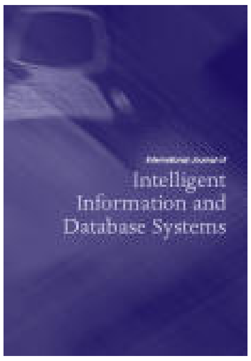


International Journal of Intelligent Information and Database Systems

ISSN online: 1751-5866 - ISSN print: 1751-5858
https://www.inderscience.com/ijiids

Securing big graph databases: an overview of existing access control techniques

Basmah Alzahrani, Asma Cherif, Suhair Alshehri, Abdessamad Imine

DOI: 10.1504/IJIIDS.2024.10063201

Article History:

Received:	30 May 2023
Last revised:	16 September 2023
Accepted:	18 October 2023
Published online:	02 April 2024

Securing big graph databases: an overview of existing access control techniques

Basmah Alzahrani*

Department of Information Technology,
King Abdulaziz University,
Jeddah, Saudi Arabia
Email: balzahrani0271@stu.kau.edu.sa
*Corresponding author

Asma Cherif

Department of Information Technology,
King Abdulaziz University,
Jeddah, Saudi Arabia
and
Center of Excellent in Smart Environment Research,
King Abdulaziz University,
Jeddah, Saudi Arabia
Email: acherif@kau.edu.sa

Suhair Alshehri

Department of Information Technology,
King Abdulaziz University,
Jeddah, Saudi Arabia
Email: sdalshehri@kau.edu.sa

Abdessamad Imine

Université de Lorraine,
CNRS,
Inria, Vandoeuvre-lès-Nancy,
54506 Nancy, France
Email: abdessamad.imine@loria.fr

Abstract: Recently, the rapid evolution of technology has resulted in a significant increase in the volume of data generated by both users and organisations. This, in turn, has given rise to the big data phenomenon. However, traditional relational databases are ill-equipped to handle such vast and complex data. To address this challenge, the NoSQL big data management system has emerged as an efficient alternative. Within this system, the graph database has garnered significant attention from researchers due to its ability to handle complex relationships, such as those found in social networks. However, security remains a critical concern, particularly for sensitive and private data. Therefore, this survey seeks to explore recent solutions for securing graph databases, including techniques such as access control, view-based, and query rewriting approaches, as well as pattern matching algorithms for answering queries. As a result, our survey will contribute to filling the gap in existing research, as none of the previous surveys have examined these specific topics. Additionally, the survey provides recommendations for future research in this area.

Keywords: big data; graph database; access control; view; graph pattern matching; GPM.

Reference to this paper should be made as follows: Alzahrani, B., Cherif, A., Alshehri, S. and Imine, A. (2024) 'Securing big graph databases: an overview of existing access control techniques', *Int. J. Intelligent Information and Database Systems*, Vol. 16, No. 3, pp.286–315.

Biographical notes: Basmah Alzahrani received her BS and MSc in Information Technology from the King Abdulaziz University, Jeddah, Saudi Arabia, in 2016 and 2023, respectively. She is currently a Lecturer in the Information Technology Department, Faculty of Computing and Information Technology, King Abdulaziz University. Her current research interests include big data, graph database and access control.

Asma Cherif received her MSc and PhD in Computer Science from the Lorraine University, France, in 2008 and 2012, respectively. She conducted extensive research with the French Research Laboratory, Inria Nancy Grand Est. Since 2022, she has been leading the IoT Ecosystems Research Team, Center of Excellence in Smart Environment Research (CESER), King Abdulaziz University, Saudi Arabia. She is currently an Associate Professor with the Faculty of Computing and Information Technology, King Abdulaziz University. Her research interests include distributed systems and communication networks, collaborative applications, security, cloud/edge computing, computational intelligence, and the internet of things.

Suhair Alshehri received her PhD in Computing and Information Sciences from the Golisano College of Computing and Information Sciences, Rochester Institute of Technology, in 2014. She is currently an Associate Professor with the Information Technology Department, Faculty of Computing and Information Technology, King Abdulaziz University. Her main research interests include security and privacy in computer and information systems and applied cryptography.

