# Benchmarking Study – Checkpoint 2

# Identifying Use Case, Sample Data Sources,

# and Database Systems Study

Pooja Bendre, Ezhilarasu R, and Ritesh Kumar

2024WI\_MS\_DSP\_420-DL\_SEC61\_SEC62: Database Systems

Benchmarking Study – Term Checkpoint 3

# Abid Ali and Jaya R

# February 26, 2024

**Abstract:**

The comprehensive study is designed to conduct a detailed benchmarking analysis of four widely used database tools: PostgreSQL, SQLAlchemy, MySQL CommandLine Client, and DbGate. By executing a range of queries, this research aims to quantify the performance in terms of time efficiency, CPU load, and memory utilization. Emphasizing the comparative advantages and drawbacks of direct database interactions against the employment of ORMs and GUIs, the study navigates through the complexities of various workloads and query intricacies. These tools, which are integral in managing relational databases such as PostgreSQL and MySQL, are dissected to illuminate the most efficient practices in database operations across different scenarios. The insights garnered from this research will serve as a guide for selecting the appropriate tools and methodologies for database management in diverse operational environments, potentially influencing best practices in database administration and development.

**Introduction:**

In response to the escalating complexity and critical nature of financial data management within the stock exchange milieu, this project employs a tailored array of database tools to tackle the massive influx of data. The project leverages PostgreSQL for its robustness and well-established reliability in object-relational database management, SQLAlchemy for its dynamic SQL toolkit capabilities coupled with a powerful ORM layer, the MySQL CommandLine Client for its streamlined command-line interactions, and DbGate for its versatile cross-platform database client that supports a variety of database systems. Each tool is meticulously chosen and evaluated to handle a large and intricate dataset that encompasses over a decade of historical performance data from the National Stock Exchange of India, which features 3.84 million rows across nine columns. This database encapsulates crucial data points, such as stock performance metrics and volume, from March 2014 to February 2024 for over two thousand stocks. Through a rigorous benchmarking process, the research aims to ascertain the optimal database system that balances efficiency with the complexity of handling the high volume, velocity, and variety of financial data, thereby empowering the finance sector with the tools to drive innovation and enhance decision-making in market analysis and risk management.

**Literature Review:**

The field of financial data analysis has garnered significant interest, sparking numerous studies on enhancing database technologies for sophisticated datasets. A notable work, "A Performance Comparison of SQL and NoSQL Databases (2013)" by Le et al., investigated the efficiency of SQL versus NoSQL databases. Silva and Almeida highlighted SQL's robustness in managing Big Data across distributed, scalable systems, emphasizing its capacity to handle datasets characterized by large volume, high velocity, and diverse variety.

Additionally, an article by Matsiaka titled "PostgreSQL vs MySQL: A Detailed Comparison for Database Selection" for MarketSplash provided an in-depth analysis of these two databases. Shah et al., in their study "Performance Study of Time Series Databases," identified marked differences in data injection and query execution times between real and synthetic datasets. Furthermore, Dey et al.'s work, "Predictive Analytics with Structured and Unstructured Data – A Deep Learning Approach," introduced a deep learning framework for predictive analytics that leverages both structured and unstructured data.

These contributions highlight a movement towards a range of database technologies, each with distinct advantages for managing financial data. Our research builds on these insights, aiming to conduct a thorough benchmarking study to clarify the most suitable database system for analyzing financial market data.

**Methods:**

Our methodology for evaluating database tools involves a comprehensive approach that includes the selection of databases, tools, and the configuration of a testing environment. We have selected PostgreSQL, SQLAlchemy, MySQL CommandLine Client, and DbGate based on their prevalence and typical use scenarios in handling financial data. The database contains a decade of historical stock performance data from the National Stock Exchange of India, with 3.84 million rows and nine columns of attributes, representing an extensive dataset to work with.

We established a controlled testing environment by deploying all four database tools on the same Windows machine to ensure a fair comparison. A suite of 10 standardized benchmark queries, representing common database operations, was developed to evaluate the tools' performance. These queries were executed repeatedly to gather data on execution time, CPU consumption, and memory utilization.

