Use the sqlcmd Utility

SQL Server 2012 7 out of 12 rated this helpful

The **sqlcmd** utility is a command-line utility for ad hoc, interactive execution of Transact-SQL statements and scripts and for automating Transact-SQL scripting tasks. To use **sqlcmd** interactively, or to build script files to be run using **sqlcmd**, users must understand Transact-SQL. The **sqlcmd** utility is typically used in the following ways:

- Users interactively enter Transact-SQL statements in a manner similar to working at the command prompt. The results are displayed at the command prompt. To open a Command Prompt window, click **Start**, click **All Programs**, point to **Accessories**, and then click **Command Prompt**. At the command prompt, type **sqlcmd** followed by a list of options that you want. For a complete list of the options that are supported by **sqlcmd**, see sqlcmd Utility.
- Users submit a **sqlcmd** job either by specifying a single Transact-SQL statement to execute, or by pointing the utility to a text file that contains Transact-SQL statements to execute. The output is usually directed to a text file, but it can also be displayed at the command prompt.
- SQLCMD mode in SQL Server Management Studio Query Editor.
- SQL Server Management Objects (SMO)
- SQL Server Agent CmdExec jobs.

Typically Used sqlcmd Options

The following options are used most frequently:

- The server option (**-S**) that identifies the instance of Microsoft SQL Server to which **sqlcmd** connects.
- Authentication options (**-E**, **-U**, and **-P**) that specify the credentials that **sqlcmd** uses to connect to the instance of SQL Server.

  Note

  The **-E** option is the default and does not have to be specified.
- Input options (**-Q**, **-q**, and **-i**) that identify the location of the input to **sqlcmd**.
- The output option (**-o**) that specifies the file in which **sqlcmd** is to put its output.

Connecting to the sqlcmd Utility

The following are common uses of the **sqlcmd** utility:

- Connecting to a default instance by using Windows Authentication to interactively run Transact-SQL statements:

```
sqlcmd -S <ComputerName>
```

Note

In the previous example, **-E** is not specified because it is the default and **sqlcmd** connects to the default instance by using Windows Authentication.

- Connecting to a named instance by using Windows Authentication to interactively run Transact-SQL statements:

```
sqlcmd -S <ComputerName>\<InstanceName>
```

or

```
sqlcmd -S .\<InstanceName>
```

- Connecting to a named instance by using Windows Authentication and specifying input and output files:

```
sqlcmd -S <ComputerName>\<InstanceName> -i <MyScript.sql> -o <MyOutput.rpt>
```

- Connecting to the default instance on the local computer by using Windows Authentication, executing a query, and having **sqlcmd** remain running after the query has finished running:

```
sqlcmd -q "SELECT * FROM AdventureWorks2012.Person.Person"
```

- Connecting to the default instance on the local computer by using Windows Authentication, executing a query, directing the output to a file, and having **sqlcmd** exit after the query has finished running:

```
sqlcmd -Q "SELECT * FROM AdventureWorks2012.Person.Person" -o MyOutput.txt
```

- Connecting to a named instance using SQL Server Authentication to interactively run Transact-SQL statements, with **sqlcmd** prompting for a password:

```
sqlcmd -U MyLogin -S <ComputerName>\<InstanceName>
```

Note

To see a list of the options that are supported by the **sqlcmd** utility run: `sqlcmd -?`.

Running Transact-SQL Statements Interactively by Using sqlcmd

You can use the **sqlcmd** utility interactively to execute Transact-SQL statements in a Command Prompt window. To interactively execute Transact-SQL statements by using **sqlcmd**, run the utility without using the **-Q**, **-q**, **-Z**, or **-i** options to specify any input files or queries. For example:

```
sqlcmd -S <ComputerName>\<InstanceName>
```

When the command is executed without input files or queries, **sqlcmd** connects to the specified instance of SQL Server and then displays a new line with a `1>` followed by a blinking underscore that is named the **sqlcmd** prompt. The `1` signifies that this is the first line of a Transact-SQL statement, and the **sqlcmd** prompt is the point at which the Transact-SQL statement will start when you type it in.

At the **sqlcmd** prompt, you can type both Transact-SQL statements and **sqlcmd** commands, such as **GO** and **EXIT**. Each Transact-SQL statement is put in a buffer called the statement cache. These statements are sent to SQL Server after you type the **GO** command and press ENTER. To exit **sqlcmd**, type **EXIT** or **QUIT** at the start of a new line.

To clear the statement cache, type **:RESET**. Typing **^C** causes **sqlcmd** to exit. **^C** can also be used to stop the execution of the statement cache after a **GO** command has been issued.