Abdessamad Imine received his MSc and PhD in Computer Science from the University of Sciences and Technology of Oran (USTO), Algeria, and University Henri Poincaré of Nancy, France, respectively. He is currently Associate Professor HdR at Lorraine University and senior researcher at INRIA-LORIA Center of Nancy. His research interests include security for collaborative systems and social networks, formal design of optimistic protocols and formal methods for specifying and verifying distributed systems.

1 Introduction

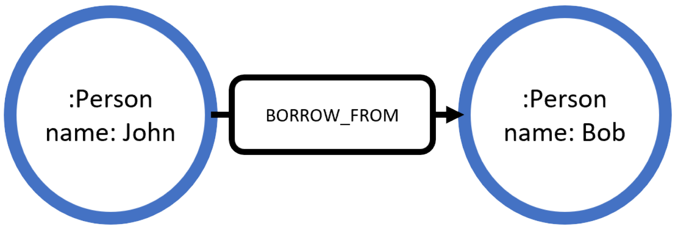
In recent years, technological advancements have rapidly evolved and manifested in various forms, such as automated systems, social networking, and the internet of things (IoT). While these developments make our lives more convenient, they come at a cost from a technical perspective.

Behind the scenes, an enormous amount of data is generated quickly, giving rise to the big data (BD) phenomenon. This phenomenon possesses distinct characteristics, commonly referred to as the '5Vs': volume, variety, velocity, value and veracity (Colombo and Ferrari, 2018). These 5Vs make managing BD using traditional database systems intractable, as they are ill-equipped to handle such large and complex sets of data.

A new type of database management system, called Not Only SQL (NoSQL), has emerged as a powerful and popular system for handling large amounts of heterogeneous data. NoSQL databases offer support for four data models including key-value (e.g., BerkeleyDB), wide-column (e.g., Cassandra), document-oriented (e.g., MongoDB), and graph (e.g., Neo4j) (Morgado et al., 2018). In this survey, we specifically focus on the graph database due to its numerous strengths.

Indeed, graph databases (GDBs) are highly effective in managing complex relationships, especially in social, informational, technological, and biological networks (Angles and Gutierrez, 2008). This is why they have been widely adopted by social media giants such as Google, Facebook, and Twitter (Patil et al., 2018), as they accurately depict real-world entities. The fundamental components of any graph model are nodes (i.e., entities) and edges (i.e., relationships). Most researchers have focused on studying a specific type of graph known as the directed graph (DG), which is characterised by edges with directions. This implies that when considering two nodes A and B connected by a relationship represented by an arrow from A to B, $A \rightarrow B \neq B \rightarrow A$. Figure 1 provides an illustration of a DG, where two nodes are labelled with the names of two individuals and a relation labelled with 'BORROW_FROM'. This graph indicates that John borrowed some money from Bob. The direction of the edge is critical as it conveys different information based on its orientation.

Figure 1 Example of a directed graph (see online version for colours)



1.1 Problem definition and motivations

Although a big-graph database can effectively support the five Vs of BD, it is critical to prioritise security measures to safeguard sensitive and private information stored in GDBs from unauthorised access. By implementing robust security protocols, data integrity can be maintained, and user satisfaction can be enhanced.

Access control (AC) is an essential security service that determines who can perform specific actions on sensitive information. It involves setting policies (i.e., rules) that define eligible users (i.e., subjects) and the actions (e.g., read) they can take on particular data (i.e., objects). For example, a policy may allow certain users to read certain documents but not edit them.

Recent solutions in this field can be categorised can be categorised based on the AC enforcement mechanism into four into four distinct groups:

1 *Engine-level (runtime):* This depends on assigning the access control lists (ACLs) to nodes.

2 *View-based:* This is based on a view-per-role foundation, implying that each role has a view comprising all accessible data.

3 *Pre-processing:* It evaluates the query against the access control policies (ACPs); if it is authorised, then the query is executed, otherwise, either

   a rewriting the query to guarantee inquiring only approved data (i.e., query rewriting)

   b declining the query (i.e., no rewriting).

4 *Post-processing:* It is the opposite of pre-processing. It executes the query, then before reflecting the result to the user, it prunes it by removing inaccessible data (Thimma et al., 2013).

