NosDB Administrators' Guide - PowerShell

Version 1.3

Table of Contents

1. Preface

Welcome to NosDB! NosDB is a schema-less and scalable NOSQL database solution to handle ad-hoc querying on huge amounts of real-time, unstructured data. As NosDB scales out to accommodate the rapidly increasing volume of your data, it applies robust data distribution strategies to ensure availability and fault tolerance at all times. Keeping in mind the suitability of NosDB for Big Data applications, MapReduce and Aggregation support has also been introduced to dramatically enhance performance due to parallel processing.

NosDB features and tools are designed to be tuned flexibly into applications of any size – from small to enterprise-wide global installations.

Support

NosDB provides various sources of technical support. Please refer to Alachisoft's Support page to select a support resource you find suitable for your issue.

To request additional features in the future, or if you notice any discrepancy regarding this document, please drop an email at support@alachisoft.com.

Document Conventions

The following conventions in text have been used throughout this document:

Convention	Description
bold	Specifies terms of importance for the reader.
`monospace`	Specifies inline code snippets, file, class, interface names.
`monospace`	Specifies inline code snippets and commands.
📝	Specifies additional and vital information for the user.
ℹ	Specifies any significant step to be taken in your application.

2. Setting up NosDB PowerShell Environment

NosDB provides integration with Windows PowerShell to easily automate NosDB processes on your network. You can manage NosDB remote tasks on your network through a single computer using PowerShell scripts.

NosDB PowerShell Provider is only compatible with **PowerShell 4.0 and above**.

- Click on the icon in NosDB Management Studio.

Or

- Search for **NosDB PowerShell Management** and **Run as Administrator**.
- You will directed to NosDB's PowerShell module, indicated by the PS at the start of the command line.
- In order to work in NosDB environment through PowerShell, enable PowerShell script execution by setting the `ExecutionPolicy` to `RemoteSigned.`

```
Set-ExecutionPolicy RemoteSigned
```

Once the NosDB environment has been set up, you can now create or connect to the database cluster to carry out NosDB's tasks through PowerShell.

3. Create Database Cluster

`New-DatabaseCluster` creates a database cluster. However, a database cluster must contain a shard with at least one node.

Before creating a database cluster, the following pre-conditions must be fulfilled:

- All machines specified as shard nodes must have both the database and configuration services running.
- These machines must **not** be part of any other database cluster.

Parameters	Description	Default Value
[-Name]*	Name of database cluster to be created	-
[-Shard]*	Name of shard to be added to the cluster	-
[-Port]*	Port for the shard in the cluster	-
[-Server]	Array of IP addresses and corresponding priorities	First node IP resolved by DNS and priority 1

```
New-DatabaseCluster [-Name] [-Shard] [-Port] [-Server]
```

Examples:

```
New-DatabaseCluster -Name Pleiades -Shard shard1 -Port 2101
```

```
New-DatabaseCluster -Name Pleiades -Shard shard1 -Port 2101 -Server 192.168.0.15[3],
192.168.2.14
```

Troubleshooting

Unable to create the cluster.
This occurs if the environment variables have not been refreshed after the installation and might require a reboot of the system.
However, if you do not want to reboot, you can import the NosDB PowerShell module (`NosDBPS.dll`) found in the location: `%INSTALL_DIR%\bin\nosdbps\`.
Execute the following command to import the module:

```
Import-Module NosDBPS
```

4. Connect to Database Cluster

`Connect-DatabaseCluster` establishes a connection with the database cluster which will manage shards, view database details and manage database cluster nodes using the `cluster`.

Parameters	Description	Default Value
`[-Server]`	IP of the node connecting to the database cluster	First node IP resolved by DNS
`[-Port]`	Port of NosDB configuration service for the cluster	9950
`[-StandAlone]`	To specify if the database is standalone	False

```
Connect-DatabaseCluster [-Server] [-Port] [-StandAlone]
```

Examples:

- **If cluster is on the local machine**

```
Connect-DatabaseCluster
```

- **If cluster is on a remote machine**

```
Connect-DatabaseCluster -Server 192.168.1.187 -Port 2147
```

- **If the database is standalone and on a remote location**

```
Connect-DatabaseCluster -Server 192.168.1.187 -Port 2147 -StandAlone
```

Troubleshooting

Unable to connect to the cluster.
This occurs if the environment variables have not been refreshed after the installation and might require a reboot of the system.
However, if you do not want to reboot, you can import the NosDB PowerShell module (`NosDBPS.dll`) found in the location: `%INSTALL_DIR%\bin\nosdbps\`.
Execute the following command to import the module:

```
Import-Module NosDBPS
```

Once connected, the following scripts will be executed from the drive `PS NosDB:\>`. This drive communicates with the underlying database and all commands will be executed through it.

