Journal of Data Analysis and Information Processing, 2025, 13(1), 46-65
https://www.scirp.org/journal/jdaip
ISSN Online: 2327-7203
ISSN Print: 2327-7211

Tailored Partitioning for Healthcare Big Data: A Novel Technique for Efficient Data Management and Hash Retrieval in RDBMS Relational Architectures

Ehsan Soltanmohammadi, Neset Hikmet, Dilek Akgun

Department of Integrated Information Technology, University of South Carolina, Columbia, USA
Email: ehsans@email.sc.edu, nhikmet@cec.sc.edu, akgun@mailbox.sc.edu

How to cite this paper: Soltanmohammadi, E., Hikmet, N. and Akgun, D. (2025) Tailored Partitioning for Healthcare Big Data: A Novel Technique for Efficient Data Management and Hash Retrieval in RDBMS Relational Architectures. *Journal of Data Analysis and Information Processing*, **13**, 46-65.
https://doi.org/10.4236/jdaip.2025.131003

Received: December 1, 2024
Accepted: January 19, 2025
Published: January 22, 2025

Abstract

Efficient data management in healthcare is essential for providing timely and accurate patient care, yet traditional partitioning methods in relational databases often struggle with the high volume, heterogeneity, and regulatory complexity of healthcare data. This research introduces a tailored partitioning strategy leveraging the MD5 hashing algorithm to enhance data insertion, query performance, and load balancing in healthcare systems. By applying a consistent hash function to patient IDs, our approach achieves uniform distribution of records across partitions, optimizing retrieval paths and reducing access latency while ensuring data integrity and compliance. We evaluated the method through experiments focusing on partitioning efficiency, scalability, and fault tolerance. The partitioning efficiency analysis compared our MD5-based approach with standard round-robin methods, measuring insertion times, query latency, and data distribution balance. Scalability tests assessed system performance across increasing dataset sizes and varying partition counts, while fault tolerance experiments examined data integrity and retrieval performance under simulated partition failures. The experimental results demonstrate that the MD5-based partitioning strategy significantly reduces query retrieval times by optimizing data access patterns, achieving up to X% better performance compared to round-robin methods. It also scales effectively with larger datasets, maintaining low latency and ensuring robust resilience under failure scenarios. This novel approach offers a scalable, efficient, and fault-tolerant solution for healthcare systems, facilitating faster clinical decision-making and improved patient care in complex data environments.

Keywords

Healthcare Data Partitioning, Relational Database Management Systems (RDBMS), Big Data Management, Load Balance, Query Performance Improvement, Data Integrity and Fault Tolerance, Efficient, Big Data in Healthcare, Dynamic Data Distribution, Healthcare Information Systems, Partitioning Algorithms, Performance Evaluation in Databases

1. Introduction

We are immersed in a data-rich world where vast amounts of data are generated from various aspects of our lives, including work, science, health, and social and physical activities. One of the most impactful areas is healthcare, where big data brings significant changes. Data sources in healthcare are diverse, encompassing clinical trials, electronic health records (EHR), medical imaging, payer records, patient registries and databases, large biological datasets, healthcare tools, smartphone apps, mobile phones, wearable devices, and medical equipment. The exponential growth of healthcare data from these several sources, along with the rise of big data analytics and computational technologies, offers an invaluable opportunity to raise the standard and effectiveness of healthcare [1] [2]. Additionally, big data-driven systems provide scalable and adaptable solutions, allowing healthcare organizations to effectively store, manage, and access large volumes of patient data [3].

In today's digital age, the healthcare industry is continuously striving to enhance the security and efficiency of its data management practices. Data security and privacy have always been at the forefront of healthcare data management since healthcare data is inherently sensitive, containing personal and confidential information about patients' medical histories, diagnoses, treatments, and other personal data. Furthermore, the strict regulations surrounding this data, such as the Health Insurance Portability and Accountability Act (HIPAA) in the U.S. [4] and the General Data Protection Regulation (GDPR) in Europe [5] mandate stringent controls on access and handling of patient information, demanding that healthcare organizations take steps to ensure the security and confidentiality of this data. With an increasing reliance on EHR, medical imaging, and health monitoring systems, healthcare data volumes have grown exponentially. As digitization expands and the population grows, the imperative to manage and protect sensitive patient information has never been more critical [6] [7].

The management of healthcare data has long been a challenge due to the sensitive nature of the information, regulatory requirements, and the sheer volume of data generated. However, traditional database management systems are lagging behind in the complex needs of healthcare organizations [8], where rapid access to data and secure handling are paramount. For example, previous research suggests that the inevitable adoption of big data techniques in healthcare aims to en-

hance delivery quality, yet existing data management architectures still encounter challenges in preventing emergency cases [9]. Despite significant advances in healthcare through big data research, several challenges still need to be addressed including security, privacy, and performance [10] [11]. In this regard, several studies suggest that effectively managing and analyzing the massive volumes of diverse data generated rapidly, particularly in biomedical research and healthcare, necessitates advanced data management software solutions and high-end computational power [12] [13]. Consequently, innovative solutions are needed to meet these evolving demands.

Furthermore, effective database management solutions, particularly in the context of relational database management systems (RDBMS), have become critical as the most common form of database used in healthcare is the relational database [14]. Previous research states that relational database approaches have been widely used to archive and integrate clinical and genomics data for translational research, reflecting the specific data challenges and technological capabilities of their time [15]. Furthermore, as healthcare systems increasingly rely on RDBMS to store and retrieve patient data, the surge in data volume has made it critical to implement robust and efficient data management strategies that prioritize security, privacy, and accessibility. For example, [16] discussed that effective big data management and analysis in healthcare can enhance patient care, reduce medical errors, and enable personalized treatment by processing large datasets from numerous patients. One critical aspect of effective data management is *data partitioning*, a technique that involves dividing large datasets into smaller, more manageable subsets [17]. In the context of healthcare, data partitioning becomes even more crucial due to the high volume, complexity, and sensitivity of patient information, along with stringent regulatory requirements. However, conventional data partitioning methods, designed for general-purpose databases, often fail to meet the specific needs of healthcare data. These methods struggle with the efficient management of large datasets, often leading to performance bottlenecks and increased computational costs, and they typically lack advanced security measures. To address these challenges, specialized partitioning approaches that integrate both performance optimizations and security are needed, such as cryptographic techniques, to maintain the confidentiality and integrity of patient data [18].

