عنوان مقاله: مدیریت خطای تراکنش در SQL Server
نویسنده مقاله: محمد سلیم آبادی
تاریخ انتشار: آبان ماه ۱۳۹۳
منبع: https://nikamooz.com/transaction-error-in-sql-server-management

مقدمه

یکی از مهمترین مباحث در مدیریت تراکنش در SQL Server در حقیقت مدیریت خطا در تراکنش‌ها می‌باشد، در این مقاله می‌خواهیم با به دام انداختن خطاهای تراکنش در SQL Server آشنا شویم

در این مقاله یاد خواهید گرفت:

- انواع روش های به دام انداختن خطا (Error Trapping)
- @@error
- TRY…CATCH
- مفهوم سطح شدت خطا (Error Severity Level)
- استفاده از تراکنش در TRY…CATCH و برعکس
- خطاهایی که به دام نمی افتند

SQL Server بطور پیش فرض در صورت بروز خطا در داخل یک تراکنش، آن را بصورت اتوماتیک و ضمنی برای شما ROLLBACK نخواهد کرد. پس ناچاریم با به دام انداختن خطا ها کار Rollback و صدور پیغام مقتضی را خود به عهده بگیریم. بنابراین لازم است کمی با بحث Error Handling در T-SQL آشنا شوید.

در مقاله مدیریت تراکنش در SQL Server (بررسی ویژگی های تراکنش و تعریف آن) به ابتدایی ترین روش مدیریت خطا که مدتها بود برنامه نویسان SQL Server از آن استفاده می کردند (یعنی بررسی مقدار متغیر @@error) اشاره ای مختصر و در حد نیاز داشتیم. اما این شیوه به دلیلی که در ادامه به آن اشاره خواهد شد کارامد نیست.

یکی از مشکلاتی که در این شیوه مطرح است، مربوط به مدت زمان اعتبار مقدار متغیر @@error است. در واقع اعتبار مقدار این متغیر تا قبل از اجرای عبارت بعدی است. لذا اگر در مکانی از کد، احتمال بروز خطا متصور است، حتما از یک متغیر محلی و ثبت مقدار @@error و یا پردازش مستقیم خطای رخ داده استفاده کنید.

برای پی بردن به این واقعیت در اینجا دو مثال آورده‌ام.

مثال ۱:

کدهای زیر را در یک پنجره Query اجرا کنید و نتیجه (قسمت Result) را مشاهده کنید. چه می‌بینید؟

```
SELECT 1/0
SELECT @@ERROR
SELECT @@ERRORprint 'hello world!'
```

اولین دستور با شکست مواجه خواهد شد و پیغام خطایی مبنی بر تقسیم بر صفر (Divide by zero) صادر خواهد کرد. پس متغیر @@error حاوی شماره مربوط به خطا خواهد شد (که در اینجا شماره خطا ۸۱۳۴ است)

دومین دستور مقدار متغیر @@error را بر میگرداند (که همان عدد مذکور است) و سپس مقدار صفر به متغیر @@error اختصاص داده می شود (که به معنای اجرای موفقیت آمیز آخرین دستور است).

حالا سومین دستور مقدار ۰ را به جای عدد ۸۱۳۴ برمیگرداند.

این موضوع محدود به اینگونه دستورات نمیشود. بر اساس مستندات Microsoft حتی عبارات شرطی مثل IF مقدار @@error را reset می کنند.

مثال ۲:

```
SELECT 1/0
IF @@ERROR &gt; 0 SELECT @@ERROR
```

اولین @@error آخرین شماره خطا را نگه داشته است که عددی بزرگتر از صفر است. پس شرط دستور IF صحیح است در نتیجه @@error انتخاب می شود، اما دومین @@error که بعد از بررسی دستور IF اجرا می شود بر خلاف تصور شماره خطا مربوط به آخرین دستور که شرط IF باشد را نگهداشته است که عدد صفر است. به بیان دیگر دستور IF مقدار @@error را reset کرده است.

نکته ۱: لیست تمام پیغام های خطا در ویوی sys.messages قرار دارد. توجه داشته باشید که هر پیغام خطا به چند زبان در جدول وجود دارد. لذا تعداد سطرهای این جدول متناظر با تعداد کل خطاها نیست.

نکته ۲: در مقاله قبل برای اینکه بررسی کنیم آیا خطایی رخ داده است یا خیر، مقدار متغیر @@error را با عملگر "نامساوی با صفر" مقایسه میکردیم. اما بر اساس داده های این جدول خطایی با شماره ی کمتر از ۲۱ نداریم. در نتیجه بنظر میرسد عملگر "بزرگتر از صفر" کفایت کند. و از طرفی پیغام خطاهایی که کاربر تعریف می کند (user-defined) بواسطه ی sp_addmessage باید شماره ای بالاتر از پنجاه هزار داشته باشند.