5. Configure Shards

To manage shards, switch context from `cluster` to `shards` in NosDB PSDrive.

```
PS NosDB:\cluster\shards>
```

5.1. Add Shard

`Add-Shard` adds a new shard to the database cluster.

All shards added to the database cluster are auto-started by default.

Parameters	Description	Default Value
`[-Name]*`	Unique shard name	-
`[-Port]*`	Port for the shard	-
`[-Server]`	Array of IP addresses and corresponding priorities	First node IP resolved by DNS and priority 1
`[-HeartBeat]`	Heartbeat interval for the shard nodes in seconds	5

```
Add-Shard [-Name] [-Port] [-Server] [-HeartBeat]
```

Examples:

```
Add-Shard -Name shard1 -Port 2101
```

```
Add-Shard -Name shard1 -Port 2101 -Server 192.168.0.15[3],192.168.2.14 -HeartBeat 8
```

```
Add-Shard -Name shard1 -Port 2101 -Server 192.168.0.15[3],192.168.2.14
```

5.2. Remove Shard

`Remove-Shard` removes the specified shard from the database cluster.

Parameters	Description	Default Value
`[-Name]*`	Name of shard to be removed	-
`[-Quiet]`	To execute the operation in quiet	-

	mode	
[-Forced]	To specify if the removal is forceful	False

```
Remove-Shard [-Name] [-Quiet] [-Forceful]
```

Examples:

```
Remove-Shard shard1
```

```
Remove-Shard shard1 -Quiet -Forced
```

```
Remove-Shard shard1 -Quiet
```

```
Remove-Shard shard1 -Forced
```

5.3. Stop Shard

`Stop-Shard` stops the specified shard of the database cluster. This command works in the context of "shards" and the specific shard as well.

Parameters	Description
[-Name]	Name of shard to be stopped

```
PS NosDB:\cluster\shards> Stop-Shard [-Name]
```

```
PS NosDB:\cluster\shards\shard1> Stop-Shard
```

5.4. Start Shard

`Start-Shard` starts any stopped shard of the database cluster. This command works in the context of `shards` and the specific shard as well.

Parameters	Description
[-Name]	Name of shard to be started

```
PS NosDB:\cluster\shards> Start-Shard [-Name]
```

```
PS NosDB:\cluster\shards\shard1> Start-Shard
```

5.5. Configure Shard Nodes

To manage nodes of a specific shard, change context from `shards` to `[shard-name]` in NosDB PSDrive.

```
PS NosDB:\cluster\shards\[shard-name]>
```

5.5.1. Add Node to Shard

`Add-Node` adds a new node to the shard of the database cluster. The NosDB database service must be installed and running on the specified machine. Note that the node must not be part of any other NosDB database cluster, however they can belong to various shards of the same cluster.

All nodes added to shards are auto-started by default.

Parameters	Description	Default Value
`[-Server]`	IP of the node to be added	First node IP resolved by DNS
`[-Priority]`	Priority of the node being added	1

```
Add-Node [-Priority] [-Server]
```

Examples:

```
Add-Node
```

```
Add-Node -Server 192.168.0.12
```

```
Add-Node -Server 192.168.0.17 -Priority 2
```

5.5.2. Remove Node from Shard

`Remove-Node` removes a node from a shard in the database cluster.

Parameters	Description	Default Value
`[-Server]*`	IP of the node to be added	Local IP resolved by DNS
`[-Quiet]`	To execute the operation in quiet mode	False

```
Remove-Node [-Server] [-Quiet]
```

Examples:

```
Remove-Node -Server 20.200.20.24
```

```
Remove-Node -Server 20.200.20.24 -Quiet
```

5.5.3. Start Node on Shard

`Start-Node` starts the specified node at the shard of the database cluster. This command works in the context of a specific shard or a specific node as well.

