|  |
| --- |
|  |
| Independent Study |
| **Visualizing live system’s performance using Power BI, SQL & Python** |
|  |
| **Jhansi Lakshmi Kammili(6370377)** |
| **3/9/2024** |

|  |
| --- |
|  |

Guided by Prof. Kiavash

# Visualizing live system’s performance using Power BI, SQL & Python

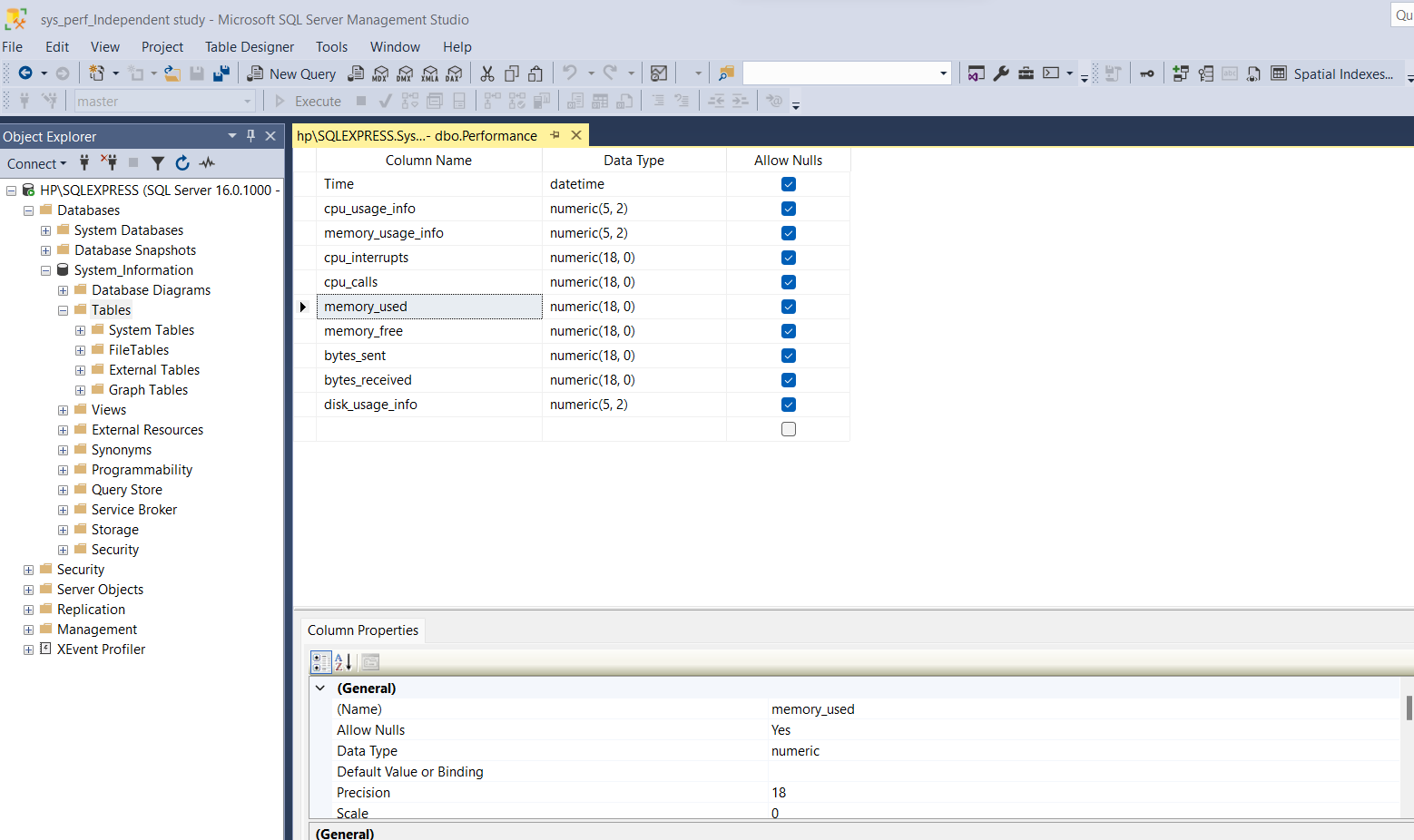
## Step 1:

Installation of SSMS

## Step 2:

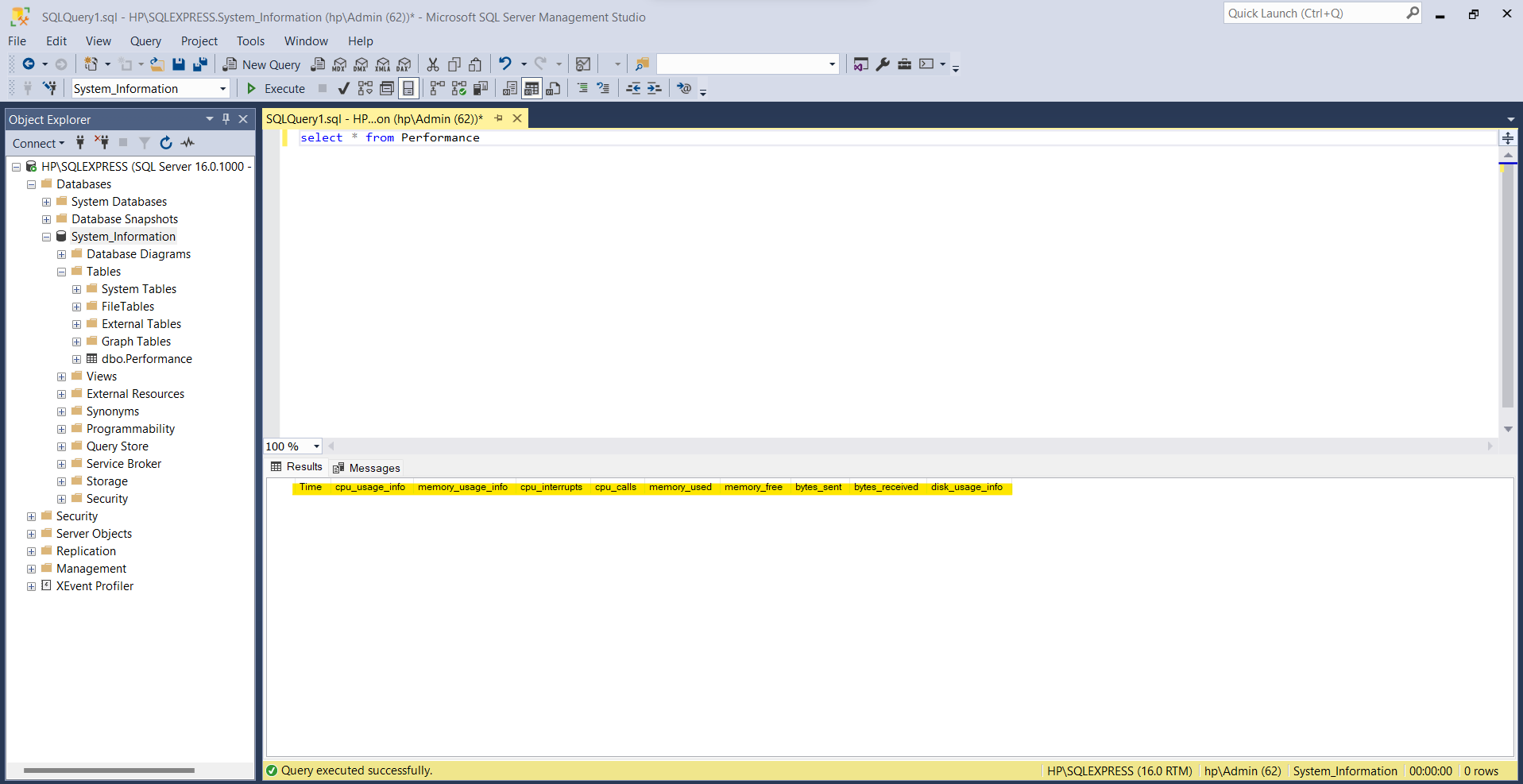
Created a table ‘Performance’

