MS SQL 2011 (Denali) - Throw

Служебное слово Throw (NEW)

Новое полезное дополнение для SQL Server 2011 (Denali) – выражение **Throw**. Разработчики на .Net уже догадались наверно, где и как оно будет использоваться.

Это слово может использоваться в сочетании с управляющей конструкцией Try…Catch и позволяет послать уведомление о возникновении ошибки времени исполнения. Когда возникает исключение, программа ищет ближайший по иерархии вверх блок Catch который может обработать исключение. Используя это выражение внутри блока Catch можно изменить вывод ошибки. Более того, теперь вызывать исключение можно произвольно в любом месте скрипта.

Далее рассмотрим различные исключения, которые бросает SQL Server начиная с версии 2000.

Для всех рассматриваемых случаев будет использоваться таблица *tbl_ExceptionTest*.

	Column Name	Data Type	Allow Nulls
🔑	Id	int	☐
▶	[Phone Number]	int	☐

Для того, чтобы не протыкивать дизайнер мышью, можно выполнить следующий скрипт для создания искомой таблицы (сгенерировано автоматически).

```
IF EXISTS (SELECT * FROM sys.objects WHERE name = 'tbl_ExceptionTest' AND type = 'U')
    DROP TABLE tbl_ExceptionTest
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[tbl_ExceptionTest](
        [Id] [int] IDENTITY(1,1) NOT NULL,
        [Phone Number] [int] NOT NULL,
 CONSTRAINT [PK_tbl_ExceptionTest] PRIMARY KEY CLUSTERED
(
        [Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
```

Далее будем пытаться добавить в таблицу несколько записей и при внесении неподходящих данных в колонку Phone Number генерировать исключения.

Обработка ошибок в SQL Server 2000 (Sphinx)

Использование глобальной переменной @@ERROR

Возвращаясь во времена использования SQL Server 2000, вспоминаем что использование переменной @@Error было на тот момент самым прогрессивным и эффективным способом

обработки ошибок. Данная переменная отвечала за возврат целочисленного значения ошибки, которое произошло в последнем выполненном выражении. Значение ошибки могло быть как положительным, так и отрицательным, лишь 0 указывал на успешность выполнения операции. Значение переменной менялось после каждого выполненного выражения.

Посмотрим на использование @@Error в действии.

```
--If the #tblExceptionTest object exists in the tempdb, then drop it
-- Если таблица #tblExceptionTest уже создана, то удалить ее.
If OBJECT_ID('tempdb..#tblExceptionTest') Is not null
Begin
        Drop Table #tblExceptionTest
End

--Create the #tblExceptionTest temporary table
-- Создать временную таблицу #tblExceptionTest
Create Table #tblExceptionTest (Id int identity, [Phone Number] varchar(10) not null)

--Beigns the transaction
--Начало транзакции
Begin Transaction TranExcp__2000_@@Error

        --Variable Declarations
        -- объявление переменных
   Declare @ErrorNum int  -- a local variable to store the @@ERROR value
-- локальная переменная хранящая номер ошибки из @@ERROR
   Declare @i int -- a local variable that acts as a counter
-- локальная переменная работающая как счетчик

   --Initialize variables
-- инициализация переменных
   Set @i =1

   --Start Operation
-- начало операции
   While(@i <= 4)
        Begin
                -- Simulating the situation where a user tries to enter a null value
to the Phone Number column
-- симуляция ситуации когда пользователь пытается ввести null в колонку Phone Number
                If(@i = 4)
                        Begin
                                Insert into #tblExceptionTest([Phone Number])
Values(null)
                                Set @ErrorNum = @@ERROR
                        End
                Else
                        -- All records will be inserted successfully
                        -- все данные будут внесены успешно
                        Begin
                                Insert into #tblExceptionTest([Phone Number])
Values(cast(@i as varchar(2)) + '12345678')
                        End
                Set @i = @i +1

        End -- конец while

        -- If there is any error, notify that and roll back the transaction
-- если есть ошибки, вывести их и откатить транзакцию
   If @ErrorNum <> 0
           Begin
                  Rollback Transaction TranExcp__2000_@@Error
                  --Raise the custom error
                -- показать специальное сообщение об ошибке
                  RAISERROR ('Attempt to insert null value in [Phone Number] is not
allowed',16,1)
           End
   -- Commit the changes
```

```
-- сохранить изменения
   Else If @ErrorNum = 0
                Begin
                        Commit Transaction TranExcp__2000_@@Error
                End
   --Display the records
-- показать записи
   Select * from #tblExceptionTest
```