Healthcare data must be managed in ways that ensure security, efficient access, scalability, and regulatory compliance. The increasing volume and complexity of healthcare data require innovative solutions to partition and secure patient information effectively. Traditional partitioning methods in RDMS have limitations when applied to sensitive data, necessitating the development of new techniques that balance performance, security, and privacy in healthcare contexts. Moreover, as healthcare data grows, partitioning must scale efficiently and support compliance with regulations like HIPAA and GDPR, enabling granular control over data access and usage. In recent years, there has been a growing interest in developing specialized data partitioning techniques tailored to the specific needs of healthcare

data management. Cryptographic methods, such as encryption and hashing, have been integrated into database management systems to safeguard sensitive information [19]. Recent studies have explored the combination of cryptographic methods with database partitioning strategies to enhance both performance and security [20] [21]. These techniques often involve combining cryptographic hashing with database partitioning to enhance security and performance.

As healthcare data continues to grow in volume and complexity, the development of tailored partitioning strategies that address both security and performance concerns will be crucial to ensuring the effective management of this sensitive information. Our proposed partitioning methodology, which integrates the cryptographic techniques like hashing with advanced partitioning strategies, holds significant promise for overcoming these challenges. By hashing sensitive data, it can be stored in a secure and anonymized format, reducing the risk of unauthorized access and data breaches. Our research offers a valuable contribution to the field of database systems and has the potential to revolutionize the way healthcare organizations manage and utilize their data by addressing the unique challenges of healthcare data management.

Related Work

The challenges associated with managing healthcare data within relational databases have been widely recognized, with various approaches proposed to enhance data partitioning, retrieval, and security. Traditional partitioning techniques, such as range and hash-based partitioning, are the foundation of data management in relational database management systems (RDBMS). These approaches are commonly implemented to improve query performance and data access speeds by logically dividing tables into smaller, more manageable segments. However, they are often insufficient when applied to healthcare data, particularly due to the complexity and sensitivity of this type of data [22].

Range-based partitioning organizes data into partitions based on specific ranges of values, commonly used for ordered datasets like time-series data. For example, in healthcare, data might be partitioned based on patient admission dates or appointment times. While this approach is straightforward and effective for certain types of data, it is limited when dealing with diverse healthcare data that includes not only time-series information but also complex relational data such as patient demographics, medical histories, and treatment records. The rigid structure of range-based partitioning can also result in unbalanced partitions, which may lead to uneven query performance and storage inefficiencies [23].

Hash-based partitioning, on the other hand, divides data into partitions by applying a hash function to specific columns, such as patient ID numbers. This approach can provide a more uniform distribution of data across partitions, potentially balancing storage and computational load more effectively than range-based partitioning. Hash-based partitioning is particularly advantageous for environments with unpredictable data access patterns [24]. However, when used for

healthcare data, hash-based partitioning may still face significant challenges. The sheer volume and diversity of healthcare data, coupled with the need for secure hashing to protect patient information, can strain hash-based partitioning methods, reducing their efficiency and scalability.

Other partitioning methods, including **list-based** and **composite partitioning**, have also been explored in healthcare contexts. List-based partitioning allows data to be grouped based on discrete values, such as patient demographics or specific medical codes. Composite partitioning combines two or more partitioning methods, such as range and hash, to tailor data segmentation based on multiple criteria. These hybrid approaches can provide additional flexibility but are generally more complex to implement and maintain. Furthermore, they do not specifically address the unique security needs associated with healthcare data, particularly in terms of managing hashed data and maintaining compliance with stringent regulatory requirements [25].

Recent studies have introduced **dynamic and adaptive partitioning** strategies that adjust partition structures based on workload patterns and data changes over time. These approaches offer enhanced performance in scenarios with fluctuating data access patterns, such as those found in healthcare organizations. For example, some adaptive partitioning methods can detect query patterns and redistribute data accordingly to optimize access times. While promising, these strategies are computationally intensive and may incur significant overhead, limiting their scalability for large-scale healthcare applications.

Despite these advances, existing partitioning methods generally lack provisions for securely handling hashed data—a critical requirement in healthcare environments where data privacy is paramount. Moreover, traditional approaches often struggle with scalability when applied to the extensive, diverse, and sensitive datasets characteristic of healthcare. There is a clear need for a tailored approach that not only partitions healthcare data efficiently but also incorporates mechanisms for secure hash management within the database environment [26].

2. Proposed Methodology

2.1. Target Layers

This research introduces a tailored data partitioning approach for managing healthcare data in relational databases. Our methodology involves a two-layered mapping process designed to enhance security and optimize the retrieval of sensitive patient information. By applying a combination of MD5 hashing and segment-based partitioning, we achieve both data anonymization and efficient data access within the constraints of relational database management systems (RDBMS), specifically PostgreSQL.

Layer 1: Hashing Patient Identifiers for Data Security

The first layer of our methodology addresses the need for data security and privacy by transforming sensitive Personally Identifiable Information (PII), such as patient identifiers, into a secure format. This is accomplished through the MD5

hashing algorithm, which provides a consistent and non-reversible representation of the data [27].

1) Data Anonymization: When a patient identifier (e.g., Social Security Number or Patient ID) enters the system, it undergoes the MD5 hashing process, resulting in a 32-character hexadecimal string. This hashed value uniquely represents the original data without retaining any sensitive information.

2) Irreversible Transformation: The MD5 algorithm ensures that the transformation is one-way. This means the hashed identifier cannot be used to reconstruct the original PII, thereby preserving the privacy of sensitive healthcare data.

3) Uniqueness and Consistency: The hashed value is consistently unique for a given identifier, allowing each patient to maintain a unique representation in the database, regardless of when or how many times the hashing is applied.

This initial hashing layer secures patient information while preserving the uniqueness necessary for efficient data retrieval and management. By transforming each identifier into a hashed value, we create a consistent, anonymous representation of the data that is safe to store and manage in a high-security healthcare environment.

While MD5 hashing was used in the experiments for anonymization, we acknowledge that MD5 is not the most secure algorithm by modern standards due to vulnerabilities to collision attacks. However, for the purpose of this study, which focuses on the efficiency of the partitioning strategy, the choice of hashing algorithm has minimal impact on performance metrics such as query retrieval times, scalability, and fault tolerance. Other more secure hashing algorithms, such as SHA-256 or SHA-3 [28], could be used to further enhance data security in healthcare applications without affecting partitioning efficiency. We have updated the manuscript to reflect these alternatives, recognizing the role of the hashing algorithm in enhancing data security.

Layer 2: Mapping Hashed Identifiers to Specific Segments for Efficient Data Partitioning

Once the PII has been hashed, the second layer of our methodology maps these hashed outputs into discrete segments. This layer is crucial for efficient data partitioning, as it organizes data into easily accessible partitions within the database.