Add various columns to store CPU usage information, memory usage information, CPU interrupts, CPU calls, memory used, memory free, bytes sent over the Internet, bytes received from the Internet, and disk usage information.



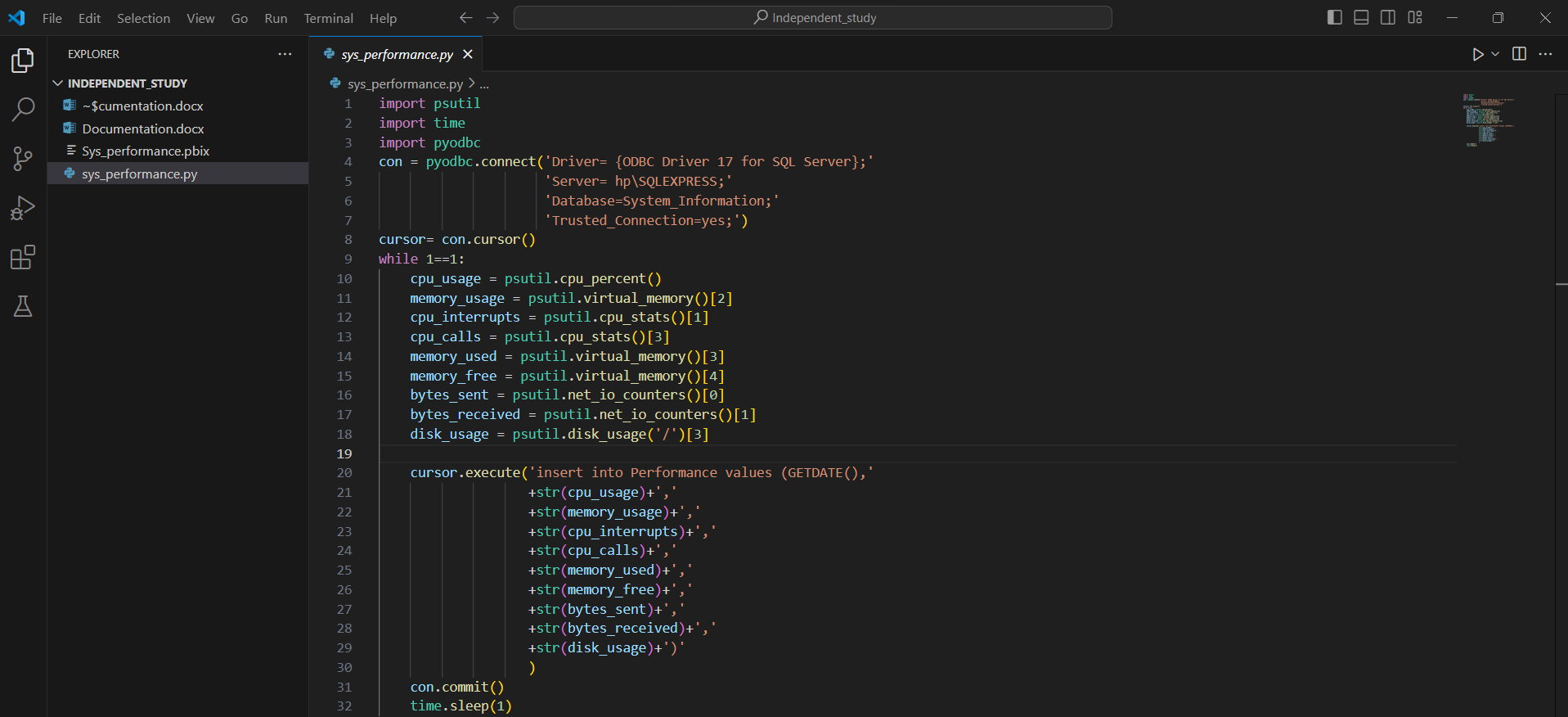
Verified if the table is created or not by using below query.

Select \* from Performance



## Step 3:

As now table is created and I feed the values for each column using python.



This Python script continuously monitors system performance metrics such as CPU usage, memory usage, network traffic, and disk usage using the `psutil` library, and inserts these metrics into a SQL Server database named `System\_Information`.

Here's a breakdown of the code:

1. Importing Required Libraries:

- `psutil`: A cross-platform library for retrieving information on running processes and system utilization (CPU, memory, disks, network, sensors).

- `time`: Provides various time-related functions.

- `pyodbc`: A Python library for connecting to databases via ODBC (Open Database Connectivity).

