật khác nhau. SQL Server đã giới thiệu các tùy chọn có thể giúp bạn xử lý lỗi hiệu quả. Thông thường, không thể nắm bắt các lỗi xảy ra ở đầu cuối của người dùng. SQL Server cung cấp câu lệnh TRY…CATCH giúp xử lý các lỗi có hiệu quả ở nền sau. Ngoài ra còn có một số hàm hệ thống in thông tin liên quan đến lỗi, có thể giúp sửa chữa các lỗi dễ dàng.

15.2 Types of Errors

Là người lập trình Transact-SQL, cần phải nhận thức được các loại lỗi khác nhau có thể xảy ra trong khi làm việc với các câu lệnh SQL Server. Bước đầu tiên người ta có thể thực hiện là xác định loại lỗi và sau đó xác định làm thế nào để xử lý hoặc vượt qua nó.

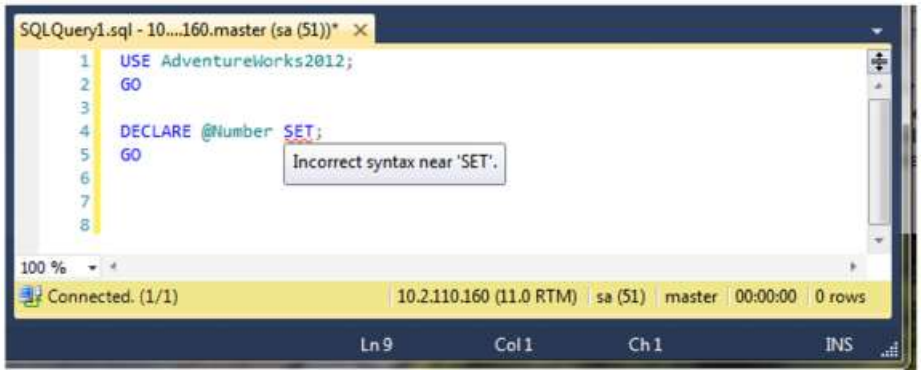
Một số loại lỗi như sau :

- **Lỗi cú pháp**

  Lỗi cú pháp là lỗi xảy ra khi mã không thể phân tích được bằng SQL Server. Lỗi như vậy được phát hiện bởi SQL Server trước khi bắt đầu quá trình thực hiện khối Transact-SQL hoặc thủ tục lưu trữ.

  Một số kịch bản, nơi xảy ra các lỗi cú pháp như sau:

  - Nếu người dùng đang gõ một toán tử hoặc một từ khóa được sử dụng một cách sai lầm, trình biên tập mã sẽ hiển thị chú giải công cụ trình bày lỗi này. Hình 15.1 hiển thị ví dụ về lỗi cú pháp.

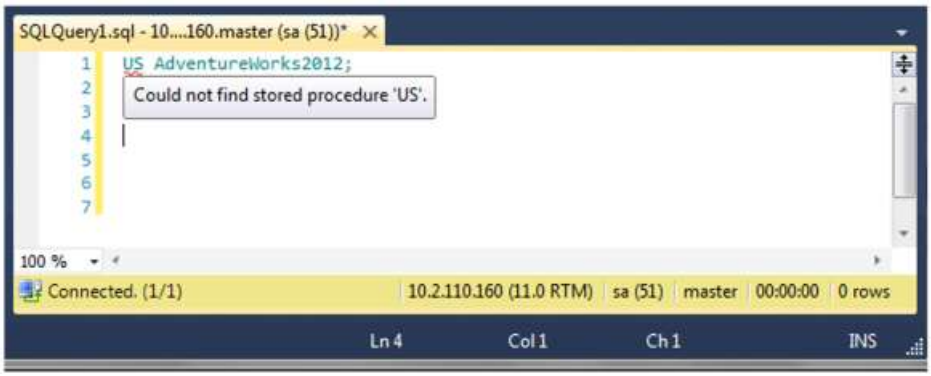


Hình 15.1: Syntax Error

  Trong hình 15.1, toán tử SET được sử dụng sai trong câu lệnh Transact-SQL, do đó lỗi cú pháp sẽ nảy sinh.

  - Nếu người dùng gõ một từ khóa hoặc một toán tử sai bởi vì người dùng không nhớ việc sử dụng hợp lệ, trình biên tập mã sẽ chỉ ra một cách thích hợp.

Hình 15.2 hiển thị ví dụ.



Hình 15.2: Wrong Keyword Error

- Nếu người dùng quên gõ một cái gì đó là bắt buộc, trình biên tập mã sẽ hiển thị lỗi khi người dùng thực hiện câu lệnh đó.

Lỗi cú pháp có thể dễ dàng được xác định khi trình biên tập mã chỉ ra chúng. Do đó, những lỗi này có thể dễ dàng khắc phục. Tuy nhiên, nếu người dùng sử dụng một ứng dụng dựa trên lệnh như sqlcmd, lỗi chỉ được hiển thị sau khi mã được thực thi.

