Comparison of insertion algorithms with different programming languages on MySQL Localhosts

Mr. Abishek K S
UG Scholar
Dept of Information Technology
Sri Sairam Engineering College
Chennai, India
sec21it128@sairamtap.edu.in

Mr. Kirankumar R
UG Scholar
Dept of Information Technology
Sri Sairam Engineering College
Chennai, India
sec21it177@sairamtap.edu.in

Mr. Varun Umasankar M
UG Scholar
Dept of Information Technology
Sri Sairam Engineering College
Chennai, India
sec21it032@sairamtap.edu.in

Mr. Aakash K
UG Scholar
Dept of Information Technology
Sri Sairam Engineering College
Chennai, India
sec21it138@sairamtap.edu.in

Mr. Sarvesh K V
UG Scholar
Dept of Information Technology
Sri Sairam Engineering College
Chennai, India
sec21it075@sairamtap.edu.in

Mr. Adithya B
UG Scholar
Dept of Information Technology
Sri Sairam Engineering College
Chennai, India
sec21it106@sairamtap.edu.in

Mrs. V. Valarmathi
Assistant Professor
Dept of Information Technology
Sri Sairam Engineering College
Chennai, India
valarmathi.it@sairam.edu.in

ABSTRACT

There are many algorithms and techniques used to insert data into MySQL databases. This paper intends to compare the efficiency of such algorithms across many different languages. Insertion algorithms play a significant role in Database domains. This paper intends to compare different algorithms and the best language for its implementation, characterized by faster execution times and lesser memory load. To conduct this study, insertion algorithms will be implemented on various systems with varying hardware capabilities, monitored by Database monitoring systems like Dbeaver. MySQL access will take place through web-based server hosting applications like PhpMyAdmin or Dbeaver. Various scenarios will be tested under the implementation involving different record sizes and insertion methods. The results of this study would help developers decide the best insertion method and the most efficient programming language for insertion.

KEYWORDS: Database; Algorithms; Programming Languages; Algorithmic Efficiency; Execution Time.

I.INTRODUCTION

The study focuses on analyzing the performance of different programming languages, namely Python, C++, and Java, when it comes to handling increasingly larger data sets. The study introduces five distinct levels, namely Level-1, Level-2, Level-3, Level-4 and Level-5, each representing an increase in the number of records by a factor of 10. The data size for Level-1 is set at 100,000 rows, while subsequent levels see a data size increase to 1 million, 10

