Multilevel Database Decomposition Framework

Fabrizio Baiardi[1], Cosimo Comella[2] and Vincenzo Sammartino[1]

[1]Università di Pisa, Largo Bruno Pontecorvo 3, 56127 Pisa (PI)
[2]Autorità Garante per la Protezione dei Dati Personali, Piazza Venezia 11, 00187 Roma

Abstract

The Multilevel Database Decomposition Framework is a strategy to enhance system robustness and minimize the impact of data breaches. The framework prioritizes robustness against cyber threats over minimizing data redundancy by decomposing a database into smaller ones to restrict user access according to the least privilege principle. For this purpose, each database the decomposition produces is uniquely associated with a set of users and the decomposition ensures that each user can access all and only the data his/her operations need. This minimizes the data a user can access and the impact of an impersonation attack.

To prevent the spreading of an intrusion across the databases it produces, the framework supports alternative allocation strategies that map the databases onto distinct virtual or physical entities according to the robustness of interest. This flexibility in allocation management ultimately reinforces defenses against evolving cyber threats and it is the main advantage of the deposition.

As a counterpart of better robustness, some tables will be replicated across the databases the decomposition returns and their updates should be properly replicated to prevent inconsistencies among copies of a table in distinct databases. We present a performance analysis to evaluate the overhead of each allocation. This offers insights into how the framework can satisfy distinct security requirements. We use these results to evaluate the effectiveness of the framework for healthcare applications.

Keywords

Decomposition, Database Allocation, Impact Assessment, GDPR

1. Introduction

The Multilevel Database Decomposition Framework (MDDF) is an innovative approach that targets the protection of personal information with a focus on the healthcare sector. It is designed to implement relational databases with high robustness as it fully satisfies the least privilege principle to effectively mitigate contemporary threats and safeguard sensitive healthcare information [1, 2].

The MDDF key notion is the decomposition of one relational database into a set of databases defined according to the user operations. This minimizes user access rights [3, 4] because each user can access all and only the data his/her operations need. MDDF strongly reduces the blast radius of a successful intrusion. Consider as an example the data that may be leaked due to an impersonation attack [5] where a threat agent impersonates a legitimate user. This strongly improves overall data security.

ITASEC24: Italian Conference on Cybersecurity, April 8-12, 2024, Salerno, IT
fabrizio.baiardi@unipi.it (F. Baiardi); c.comella@gpdp.it (C. Comella); v.sammartino@studenti.unipi.it (V. Sammartino)
0000-0001-9797-2380 (F. Baiardi); 0009-0002-4632-1179 (V. Sammartino)

To prevent the spread of intrusions across the decomposed databases, MDDF supports alternative allocation strategies. While the simplest allocation maps all databases onto the same machine, confinement and robustness are enhanced by distributing databases across distinct containers or physical/virtual machines. The ability to choose the allocation according to the robustness of interest is a fundamental improvement that MDDF offers with respect to the management of the user access rights on the resulting databases.

The databases that the MDFF produces may share tables or subsets of tables. Hence, these tables are replicated across databases, and updates are replicated to maintain consistency among multiple copies of a table and to offer the same data view to each user. Multiple copies update introduces both complexity and overhead that are proportional to the robustness of the separation [6]. Another problem of replication is the larger amount of memory utilization. We believe this issue may be neglected when considering the more robust separation among the various databases and the ability of using one copy to restore consistent information after any attack against availability. From this perspective, the amount of memory the MDDF uses is alwayslower than the one of solutions based upon distributed ledgers.

The paper unfolds as follows: Sect. 2 delves into related works and briefly reviews the least privilege principle and other concepts underlying the proposed framework. Sect. 3 describes the MDDF, emphasizing database decomposition to optimally adhere to the least privilege principle. Sect.4 exemplifies the application of. Lastly, Sect. 5 reports performance figures of multiple update overhead as a function of the allocation of databases.