پرسش از خواننده: آیا می‌توانید "نکته ۲" را نقض کنید؟

سطح شدت خطا (Error Severity Level)

Error ها دارای پارامتری به نام Severity هستند که شدت خطا را تعیین می کند. خطاهایی با شدت ۱۰ و کمتر از آن به عنوان هشدار (Warning) یا پیام های اطلاع راسانی در نظر گرفته می شوند. و Error هایی با سطح شدت ۱۱ و بالاتر به عنوان Error محسوب می شوند (فرقی نمی کند که این پیغام با دستور raiserror تولید شده یا توسط دستور دیگر).
نکته ای که قصد بیان آن را داشتم این است که @@error شماره خطاهایی با شدت ۱۰ و کمتر از آن را برنمیگرداند.

ساختار TRY…CATCH

می‌توان ادعا کرد که برای هر بخش از کد این امکان متصور است که در زمان اجرا به مشکلی پیش بینی نشده برخورد کرده و خطایی را تولید کند. در بیشتر مواقع این موضوع که بتوان راهکاری برای به دام انداختن این خطا پیدا کرد، از اهمیت ویژه ای برخوردار است. همانطور که مشاهده شد، برای آزمایش هر خط، از @@error استفاده کردیم که تعداد دستوراتمان را دو برابر می کند. واکنش بهتری نیز وجود دارد، به طوری که اگر یک مجموعه از عبارت ها، خطایی را در زمان اجرا تولید کرد، بتوان آنها را شناسایی و مهار نمود، اینجاست که ساختار TRY…CATCH مطرح می شود.

در T-SQL مشابه زبان برنامه نویسی Visual C# می توان Error ها را مورد پردازش قرار داد. این ساختار شامل دو بخش می باشد بلاک TRY و بلاک CATCH. زمانی که یک خطا در یکی از عبارات T-SQL ای که داخل بلاک TRY قرار دارد تشخیص داده شود کنترل پاس داده می شود به بلاک CATCH جایی که خطا پردازش می شود. شایان ذکر است که این ویژگی در نسخه ۲۰۰۵ به T-SQL افزوده شده است پس در نسخه ی ۲۰۰۰ به دنبال آن نگردید.

بعد از اینکه در بلاک CATCH خطا مدیریت شد کنترل منتقل می شود به اولین عبارت T-SQL که در ادامه ی عبارت END CATCH قرار دارد. در بلاک TRY عبارات T-SQL که بعد از عبارتی که موجب بروز خطا شد قرار دارند اجرا نخواهند شد.
اگر هیچ خطایی داخل بلاک TRY وجود نداشته باشد کنترل پاس داده می شود به عبارتی که بلافاصله بعد از دستور END CATCH وجود دارد (توضیحات تکمیلی را میتوانید از Book Online بدست آورید).

Syntax این ساختار:

```
BEGIN TRY
{T-SQL Statement}
END TRY
BEGIN CATCH
{T-SQL Statement}
END CATCH
```

همانند @@error این ساختار خطاهایی با شدت ۱۰ و کمتر از آن را Handle نمی کند. به این معنا که اجرای دستورات داخل بلاک TRY ادامه پیدا میکنند و کنترل به بلاک CATCH منتقل نمی شود.

داخل محدوده ی بلاک CATCH توابع سیستمی زیر به جهت بدست آوردن اطلاعات پیرامون خطای رخ داده می توانند مورد استفاده قرار گیرند مثل شماره خطا، شدت خطا، پیام خطا و غیره.

```
ERROR_NUMBER(),
ERROR_SEVERITY(),
ERROR_STATE(),
ERROR_PROCEDURE(),
ERROR_LINE(),
ERROR_MESSAGE()
```

اولین تابع مشابه @@error شماره خطا را برمیگرداند.
چهارمین تابع از بالا، نام SP یا Trigger ای را که خطا در آن اتفاق افتاده است را برمیگرداند.
نکته: توابع فوق خارج از محدوده ی بلاک CATCH مقدار NULL را بر میگردانند.
متاسفانه تمام خطاها توسط این ساختار شناسایی نمی شوند. مثل خطای ارجاع معلق (Object Name Resolution). بطور نمونه میخواهیم داده های جدولی که وجود خارجی ندارد (قبلا ایجاد نشده است) را بازیابی کنیم.

```
begin try
select * from NonExistentTable
end try
begin catch
print 'error'
end catch
```