Parameters	Description
`[-Server]`	IP of the node to be started

```
PS NosDB:\cluster\shards\shard1> Start-Node [-Server]
```

```
PS NosDB:\cluster\shards\shard1\20.200.20.44> Start-Node
```

5.5.4. Stop Node on Shard

`Stop-Node` stops the specified node at the shard of the database cluster. This command works in the context of a specific shard or a specific node as well.

Parameters	Description
`[-Server]`	IP of the node to be stopped

```
PS NosDB:\cluster\shards\shard1> Stop-Node [-Server]
```

```
PS NosDB:\cluster\shards\shard1\20.200.20.44> Stop-Node
```

6. Data Definition Language (DDL) Querying on NosDB

NosDB PowerShell allows you to execute DDL queries over the underlying database directly through PowerShell. Since the queries are executed through `ExecuteNonQuery()`, there is no result returned to the command prompt, apart from any exceptions occurring during the execution of the query. You can perform `CREATE`, `DROP`, `ALTER`, `BACKUP` and `RESTORE` queries via NosDB PowerShell.

Commands are invoked through the `Invoke-SQL` command. NosDB offers two methods of executing queries through `Invoke-SQL`:

Parameters	Description
`[-Query]`	Query to be executed
`[-InputFile]`	Path of the sql file containing DDL statements

Any parameters within the query will be passed in JSON format as explained in Programmers' Guide.

6.1. -Query

```
Invoke-SQL [-Query]
```

Examples:

1. **CREATE**

```
Invoke-SQL -Query 'CREATE DATABASE northwind {"MultiFile": false, "CacheSize":2117,
"MaxCollections": 17}'
```

2. **DROP**

```
Invoke-SQL -Query 'DROP COLLECTION Products {"Database": "northwind"}'
```

3. **ALTER**

```
Invoke-SQL -Query 'ALTER DATABASE northwind {"Journal": {"FileSizeLimit": 991}}'
```

4. BACKUP

For Backup, NosDB automatically detects whether the backup is distributed or consolidated, based on the `Path` provided. Make sure that if the domain IP is being provided, it should be in the syntax "xx-xxx-xx-xx".

```
Invoke-SQL -Query 'BACKUP DATABASE northwind {"BackupType": "Full", "Path":
"\\server1\Backups", "UserName": "domain1\john",  "Password": "admin1234"}'
```

5. RESTORE

Similarly for Restore, the restoration from distributed or consolidated is determined by the path of the backed up database provided.

```
Invoke-SQL -Query 'RESTORE DATABASE northwind_restored {"RestoreType": "Full", "Path":
"\\server1\Backups\northwind-20160624162210440-completed",  "UserName": "domain1\john",
"Password": "admin1234", "SourceDatabase": "northwind"}'
```

6.2. –InputFile

```
Invoke-SQL [-InputFile]
```

Example:

```
Invoke-SQL -inputFile "D:\SQLProjects\SQLScripts.sql"
```

7. Data Manipulation Language (DML) Querying on NosDB

All non-query DML operations can be performed through PowerShell which include `SELECT`, `INSERT`, `UPDATE` and `DELETE`.

1. SELECT

```
Invoke-SQL -Query 'SELECT Category.Name, Category.Description FROM Products WHERE Name =
"Chai"'
```

2. INSERT

```
Invoke-SQL -Query 'INSERT INTO Products (ID, ProductName, SupplierID, UnitPrice,
UnitsInStock, Discontinued, Category.ID, Category.Name) VALUES
('1','Chai',1,18.000,39,'false',1,'Beverages')'
```

3. UPDATE

```
Invoke-SQL -Query 'UPDATE Orders SET "Order".OrderDetails.Discount = 1 WHERE
"Order".OrderDetails.Quantity > 100'
```

4. DELETE

```
Invoke-SQL -Query 'DELETE FROM Orders WHERE $Order$.OrderDetails[0].Quantity = 100'
```

8. Perform Stress Test on Database

Once the database has been configured, NosDB PowerShell allows testing the stress performance of the connected database.