Transact-SQL statements that are entered in an interactive session can edited by entering the **:ED** command and the **sqlcmd** prompt. The editor will open and, after editing the Transact-SQL statement and closing the editor, the revised Transact-SQL statement will appear in the command window. Enter **GO** to run the revised Transact-SQL statement.

Quoted Strings

Characters that are enclosed in quotation marks are used without any additional preprocessing, except that quotations marks can be inserted into a string by entering two consecutive quotation marks. SQL Server treats this character sequence as one quotation mark. (However, the translation occurs in the server.) Scripting variables will not be expanded when they appear within a string.

For example:

```
sqlcmd

PRINT "Length: 5"" 7'";

GO
```

Here is the result set.

```
Length: 5" 7'
```

Strings That Span Multiple Lines

sqlcmd supports scripts that have strings that span multiple lines. For example, the following `SELECT` statement spans multiple lines but is a single string executed when you press the ENTER key after typing `GO`.

```
SELECT First line

FROM Second line

WHERE Third line;

GO
```

Interactive sqlcmd Example

This is an example of what you see when you run **sqlcmd** interactively.

When you open a Command Prompt window, there is one line similar to:

```
C:\> _
```

This means the folder `C:\` is the current folder, and if you specify a file name, Windows will look for the file in that folder.

Type **sqlcmd** to connect to the default instance of SQL Server on the local computer, and the contents of the Command Prompt window will be:

```
C:\>sqlcmd

1> _
```

This means you have connected to the instance of SQL Server and `sqlcmd` is now ready to accept Transact-SQL statements and `sqlcmd` commands. The flashing underscore after the `1>` is the `sqlcmd` prompt that marks the location at which the statements and commands you type will be displayed. Now, type **USE AdventureWorks2012** and press ENTER, and then type **GO** and press ENTER. The contents of the Command Prompt window will be:

```
sqlcmd

USE AdventureWorks2012;

GO
```

Here is the result set.

```
Changed database context to 'AdventureWorks2012'.
```

```
1> _
```

Pressing ENTER after entering `USE AdventureWorks2012` signaled `sqlcmd` to start a new line. Pressing ENTER, after you type `GO,` signaled `sqlcmd` to send the `USE AdventureWorks2012` statement to the instance of SQL Server. `sqlcmd` then returned a message to indicate that the `USE` statement completed successfully and displayed a new `1>` prompt as a signal to enter a new statement or command.

The following example shows what the Command Prompt window contains if you type a `SELECT` statement, a `GO` to execute the `SELECT`, and an `EXIT` to exit `sqlcmd`:

```
sqlcmd

USE AdventureWorks2012;

GO

SELECT TOP (3) BusinessEntityID, FirstName, LastName

FROM Person.Person;

GO
```

Here is the result set.

```
BusinessEntityID FirstName LastName

----------- -------------------------------- -----------

1 Syed Abbas

2 Catherine Abel

3 Kim Abercrombie

(3 rows affected)

1> EXIT

C:\>
```

The lines after line `3> GO` are the output of a `SELECT` statement. After you generate output, `sqlcmd` resets the `sqlcmd` prompt and displays `1>`. After entering `EXIT` at line `1>`, the Command Prompt window displays the same line it did when you first opened it. This indicates that `sqlcmd` has exited its session. You can now close the Command Prompt window by typing another `EXIT` command.

Running Transact-SQL Script Files by Using sqlcmd

You can use **sqlcmd** to execute database script files. Script files are text files that contain a mix of Transact-SQL statements, **sqlcmd** commands, and scripting variables. For more information about how to script variables, see Use sqlcmd with Scripting Variables. **sqlcmd** works with the statements, commands, and scripting variables in a script file in a manner similar to how it works with statements and commands that are entered interactively. The main difference is that **sqlcmd** reads through the input file without pause instead of waiting for a user to enter the statements, commands, and scripting variables.

There are different ways to create database script files:

- You can interactively build and debug a set of Transact-SQL statements in SQL Server Management Studio, and then save the contents of the Query window as a script file.
- You can create a text file that contains Transact-SQL statements by using a text editor, such as Notepad.