2. State of the Art

The principle of least privilege (PoLP) suggests minimizing the access rights of each system user so the user never owns access rights he/she does not need. This has several implications for the management of access rights and prevents the adoption of strategies based upon a hierarchical level of privileges. Systems satisfying the PoLP minimize the impact of an intrusion where a threat agent impersonates a legal user. State-of-the-art strategies to face the challenges posed by evolving cyber threats and cloud-based solutions improve database security by merging the PoLP with measures such as role-based access control (RBAC), encryption, tokenization, dynamic data masking, and pseudonymization. As discussed in the following, most of these solutions can be integrated with the MDDF to address the current risk scenarios [7, 8].

Role-Based Access Control (RBAC) [9] complements PoLP by assigning access permissions based on predefined roles. This streamlines user privilege management, ensuring individuals have access only to resources essential for their specific organizational roles.

Encryption and **Tokenization** are crucial components in securing sensitive data [10] that, respectively, transform data into unreadable formats and replace sensitive data with non-sensitive placeholders.

Dynamic Data Masking (DDM) [11] obfuscates sensitive information dynamically to ensure that only authorized individuals can access and view sensitive data.

Adaptive and comprehensive security measures are essential to counter the constantly evolving **cyber threat landscape** of databases that includes challenges such as ransomware, advanced persistent threats (APTs), and insider threats [12].

The widespread adoption of **cloud-based database** solutions introduces new security concerns [13] because protecting data in a cloud requires guarding against unauthorized access, data breaches from users of the same provider, and compliance with data residency regulations.

Pseudonymization is another vital facet of database security as it replaces personally identifiable information (PII) with artificial identifiers or pseudonyms [14]. This adds another layer of privacy protection by making it challenging to directly associate sensitive data with specific individuals. Pseudonymization contributes to compliance with data protection regulations as it allows organizations to leverage data for legitimate purposes without compromising individual privacy. The implementation of pseudonymization within database systems enhances data security by both reducing the impact of unauthorized access to personal information and limiting exposure in the event of a breach.

3. Multilevel Database Decomposition Framework

This section details the decomposition step by step to outline the underlying principles. To this end, it is essential to explain how to decompose a database and then move from the decomposition of a database to that of its tables. Then, the section highlights the synchronization problems generated by a decomposition where the same table is shared, i.e. it belongs to distinct databases. This requires that some operations on the table in one of the databases resulting from the decomposition fire an automatic update of databases sharing the same table.

3.1. General Approach

MDDF aims to fully satisfy the **principle of least privilege** (PoLP)[3] by decomposing a relational database shared among a set of users to minimize user access to the data involved in the operations each user can invoke. To this purpose, starting from a system where each user can access any information in the database, the framework produces a system with distinct databases and where each user is granted access to just one of the databases, ie to the smallest amount of data to implement the operations of interest. This is achieved by distributing the tables in the original database to several databases, each consisting of only a subset of those in the original one. Each resulting database includes all the tables to implement the operations of a class of users and if the operations do not access an attribute of a table, the attribute is dropped. Each user belongs to one class and it is assigned access rights on the tables in one of the databases the decomposition returns. This strategy satisfies the PoLP as it assigns all and only the access rights the user needs. MDDF can be integrated with any of the various mechanisms discussed in the previous section as any of these mechanisms can be applied to each of the databases. The last step of the framework maps the databases it builds to distinct containers, virtual or physical machines to confine a successful intrusion to one of the databases.

Definition and Purpose: Normal forms apply decomposition to simplify the understanding and management of the database schema. This helps to reduce redundancy, to improve maintainability, and to enhance the overall database performance.

The decomposition of a database, e.g. a set of tables, returns subsets of these tables. Each table is either an original table or a subset of the attributes, the columns, of an original table.

The subsets of tables are not partitions because they can share some tables. Each resulting subset is stored separately, and it can be accessed and managed independently.

The decomposition is implemented through a sequence of steps that, given a database and a set of users, can be described as follows:

- Traditional Normalization in 3NF
- Identification of User Groups
- Creation of Database Subsets
- Association of Groups and Subsets
- Configuring Access Mechanisms
- Choosing the Confinement Robustness

3.2. Steps

The following subsections will show in detail the steps to be followed to apply the framework.

3.2.1. Traditional Normalization in 3NF

Normalization is a process that structures data in a database to eliminate redundancy and dependency. The third Normal Form (3NF) is a widely used standard that reduces data redundancy. We assume this normalization has already been applied to ensure that the database is well structured and data is stored consistently.