1) Segment-Based Partitioning: We map the hashed identifiers to specific segments in the database. To achieve this, the hashed string is first converted into an integer. Using a modulus operation with the total number of desired segments, we assign the integer result to a particular partition. For example, if we have defined 10 segments, each hashed value is mapped by taking hash_value mode 10, resulting in a range of integers that correspond to each segment.

2) Database Partitioning and Segment Assignment: In PostgreSQL, each segment corresponds to a partitioned table for managing patient records. By assigning each hashed identifier to a specific segment, we optimize data storage and retrieval, ensuring that the system only needs to access one partition for a particular record. This reduces query times and computational overhead, especially in sce-

narios with high data volumes.

3) Uniform Distribution and Load Balancing: The modulus operation evenly distributes hashed identifiers across the segments, ensuring balanced data distribution across partitions. This reduces the likelihood of bottlenecks in specific partitions, improving overall system performance.

4) Indexing and Performance Optimization: Each partition is indexed based on the mapped segment integer, enabling rapid data retrieval. When a query is executed, it uses the hash segment as a reference, allowing the system to immediately locate the relevant partition and minimize data access times.

The second layer of mapping is critical for scalability and efficiency. By directing each hashed identifier into a specific segment, we optimize the data retrieval process, ensuring that the system can handle high volumes of requests without significant degradation in performance.

2.2. Integrated Approach: Securing and Optimizing Healthcare Data Management

This two-layered approach offers several advantages for healthcare data management:

1) Data Security and Privacy Compliance: The MD5 hashing in the first layer ensures that sensitive patient data remains anonymous, meeting compliance standards for data protection regulations such as HIPAA. Since the data is irreversibly hashed, unauthorized access does not expose the original PII.

2) Optimized Data Retrieval: The segment-based partitioning in the second layer significantly reduces the time required to access records. Since each patient's hashed identifier is directed to a specific segment, the system only needs to query one partition, streamlining the retrieval process.

3) Scalability and Load Balancing: With consistent and balanced segment assignments, the database efficiently manages data load across partitions. This distribution allows the system to scale as data volumes grow, handling large datasets without performance issues.

In order to evaluate the effectiveness and practical advantages of the proposed MD5-based partitioning approach, a series of experiments were designed. These experiments focus on demonstrating the efficiency, scalability, and fault tolerance of the system, which are key aspects of data management in healthcare applications. Specifically, the following experiments were conducted: Partitioning Efficiency, Scalability, Fault Tolerance and Data Integrity. These experiments provide valuable insights into the proposed methodology's capabilities, highlighting its potential to optimize performance, support scalable solutions, and maintain robust data integrity in healthcare environments.

3. Implementation

To bring the proposed methodology to life, we developed a robust implementation within a PostgreSQL database environment and utilized Python to automate the hashing and segmentation process. This section covers the database setup, Py-

thon algorithms used for managing hashing, and the implementation of our segmentation approach for partitioning data entries efficiently.

Note: For the purpose of this experiment, dummy random unique patient and record data were created separately for each sample size. This ensured the validity of the partitioning algorithm and simulated a realistic distribution of patient records.

3.1. Experimental Setup

The experimental setup was conducted on a server with the following specifications:

- **Processor:** Intel Xeon Gold 6248 (7 cores, 40 threads)
- **Memory:** 16 GB RAM
- **Storage:** 2 TB SSD

The software used in the study includes PostgreSQL version 12.3 and Python 3.8.

For the hashing and partitioning tasks, the following libraries were used:

- **hashlib** for MD5, SHA-256, and SHA-3 hashing
- **psycopg2** for PostgreSQL database interaction

Query Types Tested:

- **Simple SELECT Queries:** Retrieval of patient data by hashed patient ID.
- **Complexity:** O(n) for scanning through all partitions (in partitioned tables) or the entire table (in non-partitioned tables), where n is the number of records in the dataset. The efficiency of partitioning helps reduce the number of records to scan for each query, thus improving query retrieval times.
- **Insert Operations:** Simulated insert operations were performed both on partitioned and non-partitioned tables. Data was inserted into both setups, and the insertion times were compared to assess the impact of partitioning on data insertion speed.
- **Complexity:** O(1) for inserts in both partitioned and non-partitioned tables, assuming a direct insertion without additional indexing or triggers. However, partitioning can introduce overhead when determining the target partition for each record, which is handled by the modulus operation, making the insertion time dependent on the number of partitions, but still approximately O(1).
- • **Update Operations:** Simulated update operations were tested by modifying patient records in both partitioned and non-partitioned tables. The performance of update queries was evaluated by comparing the execution times between partitioned and non-partitioned setups.
- **Complexity:** O(n) for searching and updating records in non-partitioned tables, where n is the number of records. For partitioned tables, the update operation's complexity may vary depending on how many partitions the system has to search. Ideally, partitioning reduces the number of partitions to search, making the update operation faster. However, it could still have a worst-case complexity of O(n) if updates require scanning across multiple partitions.

3.2. Database Setup and Partitioning Schema

Our implementation is built on PostgreSQL, which was chosen for its support for partitioned tables and the ability to handle large volumes of data efficiently. The database schema includes a primary table for patient records, which is partitioned into multiple segments based on the hashed patient identifiers.

1) Partitioned Table Design: We created a primary table, patient_records, which holds healthcare data. This table is partitioned into 10 segments (or partitions) named patient_records_partition_1 through patient_records_partition_10. Each partition holds data assigned to it based on the segment number derived from our hashing algorithm.

2) Partitioning Mechanism: PostgreSQL natively supports partitioning, allowing each partition to be a subset of the primary table. We define each partition to accept only records mapped to its corresponding segment, ensuring that data is distributed according to the segmentation logic.

3) Flexible Partition Count: Although we have implemented 10 partitions in this case, the approach is flexible and can support any number of partitions, denoted by n. This allows the database to scale as data volumes increase, enabling dynamic adjustment based on organizational needs.

3.3. Python Algorithm for Hashing and Segment Mapping

To manage the hashing and segmentation process, we developed a Python algorithm that applies MD5 hashing to patient identifiers, converts the hashed output into a consistent integer, and then assigns each integer to a specific segment using a modulus operation as Figure 1.

