Advanced Scripting   
ODBC Data

Last Updated: 11/5/2021 4:04 PM Version 1  
Document Prepared for: CIT361 Student

# Name Zach Lucas ID 895649438

# Instructions

Save a copy of this document. Answer all questions directly in this document. You will save and upload this completed document as your homework submission.

# Overview

In this exercise you will work with a Microsoft Access Database Using ODBC. ODBC can be used to access many database types including oracle, mysql, postgreSQL, DB2, Excel and many more.

# Requirements

PoweShell on Windows (ODBC is not available in Linux)

Class sample files. <http://lf.citwdd.net/psfiles.zip>

# Setup

# Task 1—Discover Installed ODBC Drivers.

Before you can use ODBC you need to ensure the drivers are installed.

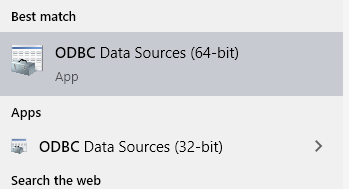
## Steps

1. List the installed drivers  
   Get-OdbcDriver |Ft
   1. Look for a driver that supports \*.mdb files. You may see multiple drivers. Look for the English named driver. What is the exact driver name? Microsoft Access Driver
   2. What platform is the driver for? There is both a 32-bit and 64-bit

# Task 2—Data Source Names

A Data Source Name is a preconfigured data source that is stored on your computer. It can be stored per user or per machine. The easiest way to create a DSN in with the ODBC applet in the control panel. You can also Manage DSNs with PowerShell.

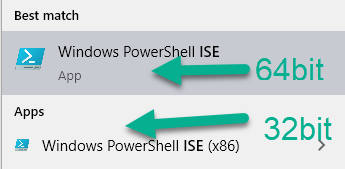
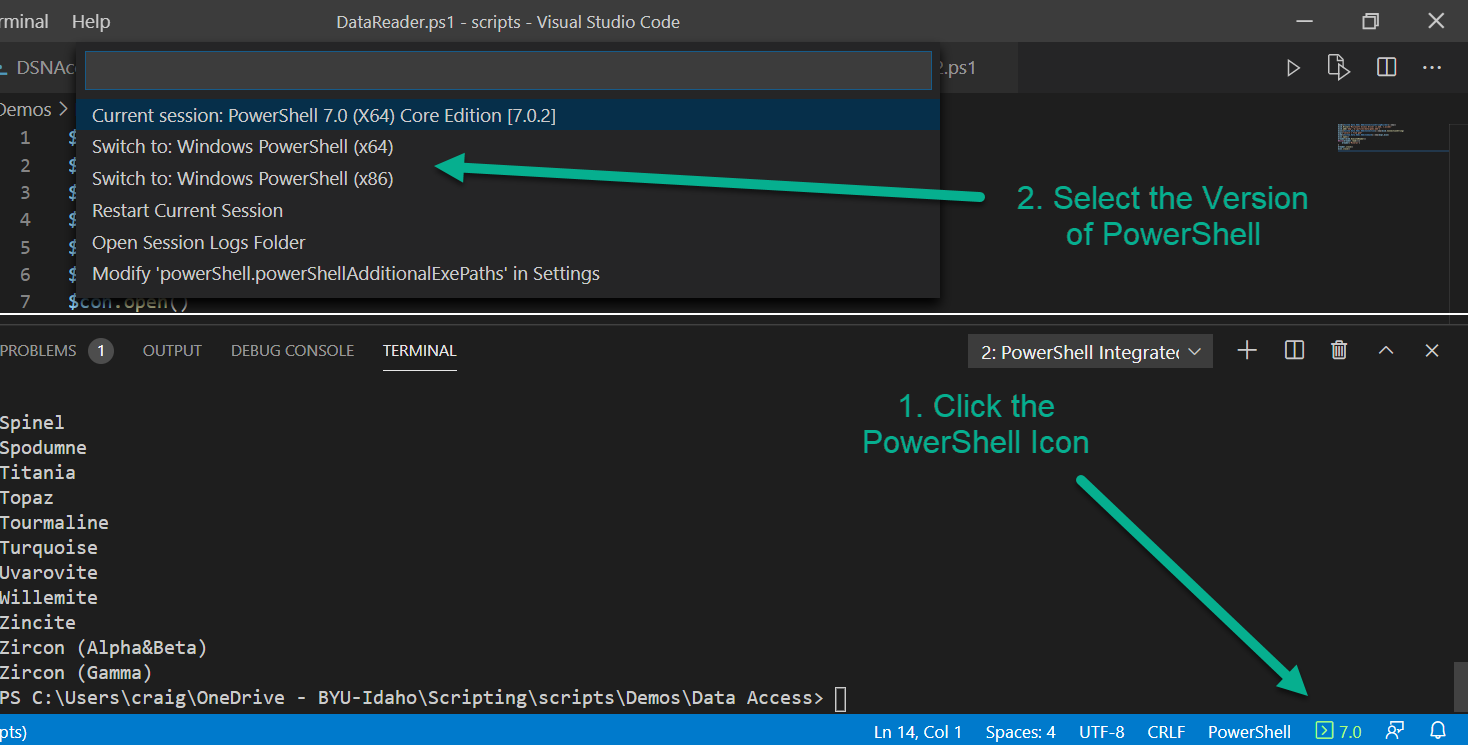
## Steps