2. Establishing Database Connection:

- It connects to a SQL Server database named `System\_Information` running on the server `hp\SQLEXPRESS`. The connection is made using Windows Authentication (Trusted\_Connection=yes).

3. Setting up Cursor:

- It creates a cursor object for executing SQL queries on the database.

4. Continuous Monitoring Loop:

- The script enters an infinite loop (`while 1==1`) to continuously monitor system performance.

- Inside the loop, it retrieves various performance metrics using `psutil` functions.

- Metrics collected include CPU usage, memory usage, CPU interrupts, CPU calls, memory used, memory free, bytes sent/received over the network, and disk usage.

- Each metric is retrieved using specific `psutil` functions.

- The script then inserts these metrics into a table named `Performance` in the SQL Server database using an SQL `INSERT` statement.

- `GETDATE()` function is used to get the current timestamp for each record.

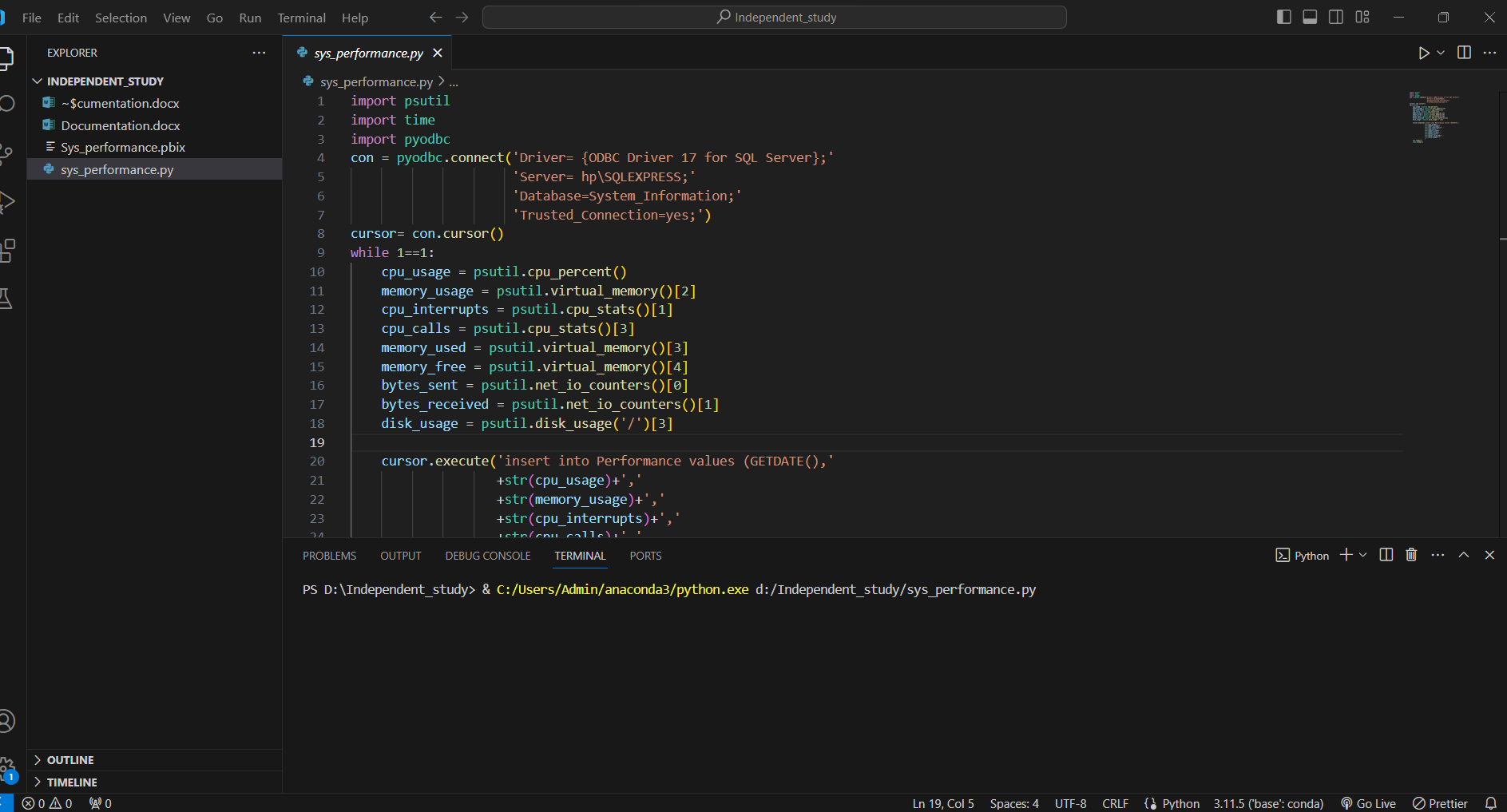
- After executing the `INSERT` statement, the changes are committed to the database using `con.commit()`.

- The loop pauses execution for 1 second using `time.sleep(1)` to control the frequency of data collection.

