November 18, 2020 | PowerShell SQL Server

Querying Microsoft SQL Server (MSSQL) Database with PowerShell

In this article we will discuss all effective ways to connect to a Microsoft SQL Server and run SQL queries from Powe There are many ways how you can work with SQL Server using PowerShell, and it is easy to get confused when you lots of articles in the Web, since all of them describe different methods, and even an experienced administrator may questions.

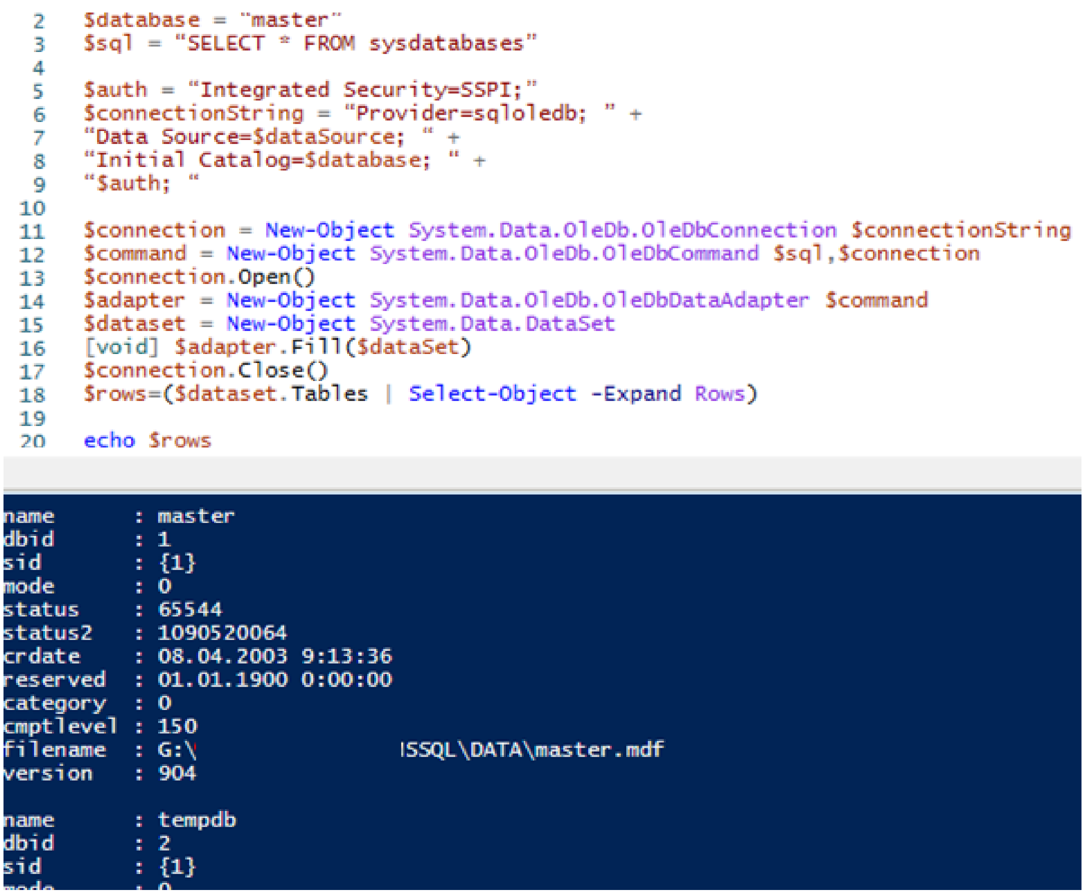
Contents:

T-SQL Queries in PowerShell Using System.Data.OleDb

Since PowerShell can access .NET Framework classes, you can use classes from **System.Data.OleDb** to execute T- queries.

