Independent Study Report

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

Jhansi Lakshmi Kammili(6370377)

Guided by
Dr. Kiavash Bahreini

INTRODUCTION

In this independent study, the objective was to develop a comprehensive system for real-time monitoring of system performance metrics using a combination of Power BI for visualization, SQL Server for data storage, and Python for data collection. By integrating these technologies, the aim was to create an efficient solution for monitoring and analyzing system performance metrics in real-time.

SQL Server Integration Services (SSIS):

SQL Server Integration Services (SSIS) is a component of the Microsoft SQL Server database software that is used to perform a wide range of data integration and transformation tasks. SSIS enables users to create, manage, and deploy workflows for extracting, transforming, and loading (ETL) data from various sources into SQL Server databases or other destinations. It provides a graphical interface for designing data integration workflows, making it easier to handle complex data integration scenarios.

SQL Server:

SQL Server is a relational database management system (RDBMS) developed by Microsoft. It provides a secure, reliable, and scalable platform for storing and managing structured data. SQL Server supports various features such as data storage, data retrieval, data manipulation, and data analysis. It also offers advanced capabilities for business intelligence, reporting, and analytics. SQL Server is widely used by organizations of all sizes for managing their data and running mission-critical applications.

Power BI:

Power BI is a business analytics tool developed by Microsoft that enables users to visualize and analyze data from various sources in interactive dashboards and reports. It provides a suite of tools for data preparation, data modeling, data visualization, and data sharing. Power BI allows users to connect to different data sources, including databases, files, and online services, to create unified views of their data. It supports self-service BI capabilities, enabling users to explore data, gain insights, and make data-driven decisions. Power BI is widely used across industries for business intelligence, reporting, and data visualization purposes.

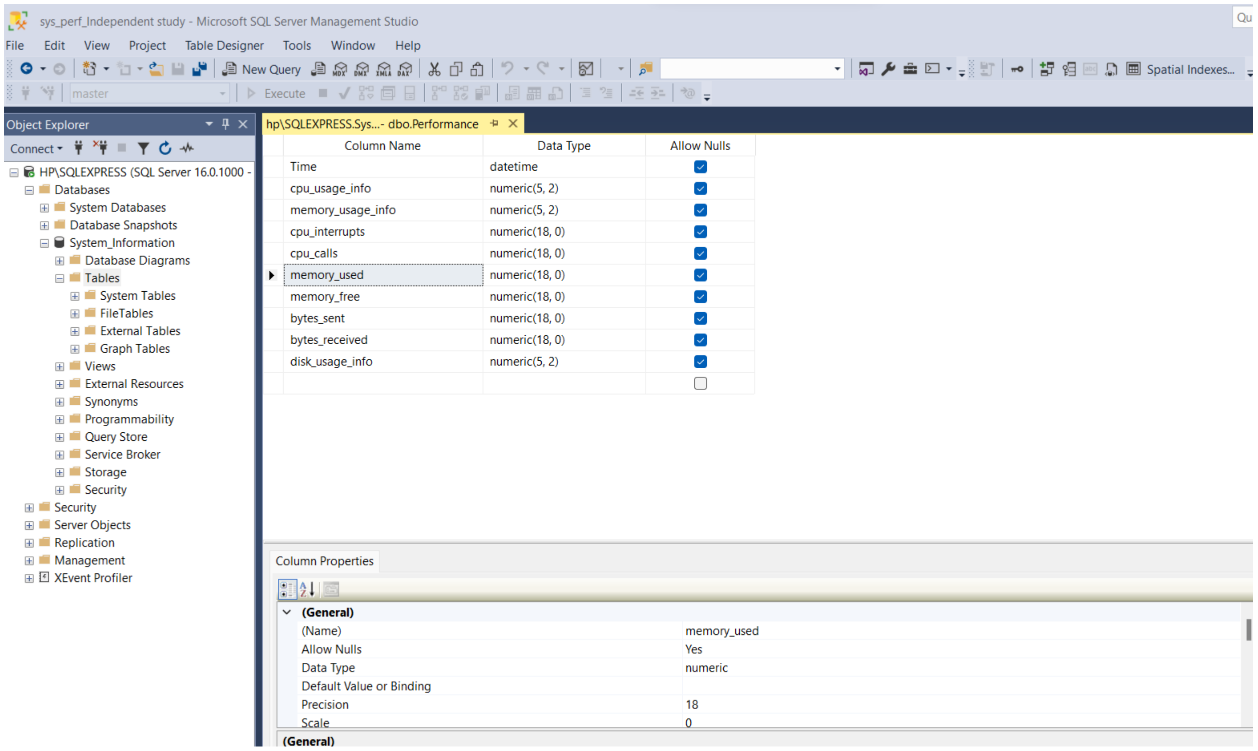
Methodology

Step 1: Installation of SQL Server Management Studio (SSMS)

SQL Server Management Studio (SSMS) was installed to provide a graphical interface for managing SQL Server databases. This step ensured a convenient environment for database administration and query execution.