3.2.2. Identification of User Groups

Identifying users is a preliminary step that identifies all users who interact with the database and analyzes the needs of the operations of each user. In this way, we create groups of users tailored to their needs because two users belong to the same group if they invoke the same operations on the same tables. This allows access to sensitive information to be limited only to users who are entitled to operate on such information according to the *need-to-know principle*.

3.2.3. Creation of Database Subsets

The creation of multiple databases, each a subset of the original one is the central step of the framework. It creates a distinct database for each user group, according to the operation the users in the group execute and the data these operations access. We consider the worst case, e.g. any table and any attribute a user may require should belong to the corresponding subset to ensure that users can access all and only the information they need. As a consequence, some tables will be shared among users in distinct groups and they appear in distinct databases. The attributes of these shared tables implement data exchange among users in distinct groups. This requires that an update in a table is spread across all the databases where the table, or the attribute, appears.

3.2.4. Association of Groups and Subsets

The strategy to decompose the database obviously results in a biunivocal association between user groups and the databases the decomposition returns. The association is the input to define user access rights. The membership in a group is dynamic because it depends upon roles assigned by the organization.

3.2.5. Configuring the Access Mechanism

This step offers a first level of security by granting each user the access rights to access all and only the information in the database associated with the user group. This prevents users from accessing information in the original database that is not necessary for their work. The detailed implementation of this step depends upon both the underlying operating system and database management system that have been adopted. The restriction of users to access only the information they need minimises the blast radius of an intrusion ie the amount of data that may be lost because of data breaches and unauthorized access.

3.2.6. Choosing the Confinement Robustness

The definition of allocation of the databases from the previous steps onto physical machines, virtual machines, VMs, and containers determines the confinement level according to the robustness of the overall system that is required to protect the various information. The choice of allocation usually depends on the critical level of the information in a database and on how much the solution should confine an intrusion or an impersonation.

3.3. Confinement

MDDF enables the designer to select different confinement levels, crucial for aligning the allocation with required security levels, resource usage, management complexity, and other factors. In more detail, each allocation offers a distinct robustness that also depends upon vulnerabilities in containers or virtual machines [15, 16, 17].

Mapping	Confinement Robustness
Distinct physical machines	High
Distinct VMs	Medium-High
Distinct containers	Medium
Simple database decomposition	Low

Table 1
Robustness levels of Alternative Database Mappings.

Table 1 outlines the confinement levels, or robustness, resulting from various database allocations. Allocating databases to distinct physical machines provides the highest level of confinement, while a simple database decomposition offers the lowest.

Each alternative has its pros and cons:

- **Distinct Physical Machines:** High safety and confinement due to physical separation but high resource usage and management complexity.
- **Distinct VMs:** Good confinement with customizable resource allocation, but with a resource overhead and expertise needed for management.
- **Distinct Containers:** Lightweight solution with minimal overhead, but potential security risk due to shared underlying OS.
- **Simple Database Decomposition:** Easier management and fewer resources, but low confinement and potential data integrity issues.

Even if an intrusion can attack the second allocation and the third one by exploiting vulnerabilities in containers or in VMs, all the allocations that MDDF supports offer a better robustness to intrusions than a simple transformation to the third normal form. The choice of the proper allocation should align with the specific requirements of the system of interest, balancing confinement, resource efficiency, and management complexity.

4. An Example

This example considers a scenario with the information a healthcare organization, ie a hospital, manages information about operators, patients, prescriptions, and related administration. The scenario assumes that initially, the hospital uses a centralized database and the tables in the database record patient details and prescriptions and all the data to satisfy the needs of medical staff and patients. The overall database management system ensures medical professionals have access to proper patient information, while patients can conveniently retrieve their own medical records and prescription details. Further users exist according to the various roles in the organization. Among the possible groups of users, we also include statisticians and the corresponding operations. The administrative staff manages costs, while statisticians analyze aggregated data without compromising individual patient identities. When computing statistics, the names of the patients are replaced with pseudonymous IDs, ensuring a high level of privacy, and a mapping table simplifies the association between these pseudonymous IDs and patients.