➢ **Lỗi thời gian chạy**

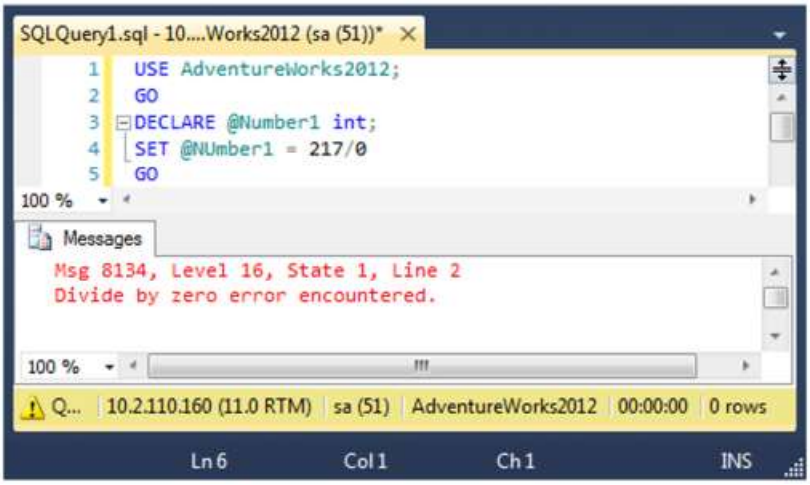
Lỗi thời gian chạy là lỗi xảy ra khi ứng dụng cố gắng thực hiện một hành động không được hỗ trợ bởi SQL Server cũng không phải bởi hệ điều hành. Lỗi thời gian chạy đôi khi rất khó để khắc phục bởi chúng không được xác định rõ ràng hoặc bên ngoài đối với cơ sở dữ liệu.

Một số trường hợp các lỗi thời gian chạy có thể xảy ra như sau:

- Thực hiện một phép tính như chia cho 0
- Thử thực thi mã không được định nghĩa rõ ràng

Concepts

Hình 15.3 trình bày lỗi chia cho số 0.

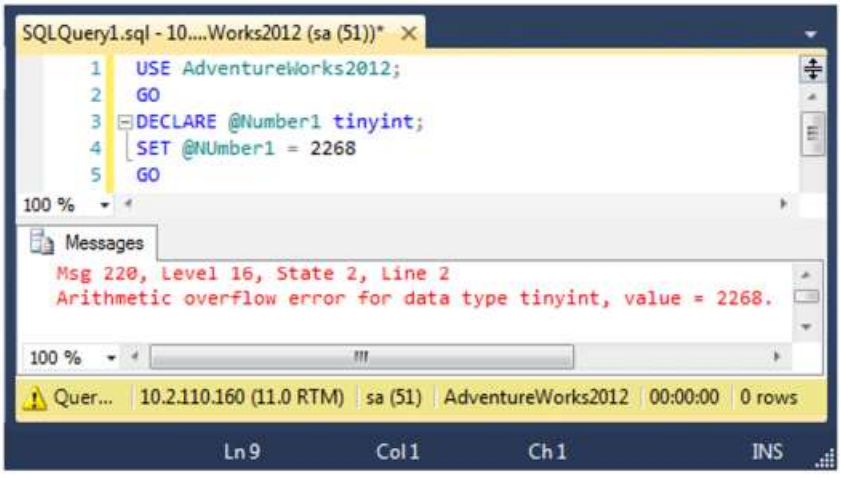


Hình 15.3: Divide by Zero Error

Ở đây, trình biên tập mã sẽ không hiển thị bất kỳ lỗi nào trước khi thực hiện vì không có lỗi cú pháp.

- Sử dụng một thủ tục lưu trữ, hoặc một hàm, hoặc một trigger không có sẵn
- Cố gắng thực hiện một hành động mà một đối tượng hoặc một biến không thể xử lý
- Cố gắng truy cập hoặc sử dụng bộ nhớ máy tính không đủ

Hình 15.4 hiển thị ví dụ về việc cố gắng lưu trữ một giá trị vào biến không đáp ứng phạm vi chỉ định. Trong trường hợp này, lỗi tràn số học xảy ra.



Hình 15.4:Arithmetic Overflow Error

- Cố gắng thực hiện một hành động trên các loại không tương thích
- Sử dụng các câu lệnh điều kiện sai

15.3 Thực hiện xử lý lỗi

Trong khi phát triển bất kỳ ứng dụng nào, một trong những điều quan trọng nhất mà người dùng cần phải quan tâm là xử lý lỗi. Trong cùng một cách, người dùng cũng phải quan tâm đến việc xử lý ngoại lệ và các lỗi trong khi thiết kế cơ sở dữ liệu. Có thể sử dụng các cơ chế xử lý lỗi khác nhau. Một số trong số này như sau:

➢ Khi thực thi một số câu lệnh DML như là **INSERT, DELETE**, và **UPDATE**, người dùng có thể xử lý các lỗi để đảm bảo đầu ra chính xác.

➢ Khi giao tác không thành công và người dùng cần quay lui giao tác, có thể hiển thị thông báo lỗi thích hợp.

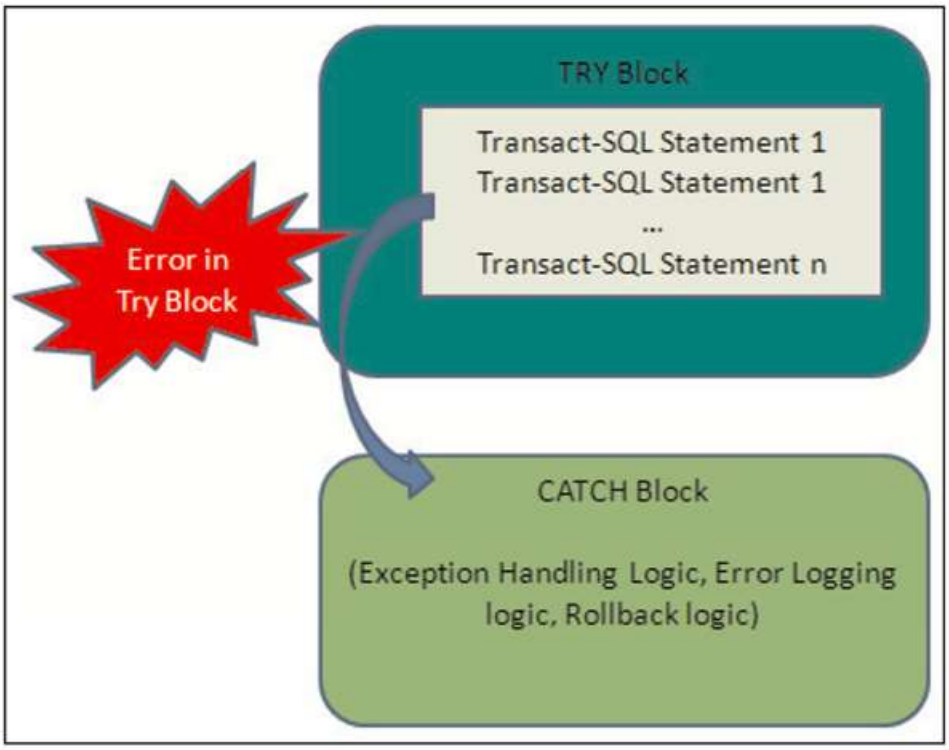
➢ Khi làm việc với các con trỏ trong SQL Server, người dùng có thể xử lý các lỗi để đảm bảo có các kết quả chính xác.