Step 2: Creation of Performance Table in SQL Server

A table named 'Performance' was created within the SQL Server database. This table served as the repository for storing various system performance metrics such as CPU usage, memory usage, network traffic, and disk usage. Each metric was represented by a column in the table, allowing for organized data storage and retrieval.
Added 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.



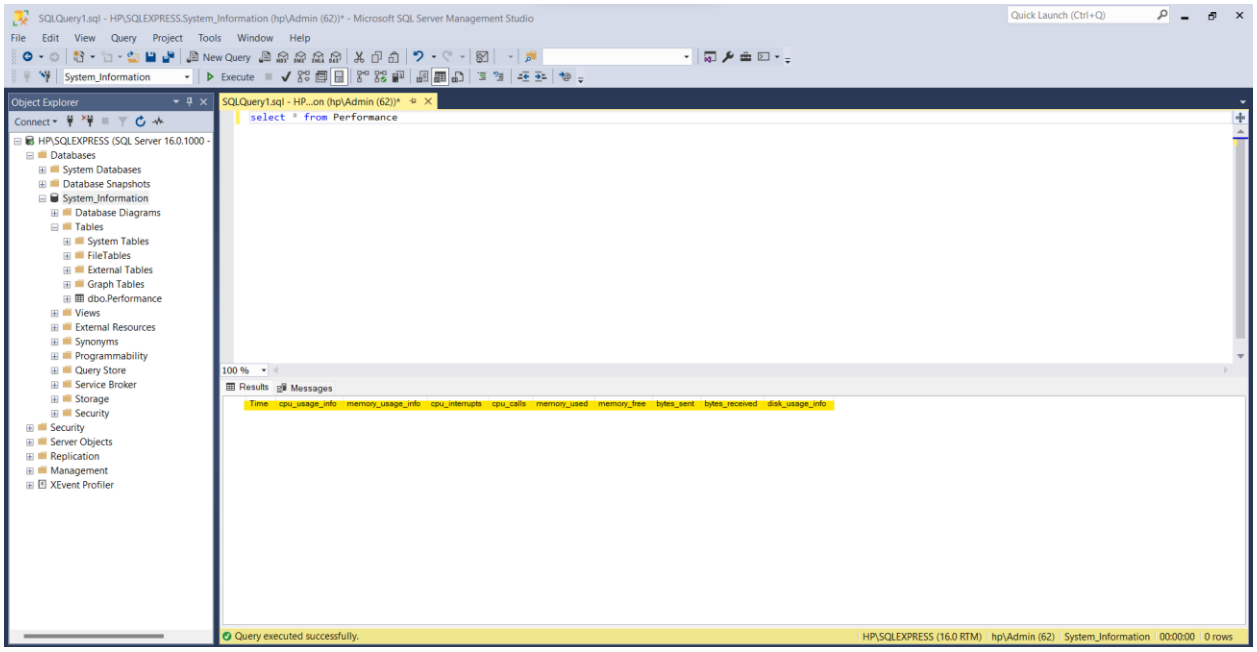
Column Name	Data Type	Allow Nulls
Time	datetime	✓
cpu_usage_info	numeric(5, 2)	✓
memory_usage_info	numeric(5, 2)	✓
cpu_interrupts	numeric(18, 0)	✓
cpu_calls	numeric(18, 0)	✓
memory_used	numeric(18, 0)	✓
memory_free	numeric(18, 0)	✓
bytes_sent	numeric(18, 0)	✓
bytes_received	numeric(18, 0)	✓
disk_usage_info	numeric(5, 2)	✓
		☐

(General)

(Name)	memory_used
Allow Nulls	Yes
Data Type	numeric
Default Value or Binding	
Precision	18
Scale	0

(General)

Verified if the table is created or not by using below query.
*Select * from Performance*



