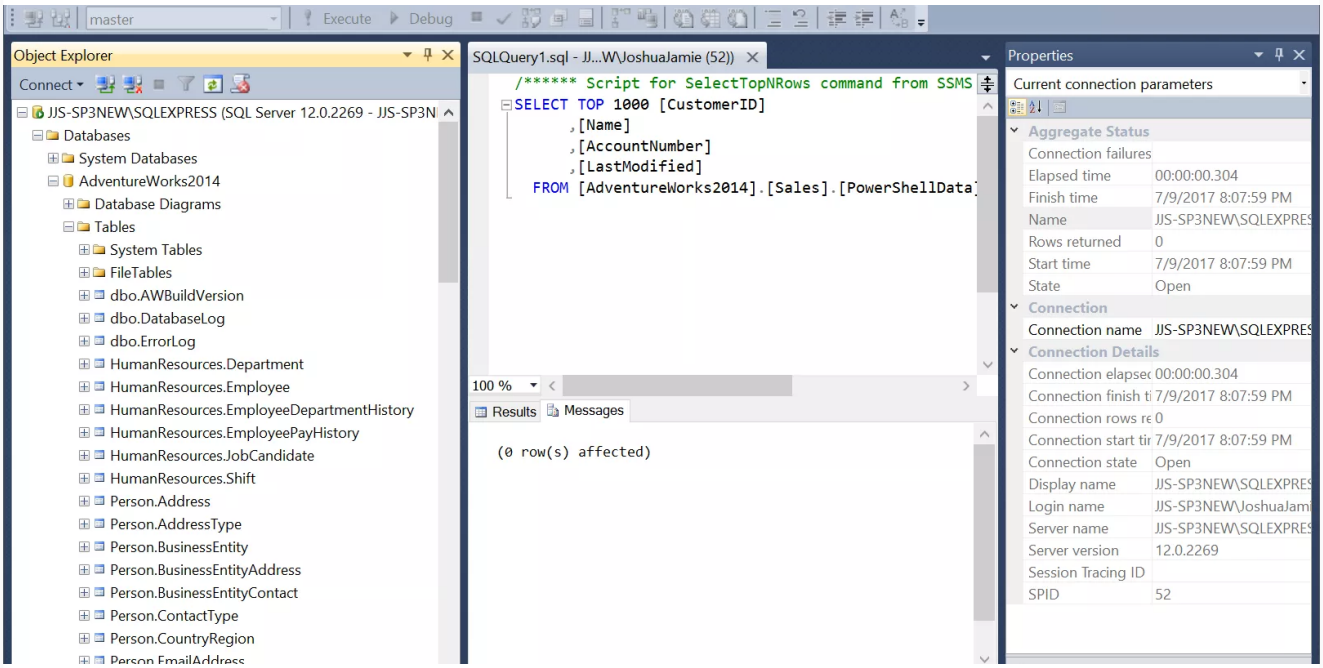
Virtually Sober
If there is free booze and Virtualization; I'm there!

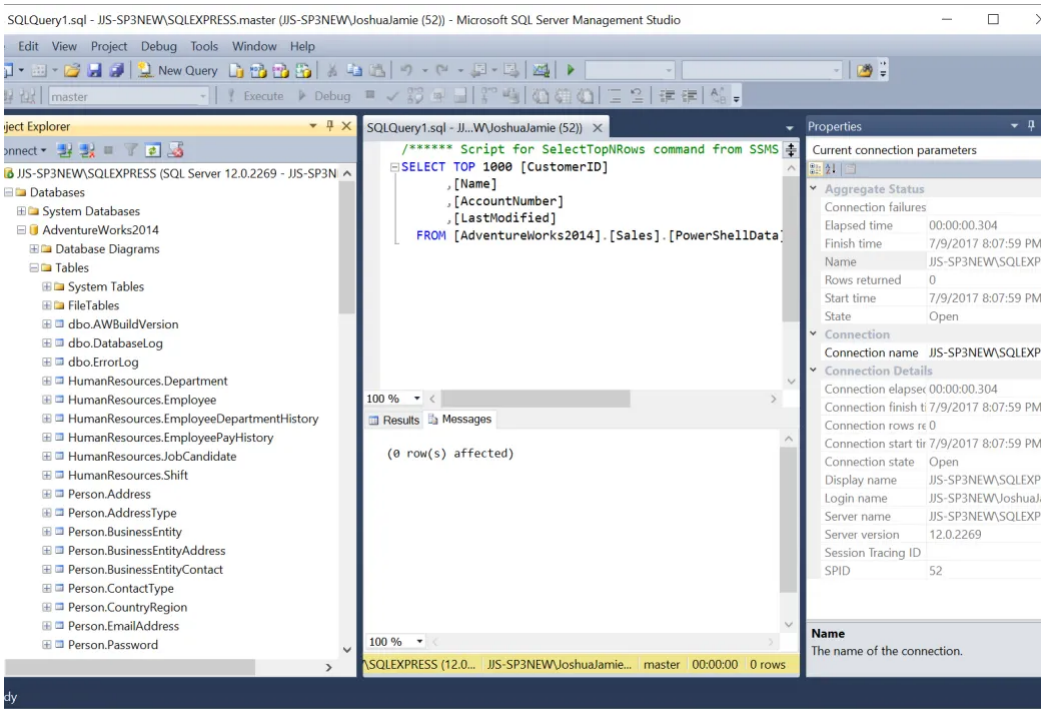
Working with SQL databases using PowerShell