15.4 TRY... CATCH

Các câu lệnh TRY…CATCH được sử dụng để thực hiện xử lý ngoại lệ trong Transact- SQL. Một hoặc nhiều câu lệnh Transact-SQL có thể được đưa vào trong khối TRY. Nếu lỗi xảy ra trong khối TRY, kiểm soát được truyền cho khối CATCH có thể chứa một hoặc nhiều câu lệnh.

Concepts

Hình 15.5 minh họa logic **TRY…CATCH**.



Hình 15.5: TRY...CATCH Logic

Sau đây là cú pháp cho các câu lệnh **TRY…CATCH**.

Cú pháp :

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

trong đó,

sql_statement : chỉ ra bất kỳ câu lệnh Transact-SQL nào.

statement_block : chỉ ra nhóm các câu lệnh Transact-SQL trong khối **BEGIN…END.**

Concepts

Cấu trúc **TRY…CATCH** sẽ bắt tất cả các lỗi thời gian chạy có mức độ nghiêm trọng cao hơn 10 và không đóng kết nối cơ sở dữ liệu. Khối TRY theo sau là khối CATCH liên quan. Khối **TRY…CATCH** không thể kéo dài nhiều đợt hoặc nhiều khối câu lệnh Transact-SQL.

Nếu không có lỗi trong khối TRY, sau khi câu lệnh cuối cùng trong khối TRY đã thực hiện, kiểm soát được truyền tới câu lệnh tiếp theo sau câu lệnh **END CATCH**. Nếu có một lỗi trong khối TRY, kiểm soát được truyền tới câu lệnh đầu tiên bên trong khối **CATCH**. Nếu **END CATCH** là câu lệnh cuối cùng trong một trigger hoặc một thủ tục lưu trữ, kiểm soát được truyền trở lại vào khối gọi.

Code Snippet 1 trình bày một ví dụ đơn giản về các câu lệnh **TRY…CATCH**.

Code Snippet 1:

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

Trong mã này, nỗ lực được thực hiện để chia một số cho số 0. Điều này sẽ gây ra lỗi, do đó, câu lệnh TRY…CATCH được sử dụng ở đây để xử lý lỗi.

Cả hai khối TRY và CATCH có thể chứa cấu trúc TRY…CATCH lồng nhau. Ví dụ, khối CATCH có thể có một cấu trúc TRY…CATCH nhúng để xử lý các lỗi phải đối mặt với mã CATCH. Lỗi gặp phải trong khối CATCH được đối xử giống như các lỗi được tạo ra ở nơi khác. Nếu khối CATCH bao quanh một cấu trúc TRY…CATCH, bất kỳ lỗi nào trong khối TRY lồng nhau truyền kiểm soát tới khối CATCH lồng nhau. Nếu không có cấu trúc TRY…CATCH lồng nhau, lỗi được truyền lại cho hàm gọi.

Cấu trúc TRY…CATCH cũng có thể bắt các lỗi không được xử lý từ các trigger hoặc thủ tục lưu trữ sẽ thực hiện thông qua mã trong khối TRY. Tuy nhiên, là một cách tiếp cận khác, các trigger hoặc thủ tục lưu trữ cũng có thể bao bọc các cấu trúc TRY…CATCH của chính mình để xử lý các lỗi được tạo ra thông qua mã của chúng.

Câu lệnh GOTO có thể được sử dụng để nhảy đến một nhãn bên trong cùng một khối TRY…CATCH hoặc để rời khỏi khối TRY…CATCH. Cấu trúc TRY…CATCH không nên được sử dụng trong một hàm do người dùng định nghĩa.

15.5 Thông tin lỗi

Thực hành tốt là hiển thị thông tin lỗi cùng với lỗi, để nó có thể giúp giải quyết lỗi một cách nhanh chóng và hiệu quả.

Để đạt được điều này, các hàm hệ thống cần phải được sử dụng trong khối CATCH để tìm thông tin về lỗi đã khởi xướng khối CATCH thực thi.

Concepts

Những hàm hệ thống như sau:

- **ERROR _ NUMBER()** : trả về số của lỗi.
- **ERROR _ SEVERITY()** : trả về mức độ nghiêm trọng.
- **ERROR _ STATE()** : trả về số trạng thái của lỗi.
- **ERROR _ PROCEDURE()** : trả về tên của trigger hoặc thủ tục lưu trữ nơi đã xảy ra lỗi.
- **ERROR _ LINE()** : trả về số của dòng đã gây ra lỗi.
- **ERROR _ MESSAGE()** : trả về văn bản đầy đủ của lỗi. Văn bản có giá trị được cung cấp cho các tham số như là tên đối tượng, chiều dài, hoặc thời gian.

Những hàm này trả về NULL khi chúng được gọi ra bên ngoài phạm vi của khối **CATCH**.