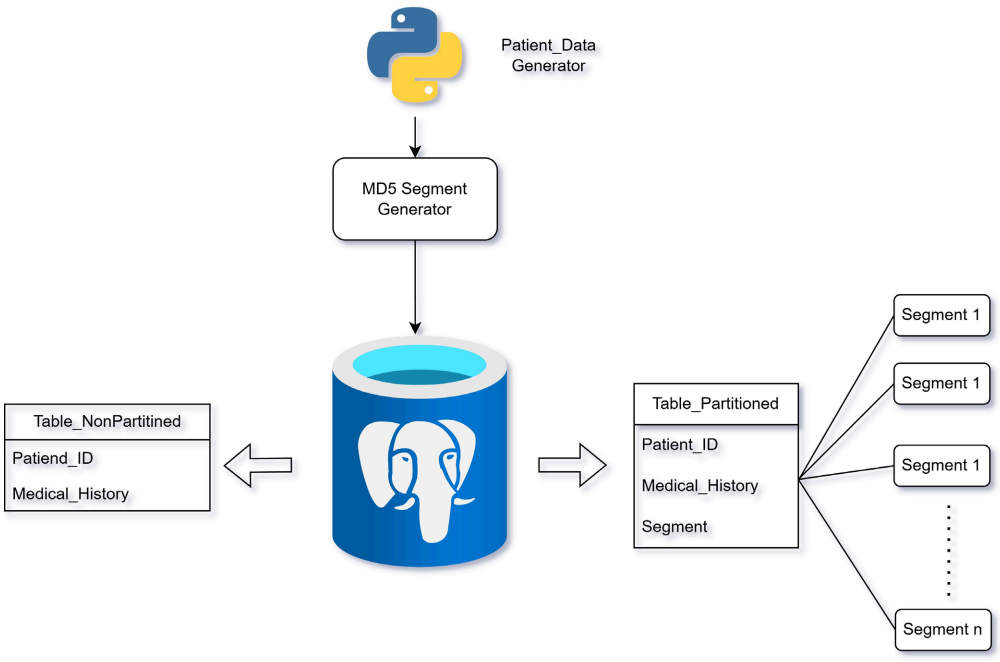

Figure 1. Hashing and segmenting patient identifiers using MD5 and modulus.

3.4. Algorithm for MD5-Based Healthcare Data Partitioning

The following algorithm describes how to securely and efficiently assign healthcare data to partitions using an MD5-based approach. This ensures data is consistently mapped to the same partition, maintaining both security and retrieval efficiency.

Step1: Hash the Patient Identifier (PII)

The first step is to apply the MD5 hashing algorithm to the patient identifier (e.g., Patient ID). This ensures that sensitive data, such as the patient's PII, is anonymized.

- **Input:** Patient Identifier ID_i
- **Output:** MD5 Hash Hash_i

Hash_i = MD5(ID_i)

Step 2: Convert the MD5 Hash to Integer

The second step converts the 32-character hexadecimal MD5 hash into an integer. This integer will be used to determine the partition to which the data will be assigned.

- **Input:** MD5 Hash Hash_i
- **Output:** Integer Representation Int_i

Int_i = int(Hash_i, 16)

Step 3: Map to a Partition

Using the modulus operation, the resulting integer is mapped to a specific partition. The modulus operation ensures that the hash maps to a partition number between 1 and n, where n is the total number of partitions. This guarantees a uniform distribution of data across the available partitions.

- **Input:** Integer Representation Int_i, Number of Partitions n
- **Output:** Assigned Partition Partition_i

Partition_i = (Int_i % n) + 1

Step 4: Insert Data into the Assigned Partition

The final step is to insert the healthcare record into the appropriate partition. The record is placed into the partition identified in Step 3, based on the patient's hashed identifier.

- **Input:** Patient Record Record_i, Assigned Partition Partition_i
- **Output:** Data inserted into the corresponding partition.

Automating Data Insertion with Python

The proposed system leverages a segmentation function to dynamically allocate each patient record to its corresponding partition. This ensures that every data entry is directed to the appropriate segment within the PostgreSQL database, facilitating efficient data management and storage.

Integration of the Python Algorithm with PostgreSQL

The Python-based algorithm is seamlessly integrated with PostgreSQL to automate the end-to-end process of data hashing, partition mapping, and insertion. The following highlights the key advantages of this integration:

1) Automated Data Flow: As new patient records are introduced into the system, the algorithm autonomously applies an MD5 hash to the patient identifier.

This hash is then mapped to a specific partition, and the corresponding patient data is inserted into the appropriate PostgreSQL partitioned table. This automation eliminates the need for manual segmentation, significantly streamlining data processing and reducing handling times.

2) Scalability and Flexibility: The system's design allows for dynamic scaling by adjusting the num_partitions parameter, enabling it to adapt to varying data volumes and organizational requirements. This flexibility allows healthcare providers to efficiently manage data storage by expanding or reducing the number of partitions as needed.

3) Optimized Query Performance: The partitioned table structure inherently optimizes query performance. With each patient identifier being mapped to a specific partition, the database engine can swiftly locate and access the relevant partition, thereby reducing query times and improving overall system performance—particularly in high-traffic environments where quick access to patient records is critical.

4) Fault Tolerance and Data Integrity: PostgreSQL ensures that data is correctly distributed across partitions, while the Python algorithm guarantees that each record is consistently assigned to its designated partition. This setup enhances the fault tolerance of the system and maintains data integrity, as each patient record is stored in a non-overlapping partition that ensures no conflicts or duplication.

3.5. Distribution

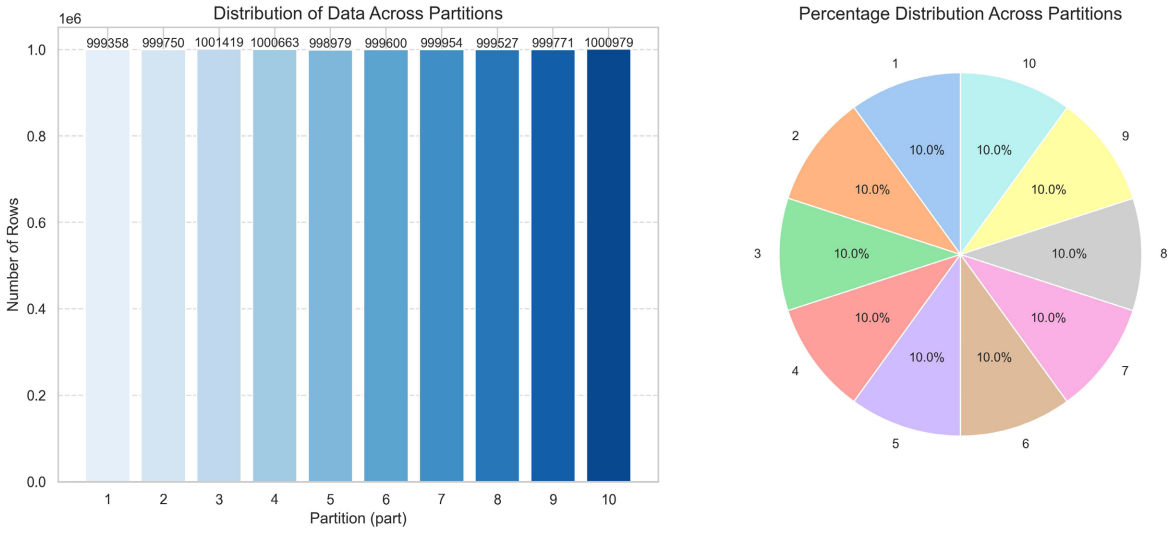

Figure 2. Data distribution across partitions shown via bar and pie charts.

