3

SQL Editing Reimagined

Database developers typically spend a great deal of time creating and editing SQL queries, so it only makes sense to make this experience as helpful, convenient and productive as possible. Azure Data Studio has truly reimagined how developers interact with SQL coding, and for that matter, all the languages supported on the platform.

This is accomplished by focusing directly on keyboard interactions, which include IntelliSense, keywords, code snippets, and existing object definitions. Much of the ADS User Interface is also configurable, providing customizable color themes, zoom levels, window options, fonts and icons.

# IntelliSense, Snippets and Object Definitions

To get started with entering SQL queries, you can either click on ‘New query’ from the Welcome page, or for more specificity, right click on your target Database in the ‘Side Bar’, and then choose ‘New Query’ as shown in figure 3-1:

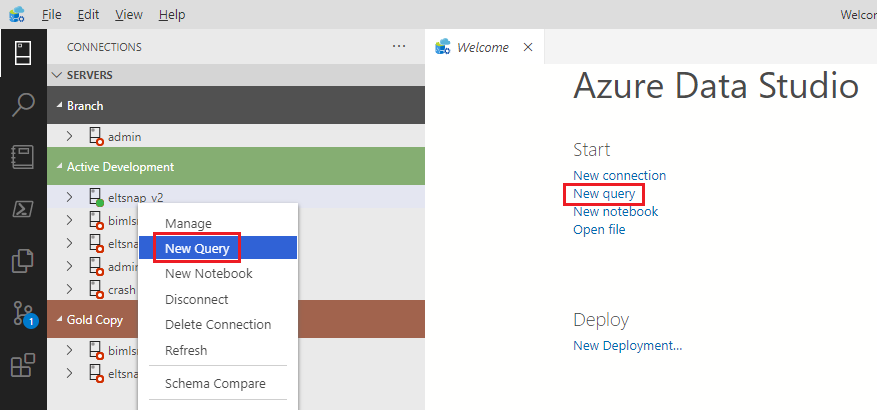


Figure 3-1. Create a New SQL Query

This will open a blank editing window which where you simply start typing. As you start typing, each keystroke may offer a suggestion, providing IntelliSense as shown in the pop-up window for Figure 3-2. Notice that the FROM keyword is highlighted in the sorted pop-up list which indicates that you simply need to hit the tab key to implement this substitution.

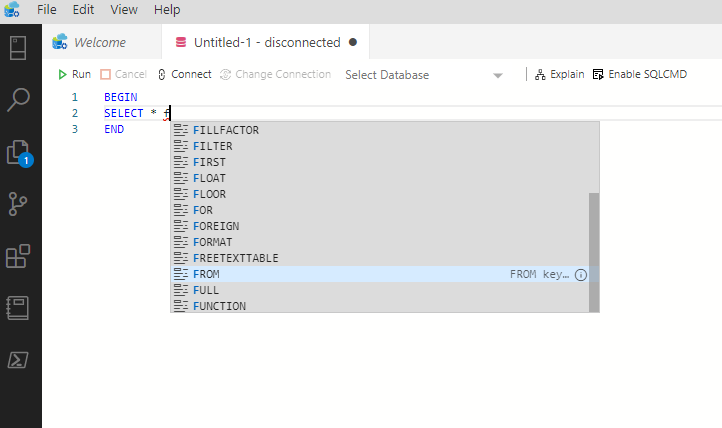


Figure 3-2. IntelliSense Keyword Pop-up

The up and down arrows provide navigation within this list, as does a mouse-click on the desired keyword. Also notice that figure 3-2 has the ‘Side Bar’ hidden. Use the key sequence Ctrl+B to toggle this screen section off and on. You can achieve the same effect by clicking on an icon in the ‘Activity Bar’, or optionally use the top ‘Menu Bar’ selections: View, Appearance, Show Side Bar.

## Code Snippets

One of my favorite editing features in ADS are ‘Code Snippets’. These can be a huge timesaver, are integrated into the IntelliSense user experience, and are customizable. As an example, let’s say you wanted to create a table. By typing createtable you will see the ‘camel cased’ snippet sqlCreatTable pop-up. You just hit ‘Tab’ when highlighted, or click on the snippet in the pop-up, and you will get a ‘Create Table’ template as displayed in figure 3-3:

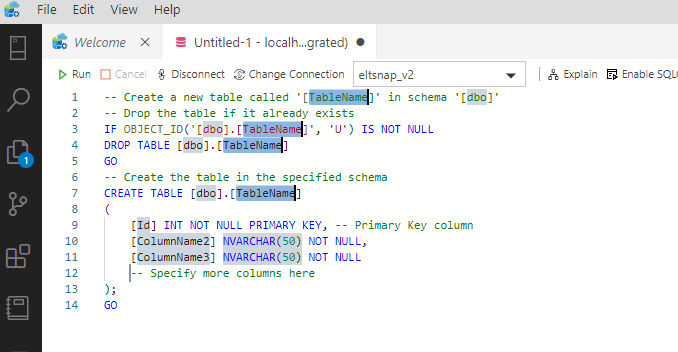


Figure 3-3. Create Table (Default) Snippet

You may be surprised to now see 4 blinking cursors! This is because you are automatically placed in the process of completing defined variable replacements, the first of which (TableName) has a total of 4 instances. If you were to type ‘product’ the variable replacement would occur 4 times and figure 3-4 would be the result.