6

Published July 10, 2017 by Joshua Stenhouse



One of the things I love about PowerShell is the sheer number of use cases you can address. A perfect example is working with SQL databases. If you think you need to wait for a DBA to give you data from a SQL database then you'd be wrong. You can use SELECT queries to pull data directly into PowerShell from any SQL database and not only that, you can also INSERT, UPDATE, DELETE or whatever else you desire, all from the comfort your scripts using invoke-sqlcmd in the Sqlserver PowerShell module.



If you've never connected to a SQL Instance using PowerShell before then I recommend reading this newer blog post here with pre-built examples:

https://virtuallysober.com/2018/05/30/connecting-to-microsoft-sql-databases-using-powershell-invoke-sqlcmd/

To test you will first need a SQL database! I recommend not using something in production, better to download and install SQL Server Express 2014/2016 then restore the "Adventure Works 2014" database (which I will be using to demonstrate). Download it from here:

Adventure Works 2014 Full Database Backup.zip

With a SQL instance installed and database to query let's start by loading the SqlServer PowerShell module. In SQL 2014 it was a standalone installer, initially in 2016 builds it was bundled on the host, but it has since been moved into the online PowerShell Gallery. I recommend using PowerShell 5.x onwards. If you're not sure what version you are on type $PSVersionTable to see.

All we need is internet access on the host running the script to install and import the SqlServer module:

```
################################################
# Checking to see if the SqlServer module is already installed, if not
installing it
```

```
##############################################
$SQLModuleCheck = Get-Module -ListAvailable SqlServer
if ($SQLModuleCheck -eq $null)
{
write-host "SqlServer Module Not Found - Installing"
# Not installed, trusting PS Gallery to remove prompt on install
Set-PSRepository -Name PSGallery -InstallationPolicy Trusted
# Installing module, requires run as admin for -scope AllUsers, change to
CurrentUser if not possible
Install-Module -Name SqlServer –Scope AllUsers -Confirm:$false -AllowClobber
}
##############################################
# Importing the SqlServer module
##############################################
Import-Module SqlServer
```

We can now specify the variables required to connect to the local or remote SQL instance/database desired. The ".\InstanceName" connects to a local SQL instance for testing and I'm using the sa account to authenticate:

```
$SQLInstance = ".\SQLEXPRESS"
$SQLDatabase = "AdventureWorks2014"
$SQLUsername = "sa"
$SQLPassword = "Password123!"
```

With the SqlServer module imported and required variables set, let's run a SELECT query to pull data into PowerShell and assign it to a variable:

```
##############################################
# Example of SELECT to pull data from a SQL table
##############################################
# Example of SQL Select query to pull data from a specific database table
$SQLQuery1 = "USE $SQLDatabase
SELECT * FROM Sales.Customer"
$SQLQuery1Output = Invoke-Sqlcmd -query $SQLQuery1 -ServerInstance
$SQLInstance -Username $SQLUsername -Password $SQLPassword
# Showing count of rows returned
$SQLQuery1Output.Count
# Selecting first 100 results
$SQLQuery1OutputTop100List = $SQLQuery1Output | select -first 100
$SQLQuery1OutputTop100List
# Selecting customer by ID
$SQLQuery1OutputCustomer = $SQLQuery1Output | Where-Object {$_.CustomerID -eq
"100"}
$SQLQuery1OutputCustomer
```

How easy was that? Let's now take this a step further by querying 2 tables and using PowerShell to combine the data:

```
##############################################
# Example of multiple SELECT queries to combine data from SQL tables
##############################################
# Example of linking data between 2 tables by first getting the customer name
from the store data
$SQLQuery2 = "USE $SQLDatabase
SELECT * FROM Sales.Store"
$SQLQuery2Output = Invoke-Sqlcmd -query $SQLQuery2 -ServerInstance
$SQLInstance -Username $SQLUsername -Password $SQLPassword
# Now I'm going to use the StoreID from the Sales.Customer to get the
Customer Name from Sales.Store by matching it to the BusinessEntityID
foreach ($CustomerRow in $SQLQuery1Output)
{
$CustomerName = $SQLQuery2Output | where-object {$_.BusinessEntityID -eq
$CustomerRow.StoreID} | select -ExpandProperty Name
write-host "ID:"$CustomerRow.CustomerID
write-host "Name:$CustomerName"
}
# But this is just writing out the info, what if you want to build a new
table array in PowerShell containing both fields? Simple! Let's create an
array this time.
# Creating the array, run once not per customer
$CustomerArray = @()
# Reducing the scope of the process to run faster for the demo
$CustomersFiltered = $SQLQuery1Output | select -first 100
# Performing for each customer operation, but this time saving the data to an
array
foreach ($CustomerRow in $CustomersFiltered)
{
# Getting customer name again
$CustomerName = $SQLQuery2Output | where-object {$_.BusinessEntityID -eq
$CustomerRow.StoreID} | select -ExpandProperty Name
# Adding customer name and existing info to customer array line
$CustomerArrayLine = new-object PSObject
$CustomerArrayLine | Add-Member -MemberType NoteProperty -Name "CustomerID" -
Value $CustomerRow.CustomerID
$CustomerArrayLine | Add-Member -MemberType NoteProperty -Name "Name" -Value
$CustomerName
$CustomerArrayLine | Add-Member -MemberType NoteProperty -Name
"AccountNumber" -Value $CustomerRow.AccountNumber
$CustomerArrayLine | Add-Member -MemberType NoteProperty -Name "LastModified"
-Value $CustomerRow.ModifiedDate
# Adding customer line to new customer array
$CustomerArray += $CustomerArrayLine
}
# Ouputting the array to show the combination of data from 2 SQL tables
```

```
$CustomerArray
# Boom. Headshot?
```

You can see I'm keeping my SQL queries very short in these examples, by pulling in all the rows, then I'm using PowerShell to filter the data. There is no reason that you can't do this in the SQL query first or both, depending on which is more efficient in your environment. I.E:

```
$SQLQuery = "USE Database
SELECT * FROM Table
WHERE Column='$Variable';"
```

That's SELECT queries covered, next up is creating a database table. I'm going to create a table ready to insert the data we created in $CustomerArray:

```
################################################
# Example of CREATE TABLE
################################################
# Create a table in SQL
$SQLQuery3 = "CREATE TABLE $SQLDatabase.Sales.PowerShellData (
    CustomerID int,
    Name varchar(255),
    AccountNumber varchar(10),
    LastModified datetime
);"
$SQLQuery3Output = Invoke-Sqlcmd -query $SQLQuery3 -ServerInstance
$SQLInstance -Username $SQLUsername -Password $SQLPassword
```

With the new table created, here is how we can INSERT the data stored in the PowerShell array into SQL:

```
################################################
# Example of INSERT INTO a table
################################################
# Inserting array into the new table we created
ForEach ($Customer in $CustomerArray)
{
# Setting the values as variables first, also converting to correct format to
match SQL DB
$CustomerID = $Customer.CustomerID -as [int]
$Name = $Customer.Name
$AccountNumber = $Customer.AccountNumber
$LastModified = $Customer.LastModified -as [datetime]
# Creating the INSERT query using the variables defined
$SQLQuery4 = "USE $SQLDatabase
```

```
INSERT INTO Sales.PowerShellData (CustomerID, Name, AccountNumber,
LastModified)
VALUES('$CustomerID','$Name','$AccountNumber','$LastModified');"
# Running the INSERT query
$SQLQuery4Output = Invoke-Sqlcmd -query $SQLQuery4 -ServerInstance
$SQLInstance -Username $SQLUsername -Password $SQLPassword
}
# Run refresh on your SQL DB and you will see the new data inserted
```

Finally, I'm going to modify a column in each row in a table with a new date/time to show you how to use UPDATE queries:

```
##############################################
# Example of UPDATE of a table
##############################################
# Setting current date time in valid SQL date/time format
$CurrentDateTime = "{0:yyyy-MM-dd HH:mm:ss}" -f (Get-Date)
# Updating each customer row
ForEach ($Customer in $CustomerArray)
{
# Setting the values as variables first, also converting to correct format to
match SQL DB
$CustomerID = $Customer.CustomerID -as [int]
# Creating the UPDATE query
$SQLQuery5 = "USE $SQLDatabase
UPDATE Sales.PowerShellData
SET LastModified='$CurrentDateTime'
WHERE CustomerID ='$CustomerID';"
# Running the INSERT query
$SQLQuery5Output = Invoke-Sqlcmd -query $SQLQuery5 -ServerInstance
$SQLInstance -Username $SQLUsername -Password $SQLPassword
}
```

And that's it! With these simple example, the world of SQL is now within the grasp of your PowerShell scripts. You can download all the examples in the zip file below:

UsingSQLWithPowerShellExamplesv1.zip

So, what will you do with this new power? Here you can see a set of examples I wrote for creating a vSphere Change Management Database:

vSphereCMDB

Even if you aren't working with anything to do with VMware, downloading the vSphereCMDB script will give you more examples to work from so I definitely

recommend it. Happy scripting,

@JoshuaStenhouse

Share this:

Twitter Facebook LinkedIn Reddit Email

Like this:

Like

One blogger likes this.

Related

Using PowerShell to report on thousands of Microsoft SQL Instances & Databases
June 14, 2018
In "Microsoft SQL"

Importing CSV files into a Microsoft SQL DB using PowerShell
June 6, 2018
In "Microsoft SQL"

Connecting to Microsoft SQL Databases using PowerShell Invoke-Sqlcmd
May 30, 2018
In "Microsoft SQL"

Published in **Microsoft SQL** and **Uncategorized**

database DBA import-module instance invoke-sqlcmd PowerShell SQL SqlServer

Creating a vSphere Change Management DB for free – Step 1 of 4 | Virtually Sober

Reply

[…] my last blog post I gave an introduction to Working with SQL Databases using PowerShell. What I'd now like to do is share an example of a cool use case for combining PowerShell, SQL and […]

Eddie

Reply

Hi Great article, thanks.

Only thing is that the “Invoke-SQLCmd” belongs to the older SQLPS module not the newer SQLServer module of PowerShell. Run: (Get-Command -Name Invoke-Sqlcmd).ModuleName;

My question: Is there a replacement equivalent to Invoke-Sqlcmd in the SqlServer module?
TIA

Joshua Stenhouse

Reply

Hey Eddie! I’m using the latest version and it’s still Invoke-Sqlcmd for me?

Get-Module -Name SqlServer

– Shows I’m on version 21.0.17240.

Get-Module -Name SqlServer | Select -ExpandProperty ExportedCommands

– Shows the commands, which includes Invoke-Sqlcmd. Finally,

Find-Module SqlServer

– Shows the latest version (as of 04/20/18) is 21.0.17240 so I’m up to date. Hope this helps you troubleshoot and please let me know if I’m missing something.

Using PowerShell & REST APIs to Monitor Rubrik Nodes v1 – Virtually Sober

Repl

[…] can easily then output these results with Export-CSV, save them to a SQL DB with examples here, or send an email using Send-MailMessage. In all 3 of these reports, you can see I have a healthy […]

Sam

Reply

It appears you can only user SQL authentication when using invoke-command, I am not seeing examples to where you can use trusted_connection with Invoke command.

Joshua Stenhouse

Reply

Hey! Did you check out this:

https://virtuallysober.com/2018/05/30/connecting-to-microsoft-sql-databases-using-powershell-invoke-sqlcmd/

Leave a Reply

Enter your comment here...