The benchmarking process was rigorous, ensuring that each tool was tested under identical conditions. This methodical approach allowed us to generate a robust dataset for analysis. By interpreting this data, we aim to shed light on the performance characteristics of each tool, providing valuable insights for database administrators and financial analysts in the finance sector. This methodological rigor is critical for our goal of identifying the most efficient database tool for managing and analyzing voluminous financial datasets, thus enhancing the capacity for insightful market analysis and informed decision-making in the fast-paced financial industry.

**Results:**

The benchmarking results across ten queries present a multifaceted view of the performance of PostgreSQL, SQLAlchemy (with PostgreSQL), MySQL CommandLine Client, and DbGate (with MySQL), are listed in Table 1. PostgreSQL showed remarkably consistent and low execution times, particularly in simpler queries (Query 1 and 2), with moderate CPU usage that spiked significantly in Query 6 and 7. Memory usage for PostgreSQL remained consistently low or negligible.

SQLAlchemy, acting as an ORM for PostgreSQL in Python, generally demonstrated higher execution times than PostgreSQL, especially noticeable in complex queries (Query 9 and 10). This suggests that the abstraction layer of ORM adds overhead. However, CPU usage was mixed; it was similar to PostgreSQL for simpler queries but much lower for complex ones, indicating efficient CPU utilization under heavy tasks. Memory usage showed occasional spikes, which is characteristic of Python's memory management.

MySQL CommandLine Client displayed very efficient execution times in the simplest queries (Query 1 and 2) but struggled with more complex operations (Query 10), suggesting limitations in handling more demanding tasks. CPU usage was generally lower than PostgreSQL and SQLAlchemy for simpler queries, but this changed as the queries became more complex, sometimes exceeding PostgreSQL’s CPU usage.

DbGate had generally longer execution times across all queries, which might be attributed to the GUI overhead. Despite this, CPU usage was relatively controlled, not the highest in any query, but memory usage was consistently negligible.

Overall, the results indicate that while ORMs like SQLAlchemy add overhead in execution time, they can be more efficient in CPU usage during complex operations. CommandLine clients offer fast execution times for simple operations but may not scale as well with complexity. GUI-based tools such as DbGate may introduce additional overhead that affects performance. The choice of tool should thus be aligned with the specific requirements of the task, considering the trade-offs between execution time, CPU, and memory usage.

**Conclusion:**

The study's conclusion highlights the performance variability among the database tools when managing complex financial datasets. PostgreSQL and SQLAlchemy, despite using the same database system, show different time efficiencies, with PostgreSQL generally outperforming SQLAlchemy. Memory usage was minimal across most tools and queries, but SQLAlchemy exhibited occasional spikes, potentially due to Python's object handling. CPU usage was significantly high for MySQL CommandLine Client in complex queries, reflecting possible inefficiencies in resource utilization. DbGate, while user-friendly, had longer execution times, likely due to GUI overhead. The findings suggest that while ORMs and GUIs add convenience, they may also introduce performance trade-offs. Database selection for financial data management should thus consider specific use-case requirements, balancing the need for speed and efficiency against the ease of use and functionality provided by different interfaces and tools.