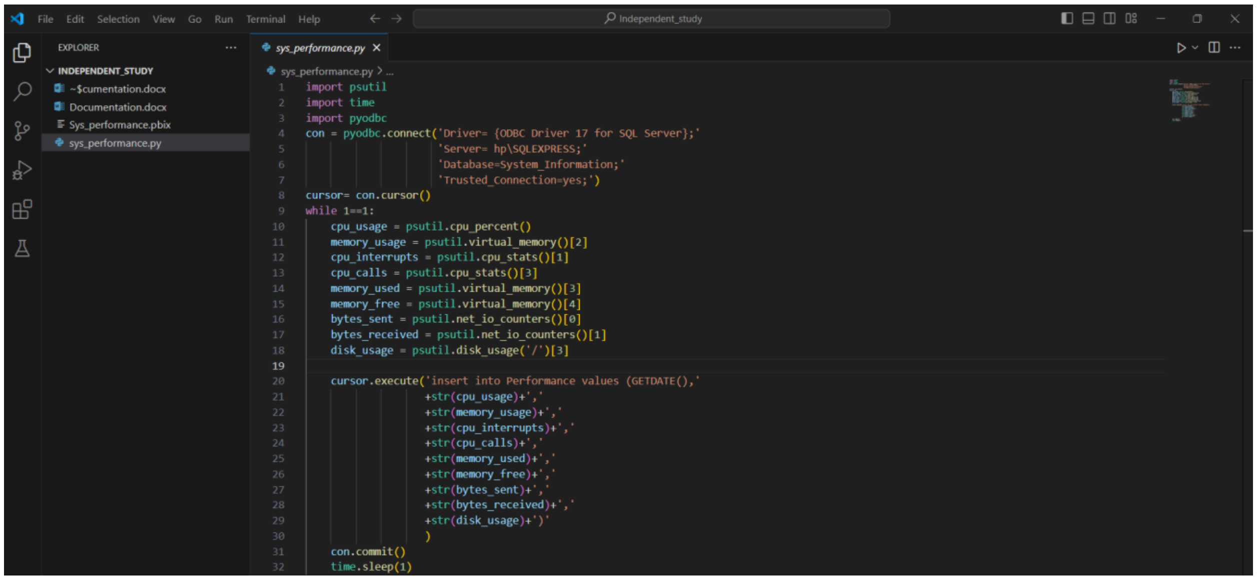
```sql
select * from Performance
```

```
Time  cpu_usage_info  memory_usage_info  cpu_interrupts  cpu_calls  memory_used  memory_free  bytes_sent  bytes_received  disk_usage_info
```

```
Query executed successfully.
```

Step 3: Python Script for Real-time Data Collection

A Python script was developed to collect real-time system performance metrics using the psutil library. This script continuously monitored the system and retrieved metrics such as CPU usage, memory usage, network traffic, and disk usage. The collected data was then inserted into the 'Performance' table in the SQL Server database, ensuring that the data was stored in a structured format for further analysis.



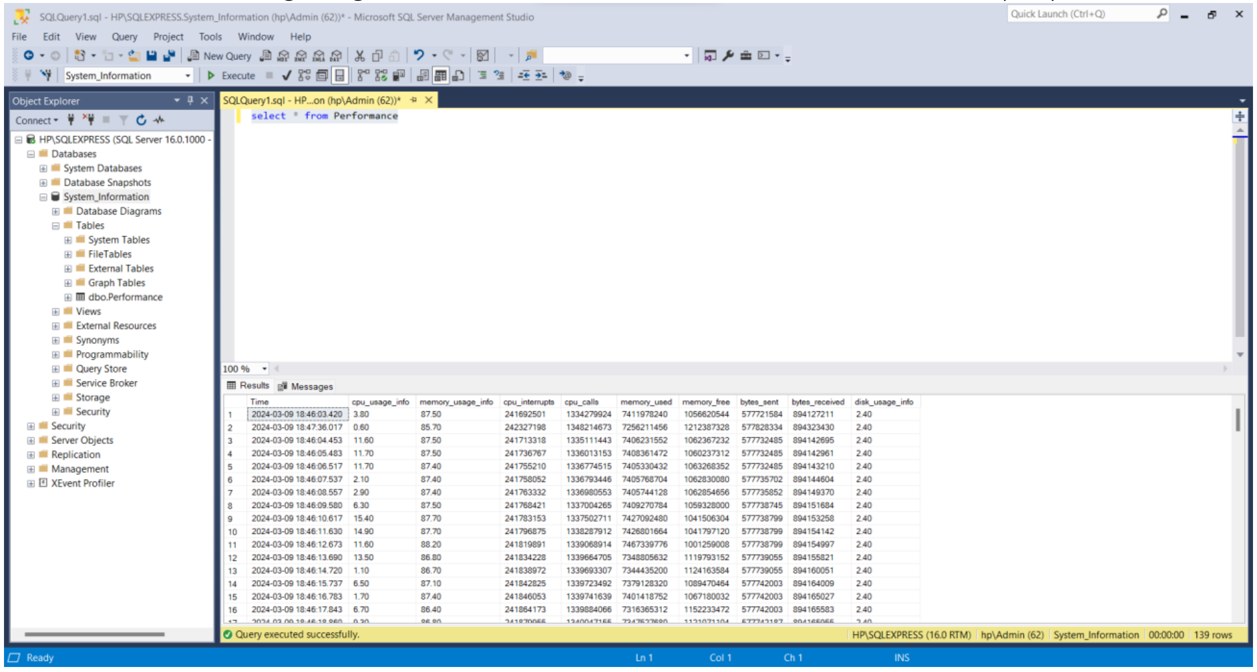
```python
import psutil
import time
import pyodbc
con = pyodbc.connect('Driver= {ODBC Driver 17 for SQL Server};'
                     'Server= hp\SQLEXPRESS;'
                     'Database=System_Information;'
                     'Trusted_Connection=yes;')
cursor= con.cursor()
while 1==1:
    cpu_usage = psutil.cpu_percent()
    memory_usage = psutil.virtual_memory()[2]
    cpu_interrupts = psutil.cpu_stats()[1]
    cpu_calls = psutil.cpu_stats()[3]
    memory_used = psutil.virtual_memory()[3]
    memory_free = psutil.virtual_memory()[4]
    bytes_sent = psutil.net_io_counters()[0]
    bytes_received = psutil.net_io_counters()[1]
    disk_usage = psutil.disk_usage('/')[3]

    cursor.execute('insert into Performance values (GETDATE(),'
                   +str(cpu_usage)+','
                   +str(memory_usage)+','
                   +str(cpu_interrupts)+','
                   +str(cpu_calls)+','
                   +str(memory_used)+','
                   +str(memory_free)+','
                   +str(bytes_sent)+','
                   +str(bytes_received)+','
                   +str(disk_usage)+')'
                   )
    con.commit()
    time.sleep(1)
```