4.1. Users and Database

The Healthcare Database (HealthDB) of the organization is designed to meet the different needs of its users, including Patients, Medical Doctors, Nurses, Administrative Staff, and Statisticians.

Users:

- **Patients (Patients)**: This group requires read access to their medical records, prescription details, exam results, and associated costs. Access is facilitated through a pseudonymous ID for privacy.
- **Medical Doctors (Doctors)**: Medical professionals need read and write access to patient information, medical history, and the ability to prescribe medications and order exams.

- **Nurses (Nurses)**: Nurses require read and write access to patient information, medical history, and the ability to record exam results and administer prescribed medications.
- **Administrative Staff (Admins)**: This group focuses on write access to the CostsTable for billing purposes. They may also have read access to other relevant information.
- **Statisticians (Statisticians)**: Statisticians have read-only access to aggregated data in the Patients table for statistical analysis. They cannot view sensitive information like names and addresses.

Database:

- **Healthcare Database (HealthDB)**: This is the centralized database that stores information of interest in the following tables:
  - **Patient Information Table (PatientTable)**: Includes pseudonymous IDs (PatientID), comprehensive medical history (MedicalHistory), and IDs of assigned doctors (AssignedDoctorID) and nurses (AssignedNurseID). A mapping table (MappingTable) manages the link between real patient identities and pseudonymous IDs where each ID is automatically generated on the first hospital visit.
  - **Sensitive Data Table (SensitiveDataTable)**: Contains sensitive patient information such as real names (RealName), addresses (Address), and details of who pays the bill (PayerDetails). It is linked to PatientTable via pseudonymous IDs.
  - **Prescription Records Table (PrescriptionTable)**: Stores data related to prescriptions, including unique prescription IDs (PrescriptionID), details of prescribed medications (MedicationDetails), dosage (Dosage), prescribing doctor IDs (PrescribingDoctorID), and associated costs (AssociatedCost).
  - **Exam Records Table (ExamTable)**: Stores information about tests and exams, including unique IDs (examID) and test and exam results (Results).
  - **Costs Table (CostsTable)**: Manages the costs of medicines and exams, with transaction IDs (TransactionID), item descriptions (Item), and associated costs (Cost). Accessed by administrative staff for billing purposes.
  - **Mapping Table (MappingTable)**: Simplifies the mapping between pseudonymous patient IDs (PatientID) and their real identities (RealIdentityID).
  - **Statistics Table (StatisticsTable)**: Stores aggregated and de-identified data from the Patients table, accessible to statisticians for analysis. Includes aggregated data IDs (AggregatedDataID) and de-identified aggregated data (AggregatedData).

4.2. MDDF Implementation Steps

The implementation of MDDF adopts a systematic approach to satisfy the unique needs of user groups of the healthcare system. The key steps involve identifying users and their specific needs, computing the access permissions to satisfy these needs, and ensuring secure data management within the Healthcare Database (HealthDB). In the following, we delve into a short, high-level overview of these crucial implementation steps.

1. **Identify Users and Their Needs**:
   a) **Patients**
      - **Needs**:
        - Read access to their own medical records (MedicalHistory in PatientTable).
        - Read access to their prescription details (MedicationDetails, Dosage in PrescriptionTable).
        - Read access to their exam records (Results in ExamTable).
      - **Required Access**: Read access to PatientTable, PrescriptionTable, ExamTable in the Healthcare Database (HealthDB).
   b) **Doctors**
      - **Needs**:
        - Read and write access to patient information for assigned patients (PatientTable).
        - Write access to prescribe medications (PrescriptionTable).
        - Write access to order exams (ExamTable).
      - **Required Access**: Read and write access to PatientTable, DoctorsTable, PrescriptionTable, Examtable in the Healthcare Database (HealthDB).
   c) **Nurses**
      - **Needs**:
        - Read and write access to patient information for assigned patients (PatientTable).
        - Write access to record exam results (Examtable).
        - Write access to administer prescribed medications (PrescriptionTable).
      - **Required Access**: Read and write access to PatientTable, NursesTable, Examtable, PrescriptionTable in the Healthcare Database (HealthDB).
   d) **Administrative Staff (Admins)**
      - **Needs**:
        - Write access to the CostsTable for billing purposes (CostsTable).
      - **Required Access**: Write access to CostsTable in the Healthcare Database (HealthDB).
   e) **Statisticians**
      - **Needs**:
        - Read-only access to aggregated data in the PatientsTable for statistical analysis (StatisticsTable).
      - **Required Access**: Read-only access to StatisticsTable in the Healthcare Database (HealthDB).
2. **Create Database Subsets**
   This step produces the distinct database subsets of HealthDB for the various user groups. These databases areMedDB for medical staff, PatientDB for patients, AdminDB for administrative staff, and StatDB for statisticians. Since each database only include the relevant tables and fields the designated user roles require. we adhere to the principle of least privilege. The structure of each database the decomposition returns are listed below:
   Subset MedDB: This subset includes only the tables that in the main database are relevant to Medical Staff.