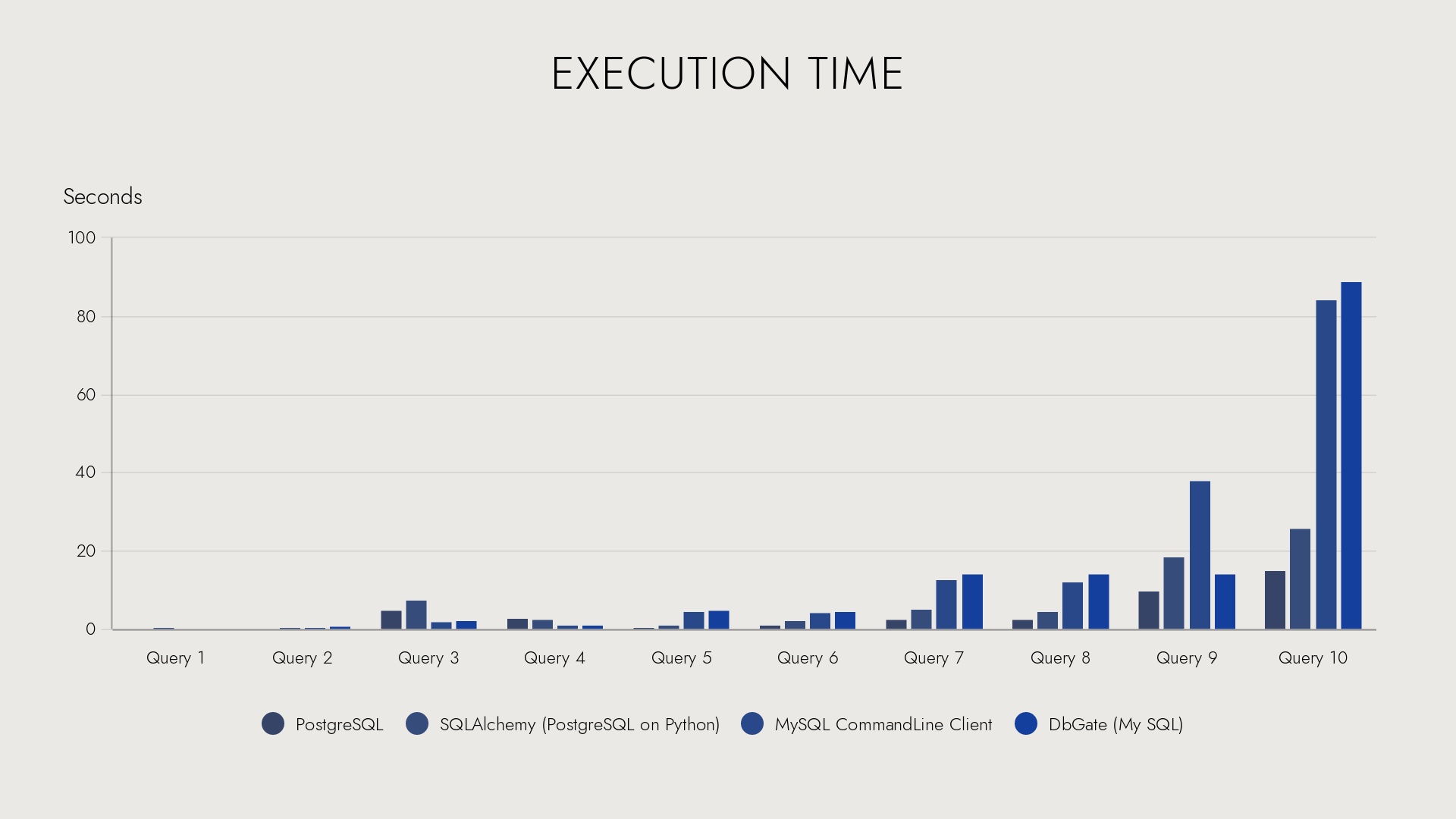
**References:**

1. Data: NSE India Futures & Options Daily (2000-20) <https://www.kaggle.com/datasets/tanay001/nseindia-futures-options-daily/data>
2. NSE Historical Reports <https://www.nseindia.com/resources/historical-reports-capital-market-daily-monthly-archives>
3. Practical SQL – Second Edition, Debarros A., <https://nostarch.com/practical-sql-2nd-edition>
4. Data Engineering with Python, Crickard P., <https://www.packtpub.com/en-fi/product/data-engineering-with-python-9781839214189?type=ebook>
5. Efficient MySQL Performance, Nichter D., <https://www.oreilly.com/library/view/efficient-mysql-performance/9781098105082/#:~:text=Daniel%20Nichter%20shows%20you%20how,the%20most%20important%20MySQL%20metrics>
6. Building Knowledge Graphs: A Practitioner’s Guide, Barrasa J. and Webber J, <https://neo4j.com/knowledge-graphs-practitioners-guide/?utm_source=google&utm_medium=PaidSearch&utm_campaign=GDB&utm_content=AMS-X-SEM-Category-Expansion-Evergreen-Search&utm_term=&gad_source=1&gclid=CjwKCAiAiP2tBhBXEiwACslfnlvKh8bgC8n93lU8rhc_BJk5IR-eFhNCW5BNieF3L8AWfPS8rEagZBoC6ugQAvD_BwE>