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.



```
select * from Performance
```

	Time	cpu_usage_info	memory_usage_info	cpu_interrupts	cpu_calls	memory_used	memory_free	bytes_sent	bytes_received	disk_usage_info
1	2024-03-09 18:46:03.420	3.80	87.50	241692501	1334279924	7411978240	1056620544	577721584	894127211	2.40
2	2024-03-09 18:47:36.017	0.60	85.70	242327198	1348214673	7256211456	1212387328	577828334	894323430	2.40
3	2024-03-09 18:46:04.453	11.60	87.50	241713318	1335111443	7406231552	1062367232	577732485	894142695	2.40
4	2024-03-09 18:46:05.483	11.70	87.50	241736767	1336013153	7408361472	1060237312	577732485	894142961	2.40
5	2024-03-09 18:46:06.517	11.70	87.40	241755210	1336774515	7405330432	1063268352	577732485	894143210	2.40
6	2024-03-09 18:46:07.537	2.10	87.40	241758052	1336793446	7405768704	1062830080	577735702	894144604	2.40
7	2024-03-09 18:46:08.557	2.90	87.40	241763332	1336980553	7405744128	1062854656	577735852	894149370	2.40
8	2024-03-09 18:46:09.580	6.30	87.50	241768421	1337004265	7409270784	1059328000	577738745	894151684	2.40
9	2024-03-09 18:46:10.617	15.40	87.70	241783153	1337502711	7427092480	1041506304	577738799	894153258	2.40
10	2024-03-09 18:46:11.630	14.90	87.70	241796875	1338287912	7426801664	1041797120	577738799	894154142	2.40
11	2024-03-09 18:46:12.673	11.60	88.20	241819891	1339068914	7467339776	1001259008	577738799	894154997	2.40
12	2024-03-09 18:46:13.690	13.50	86.80	241834228	1339664705	7348805632	1119793152	577739055	894155821	2.40
13	2024-03-09 18:46:14.720	1.10	86.70	241838972	1339693307	7344435200	1124163584	577739055	894160051	2.40
14	2024-03-09 18:46:15.737	6.50	87.10	241842825	1339723492	7379128320	1089470464	577742003	894164009	2.40
15	2024-03-09 18:46:16.783	1.70	87.40	241846053	1339741639	7401418752	1067180032	577742003	894165027	2.40
16	2024-03-09 18:46:17.843	6.70	86.40	241864173	1339884066	7316365312	1152233472	577742003	894165583	2.40

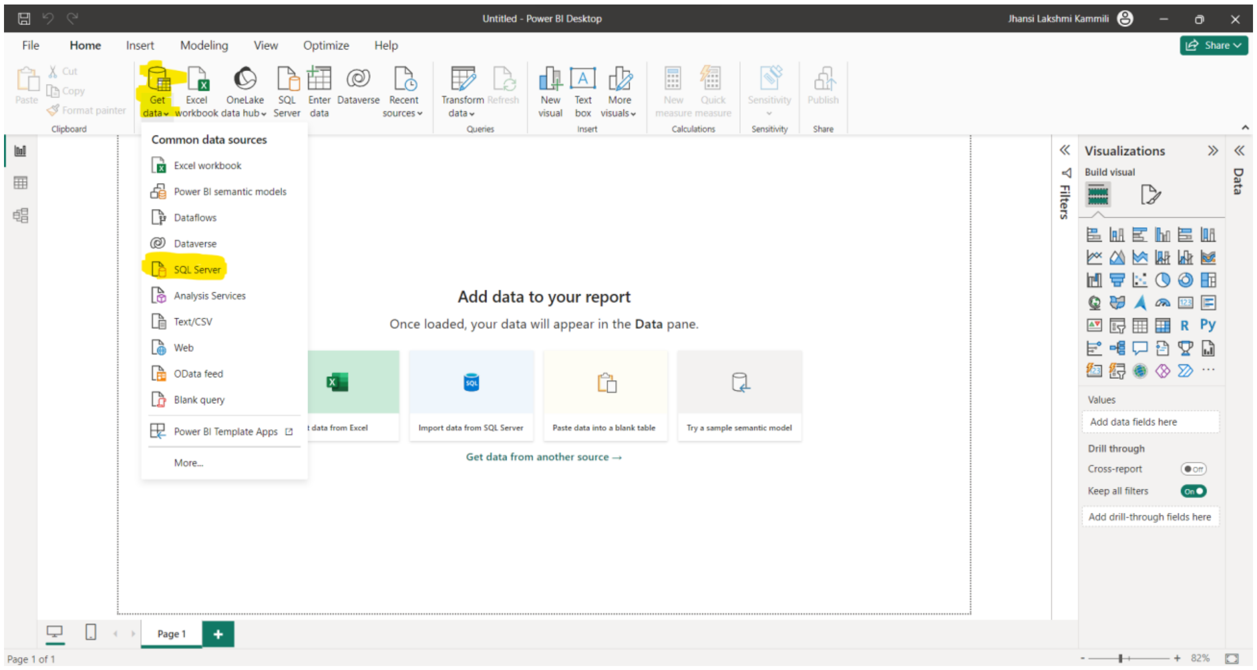
Query executed successfully. | HP\SQLEXPRESS (16.0 RTM) | hp\Admin (62) | System_Information | 00:00:00 | 139 rows