Key tables comprise:

- **PatientTable**: PatientID, MedicalHistory, AssignedDoctorID, AssignedNurseID
- **DoctorsTable**: DoctorID, Name, Specialty
- **NursesTable**: NurseID, Name, AssignedPatients
- **examsTable**: examID, Results
- **PrescriptionsTable**: PrescriptionID, MedicationDetails, Dosage, PrescribingDoctorID
- **MedicineCostsTable**: TransactionID, Item, Cost

Subset PatientDB: This subset includes only the tables in the main database relevant to Patients. Key tables include:

- **PatientTable**: PatientID, MedicalHistory, AssignedDoctorID, AssignedNurseID
- **SensitiveDataTable**: PatientID, RealName, Address, PayerDetails
- **examsTable**: examID, Results
- **PrescriptionsTable**: PrescriptionID, MedicationDetails, Dosage, PrescribingDoctorID

Subset AdminDB: This subset includes only the tables from the main database relevant to Administrative Staff. Key tables comprise:

- **CostsTable**: TransactionID, Item, Cost

Subset StatDB: This subset includes only the main database tables relevant to Statisticians. Key tables comprise:

- **StatisticsTable**: AggregatedDataID, AggregatedData

These subsets ensure that each user group can access all and only to the information the corresponding roles require according to the principle of least privilege.

3. **Associating User Groups and Database Subsets**: The biunivocal association between a user group and one of the databases the decomposition returns is the fundamental input for the next step that defines user access rights.

    - **Define User-Subset Mapping**: The association between each user group and a database subset map, Medical Staff into the MedDB subset, Patients into the PatientDB subset while Doctors should be mapped to the MedDB subset (with access to fields like DoctorID, Name, Specialty), Nurses should be mapped into the MedDB subset (with access to fields like NurseID, Name, AssignedPatients), and Administrative Staff should be mapped into the AdminDB subset (with access to fields like TransactionID, Item, Cost). Statisticians are to be mapped into the StatDB subset (with access to fields like AggregatedDataID, AggregatedData).
    - **Use Mapping for Access Control**: Starting from the mappings previously defined, we can configure access control mechanisms. The mapping implies the assignment of the corresponding access rights and permissions to each user in a user group on the corresponding database subset.

- **Regularly Update Mapping**: User groups should be reviewed with a fixed frequency because the group of a user can be updated to take into account changes in roles, responsibilities, and database structure. This ensures that access control remains aligned with organizational requirements.

The overall decomposition process ensures that the mapping between users and database subsets is clearly defined, access control mechanisms are implemented according to this mapping, and updates occur to adapt to organizational changes.

4. **Configure Access Mechanisms**: After creating the database subsets, it is essential to define who has access to each subset and what level of access is allowed:
   a) **Patients** should have read access to the **PatientDB** subset, allowing them to view and access their pseudonymous medical records, prescription details (fields: PrescriptionID, MedicationDetails, Dosage, PrescribingDoctorID), and exam records (fields: examID, Results).
   b) **Doctors** should have read and write access to the **MedDB** subset, similar to Medical Staff, with the additional ability to prescribe medications (fields: PrescriptionID, MedicationDetails, Dosage, PrescribingDoctorID) and order exams (fields: examID, Results).
   c) **Nurses** should have read and write access to the **MedDB** subset as Medical Staff, but with the additional ability to record exam results (fields: examID, Results) and administer prescribed medications (fields: PrescriptionID, MedicationDetails, Dosage, PrescribingDoctorID).
   d) **Administrative Staff (Admins)** should have write access to the **AdminDB** subset, to enable them to manage costs associated with medicines and exams (fields: TransactionID, Item, Cost).
   e) **Statisticians** should have read-only access to the **StatDB** subset, to enable them to analyze aggregated and de-identified data in the **PatientsTable** for statistical purposes (fields: AggregatedDataID, AggregatedData).