Examples

A. Running a script by using sqlcmd

Start Notepad, and type the following Transact-SQL statements:

```
USE AdventureWorks2012;

GO

SELECT TOP (3) BusinessEntityID, FirstName, LastName

FROM Person.Person;

GO
```

Create a folder named `MyFolder` and then save the script as the file `MyScript.sql` in the folder `C:\MyFolder`. Enter the following at the command prompt to run the script and put the output in `MyOutput.txt` in `MyFolder`:

```
sqlcmd -i C:\MyFolder\MyScript.sql -o C:\MyFolder\MyOutput.txt
```

When you view the contents of `MyOutput.txt` in Notepad, you will see the following:

```
Changed database context to 'AdventureWorks2012'.

BusinessEntityID FirstName LastName

---------------- ----------- -----------

1 Syed Abbas

2 Catherine Abel

3 Kim Abercrombie


(3 rows affected)
```

B. Using sqlcmd with a dedicated administrative connection

In the following example, `sqlcmd` is used to connect to a server that has a blocking problem by using the dedicated administrator connection (DAC).

```
C:\>sqlcmd -S ServerName -A

1> SELECT blocked FROM sys.dm_exec_requests WHERE blocked <> 0;

2> GO
```

Here is the result set.

```
spid blocked

------ -------

62 64

(1 rows affected)
```

Use `sqlcmd` to end the blocking process.

```
1> KILL 64;

2> GO
```

C. Using sqlcmd to execute a stored procedure

The following example shows how to execute a stored procedure by using `sqlcmd`. Create the following stored procedure.

```
USE AdventureWorks2012;

IF OBJECT_ID ( ' dbo.ContactEmailAddress, 'P' ) IS NOT NULL

DROP PROCEDURE dbo.ContactEmailAddress;

GO

CREATE PROCEDURE dbo.ContactEmailAddress

(

@FirstName nvarchar(50)

,@LastName nvarchar(50)

)

AS

SET NOCOUNT ON

SELECT EmailAddress

FROM Person.Person

WHERE FirstName = @FirstName

AND LastName = @LastName;
```

```
SET NOCOUNT OFF
```

At the `sqlcmd` prompt, enter the following:

```
C:\sqlcmd

1> :Setvar FirstName Gustavo

1> :Setvar LastName Achong

1> EXEC dbo.ContactEmailAddress $(Gustavo),$(Achong)

2> GO

EmailAddress

-----------------------------

gustavo0@adventure-works.com
```

D. Using sqlcmd for database maintenance

The following example shows how to use `sqlcmd` for a database maintenance task. Create `C:\BackupTemplate.sql` with the following code.

```
USE master;

BACKUP DATABASE [$(db)] TO DISK='$(bakfile)';
```

At the `sqlcmd` prompt, enter the following:

```
C:\ >sqlcmd

1> :connect <server>

Sqlcmd: Successfully connected to server <server>.

1> :setvar db msdb

1> :setvar bakfile c:\msdb.bak

1> :r c:\BackupTemplate.sql

2> GO

Changed database context to 'master'.

Processed 688 pages for database 'msdb', file 'MSDBData' on file 2.

Processed 5 pages for database 'msdb', file 'MSDBLog' on file 2.

BACKUP DATABASE successfully processed 693 pages in 0.725 seconds (7.830 MB/sec)
```

E. Using sqlcmd to execute code on multiple instances

The following code in a file shows a script that connects to two instances. Notice the `GO` before the connection to the second instance.

```
:CONNECT <server>\,<instance1>

EXEC dbo.SomeProcedure

GO

:CONNECT <server>\,<instance2>

EXEC dbo.SomeProcedure

GO
```

E. Returning XML output

The following example shows how XML output is returned unformatted, in a continuous stream.

```
C:\>sqlcmd -d AdventureWorks2012

1> :XML ON

1> SELECT TOP 3 FirstName + ' ' + LastName + ', '

2> FROM Person.Person

3> GO

Syed Abbas, Catherine Abel, Kim Abercrombie,
```

F. Using sqlcmd in a Windows script file

A **sqlcmd**command such as `sqlcmd -i C:\InputFile.txt -o C:\OutputFile.txt,` can be executed in a .bat file together with VBScript. In this case, do not use interactive options. **sqlcmd** must be installed on the computer that is executing the .bat file.

