iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** IV **Month of publication:** April 2025

DOI: https://doi.org/10.22214/ijraset.2025.68419

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IV Apr 2025- Available at www.ijraset.com

Enhancing Database Efficiency: A Comprehensive Approach to DB Lock Optimization

Shubham Pal Singh[1], Dr.Rishi Mohan Bhatnagar[2], Yash Pal Singh[3], Khushi Pal[4]
Artificial Intelligence Developer Tata Consultancy Services North America Birla Institute of Science and Technology Pilani

Abstract: ***Typically, large-scale data transfers into the database cause lots of locking which negatively affects query execution, concurrency, as well as blocking operations such as ALTER, UPDATE, and DELETE. To ensure data consistency, a set of locks are necessary, that can become a performance bottleneck as well as cause deadlocks. The contributions of this paper include three novel techniques to improve database operation and defeat large data loads through lock contention. We minimize data handling efficiency by converting lists into sets first. The second is that data can be processed outside the main database by using temporary or virtual memory, which minimizes the lock durations. Third, by using 'placeholders' when inserting data, the operations stay non-blocking and do not affect other transactions. We implement these methods to show how the performance of the database can be increased, how high the concurrency is, and how we can prevent locking issues in the database in the future.***
Keywords: Glue Jobs, Databases, PostgreSQL, Service Now, Cloud Watch

I. INTRODUCTION

Large-scale data integration is very important and when it comes to these integrations, the databases have many factors to be taken into consideration to get it done efficiently. The problem addressed in this research is the database performance and the foreign key violations during the data insertion and deletion in the highly interdependent tables. Our system has 62 tables involved and the four core tables, Provider, Sides, Cases, and Benefit Profiles, are linked through foreign key constraints and other tables use unique keys for integrity of data. However, the most difficult problem comes from the manipulation of data in these core tables with frequent data violations of foreign keys that cause transaction failures and data inconsistencies.

To address this challenge, we adopted the following optimization strategies in order to enhance the optimization of data insertion and insertion. While using AWS Glue jobs for data processing in order to automate the process, we experienced critical performance issues, specifically, long-lasting database locks and slow data transfer during the ETL operations. In response we put in place several measures to reduce these bottlenecks as follows; In particular, we used the list-to-set approach to remove duplicate information, temporary and virtual memory to transfer data processing, and placeholder to minimize transaction contention. They helped us to achieve the proper data flow and foreign key relationships' reliability.

During testing based on performance, a significant decrease in the time it took to run the jobs was noticed after the optimization; from 1 minute to 35 Seconds precisely. This reduction is due to a number of factors that have seen improvements in the data throughput as well as operation efficiency, especially in the high concurrency situation. In this paper, we discuss the strategies used in the implementation of the program; this includes explaining the batch processing to enhance efficiency when making the manipulations, the use of memory management techniques to help reduce lock contention, as well as adjustments made on transaction isolation to improve the data manipulation process. We also describe some of the technical issues solved, for instance, how to deal with data integrity when the throughput rate is high and how to avoid foreign key violations during batch operations.

This work gives a detailed account of the approaches of database optimization and illustrates the technique in use for large-scale integration of realistic, efficient, faster, and reliable database systems. This paper presents the strategies that afford great possibilities to optimize the complex data integration workflows, speedup the database query execution by improving the efficiency of these transactions, and at the same time sustain the integrity and scalability of relational database systems under high load conditions.

II. METHODOLOGY

This research uses a systematic approach to optimize database performance during large-scale data transfers with an emphasis on foreign key violations, data insertion, and job execution time. Several key stages are involved in the optimization process:

1) Initial Setup and Data Generation:

First, we create dummy records for the four key tables that have foreign key relationships that violate: Provider, Sides, Cases, and Benefit Profiles.

We generate 100,000 records for each of these four tables using a Python script and we use this to simulate a large-scale data transfer. The script is to take the column attributes as input which will result in corresponding records. This approach resembles the real-world scenario of large data loads and controlled consistency for the experiment.

