Introduction to PowerShell and SQL Server:

-Run SQL Server Management Studio

-Right click -> Start PowerShell

SQL Server PowerShell CLI pops up

Cd .\Jobs

Working with Services:

# Finding Services using Status

Get-Service | Where-Object {$\_.status –eq “Running”}

# Finding Services using name that includes sql

Get-Service | Where-Object {$\_.name –like “\*sql\*”}

# Sort Service by Status, Stopped -> Running

Get-Service | Where-Object {$\_.name –like “\*sql\*”} | Sort-Object displayname

PowerShell and the SQLConnection.NET classes:

-Run SQL Server PowerShell in Studio

# Creating a new SQL Connection variable using PS Object with .NET classes

$SqlConnection = New-Object System.Data.sqlclient.sqlconnection

# Specifying the Connection Details in Strings for SQL Connection

$SqlConnection.ConnectionString = “Server=localhost;Database=Adventureworks2012;Integrated Security=True”

# Creating a SQL Command

$SqlCmd = New-Object System.Data.SqlClient.SqlCommand

# Specifying the Command Text

$SqlCmd.CommandText = “sp\_helpdb”

# The SQL command connection is $SqlConnection defined earlier above

$SqlCmd.Connection = $SqlConnection

# Set SQL command Timeout to 0 to stay connected

$SqlCmd.CommandTimeout = 0

# Create a SQL Adapter

$SqlAdapter = New-Object System.Data.SqlClient.SqlDataAdapter

# Set the SQL Adapter command to $SqlCmd defined above

$SqlAdapter.SelectCommand = $SqlCmd

# Define $dataSet using .NET class

$dataSet = New-Object System.Data.DataSet

$SqlAdapter.Fill($dataSet)

$SqlConnection.Close()

$dataSet.Tables[0]

Loading the SMO Assemblies into PowerShell:

Sqlps.exe

# Showing all SQL server database names

$sqlServer = New-Object (“Microsoft.SqlServer.Management.Smo.Server”) “localhost”

Foreach($sqlDatabase in $sqlServer.databases) ($sqlDatabase.name)

$sqlServer | Get-Member | More

PowerShell.exe

# Load SMO first

[reflection.assembly]::LoadWithPartialName(“Microsoft.SqlServer.SMO”)

$sqlServer = New-Object (“Microsoft.SqlServer.Management.Smo.Server”) “localhost”

$sqlServer | get-member | more

Working with SQL Server using SMO:

Sqlps.exe

# To show all database names

$sqlserver = New-Object(“Microsoft.SqlServer.Management.SMO.Server”) “localhost”

Foreach($sqldatabase in $sqlserver.databases) {$sqldatabase.name}

# To show all properties & method associated with the SQL Server

$sqlserver | Get-Member | More

Working with Databases using SMO:

Sqlps.exe

# Create a PS Object for SQL Server Databases with .NET class

$Database = New-Object “Microsoft.SqlServer.Management.SMO.Database”

# To show all methods associated with SMO Database

$Database | Get-Member –Type Methods

$Database | Get-Member –Type Properties

# Set the current directory of sqlps.exe to the SQL Server path

Set-Location SQLSERVER:\SQL\localhost\Default\Databases

$Database | Get-Member –Type Methods

# Now set to a specific database, the tables

Set-Location “SQLSERVER:\SQL\Localhost\DEFAULT\Databases\Adventureworks2012\Tables”

Get-ChildItem | Where{$\_.Schema –eq “sales”}

Working with Tables using SMO:

-Management Studio

-Start PowerShell

Sqlps.exe

$s = New-Object (“Microsoft.SqlServer.Management.SMO.Server”) “localhost”

$ Create a PS Object for Database

$db = New-Object (“Microsoft.SqlServer.Management.SMO.Scripter”) ($s)

# Create a PS Object for Scripter

$scrp = New-Object (‘Microsoft.SqlServer.Management.Smo.Scripter’) ($s)

$scrp.Options.AppendToFile = $True

$scrp.Options.ClusteredIndexes = $True

$scrp.Options.DriAll = $True

$scrp.Options.ScriptDrops = $False

$scrp.Options.IncludeHeaders = $True

$scrp.Options.ToFileOnly = $True

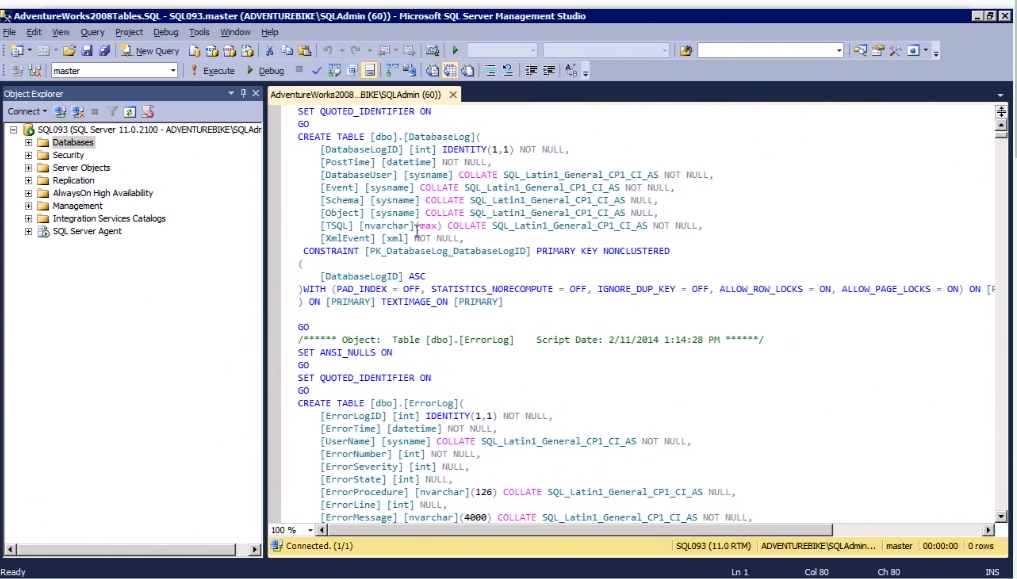
$scrp.Options.Indexes = $True

$scrp.Options.WithDependencies = $True

$scrp.Options.FileName = “C:\Temp\ADW.SQL”