`Test-Stress` creates a test collection named "`nosdb_test_coll`" and performs multiple Insert, Update, Delete and Get operations on it from single or multiple clients. This command only works in the context of the specified database.

```
PS NosDB:\cluster\databases\[database-name]>
```

Parameters	Description
`[-ThreadCount]`	Number of client threads
`[-TestCaseIterations]`	Number of iterations in a test case
`[-TestCaseIterationDelay]`	Delay between each iteration of the test case
`[-GetsPerIteration]`	Number of Get operations in one iteration of the test case
`[-UpdatesPerIteration]`	Number of Update operations in one iteration of the test case
`[-InsertsPerIteration]`	Number of Insert operations in one iteration of the test case
`[-DeletesPerIteration]`	Number of Delete operations in one iteration of the test case
`[-ReportingInterval]`	To specify the interval after which the stats are reported
`[-TotalIteration]`	The number of times the test case is to be executed

```
Test-Stress [–ThreadCount]  [-TestCaseIterations] [-TestCaseIterationDelay] [-
GetsPerIteration]  [-UpdatesPerIteration] [-DeletesPerIteration] [-InsertsPerIteration [-
ReportingInterval]  [-TotalIteration]
```

Examples:

```
Test-Stress
```

```
Test-Stress -ThreadCount 3 -ReportingInterval 5
```

```
Test-Stress -ThreadCount 2 -TestCaseIterations 2 -TestCaseIterationDelay 1 -
GetsPerIteration 10 -UpdatesPerIteration 10 -DeletesPerIteration 10 -InsertsPerIteration
10 -ReportingInterval 5 -TotalIteration 2
```

9. Deploy Assemblies

NosDB PowerShell allows deploying assemblies over the server, which may include MapReduce implementations, Triggers and UDFs.

`Register-Assemblies` registers the assemblies to be deployed over the server. Once registered, the assemblies can be found in the specified folder at the location:

`%INSTALL_DIR%\database\deployment`.

Parameters	Description
`[-AssemblyPath]*`	File location of the assemblies to be registered
`[-DependentAssemblyPath]`	File location of the dependent assemblies to be registered

If there are multiple assemblies within the path, all of them will be deployed.

```
Register-Assemblies [–AssemblyPath] [-DependentAssemblyPath]
```

Example:

```
Register-Assemblies  -AssemblyPath "C:\Assembly" -DependentAssemblyPath "C:\Dependent"
```

10. Convert Standalone Databases to Clustered Databases

NosDB offers the flexibility of converting a standalone database into a clustered database through the Hash Based Distribution Strategy.

Before converting a standalone database to clustered database, the following pre-conditions must be fulfilled:

- The entire cluster must be running during conversion.
- The standalone database must exist on a node which is the primary node of any shard in the cluster.
- No client operations are to be performed during the conversion, as state transfer is taking place.

To convert a stand-alone database to clustered, switch context to the specified standalone database in NosDB PSDrive.

```
PS NosDB:\standalone\databases\[database-name]>
```

`ConvertTo-ClusteredDatabase` converts the standalone database into a clustered database, given that the aforementioned pre-conditions have been fulfilled.

Parameters	Description	Default Value
`[-ClusterConfigServer]`	IP of the active node in the running service	First node IP resolved by DNS
`[-Port]`	Port of the active node in the running service	9950
`[-NewDatabaseName]`	The database name after conversion	Standalone database name

```
ConvertTo-ClusteredDatabase [-ClusterConfigServer] [-Port] [-NewDatabaseName]
```

Examples:

```
ConvertTo-ClusteredDatabase -NewDatabaseName CastleBlack
```

```
ConvertTo-ClusteredDatabase -ClusterConfigServer 192.168.1.125 -Port 2188 -NewDatabaseName
CastleBlack
```

11. Export Cluster Configuration

NosDB PowerShell allows exporting configuration of the specified database cluster to a file in JSON format.

`Export-Configuration` exports the configuration to a JSON file named as *clustername_timestamp*.json.

Parameter	Description
`[-Path]`	Location for the output file

```
Export-Configuration [-Path]
```

Example:

```
Export-Data -Path 'D:\NorthwindConfiguration'
```

12. Export Data from Collection

NosDB PowerShell allows exporting data from a collection to a file in JSON or CSV format. In case a query is specified, the result of the query is stored to a file.

12.1. Pre-conditions for exporting data

JSON Export

Since data is stored in the collection in JSON format, the data export from JSON to JSON is smooth and requires no special limitations.