- Sử dụng **TRY…CATCH** với thông tin lỗi

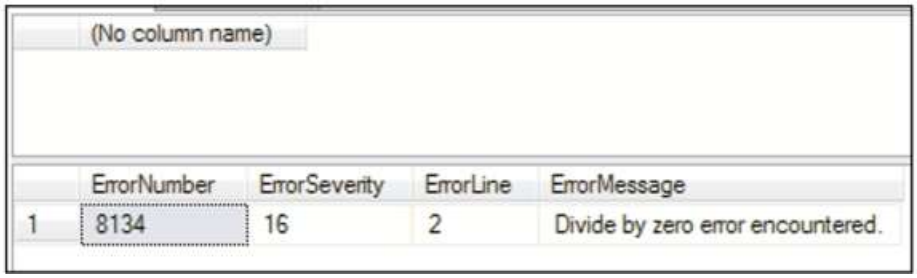
  Code Snippet 2 trình bày một ví dụ đơn giản hiển thị thông tin lỗi.

  Code Snippet 2 :

```
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
```

Trong mã này, câu lệnh SELECT sẽ gây ra một lỗi chia cho số không được xử lý sử dụng câu lệnh TRY…CATCH. Lỗi gây ra việc thực thi để nhảy đến khối CATCH trong đó thông tin lỗi sẽ được hiển thị.

Hình 15.6 trình bày kết quả của thông tin lỗi. Tập kết quả đầu tiên là trống vì câu lệnh không thành công.



(No column name)

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

Hình 15.6: Error Information

Code Snippet 3 trình bày một thủ tục lưu trữ có chứa các hàm xử lý lỗi.

Code Snippet 3:

```
USE AdventureWorks2012;
GO
IF OBJECT_ID ('sp_ErrorInfo', 'P') IS NOT NULL
 DROP PROCEDURE sp_ErrorInfo;
GO
CREATE PROCEDURE sp_ErrorInfo
AS
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;
GO
```

Concepts

```
BEGIN TRY
 SELECT 217/0;
END TRY
BEGIN CATCH
 EXECUTE sp_ErrorInfo;
END CATCH;
```

Trong đoạn mã này, khi lỗi xảy ra, khối **CATCH** của cấu trúc **TRY…CATCH** được gọi ra và thông tin lỗi được trả lại**.**

- **Sử dụng TRY…CATCH với giao tác**

  Code Snippet 4 thể hiện khối TRY…CATCH làm việc bên trong một giao tác**.**

  Code Snippet 4:

```
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
```

Trong mã này, khối TRY…CATCH làm việc trong giao tác. Câu lệnh bên trong khối TRY tạo ra một lỗi vi phạm ràng buộc như sau:

The DELETE statement is conflicted with the REFERENCE constraint "FK_BillOfMaterials_Product_ProductAssemblyID". The conflict occurred in database "AdventureWorks2012", table "Production.BillOfMaterials", column 'ProductAssemblyID'.

- **Các giao tác không thể thực hiện được**

  Nếu lỗi được tạo ra trong khối TRY, nó làm cho trạng thái của giao tác hiện tại thành không hợp lệ và giao tác được coi là một giao tác không được thực hiện. Giao tác không thể thực hiện được chỉ thực hiện các hoạt động ROLLBACK TRANSACTION hoặc đọc. Giao tác không thực thi bất kỳ câu lệnh Transact-SQL nào mà thực hiện hoạt động COMMIT TRANSACTION hoặc viết.

  Hàm XACT_STATE trả về -1 nếu giao tác đã được phân loại như là một giao tác không thể thực hiện được. Khi khối lệnh đã hoàn thành, Database Engine quay lui bất kỳ giao tác không thể thực hiện được nào. Nếu không có thông báo lỗi nào được gửi đi khi các giao tác đi vào trạng thái không thể thực hiện được khi hoàn thành khối lệnh, khi đó các thông báo lỗi được gửi cho máy khách. Điều này chỉ ra rằng một giao tác không thể thực hiện được được phát hiện và quay lui.

15.6 @@ERROR

Hàm @@**ERROR** trả về số của lỗi cho câu lệnh Transact-SQL cuối cùng đã thực hiện.

Sau đây là cú pháp cho hàm @@**ERROR**.

Cú pháp :

```
@@ERROR
```

Hàm hệ thống @@**ERROR** trả về giá trị thuộc loại số nguyên. Hàm này trả về 0, nếu câu lệnh Transact- SQL trước không gặp phải lỗi nào. Nó còn trả về số của lỗi chỉ khi các câu lệnh trước đó gặp phải lỗi. Nếu lỗi là một trong những danh sách các lỗi trong dạng xem danh mục sys.messages bao gồm giá trị từ cột **sys.messages.messages_id** cho lỗi đó. Người dùng có thể xem văn bản có liên quan đến số của lỗi @@**ERROR** trong dạng xem danh mục sys.messages.

Code Snippet 5 trình bày cách sử dụng @@ERROR để kiểm tra các vi phạm ràng buộc.

Code Snippet 5:

```
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
```

Trong đoạn mã này, @@**ERROR** được sử dụng để kiểm tra vi phạm ràng buộc kiểm tra (trong đó có số của lỗi 547) trong câu lệnh UPDATE.

Nó sẽ hiển thị thông báo lỗi sau:

Check constraint violation has occurred.

15.7 RAISERROR

Câu lệnh RAISERROR bắt đầu quá trình xử lý lỗi cho một phiên và hiển thị thông báo lỗi. RAISERROR có thể tham khảo một thông báo do người dùng định nghĩa được lưu trữ trong dạng xem danh mục sys.messages hoặc xây dựng các thông báo lỗi động vào thời gian chạy. Thông báo này được trả về như một thông báo lỗi máy chủ cho ứng dụng gọi hoặc cho khối CATCH liên quan của cấu trúc **TRY…CATCH**.

Sau đây là cú pháp cho câu lệnh **RAISERROR**.