ForEach($table in $db.tables) {$scrp.Script($table)}

# Open C:\Temp\AdventureWorks2008Tables.sql

# All Tables in that Database are now scripted into Objects that could be used in a testing environment

Backing up and Restoring with SMO:

Sqlps.exe

Backup SqlDatabase –ServerInstance localhost –Database AdventureWorks2012

-BackupAction Database

# Default Backup Path:

# C:\ProgramFiles\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Backup

# AdventureWork2012.bak

# Specify a backup path for .bak

$dt = Get-Date –Format yyyyMMdd

$dbname = ‘AdventureWorks2012’

Backup-SqlDatabase –ServerInstance localhost –Database $dbname

-BackupFile “C:\backups\$($dbname)\_db\_$($dt).bak”

$dbname = ‘Adventureworks2012’

Restore-SqlDatabase –ServerInstance localhost –Database $dbname

-ReplaceDatabase

Managing SQL Server Services:

PowerShell.exe as administrator

Get-Service | Where-Object {$\_.name –like “\*sql\*”}

# Changing Service StartupType

Set-Service mssqlserver –StartupType “manual”

# Stopping a Service

Net stop mssqlserver

# Starting a Service

Net start mssqlserver

Managing Client Network Protocols with WMI:

# Display the enabled protocols using SMO

$smo = ‘Microsoft.SqlServer.Management.Smo.’

$wmi = New-Object ($smo + ‘Wmi.ManagedComputer’).

[System.Reflection.Assembly]::LoadWithPartialName

(“Microsoft.SqlServer.SqlWmiManagement”) | Out-Null

$m = New-Object (“Microsoft.SqlServer.Management.Smo.Wmi.ManagedComputer”) “localhost”

$m.ClientProtocols | Select DisplayName, IsEnabled

Return protocols for each instance

$inst = $m.ServerInstances

Foreach ($i in $inst) {

[string]$nm = $i.Name

$proto = $i.ServerProtocols

Foreach ($p in $proto) {

[string]$dispnm = $p.DisplayName

[string]$enabled = $p.IsEnabled

Write-Output “Instance Name: $nm”

Write-Output “Protocol Name: $dispnm”

Write-Output “Is Enabled: $enabled”

Write-Output “”

}

}

Use WMI to disable named pipes

(Get-WmiObject –NameSpace root\Microsoft\SqlServer\ComputerManagement11

-Class ClientNetworkProtocol –Filter “ProtocolName=’np’”).SetDisable()

$smo = ‘Microsoft.SqlServer.Management.Smo.’

$vmi = New-Object ($smo + ‘Wmi.ManagedComputer’).

# Load SMO Wmi.ManagedComputer assembly

[System.Reflection.Assembly]::LoadWithPartialName (“Microsoft.SqlServer.SqlManagement”) | Out-Null

# Connect to the instance using SMO

$m = New-Object (“Microsoft.SqlServer.Management.Smo.Wmi.ManagedComputer”) “localhost”

# Return the protocols and whether or not they’re enabled

$m.ClientProtocols | Select DisplayName, IsEnabled

# Use WMI to disable named pipes

(Get-WmiObject –Namespace root\Microsoft\SqlServer\ComputerManagement11

-Class ClientNetworkProtocol –Filter “ProtocolName=’np’”).SetDisable()

# Use WMI to enable named pipes

(Get-WmiObject –Namespace root\Microsoft\SqlServer\ComputerManagement11

-Class ClientNetworkProtocol –Filter “ProtocolName=’np’”).SetEnable()

Managing SQL Server Client Aliases with WMI:

PowerShell.exe

# To read Aliases from local SQL Configuration Manager

Get-WmiObject –Namespace ‘root\Microsoft\SqlServer\ComputerManagement11’

-Class ‘SqlServerAlias’ | Format-Table –Property ‘AliasName’, ‘ServerName’, ‘ProtocolName’, ‘ConnectionString’

# To create new SQL Aliases

Sql Server Configuration Manager -> SQL Native Client 11.0 Configuration -> Aliases

New Alias -> Alias Name, Port No 1433, Protocol TCP/IP, Server localhost

Changing FileStream Settings with WMI:

Sql Server Configuration Manager -> Local -> SQL Server Service ->

SQL Server (MSSQLSERVER) -> FILESTREAM

Checked ‘Enable FILESTREAM for Transact-SQL access’

Checked ‘Enable FILESTREAM for file I/O access’

Windows share name: MSSQLSERVER

PowerShell.exe

$instance = “MSSQLSERVER”

$wmi = Get-WmiObject –Namespace “ROOT\Microsoft\SqlServer\ComputerManagement11” –Class FilestreamSettings |

Where {$\_.InstanceName –eq $instance}

# Enable FileStream

$wmi.EnableFileStream(3, $instance)

# Restart Service

Get-Service –Name $instance | Restart-Service –Force