Data will be exported in the form of an **array of JSON documents** to the JSON file.

CSV Export

A CSV file is comparatively more rigid than JSON, thus the collection data in JSON has to be parsed accordingly before export. As a CSV file requires headers, NosDB creates the headers by taking the first document of the collection and parsing the attributes accordingly.

However, because of the comparatively more rigid nature of CSV, the following tips must be kept in mind before exporting the data:

1. The data sequence and number of attributes in the collection should be consistent as the CSV headers are created using the first document in the collection. Hence, in successive documents, the number of attributes can be lesser than those defined as the header but should preferably not be more than them.

2. Make sure that a multivalve attribute (array or embedded document) has the same data type. For example, if *Address* is defined as an array in one document and as an embedded document in the other, a conflict will occur during export of the CSV while populating the headers.

```
{
        "_key": "DA136",
        "Name": "Emily",
        "Address": ["20", "EastSide", "TX"]
 },

{
        "_key": "DA123",
        "Name": "John",
        "Address":
        {
                "House": "45",
                "Street": "WestSide",
                "State": "TX"
        }
 }
```

   This is because if the file is exported with a query such as "`SELECT _key, Name, Address.State FROM Employees`", the document *DA136* will not return any meaningful data in the result set due to the inconsistency in format.

`Export-Data` exports either the whole data from a collection, or filtered data as a query result to the file. This command only works in the context of the specified collection.

```
PS NosDB:\cluster\databases\[database-name]\collections\[collection-name]>
```

Parameters	Description	Default Value
`[-Format]*`	Format of data in the output file	-
`[-Path]*`	Location for the output file	-
`[-FileName]`	Custom name for the output file	*DatabaseName_CollectionName*
`[-Query]`	Query to filter the data to be exported	-

```
Export-Data [-Format] [–Path] [-FileName] [-Query]
```

Examples:

```
Export-Data -Format CSV -Path 'D:\Northwind Data'
```

```
Export-Data -Format JSON -Path 'D:\Northwind Data' –FileName 'GreaterThan25' -Query
'Select * from Product where ProductID > 25'
```

13. Import Data to Collection

NosDB PowerShell allows importing data from a JSON or CSV file to a collection. Moreover, the client has the facility to update the data if any data with the same key exists in the collection.

13.1. Pre-conditions for importing data

JSON Import

Since data is stored in the collection in JSON format, the data export from JSON to JSON is smooth and requires no special limitations except that the JSON file being provided is an **array of JSON documents**.

CSV Import

Before importing CSV to a NosDB collection, consistency has to be ensured between the format of data in the CSV file and JSON collection format. Hence, just as with data export, a CSV file import requires taking the following guidelines into consideration:

1. The header and its corresponding value should not mismatch as NosDB will populate the collection according to the provided format.

2. The format of any date specified within the CSV should comply with **ISO 8601** (YYYY-MM-DD). Any other format of date will be treated as string.

3. Since CSV is schema-independent, the system is actually "guessing" the datatypes while importing it into the collection. Hence, a numerical value specified as a string will be treated as string in the collection, which might lead to inconsistency.

4. Any attribute named "`_key`" in the CSV will be treated as the document key by default. If no such attribute exists, NosDB will generate the document key automatically.

`Import-Data` imports data from the specified file into the specified collection. This command only works in the context of the specified collection.

```
PS NosDB:\cluster\databases\[database-name]\collections\[collection-name]>
```

Parameters	Description	Default Value
`[-Path]*`	Location of the input file	-
`[-Format]*`	Format of data in the input file	-
`[-UpdateMode]`	To update data in the collection if it contains the same key	False

```
Import-Data [–Path] [-Format] [-UpdateMode]
```

Examples:

```
Import-Data -Format JSON -Path ‘D:\Northwind Data\product.json’
```

```
Import-Data -Format CSV -Path ‘D:\Northwind Data\product.csv’ -UpdateMode
```

14. Move Collection within Shards

NosDB PowerShell allows moving either single-sharded or capped collections within the shards, as they are contained within a single shard. Normal collections are not moved as they reside on various shards.

