Инструкции управления потоком (Инструкции FCL)

Ключевыми словами языка управления потоком Transact-SQL являются следующие:

```
BEGIN...END
BREAK
CONTINUE
GOTO
IF...ELSE
RETURN
TRY...CATCH
WAITFOR
WHILE
```

В сочетании с инструкциями языка управления потоком могут использоваться и другие конструкции Transact-SQL:

```
CASE
Комментарии (в стиле Си /* комментарий */ и в стиле Ада -- комментарий)
DECLARE
SET
EXECUTE
PRINT
RAISERROR
```

DECLARE

Переменные объявляются в теле пакета или процедуры при помощи инструкции DECLARE, а значения им присваиваются при помощи инструкций SET или SELECT. Т. о., областью локальной переменной является пакет или процедура, в которых она объявлена. После объявления все переменные инициализируются значением NULL. Переменная не может принадлежать к типу данных text, ntext или image.

Синтаксис

```
DECLARE  { @локальная_переменная [AS] тип_данных [ = константное_выражение ] } [ ,...n]
```

Примеры

```
DECLARE @find varchar(30) = 'Man%';
```

SET

Устанавливает локальную переменную, предварительно созданную при помощи инструкции DECLARE @локальная_переменная, в указанное значение.

Синтаксис

```
SET @локальная_переменная = выражение
```

Примеры

```
USE AdventureWorksLT;
GO
DECLARE @find varchar(30);
SET @find = 'Man%';
SELECT LastName, FirstName, Phone
FROM SalesLT.Customer
WHERE LastName LIKE @find;
```

LastName	FirstName	Phone
Manchepalli	Ajay	1 (11) 500 555-0174
Manzanares	Tomas	1 (11) 500 555-0178

```
USE AdventureWorksLT;
GO
DECLARE @rows int;
SET @rows = (SELECT COUNT(*) FROM SalesLT.Customer);
SELECT @rows AS N'Число строк';
```

Число строк
440

EXECUTE

Выполняет командную строку — строку символов, в которой содержится пакет Transact-SQL или один из следующих модулей: системная хранимая процедура, пользовательская хранимая процедура, скалярная пользовательская функция или расширенная хранимая процедура.

Синтаксис

```
[ { EXEC | EXECUTE } ]
    {
      [ @код_возврата = ]
      { имя_модуля | @переменная_именя_модуля }
        [ [ @параметр = ] { значение | @параметр [ OUTPUT ] | [ DEFAULT ] }
        ]
      [ ,...n ]
    }
[;]
```

Примеры

А. Вызов EXECUTE с передачей единственного аргумента

```
USE AdventureWorks;
GO
EXEC dbo.uspGetEmployeeManagers 6;
GO
```

При выполнении переменная может быть явно поименована

```
USE AdventureWorks;
GO
EXEC dbo.uspGetEmployeeManagers @EmployeeID = 6;
GO
```

Если приведенная инструкция является первой в пакете или сценарии или sqlcmd, то указание EXEC не требуется

```
USE AdventureWorks;
GO
dbo.uspGetEmployeeManagers 6;
GO
-- или
dbo.uspGetEmployeeManagers @EmployeeID = 6;
GO
```

Б. Передача нескольких аргументов

```
USE AdventureWorks;
GO
DECLARE @CheckDate datetime;
SET @CheckDate = GETDATE();
EXEC dbo.uspGetWhereUsedProductID 819, @CheckDate;
GO
```

В. Использование в EXECUTE переменной хранимой процедуры

```
DECLARE @proc_name varchar(30);
```

```
SET @proc_name = 'sys.sp_who';
EXEC @proc_name;
```

Г. Выполнение пользовательской функции с помощью EXECUTE

```
USE AdventureWorksLT;
GO
DECLARE @returnstatus nvarchar(15);
SET @returnstatus = NULL;
EXEC @returnstatus = dbo.ufnGetSalesOrderStatusText @Status = 2;
PRINT @returnstatus;
GO
```

PRINT

Возвращает клиенту пользовательское сообщение.

Синтаксис

```
PRINT строковая_константа | @локальная_переменная | строковое_выражение
```

Пример

```
DECLARE @PrintMessage NVARCHAR(50);
SET @PrintMessage = N'Это сообщение было напечатано '
    + RTRIM(CAST(GETDATE() AS NVARCHAR(30)))
    + N'.';
PRINT @PrintMessage;
GO
```

Для возвращения сообщений можно также использовать функцию RAISERROR.