Общий смысл скрипта сводиться к тому, что в последней записи мы намеренно вызываем ошибку и читаем ее значение из локальной переменной. Если значение ошибки не равно нулю, то показываем осмысленное предупреждение пользователю. Если ошибок нет, то сохраняем результаты.

Выполнение данного скрипта приведет к появлению ошибки, как показано ниже

Msg 515, Level 16, State 2, Line 26 Cannot insert the value NULL into column 'Phone Number', table 'tempdb.dbo.#tblExceptionTest_____000000000023'; column does not allow nulls. INSERT fails. The statement has been terminated. Msg 50000, Level 16, State 1, Line 43 Attempt to insert null value in [Phone Number] is not allowed

Естественно, что вся транзакция откатится назад и ничего не будет внесено в таблицу.

Недостатки подхода с использованием @@Error

- Значение переменной @@Error должно быть проверено **сразу** после выполнения запроса/команды.
- Так как @@Error постоянно меняется, то мы вынуждены заводить отдельную переменную для сохранения и вывода кода ошибки.
- Вместе со специальным сообщением об ошибке указывающей на логический смысл ошибки выводится техническая информация, которая пользователям не интересна.

Если вы хотите узнать больше деталей и нюансов по использованию @@Error, то советую обратиться к статье про @@Error.

Использование глобальной переменной @@TRANCOUNT

Эта переменная возвращает количество транзакций выполняющихся в момент обращения к переменной. Из описания уже понятно, что она постоянна примерно в той же мере, что и @@ERROR, т.е. постоянно меняется во время исполнения транзакций. Это опять подводит нас к тому, чтобы использовать локальные переменные для хранения значений в интересующий момент времени.

Каждый вызов BEGIN TRANSACTION увеличивает значение @@TRANCOUNT на 1 и каждый вызов COMMIT TRANSACTION уменьшает ее значение на 1. ROLLBACK TRANSACTION не изменяет значения @@TRANCOUNT. Записи считаются внесенными только когда значение @@TRANCOUNT достигнет 0.

Рассмотрим использование @@TRANCOUNT на следующем примере.

```
--If the #tblExceptionTest object exists in the tempdb, then drop it
-- если таблица #tblExceptionTest существует, то удаляем ее
If OBJECT_ID('tempdb..#tblExceptionTest') Is not null
Begin
        Drop Table #tblExceptionTest
End
```

```
--Create the #tblExceptionTest temporary table
-- создаем временную таблицу
Create Table #tblExceptionTest (Id int identity, [Phone Number] varchar(10) not null)

--Beigns the transaction
-- начинаем транзакцию
Begin Transaction TranExcp__2000_@@TRANCOUNT

        --Variable Declarations
        --объявление переменных
   Declare @TransactionCount int  -- a local variable to store the @@TRANCOUNT value
-- локальная переменная хранящая значение @@TRANCOUNT
   Declare @i int -- a local variable that acts as a counter
-- счетчик

   --Initialize variables
        -- инициализация счетчика
   Set @i =1

   --Start Operation
-- старт эксперимента
   While(@i <= 4)
        Begin
                -- Simulating the situation where a user tries to enter a null value
to the Phone Number column
                -- симуляция ситуации когда пользователь пытается ввести null в
колонку Phone Number
                If(@i = 4)
                        Begin
                                Insert into #tblExceptionTest([Phone Number])
Values(null)
                                Set @TransactionCount = @@TRANCOUNT
                        End
                Else
                        -- All records will be inserted successfully
                        -- все записи будут внесены успешно
                        Begin
                                Insert into #tblExceptionTest([Phone Number])
Values(cast(@i as varchar(2)) + '12345678')
                        End
                Set @i = @i +1

        End -- конец while

        -- If there is any error, notify that and roll back the transaction
-- если есть ошибки, то уведомить об этом и откатить транзакцию
   If @TransactionCount <> 0
            Begin
                    Rollback Transaction TranExcp__2000_@@TRANCOUNT
                     --Raise the custom error
                -- показ специальной ошибки
                  RAISERROR ('Attempt to insert null value in [Phone Number] is not
allowed',16,1)
            End
   -- Commit the changes
-- подтверждение изменений
   Else If @TransactionCount = 0
                Begin
                        Commit Transaction TranExcp__2000_@@TRANCOUNT
                End
   --Display the records
-- вывод записей
   Select * from #tblExceptionTest
```