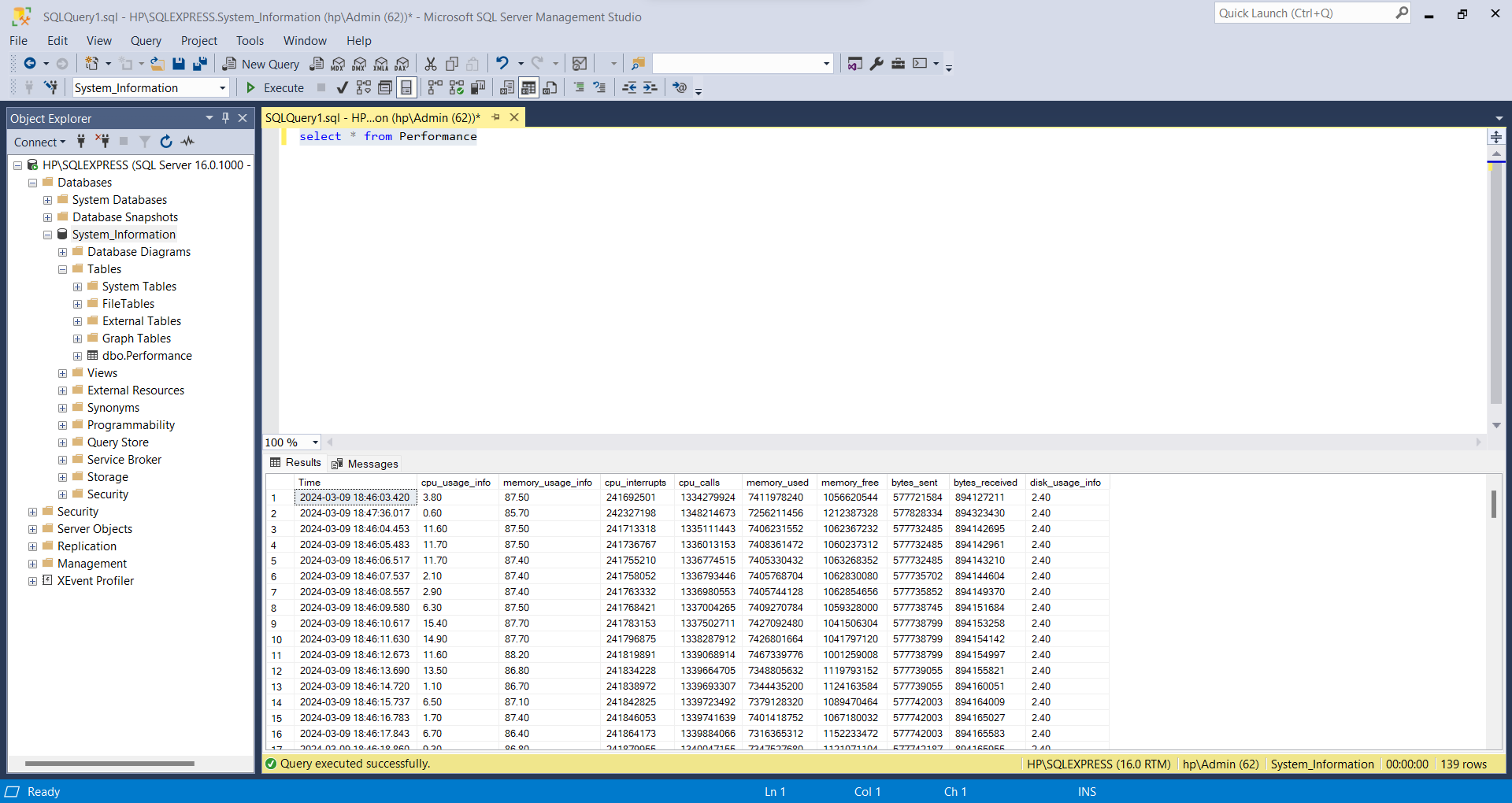
5. Closing Database Connection:

- Since there's no explicit closure of the database connection in the code, its good practice to close the connection after the loop ends, although in this script, it's not shown.

I’ve executed the current python file and is collecting the data from the system and feeding these values to the table that I’ve created earlier.



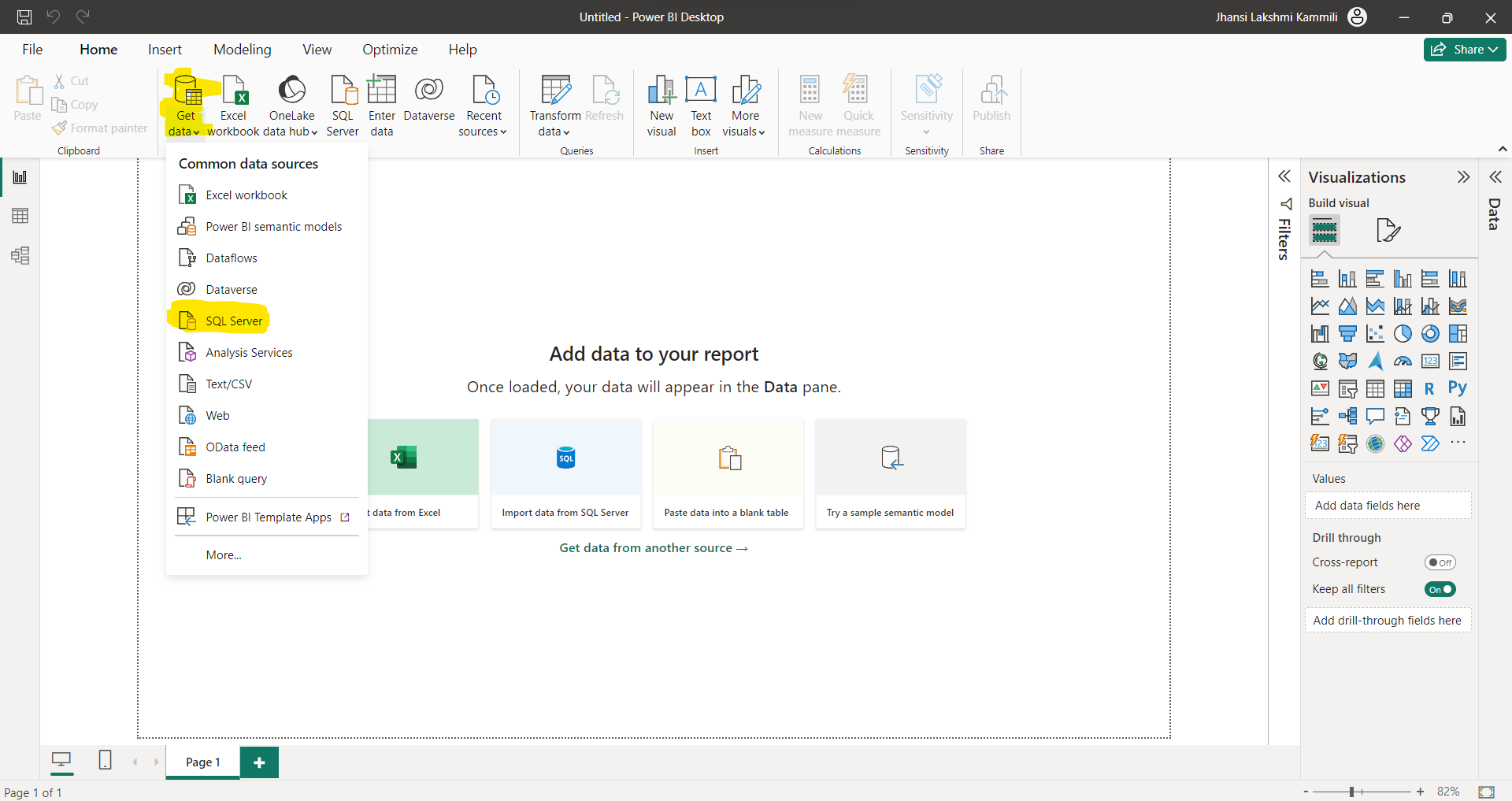
To test if the values are getting stored in the ‘Performance’ table I’ve execute select query.