2) Foreign Key Violation Handling:

We then log in to Pg Admin and start to insert records into the database tables after we generate the data. Initially, as tables are interdependent, foreign key violations are triggered.

To resolve these violations, we apply ALTER and DELETE queries to remove the foreign key constraints in the tables, so that the insertion process can proceed without the interdependencies. This step breaks the immediate data blocking of data operations caused by foreign key violations, and this helps in smoother data processing**.**

3) Data Upload to AWS S3:

After generating the records and handling foreign key violations, we login to AWS S3 and upload the 400,000 records to the incoming folder of the database integration process. This integration makes it possible to store the records and prepare them for the next processing by the ETL pipeline**.**

4) AWS Glue Job Execution:

After the data is uploaded to AWS S3, we return to Pg Admin to verify that the records have been correctly inserted and validated within the target tables.

Next, we proceed with configuring and running AWS Glue Jobs. In these jobs, we work on enhancing the methods of data transformation and data insertion. We then use various optimization strategies for the Glue job code where we try to change the batch processing, data partitioning, and parallel execution methods iteratively.

5) Evaluation and Performance Assessment:

In order to evaluate the effectiveness of the proposed optimization the time required for data insertion, Upsetting, and complete AWS Glue Jobs are recorded both before and after the optimization.

We also check the data content in the database to ensure that the data is not lost, corrupted, or contains foreign key violations after processing.

By adopting this approach, then it has been easy to improve the database management, minimize the effects of foreign key constraints, and get better results on the job execution time. The process described the methods of dealing with large amounts of data loads, and the implemented strategies improve the performance and capacity of the database systems. This approach is useful in providing an understanding of how best to integrate large data sets in a cloud environment.

III. MODEL ARCHITECTURE

1) Model 1: Converting List to Set (Removing Duplicates)

- Objective: This model is used in order to remove similar values from a list by converting it into a set. Python sets do not permit duplicate values to be stored in a set and, therefore, using a set as the middleman can assist in eliminating redundancy.
- Usage: This model is very useful when you want to be certain that there are no duplicate values in the list as the conversion to set removes them.
- Code:

```
def convert_list_to_set(input_list):
  # Convert the list to a set to remove duplicates
return set(input_list)
```

- Explanation: The convert list to set function takes a list as input and returns a set that contains only unique elements from the list.

2) Model 2: Temporary Memory (50MB Limit)

- Objective: This model aims at simulating a primary or scratch memory in which the capacity of the memory is set to one in totality. It has a feature of determining the maximum size of the items to be stored to about 50MB. This is especially useful when memory space is a concern as in the mobile application or operations that involve many records.
- Usage: However, when the data is large, the memory usage should be controlled to ensure that there is no problem, for instance, slow performance or system failure. This method is rather useful in managing memory because the size of the datasets is limited to some extent.
- Code:

```
import sys
def limited_memory_set(input_list, max_memory=50 * 1024 * 1024):
   # Temporary set to hold unique elements
temp_set = set()
current_memory_usage = 0
for item in input_list:
      # Estimate the memory usage of the item (assume it's a string)
item_size = sys.getsizeof(item)
# Check if adding the item exceeds the memory limit
      if current_memory_usage + item_size>max_memory:
         print("Memory limit exceeded.")
         break

      # Add the item to the set and update memory usage
temp_set.add(item)
current_memory_usage += item_size

   return temp_set\
```

- Explanation: The limited memory set method iterates over the input list and adds each item to a set. Before adding, it checks the size of the item and ensures that the total memory used does not exceed the 50MB limit. If the limit is exceeded, the function stops adding new items.