7. A performance comparison of SQL and NoSQL databases. IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing - Proceedings. 15-19. 10.1109/PACRIM.2013.6625441 – Li, Yishan & Manoharan, Sathiamoorthy. (2013). <https://www.researchgate.net/publication/261079289_A_performance_comparison_of_SQL_and_NoSQL_databases>
8. SQL: From Traditional Databases to Big Data – Silva, Yasin & Almeida, Isadora & Queiroz, Michell. (2016). <https://www.researchgate.net/publication/311488672_SQL_From_Traditional_Databases_to_Big_Data>
9. PostgreSQL Vs MySQL: A Detailed Comparison For Database Selection - [Matsiaka](https://marketsplash.com/author/maksym-2/), MarketSplash. <https://marketsplash.com/tutorials/mysql/postgresql-vs-mysql/>
10. Performance Study of Time Series Databases – Shah, Bonil & Jat, P. & Sashidhar, Kalyan. (2022). <https://www.researchgate.net/publication/363128579_Performance_Study_of_Time_Series_Databases>
11. Predictive Analytics with Structured and Unstructured Data – A Deep Learning Approach (2017) – Lipika Dey, Hardik Meisheri and Ishan Verma. <https://www.comp.hkbu.edu.hk/~iib/2017/Dec/article5/iib_vol18no2_article5.pdf>