Step 4: Integration with SQL Server Integration Services (SSIS)

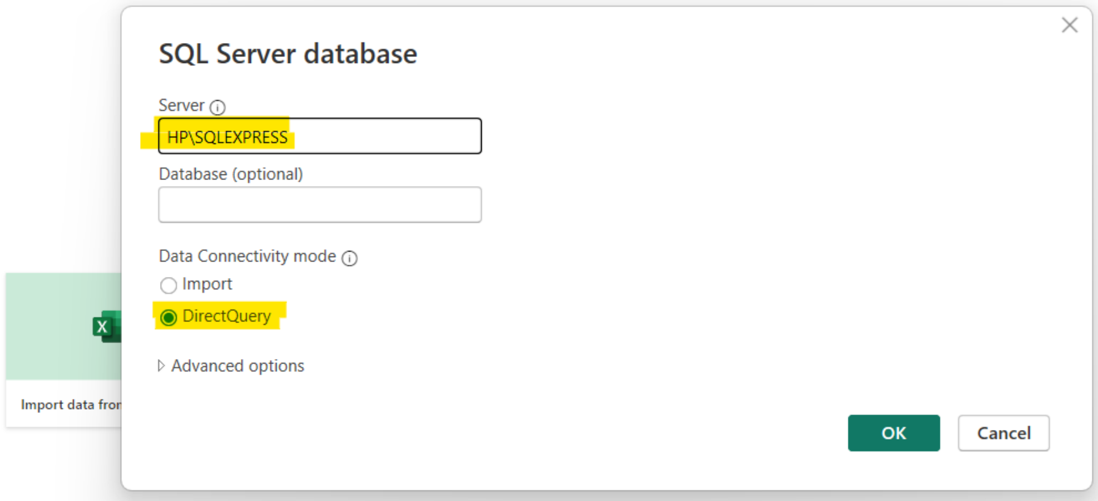
SQL Server Integration Services (SSIS) was utilized to automate the process of data integration. A package was created in SSIS to extract data from various sources, including flat files, databases, and external APIs. This package transformed the data as necessary and loaded it into the 'Performance' table in the SQL Server database. By integrating SSIS into the workflow, the process of data collection and integration was streamlined and automated, reducing manual effort and improving efficiency.

Step 5: Connecting Power BI to SQL Server for Visualization

Power BI, a powerful business analytics tool, was installed and connected to the SQL Server database. By establishing this connection, Power BI was able to directly query the 'Performance' table in the SQL Server database, enabling real-time data analysis and visualization. Power BI was used to create interactive dashboards and visualizations that provided insights into system performance metrics, allowing users to monitor the health and performance of the system in real-time.



To get the data from SQL server
Get data>SQL server


SQL Server database

Server: HP\SQLEXPRESS

Database (optional):

Data Connectivity mode: Import [✓ DirectQuery]