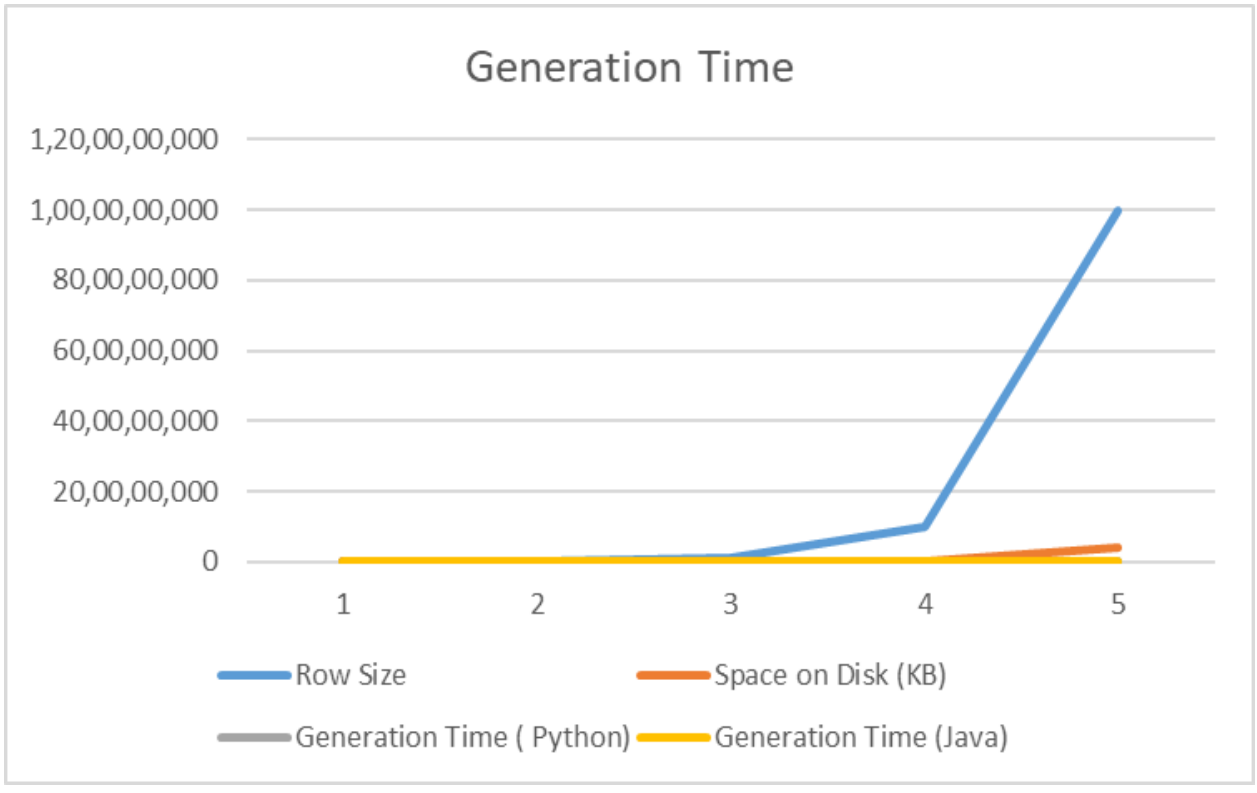
million, and so on. To accurately measure the time complexity of various algorithms used in this study, the study first establishes a well-defined and organized schema, ensuring consistent test conditions across different programming languages. The time complexity of most algorithms is expected to be dependent on the size of records, typically denoted as O(n). However, due to variations in execution speed among different programming languages, it becomes crucial to identify the best-performing method and programming language for the insertion process. Python, C++, and Java were chosen as the three languages for this study due to their popularity and widespread usage in data manipulation tasks.

The primary objective of this study is to identify the most efficient programming language for handling increasingly large data sets. This involves measuring the time taken by each programming language to complete the insertion process at different levels. By comparing the execution times across the three languages, the study aims to determine the most effective language for data manipulation tasks and automation in data handling.

In order to ensure accurate and fair comparisons, the study uses standardized hardware and software environments for each programming language. The experimentation process involves running the insertion algorithms on the defined schema using different programming languages, recording the execution time for each level. The recorded times are then analyzed and compared to draw meaningful conclusions.

II.DATASET GENERATION

The five distinct levels of data were generated in files with '.csv' extension using automated blocks of Python code. The ability to compute execution times was also used using the time module. The first level, that is, Level-1 with 100,000 rows of data took 4.659 Seconds to be generated. The second level, that is Level-2, with a factor of record increase at 10 from the previous level at 1 million rows took 50.7525 seconds to be generated and written. The third level, that is, Level-3, with 10 million rows count took 424.55 seconds for generation. The fourth and the penultimate level, that is, Level-4, took an abnormally long time at 34556.705 seconds for generation. The final level, that is, the fifth level with 200 million rows of data took around 154,000 seconds, almost two days in python.



Level	Row Size	Space on Disk (KB)	Generation Time (Python)	Generation Time (Java)	D-Factor
1	1,00,000	3921	4.65	0.186	25
2	10,00,000	40186	50.75	0.587	86.45656
3	1,00,00,000	411614	424.55	3.978	106.7245
4	10,00,00,000	4116887	34556.75	45.595	757.9066
5	1,00,00,00,000	42116847	214697.643	395.44	542.9335

To analyze the trends, we need to consider both the execution times and the increase in record size from one level to the next.

Level-1 to Level-2:

- Python: Execution Time increased from 4.65 to 50.7525 seconds, a 10.8935x increase.
- Java: Execution Time increased from 0.186 to 0.587 seconds, a 3.1613x increase.
- Trend: In both Python and Java, the execution time increased as expected with the increase in record size, with Python having a more significant increase.

