PowerShell

При использовании системы SQL Server 2012, куда по умолчанию устанавливается оболочка PowerShell 2.0, вам следует импортировать модуль sqlps, который содержит все команды PowerShell, имеющие отношение к SQL Server. Однако перед тем как выполнять процедуру импортирования, необходимо настроить политику выполнения таким образом, чтобы она допускала возможность выполнения сценариев. Как минимум, запустите следующую команду в сеансе работы с более высоким уровнем разрешений:

PS C:> set-executionpolicy remotesigned -force

Эту политику необходимо определять только один раз. Отметим, что при вводе указанной команды не следует вводить символы PS C:>, поскольку они не являются частью команды. Это всего лишь приглашение на ввод команд, указывающее текущий каталог, в котором вам следует работать. При изучении технологии PowerShell полезно включать данную информацию, и именно по этой причине я обозначаю ее здесь, а также в последующих командах.

Далее импортируйте модуль sqlps с помощью следующей команды:

PS C:> import-module sqlps

Вполне возможно, что в процессе импортирования модуля вы получите предупреждение о недопустимых глаголах. Все команды PowerShell, которые называются составными командами (cmdlet), должны соответствовать стандарту именования «глагол — существительное». При этом применяемые глаголы должны входить в список стандартных глаголов. NET, таких, как Get, Set и Remove. Соблюдение стандарта именования облегчает поиск команд. В модуль sqlps входит несколько команд, отступающих от этого стандарта. Иначе говоря, поиск одной из импортированных команд SQL, возможно, будет несколько сложнее.

Чтобы просмотреть входящие в состав модуля упомянутые составные команды, введите в командной строке следующую команду:

PS C:> get-command -module sqlps

На экране появится список, подобный приведенному на экране 1.

|  |
| --- |
| Получение списка доступных составных команд в?модуле sqlps |
| **Экран 1. Получение списка доступных составных команд в?модуле sqlps** |

Отметим, что при использовании PowerShell 3.0 для запуска команды, входящей в состав модуля, например команды Restore-SQLDatabase, необязательно предварительно импортировать модуль. А в версии PowerShell 2.0 для использования любой из команд модуля sqlps этот модуль необходимо сначала импортировать, что я и сделал несколько секунд назад.

Насколько можно судить по именам составных команд на экране 1, если соглашение о стандартном именовании «глагол-существительное» соблюдено, пользователь может без особого труда понять, какую задачу выполняет та или иная команда. Но если у вас есть сомнения, можете обратиться за помощью к программе PowerShell. Допустим, вы хотите узнать, какую операцию выполняет команда Invoke-SQLCmd. Введите следующую команду:

PS C:> help invoke-sqlcmd

Как показано на экране 2, данная команда возвращает сведения о том, какое действие выполняет составная команда Invoke-SQLCmd и как ею пользоваться.

|  |
| --- |
| Получение справочной информации о составной команде Invoke-SQLCmd |
| **Экран 2. Получение справочной информации о составной команде Invoke-SQLCmd** |

Если вам требуется более подробная информация, можете добавить параметр -full следующим образом:

PS C:> help invoke-sqlcmd -full

Эта команда возвратит все запрошенные сведения, включая подробности, касающиеся параметра, и примеры использования. Некоторые составные команды содержат ссылки на файл оперативной подсказки, в котором будет содержаться самая свежая информация. Если оперативная ссылка предусмотрена, вы можете запустить команду наподобие следующей:

PS C:> Help get-eventlog -online

А на ряде сайтов в Интернете имеется даже содержимое, размещенное членами сообщества пользователей.