Cú pháp :

```
RAISERROR ( { msg_id | msg_str | @local_variable }
    { ,severity ,state }
    [ ,argument [ ,...n ] ] )
    [ WITH option [ ,...n ] ]
```

trong đó,

msg_id : chỉ ra số của thông báo lỗi do người dùng định nghĩa được lưu trữ trong dạng xem danh mục sys.messages sử dụng sp_addmessage.

Concepts

msg_str: Chỉ ra các thông báo do người dùng định nghĩa với định dạng. msg_str là một chuỗi ký tự với thông số kỹ thuật chuyển đổi nhúng tùy chọn. Thông số kỹ thuật chuyển đổi có định dạng sau:

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

Các tham số có thể được sử dụng trong msg_str như sau:

{h | l} type: Chỉ ra việc sử dụng các loại ký tự **d, i, o, s, x, X**, hoặc **u**, và tạo ra **shortint(h)** hoặc **longint(l)**.

Sau đây là một số thông số kỹ thuật loại:

d hoặc i : Chỉ ra số nguyên có dấu

o : Chỉ ra bát phân không dấu

x hoặc X : Chỉ ra thập lục phân không dấu

flag : Chỉ ra mã xác định giãn cách và sự cân chỉnh giá trị thay thế. Cái này có thể bao gồm các biểu tượng như - (trừ) và +(cộng) để chỉ ra căn chỉnh trái hoặc để chỉ ra giá trị tương ứng là loại có dấu.

precision : Chỉ ra số lượng ký tự tối đa được lấy từ giá trị đối số cho các giá trị chuỗi. Ví dụ, nếu một chuỗi có năm ký tự và độ chính xác là 2, chỉ có hai ký tự đầu tiên của giá trị chuỗi được sử dụng.

width : Chỉ ra một số nguyên định nghĩa chiều rộng tối thiểu cho trường, trong đó giá trị đối số được đặt.

@local_variable: Chỉ ra một biến thuộc bất kỳ kiểu dữ liệu ký tự hợp lệ nào có chứa chuỗi được định dạng trong cùng một cách như **msg_str**.

severity : Mức độ nghiêm trọng từ 0 đến 18 được chỉ ra bởi bất kỳ người dùng nào. Mức độ nghiêm trọng từ 19 đến 25 được quy định bởi các thành viên của vai trò máy chủ cố định sysadmin hoặc người dùng với cấp phép ALTER TRACE. Mức độ nghiêm trọng từ 19 đến 25 sử dụng tùy chọn WITH LOG là bắt buộc. option: Chỉ ra tùy chọn tùy chỉnh cho lỗi này.

option : Chỉ ra tùy chọn tùy chỉnh cho lỗi này.

Table 15.1 liệt kê những giá trị cho các tùy chọn tùy chỉnh

Value	Description
LOG	Ghi lại lỗi trong nhật ký lỗi và nhật ký ứng dụng cho thể hiện của Công cụ cơ sở dữ liệu Microsoft SQL Server.
NOWAIT	Gửi thông báo trực tiếp cho máy khách
SETERROR	Đặt các giá trị ERROR_NUMBER và @@ERROR thành msg_id hoặc 5000 không phân biệt mức độ nghiêm trọng.

Table 15.1: Type Specification Values

Khi **RAISERROR** thực thi với mức độ nghiêm trọng là 11 hoặc cao hơn trong khối **TRY**, nó sẽ chuyển điều khiển đến khối **CATCH** liên quan.

Concepts

Những lỗi sau đây được trả về lại cho hàm gọi nếu **RAISERROR** thực hiện:

- Ra khỏi phạm vi của bất kỳ khối **TRY** nào
- Có mức độ nghiêm trọng là 10 hoặc thấp hơn trong khối **TRY**
- Có mức độ nghiêm trọng là 20 hoặc cao hơn sẽ chấm dứt kết nối cơ sở dữ liệu

Khối **CATCH** có thể sử dụng câu lệnh **RAISERROR** để ném lại lỗi đã gọi ra khối CATCH. Đối với điều này, sẽ cần phải biết thông tin lỗi ban đầu có thể thu được thông qua các hàm hệ thống **ERROR_NUMBER** và **ERROR_MESSAGE.**

Theo mặc định, @@ERROR được đặt là 0 cho các thông báo có mức độ nghiêm trọng từ 1 đến 10.

Code Snippet 6 trình bày cách để xây dựng câu lệnh **RAISERROR** để hiển thị câu lệnh lỗi tùy chỉnh.

Code Snippet 6:

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

Trong đoạn mã này, câu lệnh **RAISERROR** có tham số đầu tiên của N'số sêri' thay đổi thông số kỹ thuật chuyển đổi đầu tiên là %s, và tham số thứ hai là 23 thay đổi chuyển đổi thứ hai là %d. Đoạn mã này hiển thị **'Đây là thông báo lỗi số sêri 23'**. Code Snippet 7 trình bày cách sử dụng câu lệnh **RAISERROR** để trả về cùng một chuỗi.

Code Snippet 7:

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

Trong đoạn mã này, câu lệnh **RAISERROR** trở về cùng một chuỗi **Hel**. Câu lệnh đầu tiên chỉ ra chiều rộng và các giá trị độ chính xác và câu lệnh thứ hai chỉ ra thông số kỹ thuật chuyển đổi.

Người dùng cũng có thể trả về thông tin lỗi từ khối **CATCH**.

Code Snippet 8 trình bày cách sử dụng câu lệnh RAISERROR bên trong khối TRY.