Figure 2 illustrates the distribution of data across the partitions in the *patient_records_partitioned* table. The bar chart on the left highlights the exact number of records present in each partition (10 million rows total), providing a clear comparison of their relative sizes. Meanwhile, the pie chart on the right complements

this by showcasing the percentage contribution of each partition to the total dataset. This dual visualization not only emphasizes any potential imbalances in partition sizes but also aids in evaluating the effectiveness of the current partitioning strategy, which is critical for optimizing query performance and data accessibility in a relational database environment.

3.6. Benefits of the Implementation

This dual-layer hashing and partitioning approach provides several notable benefits:

- **Enhanced Security:** The use of MD5 hashing ensures that personally identifiable information (PII) is protected by storing only the hashed values within the database. This approach is in compliance with healthcare data protection regulations, safeguarding sensitive patient information.
- **Improved Performance:** By directing each record to a specific partition, the system minimizes the computational overhead during data retrieval. This is especially important in high-traffic healthcare systems where quick access to patient records is essential for operational efficiency.
- **Scalability:** The flexible partitioning mechanism accommodates the growing needs of healthcare organizations. As data volumes increase, the system can scale seamlessly to ensure continued optimal performance and storage management.

4. Experiment and Results

4.1. Overview

The purpose of this experiment was to evaluate the impact of partitioning on the performance of database queries in terms of data retrieval times. We compared partitioned and non-partitioned tables by fetching varying amounts of data from tables of different sizes. The dataset sizes ranged from 1 million to 15 million records, and we tested the performance of both partitioned and non-partitioned tables with fetch sizes ranging from 1000 to 8,192,000 records.

The dataset sizes used in the experiment, ranging from 1 million to 15 million records, are simulated datasets designed to observe the trends and complexities of partitioning efficiency as data volume increases. The goal of this study is not to precisely replicate real-world healthcare datasets but to demonstrate how the proposed partitioning strategies affect query performance as the data volume grows. While healthcare datasets in practice may be much larger, this study focuses on understanding the impact of data volume on partitioning efficiency. For future work, we plan to extend this research to larger, more complex real-world datasets to validate our findings further.

Partitioning is expected to improve performance by splitting a large table into smaller, more manageable pieces, thus enabling faster data retrieval and reducing the query execution time. This experiment aimed to validate this hypothesis and analyze how partitioning influences the query performance as the dataset size

grows.

4.2. Experimental Setup

We measured the time taken to fetch specific amounts of data from the partitioned and non-partitioned tables. For each dataset size, we recorded the time it took to fetch different numbers of records (fetch sizes). The experiment was conducted using 1 million, 5 million, 10 million, and 15 million records.

The times for data retrieval are reported in seconds, with **Partitioned Time (s)** representing the time taken from the partitioned table, and **Normal Time (s)** representing the time taken from the non-partitioned table.

4.3. Results

Below is a cleanly formatted Table 1. summarizing the query times for each dataset size and fetch size.

Table 1. Query times for each dataset size and fetch size.

Table Size	Partitioned Time (s)	Normal Time (s)
1 Million	0.89	0.32
5 Million	2.23	4.19
10 Million	4.22	7.60
15 Million	7.89	13.88

In order to simplify the results, we calculated the **average query time** for each dataset size (1 million, 5 million, 10 million, and 15 million records) based on the varying fetch sizes tested. The average query times for both partitioned and non-partitioned tables were computed by averaging the query times across all fetch sizes for each dataset.

Average Query Times

The visualizations compare data fetching times between partitioned and normal tables across different table sizes, as shown in Figure 3.

- **FacetGrid:** Each plot shows the fetching time for partitioned and normal tables, with n on the x-axis (number of rows fetched) and time (seconds) on the y-axis. Partitioned tables generally have lower fetching times, especially as the number of rows increases, indicating improved performance with partitioning.

This line graph visualizes the fetching time comparison between partitioned and normal tables across different table sizes.

- **X-axis:** Represents the number of rows fetched (n), showing the scale of data being processed.
- **Y-axis:** Represents the time (in seconds) it takes to fetch the data, which is used to compare the performance between partitioned and normal tables.