**Использование провайдера SQL Server**

Наряду с составными командами PowerShell позволяет задействовать программный компонент, именуемый провайдером. Как правило, провайдер обеспечивает интерфейс управления используемой технологией. Например, оболочка PowerShell без предварительной настройки содержит провайдеры для взаимодействия с файловой системой, реестром и хранилищем сертификатов. В состав некоторых модулей и оснасток PowerShell входят дополнительные провайдеры, такие, как Active Directory. Просмотреть перечень установленных провайдеров можно с помощью команды

PS C:> get-psprovider

Модуль sqlps включает в себя провайдер SQLServer. Получить сведения о провайдере SQLServer вы можете с помощью следующей команды:

PS C:> help sqlserver

Результаты ее выполнения представлены на экране 3.

|  |
| --- |
| Получение справочной информации о провайдере SQL Server |
| **Экран 3. Получение справочной информации о провайдере SQL Server** |

Я рекомендую вам запустить эту команду и посвятить несколько минут изучению сведений о провайдере SQLServer. Главная мысль, содержащаяся в этих данных, состоит в том, что с помощью упомянутого провайдера пользователь может перемещаться по экземпляру SQL Server, как если бы это была файловая система, и извлекать информацию или вносить изменения, используя методы, которые применяются при работе с объектами PowerShell. Так, составная команда Set-Location используется для изменения текущего каталога, а команда Get-ChildItem — для формирования перечня файлов и каталогов, содержащихся в текущем каталоге. Тот, кто знаком с командами командной оболочки Windows (cmd.exe), наверняка будет охотно использовать соответствующие псевдонимы данной составной команды — cd и dir. Если, скажем, у вас возникнет потребность перейти на накопитель SQL Server и сформировать опись его содержимого, потребуется выполнить такие команды:

PS C:> cd sqlserver:

PS SQLSERVER:> dir

Подобным же образом вы можете обратиться к своему экземпляру SQL Server и пройти по его содержимому. Например, если вашему экземпляру SQL Server присвоено имя chi-db01, нужно запустить команды:

PS C:> cd sqlchi-db01default

PS SQLSERVER:sqlchi-db01default> dir

На экране 4 представлены результаты выполнения обоих наборов команд.

|  |
| --- |
| Перечень накопителей PSDrives и опись их содержимого |
| **Экран 4. Перечень накопителей PSDrives и опись их содержимого** |

Просмотрев результаты выполнения команд, вы можете переходить на следующий уровень. Скажем, как показано на экране 4, одна из папок именуется Logins. Просмотреть список учетных записей для входа можно с помощью следующей команды:

PS SQLSERVER:sqlchi-db01default> dir logins

Результаты ее выполнения представлены на экране 5.

|  |
| --- |
| Перечень учетных записей для входа |
| **Экран 5. Перечень учетных записей для входа** |

Учетные записи для входа представляют собой объекты, а это означает, что для вывода дополнительной информации о них вы можете воспользоваться другими составными командами PowerShell. Так, чтобы просмотреть дополнительные сведения о двух учетных записях для входа Globomantics, вы можете переслать предложению where или передать по конвейеру (|) результаты выполнения команды dir:

PS SQLSERVER:sqlchi-db01default> dir logins |

where {$\_.name -match “^Globomantics”} |

Select Name,CreateDate,DatelastModified,DefaultDatabase |

format-list

В журнале данная команда размещается на нескольких строках, но при работе на консоли PowerShell вам следует вводить ее одной строкой. Это касается и других команд, переносимых на следующую строку. Как показано на экране 6, данная команда формирует список всех учетных записей для входа из домена Globomantics, а также сведения о том, когда была создана каждая из учетных записей для входа, когда она была изменена в последний раз и с какой базой данных ассоциируется по умолчанию. Причем все эти сведения представлены в удобном для чтения формате.

|  |
| --- |
| Получение сведений об учетных записях для?входа Globomantics |
| **Экран 6. Получение сведений об учетных записях для?входа Globomantics** |

Если выражение, содержащееся в этой команде, кажется непонятным, пусть это вас не смущает. Стоит немного попрактиковаться — и вы сможете понимать его безо всякого напряжения. Я хочу, чтобы вы получили представление о том, чего я смог добиться с помощью базового выражения PowerShell.