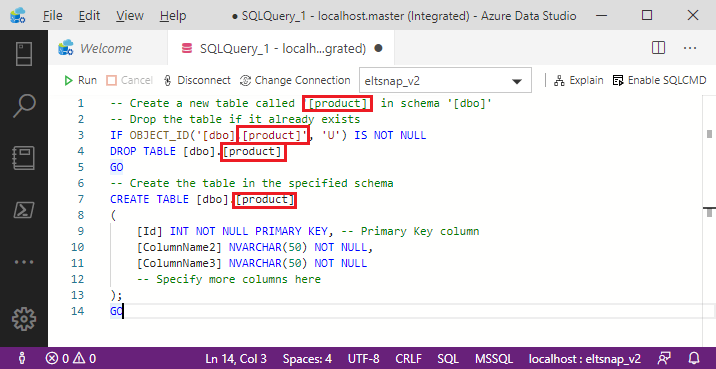


Figure 3-4. Snippet Variable Replacement

To move to the next defined variable, press the ‘Tab’ key again and you will see the next variable (used for the schema name) which also has 4 occurrences that can be changed simultaneously. The next ‘Tab’ will take you to the first table column (in this case: [Id]), and so on until all variables have been visited.

This ‘default’ (built-in) snippet is a nice start to creating a table, but you may be thinking “I’d like my snippet customized for our coding standards”. Not to worry, in chapter 9 we will cover how you can easily create your own snippets. Just like the built-in snippets, these will automatically surface in your SQL editor window using the same IntelliSense driven keystrokes.

## Object Definitions

While you are editing your SQL Queries, it is a common requirement to reference ‘Object Definitions’ within your database model. For example, let’s say you are querying a certain table column and need to know if it could contain NULL values. In this case, the standard IntelliSense capability of suggesting ‘column names’ falls a bit short. Instead, what is needed is the full definition of the table object.

Since your database could contain hundreds of tables, each of which could have many columns, it can be a pain to quickly retrieve table and column definitions by browsing these object definitions in the ‘Side Bar’. To remedy this situation, ADS provides direct access to object definitions, without leaving the editor window. Simply ‘right click’ on any table name in your query, and a couple options will pop-up. Figure 3-5 captures this pop-up when right clicking on the table name oledb\_connection:

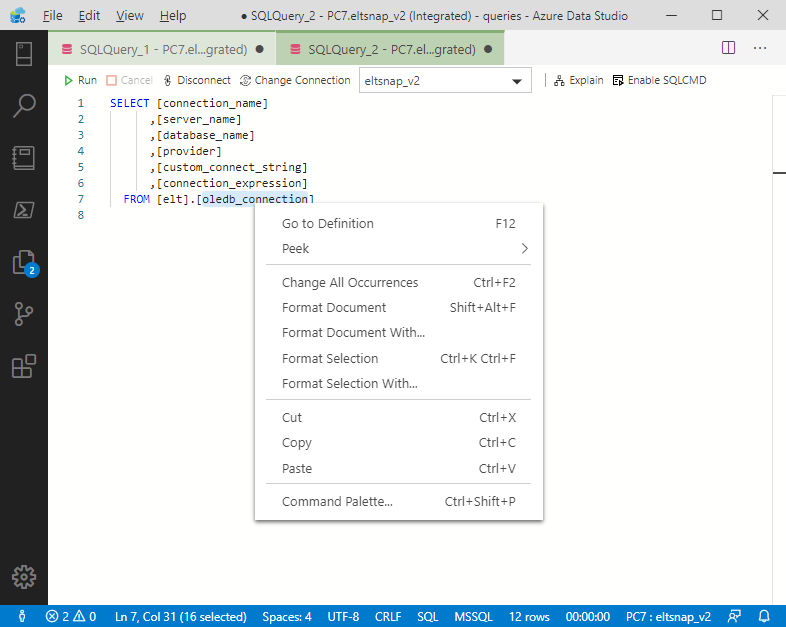


Figure 3-5. Accessing ‘Object Definitions’

The first two options on this pop-up will provide you with the table definitions. The first “Go to Definition” option will open a new editor window with the table definition in the form of a table create statement. Since this is a runnable script, this method provides a convenient way to change the definition of the table if needed (assuming you are not concerned with losing any data within in the table).

The second “Peek” option, will provide you with the same definition, but in this case displayed in the existing editor window as shown in the figure 3-6:

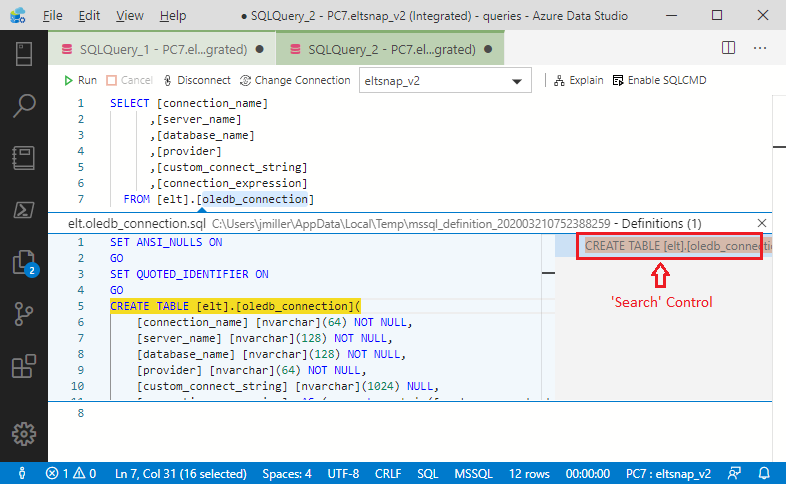


Figure 3-6. Peek ‘Object Definitions’ Option

In the event the table definition includes many columns, you can use the ‘Search Control’ on the right side of the screen to search for a specific column definition.

### Creating a Snippet for Column Definitions

You may be thinking, “This is helpful for retrieving a column definition located a single table”, but what if I want to see how the same column is defined in all tables?”. Good question, and one that could be answered by creating a snippet. A good place to start with snippets is writing the base query, which for our case will use the INFORMATION\_SCHEMA.COLUMNS system view on the msdb ‘’system’ database. In this query, we will be searching for all definitions of the plan\_name column:

select \* from msdb.INFORMATION\_SCHEMA.COLUMNS where COLUMN\_NAME = 'plan\_name'

A subset of the results from running the above query are shown in figure 3-7:

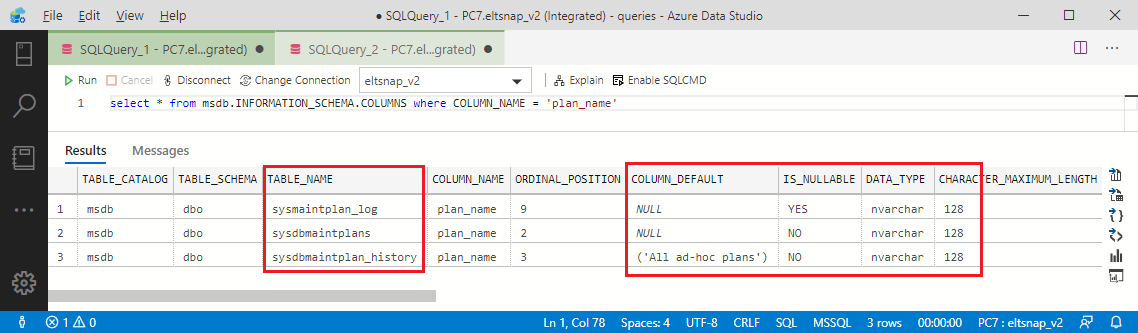


Figure 3-7. Sample Query using Information\_Schema.Columns

The result set above reveals that the column plan\_name is found in 3 tables within the msdb database, has a consistent data type, but varies in terms of nullability and default values, commensurate with how each table uses the column. You could now save this snippet (or ‘template’) as a stand-alone query simply residing in your file system, or convert it to a formal ‘ADS Snippet’. The former could be accessed by using the ‘Menu Bar’ File, Open commands, and the later would be retrieved by keystrokes directly in the SQL editor window. Another consideration with this decision is that a formal ‘ADS Snippet’ can optionally provide variable substitution, which can greatly simplify the re-use of your custom snippet.

The next section will cover how to save your ADS queries and customized snippets.

# Saving Queries and Snippets

When working with multiple ‘file based’ queries, it is helpful to organize related scripts into a common folder structure. To achieve this, you simply select (or optionally create) a folder using the ‘Menu Bar’ File, Open Folder command as shown in figure 3-8. This will establish your ‘current’ folder context:

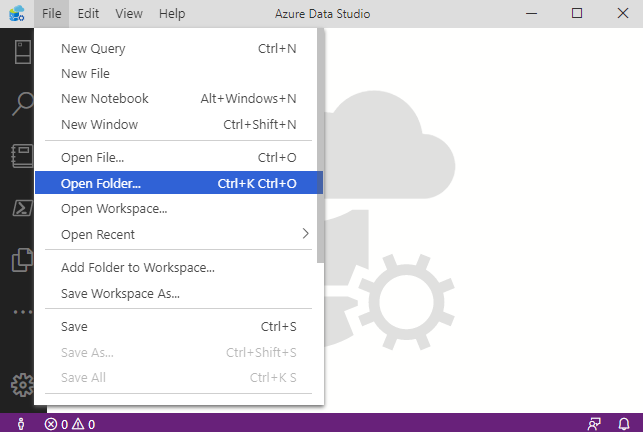


Figure 3-8. File, Open Folder Command

In the case you need to create a New folder, you can still use the ‘Open Folder’ dialog box. This is done by clicking in the ‘white space’ (next to the existing folders) where you will be able to enter a new folder name via a pop-up window. The navigation for this user action is presented in figure 3-9:

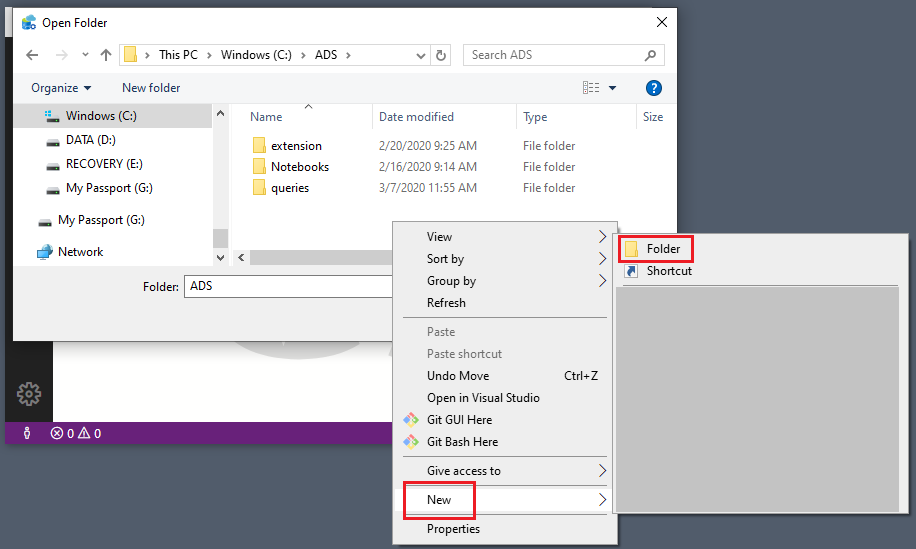


Figure 3-9. Specifying a Folder Name for Queries

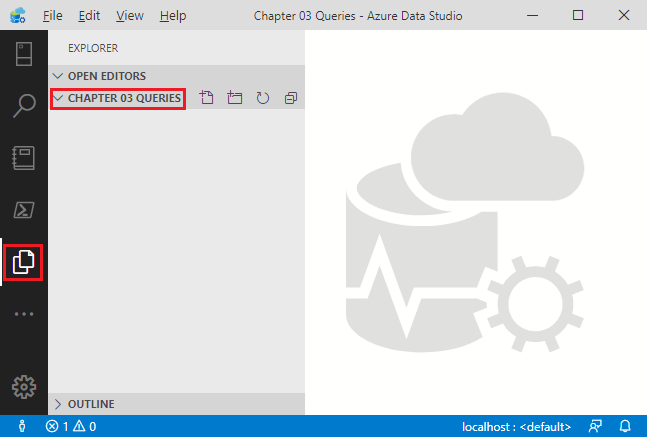
Once you have selected your ‘current’ folder context, queries and scripts that you subsequently save will be placed in this folder by default. The File icon in the ‘Activity Bar’ as shown in Figure 3-10 will provide the name of your current folder context.

Figure 3-10. Current Folder Context

Keep in mind that your working folders could later be tied to GitHub or other source control system. Consequently, your folder organization and naming conventions should be considered. Even if you are not sharing with others, you may find that GitHub is a convenient repository to store your queries and scripts. This is both in terms of safe keeping, as well as accessibility when away from your primary workstation. See Chapter 13 for a ‘Deep Dive’ into GitHub and ADS.

Now that we have a ‘current’ folder, let’s tweak and save our earlier INFORMATION\_SCHEMA.COLUMNS query into the file system. Here is the bit more generic query we’ll use:

select \* from INFORMATION\_SCHEMA.COLUMNS where COLUMN\_NAME = 'column\_name'

As you might have guessed, pressing CTRL + S will open the ‘Save’ dialog box, or you could use File, Save, from the ‘Menu Bar’. In either case you will receive the dialog box shown in figure 3-11 where you can name your file-based query:

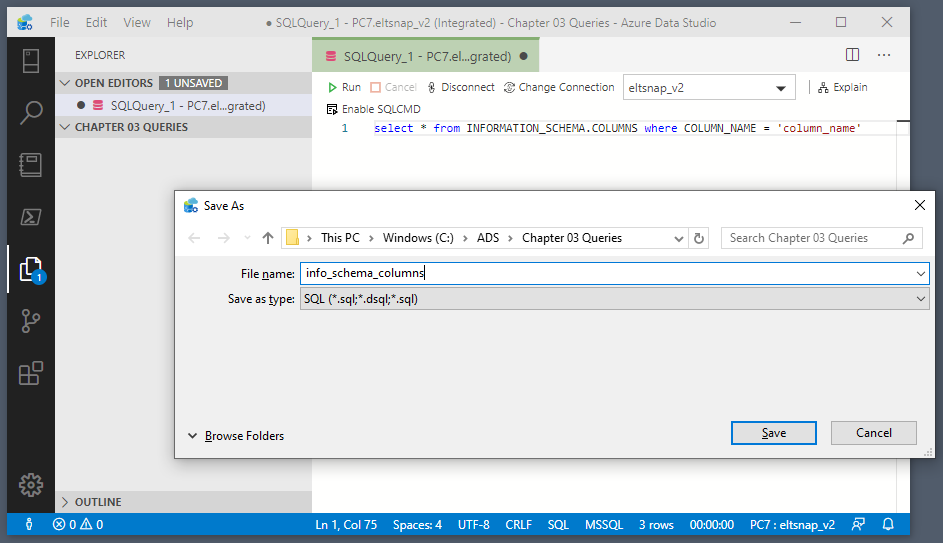


Figure 3-11. Save Query File

Ok, saving a file is admittedly a pretty basic user action. However, what if you would like to save this query as a reusable ADS Snippet? Well for starters we will want to make another tweak to this script which will invoke variable substitution logic on re-use. This is achieved by replacing 'column\_name' with the parameter syntax ${1:TableName}:

select \* from INFORMATION\_SCHEMA.COLUMNS

where COLUMN\_NAME = '${1:ColumnName}'