Code Snippet 8:

```
BEGIN TRY
   RAISERROR ('Raises Error in the TRY block.', 16, 1);
END TRY
BEGIN CATCH
   DECLARE @ErrorMessage NVARCHAR(4000);
   DECLARE @ErrorSeverity INT;
   DECLARE @ErrorState INT;
   SELECT
        @ErrorMessage = ERROR_MESSAGE(),
        @ErrorSeverity = ERROR_SEVERITY(),
@ErrorState = ERROR_STATE();
   RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState);
END CATCH;
```

Trong đoạn mã này, câu lệnh **RAISERROR** được sử dụng bên trong khối **TRY** có mức độ nghiêm trọng 16, từ đó dẫn đến việc thực thi để chuyển đến khối **CATCH** liên kết.

RAISERROR sau đó được sử dụng bên trong khối **CATCH** để trả về thông tin lỗi về lỗi ban đầu.

15.8 ERROR_STATE

Hàm hệ thống **ERROR_STATE** trả về số lỗi trạng thái làm cho khối **CATCH** của cấu trúc **TRY…CATCH** thực hiện. Sau đây là cú pháp cho hàm hệ thống **ERROR_STATE**.

Cú pháp:

```
ERROR_STATE ( )
```

Khi được gọi trong khối **CATCH**, nó trả về số trạng thái của thông báo lỗi đã làm cho khối **CATCH** được chạy. Hàm này trả về **NULL** khi chúng được gọi ra bên ngoài phạm vi của khối **CATCH**.

Có các thông báo lỗi cụ thể được nêu ra tại các điểm khác nhau trong đoạn mã cho Công cụ cơ sở dữ liệu SQL Server. Ví dụ, lỗi **1105** được sinh ra cho một số điều kiện khác nhau. Mỗi điều kiện cụ thể gây nên lỗi sẽ gán mã trạng thái duy nhất này.

ERROR_STATE được gọi từ bất cứ nơi nào trong phạm vi khối **CATCH.**
ERROR_STATE trả về trạng thái lỗi bất kể bao nhiêu lần nó được thực thi hoặc cho dù nó được thực thi trong phạm vi của khối **CATCH**. Đây là trong so sánh với những hàm như **@@ERROR** chỉ trả về số lỗi trong câu lệnh ngay sau cái gây ra lỗi, hoặc trong câu lệnh đầu tiên của khối **CATCH.**

Concepts

Người dùng có thể sử dụng ERROR_STATE trong khối CATCH. Code Snippet 9 trình bày cách sử dụng câu lệnh ERROR_STATE bên trong khối TRY.

Code Snippet 9:

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

Trong đoạn mã này, câu lệnh **SELECT** tạo ra lỗi chia cho số 0. Câu lệnh **CATCH** sau đó sẽ trả về trạng thái của lỗi. The **ERROR_STATE** được hiển thị là 1.

15.9 ERROR_SEVERITY

Hàm **ERROR_SEVERITY** trả về mức độ nghiêm trọng của lỗi gây ra khối **CATCH** của cấu trúc **TRY…CATCH** sẽ được thực thi.

Sau đây là cú pháp cho **ERROR_SEVERITY**.

Cú pháp :

```
ERROR_SEVERITY ( )
```

Nó trả về giá trị NULL nếu được gọi ra bên ngoài phạm vi của khối **CATCH**. **ERROR_STATE** có thể được gọi bất cứ nơi nào trong phạm vi khối CATCH. Trong các khối CATCH lồng nhau, **ERROR_SEVERITY** sẽ trả về mức độ nghiêm trọng của lỗi cụ thể cho phạm vi của khối **CATCH** nơi nó được tham chiếu. Người dùng có thể sử dụng hàm **ERROR_SEVERITY** trong khối **CATCH**.

Code Snippet 10 trình bày cách hiển thị mức độ nghiêm trọng của lỗi.

Code Snippet 10:

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

```
GO
END TRY
```

Trong đoạn mã này, một nỗ lực để chia cho số 0 tạo ra lỗi và làm cho khối CATCH hiển thị lỗi nghiêm trọng là 16.

15.10 ERROR_PROCEDURE

Hàm **ERROR_PROCEDURE** trả về trigger hoặc tên thủ tục lưu trữ nơi đã xảy ra lỗi đã làm cho khối **CATCH** của một cấu trúc **TRY…CATCH** được thực thi. Sau đây là cú pháp của **ERROR_PROCEDURE**.

Cú pháp :

```
ERROR_PROCEDURE ( )
```

Nó trả về kiểu dữ liệu nvarchar. Khi hàm được gọi ra trong khối **CATCH**, nó sẽ trả về tên của thủ tục lưu trữ nơi đã xảy ra lỗi. Hàm này trả về giá trị NULL nếu lỗi đã không xảy ra trong trigger hoặc một thủ tục lưu trữ. **ERROR_PROCEDURE** có thể được gọi từ bất cứ nơi nào trong phạm vi khối **CATCH**. Hàm này còn trả về **NULL** nếu hàm này được gọi ra bên ngoài phạm vi của khối **CATCH**.

Trong các khối **CATCH** lồng nhau, **ERROR_PROCEDURE** trả về trigger hoặc tên thủ tục lưu trữ cụ thể cho phạm vi của khối **CATCH**, nơi nó được tham chiếu.

Code Snippet 11 trình bày việc sử dụng hàm **ERROR_PROCEDURE**.

Code Snippet 11:

```
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
```

Trong đoạn mã này, thủ tục lưu trữ **usp_Example** tạo ra lỗi chia cho số 0. Vì vậy hàm **ERROR_PROCEDURE** trả về tên của thủ tục lưu trữ này, nơi đã xảy ra lỗi.

Code Snippet 12 trình bày việc sử dụng hàm **ERROR_PROCEDURE** cùng với các hàm khác.