Here is a sample PowerShell script to connect SQL Server using System.Data.OleDb class. Let's run a SELECT query table in the MS SQL database:

```
 $dataSource = “lon-sql01\testdb”
$database = “master”
$sql = “SELECT * FROM sysdatabases”
$auth = “Integrated Security=SSPI;”
$connectionString = “Provider=sqloledb; ” +
“Data Source=$dataSource; “ +
“Initial Catalog=$database; “ +
“$auth; “
$connection = New-Object System.Data.OleDb.OleDbConnection $connectionString
$command = New-Object System.Data.OleDb.OleDbCommand $sql,$connection
$connection.Open()
$adapter = New-Object System.Data.OleDb.OleDbDataAdapter $command
$dataset = New-Object System.Data.DataSet
[void] $adapter.Fill($dataSet)
$connection.Close()
$rows=($dataset.Tables | Select-Object -Expand Rows)
echo $rows
```



```
$database = "master"
$sql = "SELECT * FROM sysdatabases"

$auth = "Integrated Security=SSPI;"
$connectionString = "Provider=sqloledb; " +
"Data Source=$dataSource; " +
"Initial Catalog=$database; " +
"$auth; "

$connection = New-Object System.Data.OleDb.OleDbConnection $connectionString
$command = New-Object System.Data.OleDb.OleDbCommand $sql,$connection
$connection.Open()
$adapter = New-Object System.Data.OleDb.OleDbDataAdapter $command
$dataset = New-Object System.Data.DataSet
[void] $adapter.Fill($dataSet)
$connection.Close()
$rows=($dataset.Tables | Select-Object -Expand Rows)

echo $rows
```

```
name      : master
dbid      : 1
sid       : {1}
mode      : 0
status    : 65544
status2   : 1090520064
crdate    : 08.04.2003 9:13:36
reserved  : 01.01.1900 0:00:00
category  : 0
cmptlevel : 150
filename  : G:\              ISSQL\DATA\master.mdf
version   : 904

name      : tempdb
dbid      : 2
sid       : {1}
```

Here is an example of a PowerShell script to execute an INSERT/UPDATE/DELETE query against MSSQL database:

```
 $dataSource = “lon-sql01\testdb”
$database = “test”
$sql = "insert into test_table (test_col) Values ('Test')"
$auth = “Integrated Security=SSPI;”
$connectionString = “Provider=sqloledb; ” +
“Data Source=$dataSource; ” +
“Initial Catalog=$database; ” +
“$auth; “
$connection = New-Object System.Data.OleDb.OleDbConnection $connectionString
$command = New-Object System.Data.OleDb.OleDbCommand $sql,$connection
$connection.Open()
$command = New-Object data.OleDb.OleDbCommand $sql
$command.connection = $connection
$rowsAffected = $command.ExecuteNonQuery()
```

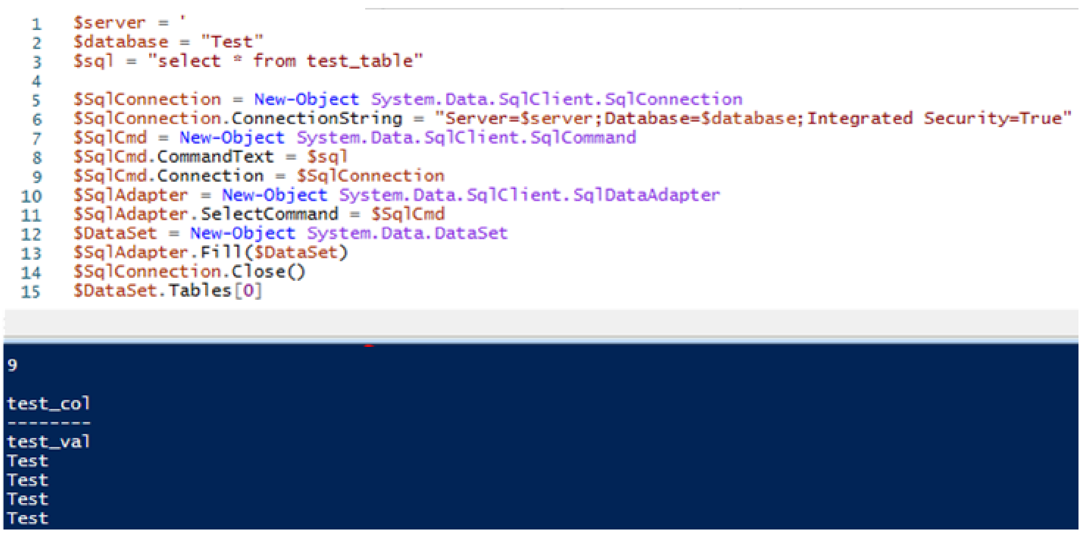
The `$rowsAffected` variable contains the number of added or changed rows. To run an update or delete query, just the line of the SQL query in the `$sql` variable.