Note: for simpler snippet coding, we will place this query on a single line in the full json snippet syntax:

{ "Information Schema for Columns": {

"prefix": "InfoSchemaColumns",

"body": "select \* from INFORMATION\_SCHEMA.COLUMNS where COLUMN\_NAME = '${1:ColumnName}'" } }

The first line has the literal "Information Schema for Columns " which is the snippet name. The next line contains the prefix “InfoSchemaColumns” which will cause this snippet to surface based on character matches, which do not necessarily need to be sequential. For example, this snippet would be found by typing ‘infcol’. The third line is the snippet code itself, which will appear upon pop-up selection in your editor window.

Note: we go into much more detail on snippets in Chapter 09.

To save the snippet, press CTRL+SHIFT+P (or from the ‘Menu Bar’ click on View, Command Palette), enter ‘snippet’ in the search box, and select “Preferences: Configure User Snippets” as shown in figure 3-12

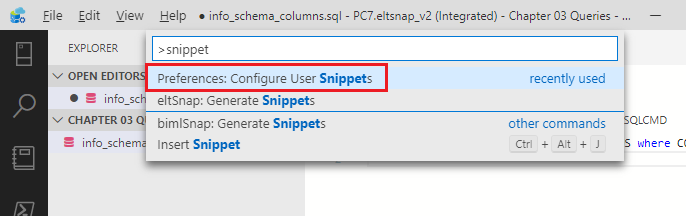


Figure 3-12. Configure User Snippets

Next enter ‘sql’ into the snippet search, and select the file: sql.json as displayed in figure 3-13:

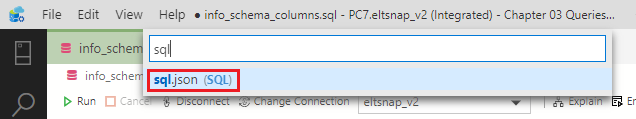


Figure 3-13. sql.json Snippets File

And finally paste in your json script as shown in figure 3-14:

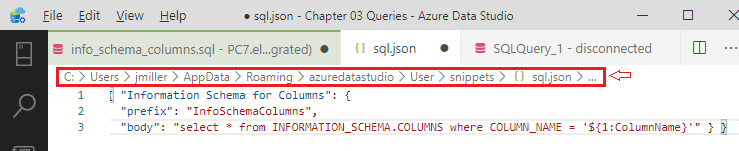


Figure 3-14. A Sample sql.json Snippet

Notice the above window also provides the physical location of the ‘sql.json’ file that you are modifying. Press CTRL+S to save your changes, and enable your new snippet to be used. Press CTRL+N to create a new query window and type the character sequence: ‘infcol’. You should see the snippet pop-up as rendered in figure 3-15:

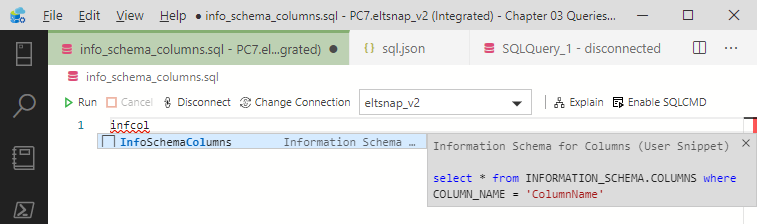


Figure 3-15. Using IntelliSense to Find a Snippet

Selecting this snippet will produce the SQL code in your editor window, with the cursor highlighting the field (in this case ‘ColumnName’) which you will want to replace with the actual column name you are researching. Figure 3-16 displays the snippet which is now readily available with just a few keystrokes:

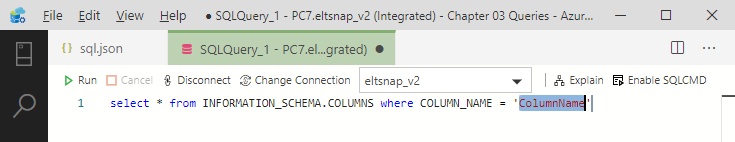


Figure 3-16. Variable Highlighting for a Snippet

As mentioned earlier, we have much more content to come later in the book on ‘ADS Snippets’, which by the way can apply to any of the Azure Data Studio supported languages including PowerShell and Python.

# Top Down View with Minimap

Life at times can be detailed and complex. The same can be true of ‘SQL code’, or for that matter ‘code’ written in any language. Most database developers have written, reviewed, or maintained SQL code that exceeds hundreds, or perhaps even a thousand lines. While ‘big code’ can be intimidating, ADS has a Minimap feature that can at least soften the pain of working with voluminous lines of SQL.

For the following example you can use any larger script. If more lines are needed, just use copy/paste to multiply the SQL code you have since we will not be ‘executing’ this code.