   After defining the relationships between the user groups and the various database subsets, some further procedures have to be adopted to guarantee security and data management. Below, we list some further details of each procedure:

   - **Authentication and Authorization**: It ensures that every user is authenticated, allowing only authorized users to access data in their respective subsets. This may include the use of usernames and passwords, two-factor authentication, or other secure authentication methods.
     Additionally, it is crucial to assign specific roles and permissions based on user responsibilities. For example, Medical Staff and Doctors must have the role "Medical Professional" with full permissions for the **MedDB** subset, while Patients must have appropriate read access roles for the **PatientDB** subset. Nurses and Administrative Staff must also be assigned roles with relevant permissions.
   - **Auditing and Monitoring**: It implements an audit system to track who accesses the data and what operations are invoked. This system will help to detect suspicious

activities or unauthorized access. As an example, it is critical to record who accesses tables in the **MedDB** and **AdminDB** subsets and to track the updates to these data.

These procedures ensure that an organization manage in the proper way data and security, according to the specific needs and responsibilities of each type of user.

5. **Synchronize Shared Tables**: This step identifies shared tables, i.e. tables that appear in distinct database subsets. In this example, multiple copies of several tables exist. Hence, it is essential to synchronize the updates of these tables to ensure data consistency and integrity despite replication.
   When multiple users or divisions can update a shared table, the following steps occur:

   - **Identify Common Tables**: This first step has to identify the tables containing shared data among subsets. For instance, in our healthcare organization, tables such as **PatientTable** (fields: PatientID, MedicalHistory, AssignedDoctorID, AssignedNurseID) and **examsTable** (fields: examID, Results) are shared between the **MedDB** and **PatientDB** subsets, and synchronization is crucial to avoid data inconsistencies.
   - **Implement Synchronization Rules**: It defines rules to implement data synchronization among subsets. These rules cover scenarios such as updates, insertions, and deletions. For example, when medical staff updates patient data in the **MedDB** subset, these changes must be reflected in the **PatientDB** subset so that patients have access to the updated information. Similar synchronization rules have to be established for shared tables among other subsets.
   - **Plan Synchronization**: It plans when data synchronization occurs. For instance, update to critical data should be immediate to ensure immediate alignment. Less critical data can tolerate a weaker consistency. As an example, data update for administrative staff can occur once a day, usually at night. A weekly update may be appropriate for statisticians. The synchronization strategy should consider the needs of all user groups, including Medical Staff, Patients, Doctors, Nurses, and Administrative Staff.

   The choice of the synchronization strategy should tune the synchronization overhead to the urgency and relevance of data for each user group while preserving consistency across different subsets and assuring the integrity of all the organization data.

6. **Optimizing Confinement Robustness**
   The optimization of confinement robustness involves the definition of the degree of physical and logical separation among the database subsets according to data sensitivity and security needs. The steps to follow are:

   - **Data Classification**: It classifies data based on its sensitivity level. For example, personal information (RealName, Address, PayerDetails) and medical history (MedicalHistory) should be classified as highly sensitive, while public or non-sensitive data are classified as less critical.
   - **Assigning Security Levels**: It assigns a security level to each database subset based on data classification. For instance, the financial database (CostsTable) requires the highest security level, while the other database may require a lower one.