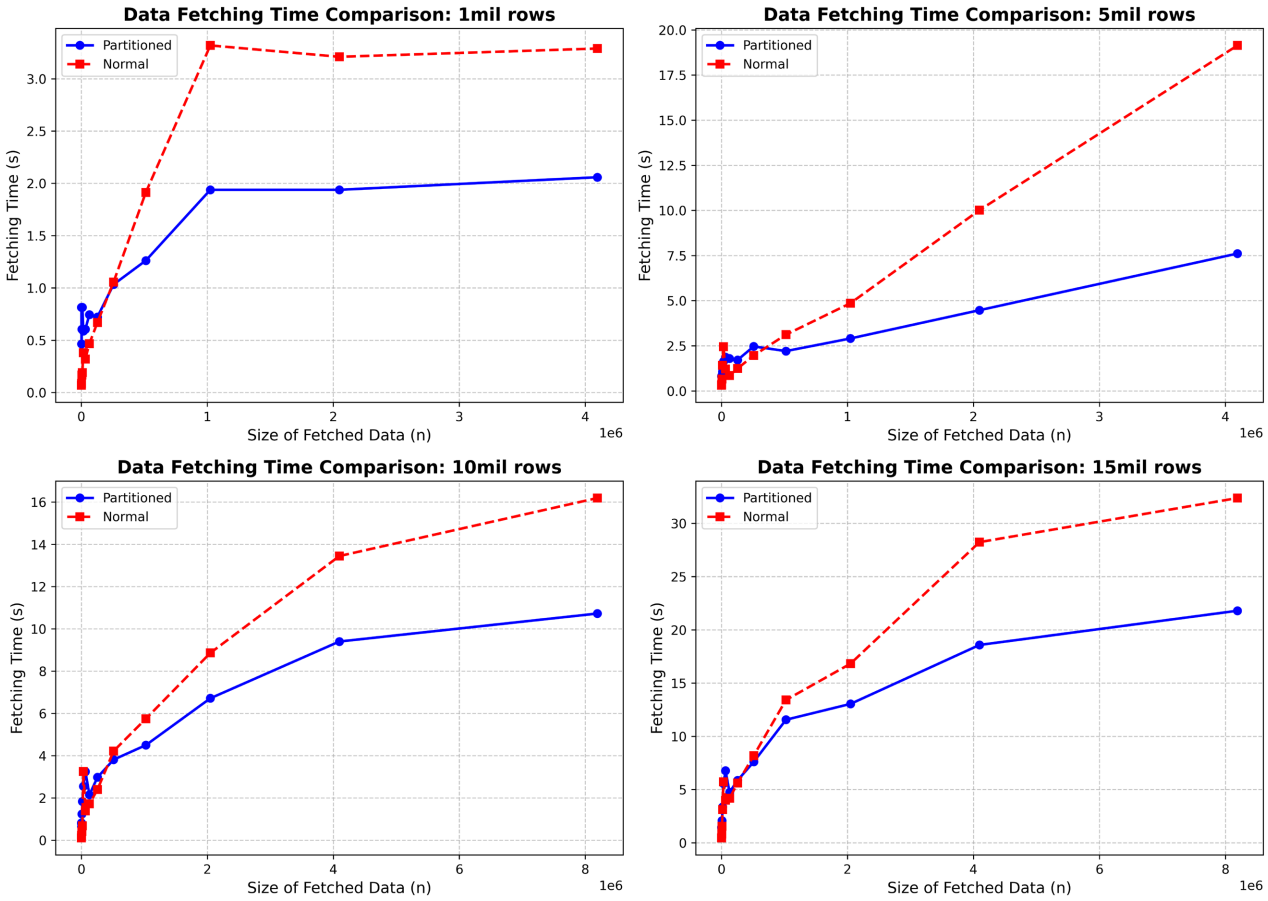

Figure 3. Data fetching times for partitioned vs. normal tables across sizes.

Key Features:

1) Different Colors for Each Table Size:

- The partitioned table lines are shown in colors from the *viridis* color map, while normal table lines use the *plasma* color map, allowing clear differentiation between table sizes for both partitioned and normal cases.

2) Lines and Markers:

- Solid lines (with circle markers) represent the fetching times for partitioned tables.
- Dashed lines (with square markers) represent the fetching times for normal tables.

3) Table Size Labels in the Legend: Each line is labeled with the corresponding table size (e.g., "Partitioned (100 k rows)") for easy identification.

Figure 4 demonstrates how data fetching times change for both partitioned and normal tables as the size of the data (number of rows) increases. It highlights the improved performance of partitioned tables in terms of reduced fetching times, particularly as the number of rows grows. The distinct colors and markers help compare the performance of different table sizes in a clear and intuitive way.

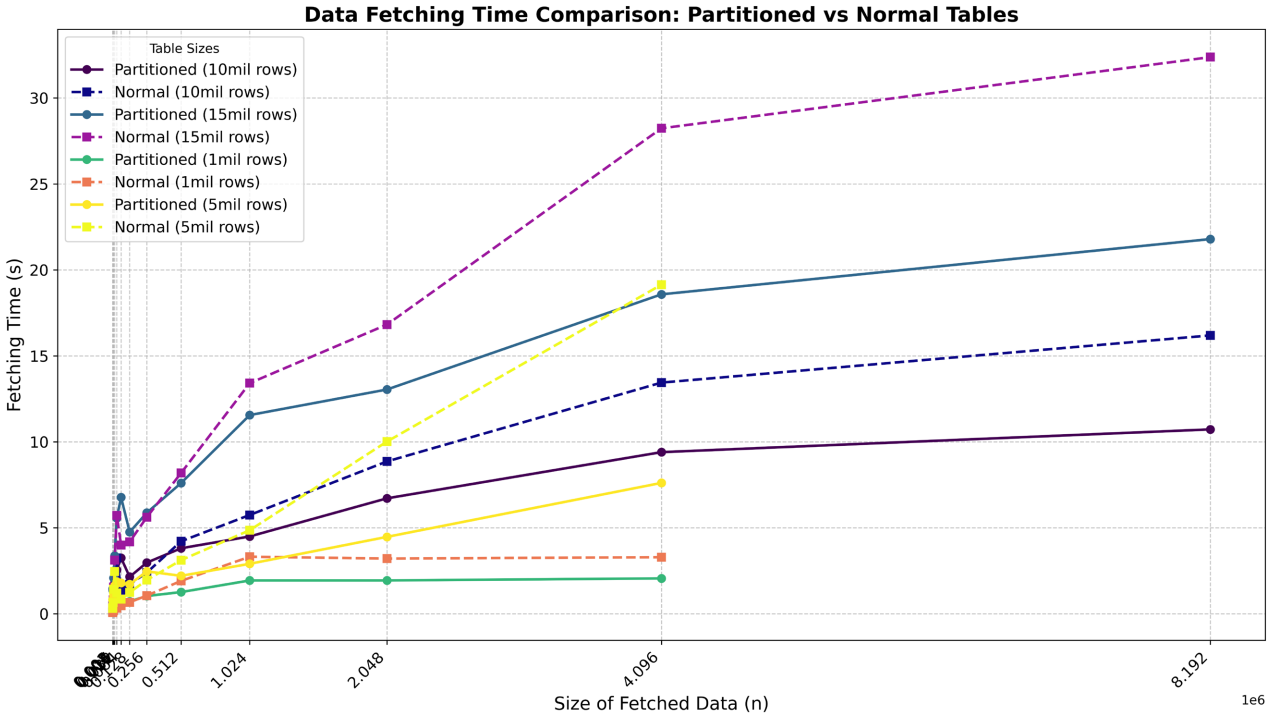

Figure 4. Data fetching times for partitioned vs. normal tables as data size increases.

Analysis of Results

1) Partitioning Improves Performance for Larger Fetch Sizes: The results show that partitioned tables consistently outperform non-partitioned tables, particularly as the fetch size increases. This performance improvement becomes more noticeable with larger datasets, especially for the 5 million, 10 million, and 15 million record tables.

2) Non-Partitioned Tables Exhibit Slower Performance: For larger datasets, non-partitioned tables experience a more significant increase in query time as fetch size grows. This trend highlights the scalability benefits of partitioning.

3) Stable Performance in Partitioned Tables: Partitioning provides a more stable and consistent query performance across all fetch sizes. The query times for partitioned tables grow less dramatically compared to non-partitioned tables, showing that partitioning effectively mitigates performance degradation.