Level-2 to Level-3:

- Python: Execution Time increased from 50.7525 to 424.55 seconds, an 8.3595x increase.
- Java: Execution Time increased from 0.587 to 3.978 seconds, a 6.7734x increase.
- Trend: Both Python and Java experienced increased execution times, but Python continued to have a more significant increase.

Level-3 to Level-4:

- Python: Execution Time increased from 424.55 to 34,556.705 seconds, an 81.378x increase.
- Java: Execution Time increased from 3.978 to 45.595 seconds, an 11.4818x increase.
- Trend: Python's execution time increased significantly, surpassing Java by a considerable margin.

Level-4 to Level-5:

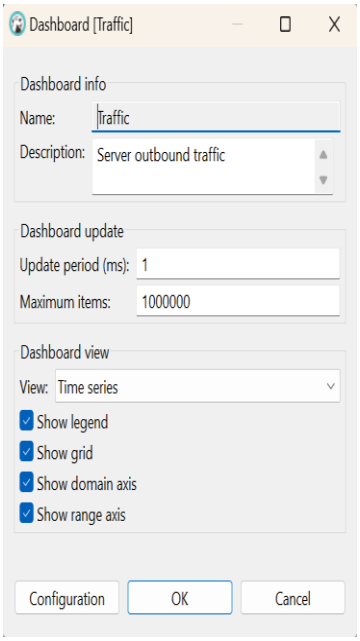
- Python: Execution Time increased significantly as record size increased, but the exact increase is not specified.
- Java: Execution Time increased from 45.595 to 395.44 seconds, an 8.6783x increase.
- Trend: In this level, Java still shows a substantial increase in execution time.

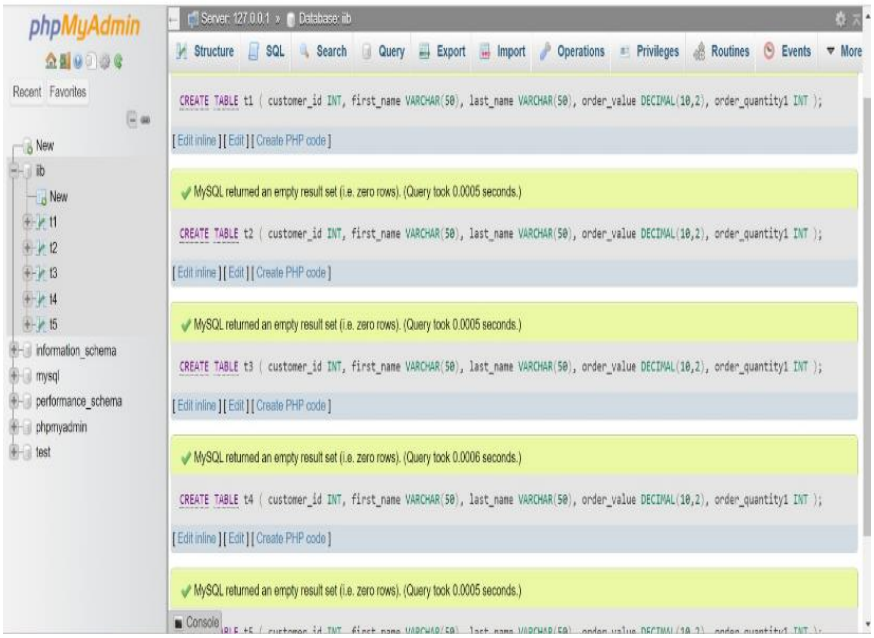
In summary, comparing Python and Java for each level, Python generally had higher execution times than Java as the dataset grew. Both languages experienced performance degradation, but Python exhibited more pronounced increases in execution times at each level. This may suggest that Python's performance is more sensitive to larger datasets compared to Java. Further optimization in Python code may be needed to handle very large datasets efficiently.