С помощью богатого языка и набора инструментов PowerShell вы сможете получить намного больший объем сведений. Так, если в вашей системе PowerShell установлена интегрированная среда сценариев PowerShell Integrated Scripting Environment (ISE), вы можете просмотреть содержимое баз данных экземпляра chi-db01, для чего необходимо выполнить команду, подобную следующей:

PS SQLSERVER:sqlchi-db01default> dir databases |

sort Size |

select Name,CreateDate,Status,RecoveryModel,Size,Owner |

out-gridview

Подключение среды PowerShell ISE необходимо, поскольку для передачи выходных данных в расположенную в отдельном окне интерактивную таблицу, показанную на экране 7, команда использует составную команду Out-GridView. Как видите, в таблице содержится список баз данных, отсортированных по размеру, а также некоторые другие сведения, касающиеся этих баз данных.

|  |
| --- |
| Отображение в интерактивной таблице перечня баз данных, отсортированных по размеру |
| **Экран 7. Отображение в интерактивной таблице перечня баз данных, отсортированных по размеру** |

Кроме того, пользователь может вносить в объекты изменения с помощью объектной модели PowerShell. Для примера я во всех подробностях опишу процедуру изменения числа допустимых попыток обращения к учетной записи (то есть процедуру изменения свойства AccountRetryAttempts) для Database Mail. Сначала нужно перейти в каталог Mail экземпляра chi-db01 и сформировать перечень составляющих его элементов.

PS SQLSERVER:sqlchi-db01default> cd mail

PS SQLSERVER:sqlchi-db01defaultmail> dir

В содержимом каталога вы обнаружите объект ConfigurationValues. Направив по конвейеру результаты выполнения команды dir ConfigurationValues составной команде Get-Member, вы сможете получить имена и описания свойств и методов упомянутого объекта. Команда выглядит следующим образом:

PS SQLSERVER:sqlchi-db01defaultmail>

dir ConfigurationValues | get-member

Результаты ее выполнения представлены на экране 8.

|  |
| --- |
| Получение имен и описаний свойств и методов объекта ConfigurationValues |
| **Экран 8. Получение имен и описаний свойств и методов объекта ConfigurationValues** |

Проанализировав полученную информацию, вы сможете определить, какие свойства будут использоваться. В данном случае вы будете задействовать свойства Name и Value и использовать их в команде.

PS SQLSERVER:sqlchi-db01defaultmail>

dir ConfigurationValues | Select Name,Value

Результаты выполнения команды отображены на экране 9. Как видите, число допустимых попыток обращения к учетной записи на настоящий момент установлено равным 1.

|  |
| --- |
| Извлечение текущего значения свойства AccountRetryAttempts |
| **Экран 9. Извлечение текущего значения свойства AccountRetryAttempts** |

Чтобы понять, как пользоваться объектами накопителя SQLServer, вы будете многократно передавать эти объекты с помощью конвейера. Чтобы облегчить выполнение этой процедуры, можете сохранять объекты в переменных. Для сохранения объекта ConfigurationValues в переменной $config выполните команду:

$config = dir ConfigurationValues

Переменная $config представляет собой массив объектов конфигурации, значения которых мы можем получать и устанавливать. Поскольку речь идет о массиве с индексом, начинающимся с нуля, а свойство AccountRetryAttempts — это первый ключ в данном массиве, для получения и установки его значения можно использовать параметр $config[0]. К примеру, следующая команда считывает текущее значение свойства AccountRetryAttempts (то есть 1):

PS SQLSERVER:sqlchi-db01defaultmail> $config[0].value

Эта команда устанавливает значение параметра, равное 3:

PS SQLSERVER:sqlchi-db01defaultmail> $config[0].value=3

Дабы убедиться в том, что данное значение изменено, мы можем использовать в этой команде Get-Item:

PS SQLSERVER:sqlchi-db01defaultmail>

get-item ConfigurationValuesAccountRetryAttempts |

select Name,Value

Результаты выполнения команды представлены на экране 10. Как вы можете убедиться, число допустимых попыток обратиться к учетной записи увеличилось до трех.

|  |
| --- |
| Подтверждение получения свойством AccountRetryAttempts нового значения |
| **Экран 10. Подтверждение получения свойством AccountRetryAttempts нового значения** |

Я готов признать, что управление системой SQL Server с помощью провайдера SQL Server, описанное мною выше, может представлять определенную трудность. Поэтому при возможности вы, вероятно, захотите воспользоваться составными командами sqlps, поскольку с ними работать проще.