В данном скрипте мы опираемся на количество закрытых транзакций. Транзакции могут быть вложенные, так что такой способ имеет право на существование.

Для получения дополнительной информации по @@TRANCOUNT обратитесь на MSDN.

Использование глобальной переменной @@ROWCOUNT

Данная переменная возвращает количество измененных строк в результате выполнения запроса/команды.

Поведение такое же, как и у предыдущих двух, так что сохраняем промежуточные результаты в локальную переменную для последующего анализа.

Пример:

```
--If the #tblExceptionTest object exists in the tempdb, then drop it
If OBJECT_ID('tempdb..#tblExceptionTest') Is not null
Begin
        Drop Table #tblExceptionTest
End

--Create the #tblExceptionTest temporary table
Create Table #tblExceptionTest (Id int identity, [Phone Number] varchar(10) not null)

--Beigns the transaction
Begin Transaction TranExcp__2000_@@ROWCOUNT
--Create a Save Point
Save Transaction TranExcp__SavePoint

        --Variable Declarations
   Declare @RowCount int  -- a local variable to store the @@ROWCOUNT value
   Declare @i int -- a local variable that acts as a counter

   --Initialize variables
   Set @i =1

   --Start Operation
   While(@i <= 4)
        Begin
                -- Simulating the situation where a user tries to enter a null value
to the Phone Number column
                If(@i = 4)
                        Begin
                                Insert into #tblExceptionTest([Phone Number])
Values(null)
                                Set @RowCount = @@ROWCOUNT
                        End
                Else
                        -- All records will be inserted successfully
                        Begin
                                Insert into #tblExceptionTest([Phone Number])
Values(cast(@i as varchar(2)) + '12345678')
                        End
                Set @i = @i +1

        End -- End of while

        -- If there is any error, notify that and roll back the transaction
   If @RowCount = 0
            Begin
                --Roll the transaction back to the most recent save transaction
                Rollback Transaction TranExcp__SavePoint
                 --Raise the custom error
                RAISERROR ('Attempt to insert null value in [Phone Number] is not
allowed',16,1)
            End
   -- Commit the changes
   Else If @RowCount <> 0
                Begin
                        Commit Transaction TranExcp__2000_@@ROWCOUNT
                End
```

```
   --Display the records
   Select * from #tblExceptionTest
```

В данном случае мы ожидаем, что вставится одна запись в таблицу, но если количество вставленных записей равно нулю, то явно что-то не в порядке.

Для того, чтобы получить больше деталей по использованию @@ROWCOUNT читайте MSDN.

Обработка ошибок в SQL Server 2005/2008 (Yukon/Katmai)

После вывода на рынок SQL Server 2005 и развития его идей в SQL Server 2008 у разработчиков на TSql появился новый блок Try…Catch. Теперь стало возможно перехватывать исключения без потери транзакционного контекста.