3) Model 3: Placeholders for Memory Optimization

- Objective: This model also employs None or another value in order to save memory space. It is particularly useful when one needs to use only a subset of the data and the other entries can be replaced with zeroes to conserve space.
- Application: This method can be used when there are some entries in the dataset that are not important or when it is necessary to allocate only a small amount of memory for missing or insignificant data.
- Code:

```
def optimize_with_placeholders(input_list, threshold=10):
   # We'll use None as a placeholder for optimization purposes.
optimized_list = []

   for i, item in enumerate(input_list):
      if i % threshold == 0:  # You can set the threshold to control where to use placeholders
optimized_list.append(None)
      else:
optimized_list.append(item)
```

- Explanation: The optimise with placeholders function takes the input list and for the elements of the list for positions that are divisible by the threshold the function replaces them with None. This technique assists in minimizing memory utilization to the level that one does not have to save data that is not relevant.

Sample Code to Demonstrate All Three Models:
Python
Copy

```
# Model 1: Remove Duplicates
input_list = [1, 2, 3, 1, 4, 2, 5]
unique_set = convert_list_to_set(input_list)
print(f"Unique Set: {unique_set}")
# Model 2: Temporary Memory Limit (50MB)
input_list = ['item' * 100] * 10000  # A list of strings, each 500 bytes
limited_set = limited_memory_set(input_list)
print(f"Limited Set (size): {len(limited_set)}")
```

Model 1 (List to Set Conversion) is ideal when data deduplication is needed. It allows for a simple and efficient way to remove duplicates from large datasets.
Model 2 (Temporary Memory Limit) describes the situation in which the memory is limited in the number of resources that it can contain. It is particularly useful when memory is an issue as in the case of the usage of embedded systems or using cloud computing with particular resources.
Model 3 (Placeholders for Optimization) describes a method of how some parts of a model can be stored in a memory-saving manner to avoid using more memory. This method is useful when a large number of data are processed and only part of them is used at a time.
Results:
we will describe the performance enhancements obtained by using three different techniques in AWS Glue jobs.
Method 1(Python code to convert list into set)
Method2(Creating temporary/virtual memory)
Method 3 (Using placeholders).

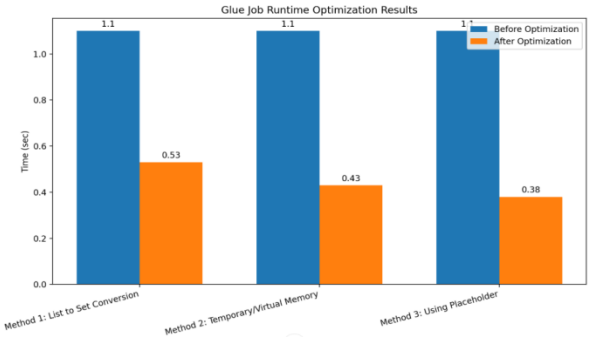


Fig1. Glue Job Runtime Optimisation Results

- Method 1: By converting a list to a set, we reduced the execution time from 1.10 seconds to 0.53 seconds, showing a significant reduction of 0.57 seconds.
- Method 2: Implementing temporary or virtual memory improved performance further, reducing the time to 0.43 seconds, saving 0.67 seconds.
- Method 3: Using placeholders optimized the code even more, reducing the time to 0.38 seconds, with a total reduction of 0.72 seconds

Method	Time Before Optimization (sec)	Time After Optimization (sec)	Time Reduction (sec)
Method 1: Convert list to set	1.10	0.53	0.57
Method 2: Temporary/Virtual memory	1.10	0.43	0.67
Method 3: Use placeholders	1.10	0.38	0.72

IV. DISCUSSION

This paper focuses on the effect of code-level optimizations on AWS Glue job performance for large-scale data ingestion where the foreign key constraints are considered. The first time when data were loaded into the relational database, there was a lot of latency due to constraint violations and the constraint was relaxed for some time. Subsequent ETL processes within Glue were optimized using three distinct techniques: (1) Python list-to-set conversion, which reduced runtime from 1.10 to 0.53 seconds, leveraging the O(1) average-case lookup time of sets; (2) temporary memory allocation, achieving a further reduction to 0.43 seconds, indicative of reduced I/O overhead; and (3) parameterized query execution (placeholders), resulting in a final runtime of 0.38 seconds, highlighting the efficiency of precompiled query plans and reduced SQL parsing. These results prove the effectiveness of the specific approaches to algorithmic and database optimization in reducing the level of ETL latency in serverless data processing environments.