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

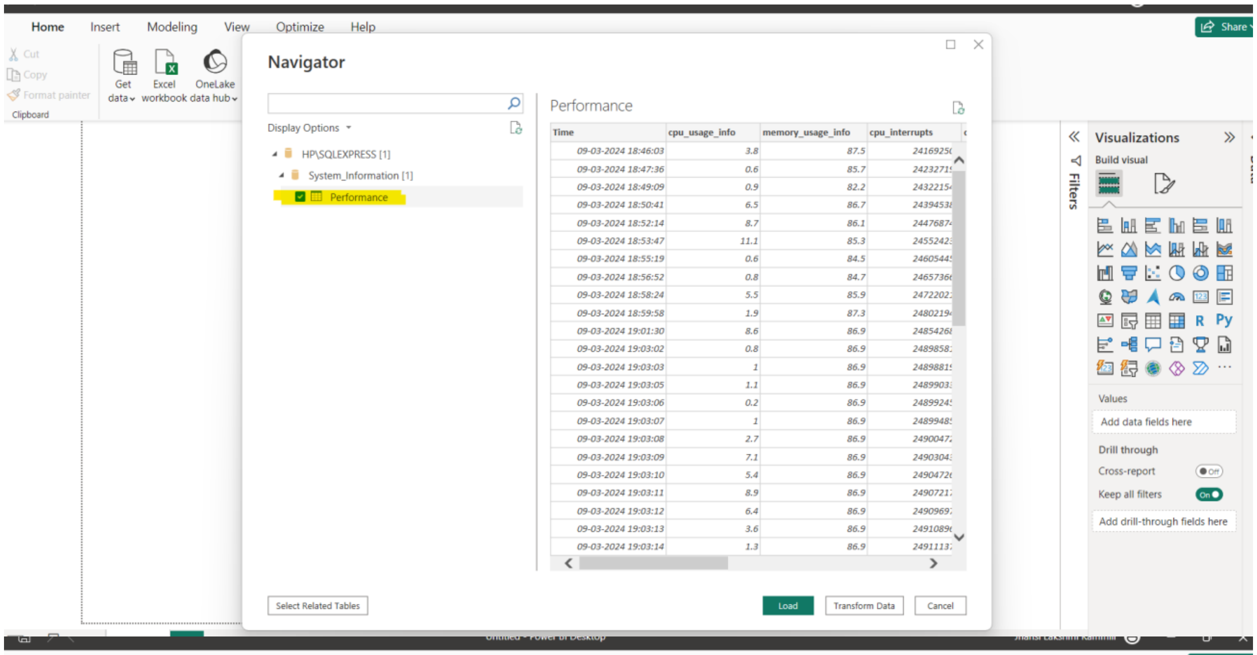
When to use Direct Query?

- 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?

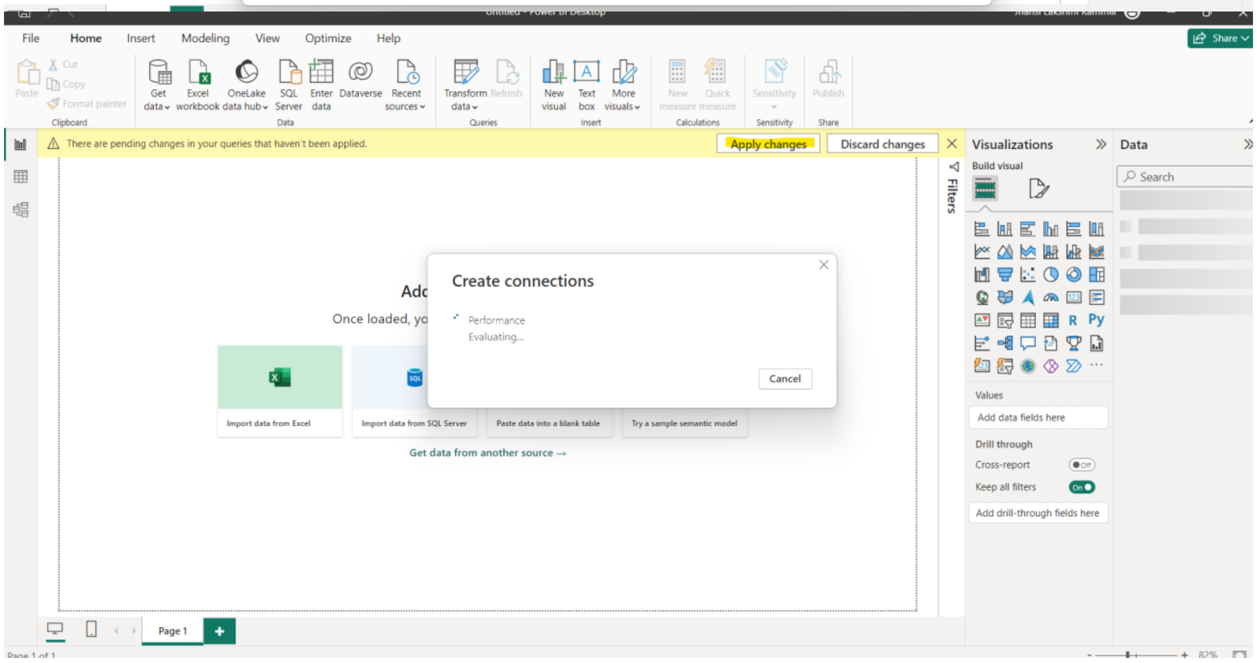
- Use import when dealing with moderately sized datasets or when you need to perform extensive transformations and calculations within Power BI.
- Suitable for scenarios where data is relatively static or where periodic data refreshes are acceptable.
- Provides faster query performance as data is loaded 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.