## Step 4:

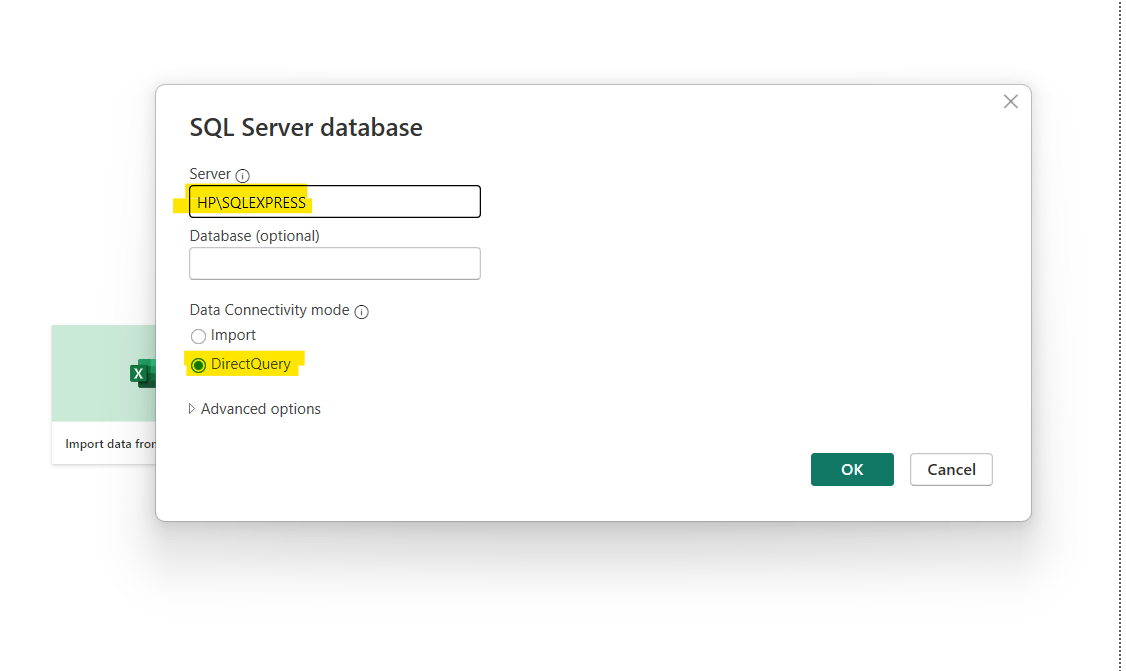
Connecting Power BI to SQL Server

Installed Power BI



To get the data from SQL server

Get data>SQL server



Enter the server details and I’ve opted Data connectivity mode as Direct Query.

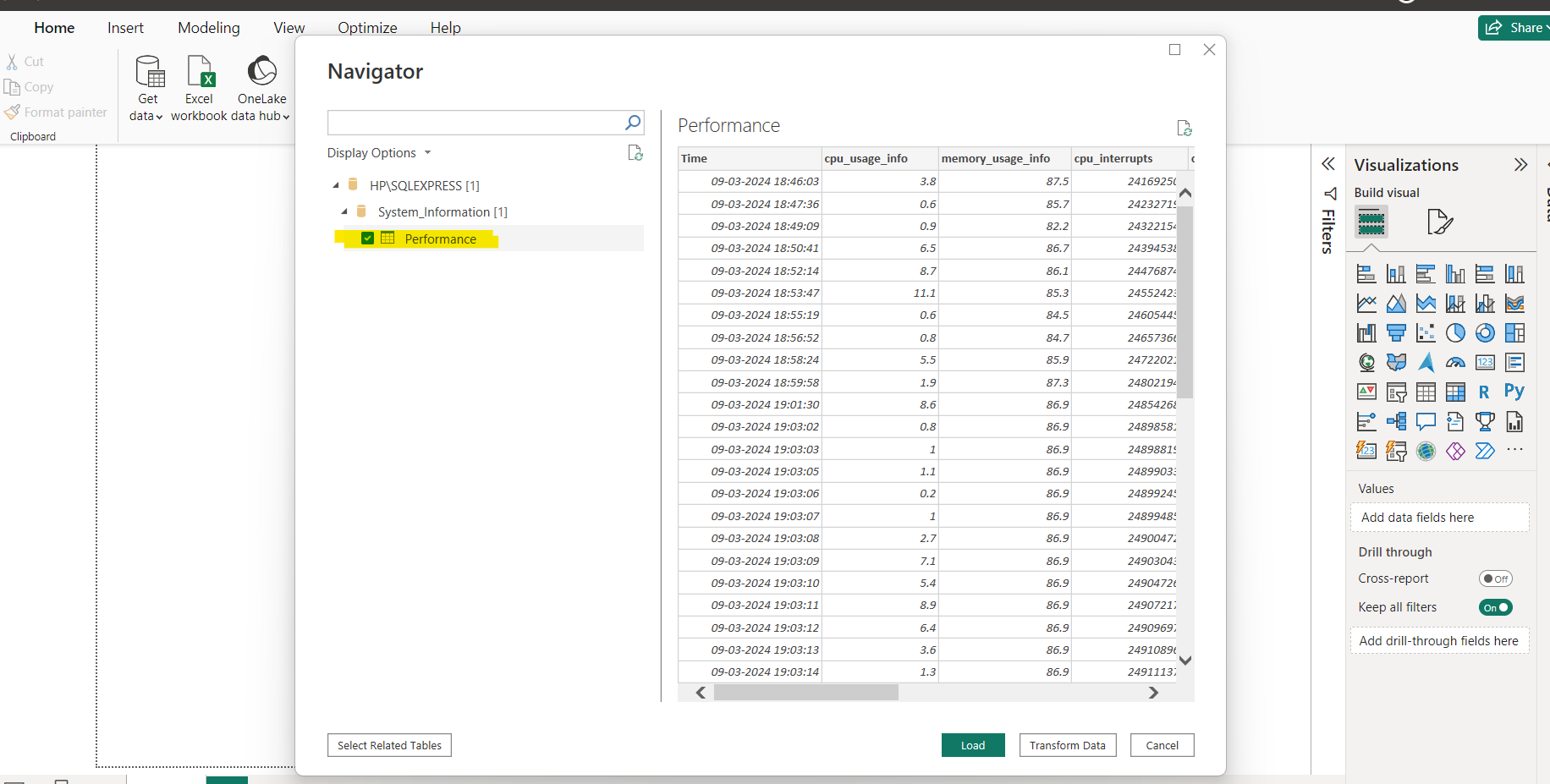
When to use Direct Query?