1. Based upon the information your discovered int Task1 you will need to open either the 32 or 64 bit version of the ODBC Data Sources configuration tool. Click start then type ODBC. You should see  
   
2. Launch the appropriate configuration tool based upon the platform of the driver you discovered.
3. The User DSN tab should be open. Click the **Add** button to create a new DNS.
4. Select the **Microsoft Access Driver**.
5. Click **Finish**. A setup windows should appear.
   1. Enter **Gems** for the Data Source Name
   2. Click the **Select** button to choose the database file. The file is psfiles\data\gems.mdb
   3. When done click OK to close the configuration screen
6. Click OK to close the config tool
7. Now any time you want to connect to the gems.mdb database you can simply use the DSN gems.

# Task 3—Accessing Data Via a DataAdapter

A data adapter is an object that does all the communication between the database server/file and the in-memory copy of the returned data. It requires very little knowledge of SQL to work with.

## Steps

1. Read data from the gems database into a dataset via a data adapter. Now you will enter the code to setup a data adapter and dataset using the previously created DNS. You will want to use either VSCode or the PowerShell ISE. It is important to run the appropriate version of PowerShell based on the driver you selected. If you are using a 32bit driver you must use the 32bit version of the ISE, or the 32bit version of PowerShell in VSCode.
   1. If you are using the ISE just start the correct version  
      
   2. If you are using VSCode you can select the version of powershell by clicking on the PowerShell icon int the taskbar, then selecting the version of PowerShell from the menu.  
      
2. Create a file named **DataAccessDA.ps1**
3. First Create a DataAdapter. The Data adapter’s constructor take two arguments. First is the sql you want to execute to fill the dataset with. The second is the ConnectionString that describes the driver you want to use. In this case the Connection string specifies using a DSN.  
   $da=[System.Data.Odbc.OdbcDataAdapter]::new('select \* from gem','DSN=gems')
4. Now you need a place to put the data. Create a DataSet Object  
   $ds=[System.Data.DataSet]::new()
5. Instruct the DataAdapter to execute the SQL and fill the dataset with the results  
   $da.fill($ds)
6. Finally show the results  
   $ds.Tables[0]|ft
7. Run the code. If you get an error that says *“The specified DSN contains an architecture mismatch between the Driver and Application”* you need to change your version of PowerShell.
8. Save your PowerShell file.
9. The DataSet contains a collection of tables which in turn contain a collection of rows. Each row represents a row of data returned. The Row has an index for each column in the table. After a successful run of your script use the interactive shell to explore the dataset.
   1. How many tables were returned?1  
      $ds.tables.count
   2. How many rows are there in the first table? 68  
      $ds.tables[0].rows.count
   3. Take a look at some of the data  
      $ds.tables[0]|select mineral,hardness|sort hardness
   4. Look at just the hard minerals  
      $ds.tables[0]|? hardness -ge 5|select mineral,hardness|sort hardness -Desc
   5. How many gems have a hardness of 5 or more? 56
10. Copy your script here  
    (SCRIPT I TYPED TO FIND THE NUMBER OF GEMS THAT HAD A HARDNESS OF 5 OR MORE) $ds.tables[0]|? Hardness -ge 5|select mineral,hardness|sort hardness -Desc|measure  
      
    (SCRIPT)  
    $da=[System.Data.Odbc.OdbcDataAdapter]::new('select \* from gem','DSN=gems') $ds=[System.Data.DataSet]::new()  
    $da.fill($ds)  
    $ds.Tables[0]|ft