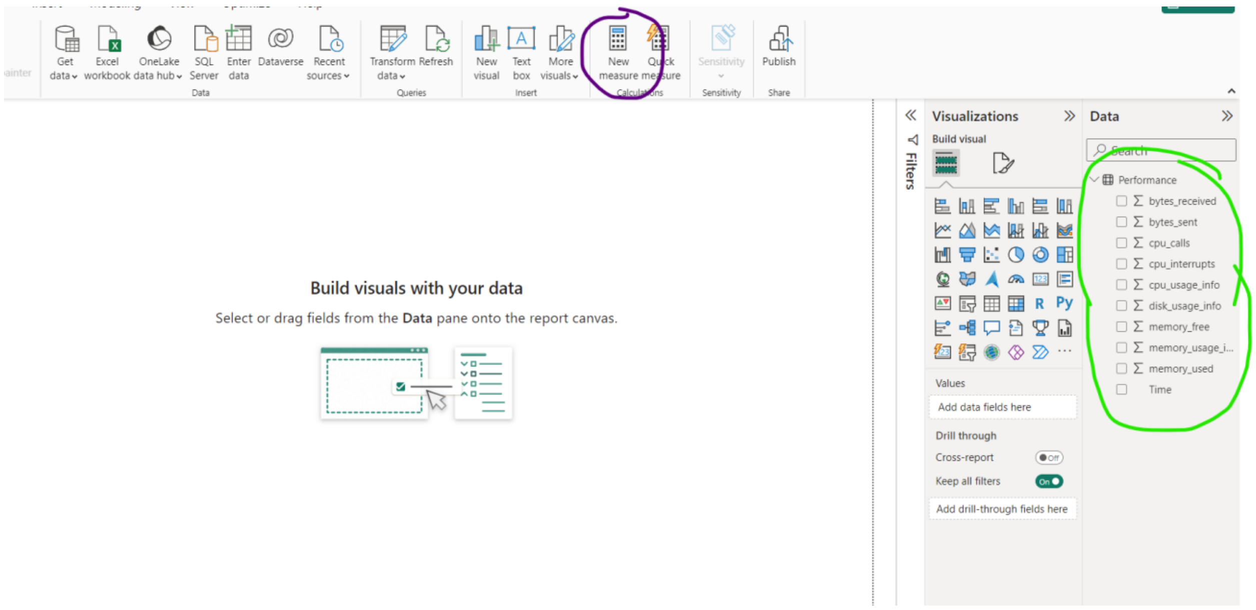
Performance

Time	cpu_usage_info	memory_usage_info	cpu_interrupts
09-03-2024 18:46:03	3.8	87.5	2416925
09-03-2024 18:47:36	0.6	85.7	2423271
09-03-2024 18:49:09	0.9	82.2	2432215
09-03-2024 18:50:41	6.5	86.7	2439453
09-03-2024 18:52:14	8.7	86.1	2447687
09-03-2024 18:53:47	11.1	85.3	2455242
09-03-2024 18:55:19	0.6	84.5	2460544
09-03-2024 18:56:52	0.8	84.7	2465736
09-03-2024 18:58:24	5.5	85.9	2472202
09-03-2024 18:59:58	1.9	87.3	2480219
09-03-2024 19:01:30	8.6	86.9	2485426
09-03-2024 19:03:02	0.8	86.9	2489858
09-03-2024 19:03:03	1	86.9	2489881
09-03-2024 19:03:05	1.1	86.9	2489903
09-03-2024 19:03:06	0.2	86.9	2489924
09-03-2024 19:03:07	1	86.9	2489948
09-03-2024 19:03:08	2.7	86.9	2490047
09-03-2024 19:03:09	7.1	86.9	2490304
09-03-2024 19:03:10	5.4	86.9	2490472
09-03-2024 19:03:11	8.9	86.9	2490721
09-03-2024 19:03:12	6.4	86.9	2490969
09-03-2024 19:03:13	3.6	86.9	2491089
09-03-2024 19:03:14	1.3	86.9	2491113



There are pending changes in your queries that haven't been applied.

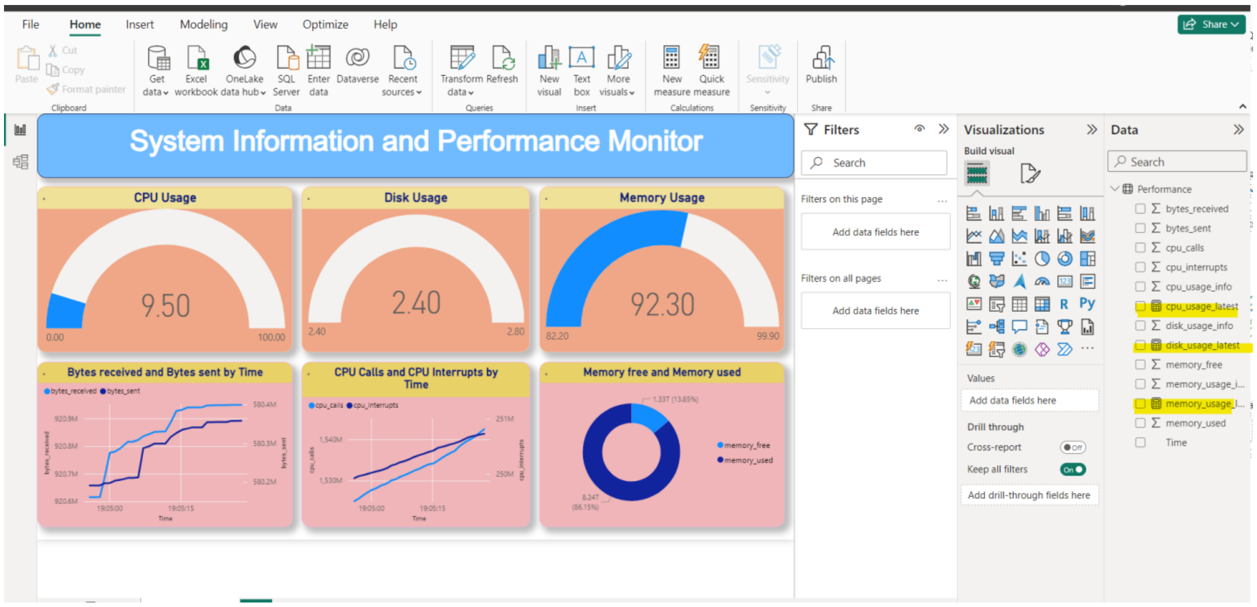
After loading could see the Performance table in the data pane.