Пример на использование блока Try … Catch.

```
--If the #tblExceptionTest object exists in the tempdb, then drop it
If OBJECT_ID('tempdb..#tblExceptionTest') Is not null
Begin
        Drop Table #tblExceptionTest
End

Begin TRY
   --Create the #tblExceptionTest temporary table
        Create Table #tblExceptionTest (Id int identity, [Phone Number] varchar(10)
not null)

    Begin Transaction TranExcpHandlingTest_2005_2008
          --Variable Declarations
       Declare @i int  -- a local variable that acts as a counter

           --Initialize variables
       Set @i =1

       --Start Operation
       While(@i <= 4)
        Begin
                          -- Simulating the situation where a user tries to enter a
null value to the Phone Number column
            If(@i = 4)
                Begin
                    Insert into #tblExceptionTest([Phone Number]) Values(null)
                End
            Else
                -- All records will be inserted successfully
                Begin
                    Insert into #tblExceptionTest([Phone Number]) Values(cast(@i as
varchar(2)) + '12345678')
                End
            Set @i = @i +1

        End -- End of while

                 --If everything goes smooth, then commit the transaction
        Commit Transaction TranExcpHandlingTest_2005_2008

End Try
Begin Catch
          --Handle the error
       Begin
                   --Rollback the transaction
          Rollback Transaction TranExcpHandlingTest_2005_2008
                   --Raise the custom error
```

```
            RAISERROR ('Attempt to insert null value in [Phone Number] is not
allowed',16,1)
        End
End Catch

--Display the records
Select * From #tblExceptionTest
```

В примере больше не используется вспомогательных переменных для определения ошибки выполнения скрипта по косвенным признакам.

После запуска скрипта получим сообщение следующего вида:

After execution, we will receive the below

Msg 50000, Level 16, State 1, Line 45 Attempt to insert null value in [Phone Number] is not allowed

Как вы уже наверно заметили, на этот раз вывелось только то, что было задано в сообщении об ошибке. Никаких дополнительных, смущающих пользователя сообщений, SQL Server не показал. Выполняемый код обрамлен в блоке try и обработка ошибки в блоке catch. Получается чистый и ясный для понимания код. Если весь желаемый код прошел без ошибок, то код из блока Catch не будет вызван.

Самое важное то, что Catch блок представляет набор функций для детального разбора причин ошибки и возможность информирования пользователя на должном уровне. Функции для разбора исключительной ситуации:

- ERROR_NUMBER
- ERROR_SEVERITY
- ERROR_STATE
- ERROR_LINE
- ERROR_PROCEDURE
- ERROR_MESSAGE

С помощью этих функций попробуем переписать Catch блок скрипта, что бы представлен до этого.

```
Begin Catch
           --Handle the error
       Begin
                   --Rollback the transaction
          Rollback Transaction TranExcpHandlingTest_2005_2008
                   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;
       End
End Catch
```

Теперь мы получим такой ответ от сервера:

	ErrorNumber	ErrorSeverity	ErrorState	ErrorProcedure	ErrorLine	ErrorMessage
1	515	16	2	NULL	24	Cannot insert the value NULL into column 'Phone ...

Недостатки использования функции RiseError

1 Если вспомнить, что показывала эта функция вызванная в Catch блоке, то заметим, что она ссылалась на строку номер 45, как источник проблем.

```
Msg 50000, Level 16, State 1, Line 45
Attempt to insert null value in [Phone Number] is not allowed
```

Однако в действительности ошибка произошла в строке номер 24, так где было написано

Insert into #tblExceptionTest([Phone Number]) Values(null)

В то время как функция ERROR_LINE() возвращает всегда реальное место возникновения ошибки. Еще один способ, чтобы показать работу новых функций будет такой:

```
Begin Catch
          --Handle the error
       Begin
                  --Rollback the transaction
             Rollback Transaction TranExcpHandlingTest_2005_2008
               DECLARE @errNumber INT = ERROR_NUMBER()
               DECLARE @errMessage VARCHAR(500) = 'Attempt to insert null value in [Phone
Number] is not allowed'
            --Raise the custom error
              RAISERROR('Error Number: %d, Message: %s', 16, 1, @errNumber, @errMessage)
       End
End Catch
```

В этом случае движок SQL Server выдаст такое сообщение:

```
Msg 50000, Level 16, State 1, Line 45
Error Number: 515, Message: Attempt to insert null value in [Phone Number] is not allowed
```

Из чего можно заключить, что использование RiseError не дает возможности указать на реальное место в скрипте, где произошла исключительная ситуация.

