Ins & Outs of sp_execute_external_script - II

A drilldown into sp_execute_external_script

Agenda

- SPEES
- Parameters
  - @parallel
  - @params
  - @parameter1
- sqlrutils
- Data Types

SPEES Signature

```
sp_execute_external_script
             @language = N'language',
             @script = N'script',
             [ @input_data_1 =  'input_data_1']
             [ , @input_data_1_name = N'input_data_1_name' ]
             [ , @output_data_1_name = N'output_data_1_name' ]
             [ , @parallel = 0 | 1 ]
             [ , @params = N'@parameter_name data_type [ OUT | OUTPUT ] [ ,...n ]' ]
             [ , @parameter1 = 'value1' [ OUT | OUTPUT ] [ ,...n ] ]
[WITH <execute_option>]
```

Parameters and SPEES

- What if your script requires parameters?
- How do we send in params to SPEES?

```
# set a variable
multiplier <- 5
#get the iris data set as a data frame
iris_dataset <- iris
# grab the setosa species
setosa <- iris[iris$Species == 'setosa',]
# calculate mean of the Sepal.Width for setosa
menSepWidth <- mean(setosa$Sepal.Width)
# use the multiplier to do some "stuff
iris_dataset$Sepal.Length <- iris_dataset$Sepal.Length * multiplier
# look at the resulting dataset
View(iris_dataset$Sepal.Length)
# print out the mean
print(menSepWidth)
```

sp_executesql

- SPEES is similar to sp_executesql.
- Executes a Transact-SQL statement or batch.
- The Transact-SQL statement or batch can contain embedded parameters.
- The parameters are defined by one string containing the definitions of all parameters (incl. data types, IN | OUT, etc.)
- The parameters defined in the string is then defined with name and value.
- For SPEES it works the same.

Parameter Definitions in SPEES

- To define parameters in SPEES you use @params.
- This defines the parameters you use, direction and data types.
- The parameters then needs to be defined with name and value
  - @parameter1, etc.
- The parameters are defined in the script by name, without @.

Parameter Code

```
DECLARE @meanOut float;
DECLARE @mult int = 5;
DECLARE @sp nvarchar(20) = 'setosa'
EXEC sp_execute_external_script
      @language = N'R',
      @script = N'
          setosa <- MyDataSet
          meanSepWidth <- mean(setosa$SepalWidth)
          multiplier <- multip
          setosa$SepalLength <- setosa$SepalLength * multiplier
          OutputDataSet <- data.frame(setosa$SepalLength)',
      @input_data_1 = N'SELECT * FROM dbo.tb_irisdata_full
                        WHERE Species = @specie',
      @input_data_1_name = N'MyDataSet',
      @params = N'@specie nvarchar(20), @multip int, @meanSepWidth float OUT',
      @specie = @sp,
      @multip = @mult,
      @meanSepWidth = @meanOut OUT;

SELECT  @meanOut AS MeanSepWidth;
GO
```

Productionalise

- In production you probably do not call sp_execute_external_script directly.
- You call it from inside a stored procedure.
  - send in the parameters into the "outer procedure"

Procedure

```
CREATE PROCEDURE dbo.pr_DoIrisStuff   @species nvarchar(20)
                                    , @multi int
                                    , @meanOut float OUT
AS
BEGIN
  EXEC sp_execute_external_script
      @language = N'R',
      @script = N'
          setosa <- MyDataSet
          meanSepWidth <- mean(setosa$SepalWidth)
          multiplier <- multip
          setosa$SepalLength <- setosa$SepalLength * multiplier
          OutputDataSet <- data.frame(setosa$SepalLength)',
      @input_data_1 = N'SELECT * FROM dbo.tb_irisdata_full
                        WHERE Species = @specie',
      @input_data_1_name = N'MyDataSet',
      @params = N'@specie nvarchar(20), @multip int, @meanSepWidth float OUT',
      @specie = @species, @multip = @multi, @meanSepWidth = @meanOut OUT;

  SET @meanOut = @meanOut;
END
GO
```

http://nielsberglund.com

Execute Procedure

```
DECLARE @meanOut float;
DECLARE @mult int = 5;
DECLARE @sp nvarchar(20) = 'setosa'

EXEC dbo.pr_DoIrisStuff   @species = @sp
                        , @multi = @mult
                        , @meanOut =  @meanOut OUT

SELECT  @meanOut AS MeanSepWidth;
```

sqlrutils

- sqlrutils is an R package.
- It helps the user to create an outer procedure.
- Methods to define input and output data, input and output parameters.
- Register the outer procedure with a database (optionally).
- Run the stored procedure from an R development environment.

Usage sqlrutils

- You run methods in the package from an R IDE.
- Best practice to turn your R script into a function.
- Define the different parameters.
- Generate the procedure

Code sqlrutils - I

```
library(sqlrutils)

irisFunc <- function(multip, MyDataSet, specie) {

  setosa <- MyDataSet[MyDataSet$Species == specie,]
  meanSepWidth <- mean(setosa$SepalWidth)
  multiplier <- multip
  setosa$SepalLength <- setosa$SepalLength * multiplier
  sepLength <- data.frame(setosa$SepalLength)
  retList <- list(sepLengthData = sepLength,
                  meanSepWidthValue = meanSepWidth)

  return(retList)
}
```

Code sqlrutils - II

```
inParam <- InputParameter("multip", "numeric")
inSpeci <- InputParameter("specie", "character")
inData <- InputData(name = "MyDataSet",
                    defaultQuery = paste0("SELECT * FROM dbo.tb_irisdata_full"))
outParam <- OutputParameter("meanSepWidthValue", "numeric")
outputData <- OutputData("sepLengthData")
```

```
sp <- StoredProcedure(irisFunc,
                      "pr_IrisFunc",
                      inData,
                      inParam,
                      inSpeci,
                      outParam,
                      outputData,
                      filePath = "C:\\Temp"
                      )
```

Execute Generated Proc

```
DECLARE @meanW float;
EXEC pr_IrisFunc @parallel_outer = 0,
                 @input_data_1_outer = N'SELECT * FROM dbo.tb_irisdata_full',
                 @multip_outer = 5,
                 @specie_outer = N'setosa',
                 @meanSepWidthValue_outer = @meanW output

SELECT @meanW
```

Other Parameters

- Two parameters we have not touched upon
  - `@parallel`: enables parallel execution (parallel qp req.).
  - `@r_rowsPerRead`: streams data into the script
- We cover them in the performance session

Data Types

- Certain data types not supported:
  - cursor
  - timestamp
  - datetime2, datetimeoffset, time
  - sql_variant
  - text, image
  - xml
  - hierarchyid, geometry, geography
  - CLR user-defined types
- Values can be CAST:ed to supported type.

Summary

- You can define your own parameters via @params and @parameter1.
- The parameters are referenced in the script with their name and without @.
- The R package sqlrutils can help to create stored procedures calling sp_execute_external_script
- Certain data types not supported.