**Использование команд sqlps**

Чтобы работать с командами sqlps, необходимо предварительно установить в системе модуль sqlps. Команда, которую я считаю наиболее ценной, называется Invoke-SQLCmd. Она была реализована в версии SQL Server 2008, так что обязательно прочитайте справочный файл для той системы, которую эксплуатируете.

Как явствует из названия команды Invoke-SQLCmd, ее задача — запускать команды SQL. Эта задача выполняется с помощью параметра Query. Фактически любое выражение T-SQL может быть выполнено из командной строки PowerShell, скажем, таким образом:

PS C:> invoke-sqlcmd -query «select @@version»

На экране 11 показаны примерные результаты.

|  |
| --- |
| Считывание версии SQL Server |
| **Экран 11. Считывание версии SQL Server** |

Я работал на сервере SQL Server и запускал команду с локального экземпляра SQL Server, так что по умолчанию она выполнялась для локального компьютера. Однако пользователь может указать и другую систему, а также другой экземпляр, следующим образом:

PS C:Usersadministrator> invoke-sqlcmd «select @@version»

-ServerInstance «chi-win7-22»

Установив модули SQL Server PowerShell на своей настольной системе, вы сможете из одного узла управлять несколькими серверами, что весьма удобно. Но пока давайте выполним несколько команд на локальном сервере. Допустим, у вас имеется сценарий New-ComputerDataDB.sql, обеспечивающий создание базы данных ComputerData. Вы можете запустить его на выполнение с помощью составной команды Invoke-SQLCmd:

PS C:> invoke-sqlcmd

-InputFile C:scriptsNew-ComputerDataDB.sql

Помимо создания базы данных вы можете сформировать новую таблицу из командной строки PowerShell. Для этого вам нужно будет создать строку ввода(here-string), содержащую соответствующий запрос. Строка here-string представляет собой строку, занимающую несколько строк и созданную с использованием особого метода. Вот пример такой строки, предназначенной для формирования таблицы Computers:

PS C:> $query=@«

CREATE TABLE dbo.Computers(

Computername ntext NOT NULL,

ReportDate datetime NULL,

PhysicalMemoryBytes bigint NULL,

OS ntext NULL,

ServicePack int NULL,

RegisteredOwner ntext NULL,

RegisteredOrganization ntext NULL

)

»@

Следует заметить, что если вы создаете строку here-string в интегрированной среде сценариев PowerShell ISE, вам нужно проследить за тем, чтобы все строки были выровнены по левому краю. После запуска команды на выполнение переменная $query будет содержать код, необходимый для создания таблицы Computers. После этого вы сможете запустить этот код с помощью такой, например, команды:

PS C:> Invoke-Sqlcmd -query $query

-ServerInstance CHI-DB01

-Database ComputerData

Эта команда формирует таблицу Computers в базе данных ComputerData, которая размещается в экземпляре chi-db01.

Помимо создания таблиц и баз данных вы можете выполнять множество других заданий. Например, я написал несложную функцию Update-MyDB, позволяющую обновлять таблицу Computers с помощью данных, полученных из моей сети. Далее я загрузил эту функцию в свой сеанс PowerShell с помощью команды Invoke-SQLCmd:

Function Update-MyDB {

Param([System.Management.ManagementBaseObject]$os)

invoke-sqlcmd -query «INSERT INTO Computers

(Computername,ReportDate,PhysicalMemoryBytes,OS,

ServicePack,RegisteredOwner,RegisteredOrganization)

Values ('$($os.csname)','$(get-date)','

$(($os.TotalVisibleMemorySize \* 1kb))','

$($os.caption)','$($os.ServicePackMajorVersion)','

$($os.RegisteredUser)','$($os.Organization)' )»