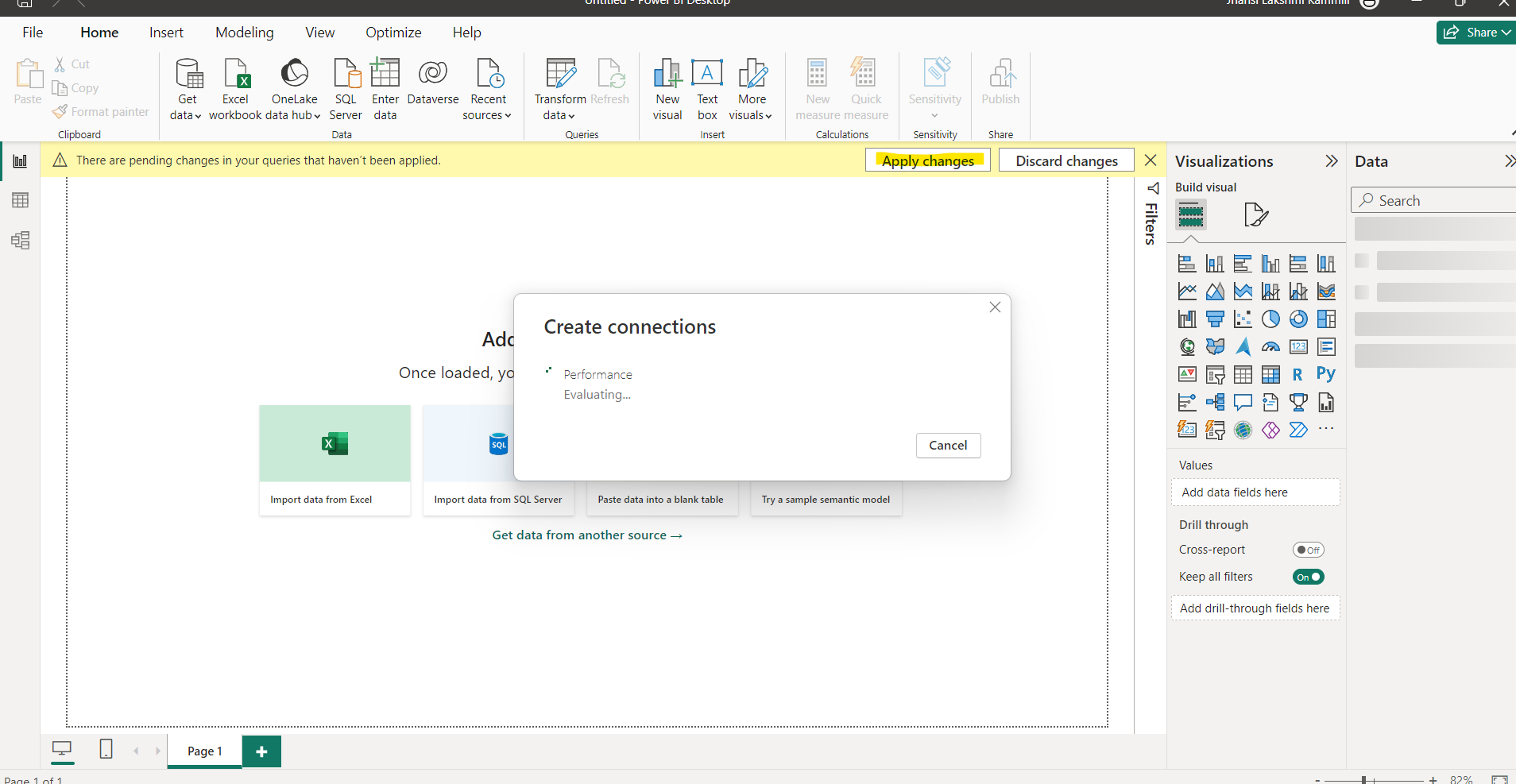
* Use direct query when dealing with large datasets or real-time data sources.
* Ideal for scenarios where you want to analyze the most up-to-date data without storing it locally.
* Suited for databases or data sources that support efficient querying directly, such as SQL Server, Azure SQL Database, Analysis Services, etc.
* Provides real-time or near real-time data analysis capabilities.

When to use Import?