2 Следующий недостаток функции RiseError состоит в том, что нет возможности повторно инициировать тоже самое исключение, для передачи вверх по иерархии вызовов. Так, если переписать блок Catch как показано ниже

```
Begin Catch
          --Handle the error
       Begin
                  --Rollback the transaction
             Rollback Transaction TranExcpHandlingTest_2005_2008
                  --Raise the custom error
                  RAISERROR(515, 16, 1)
       End
End Catch
```

То полученное сообщение об ошибке будет таким:

Msg 2732, Level 16, State 1, Line 46 Error number 515 is invalid. The number must be from 13000 through 2147483647 and it cannot be 50000

Причной этого является то, что для инициирования нового сообщения об ошибке, номер ошибки должен содержаться в таблице **sys.messages**.

Для более детального изучения функции RiseError, рекомендуется к прочтению:

- Детальный взгляд на RiseError в SQL Server 2005
- Использование RaiseError

Обработка ошибок в SQL Server 2011 (Denali)

Упомянутые выше недостатки функции RiseError могут быть успешно преодолены с помощью **новой команды Throw**.

Первый недостаток функции RiseError, на который мы указали ранее, невозможность сослаться на точную строку возникновения ошибки. Рассмотрим насколько далеко от места возникновения ошибки мы оказываемся при использовании команды Throw.

Перепишем блок Catch с использованием команды Throw.

```
Begin Catch
           --Handle the error
       Begin
                   --Rollback the transaction
          Rollback Transaction TranExcpHandlingTest_2011;
                   --Throw the error
          THROW
       End
End Catch
```

Вывод будет таким:

```
Msg 515, Level 16, State 2, Line 24
Cannot insert the value NULL into column 'Phone Number', table 'tempdb.dbo.#tblExceptionTest_____
000000000085'; column does not allow nulls. INSERT fails.
```

Это точно то место, где произошла ошибка. Что ж, работает пока на отлично.

Вторым недостатком было то, что функция RiseError не может повторно инициировать исключение потому, что RAISE ERROR ожидает номер ошибки, который хранится в таблице sys.messages. Команда Throw не ожидает, что номер ошибки должен быть из диапазона системной таблицы sys.messages, однако номер можно задать из диапазона от 50000 до 2147483647 включая обе границы.

Снова изменим блок Catch в соответствии с новыми знаниями.

```
Begin Catch
           --Handle the error
       Begin
                   --Rollback the transaction
          Rollback Transaction TranExcpHandlingTest_2011;
                   --Throw the error
          THROW 50001,'Attempt to insert null value in [Phone Number] is not
allowed',1
       End
End Catch
```

Результатом возникновения исключения будет

Msg 50001, Level 16, State 1, Line 45 Attempt to insert null value in [Phone Number] is not allowed

На данный момент SQL Server предоставляет множество путей для отлова ошибок, но до сих пор не все ошибки могут быть пойманы с помощью блока Try…Catch. Например:

- Синтаксические ошибки отлавливаются редактором запросов в SSMS
- Неправильные имена объектов

Если попробовать подать на выполнение следующий скрипт:

```
Begin Try
         -- --Invalid object tblInvalid
     Insert Into tblInvalid(Id,DOB) Values(1,DATEADD(year,1,'2011-02-26'))
End Try

Begin Catch
          --Throw the error
     THROW
End Catch
```

Получим сообщение об ошибке следующего плана:

Msg 208, Level 16, State 0, Line 3 Invalid object name 'tblInvalid'.