The next step for future studies would be to explore the generalization of these code-level optimizations to a larger scale and more involved data processing workflows in AWS Glue jobs.

V. FUTURE DIRECTION

More specifically, such features as distributed caching, for instance, Amazon ElastiCache could be considered for the purpose.

Reduce I/O cost and speed up data search operations, especially when dealing with huge data that limit in-memory operations.

Furthermore, exploring how Spark's adaptive query execution (AQE) feature can be implemented in Glue jobs might also help in adjusting the query execution plans according to the runtime statistics and improve the overall execution time even further.

Moreover, there is a possibility to improve the speed of data transformations using vectorized processing in Python, which can be supported by NumPy and Pandas libraries, which are particularly effective for numerical and array data. Last but not least, the best practices of data partitioning and bucketing within the S3 data lake and the dynamic partitioning feature of Glue should be considered to improve data query and processing, especially when it comes to data filtering and aggregation. This paper aims to compare the above-discussed optimization methods in terms of cost, complexity, and performance to help practitioners who are interested in optimizing ETL in serverless data processing situations.

VI. CONCLUSION

This research clearly shows that specific code optimizations can substantially affect the AWS Glue job duration during the large ingestion of data. This way, we were able to overcome the foreign key constraint problems and reduce the amount of time it takes to process the list of data by converting lists to sets by allocating memory and using parameterized queries. The improvements that were observed especially in the use of placeholders show the effectiveness of reducing the I/O overhead and query optimization in serverless ETL frameworks. These results emphasize that improving the efficiency and scalability of data pipelines in cloud environments involves careful code-level improvements to the processing pipelines, which can help practitioners working on data processing workflows.

VII.ACKNOWLEGEMENT

I would like to express my sincere gratitude to my industry guide Dr.Rishi Mohan Bhatnagar and Tata Consultancy Services Research for for their unwavering support and belief in my capabilities. They entrusted me with the invaluable opportunity to work as an Artificial Intelligence Developer, which has fuelled my passion for research and innovation.