RAISERROR

Создает сообщение об ошибке и запускает обработку ошибок для сеанса. Инструкция RAISERROR может либо ссылаться на определенное пользователем сообщение, находящееся в представлении каталога sys.messages, либо динамически создавать сообщение. Это сообщение возвращается как сообщение об ошибке сервера вызывающему приложению или соответствующему блоку CATCH конструкции TRY…CATCH.

Синтаксис

```
RAISERROR ( { номер_ошибки | строковая_константа | @локальная_переменная }
    { , серьезность_ошибки , состояние_ошибки }
    [ , аргумент [ ,...n ] ] )
```

Пример

```
DECLARE @ErrorMessage NVARCHAR(4000);
DECLARE @ErrorSeverity INT;
DECLARE @ErrorState INT;

SELECT  @ErrorMessage = ERROR_MESSAGE(),
        @ErrorSeverity = ERROR_SEVERITY(),
        @ErrorState = ERROR_STATE();

RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState);
```

Преимущества функции RAISERROR перед функцией PRINT:

- функция RAISERROR поддерживает вставку аргументов в строку сообщения об ошибке с помощью механизма, основанного на функции printf из стандартной библиотеки языка C;
- в дополнение к текстовому сообщению, функция RAISERROR может в сообщении указать уникальный номер ошибки, степень ее серьезности и код состояния;
- функция RAISERROR может быть использована для возвращения пользовательских сообщений, созданных с помощью системной хранимой процедуры sp_addmessage.

TRY...CATCH

Синтаксис

```
BEGIN TRY
     { sql_инструкция | блок_инструкций }
END TRY
BEGIN CATCH
     { sql_инструкция | блок_инструкций }
END CATCH
[ ; ]
```

Примеры

```
BEGIN TRY
    BEGIN TRANSACTION
    DELETE [Order Details] WHERE OrderID IN
        (SELECT OrderID FROM Orders WHERE CustomerID = 'ALFKI')
    DELETE Orders WHERE CustomerID = 'ALFKI'
    DELETE Customers WHERE CustomerID = 'ALFKI'
    PRINT 'committing deletes'
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION
    RETURN
END CATCH

USE AdventureWorksLT;
GO

BEGIN TRY
    -- Генерация ошибки деления на ноль.
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS N'Номер ошибки',
        ERROR_SEVERITY() AS N'Серьезность ошибки',
        ERROR_STATE() AS N'Состояние ошибки',
        ERROR_PROCEDURE() AS N'Имя хранимой процедуры',
        ERROR_LINE() AS N'Номер строки',
        ERROR_MESSAGE() AS N'Текст сообщения об ошибке';
END CATCH;
GO
```

Результаты выполнения этого сценария будут иметь вид

Номер ошибки	Серьезность ошибки	Состояние ошибки	Имя хранимой процедуры	Номер строки	Текст сообщения об ошибке
8134	16	1	NULL	4	Divide by zero error encountered.

WHILE

Ставит условие повторного выполнения SQL-инструкции или блока инструкций. Эти инструкции вызываются в цикле, пока указанное условие истинно. Вызовами инструкций в цикле WHILE можно контролировать из цикла с помощью ключевых слов BREAK и CONTINUE.

Синтаксис

```
WHILE булевское_выражение
     { sql_инструкция | блок_инструкций }
```

```
    [ BREAK ]
    { sql_инструкция | блок_инструкций }
    [ CONTINUE ]
    { sql_инструкция | блок_инструкций }
```

Пример

```
declare @i int set @i = 1
declare @sum int set @sum = 0
while @i <= 100
 begin
      set @sum = @sum + ROUND(RAND(),0)
      set @i = @i + 1
 end
select @sum
```

WAITFOR

Блокирует выполнение пакета, хранимой процедуры или транзакции до наступления указанного времени или интервала времени, либо заданная инструкция изменяет или возвращает, по крайней мере, одну строку.

Синтаксис

```
WAITFOR { DELAY 'период_времени_ожидания' | TIME 'время_завершения_инструкции' }
```

Пример

```
USE msdb;
EXECUTE sp_add_job @job_name = 'TestJob';
BEGIN
    WAITFOR TIME '22:20';
    EXECUTE sp_update_job @job_name = 'TestJob',
        @new_name = 'UpdatedJob';
END;
GO
```