To get started with Minimap, you will first want to turn this feature on via the ‘Command Palette” by pressing CRTL+SHIFT+P, entering ‘minimap’, and selecting “View: Toggle Minimap” as shown in figure 3-17:

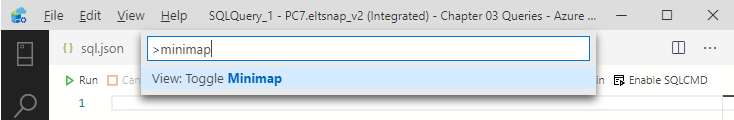


Figure 3-17. Toggle Minimap On

With any script in place, and with Minimap turned on, you will see a birds-eye outline of your code on the right side of the editor window, with your current ‘cursor position’ highlighted as a thin blue line as displayed in figure 3-18:

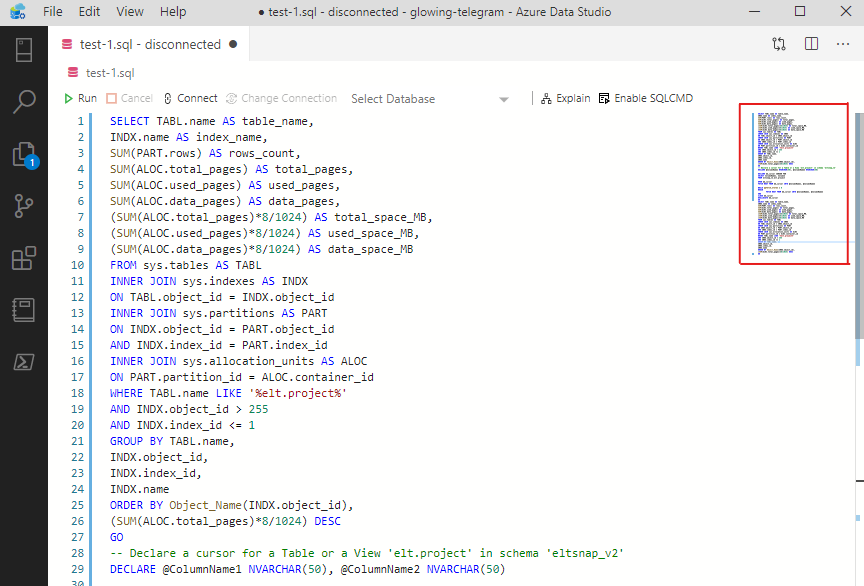


Figure 3-18. SQL Editing with Minimap Outline

Syntax error are reflected in the Minimap with red highlighting as shown in figure 3-19. This is helpful by providing immediate feedback, as well as proximity of the error, even if the actual (readable) SQL code is ‘off screen’.

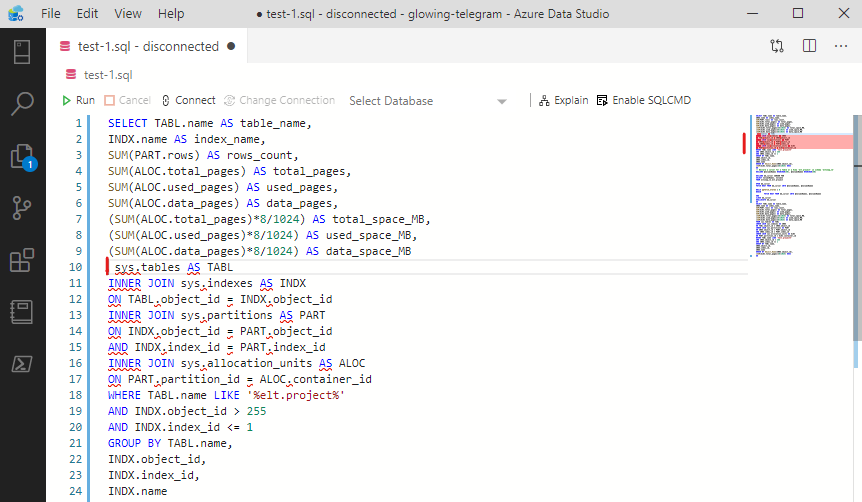


Figure 3-19. Minimap Syntax Errors

Another nice feature of Minimap is the ability to view a large selection of your code, even if it spans hundreds of lines, and requiring excess ‘scrolling’ to visualize. Figure 3-20 demonstrated this capability, albeit on a smaller scale.