Получается что почти невозможно перехватить такие типы ошибок.

Но. Как всегда есть небольшой трюк как осуществить желаемое. Основная идея в том, чтобы сделать две хранимых процедуры и вызывать одну из другой в блоке Try…Catch и ловить исключение. Для доказательства нашего предположения используем для экспериментов следующий скрипт.

```
 --Check if the stored procedure exists. If so drop it
-- проверить существование процедуры, если есть, то удалить
If Exists (Select * from sys.objects where name = 'usp_InternalStoredProc' and type =
'P')
    Drop Procedure usp_InternalStoredProc
Go
-- Create the internal stored procedure
-- создать внутреннюю хранимую процедуру
Create Procedure usp_InternalStoredProc
As
Begin
    Begin Transaction TranExcpHandlingTest_2011
       Begin Try
                        --Invalid object tblInvalid
                        -- обращение к несуществующему объекту
                        Insert Into tblInvalid(Id,DOB) Values(1,DATEADD(year,1,'2011-
02-26'))
                        --Commits the transaction
                        -- закрытие транзакции
                        Commit Transaction TranExcpHandlingTest_2011
                End Try
       Begin Catch
                   If @@TRANCOUNT > 0 Rollback Transaction TranExcpHandlingTest_2011
          Print 'In catch block of internal stored procedure.... throwing the
exception';
           -- Throw the exception
         -- инициирование исключения
          THROW
       End Catch
End
Go

-- Script for creating the External stored procedure
```

```
--Check if the stored procedure exists. If so drop it
-- скрипт для создания внешней хранимой процедуры
-- проверка существования процедуры, если есть, то удалить
If Exists (Select * from sys.objects where name = 'usp_ExternalStoredProc' and type =
'P')
    Drop Procedure usp_ExternalStoredProc
Go
-- Create the external stored procedure
-- создание внутренней хранимой процедуры
Create Procedure usp_ExternalStoredProc
As
Begin
    Begin Try
       --Call the internal stored procedure
        -- вызов внутренней процедуры
        Exec usp_InternalStoredProc
    End Try
    Begin Catch
        Print 'In catch block of external stored procedure.... throwing the
exception';
                 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;
        THROW
    End Catch
End
Go

--Executing the outer procedure
-- вызов внешней процедуры
Exec usp_ExternalStoredProc
```

При запуске процедуры **ExternalStoredProc** получим сообщение:

```
In catch block of external stored procedure.... throwing the exception
(1 row(s) affected)

Msg 208, Level 16, State 1, Procedure usp_InternalStoredProc, Line 8
Invalid object name 'tblInvalid'.
```

И панель Result отобразит следующие данные:

Results | Messages

	ErrorNumber	ErrorSeverity	ErrorState	ErrorProcedure	ErrorLine	ErrorMessage
1	208	16	1	usp_InternalStoredProc	8	Invalid object name 'tblInvalid'.

Что нам и требовалось!

Теперь немного объяснений как работает код. У нас есть 2 хранимых процедуры: *usp_InternalStoredProc* и *usp_ExternalStoredProc*. В *usp_InternalStoredProc* мы пытаемся вставить запись в несуществующую таблицу #tblInnerTempTable, в результате чего получаем исключительную ситуацию, которая в свою очередь отлавливается внешним блоком Catch, расположенным во внешней процедуре.

Более того, строка и текст ошибки полностью соответствуют нашим ожиданиям и указывают на точное место.

Очень важно не забыть закрыть точкой с запятой предстоящее перед THROW выражение во внешней процедуре. THROW должен быть новым набором команд. В противном случае получите ошибку

Incorrect syntax near 'THROW'.

Больше информации о THROW можно подчерпнуть из MSDN.

Дальше будет интереснее, так что оставайтесь на связи. Перевод.

Hard'n'heavy!

Violet Tape