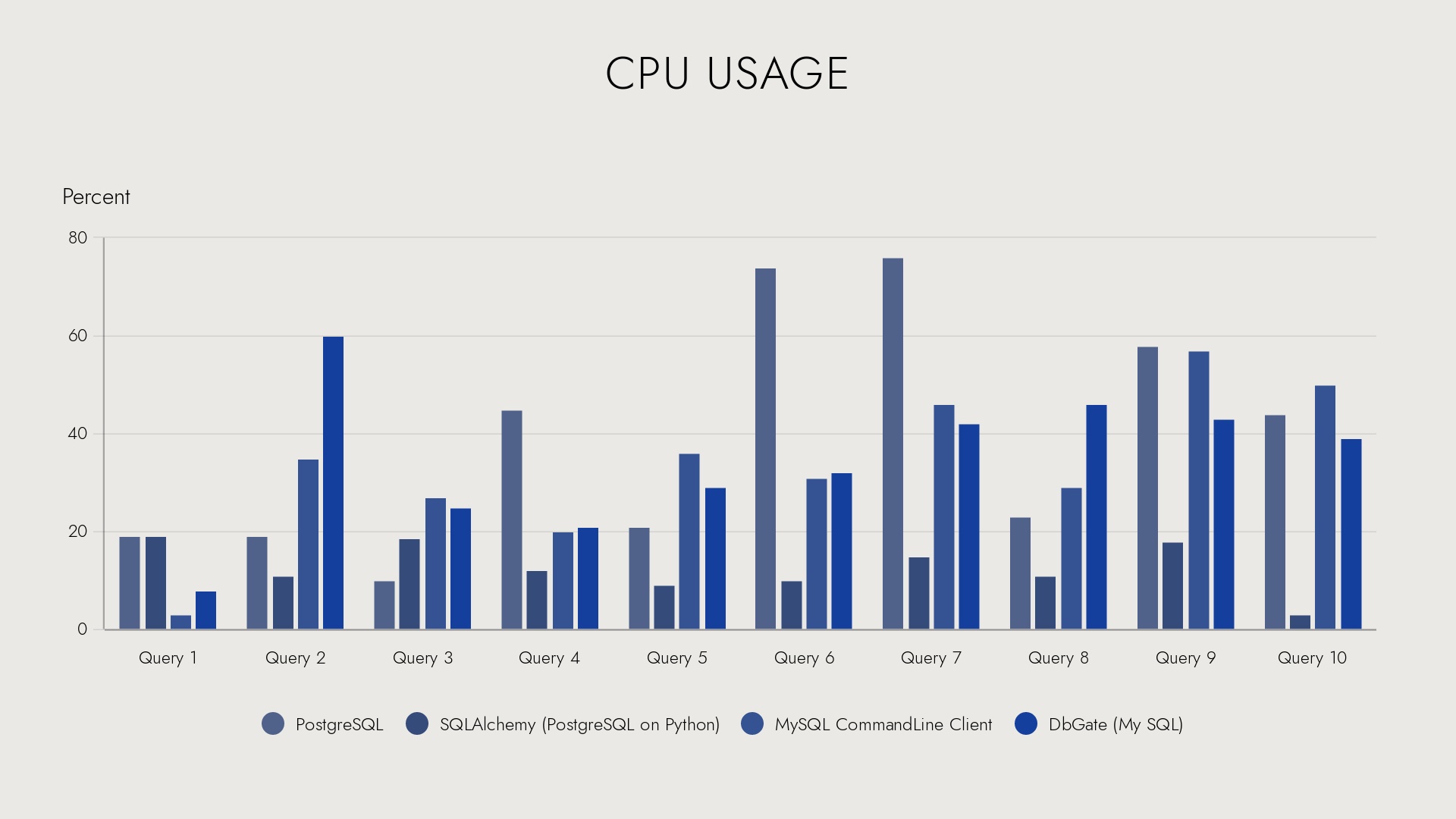
**Appendix:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **PostgreSQL** | **SQLAlchemy (PostgreSQL on Python)** | **MySQL CommandLine Client** | **DbGate (My SQL)** |
| **Query 1** | **Time** | 0.07 | 0.68 | 0.02 | 0.38 |
| **CPU** | 19.0% | 19.3% | 3.0% | 8.0% |
| **Memory** | 0.10 | 0.00 | 0.00 | 0.00 |
| **Query 2** | **Time** | 0.06 | 0.59 | 0.58 | 0.75 |
| **CPU** | 19% | 11% | 35% | 60% |
| **Memory** | 0.10 | 0.20 | 0.00 | 0.00 |
| **Query 3** | **Time** | 4.88 | 7.58 | 2.14 | 2.30 |
| **CPU** | 10% | 19% | 27% | 25% |
| **Memory** | 0.10 | 0.00 | 0.00 | 0.00 |
| **Query 4** | **Time** | 2.77 | 2.62 | 1.16 | 1.20 |
| **CPU** | 45% | 12% | 20% | 21% |
| **Memory** | 0.00 | 0.00 | 0.00 | 0.00 |
| **Query 5** | **Time** | 0.72 | 1.21 | 4.62 | 4.90 |
| **CPU** | 21% | 9% | 36% | 29% |
| **Memory** | 0.00 | 0.30 | 0.00 | 0.00 |
| **Query 6** | **Time** | 1.22 | 2.45 | 4.46 | 4.70 |
| **CPU** | 74% | 10% | 31% | 32% |
| **Memory** | 0.00 | 0.20 | 0.00 | 0.00 |
| **Query 7** | **Time** | 2.65 | 5.22 | 12.92 | 14.15 |
| **CPU** | 76% | 15% | 46% | 42% |
| **Memory** | 0.00 | 0.20 | 0.00 | 0.00 |
| **Query 8** | **Time** | 2.54 | 4.73 | 12.07 | 14.25 |
| **CPU** | 23% | 11% | 29% | 46% |
| **Memory** | 0.00 | 0.20 | 0.00 | 0.00 |
| **Query 9** | **Time** | 9.83 | 18.58 | 37.94 | 14.33 |
| **CPU** | 58% | 18% | 57% | 43% |
| **Memory** | 0.00 | 0.30 | 0.00 | 0.00 |
| **Query 10** | **Time** | 15.19 | 25.90 | 84.30 | 89.00 |
| **CPU** | 44% | 3% | 50% | 39% |
| **Memory** | 0.00 | 0.20 | 0.00 | 0.00 |