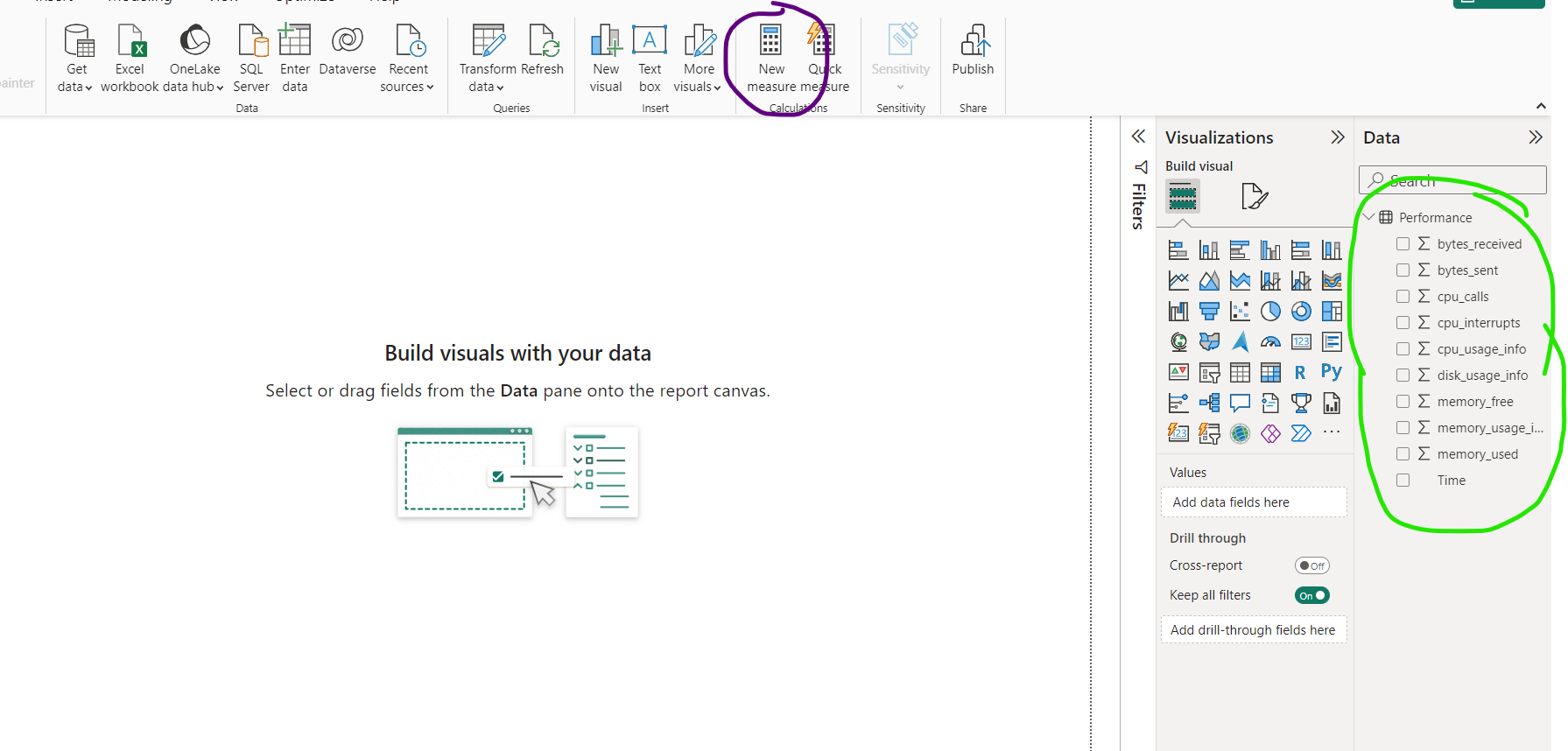
* Use import when dealing with moderately sized datasets or when you need to perform extensive transformations and calculations within Power BI.
* Suitable for scenarios where data is relatively static or where periodic data refreshes are acceptable.
* Provides faster query performance as data is loaded into Power BI's in-memory engine (VertiPaq).
* Offers more flexibility in terms of data modeling, transformations, and custom calculations within Power BI.

Load the Performance table that I’ve have created. And once I click on load, I could access the data from the SQL server and create a dashboard.





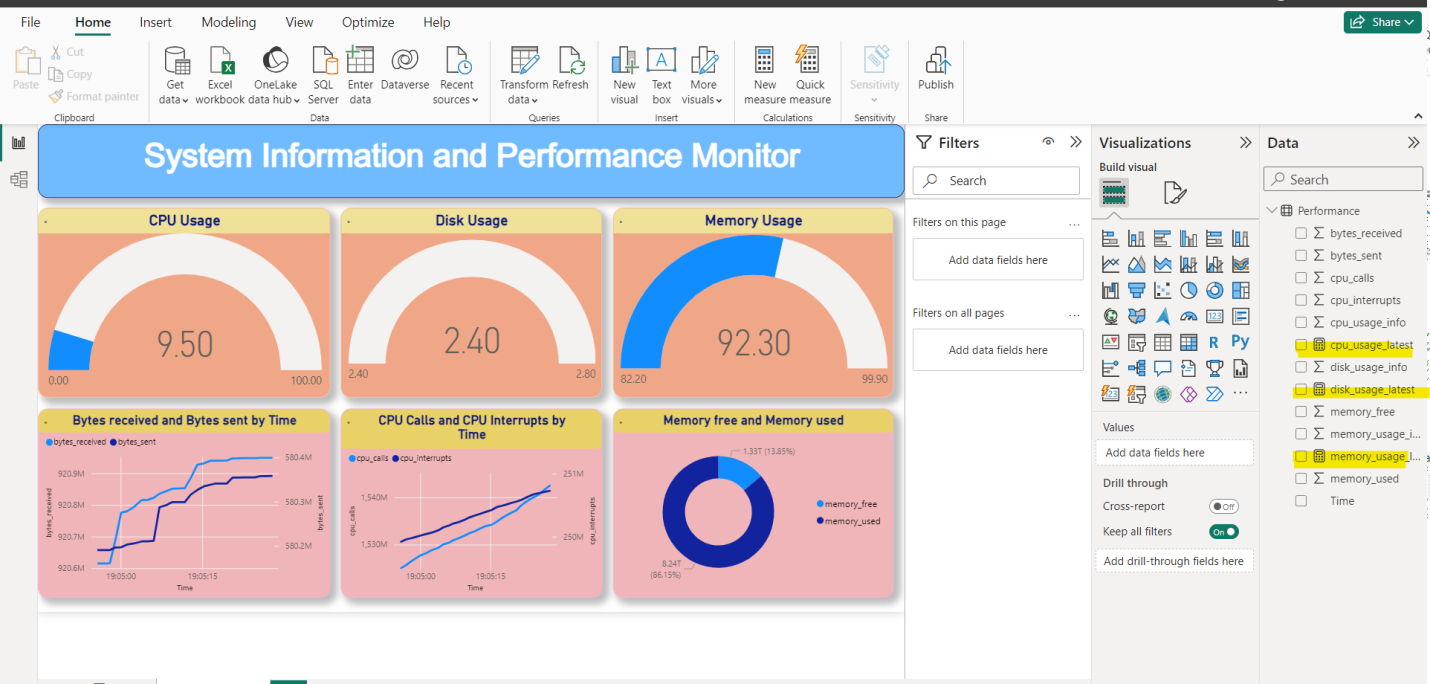
After loading could see the Performance table in the data pane.



I’ve created few measures using DAX queries.