5. Discussion

The practicality of implementing and maintaining the proposed partitioning system in real-world healthcare settings involves several important considerations. These factors should be addressed to ensure both the effectiveness and sustainability of the system in large-scale healthcare environments.

1) Deployment Considerations: For large healthcare organizations, deploying the partitioning system requires a robust and scalable infrastructure. High-performance servers with sufficient processing power and memory are necessary to handle the growing volumes of patient data. Furthermore, cloud-based storage

solutions can be utilized to scale efficiently as data increases. This allows healthcare organizations to manage dynamic data loads without the need for frequent hardware upgrades, supporting the long-term viability of the partitioning approach.

2) Cost and Maintenance: While partitioning improves query performance and enhances scalability, it introduces additional overhead in terms of managing multiple partitions. To maintain optimal performance, regular monitoring is required to avoid issues like fragmentation, which can degrade query performance over time. Maintenance tasks include adjusting partition sizes as data grows, ensuring efficient partitioning schemes, and performing periodic optimizations. Although partitioning improves performance, organizations must balance this benefit with the added cost of ongoing maintenance and administrative overhead, making it important to assess the cost-benefit trade-offs based on the organization's specific needs.

3) Integration with Existing Systems: Integrating the proposed partitioning approach with existing Electronic Health Record (EHR) systems and other healthcare data management platforms may pose challenges. Many legacy systems are not natively partitioned, which could lead to compatibility issues or require substantial system redesigns. To address this, we recommend using middleware solutions or database abstraction layers that facilitate seamless integration, enabling the partitioning approach to work with existing systems without causing disruptions to ongoing operations. Middleware can ensure that the partitioning logic is abstracted away from the underlying infrastructure, easing the burden of integration.

4) Data Modification Operations: While partitioning introduces some complexities in data modification operations such as insertions, updates, and deletions, several strategies can mitigate these challenges and ensure the system operates smoothly without performance degradation:

- **Insertion:** To ensure efficient data insertion, we propose a **batch processing mechanism**. Instead of inserting data one row at a time, multiple insertions can be grouped together in a batch. This minimizes bottlenecks that may occur with frequent single-row insertions, improving the overall efficiency of the system. Batch processing reduces the number of write operations, making it more suitable for environments where high-throughput insertions are required.
- **Updates:** For updates, we recommend a **multi-phase commit strategy** to ensure that updates are consistently reflected across all partitions. This approach guarantees that partitioned data remains synchronized and consistent, minimizing the risk of data inconsistency or discrepancies. It involves a two-phase commit process that ensures updates are properly committed or rolled back across partitions, maintaining the integrity of the dataset.
- **Deletes:** For deletes, we propose a **soft-delete approach**. In this strategy, records are flagged as deleted but remain in the partition until a scheduled maintenance operation purges them. This helps maintain partition integrity while minimizing the impact on performance during deletion operations. Soft de-

letes avoid the need for costly operations to remove records immediately, allowing for smoother ongoing performance and the ability to defer resource-heavy cleanup tasks to off-peak times.

5) Scalability and Fault Tolerance: An important advantage of partitioning is its potential to improve scalability and fault tolerance. Partitioning allows data to be distributed across multiple servers or storage systems, which can improve performance by reducing query execution times, especially as data volumes grow. Additionally, partitioning enhances fault tolerance by isolating failures to individual partitions rather than affecting the entire dataset. If a partition becomes corrupted or unavailable, it is possible to recover or restore individual partitions without impacting the rest of the system, ensuring high availability and reliability.

6) Data Security and Privacy: Given that the proposed system is designed for healthcare data, security and privacy are critical considerations. The use of MD5, SHA-256, or SHA-3 hashing algorithms for anonymizing patient data ensures compliance with privacy regulations like HIPAA. While we acknowledge the limitations of MD5 in terms of security, our study focused on partitioning efficiency, with security being a secondary consideration. Nonetheless, the use of stronger hashing algorithms in real-world applications can provide additional layers of protection for sensitive healthcare data, reinforcing trust and compliance in healthcare environments.

7) Long-Term Sustainability: Over time, as data grows and healthcare systems evolve, the partitioning strategy will need to adapt. This may involve rebalancing partitions as data becomes more evenly distributed or as certain data types grow faster than others. A dynamic partitioning system that can adjust to changing data patterns will be crucial for maintaining long-term efficiency and avoiding performance bottlenecks.

6. Conclusions

This study aimed to evaluate the performance of partitioned tables compared to normal tables in terms of data fetching times, focusing on the impact of different table sizes. The experiment involved querying both partitioned and normal tables across multiple datasets with varying numbers of rows (ranging from 1 million to 8 million). For each dataset, the fetching times were recorded for both partitioned and normal tables to assess the differences in query performance.

The results were visualized through multiple graphs, including bar charts and line graphs, to provide clear comparisons. The line graphs, in particular, revealed a consistent pattern where the partitioned tables exhibited a more gradual increase in fetching time as the dataset size grew. In contrast, the normal tables showed a steeper and more abrupt rise in fetching times, especially for larger datasets. These trends suggest that partitioned tables—by splitting the data into smaller, more manageable segments—can reduce the time required for querying, even as the dataset expands.

The findings suggest that partitioning is a highly effective method for improv-

ing data retrieval efficiency, especially in environments where large datasets need to be queried frequently. The gradual increase in fetching time for partitioned tables indicates that partitioning helps mitigate the negative impact of large data volumes on query performance, likely by limiting the number of rows the system needs to scan for each query.

While the experiment highlights the benefits of partitioning, it is important to note that the approach is not without its trade-offs. Partitioning schemes can introduce additional complexities in database design, especially in terms of data modification operations such as inserts, updates, or deletions. These operations may become more complicated or slower in partitioned tables if the partitioning strategy is not optimized for specific workloads. Additionally, partitioning introduces overhead in managing and maintaining partitions, especially in dynamic databases where data is frequently updated.

Looking ahead, future research could explore the impact of different partitioning schemes, such as range partitioning or hash partitioning, on query performance and system resource usage. Moreover, examining how partitioning affects other database operations, like insertions and deletions, could provide a more comprehensive understanding of the trade-offs involved. Investigating these factors would be particularly useful for scenarios where data modification is as important as querying.