-serverinstance «CHI-DB01» -database ComputerData

} #end function

После загрузки функции я смог запустить фрагмент кода следующим образом:

$computers = Get-Content «MyComputers.txt»

foreach ($computer in $computers) {

Try {

# Get WMI information.

$wmi=Get-WmiObject win32\_operatingsystem `

-computer $computer -ErrorAction Stop

Write-Host «Inserting $($wmi.CSName)» `

-ForegroundColor Green

# Update the SQL Server table.

update-mydb $wmi

}

Catch {

Write-Warning («. {1}» `

-f $computer,$($\_.Exception.Message))

}

} #foreach

Данный фрагмент кода сначала считывает список компьютеров из файла MyComputers.txt. Затем для каждого внесенного в список компьютера он с помощью средства управления Windows Management Instrumentation (WMI) считывает сведения об операционной системе. Наконец, этот фрагмент кода запускает на выполнение функцию Update-MyDB, которая обновляет соответствующие сведения в таблице Computers. На мой взгляд, в этом проявляется самое важное достоинство управления системой SQL Server с помощью технологии PowerShell, поскольку я могу интегрировать запросы SQL в команды PowerShell. Кроме того, я могу с легкостью выполнять запрос на получение данных непосредственно из оболочки PowerShell. Для этого нужно ввести следующий код:

PS C:> Invoke-Sqlcmd -Query «Select \* from Computers»

-ServerInstance CHI-DB01 -Database ComputerData

Результаты представлены на экране 12.

|  |
| --- |
| Получение сведений о компьютерах из базы данных ComputerData |
| **Экран 12. Получение сведений о компьютерах из базы данных ComputerData** |

Вполне вероятно, что при работе в оболочке PowerShell команда Invoke-SQLCmd поможет решить большинство стоящих перед вами задач.

**Резервное копирование и восстановление баз данных**

Последние примеры, которые я представлю вам сегодня, показывают, как снимать резервные копии с баз данных, а также восстанавливать их. Выяснить, в какое время были сняты полные резервные копии базы данных, можно с помощью провайдера SQLServer:

PS C:> dir sqlserver:sqlchi-db01defaultdatabases |

select Name,LastBackupDate

Как показывают примеры результатов, представленные на экране 13, базы данных отображаются в списке с указанием даты и времени выполнения последней полной резервной копии.

|  |
| --- |
| Список резервных копий |
| **Экран 13. Список резервных копий** |

Если в качестве даты резервирования базы данных указывается 1/1/0001, это значит, что копии этой базы данных никогда не создавались. Для выполнения резервного копирования вы можете использовать составную команду Backup-SQLDatabase. Вот как выполняется резервирование содержимого базы данных ComputerData в файл:

PS C:> backup-sqldatabase -ServerInstance chi-db01

-Database ComputerData

-BackupFile «E:BackupComputerData\_01.bak»

Если в процессе копирования вам нужно использовать настройки, применяемые по умолчанию, можете выполнить резервирование всех баз данных с помощью на удивление простой команды:

PS C:> dir sqlserver:sqlchi-db01Defaultdatabases |

Backup-SqlDatabase

В первой части провайдеру SQL Server предписывается составить список баз данных. Затем этот список передается по конвейеру составной команде Backup-SQLDatabase, которая в свою очередь выполняет резервное копирование.

Восстановление базы данных с резервной копии выполняется столь же просто — благодаря наличию составной команды Restore-SQLDatabase. Если речь идет о простой базе данных, восстановление выполняется следующим образом:

PS C:> Restore-SqlDatabase

-ServerInstance CHI-DB01 -Database ComputerData

-BackupFile E:backupComputerData\_01.bak

-RestoreAction Database -ReplaceDatabase

Восстановление баз данных — это тонкая и потенциально сложная операция, так что обязательно ознакомьтесь со справкой по команде Restore-SQLDatabase, а также с примерами ее использования. Кроме того, перед тем как задействовать все эти команды в производственной среде, попрактикуйтесь в работе с ними на тестовых данных.