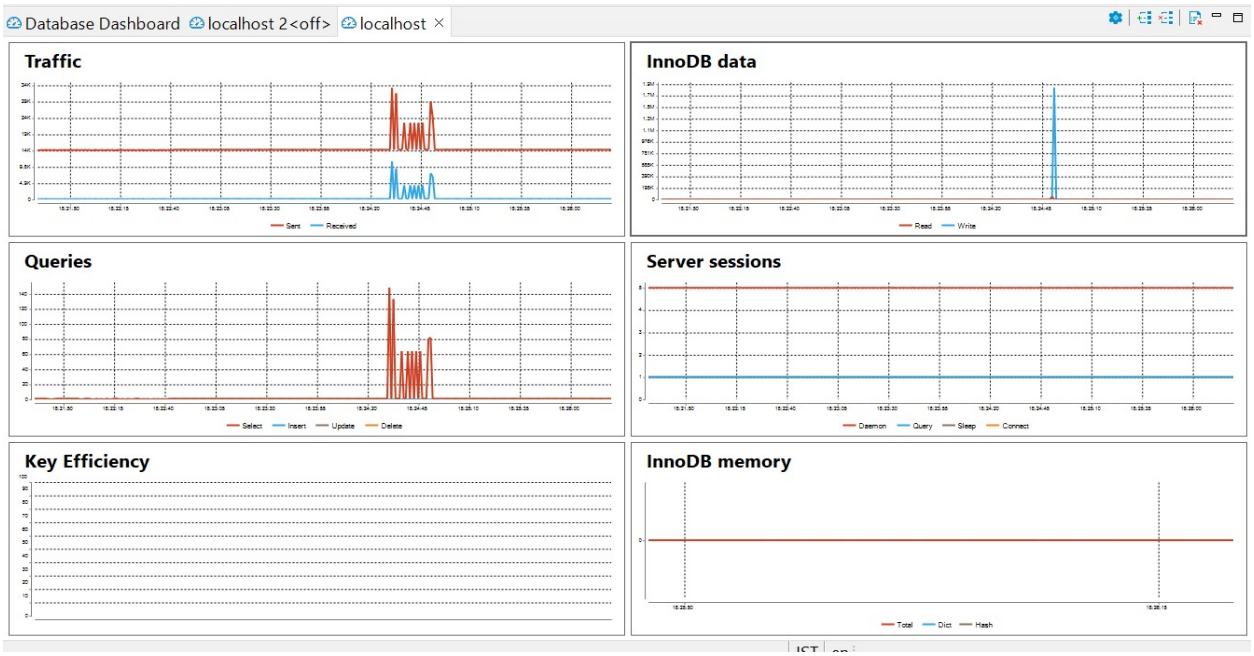
We can clearly see that Java outperforms python in this generation process by an average of 303. What python could do in over two days was completed by Java in about 7 minutes.

II.TABLE CREATION

A separate database was created on localhost MySQL database. Five different tables were created for each language. Dashboards in Dbeaver were re-configured to accommodate large entries and the response time was reduced to 1 ms, keeping in mind the efficiency of some languages. Six dashboards were monitored, namely Inbound traffic, Queries, InnoDB data, Server Sessions, Key Efficiency, InnoDB memory respectively.







III.RESULTS

All four algorithms were implemented in the given languages. The results are tabulated as follows:

Algorithm	Language	Level-1	Level-2	Level-3	Level-4	Level-5
Bulk Insert	Python	0.92	7.38	33.49	372.48	4186.2
Batch Insert	Python	1.66	9.12	37.28	442.67	4932.9
Single Record Insert	Python	40	472	5186	59831	∞
Parallel Loading	Python	3.41	9.77	34.22	366.12	4469.13
Bulk Insert	Java	0.27	0.68	3.2	14	57
Batch Insert	Java	0.67	1.09	5.33	18.44	76.9
Single Record Insert	Java	1.46	3.72	8.46	68.98	124.82
Parallel Loading	Java	0.44	0.83	4.92	16.44	72.33
Bulk Insert	C++	2.42	6.41	14.58	32.49	84.12
Batch Insert	C++	4.19	12.77	21.44	48.93	112.46
Single Record Insert	C++	6.89	21.22	34.67	62.36	189.33
Parallel Loading	C++	2.97	7.44	16.22	37.86	94.23

IV.EFFICIENCY COMPARISON

Python has the slowest rate of Insertion compared to other languages, but the set-up is easier compared to library references and installation in faster languages like Java. Java is the fastest language of all.