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.

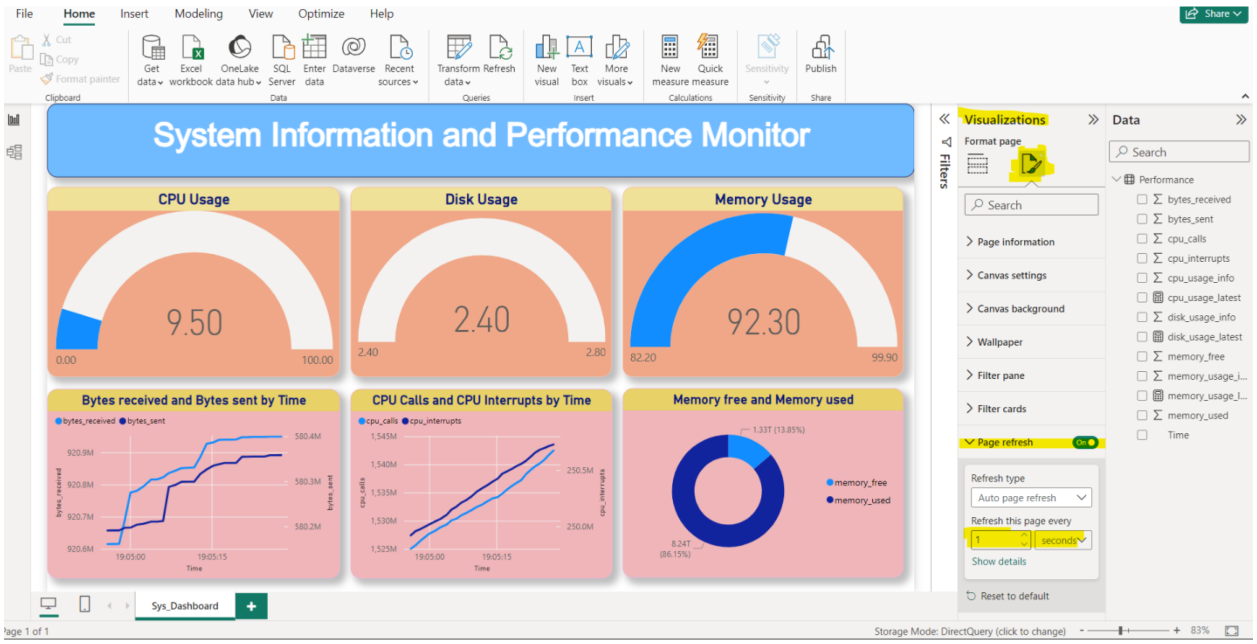
After calculating measures, I've built a dashboard.



- ➢ 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:
  - o Fixed Interval: Update visuals at a constant interval (e.g., every second or five minutes).
  - o Change Detection: Refresh visuals based on data changes rather than a specific interval.



Page refresh: On

Refresh type: Auto page refresh

Refresh this page every: 1 seconds

Results

Data Collection and Storage:

The Python script and SSIS package successfully collected and stored real-time system performance metrics in the 'Performance' table in the SQL Server database.

Visualization:

Power BI was used to create interactive dashboards and visualizations that provided insights into system performance metrics such as CPU usage, memory usage, network traffic, and disk usage.
Real-time data visualization was achieved by enabling the page refresh option in Power BI, ensuring that the dashboard displayed the most up-to-date system performance metrics.

Conclusion

The study successfully demonstrated the integration of Python, SQL Server, SSIS, and Power BI to create a comprehensive system for real-time monitoring of system performance metrics. By leveraging these technologies together, users can gain valuable insights into the health and performance of the system, enabling proactive management and decision-making.

Video reference

video2253296778.mp4

Check this video for clear understanding*