Running SQL Query in PowerShell Using System.Data.SqlClient Class

To access MS SQL Server from PowerShell, you can use another built-in .NET class – **System.Data.SqlClient**. Here example of a SELECT query in a PowerShell script with SqlClient:

```
 $server = "lon-sql01\testdb"
$database = "Test"
$sql = "select * from test_table"
$SqlConnection = New-Object System.Data.SqlClient.SqlConnection
$SqlConnection.ConnectionString = "Server=$server;Database=$database;Integrated Security=True"
$SqlCmd = New-Object System.Data.SqlClient.SqlCommand
$SqlCmd.CommandText = $sql
```

```
$SqlCmd.Connection = $SqlConnection

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

$SqlAdapter.SelectCommand = $SqlCmd

$DataSet = New-Object System.Data.DataSet

$SqlAdapter.Fill($DataSet)

$SqlConnection.Close()

$DataSet.Tables[0]
```



```
$server = '
$database = "Test"
$sql = "select * from test_table"

$SqlConnection = New-Object System.Data.SqlClient.SqlConnection
$SqlConnection.ConnectionString = "Server=$server;Database=$database;Integrated Security=True"
$SqlCmd = New-Object System.Data.SqlClient.SqlCommand
$SqlCmd.CommandText = $sql
$SqlCmd.Connection = $SqlConnection
$SqlAdapter = New-Object System.Data.SqlClient.SqlDataAdapter
$SqlAdapter.SelectCommand = $SqlCmd
$DataSet = New-Object System.Data.DataSet
$SqlAdapter.Fill($DataSet)
$SqlConnection.Close()
$DataSet.Tables[0]
```

```
9

test_col
--------
test_val
Test
Test
Test
Test
```

An example of an INSERT/DELETE/UPDATE query:

```
 $server = "lon-sql01\testdb"

$database = "Test"

$sql = "insert into test_table (test_col) Values ('Test')"

$SqlConnection = New-Object System.Data.SqlClient.SqlConnection

$SqlConnection.ConnectionString = "Server=$server;Database=$database;Integrated Security=True"

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

$SqlCmd.CommandText = $sql

$SqlCmd.Connection = $SqlConnection

$SqlConnection.Open()

$rowsAffected = $SqlCmd.ExecuteNonQuery();

$SqlConnection.Close()
```

Note. The code containing SqlClient classes is very much like the code with OleDB. These classes work in a sim way**:**

1. An MSSQL server connection object is created;

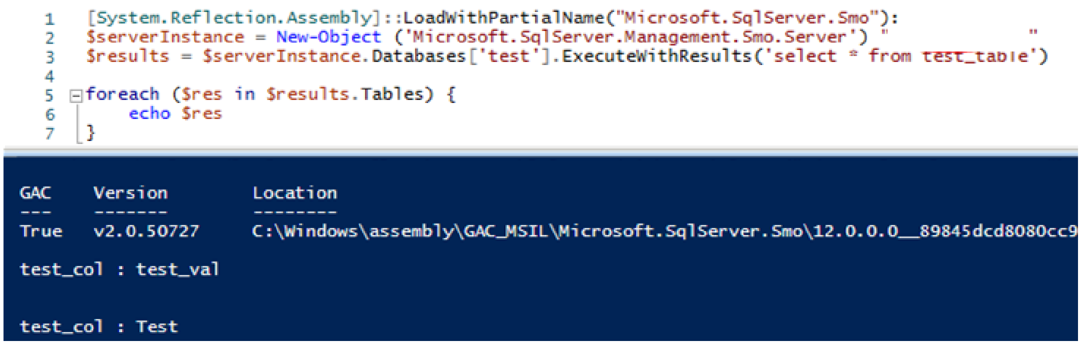
2. An object with an SQL query is created, and the connection object is assigned to it;
3. Then in case of running a SELECT query, an adapter object is created and the query is executed e context of this object;
4. In case of running an INSERT/UPDATE/DELETE query, the object with the query (containing the conne object) executes the `ExecuteNonQuery()` method.

SQL Query in PowerShell Using SQL Server Management Studio Module

To use **Microsoft.SqlServer.Smo (SMO)** classes, **SQL Server Management Studio** must be installed on your c

Load the SMO module, create a new server object and then run a SELECT query:

```
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SqlServer.Smo");
$serverInstance = New-Object ('Microsoft.SqlServer.Management.Smo.Server') "lon-sql01\testdb"
$results = $serverInstance.Databases['test'].ExecuteWithResults('select * from test_table')
foreach ($res in $results.Tables) {
$nbsp;echo $res
}
```



```
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SqlServer.Smo"):
$serverInstance = New-Object ('Microsoft.SqlServer.Management.Smo.Server') "              "
$results = $serverInstance.Databases['test'].ExecuteWithResults('select * from test_table')

foreach ($res in $results.Tables) {
    echo $res
}
```

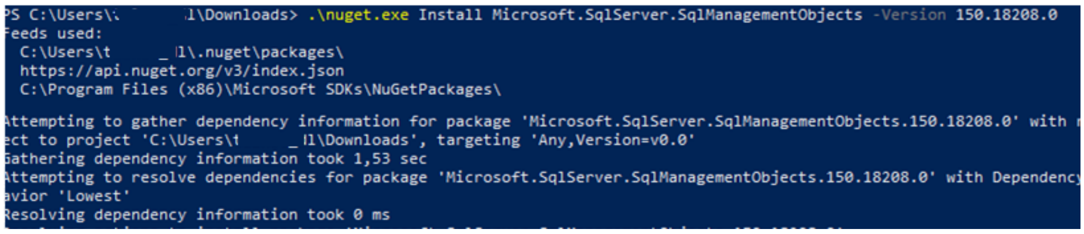
```
GAC    Version        Location
---    -------        --------
True   v2.0.50727     C:\Windows\assembly\GAC_MSIL\Microsoft.SqlServer.Smo\12.0.0.0__89845dcd8080cc9

test_col : test_val


test_col : Test
```

For an insert/update/delete query, run `ExecuteNonQuery` :

```
$db = $serverInstance.Databases['test']
$db.ExecuteNonQuery("insert into test_table (test_col) Values ('123456')")
```

Note. You can also install SMO libraries through the NuGet Package Manager**:**

1. Download **nuget.exe** https://www.nuget.org/downloads;
2. Run PowerShell as an administrator and go to the directory containing nuget.exe;
3. Run: `.\nuget.exe Install Microsoft.SqlServer.SqlManagementObjects` .



```
PS C:\Users\[redacted]l\Downloads> .\nuget.exe Install Microsoft.SqlServer.SqlManagementObjects -Version 150.18208.0
Feeds used:
  C:\Users\t[redacted]l\.nuget\packages\
  https://api.nuget.org/v3/index.json
  C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\

Attempting to gather dependency information for package 'Microsoft.SqlServer.SqlManagementObjects.150.18208.0' with r
ect to project 'C:\Users\t[redacted]l\Downloads', targeting 'Any,Version=v0.0'
Gathering dependency information took 1,53 sec
Attempting to resolve dependencies for package 'Microsoft.SqlServer.SqlManagementObjects.150.18208.0' with Dependency
avior 'Lowest'
Resolving dependency information took 0 ms
```

4. Microsoft.SqlServer.SqlManagementObjects folder with all DLLs will appear in the current directory;
5. Load the SMO library into your PowerShell session from a DLL file. Add it to your script:

```
add-type –Path
"C:\Users\username\Downloads\Microsoft.SqlServer.SqlManagementObjects.150.18208.0\lib\net45\Microsoft.Sq
```

Then SMO classes will become available for use.

Invoke-Sqlcmd Cmdlet from SQLServer PowerShell Module

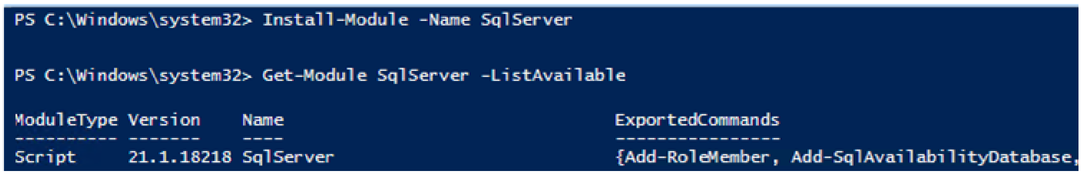
To use the **Invoke-Sqlcmd** cmdlet, install the **SqlServer for PowerShell module**. Run PowerShell with the adm privileges and execute the command:

```
Install-Module -Name SqlServer
```

(Press Y and then ENTER to accept the installer notifications.)