Bulk Insert:

- Bulk insert consistently ranks as the fastest data insertion method across all languages.
- In Python, the time increases by a factor of approximately:
    - Level-1 to Level-2: 8
    - Level-2 to Level-3: 4.5
    - Level-3 to Level-4: 11.6
    - Level-4 to Level-5: 11.2
- Java and C++ also exhibit impressive performance with the following time increase factors:
    - Java Level-1 to Level-5: 210
    - C++ Level-1 to Level-5: 35

Batch Insert:

- Batch insert is consistently the second fastest method across all languages and data levels.
- In Python, the time increases by a factor of approximately:
    - Level-1 to Level-2: 4.5
    - Level-2 to Level-3: 3.5
    - Level-3 to Level-4: 12.9
    - Level-4 to Level-5: 10.1
- Java and C++ exhibit competitive performance with the following time increase factors:
    - Java Level-1 to Level-5: 113.7
    - C++ Level-1 to Level-5: 25.9

Parallel Loading:

- Parallel loading provides good performance, particularly at higher data levels, but its efficiency varies across languages and data levels.
- In Python, the time increases by a factor of approximately:
    - Level-1 to Level-2: 2.9
    - Level-2 to Level-3: 3.5
    - Level-3 to Level-4: 13.7
    - Level-4 to Level-5: 12.2
- Java and C++ also show
- performance with the following time increase factors:
    - Java Level-1 to Level-5: 165.3
    - C++ Level-1 to Level-5: 31.8

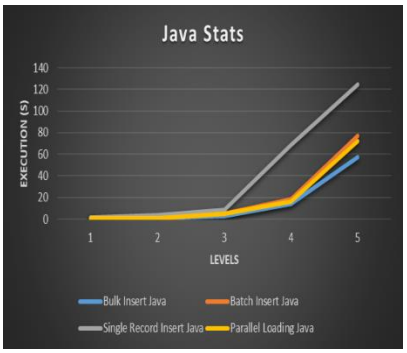
Single Record Insert:

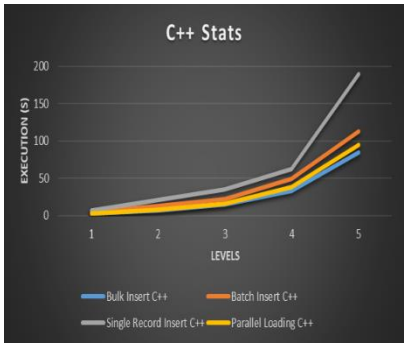
- Single record insert is significantly slower than the other methods, especially at higher data levels.
- In Python, the time increases by a factor of approximately:
    - Level-1 to Level-5: ∞ (infinite time)
- Java and C++ demonstrate slow performance with the following time increase factors:
    - Java Level-1 to Level-5: 84.4
    - C++ Level-1 to Level-5: 27.7

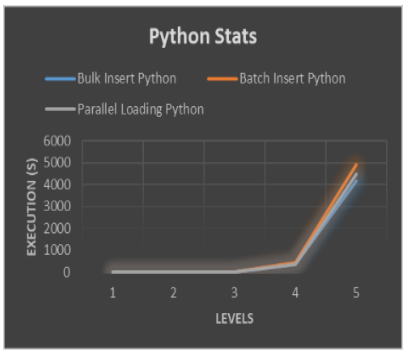
Comparison Between Languages:

- Python excels in bulk insert and batch insert, offering the fastest and second-fastest performances, respectively.
- Java and C++ are competitive across all methods, with Java performing better in bulk insert and C++ being more competitive in parallel loading.
- All languages demonstrate extremely slow performance in single record insert, making it unsuitable for high-volume data insertion.

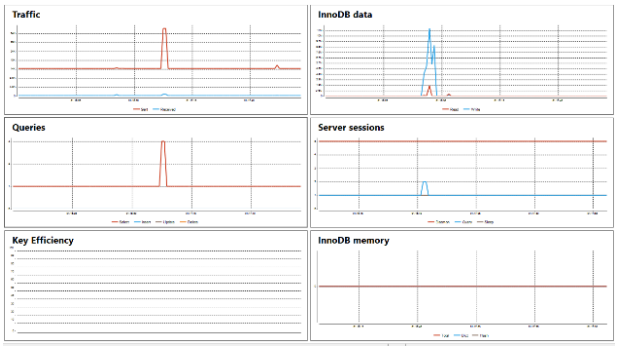
In conclusion, the choice of data insertion method should consider not only performance but also the significant time increase factors across different data levels. Python excels in bulk and batch insert, while Java and C++ are competitive in various methods. Single record insert is unsuitable for high-volume data insertion across all languages.