در اینجا کنترل به دست بلاک CATCH نمی افتد و پیغامی که مدنظر ما بود نمایش داده نمی شود.
حالا اگر همین دستور SELECT را داخل یک SP اجرا کنیم آنگاه توسط بلاک Catch مدیریت خواهد شد، به این صورت:

```
create procedure usp_example
as select * from NonExistentTable

begin try
execute usp_example
```

```
end try
begin catch
select ERROR_MESSAGE() as message_error
end catch
```

استفاده از TRY...CATCH در تراکنش

مثال ۱:

مثال زیر نشان می دهد که چگونه بلاک TRY...CATCH داخل یک تراکنش کار می کند. عبارت داخل بلاک TRY خطای "نقض قید" تولید می کند.

```
USE AdventureWorks2012;
GO
BEGIN TRANSACTION;

BEGIN TRY
-- Generate a constraint violation error.
DELETE FROM Production.Product
WHERE ProductID = 980;
END TRY
BEGIN CATCH
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;

IF @@TRANCOUNT &gt; 0
ROLLBACK TRANSACTION;
END CATCH;

IF @@TRANCOUNT &gt; 0
COMMIT TRANSACTION;
GO
```

توضیح اجمالی اینکه در این مثال عبارت موجود در بلاک Try قیدی را نقض کرده در نتیجه کنترل به بلاک Catch منتقل می شود و در آنجا اطلاعاتی راجب خطای رخ داده صادر می شه و در صورتی که تراکنش فعالی وجود داشته باشه تراکنش Rollback می شه و کنترل اجرا برنامه به دستور بعد از END CATCH منتقل می شود و سپس برای اینکه تراکنشی که قبلا Rollback شده Commit نشود بررسی می کنیم که آیا تراکنش فعالی داریم یا خیر در صورتی که تراکنش فعالی داشتیم به این معناست که عبارات بدون خطا اجرا شدن پس تراکنش را Commit می کنیم.

مثال ۲:

در این مثال تراکنش داخل بلاک TRY تعریف شده است.

پرسش از خواننده: چه لزومی دارد داخل بلاک Catch مقدار @@trancount بررسی شود؟ چرا که بنظر میرسد زمانی که کنترل به بلاک Catch منتقل شده است هنوز عمل commit صورت نگرفته است پس قطعا تراکنش فعال داریم.

```
BEGIN TRY
BEGIN TRANSACTION

INSERT INTO dbo.invoice_header
(invoice_number, client_number)
VALUES (2367, 19)

INSERT INTO dbo.invoice_detail
(invoice_number, line_number, part_number)
VALUES (2367, 1, 84367)

COMMIT TRANSACTION
END TRY
BEGIN CATCH
IF @@TRANCOUNT() &gt; 0 ROLLBACK TRANSACTION
-- And do some cool error handling
END CATCH
```

خطاهایی که به دام نمی‌افتند:

خطاهایی هستند که شما آنها را نمی توانید بصورت سنتی (@@error) به دام بی اندازید. مثل Conversion Failed که زمانی رخ میدهد که سعی کنید مقدار رشته ای را در ستونی عددی درج کنید یا امثالهم.

این کد را امتحان کنید تا در عمل این موضوع را مشاهده کنید:

```
--Declaring a variable table
declare @t table(i int);

select 'statment befor error';

--Conversion failed
insert into @t values ('string_value');

select 'statement after error', @@ERROR;
```

توضیح مثال: ابتدا یک جدول (از نوع متغیری) ایجاد می کنیم تا به واسطه ی آن عمل درج داشته باشیم. سپس یک مقدار را انتخاب می کنیم (statement befor error) که در خروجی نیز ظاهر می شود (قسمت Result) سپس سعی میکنیم مقدار string_value را داخل ستون i که از نوع integer هست درج کنیم که با شکست مواجه می شود. که این شکست باعث

break شدن batch نیز می شود در نتیجه دستورات بعدی موجود در این batch نیز اجرا نخواهد شد پس توسط تابع @@error نمی توانیم کد خطا را بدست آوریم.
البته می توانید کد فوق را به دو batch توسط GO تقسیم کنید. اما این ایده ی خوبی نیست. آیا میتوان داخل بدنه ی Stored Procedure از GO استفاده نمود؟

اما این مشکل در ساختار TRY…CATCH وجود ندارد. امتحان کنید:

```
--Declaring a variable table
declare @t table(i int);

begin try
--Conversion failed
insert into @t values ('string_value');
end try
begin catch
select @@ERROR as error_number
end catch
```