After the installation, you can make sure that the module has been installed correctly by running this command:

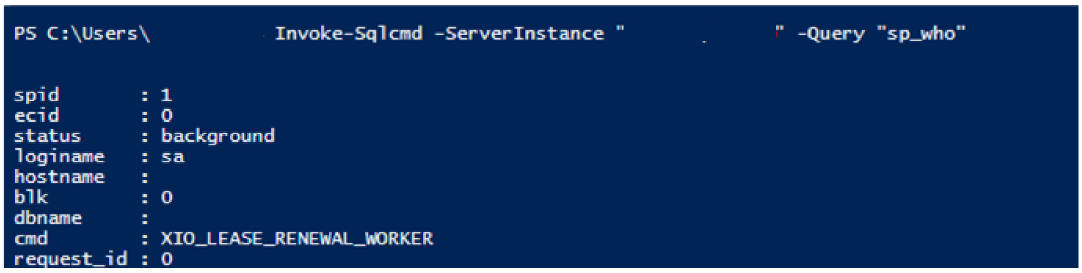
```
Get-Module SqlServer -ListAvailable
```


```
PS C:\Windows\system32> Install-Module -Name SqlServer

PS C:\Windows\system32> Get-Module SqlServer -ListAvailable

ModuleType Version    Name                             ExportedCommands
---------- -------    ----                             ----------------
Script     21.1.18218 SqlServer                        {Add-RoleMember, Add-SqlAvailabilityDatabase,
```

The **Invoke-Sqlcmd** cmdlet is easier and more intuitive than other ways of connection to an Microsoft SQL Server PowerShell. Invoke-Sqlcmd uses the same syntax for SELECT and INSERT/UPDATE/DELETE queries.

Here is an example of a Select query:

```
Invoke-Sqlcmd -ServerInstance "lon-sql01\testdb" -Query "sp_who"
```


```
PS C:\Users\          Invoke-Sqlcmd -ServerInstance "           " -Query "sp_who"

spid       : 1
ecid       : 0
status     : background
loginame   : sa
hostname   :
blk        : 0
dbname     :
cmd        : XIO_LEASE_RENEWAL_WORKER
request_id : 0
```

This is an example of an INSERT query:

```
Invoke-Sqlcmd -ServerInstance "lon-sql01\testdb" -Database "test1" -Query "insert into test_table (test_col)
('123321')"
```

Unlike other methods, a query in the `Invoke-Sqlcmd` is always set in the `-Query` parameter.

Which SQL connection option should you use?

A choice between oledb/smo/sqlclient/invoke-sqlcmd is based on the task and the environment where you are going PowerShell script.

If you want to deploy a script to multiple servers (for example, your script collects monitoring data locally), using SM SqlServer PowerShell module (Invoke-SQLcmd) is not reasonable, since you will have to install extra packages on th hosts to run the script, and it is better to avoid it if there are a lot of servers.

In its turn, the SqlServer for PowerShell module offers many other cmdlets to work with your SQL Server (you can le here: https://docs.microsoft.com/en-us/powershell/module/sqlserver). The module contains more commands to mai Server itself.

If your script will perform non-administrative tasks (is responsible for some part of business logic, for example), it is use System.Data.SqlClient/SMO, as they provide more convenient development tools. An advantage of OleDB is that work not only with an SQL Server, but also with Access/Oracle/Firebird/Interbase.

0 comment 2

previous post
How to Remove Hidden/Ghost Network Adapters in Windows?

ne
FAQ: Live Migration of Virtual Machines with V
v

RELATED READING

How to Find Inactive Computers and Users in...
January 29, 2021

Checking User Logon History in Active Directory Domain...
January 22, 2021

MS SQL Server 2019 Installation Gu Settings...
January 19, 2021

LEAVE A COMMENT

t

Get-

Get-

How

Man

How

PSW

BACK TO TOP