Code Snippet 12:

```
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_NUMBER() AS ErrorNumber,
 ERROR_SEVERITY() AS ErrorSeverity,
 ERROR_STATE() AS ErrorState,
 ERROR_PROCEDURE() AS ErrorProcedure,
 ERROR_MESSAGE() AS ErrorMessage,
 ERROR_LINE() AS ErrorLine;
 END CATCH;
GO
```

Đoạn mã này sử dụng một số hàm hệ thống xử lý lỗi có thể giúp phát hiện và khắc phục lỗi dễ dàng.

15.11 ERROR_NUMBER

Hàm hệ thống **ERROR_NUMBER** khi được gọi trong khối **CATCH** trả về số lỗi của lỗi làm cho khối **CATCH** của cấu trúc **TRY…CATCH** được thực thi. Sau đây là cú pháp của **ERROR_NUMBER**.

Cú pháp:

```
ERROR_NUMBER ( )
```

Hàm này có thể được gọi từ bất cứ nơi nào trong phạm vi khối **CATCH**. Hàm này sẽ trả về **NULL** khi nó được gọi ra bên ngoài phạm vi của khối **CATCH**.

ERROR_NUMBER trả về số lỗi bất kể bao nhiêu lần nó thực thi hoặc cho dù nó được thực thi trong phạm vi của khối **CATCH**. Cái này khác so với **@@ERROR** chỉ trả về số lỗi trong câu lệnh này ngay sau cái gây ra lỗi, hoặc câu lệnh đầu tiên của khối **CATCH**.

Code Snippet 13 trình bày việc sử dụng **ERROR_NUMBER** trong **CATCH**.

Code Snippet 13:

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

Là kết quả của đoạn mã này, số lỗi được hiển thị khi cố gắng chia cho số 0 xảy ra.

15.12 ERROR_MESSAGE

Hàm **ERROR_MESSAGE** trả về thông báo lỗi của lỗi làm cho khối **CATCH** của cấu trúc **TRY…CATCH** thực thi.

Sau đây là cú pháp của **ERROR_MESSAGE**.

Cú pháp :

```
ERROR_MESSAGE ( )
```

Concepts

Khi hàm **ERROR_MESSAGE** được gọi trong khối **CATCH**, nó sẽ trả về toàn bộ nội dung của thông báo lỗi làm cho khối **CATCH** thực thi. Văn bản bao gồm những giá trị được cung cấp cho bất kỳ tham số nào có thể thay thế được như tên đối tượng, thời gian, hoặc độ dài. Nó còn trả về **NULL** nếu hàm này được gọi ra bên ngoài phạm vi của khối **CATCH.**

Code Snippet 14 trình bày việc sử dụng **ERROR_MESSAGE** trong khối **CATCH**.

Code Snippet 14:

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

Trong đoạn mã này, tương tự như các ví dụ khác, câu lệnh **SELECT** tạo ra lỗi chia cho số 0. Khối **CATCH** hiển thị thông báo lỗi.

15.13 ERROR_LINE

Hàm **ERROR_LINE** trả về số dòng mà tại đó xảy ra lỗi trong khối TRY…CATCH..

Sau đây là cú pháp của **ERROR_LINE**.

Cú pháp:

```
ERROR_LINE ( )
```

Khi hàm này được gọi trong khối **CATCH**, nó trả về số dòng nơi đã xảy ra lỗi. Nếu lỗi đã xảy ra trong trigger hoặc thủ tục lưu trữ, nó trả về số dòng trong trigger hoặc thủ tục lưu trữ đó. Tương tự như các hàm khác, hàm này trả về **NULL** khi chúng được gọi ra bên ngoài phạm vi của khối **CATCH.**

Code Snippet 15 trình bày việc sử dụng **ERROR_LINE** trong khối **CATCH.**

Code Snippet 15:

```
BEGIN TRY
 SELECT 217/0;
END TRY
```

Concepts

```
BEGIN CATCH
 SELECT ERROR_LINE() AS ErrorLine;
END CATCH;
GO
```

Là kết quả của đoạn mã này, số dòng tại đó lỗi đã xảy ra sẽ được hiển thị.

15.14 Lỗi bị ảnh hưởng bởi cấu trúc TRY...CATCH

Cấu trúc **TRY…CATCH** không bẫy các điều kiện sau:

- Các thông báo cung cấp thông tin hoặc cảnh báo có mức độ nghiêm trọng là 10 hoặc thấp hơn
- Lỗi có mức độ nghiêm trọng là 20 hoặc cao hơn sẽ dừng lại việc xử lý tác vụ Công cụ cơ sở dữ liệu SQL Server cho phiên đó. Nếu xảy ra lỗi có mức độ nghiêm trọng là 20 hoặc cao hơn và kết nối cơ sở dữ liệu không bị gián đoạn, **TRY…CATCH** sẽ xử lý lỗi
- Sự quan tâm như là kết nối máy khách bị hỏng hoặc yêu cầu máy khách bị gián đoạn
- Khi phiên kết thúc vì câu lệnh KILL được sử dụng bởi quản trị viên hệ thống

Những loại lỗi sau đây không được xử lý bằng khối **CATCH** xảy ra ở cùng một cấp độ thực thi như của cấu trúc **TRY…CATCH**:

- Biên dịch lỗi như lỗi cú pháp để hạn chế khối lệnh không chạy
- Lỗi phát sinh trong biên dịch lại ở mức câu lệnh như là lỗi phân giải tên đối tượng xảy ra sau khi biên dịch do độ phân giải tên bị trì hoãn.

  Code Snippet 16 trình bày cách một lỗi phân giải tên đối tượng được tạo ra do câu lệnh **SELECT**.