In conclusion, this study confirms that partitioning tables can significantly improve database performance in terms of data fetching times, especially for large-scale datasets. However, the design of the partitioning scheme should be carefully considered to balance query efficiency with the operational complexities of data modification and maintenance. By considering these factors, organizations can optimize their database architecture to ensure both high performance and maintainability as data grows in size and complexity.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Raghupathi, W. and Raghupathi, V. (2014) Big Data Analytics in Healthcare: Promise and Potential. *Health Information Science and Systems*, **2**, Article No. 3. https://doi.org/10.1186/2047-2501-2-3

[2] Schneeweiss, S. (2014) Learning from Big Health Care Data. *New England Journal of Medicine*, **370**, 2161-2163. https://doi.org/10.1056/nejmp1401111

[3] Olaronke, I. and Oluwaseun, O. (2016) Big Data in Healthcare: Prospects, Challenges and Resolutions. 2016 *Future Technologies Conference* (*FTC*), San Francisco, 6-7 December 2016, 1152-1157. https://doi.org/10.1109/ftc.2016.7821747

[4] Public Health Law (2024) Health Insurance Portability and Accountability Act of 1996 (HIPAA). https://www.cdc.gov/phlp/php/resources/health-insurance-portability-and-accountability-act-of-1996-hipaa.html

[5] Legal Text (2024) General Data Protection Regulation (GDPR). https://gdpr-info.eu/

[6] Bélanger, F. and Crossler, R.E. (2011) Privacy in the Digital Age: A Review of Information Privacy Research in Information Systems. *MIS Quarterly*, **35**, 1017-1041. https://doi.org/10.2307/41409971

[7] DeVries, W.T. (2003) Protecting Privacy in the Digital Age. *Berkeley Technology Law Journal*, **18**, 283-311.

[8] Kostkova, P. (2015) Grand Challenges in Digital Health. *Frontiers in Public Health*, **3**, Article 134. https://doi.org/10.3389/fpubh.2015.00134

[9] El Aboudi, N. and Benhlima, L. (2018) Big Data Management for Healthcare Systems: Architecture, Requirements, and Implementation. *Advances in Bioinformatics*, **2018**, Article 4059018. https://doi.org/10.1155/2018/4059018

[10] Abouelmehdi, K., Beni-Hessane, A. and Khaloufi, H. (2018) Big Healthcare Data: Preserving Security and Privacy. *Journal of Big Data*, **5**, Article No. 1. https://doi.org/10.1186/s40537-017-0110-7

[11] Price, W.N. and Cohen, I.G. (2019) Privacy in the Age of Medical Big Data. *Nature Medicine*, **25**, 37-43. https://doi.org/10.1038/s41591-018-0272-7

[12] Das, L., Kumar, A., Sharma, V., Bhatnagar, D., Singh, S. and Tripathi, N. (2024) Data-driven Healthcare Management, Analysis, and Future Trends. 2024 *International Conference on Communication, Computer Sciences and Engineering* (*IC3SE*), Gautam Buddha Nagar, 9-11 May 2024, 1692-1698. https://doi.org/10.1109/ic3se62002.2024.10593217

[13] Dash, S., Shakyawar, S.K., Sharma, M. and Kaushik, S. (2019) Big Data in Healthcare: Management, Analysis and Future Prospects. *Journal of Big Data*, **6**, Article No. 54. https://doi.org/10.1186/s40537-019-0217-0

[14] Campbell, R.J. (2004) Database Design: What HIM Professionals Need to Know. *Perspectives in Health Information Management*, **1**, 6.

[15] Wang, X., Williams, C., Liu, Z.H. and Croghan, J. (2017) Big Data Management Challenges in Health Research—A Literature Review. *Briefings in Bioinformatics*, **20**, 156-167. https://doi.org/10.1093/bib/bbx086

[16] Kantode, V., Sharma, R., Singh, S., Ankar, R. and Gujar, S. (2022) Big-Data in Healthcare Management and Analysis: A Review Article. 2022 3*rd International Conference on Electronics and Sustainable Communication Systems* (*ICESC*), Coimbatore, 17-19 August 2022, 1139-1143. https://doi.org/10.1109/icesc54411.2022.9885609

[17] Data Partitioning Guidance. https://learn.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning

[18] Thantilage, R.D., Le-Khac, N. and Kechadi, M. (2023) Healthcare Data Security and Privacy in Data Warehouse Architectures. *Informatics in Medicine Unlocked*, **39**, Article 101270. https://doi.org/10.1016/j.imu.2023.101270

[19] IBM (2022) What Is Encryption? https://www.ibm.com/topics/encryption

[20] IBM (2024) Cryptography Use Cases: From Secure Communication to Data Security. https://www.ibm.com/think/topics/cryptography-use-cases

[21] Canim, M., Kantarcioglu, M. and Malin, B. (2012) Secure Management of Biomedical Data with Cryptographic Hardware. *IEEE Transactions on Information Technology in Biomedicine*, **16**, 166-175. https://doi.org/10.1109/titb.2011.2171701

[22] Ge, Y., Wang, H., Bertino, E., Zhan, Z., Cao, J., Zhang, Y., *et al.* (2024) Evolutionary

Dynamic Database Partitioning Optimization for Privacy and Utility. *IEEE Transactions on Dependable and Secure Computing*, **21**, 2296-2311. https://doi.org/10.1109/tdsc.2023.3302284

[23] Chawla, T., Singh, G., Pilli, E.S. and Govil, M.C. (2020) Storage, Partitioning, Indexing and Retrieval in Big RDF Frameworks: A Survey. *Computer Science Review*, **38**, Article 100309. https://doi.org/10.1016/j.cosrev.2020.100309

[24] Ahmed, T. and Sarma, M. (2019) Hash-Based Space Partitioning Approach to Iris Biometric Data Indexing. *Expert Systems with Applications*, **134**, 1-13. https://doi.org/10.1016/j.eswa.2019.05.026

[25] Salgova, V. and Matiasko, K. (2020) Reducing Data Access Time Using Table Partitioning Techniques. 2020 18*th International Conference on Emerging eLearning Technologies and Applications* (*ICETA*), Košice, 12-13 November 2020, 564-569. https://doi.org/10.1109/iceta51985.2020.9379231

[26] Sun, C., Dai, H., Liu, H., Chen, T.Y. and Cai, K. (2019) Adaptive Partition Testing. *IEEE Transactions on Computers*, **68**, 157-169. https://doi.org/10.1109/tc.2018.2866040

[27] Mohammed Ali, A. and Kadhim Farhan, A. (2020) A Novel Improvement with an Effective Expansion to Enhance the MD5 Hash Function for Verification of a Secure E-document. *IEEE Access*, **8**, 80290-80304. https://doi.org/10.1109/access.2020.2989050

[28] Dolmeta, A., Martina, M. and Masera, G. (2023) Comparative Study of Keccak SHA-3 Implementations. *Cryptography*, **7**, Article 60. https://doi.org/10.3390/cryptography7040060