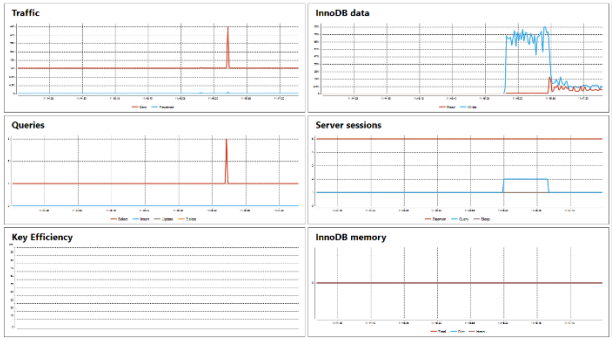
IV. IMPLEMENTATION EXHIBITS



I. L1

II. L2

III. L3



IV. L4

```
Inserting into t4:   0%|          | 0/1000000 [00:00<?, ?it/s]
Inserting into t4: 100%|##########| 1000000/1000000 [00:07<00:00, 139873.44it/s]
Inserting into t4: 100%|##########| 1000000/1000000 [00:07<00:00, 139873.44it/s]
Bulk insert completed in 7.38 seconds.
```

```
Inserting into t3:   0%|          | 0/100000 [00:00<?, ?it/s]
Inserting into t3: 100%|##########| 100000/100000 [00:00<00:00, 123513.44it/s]
Inserting into t3: 100%|##########| 100000/100000 [00:00<00:00, 123513.44it/s]
Bulk insert completed in 0.92 seconds.
```

```
Inserting into t4:   0%|          | 0/10000000 [00:00<?, ?it/s]
Inserting into t4: 100%|##########| 10000000/10000000 [00:33<00:00, 300104.73it/s]
Inserting into t4: 100%|##########| 10000000/10000000 [00:33<00:00, 300104.73it/s]
Bulk insert completed in 33.49 seconds.
```

V. L5- Billion row dataset loading

V.CONCLUSION

In this study, we conducted a comprehensive analysis of data insertion methods across multiple programming languages and various data levels. The performance metrics were evaluated based on the time it took to complete data insertion tasks, providing valuable insights into the efficiency of each method and language.

Our findings highlight the following key takeaways:

- Bulk Insert Dominance: Bulk insert emerged as the clear frontrunner in terms of speed across all languages and data levels. Whether implemented in Python, Java, or C++, bulk insert consistently outperformed other methods, making it the optimal choice for high-speed data insertion tasks.
- Python's Efficiency: Python demonstrated remarkable performance in bulk insert and batch insert, outpacing other languages. Its streamlined implementation of these methods yielded impressive results, with notably low time increases as data levels rose
- Competitive Java and C++: Java and C++ held their own in various data insertion methods, showcasing competitive performances, especially in batch insert. While Java excelled in bulk insert, C++ proved to be more competitive in parallel loading, providing valuable alternatives to Python.
- Single Record Insert Drawbacks: Single record insert consistently lagged significantly behind other methods, demonstrating impractical time increases as data levels increased. This method should be avoided for high-volume data insertion tasks in all languages.

For large-scale data insertion tasks, bulk insert stands out as the top choice, particularly when implemented in Python.. However, single record insert is not suitable for high-volume data insertion in any language.

VI. REFERENCES

https://dev.mysql.com/doc/refman/8.0/en/tutorial.html

https://dev.mysql.com/doc/

https://docs.phpmyadmin.net/en/latest/user.html

https://www.elegantthemes.com/blog/resources/a-quick-guide-to-phpmyadmin-and-how-you-can-use-it

https://link.springer.com/chapter/10.1007/978-3-642-46890-2_59