`Move-Collection` moves the collection to the specified shard. This command only works in the context of a single sharded or capped collection in NosDB PSDrive.

```
PS NosDB:\cluster\databases\[database-name]\collections\[collection-name]>
```

Parameter	Description
[-NewShard]	Name of shard on which the collection will be moved

If the name of the shard includes any special characters, specify the name within single quotes.

```
Move-Collection [–NewShard]
```

Examples:

```
Move-Collection shard1
```

```
Move-Collection ‘shard-1’
```

15. Get NosDB Version Information

`Get-NosDBVersion` returns details about the currently installed version of NosDB. The following is displayed:

- Utility version
- Copyright
- Registered User details (provided during installation)
- Edition
- Evaluation period

```
Get-NosDBVersion
```

16. Get Backup/Restore Task Information

This tool is only available in **NosDB Open Source** Edition.

`Get-TaskInfo` fetches and displays the current status of the backup and restore jobs being performed.

Parameters	Description	Default Value
`[-ShowHistory]`	Flag to show task information for either the current running task or all tasks that have been executed.	false

```
Get-TaskInfo [-ShowHistory]
```

Examples:

```
Get-TaskInfo
```

```
Get-TaskInfo -ShowHistory
```

17. View Configured Components

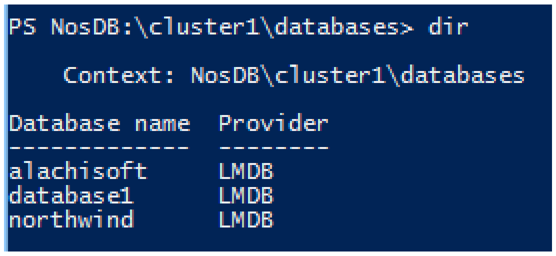
17.1.Databases

To view configured database information, switch to the context `databases` in NosDB PSDrive.

```
PS NosDB:\cluster\databases>
```

Typing `dir` will display all configured databases and the storage provider selected against it.

```
PS NosDB:\cluster\databases> dir
```



```
PS NosDB:\cluster1\databases> dir

    Context: NosDB\cluster1\databases

Database name  Provider
-------------  --------
alachisoft     LMDB
database1      LMDB
northwind      LMDB
```

17.2. Collections

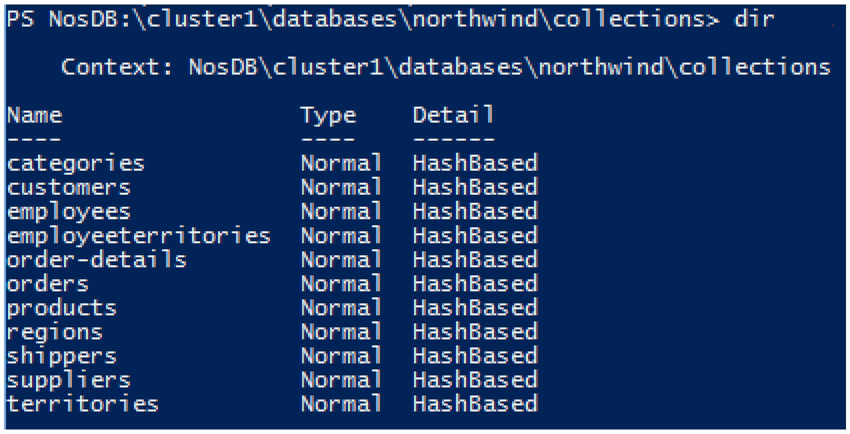
To view configured collections of a specific database, change context from `databases` to `[database-name]`.

```
PS NosDB:\cluster\databases\[database-name]>
```

- Switch context from `[database-name]` to `collections`.

```
PS NosDB:\cluster\databases\[database-name]\collections>
```

- Typing `dir` shows the configured collections and their types in the database.



```
PS NosDB:\cluster1\databases\northwind\collections> dir

    Context: NosDB\cluster1\databases\northwind\collections

Name                     Type    Detail
----                     ----    ------
categories               Normal  HashBased
customers                Normal  HashBased
employees                Normal  HashBased
employeeterritories      Normal  HashBased
order-details            Normal  HashBased
orders                   Normal  HashBased
products                 Normal  HashBased
regions                  Normal  HashBased
shippers                 Normal  HashBased
suppliers                Normal  HashBased
territories              Normal  HashBased
```