1. cpu\_usage\_latest = CALCULATE(SUM(Performance[cpu\_usage\_info]),Performance[Time]=MAX(Performance[Time]))
   * This query calculates the latest CPU usage from the `Performance` table.
   * `CALCULATE` is a DAX function used to evaluate an expression in a modified filter context. It's often used to apply filters or conditions to a calculation.
   * `SUM(Performance[cpu\_usage\_info])` calculates the sum of the `cpu\_usage\_info` column from the `Performance` table. This column presumably stores CPU usage data.
   * `Performance[Time]=MAX(Performance[Time])` is the filter condition applied to the calculation. It filters the rows where the `Time` column is equal to the maximum value of the `Time` column in the entire table. Essentially, this retrieves the row with the latest timestamp, and then calculates the sum of CPU usage for that specific timestamp.
2. disk\_usage\_latest = CALCULATE(SUM(Performance[disk\_usage\_info]),Performance[Time]=MAX(Performance[Time]))
   * Similar to the first query, this one calculates the latest disk usage from the `Performance` table.
   * `SUM(Performance[disk\_usage\_info])` calculates the sum of the `disk\_usage\_info` column from the `Performance` table. This column stores disk usage data.
   * `Performance[Time]=MAX(Performance[Time])` applies the filter condition to select the row with the latest timestamp, similar to the first query.
3. memory\_usage\_latest = CALCULATE(SUM(Performance[memory\_usage\_info]),Performance[Time]=MAX(Performance[Time]))
   * This query calculates the latest memory usage from the `Performance` table.
   * `SUM(Performance[memory\_usage\_info])` calculates the sum of the `memory\_usage\_info` column from the `Performance` table. This column probably contains memory usage data.
   * `Performance[Time]=MAX(Performance[Time])` filters the rows to select the one with the latest timestamp.

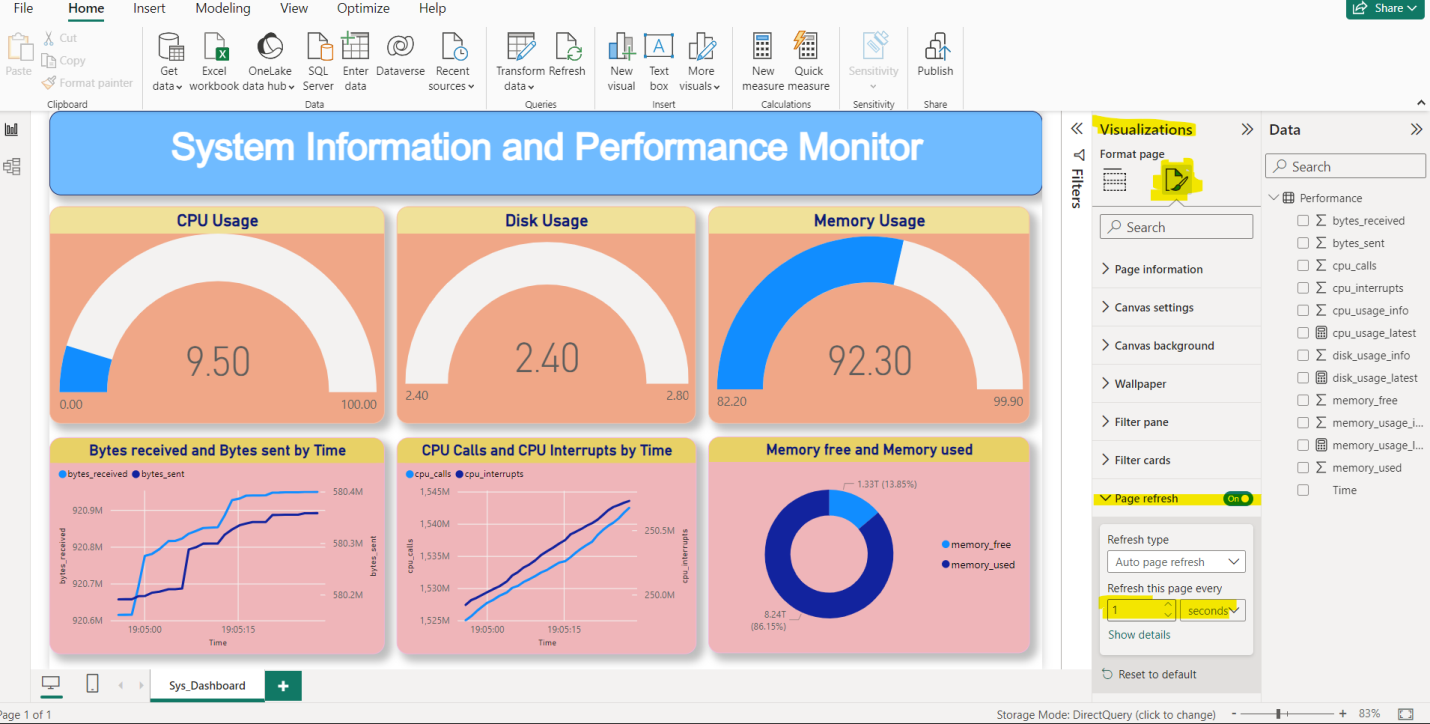
After calculating measures, I’ve built a dashboard.



* **CPU Usage:** Represented by a semi-circle gauge chart, showing the current CPU usage at 9.50.
* **Disk Usage:** Another semi-circle gauge chart displays the disk usage, which is currently at 2.40.
* **Memory Usage:** Similarly, memory usage is depicted using a semi-circle gauge chart, indicating 92.30% utilization.
* **Bytes Received and Bytes Sent by Time:** A line graph illustrates the trend of bytes received (blue line) and bytes sent (purple line) over time.
* **CPU Calls and CPU Interrupts by Time:** Another line graph shows the trends of CPU calls (blue line) and interrupts (purple line) over time.
* **Memory Free and Memory Used:** A donut chart visually represents the proportion of free memory (blue segment) to used memory (pink segment)

But to show the real-time data in the visualisation I set the page refresh option enabled. It allows keeping reports up-to-date by automatically fetching the latest data from the source.

* Select the report page for which I want to enable automatic page refresh.
* In the Visualizations pane, click the Formatting button (resembling a paint roller).
* Find the “Page refresh” section near the bottom of the pane.
* Turn page refresh on or off based on your preference.
* Depending on your needs, choose between two refresh types:
  + Fixed Interval: Update visuals at a constant interval (e.g., every second or five minutes).
  + Change Detection: Refresh visuals based on data changes rather than a specific interval.





**Check this video for clear understanding**.