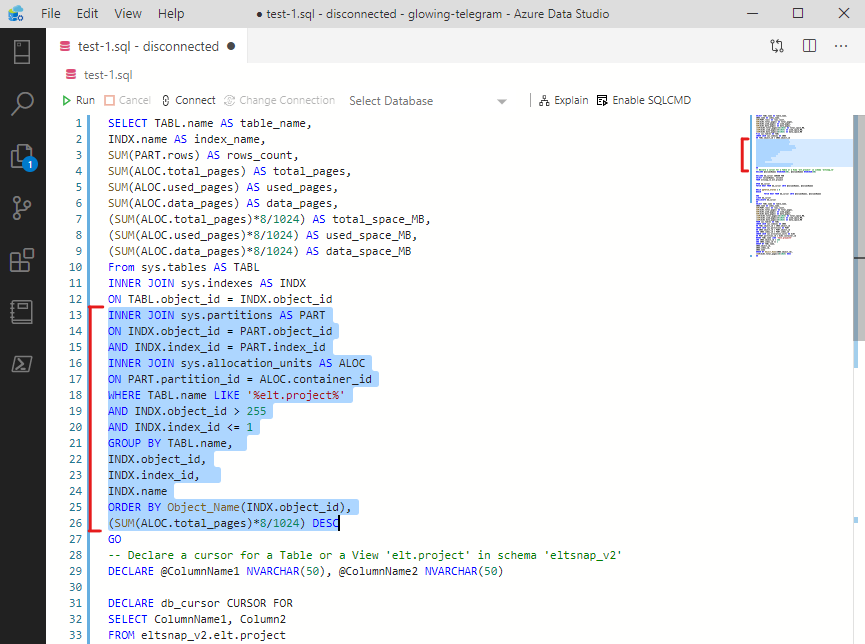


Figure 3-20. Minimap Code Selection

Finally, Minimap provides a convenient way to visualize all locations containing text you are searching for within the entire script. Figure 3-21 shows the result of searching for a table called elt.projects within a SQL script.

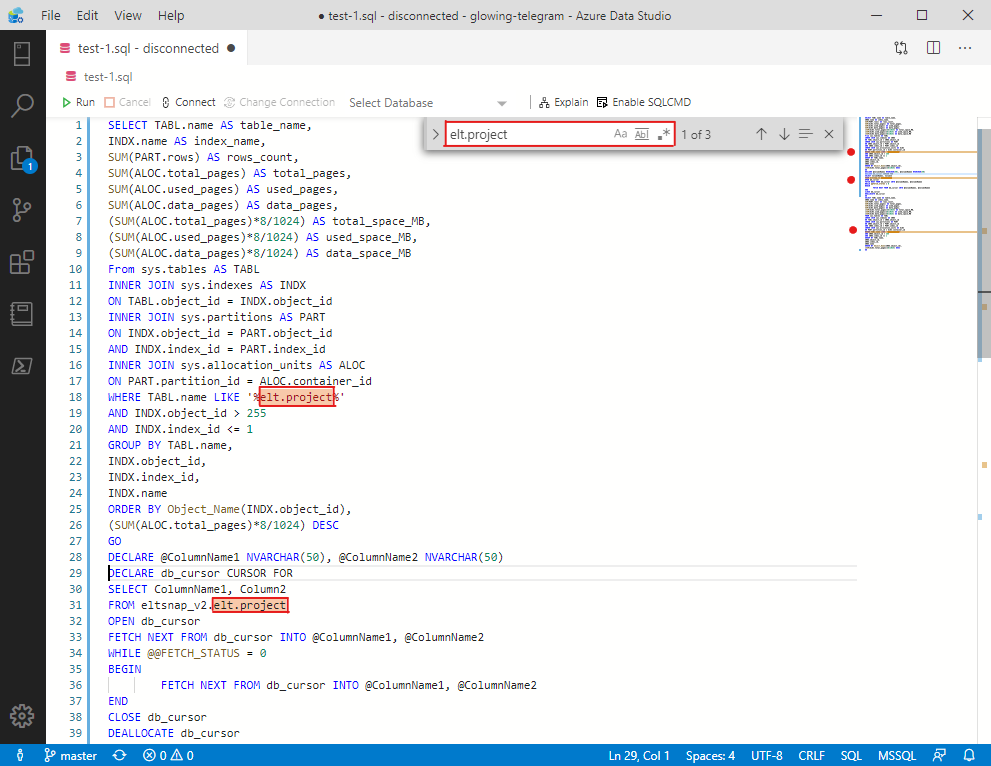


Figure 3-21. Minimap Search Results

So, the next time you find yourself editing a painfully large block of SQL code, remember to ‘toggle on’ Minimap to make your task at least a little more manageable.

# SQL Queries via the Command Terminal

An interesting capability of Azure Data Studio is the integration of a standard SQL editor with other language options. These language options are available by using Notebooks (introduced beginning in Chapter 5), and by using the integrated Terminal, which we’ll introduce next.

To open the Terminal window, press CTRL+` (backtick), or using the ‘Menu bar’, select View, Terminal. Once opened, you’ll see the Terminal pane on the bottom right section of the ADS application as shown in figure 3-22:

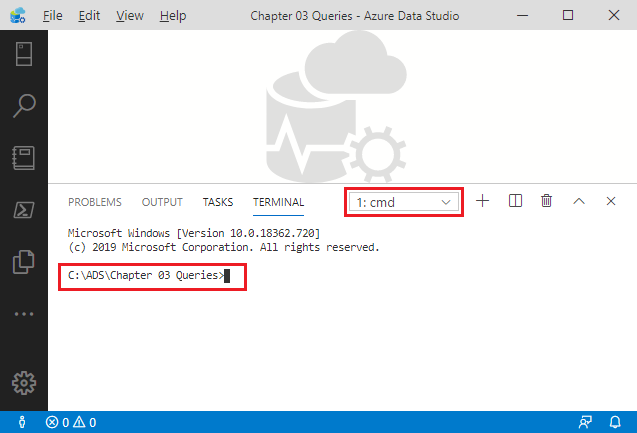


Figure 3-22. ADS Terminal Window

Notice the Terminal opens in the context of the ‘Windows Command Line’ and inherits the Folder context that was previously opened in ADS (in our case it was “Chapter 03 Queries”). Although we could query SQL via the Windows Command Line, we have more options when using PowerShell. To switch the context to PowerShell, you can simply type ‘powershell’ into the Terminal window as displayed in Figure 3-23:

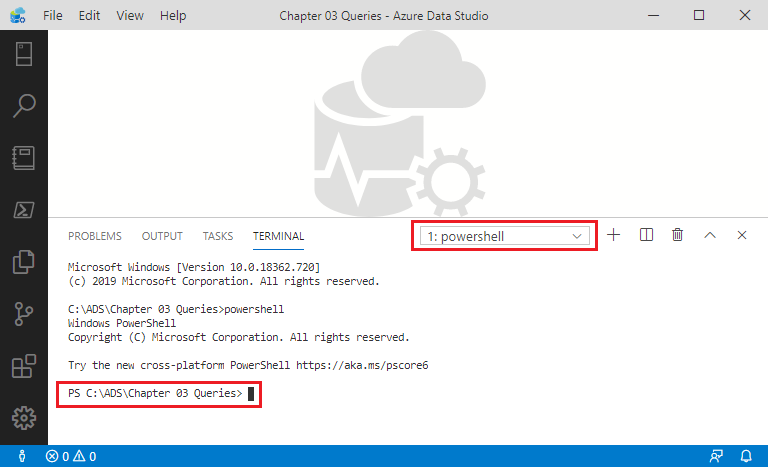


Figure 3-23. PowerShell Terminal Window

Note: In Chapter 15 we will use the new “PowerShell Integrated Console” which is activated by using the ‘Activity Bar’

To query your ‘local instance’ of SQL Server, enter the following command into the PowerShell command prompt

Invoke-Sqlcmd -Query "select \* from INFORMATION\_SCHEMA.TABLES" -ServerInstance "localhost"

And hit enter. You should see the SQL query results shown in figure 3-24:

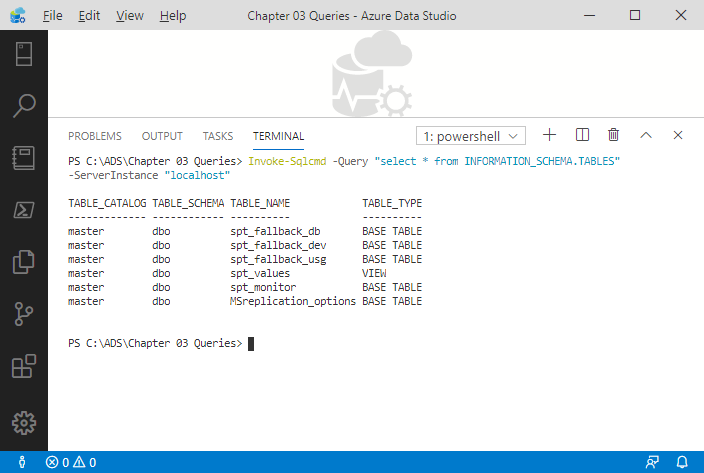


Figure 3-24. Running a SQL Query in PowerShell

Since we are in PowerShell, we have many other options for our result set, such as exporting as a CSV file. This can be achieved by running the following PowerShell script:

Invoke-Sqlcmd -Query "select \* from INFORMATION\_SCHEMA.TABLES" -ServerInstance "localhost" | export-csv -Delimiter ',' -Path "tables.csv" -NoTypeInformation

After running in the Terminal, the results of the above script are ‘piped’ to the file called ‘tables.csv’, instead of being returned into the terminal session. To see the file, click on the ‘Explorer’ Icon in the ‘Activity Bar’. The file should now be available in the ‘Side Bar’ as shown in figure 3-25:

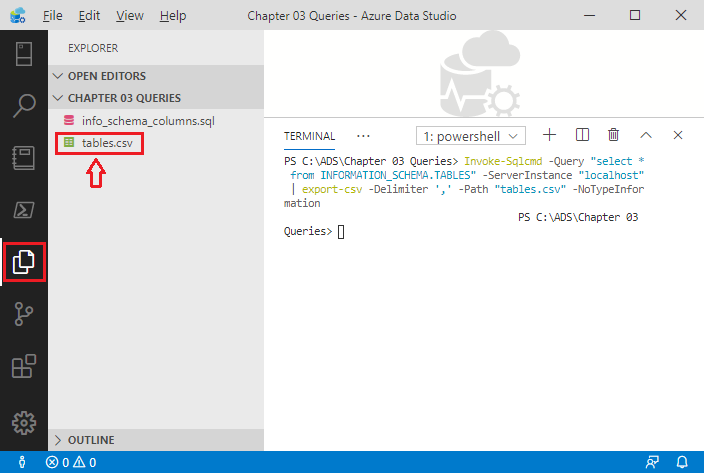


Figure 3-24. Exporting a Query using PowerShell

This is a simple but instructive example of the synthesis of traditional SQL queries, with the extended language support built into Azure Data Studio. This illustration leveraged the ADS Terminal window but read on as deeper and even surprising capabilities are ahead!