- **Allocation Choice**: To show the flexibility of MDDF, we assume that the most cost-effective solution is the one that hosts the **PatientDB** subset on a separate physical machine and maps other databases onto a further physical machine and uses virtualization to separate these databases.
  - **Separate Physical Machine (PatientDB)**: The **PatientDB** subset should be allocated to a dedicated physical server for maximum confinement. This is required when the most robust confinement is a system requirement.
  - **Virtual Machines (VMs)**: We show how to allocate the other subsets onto three virtual machines (VMs) and run them on another physical machine. In this way, VMs offer logical separation within a single shared physical server. The other databases are mapped onto the VMs as follows:
    * **VM Prescription**: A virtual machine to manage prescription data (PrescriptionID, MedicationDetails, Dosage, PrescribingDoctorID, AssociatedCost).
    * **VM Exam Costs**: A virtual machine is devoted to exam cost data (TransactionID, Item, Cost).
    * **Containerization (Mapping and Statistics)**: Within the VM "Exam Costs" machine we allocate to distinct containers the **MappingTable** and **StatisticsTable**. Containers offer lightweight and efficient confinement, striking a balance between complete confinement and resource efficiency.

  This example outlines how MDDF can support distinct degrees of physical and logical confinement across and within physical and virtual machines according to the security needs of each database subset.

5. Comparison of Alternative Synchronization Solutions

This section discusses the overhead of alternative solutions to synchronize tables shared among distinct database subsets. In particular, it compares the features and capabilities of two APIs based on, respectively, triggers and events.

A trigger is a database event that fires the execution of a code fragment as a response to predefined conditions to preserve the integrity of the database, ensuring that some actions are executed when specific changes occur.

An alternative solution adopts SymmetricDS, an open-source tool to support data replication and synchronization among databases. It is used by several companies and organizations to replicate distributed environments, such as bank branches, remote offices, or geographically separated data centers (e.g. content delivery network). In particular, it is used by OpenMRS, a collaborative open-source project to develop software to support the delivery of health care in developing countries.

5.1. Performance Comparison of the Two Solutions

This section compares the performance of the two synchronization solutions: Trigger-API and SymmetricDS. A series of experiments have been implemented to evaluate their effectiveness under various conditions.

Experiment	Trigger-API	SymmetricDS
Latency (ms) - Local Configuration	**59**	95
Latency (ms) - Remote Configuration	**181**	258
Latency (ms) - Distributed Configuration	**800**	2160
CPU Utilization - Light Workload	**5%**	25%
CPU Utilization - Medium Workload	**30%**	60%
CPU Utilization - Heavy Workload	**70%**	90%
Conflict Resolution (ms) - 100% Resolved	**76**	180
Conflict Resolution (ms) - 90% Resolved	**55**	70
Conflict Resolution (ms) - 80% Resolved	**40**	60
Two-way Synchronization (ms) - Main to Remote	**161**	215
Two-way Synchronization (ms) - Remote to Main	**166**	256
Variable Load (tps) - Peak Activity	**63**	51
Variable Load (tps) - Calm Period	37	**49**

Table 2
experiments and results for the two solutions

Table 2 presents the results of various performance metrics for Trigger-API and SymmetricDS.

In the "Latency" experiments, Trigger-API consistently outperforms SymmetricDS across all configurations, showing lower latency in local, remote, and distributed setups. In terms of CPU Utilization, Trigger-API achieves better efficiency, mainly under light and medium workloads, with significantly lower CPU usage compared to SymmetricDS. Trigger-API offers superior performance in handling data conflicts for all percentages of resolved conflicts. Regarding Two-way Synchronization, Trigger-API performs better in both directions, showing lower synchronization times than SymmetricDS. According to Variable Load testing, Trigger-API offers higher transaction rates during peak activity, while SymmetricDS shows slightly better performance during calm periods.

Overall, the table outlines a comprehensive comparison of the performance between Trigger-API and SymmetricDS, highlighting Trigger-API's superior performance in various scenarios.

6. Conclusions

The Multilevel Database Decomposition Framework offers several benefits, including:

- **Preserve privacy:** In healthcare, where sensitive patient information is handled, database decomposition improves security by restricting each user to one subset with the data the user needs to protect the privacy of individuals. In healthcare, where sensitive patient information is handled, this is crucial for complying with regulations such as GDPR [1] and HIPAA [2]. The framework supports the implementation of precise access controls, ensuring that only authorized medical staff can access specific patient information.
- **Reduced Impact of Security Breaches:** The healthcare sector is a prime target for cyber-attacks. Database decomposition limits the impact of security breaches, as compromising one subset does not expose patient records in distinct databases.