First, create the following four files:

- C:\badscript.sql

```
SELECT batch_1_this_is_an_error
GO
SELECT 'batch #2'
GO
```

- C:\goodscript.sql

```
SELECT 'batch #1'
GO
SELECT 'batch #2'
GO
```

- C:\returnvalue.sql

```
:exit(select 100)
@echo off
C:\windowsscript.bat
@echo off

echo Running badscript.sql
sqlcmd -i badscript.sql -b -o out.log
if not errorlevel 1 goto next1
echo == An error occurred

:next1

echo Running goodscript.sql
sqlcmd -i goodscript.sql -b -o out.log
if not errorlevel 1 goto next2
echo == An error occurred

:next2
echo Running returnvalue.sql
sqlcmd -i returnvalue.sql -o out.log
echo SQLCMD returned %errorlevel% to the command shell

:exit
```

- C:\windowsscript.bat

```
@echo off

echo Running badscript.sql
sqlcmd -i badscript.sql -b -o out.log
if not errorlevel 1 goto next1
echo == An error occurred

:next1

echo Running goodscript.sql
sqlcmd -i goodscript.sql -b -o out.log
if not errorlevel 1 goto next2
echo == An error occurred

:next2
echo Running returnvalue.sql
sqlcmd -i returnvalue.sql -o out.log
echo SQLCMD returned %errorlevel% to the command shell

:exit
```

Then, at the command prompt, run `C:\windowsscript.bat`:

```
C:\>windowsscript.bat

Running badscript.sql

== An error occurred

Running goodscript.sql

Running returnvalue.sql

SQLCMD returned 100 to the command shell
```

G. Using sqlcmd to set encryption on Windows Azure SQL Database

A **sqlcmd**can be executed on a connection to SQL Database data on to specify encryption and certificate trust. Two **sqlcmd** options are available:

- The -N switch is used by the client to request an encrypted connection. This option is equivalent to the ADO.net option `ENCRYPT = true`.
- The –C switch is used by the client to configure it to implicitly the trust server certificate and not validate it. This option is equivalent to the ADO.net option `TRUSTSERVERCERTIFICATE = true`.

The SQL Database service does not support all the `SET` options available on a SQL Server instance. The following options throw an error when the corresponding `SET` option is set to `ON` or `OFF`:

- SET ANSI_DEFAULTS
- SET ANSI_NULLS
- SET REMOTE_PROC_TRANSACTIONS
- SET ANSI_NULL_DEFAULT

The following SET options do not throw exceptions but cannot be used. They are deprecated:

- SET CONCAT_NULL_YIELDS_NULL
- SET ANSI_PADDING
- SET QUERY_GOVERNOR_COST_LIMIT

Syntax

The following examples refer to cases where SQL Server Native Client Provider settings include: `ForceProtocolEncryption = False`, `Trust Server Certificate = No`

Connect using Windows credentials and encrypt communication:

```
SQLCMD –E –N
```

Connect using Windows credentials and trust server certificate:

```
SQLCMD -E -C
```

Connect using Windows credentials, encrypt communication and trust server certificate:

```
SQLCMD -E -N -C
```

The following examples refer to cases where SQL Server Native Client Provider settings include: `ForceProtocolEncryption = True`, `TrustServerCertificate = Yes`.

Connect using Windows credentials, encrypt communication and trust server certificate:

```
SQLCMD -E
```

Connect using Windows credentials, encrypt communication and trust server certificate:

```
SQLCMD -E -N
```

Connect using Windows credentials, encrypt communication and trust server certificate:

```
SQLCMD -E -T
```

Connect using Windows credentials, encrypt communication and trust server certificate:

```
SQLCMD -E -N -C
```

If the provider specifies `ForceProtocolEncryption = True` then encryption is enabled even if `Encrypt=No` in the connection string.

See Also

Reference

sqlcmd Utility

Concepts

Use sqlcmd with Scripting Variables

Edit SQLCMD Scripts with Query Editor

Manage Job Steps
Create a CmdExec Job Step

Did you find this helpful? ○ Yes ○ No

Community Additions

SQLS*Plus - SQL*Plus command line too for SQL Server

There is a free tool "SQLS*Plus" (on http://www.memfix.com) which is like SQL*Plus for SQL Server. Works with SQL Server 2000, 2005, 2008 and 2012

Very flexible with data formatting (set lines size, pagesize, etc), variables (var, &, &&), spool, HTML output, etc - has lots of added functionality to isql, osql or sqlcmd

I downloaded SQLS*Plus from http://www.memfix.com

svsqldba
2/1/2014

Calling Stored Procedure example is wrong

The example of calling a stored procedure is incorrect.
Thus the following :

```
C:\sqlcmd

1> :Setvar FirstName Gustavo

1> :Setvar LastName Achong

1> EXEC dbo.ContactEmailAddress $(Gustavo),$(Achong)
```

Should be:

```
C:\sqlcmd

1> :Setvar FirstName Gustavo

1> :Setvar LastName Achong

1> EXEC dbo.ContactEmailAddress $(FirstName ),$(LastName)
```

fishwater
9/13/2012