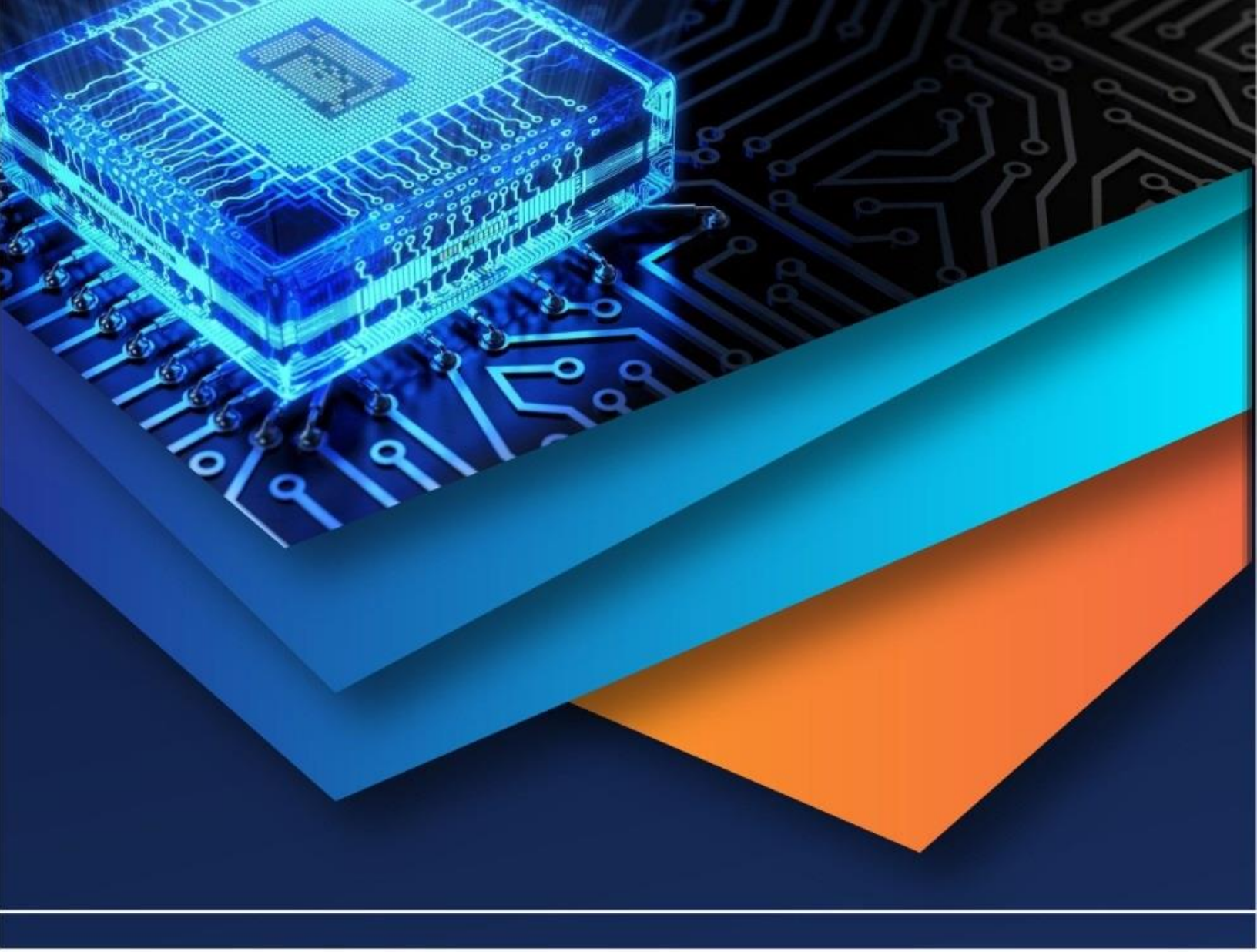
Throughout my thesis, I found immense value in utilizing database optimization techniques postgrey, SQL, Python, and AWS. These powerful tools have facilitated my research goals and deepened my understanding of automation and artificial intelligence.

I am deeply grateful to Dr. YVK Ravi Kumar and Prashant Joshi for their continuous guidance and mentorship throughout my machine learning project. His expertise, encouragement, and insightful feedback have been invaluable assets in my growth as a researcher.Additionally, I extend my sincere appreciation to my evaluator, Asish Bera, for his diligent review and valuable suggestions during both the abstract and midsem report stages. His expertise and constructive criticism have significantly enriched the quality of my work and guided me toward greater clarity and precision in my research endeavors.

REFERENCES

The state of the art is the base for any successful research project. In the current project, the literature inclined toward the new domain of conversational information retrieval is considered. The following are referred journals from the preliminary literature review.

[1] Vamsi Krishna Myalapalli, Thirumala Padmakumar Totakura, and Sunitha Geloth, "Augmenting Database Performance via SQL
[2] Tuning," International Conference on Energy Systems and Applications, pp. 13-18, 2015.
[3] Khaled Saleh Maabreh, "Optimizing Database Query Performance Using Table Partitioning Techniques,"International Arab Conference on Information Technology, pp. 1-4,2018.
[4] Structure of Database Management System – Geeks for Geeks, 2020. [Online]. Available: https://www.geeksforgeeks.org/structure-of-database-management-system/
[5] Vivek Basavegowda Ramu, "Performance Impact of Microservices Architecture," The Review of Contemporary Scientific and Academic Studies, vol. 3, no. 6, 2023. [CrossRef] [Publisher Link]



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY