Query-Level Deduplication for Optimizing Database Performance Using Python Code

Suvarna Lahanu Ghogare
Sir Visvesvaraya Institute of Technology, Sinnar

Abstract*—The increasing volume of data generated and processed by modern database management systems has created significant challenges in terms of storage utilization, query processing time, memory consumption, and data retrieval efficiency. Duplicate records in database query results introduce unnecessary redundancy, which can affect overall system performance and resource utilization. To address these challenges, this paper proposes a Hash-Based Intelligent Query-Level Data Deduplication (HIQLD) approach for identifying and eliminating duplicate records during database query processing. The proposed methodology can be integrated with relational database management systems such as MySQL and PostgreSQL. Its performance can be evaluated using parameters including query execution time, deduplication overhead, memory consumption, duplicate detection accuracy, and reduction in redundant records. Experimental comparison with conventional SQL techniques, particularly DISTINCT and GROUP BY, can help determine its effectiveness under different dataset sizes and duplication ratios. The proposed approach provides a foundation for intelligent and efficient query-result management in large-scale database environments.

***Index Terms*—Query-Level Deduplication, Database Optimization, SHA-256 Hashing, Duplicate Record Detection, SQL Query Processing, Memory Optimization, Data Redundancy.**

I. INTRODUCTION

The rapid growth of information technology, cloud computing, big data applications, and online services has resulted in the generation of enormous volumes of data. Database Management Systems (DBMS) play an important role in storing, managing, retrieving, and processing this information. Relational databases are widely used in domains such as education, healthcare, banking, e-commerce, and business management. With increasing database sizes and query execution requirements, efficient data management and query optimization have become significant challenges. One of the major challenges in database management is data redundancy, which refers to the repeated occurrence of identical or logically equivalent records. Duplicate records may arise due to repeated data insertion, data integration, database synchronization, and multiple table operations. Their presence in query results can increase processing overhead, consume unnecessary memory resources, and reduce data retrieval efficiency. Therefore, identifying and eliminating duplicate records is essential for improving database performance. Traditional relational database systems provide techniques such as SQL DISTINCT and GROUP BY for duplicate elimination. Although these techniques are effective, their performance may depend on dataset size, query complexity, indexing mechanisms, and available memory. Large-scale database environments may introduce additional computational overhead during duplicate identification and elimination. To address these challenges, this research proposes an Adaptive Hash-Based Intelligent Query-Level Deduplication (AHIQLD) approach for efficient database query processing. The proposed approach identifies duplicate records using attribute-based hash generation and hash-table-based lookup mechanisms. Initially, a user-defined SQL query is executed to retrieve records. Selected attributes are processed using a cryptographic hash function, such as SHA-256, to generate hash identifiers. These identifiers are maintained in a hash table to detect previously processed records. Unique records are retained, while potential duplicates are verified against original attribute values and eliminated. The proposed framework can be implemented using Python with relational database platforms such as MySQL or PostgreSQL. Its performance can be evaluated against conventional SQL techniques using parameters such

as query execution time, memory consumption, deduplication overhead, and duplicate detection accuracy.

II.RELATED WORK

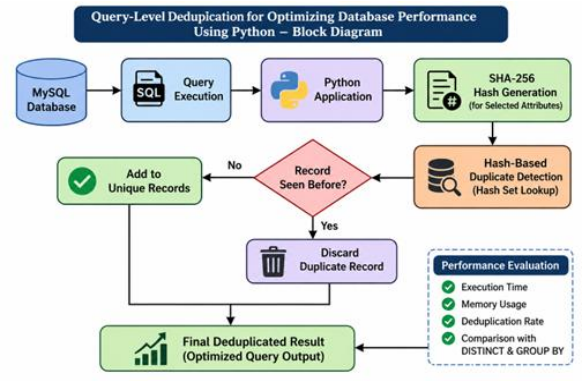
Research and engineering paradigms in database query optimization, application-level caching, and data stream deduplication provide the foundational context for query-level deduplication. Existing approaches generally fall into four primary categories: database engine-level aggregation, concurrent request coalescing, cryptographic stream deduplication, and proxy-based caching. Several researchers have studied data deduplication, duplicate record detection, hash-based techniques, and database query optimization. These studies provide a foundation for developing the proposed Query-Level Deduplication System Using Python and SHA-256.[1]A paper published in IJARIIE-ISSN (O)-2395-4396 Vol-3 Issue-4 2017 S. Ghogare 1 , Prof. S. K. Sonkar 2,[1]M.E., Computer Engineering Department, Maharashtra, India .Assistant professor, computer engineering department, Maharashtra, India" secure data auditing and deduplicating data with multiuser cloud environment" There are several features introduced in proposed system such as, data encryption, data duplication check and periodic verification of data. For better efficiency map-reduce functionality is used. As a part of contribution, system is design and developed with multiuser environment. User can share files with the other users which is uploaded by them to cloud server. With an experimental set up proposed system proves an efficiency of system in terms of Data uploading, tag generation and deduplication check time of execution.[2] L. Ao, J. Shu, and M. Li, "Data Deduplication Techniques," Journal of Software, vol. 21, no. 5, pp. 916–929, 2010. Data deduplication technologies can be divided into two categories: a) identical data detection techniques, and b) similar data detection and encoding techniques. This paper presents a systematic survey on these two categories of data deduplication technologies and analyzes their advantages and disadvantages. Besides, since data deduplication technologies can affect the reliability and performance of storage systems, this paper makes several conclusions as follows: a) How to mine data characteristic information in data deduplication has not been completely solved, and how to use data characteristic information to effectively eliminate duplicate data also needs further study; b) From the perspective of storage system design, it still needs further study how to introduce proper mechanisms to overcome the reliability limitations of data deduplication techniques and reduce the additional system overheads caused by data deduplication techniques.[3]P. Christen, "A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication," IEEE Transactions on Knowledge and Data Engineering, vol. 24, no. 9, pp. 1537–1555, 2012.Record linkage is the process of matching records from several databases that refer to the same entities. When applied on a single database, this process is known as deduplication. Increasingly, matched data are becoming important in many application areas, because they can contain information that is not available otherwise, or that is too costly to acquire. Removing duplicate records in a single database is a crucial step in the data cleaning process, because duplicates can severely influence the outcomes of any subsequent data processing or data mining. With the increasing size of today's databases, the complexity of the matching process becomes one of the major challenges for record linkage and deduplication. In recent years, various indexing techniques have been developed for record linkage and deduplication. They are aimed at reducing the number of record pairs to be compared in the matching process by removing obvious nonmatching pairs, while at the same time maintaining high matching quality. .[4] W. Xia et al., "A Comprehensive Study of the Past, Present, and Future of Data Deduplication," Proceedings of the IEEE, vol. 104, no. 9, pp. 1681–1710, 2016.Data deduplication, an efficient approach to data reduction, has gained increasing attention and popularity in large-scale storage systems due to the explosive growth of digital data. It eliminates redundant data at the file or subfile level and identifies duplicate content by its cryptographically secure hash signature (i.e., collision-resistant fingerprint), which is shown to be much more computationally efficient than the traditional compression approaches in large-scale storage systems.

III. PROBLEM STATEMENT

To develop a Python-based Query-Level Deduplication system that identifies and eliminates

duplicate records from database query results using SHA-256 hashing and hash-table techniques. The system aims to reduce data redundancy, optimize memory utilization, and improve query-result processing efficiency compared with traditional SQL techniques such as DISTINCT and GROUP BY.

IV. PPROPOSED SYSTEM



Working of each module:

1. MySQL Database

Serves incoming SELECT requests by retrieving raw data blocks without imposing resource-heavy deduplication clauses (such as DISTINCT or GROUP BY) at the database engine level.

2. Query Execution

Bypasses database-side sorting and temporary disk spillages, fetching raw query results directly into memory streams to preserve database CPU and buffer pool capacity.

3. Python Application

Ingests raw result sets from the database driver and coordinates row-by-row streaming, feature attribute extraction, hashing, and duplicate evaluation in application memory.

4. SHA-256 Hash Generation (for Selected Attributes)

Serializes the specified attribute values into a canonical string representation and applies the SHA-256 cryptographic algorithm to produce a fixed 256-bit hash value representing that specific record state.

5. Hash-Based Duplicate Detection (Hash Set Lookup)

Performs an $O (1) $ constant-time lookup of the computed SHA-256 hash against an in-memory hash set storing previously encountered hashes.

6. Record Seen Before? (Decision Block)

- No Path: Triggered when the hash key is not found in the lookup set (indicating a new, unique row).
- Yes Path: Triggered when the hash key matches an existing entry in the lookup set (indicating a duplicate row).

7. Add to Unique Records

Appends the unique row to the output buffer array and inserts its SHA-256 hash into the in-memory hash set so subsequent matching records can be identified immediately.

8. Discard Duplicate Record

Drops duplicate rows immediately from memory, preventing memory bloat and eliminating redundant network/serialization overhead.

9. Performance Evaluation

Continuously measures key performance metrics:

- Execution Time: Total wall-clock time from query submission to final result generation.
- Memory Usage: Heap memory allocation for the hash set vs. database temporary tables.
- Deduplication Rate: Percentage of duplicate records removed from the dataset.
- Comparison: Direct performance benchmarking against native SQL DISTINCT and GROUP BY queries.

10. Final Deduplicated Result (Optimized Query Output)

Outputs the fully deduplicated record set to downstream client applications, microservices, or visualization dashboards with minimal latency and optimized memory footprints.

V. METHODOLOGY

Objective

To design and implement an efficient Python-based query-level deduplication framework that identifies and eliminates duplicate records from database query results using SHA-256 fingerprinting and $O (1) $ hash table lookups, offloading resource-intensive filtering operations from the database engine to the application layer.

System overview

The system generates SHA-256 hash values for selected attributes and uses a hash set to identify duplicate records. Unique records are retained, while duplicate records are verified and eliminated. Finally, the system generates deduplicated query results and evaluates performance based on execution time, memory consumption, and deduplication efficiency

Step-by-Step Procedure

The proposed Query-Level Deduplication System using Python and MySQL follows these steps:

1. Database Creation: Create a MySQL database and table containing sample records, including duplicate entries.
2. Insert Records: Insert unique and duplicate records into the database for testing.
3. Database Connectivity: Establish a connection between Python and MySQL using the mysql-connector library.
4. Execute SQL Query: Execute the required SQL query and retrieve records from the database.
5. Select Attributes: Select the attributes (such as Name, Department, or Email) that will be used to identify duplicate records.
6. Generate SHA-256 Hash: Convert the selected attribute values into a consistent format and generate a unique SHA-256 fingerprint for each record.
7. Check Duplicate Records: Maintain a Python hash set to check whether the generated fingerprint already exists.
8. Verify Duplicate Records: If a fingerprint already exists, compare the original selected attribute values to confirm whether the record is a duplicate.
9. Remove Duplicate Records: Retain new unique records and discard confirmed duplicate records.
10. Generate Final Output: Display the deduplicated query results containing only unique records.
11. Performance Evaluation: Compare the proposed Python approach with SQL DISTINCT and GROUP BY using:

- Query execution time
- Deduplication processing time
- Memory consumption
- Number of duplicate records removed
- Deduplication ratio

12. *Display Results: Present the final results using Python and generate performance graphs using Matplotlib.*

VI. ALGORITHM

Algorithm: Query-Level Deduplication Using Python and SHA-256

Input: Database query result containing unique and duplicate records.

Output: Deduplicated query result containing only unique records.

Step-by-Step Algorithm

1. START
2. Connect Python application to the MySQL database.
3. Execute the SQL query and retrieve the result records.
4. Initialize an empty hash set Hash_Set and an empty list Unique_Records.
5. Select the attributes required for duplicate identification.
6. Read each record from the query result.
7. Extract the selected attribute values from the current record.
8. Convert the selected values into a consistent string or byte representation.
9. Generate the SHA-256 hash of the selected attribute values.
10. Check whether the generated hash exists in Hash_Set.
11. If the hash does not exist:

- Add the hash to Hash_Set.
- Add the current record to Unique_Records.

12. If the hash already exists:

- Compare the original selected attribute values.
- If the values match, identify the record as a duplicate and discard it.
- If the values differ, handle it as a hash collision and retain the distinct record.

13. Repeat Steps 6–12 until all records are processed.
14. Display the Unique_Records.
15. Calculate execution time, memory consumption, and deduplication ratio.
16. Compare the results with SQL DISTINCT and GROUP BY.
17. STOP

Step-by-Step Working

Step 1: Database Connection

Python establishes a connection with the MySQL database using the mysql-connector library.

Step 2: Query Execution

The SQL query is executed, and the database returns records containing both unique and duplicate entries.

Step 3: Initialization

Two data structures are initialized:

- Hash_Set: Stores hash values of processed records.
- Unique_Records: Stores the records identified as unique.

Step 4: Attribute Selection

The system selects specific attributes for duplicate identification, such as Student Name and Department.

Step 5: Hash Generation

The selected attribute values are converted into a consistent format. Python generates a SHA-256 hash for each record using the hashlib library.

Step 6: Duplicate Detection

The generated hash is checked against the existing hash values in Hash_Set.

- If the hash is not present, the record is considered new and added to Unique_Records.
- If the hash is already present, the original attribute values are compared to confirm whether the record is a duplicate.

Step 7: Duplicate Removal

If the selected attribute values match, the duplicate record is discarded. If they differ, the record is retained as a distinct record, ensuring correct collision handling.

Step 8: Final Result Generation

After processing all records, the system displays the final result containing only unique records according to the selected attributes.

Step 9: Performance Evaluation

The system calculates execution time, memory consumption, duplicate count, and deduplication ratio. The results are compared with SQL DISTINCT and GROUP BY.

VII. MATHEMATICAL MODEL

1. Input Parameters

Let the database query return a set of records:

$$R = \{r_1,r_2,r_3,\ldots,r_n\}$$

Where:

- R = Original query result.
- n = Total number of records.
- r_i = Individual record in the query result.

Let A represent the selected attributes used for duplicate identification.

For example:

$$A = \{\text{Name, Department}\}$$

2. Attribute Extraction

For each record r_i, extract the selected attributes:

$$K_i = A(r_i)$$

Where K_i represents the selected attribute values of record r_i.

3. Hash Generation

Generate the SHA-256 hash value for each selected attribute key:

$$H_i = SHA256(Encode(K_i))$$

Where:

- H_i = Hash value of the selected attributes.
- Encode = Consistent conversion of attribute values into bytes.
- SHA-256 = Cryptographic hash function.

4. Duplicate Detection

Two records are considered duplicates when their selected attribute values are identical:

$$Duplicate(r_i,r_j)= \begin{cases} 1, & K_i=K_j \\ 0, & K_i\neq K_j \end{cases}$$

The hash values are used for efficient candidate identification. Original attribute values are compared to confirm duplicates and handle possible hash collisions.

5. Unique and Duplicate Records

Let:

- U = Set of unique records.
- D = Set of duplicate records.

The original query result can be represented as:

$$R = U \cup D$$

The final deduplicated output is:

$$R_{final}=U$$

And:
$$|R|=|U|+|D|$$

6. Deduplication Ratio
The percentage of duplicate records removed from the query result is calculated as:
$$DR=\frac{|D|}{|R|}\times100$$

7. Memory Reduction in Result Processing
Let:
- M_o = Memory required to hold the original result records.
- M_f = Memory required to hold the final unique records.

Assuming equal-sized records:
$$Memory\ Reduction(\%)= \frac{M_o-M_f}{M_o}\times100$$
This measures the reduction in memory required for the retained result records, not necessarily the total memory consumed by the Python deduplication process.

8. Execution Time
The total processing time of the proposed approach is:
$$T_{total}=T_{query}+T_{dedup}$$
Where:
- T_{query} = Database query execution and retrieval time.
- T_{dedup} = Python-based duplicate detection and removal time.

The proposed approach can be compared with SQL DISTINCT and GROUP BY using their measured execution times.

9. Objective Function
The primary objective is to maximize duplicate elimination while preserving the correctness of unique records.
$$\boxed{\text{Maximize } |D|}$$
Subject to:
$$R=U\cup D$$
$$U\cap D=\varnothing$$
$$K_i=K_j \Rightarrow r_i,r_j \text{ belong to the same duplicate group}$$

VIII. EXPERIMENTAL SETUP

The experimental setup is designed to implement and evaluate the proposed Query-Level Deduplication System Using Python and SHA-256. The system uses Python for duplicate detection and MySQL for database management.

1. Hardware Requirements

Component	Specification
Processor	Intel Core i3 or above
RAM	Minimum 4 GB
Hard Disk	256 GB or above
Operating System	Windows / Linux
System Architecture	64-bit

2. Software Requirements

Software / Technology	Purpose
Python 3.x	Implementation of deduplication algorithm
MySQL	Database creation and record management
MySQL Connector	Python–MySQL connectivity
SHA-256 (hashlib)	Hash generation
Pandas	Data handling and analysis
Matplotlib	Performance graphs
VS Code / PyCharm	Python development environment

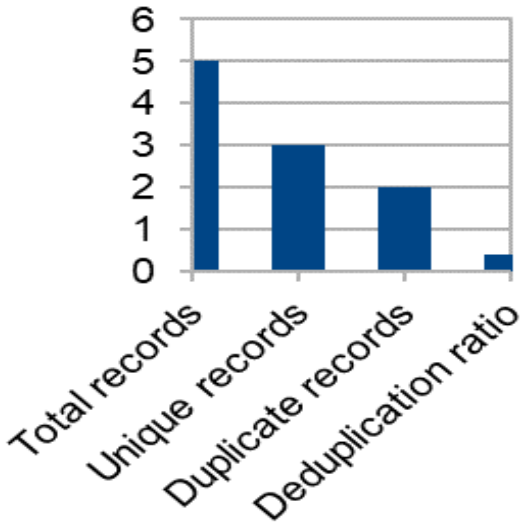
3. Dataset
A MySQL database named Deduplication_DB is created to store student records.

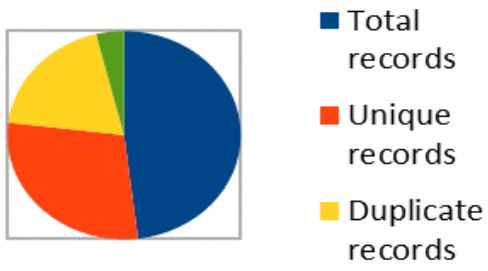
IX. RESULT TABLES AND DISCUSSION

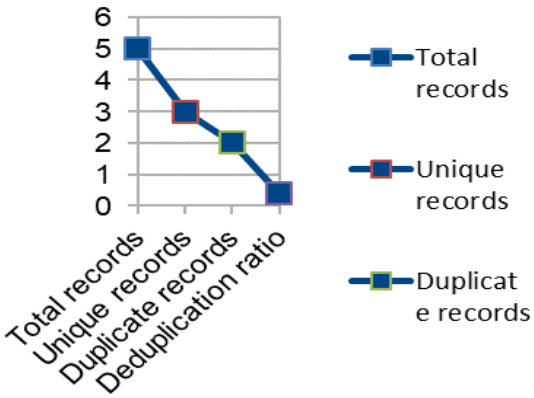
Student ID	Name	Department
101	Amit	IT
102	Sneha	Computer
103	Amit	IT
104	Rahul	IT
105	Sneha	Computer

Student ID	Name	Department	Status
101	Amit	IT	Unique
102	a	Computer	Unique
103	Amit	IT	Duplicate
104	Rahul	IT	Unique
105	a	Computer	Duplicate

Total records	record s	Duplicat e records	Deduplic ation ratio
5	3	2	40.00%







Discussion of Results:

1. Duplicate Detection: The SHA-256-based approach identifies duplicate records by generating hash values from selected attributes. Original attribute values are verified to ensure correct duplicate identification.
2. Dataset Size: As the number of records increases, the number of hash calculations and lookup operations also increases. Therefore, processing time and memory consumption should be evaluated for different dataset sizes.
3. Deduplication Efficiency: The deduplication ratio indicates the percentage of duplicate records removed from the original query result. A higher ratio means more redundant records are eliminated.
4. Memory Utilization: Removing duplicate records reduces the size of the retained result set. However, the Python hash set and processing structures also consume memory and must be considered.
5. Comparison with SQL Techniques: SQL DISTINCT and GROUP BY perform duplicate elimination within the database engine. The proposed Python approach performs deduplication after retrieving records. Therefore, its performance depends on dataset size, network transfer, hash-generation overhead, and Python memory usage.

Screen-shot of reading:

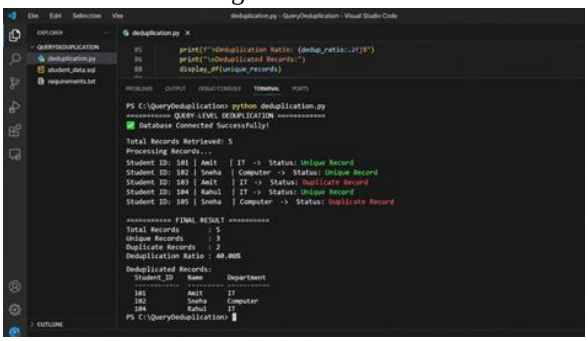


Figure: Screenshot of Reading and Execution Output – Query-Level Deduplication Using Python and SHA-256

X. CONCLUSIONS

The proposed project uses Python, MySQL, and SHA-256 hashing to identify and remove duplicate records from database query results. The system retains unique records and eliminates redundant entries using hash-set operations. In the experimental example, 5 records were processed, 3 unique records were retained, and 2 duplicates were removed, achieving a 40% deduplication ratio. The approach provides an effective method for query-result deduplication and can be extended to large-scale databases and cloud-based applications.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who contributed to the successful completion of this paper for valuable guidance, continuous support, and encouragement throughout this project. I am thankful to the Head of the Department and the Principal for providing the necessary facilities and resources. I also thank all faculty members for their valuable suggestions and support. Finally, I express my heartfelt gratitude to my family for their constant encouragement and motivation in completing this project successfully.

REFERENCES

[1] S. Ghogare and S. K. Sonkar, "Secure data auditing and deduplicating data with multiuser cloud environment," vol. 3, no. 4, 2017.

[2] L. Ao, J. Shu, and M. Li, "Data deduplication techniques," *Journal of Software*, vol. 21, no. 5, pp. 916–929, 2010.

[3] P. Christen, "A survey of indexing techniques for scalable record linkage and deduplication," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 9, pp. 1537–1555, 2012.

[4] W. Xia et al., "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[5] S. T. Ahmed and L. E. George, "Lightweight hash-based de-duplication system using the self-detection of most repeated patterns as chunks divisors," *Journal of King Saud University—Computer and Information Sciences*, vol. 34, no. 7, pp. 4669–4678, 2022, doi: 10.1016/j.jksuci.2021.04.005.

[6] T. Do, G. Graefe, and J. F. Naughton, "Efficient sorting, duplicate removal, grouping, and aggregation," ACM Transactions on Database Systems, vol. 47, no. 4, 2022, doi: 10.1145/3568027.

[7] D. Viji and S. Revathy, "Hash-indexing block-based deduplication algorithm for reducing storage in the cloud," Computer Systems Science and Engineering, vol. 46, no. 1, pp. 27–42, 2023, doi: 10.32604/csse.2023.030259.

[8] W. Andrzejewski, B. Bębel, P. Boiński, and R. Wrembel, "On tuning parameters guiding similarity computations in a data deduplication pipeline for customers records: Experience from a R&D project," Information Systems, vol. 121, Art. no. 102323, 2024, doi: 10.1016/j.is.2023.102323.

[9] G. Wagner, "BibDedupe: An open-source Python library for bibliographic record deduplication," *Journal of Open-Source Software*, vol. 9, no. 97, Art. no. 6318, 2024, doi: 10.21105/joss.06318.