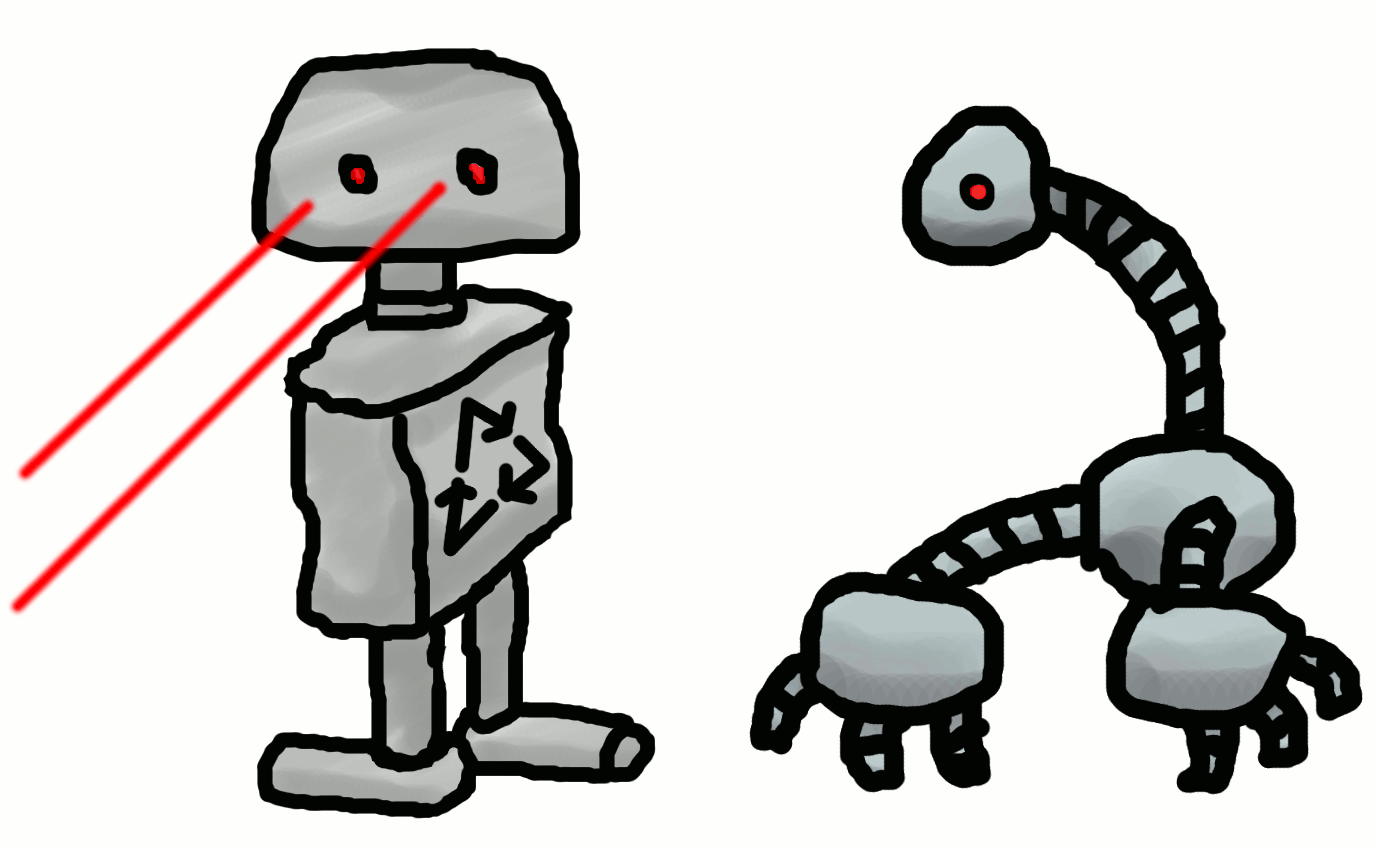
**[How To: Automate Continuous SQLServer Activity with Stored Procedures and Powershell Jobs](http://www.littlekendra.com/2011/03/02/poweshelljobs/" \o "How To: Automate Continuous SQLServer Activity with Stored Procedures and Powershell Jobs)**

by [Kendra Little](http://www.littlekendra.com/author/kendra-little/)on March 2, 2011

**The Goal[](http://www.littlekendra.com/wp-content/uploads/2011/03/Automation.gif)**

It’s often useful to be able to run a bunch of stored procedures in the background over a period of time against a test instance.

This can be nice for:

* Demos and presentations.
* Populating DMVs with data you can slice and dice.
* Learning to use things like extended events and server side trace (which are much more interesting with something to look at).
* Testing a variety of automation scripts.

This post shows you how to create several stored procedures for AdventureWorks2008R2 which will provide different result sets and have slightly different run times when run with a variety of parameters– in this case, individual letters of the alphabet.

You can then run PowerShell commands which start jobs in the background. Each job runs a stored procedure and loops through all letters of the alphabet, providing each one as a parameter. You can set the job to do that loop a configurable amount of times (the commands are set to 100). In other words, as given, each stored procedure will be run 2600 times. Since you’re running multiple jobs and they’re all going asynchronously in their own threads, you’ll have a variety of commands trying to run at the same time.

Optional: you can start the PowerShell jobs under different credentials if you need.

Alternatives: In the past, I’ve typically done things like this with T-SQL loops (often with dynamic SQL) and multiple Management Studio windows. This works OK, but it’s a little time consuming to open each window, paste everything in (or open multiple files), and start them all up. I find it much more convenient now to use scripts.

**Step 1: Create Stored Procedures with a single alphabet-based parameter**

Let’s get one thing clear: these procedures aren’t designed to run optimally, and they aren’t coded nicely.

You’ll notice these procedures have **all** sorts of problems. And that’s by design– my goals are for testing things around these, so it’s really a little better for me if they don’t play perfectly nice.

In other words, these sure ain’t for production. :)

[view source](http://www.littlekendra.com/2011/03/02/poweshelljobs/#viewSource)



[print](http://www.littlekendra.com/2011/03/02/poweshelljobs/#printSource)[?](http://www.littlekendra.com/2011/03/02/poweshelljobs/#about)

|  |  |
| --- | --- |
| 01 | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 02 | Jump in the kiddie pool | |

|  |  |
| --- | --- |
| 03 | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| 04 | USE AdventureWorks2008R2; | |

|  |  |  |
| --- | --- | --- |
| 05 | go | |
| 06 |  |

|  |  |  |
| --- | --- | --- |
| 07 | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |
| 08 | CREATE THE SCHEMA |

|  |  |
| --- | --- |
| 09 | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| 10 | IF SCHEMA\_ID(N'test') IS NULL | |

|  |  |  |
| --- | --- | --- |
| 11 | EXEC sp\_executesql N'CREATE SCHEMA test AUTHORIZATION dbo' | |
| 12 | GO |

|  |  |
| --- | --- |
| 13 |  |
| 14 | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |

|  |  |  |
| --- | --- | --- |
| 15 | CREATE Silly Stored Procedures in the Schema | |
| 16 | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

|  |  |
| --- | --- |
| 17 | IF OBJECT\_ID(N'test.EmployeeByLastName', 'P') IS NULL |
| 18 | EXEC sp\_executesql N'CREATE PROCEDURE test.EmployeeByLastName as return 0' | |

|  |  |
| --- | --- |
| 19 | GO |
| 20 | ALTER PROCEDURE test.EmployeeByLastName | |

|  |  |  |
| --- | --- | --- |
| 21 | @lName nvarchar(255) | |
| 22 | AS |

|  |  |  |
| --- | --- | --- |
| 23 | SELECT @lName = N'%' + @lName + N'%' | |
| 24 |  |

|  |  |
| --- | --- |
| 25 | select \* |
| 26 | FROM HumanResources.vEmployee | |

|  |  |  |
| --- | --- | --- |
| 27 | WHERE LastName LIKE @lName | |
| 28 | GO |

|  |  |
| --- | --- |
| 29 |  |
| 30 | IF OBJECT\_ID(N'test.EmployeeByFirstName', 'P') IS NULL | |

|  |  |  |
| --- | --- | --- |
| 31 | EXEC sp\_executesql N'CREATE PROCEDURE test.EmployeeByFirstName as return 0' | |
| 32 | GO |

|  |  |  |
| --- | --- | --- |
| 33 | ALTER PROCEDURE test.EmployeeByFirstName | |
| 34 | @fName nvarchar(255) |

|  |  |
| --- | --- |
| 35 | AS |
| 36 | SELECT @fName = '%' + @fName + '%' | |

|  |  |
| --- | --- |
| 37 |  |
| 38 | select \* | |

|  |  |  |
| --- | --- | --- |
| 39 | FROM HumanResources.vEmployee | |
| 40 | WHERE FirstName LIKE @fName |

|  |  |  |
| --- | --- | --- |
| 41 | GO | |
| 42 |  |

|  |  |
| --- | --- |
| 43 | IF OBJECT\_ID(N'test.EmployeeDepartmentHistoryByLastName', 'P') IS NULL |
| 44 | EXEC sp\_executesql N'CREATE PROCEDURE test.EmployeeDepartmentHistoryByLastName as return 0' | |

|  |  |
| --- | --- |
| 45 | GO |
| 46 | ALTER PROCEDURE test.EmployeeDepartmentHistoryByLastName | |

|  |  |  |
| --- | --- | --- |
| 47 | @lName nvarchar(255) | |
| 48 | AS |

|  |  |  |
| --- | --- | --- |
| 49 | SELECT @lName = N'%' + @lName + N'%' | |
| 50 |  |

|  |  |
| --- | --- |
| 51 | select \* |
| 52 | FROM HumanResources.vEmployeeDepartmentHistory | |

|  |  |  |
| --- | --- | --- |
| 53 | WHERE LastName LIKE @lName | |
| 54 | GO |

|  |  |
| --- | --- |
| 55 |  |
| 56 | IF OBJECT\_ID(N'test.EmployeeDepartmentHistoryByFirstName', 'P') IS NULL | |

|  |  |  |
| --- | --- | --- |
| 57 | EXEC sp\_executesql N'CREATE PROCEDURE test.EmployeeDepartmentHistoryByFirstName as return 0' | |
| 58 | GO |

|  |  |  |
| --- | --- | --- |
| 59 | ALTER PROCEDURE test.EmployeeDepartmentHistoryByFirstName | |
| 60 | @fName nvarchar(255) |

|  |  |
| --- | --- |
| 61 | AS |
| 62 | SELECT @fName = '%' + @fName + '%' | |

|  |  |
| --- | --- |
| 63 |  |
| 64 | select \* | |

|  |  |  |
| --- | --- | --- |
| 65 | FROM HumanResources.vEmployeeDepartmentHistory | |
| 66 | WHERE FirstName LIKE @fName |

|  |  |  |
| --- | --- | --- |
| 67 | GO | |
| 68 |  |

|  |  |
| --- | --- |
| 69 | IF OBJECT\_ID(N'test.ProductAndDescriptionByKeyword', 'P') IS NULL |
| 70 | EXEC sp\_executesql N'CREATE PROCEDURE test.ProductAndDescriptionByKeyword as return 0' | |

|  |  |
| --- | --- |
| 71 | GO |
| 72 | ALTER PROCEDURE test.ProductAndDescriptionByKeyword | |

|  |  |  |
| --- | --- | --- |
| 73 | @keyword nvarchar(255) | |
| 74 | AS |

|  |  |  |
| --- | --- | --- |
| 75 | SELECT @keyword = '%' + @keyword + '%' | |
| 76 |  |

|  |  |
| --- | --- |
| 77 | select \* |
| 78 | FROM Production.vProductAndDescription | |

|  |  |  |
| --- | --- | --- |
| 79 | WHERE Name LIKE @keyword OR ProductModel like @keyword OR description LIKE @keyword | |
| 80 | GO |

Once you’ve got the procedures written, you just need to set up your PowerShell commands.

**Step 2: Create PowerShell Jobs to Run the Procedures in Loops**

These commands use [PowerShell background jobs](http://msdn.microsoft.com/en-us/library/dd878288(v=vs.85).aspx).

Even if you don’t know PowerShell, if you look at these commands you can pretty easily pick out where the 1 to 100 loop is, where the a to z loop is, and what commands are being run.

Since the jobs are running to create load in the background and I don’t care about collecting query results, I pipe the output all to Out-Null.

[view source](http://www.littlekendra.com/2011/03/02/poweshelljobs/#viewSource)



[print](http://www.littlekendra.com/2011/03/02/poweshelljobs/#printSource)[?](http://www.littlekendra.com/2011/03/02/poweshelljobs/#about)

|  |  |
| --- | --- |
| 01 | #test.EmployeeByLastName |
| 02 | Start-Job -ScriptBlock {Import-Module sqlps; foreach($\_ in 1..100) {foreach ($\_ in [char]"a"..[char]"z") {Invoke-Sqlcmd -Query "exec test.EmployeeByLastName '$([char]$\_)'" -ServerInstance "YOURMACHINE\YOURINSTANCE" -Database AdventureWorks2008R2 | Out-Null }}} | |

|  |  |
| --- | --- |
| 03 |  |
| 04 | #"test.EmployeeByFirstName" | |

|  |  |  |
| --- | --- | --- |
| 05 | Start-Job -ScriptBlock {Import-Module sqlps; foreach($\_ in 1..100) {foreach ($\_ in [char]"a"..[char]"z") {Invoke-Sqlcmd -Query "exec test.EmployeeByFirstName '$([char]$\_)'" -ServerInstance "YOURMACHINE\YOURINSTANCE" -Database AdventureWorks2008R2 | Out-Null }}} | |
| 06 |  |

|  |  |
| --- | --- |
| 07 | #"test.EmployeeDepartmentHistoryByFirstName" |
| 08 | Start-Job -ScriptBlock {Import-Module sqlps; foreach($\_ in 1..100) {foreach ($\_ in [char]"a"..[char]"z") {Invoke-Sqlcmd -Query "exec test.EmployeeDepartmentHistoryByFirstName '$([char]$\_)'" -ServerInstance "YOURMACHINE\YOURINSTANCE" -Database AdventureWorks2008R2 | Out-Null }}} | |

|  |  |
| --- | --- |
| 09 |  |
| 10 | #"test.EmployeeDepartmentHistoryByLastName" | |

|  |  |  |
| --- | --- | --- |
| 11 | Start-Job -ScriptBlock {Import-Module sqlps; foreach($\_ in 1..100) {foreach ($\_ in [char]"a"..[char]"z") {Invoke-Sqlcmd -Query "exec test.EmployeeDepartmentHistoryByLastName '$([char]$\_)'" -ServerInstance "YOURMACHINE\YOURINSTANCE" -Database AdventureWorks2008R2 | Out-Null }}} | |
| 12 |  |

|  |  |
| --- | --- |
| 13 | #"test.ProductAndDescriptionByKeyword" |
| 14 | Start-Job -ScriptBlock {Import-Module sqlps; foreach($\_ in 1..100) {foreach ($\_ in [char]"a"..[char]"z") {Invoke-Sqlcmd -Query "exec test.ProductAndDescriptionByKeyword '$([char]$\_)'" -ServerInstance "YOURMACHINE\YOURINSTANCE" -Database AdventureWorks2008R2 | Out-Null }}} | |

Each command will start an asynchronous background job.

**Step 3: Manage Jobs (if needed)**

Once the jobs are running in the background, you may want to check on their status. You can do so by running:

[view source](http://www.littlekendra.com/2011/03/02/poweshelljobs/#viewSource)



[print](http://www.littlekendra.com/2011/03/02/poweshelljobs/#printSource)[?](http://www.littlekendra.com/2011/03/02/poweshelljobs/#about)

|  |  |
| --- | --- |
| 01 | get-job |

if you want to remove a job from the list, you can use Remove-Job with the job number, or you can remove all jobs (whether or not they are running) with:

[view source](http://www.littlekendra.com/2011/03/02/poweshelljobs/#viewSource)



[print](http://www.littlekendra.com/2011/03/02/poweshelljobs/#printSource)[?](http://www.littlekendra.com/2011/03/02/poweshelljobs/#about)

|  |  |
| --- | --- |
| 01 | Remove-Job \* -Force |

If you want to see the output of a job, you can use Receive-Job– supply the jobnumber. If you’re troubleshooting and want to see errors, you probably want to remove | Out-Null from the command that starts the job, and use a fewer number of loops. Then you can can receive the job’s output and see any errors.

[view source](http://www.littlekendra.com/2011/03/02/poweshelljobs/#viewSource)



[print](http://www.littlekendra.com/2011/03/02/poweshelljobs/#printSource)[?](http://www.littlekendra.com/2011/03/02/poweshelljobs/#about)

|  |  |
| --- | --- |
| 01 | Receive-Job JOBNUMBER |

Tags: [Featured](http://www.littlekendra.com/tag/featured/)