- **Reduction of complexity:** By decomposing the database into smaller subsets, users can use simpler data structures that are easier to understand.
- **Improved performance:** Access to smaller, more tailored tables and relationships improves database performance.
- **Efficient Data Retrieval for Medical Research:** Researchers can access specific subsets relevant to their studies, streamlining data retrieval for research purposes. This can contribute to advancements in healthcare through data-driven insights.
- **Scalability for Growing Healthcare Systems:** As healthcare systems expand, the multilevel database decomposition framework allows for scalability. New subsets can be added to accommodate the growing volume of patient data.

As a counterpart, the proposed framework also presents some challenges:

- **Management complexity:** Managing multiple subsets may increase the complexity of database administration and require greater resources for maintenance and updates.
- **Risk of data inconsistency:** Database decomposition requires the adoption of proper synchronization mechanisms with the resulting overhead.

References

[1] European Union, Data protection in the EU, 2023. URL: https://ec.europa.eu/info/law/law-topic/data-protection_en.

[2] D. L. Anthony, A. Appari, M. E. Johnson, Institutionalizing hipaa compliance: Organizations and competing logics in u.s. health care, Journal of Health and Social Behavior 55 (2014) 108–124. URL: https://doi.org/10.1177/0022146513520431. doi:10.1177/0022146513520431, pMID: 24578400.

[3] J. H. Saltzer, M. D. Schroeder, The protection of information in computer systems, Proc. IEEE 63 (1975) 1278–1308.

[4] R. Smith, A contemporary look at saltzer and schroeder's 1975 design principles, IEEE Security & Privacy 10 (2012) 20–25.

[5] M. Campobasso, L. Allodi, Impersonation-as-a-service: Characterizing the emerging criminal infrastructure for user impersonation at scale, in: Proc. of the IEEE Int. Symposium on Secure Software Engineering, volume 1, IEEE, 2006, p. 1.

[6] S. A. Moiz, P. Sailaja, G. Venkataswamy, S. N. Pal, Database replication: A survey of open source and commercial tools, International Journal of Computer Applications 13 (2011) 1–8.

[7] N. Al-Sayid, D. Aldlaeen, Database security threats: A survey study, in: 2013 5th International Conference on Computer Science and Information Technology, 2013, pp. 60–64.

[8] M. Humayun, N. Jhanjhi, M. Almufareh, M. Khalil, Security threat and vulnerability assessment and measurement in secure software development, Computers, Materials and Continua 71 (2022) 5039–5059.

[9] I. Singh, N. Kumar, K. Srinivasa, T. Sharma, V. Kumar, S. Singhal, Database intrusion detection using role and user behavior based risk assessment, Journal of Information Security and Applications 55 (2020) 102654.

[10] S. Ibrahim, A. Zengin, S. Hizal, A. Suaib Akhter, C. Altunkaya, A novel data encryption algorithm to ensure database security, Acta Infologica 7 (2023) 1–16.

[11] A. Cuzzocrea, H. Shahriar, Data masking techniques for nosql database security: A systematic review, in: 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 4467–4473.

[12] I. Linkov, F. Baiardi, M. V. Florin, S. Greer, J. H. Lambert, M. Pollock, B. D. Trump, Applying resilience to hybrid threats, IEEE Security & Privacy 17 (2019) 78–83.

[13] P. Yang, N. Xiong, J. Ren, Data security and privacy protection for cloud storage: A survey, IEEE Access 8 (2020) 131723–131740.

[14] M. Binjubeir, A. A. Ahmed, M. A. B. Ismail, A. S. Sadiq, M. Khurram Khan, Comprehensive survey on big data privacy protection, IEEE Access 8 (2020) 20067–20079.

[15] J. Wu, et al., An access control model for preventing virtual machine escape attack, Future Internet 9 (2017) 20.

[16] A. Y. Wong, et al., On the security of containers: Threat modeling, attack analysis, and mitigation strategies, Computers & Security 128 (2023) 103140.

[17] S. Shringarputale, P. McDaniel, K. Butler, T. La Porta, Co-residency attacks on containers are real, in: Proc. of the 2020 ACM SIGSAC Conf. on Cloud Computing Security Workshop, ACM, 2020, pp. 53–66.