# Task 4—ConnectionStrings

In this task you will connect to the same data but use a connection string rather than a DSN

## Steps

1. Create a file Named DataAccessCS.ps1
2. You can create a connection string manually or by using the ConnectionStringBuilder object. Either way you pass the connection string to the DataAdapter. You will need to know your driver name and database file name. Replace the highlighted code as appropriate for your system. $csb=[System.Data.Odbc.OdbcConnectionStringBuilder]::new()  
   $csb.Driver= 'Microsoft Access Driver (\*.mdb, \*.accdb)'  
   $csb.add('dbq','d:\psfiles\data\gems.mdb')  
   $da=[System.Data.Odbc.OdbcDataAdapter]::new('select \* from gem',$csb.ConnectionString)  
   $ds=[System.Data.DataSet]::new($da)  
   $da.fill($ds)  
   $ds.Tables[0]|ft
3. Save the file then run it to make sure it works.
4. Copy your script here  
   $csb=[System.Data.Odbc.OdbcConnectionStringBuilder]::new()  
   $csb.Driver= 'Microsoft Access Driver (\*.mdb)'  
   $csb.add('dbq','c:\Users\Zach\psfiles\data\gems.mdb')  
   $da=[System.Data.Odbc.OdbcDataAdapter]::new('select \* from gem',$csb.ConnectionString)  
   $ds=[System.Data.DataSet]::new($da)  
   $da.fill($ds)  
   $ds.Tables[0]|ft

# Task 5—Command and Connection objects

You can work with data without using the in-memory dataset. Using the OdbcCommand, and OdbcConnection Objects.

## Steps

1. Reading data: Data is retrieved from the database using a datareader object. A data reader is readonly forward only cursor to the results of your query. To retrieve the data you create a loop and call the datareader’s read method to get each row until you are done or there are no more rows. Create a script name DataAccessDR.ps1 and enter the following code.
2. First you still need a connection string so use the builder as before to create it. As before, replace the highlighted information as appropriate for your system.  
   $csb = [System.Data.Odbc.OdbcConnectionStringBuilder]::new()  
   $csb.Driver = 'Microsoft Access Driver (\*.mdb, \*.accdb)'  
   $csb.Add('dbq', 'd:\psfiles\data\gems.mdb')
3. Now create a connection object, the constructor takes a ConnectionString  
   $con = [System.Data.Odbc.OdbcConnection]::new($csb.ConnectionString)
4. Create a Command Object, the Command Object contains the SQL and passes it to the Connection object for execution on the database.  
   $cmd = [System.Data.Odbc.OdbcCommand]::new(‘select \* from gem’, $con)
5. Once you have defined the object open a connection to the database, the connection can be reused as necessary.  
   $con.Open()
6. Next call the appropriate method for the kind of SQL statement in the command. There are 3 kinds, row returning (select), non row returning (insert, update,delete), and Single Value returning, they are ExecuteReader, ExecuteNonQuery, and ExecuteScalar respectively. Since our SQL is a select statement use the ExecuteReader Method.  
   $reader = $cmd.ExecuteReader()
7. The ExecuteReader method returns a DataReader object. The DataReader object has a read method that returns the next row from the results. It returns a true if it was successful, false otherwise. The DataReader Contains the results of the read. You can retrieve the values by passing the column name to the reader object as a string like $reader[‘column’]. Complete the script (note: it is important to close the connection and reader object when done)  
   $reader = $cmd.ExecuteReader()  
   While ($reader.Read()) {  
    Write-host $reader['Mineral'] $reader['hardness']  
   }  
   $reader.Close()  
   $con.Close()
8. Save and try it out. You should get a list of the mineral names
9. Copy your script here  
   $csb=[System.Data.Odbc.OdbcConnectionStringBuilder]::new()  
   $csb.Driver= 'Microsoft Access Driver (\*.mdb)'  
   $csb.add('dbq','c:\Users\Zach\psfiles\data\gems.mdb')  
     
   $con=[System.Data.Odbc.OdbcConnection]::new($csb.ConnectionString)  
     
   $cmd=[System.Data.Odbc.OdbcCommand]::new('Select \* from gem', $con)  
     
   $con.Open()  
     
   $reader=$cmd.ExecuteReader()  
   while ($reader.Read()) {  
    Write-Host $reader['Mineral'] $reader['hardness']  
   }  
   $reader.Close()  
   $con.Close()

# Deliverable

Upload this document with completed answers to i-learn.