17.3. Indexes

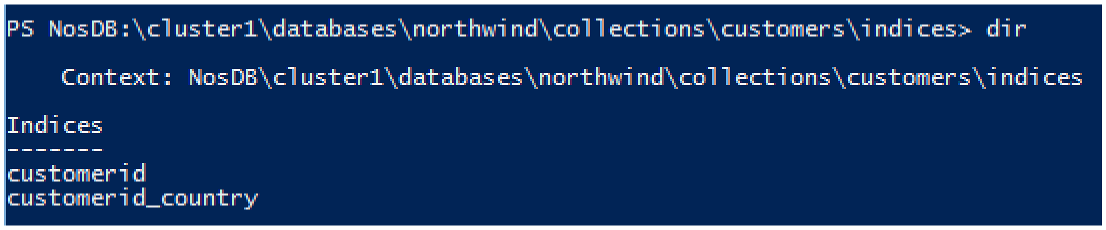
To view configured indices of a specific database, change context from `collections` to `[collection-name]`.

```
PS NosDB:\cluster\databases\[database-name]\collections\[collection-name]>
```

- Switch context from `collections` to `indices`.

```
PS NosDB:\cluster\databases\[database-name]\collections\[collection-name]\indices>
```

- Typing `dir` the configured indices in the collection.



```
PS NosDB:\cluster1\databases\northwind\collections\customers\indices> dir

    Context: NosDB\cluster1\databases\northwind\collections\customers\indices

Indices
-------
customerid
customerid_country
```

17.4.Triggers

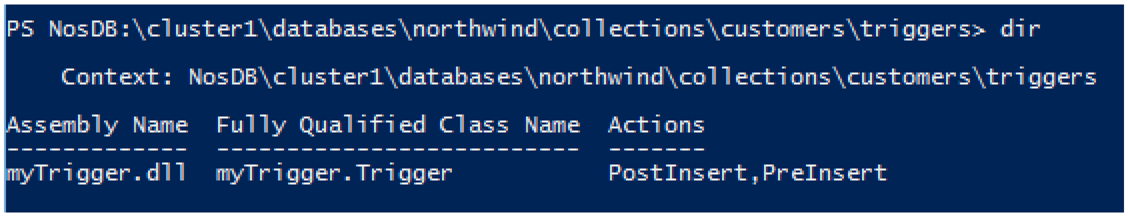
To view configured triggers of a specific collection, change context from `collections` to `[collection-name]`.

```
PS NosDB:\cluster\databases\[database-name]\collections\[collection-name]>
```

- Switch context from `collections` to `triggers`.

```
PS NosDB:\cluster\databases\[database-name]\collections\[collection-name]\triggers>
```

- Typing `dir` displays the deployed trigger's assembly name, fully qualified class name and actions in the collection.



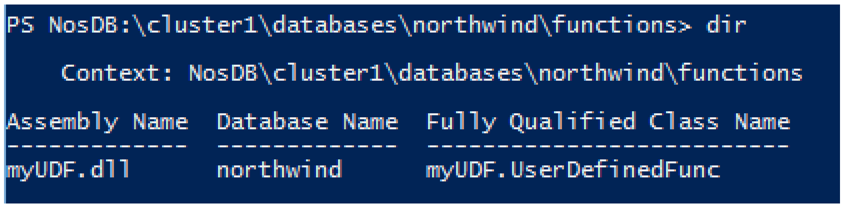
```
PS NosDB:\cluster1\databases\northwind\collections\customers\triggers> dir

    Context: NosDB\cluster1\databases\northwind\collections\customers\triggers

Assembly Name  Fully Qualified Class Name   Actions
-------------  --------------------------   -------
myTrigger.dll  myTrigger.Trigger            PostInsert,PreInsert
```

17.5. User Defined Functions

To view configured user defined functions of a specific collection, switch context from `[database-name]` to `functions`.

```
PS NosDB:\cluster\databases\[database-name]\functions>
```

- Typing `dir` displays the configured functions and their fully qualified class names in the database.



```
PS NosDB:\cluster1\databases\northwind\functions> dir

    Context: NosDB\cluster1\databases\northwind\functions

Assembly Name  Database Name  Fully Qualified Class Name
-------------  -------------  --------------------------
myUDF.dll      northwind      myUDF.UserDefinedFunc
```