Name Click here to enter name ID Click here to enter id

ODBC

Exercise 6.2

# Instructions

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 use the *Open Database Connectivity* (ODBC) application programming interface (API) to work with a Microsoft Access database file.

ODBC can be used to access a wide variety of database engines, including Oracle, MySQL, postgreSQL, IBM Db2, Snowflake, MongoDB, and many others. It can also access tables in local databases created by Microsoft Access, SQLite, Microsoft Excel, etc.

# Requirements

PowerShell (Desktop or Core) for Windows. (There exist third-party ODBC drivers for Linux and macOS, but finding, installing, or troubleshooting them is beyond our scope for this exercise.)

The sample files for the course (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 a driver with an *English* name. What is the **exact** driver name? Click or tap here to enter text.
   2. What platform is the driver for? Is it a 32- or 64-bit driver? Click or tap here to enter text.

# 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 is with the ODBC applet in the control panel. You can also manage DSNs with PowerShell.

## Steps

1. Based on the information your discovered in Task 1, you will need to open either the 32 or 64 bit version of the ODBC Data Sources configuration tool. Click Start (or tap the Windows “flag” key), then type **ODBC**. You should see:   
   A screenshot of a computer

   Description automatically generated
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 DSN.
4. Select **Microsoft Access Driver**.
5. Click **Finish**. A setup dialog window should appear.
   1. Enter **gems** for the Data Source Name.
   2. Click the **[Select]** button to choose the **gems.mdb** database file. Look for it where you unzipped the **psfiles.zip** archive file. Its path will end with **psfiles\data\gems.mdb**
   3. When finished, 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 Using an OdbcDataAdapter

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

## Steps

1. Let’s read data from the gems database into a dataset via a data adapter. Now you will enter code to setup a data adapter and a dataset, using the previously created DSN. Please use either Visual Studio Code 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 32-bit driver, you must use the 32-bit version of the ISE, or the 32-bit version of PowerShell in VSCode.
   1. If you are using the ISE, just start the correct version:   
      A close-up of a computer

      Description automatically generated
   2. If you are using VSCode, you can select the version of **powershell** by clicking on the PowerShell icon in the taskbar, then selecting the version of PowerShell from the menu.   
      A screenshot of a computer

      Description automatically generated
2. Create a script file named **DataAccessDA.ps1**
3. In your script, first Create a *data adapter*. The data adapter’s constructor takes two arguments. First is a SQL query to execute, to fill up the *in-memory* dataset. The second is the ConnectionString that describes the driver you want to use. In this case, the connection string is specified using a DSN.   
   $da=[System.Data.Odbc.OdbcDataAdapter]::new('select \* from gem','DSN=gems')
4. Now you need a place to put the data, so 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. Now that you have written those four commands in your script, save and run the script. If you get an error that says *“The specified DSN contains an architecture mismatch between the Driver and Application”* then you need to change your version of PowerShell.
8. The DataSet contains a collection of tables, and each table contains a collection of rows. Each row represents a row of returned data. The Row has an index for each column in the table. After you successfully run of your script, use the interactive shell (the ISE Console or VSCode terminal) to explore the dataset.
   1. Enter:   
      $ds.tables.count   
      How many tables were returned? Click or tap here to enter text.
   2. Enter:   
      $ds.tables[0].rows.count   
      How many rows are there in the first table? Click or tap here to enter text.
   3. Take a look at some of the data:   
      $ds.tables[0]|select mineral,hardness|sort hardness
   4. Look at just the *hard* minerals, by excluding those with hardness value lower than 5:   
      $ds.tables[0]|? hardness -ge 5|select mineral,hardness|sort hardness -Desc
   5. How many gems have a hardness of 5 or more? Click or tap here to enter text.
9. Copy the commands in your **DataAccessDA.ps1** script here:   
   Click or tap here to enter text.

# Task 4—Accessing Data Using an OdbcConnectionStringBuilder

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

## Steps

1. Create a new script file named **DataAccessCS.ps1**
2. You can create a connection string manually, or by using the OdbcConnectionStringBuilder object. Either way you pass the connection string to the data adapter. 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 and run your script to make sure it works.
4. Copy your script code here:   
   Click or tap here to enter text.

# Task 5—OdbcCommand and OdbcConnection objects

You can work with data *without* populating an in-memory dataset, using *OdbcCommand* and *OdbcConnection* objects.

## Steps

1. Reading data: data is retrieved from the database using a *data reader* object. A data reader is a *read-only forward-only cursor* to the results of your query. To retrieve the data, you create a loop and call the data reader’s **read** method to get each row until you are done or there are no more rows. Create a new script named **DataAccessDR.ps1** and enter code as follows.
2. First, you still need a connection string, so use the builder as we did before to create it. (Again, replace 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. Its constructor requires a ConnectionString argument.   
   $con = [System.Data.Odbc.OdbcConnection]::new($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', $con)
5. Once you have defined the connection and command objects, open a connection to the database. The connection can be reused as necessary.   
   **$con.Open()**
6. Next, call the appropriate method depending on the kind of the SQL statement in the command object. There are three kinds: to in the command. There are 3 kinds: those that return collections of rows (such as SELECT queries), those that do not return any rows (usually INSERT, UPDATE, or DELETE statements), and those that return a single value. These three methods are named **ExecuteReader**, **ExecuteNonQuery**, and **ExecuteScalar**, respectively. Ours is a select statement ('select \* from gem'), so we’ll use the **ExecuteReader()** method.   
   $reader = $cmd.ExecuteReader()
7. The **ExecuteReader()** method returns a **DataReader** object. It has a **Read()** method that returns the next row from the results. It returns **True** if it was successful, **False** otherwise. The **DataReader** contains the results of the read operation. You can retrieve the values by passing the column name to the reader object as a string, like **$reader[**'**column**'**]**. Complete your script with the following lines. (Note: it is *important* that your script closes the connection and reader object when it’s done!)   
     
   while ($reader.Read()) {   
    Write-host $reader['Mineral'] $reader['hardness']   
   }   
   $reader.Close()   
   $con.Close()
8. Save and try it out. The script’s output should show mineral names and hardnesses.
9. Copy your script’s code here:   
   Click or tap here to enter text.

# Deliverable

Upload this document with completed answers to I-Learn Canvas.