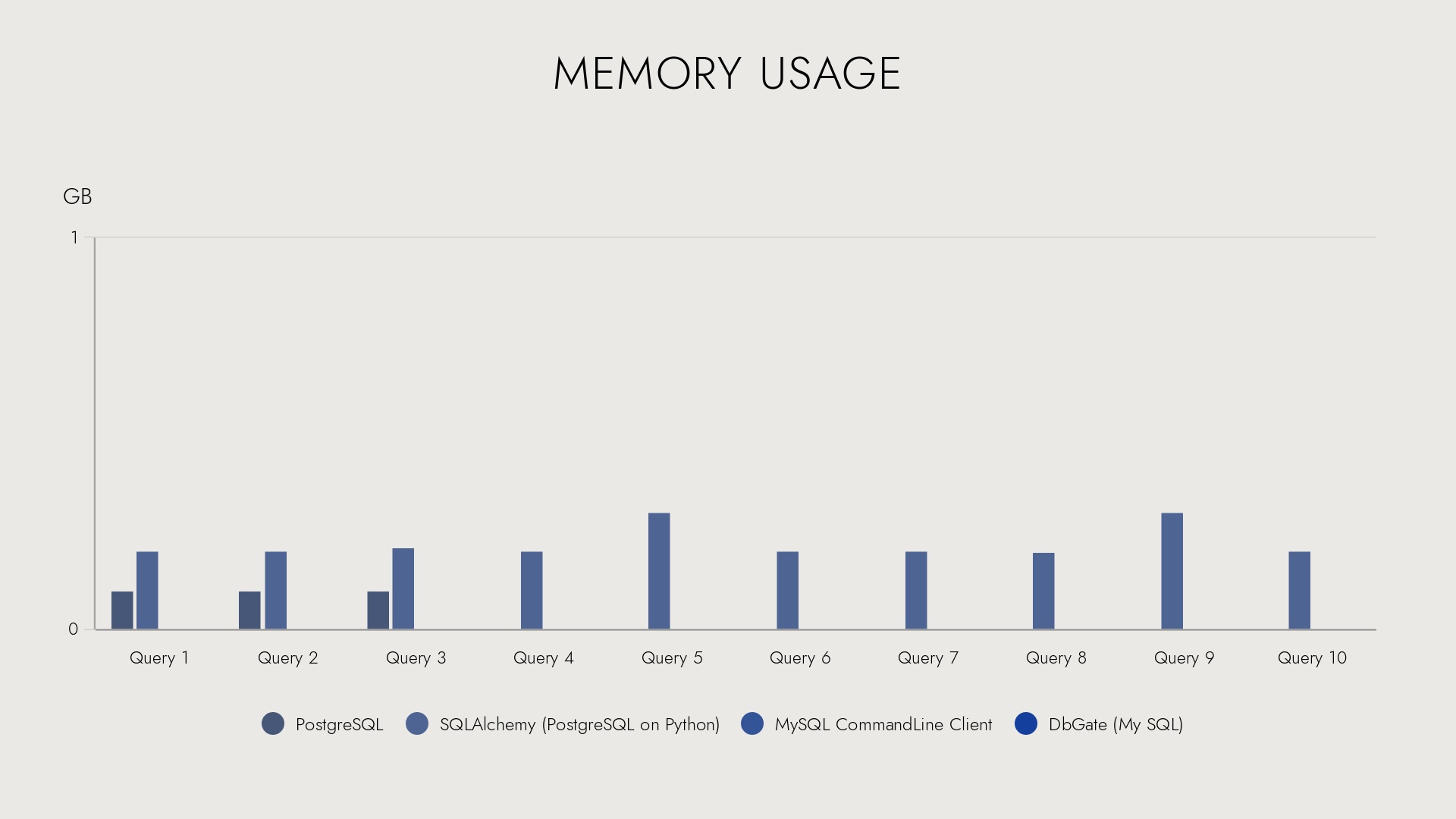
Table 1



Plot 1



Plot 2



Plot 3