  Code Snippet 16:

```
USE AdventureWorks2012;
GO
BEGIN TRY
 SELECT * FROM Nonexistent;
END TRY
```

```
BEGIN CATCH
 SELECT
 ERROR_NUMBER() AS ErrorNumber,
 ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

Đoạn mã này sẽ gây ra lỗi phân giải tên đối tượng trong câu lệnh **SELECT**. Nó sẽ không bắt được bằng cấu trúc **TRY…CATCH**.

Việc chạy câu lệnh SELECT bên trong một thủ tục lưu trữ gây ra lỗi xảy ra ở một mức độ thấp hơn khối **TRY**. Lỗi này được xử lý bởi cấu trúc **TRY…CATCH**.

Code Snippet 17 trình bày thông báo lỗi được hiển thị như thế nào trong trường hợp như vậy.

Code Snippet 17:

```
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;
```

Concepts

15.15 THROW

Câu lệnh THROW phát sinh ngoại lệ chuyển điều khiển thực thi cho khối CATCH của cấu trúc **TRY…CATCH**.

Sau đây là cú pháp của câu lệnh THROW.

Cú pháp:

```
THROW [ { error_number | @local_variable },
 { message | @local_variable },
    { state | @local_variable }
] [ ; ]
```

trong đó,

error_number : chỉ ra một hằng số hoặc biến trình bày error_number là int.

message : chỉ ra một biến hoặc chuỗi định nghĩa thông báo ngoại lệ là nvarchar(2048).

State : chỉ ra một biến hoặc hằng số từ 0 đến 255 chỉ ra trạng thái để liên kết với trạng thái của thông báo là tinyint.

Code Snippet 18 trình bày việc sử dụng câu lệnh THROW để gây ra lại một ngoại lệ.

Code Snippet 18:

```
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;
```

Trong đoạn mã này, câu lệnh THROW được sử dụng để gây ra lại một lần nữa ngoại lệ đã xảy ra lần cuối.

Concepts

Kết quả của đoạn mã này sẽ như sau:

```
(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).
```

15.16 Kiểm tra tiến độ của bạn

1. _____ chỉ xảy ra nếu người dùng viết đoạn mã mà không thể được phân tích bằng SQL Server, chẳng hạn như từ khóa sai hoặc câu lệnh không đầy đủ.

(A)	Lỗi cú pháp	**(C)**	Lỗi logic
(B)	Lỗi thời gian chạy	**(D)**	Nhật ký lỗi

2. Cấu trúc nào sau đây có thể bắt các lỗi chưa xử lý từ các trigger hoặc thủ tục lưu trữ?

(A)	IF-ELSE	(C)	RAISERROR
(B)	TRY…CATCH	(D)	@@ERROR

3. Hàm nào sau đây trả về số lỗi cho câu lệnh Transact-SQL cuối cùng đã thực thi?

(A)	ERROR_LINE	(C)	@@ERROR
(B)	RAISERROR	(D)	@@ERROR_NUMBER

4. Hàm nào say đây trả về mức độ nghiêm trọng của lỗi làm cho khối CATCH của cấu trúc TRY…CATCH được thực thi?

(A)	ERROR_LINE	(C)	ERROR_PROCEDURE
(B)	ERROR_NUMBER	(D)	ERROR_SEVERITY

5. Câu lệnh_____sinh ra ngoại lệ và truyền thực thi cho khối **CATCH** của cấu trúc **TRY…CATCH** trong SQL Server 2012.

(A)	BEGIN	(C)	THROW
(B)	END	(D)	ERROR

Concepts

15.16.1 Đáp án

1.	(A)
2.	(B)
3.	(C)
4.	(D)
5.	(C)

- Lỗi cú pháp là lỗi xảy ra khi mã không thể phân tích được bằng SQL Server.
- Lỗi thời gian chạy xảy ra khi ứng dụng cố gắng thực hiện một hành động không được hỗ trợ bởi Microsoft SQL Server cũng không phải bởi hệ điều hành.
- Các câu lệnh TRY…CATCH được sử dụng để xử lý các ngoại lệ trong Transact-SQL.
- Cấu trúc TRY…CATCH cũng có thể bắt các lỗi không được xử lý từ các trigger hoặc thủ tục lưu trữ sẽ thực hiện thông qua mã trong khối TRY.
- Các câu lệnh GOTO có thể được sử dụng để nhảy đến một nhãn bên trong cùng một khối TRY…CATCH hoặc để rời khỏi khối TRY…CATCH.
- Các hàm hệ thống khác nhau có sẵn trong Transact-SQL để in thông tin lỗi về lỗi đã xảy ra .
- Câu lệnh RAISERROR được dùng để bắt đầu quá trình xử lý lỗi cho một phiên và hiển thị thông báo lỗi.

Try It Yourself

1. Đối với các giao tác được tạo ra trong Hãy thử tự làm của Phần 10 và 15, hãy thêm các câu lệnh xử lý lỗi để chú ý đến các lỗi.

2. Acme Technologies Private Limited là một công ty phần mềm hàng đầu tại New York. Công ty đã đạt được nhiều giải thưởng như đại lý tốt nhất trong phát triển các công nghệ phần mềm. Công ty đã nhận được nhiều dự án mới về phát triển di động và web. Hiện nay, họ đang làm việc trên một dự án cơ sở dữ liệu cho Hệ thống quản lý tiền lương trong SQL Server 2012.

   Họ đã tạo ra cơ sở dữ liệu trên hệ thống quản lý tiền lương cho người lao động. Trong khi tạo ra các bảng, họ nhận được các loại lỗi khác nhau. Giả sử rằng bạn là quản trị viên cơ sở dữ liệu của Acme Technologies và tổ trưởng kỹ thuật đã giao cho bạn nhiệm vụ chỉnh sửa những lỗi đó. Thực hiện những bước sau:

   a. Viết các câu lệnh xử lý lỗi sử dụng cấu trúc TRY…CATCH cho cả hai câu lệnh bình thường cũng như thủ tục lưu trữ.

   b. Hiển thị thông tin lỗi sử dụng như sau:

      - **ERROR_NUMBER**
      - **ERROR_MESSAGE**
      - **ERROR_LINE**