Most recent solutions in the literature fall into categories 1 and 3. This implies that they mainly require traversing the entire original graph. Thus, evaluating the query and retrieving the authorised data will take time. Some solutions combine categories 2 and 3. Although they support the use of views, i.e., there is no need to iterate through the entire original graph. However, they do not support query rewriting approach, i.e., if there is not a complete answer, then there is not even an approximate answer.

AC is one of the aspects related to the graph database. The other aspects are graph database models, graph pattern matching (GPM) algorithms, views, and graph database tools. Generally speaking, graph database models refer to how data is being represented, queried, and consistent. GPM refers to the algorithm which is used in answering the graph query. View refers to a partial part of the original graph. The tools refer to the technologies used in managing the graph database. Indeed, these are the main topics that have been studied by researchers. Nevertheless, the existing surveys only addressed one or two of them. For instance, Angles and Gutierrez (2008) did an excessive survey about graph database models and graph query languages. Patil et al. (2018) have covered the tools and other miscellaneous topics in brief. Hence, neither one of them investigate the researches in AC, GPM, and views though they are crucial and hot topics in recent days.

Therefore, this survey aims to bridge the gap by examining recent solutions that target AC in graph databases. The solutions will be classified based on the aforementioned categories, and the GPM algorithms used in the database will be also discussed. Moreover, this survey will play a crucial role in identifying and addressing gaps in existing research within the field. Table 1 provides a comparison between the current survey and two previous surveys, namely Angles and Gutierrez (2008) and Patil et al. (2018). The table highlights the different topics that have been addressed in each survey, and shows that unlike Angles and Gutierrez (2008) and Patil et al. (2018), our survey investigates GPM algorithms, the recent AC solutions, and the views.

Table 1 Comparison of this survey and other surveys

	GDB models	*GPM algorithms*	*AC*	*Views*	*Tools*
Angles and Gutierrez (2008)	✓	×	×	×	×
Patil et al. (2018)	×	×	×	×	✓
This survey	×	✓	✓	✓	×

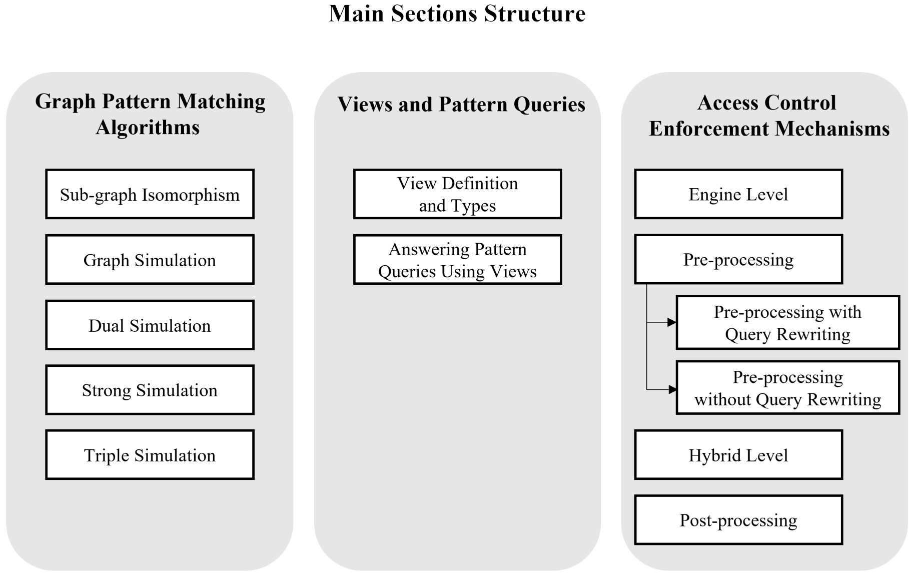
1.2 Contributions

This survey serves as a foundation for gaining a comprehensive understanding of security in graph databases by evaluating the latest solutions in the field. Moreover, it provides a comparative study of existing GPM algorithms. Additionally, it offers valuable research directions and insights for developing future solutions.

1.3 Paper outline

The rest of this paper is arranged as follows, see Figure 2. Section 2 presents the graph matching algorithms. Section 3 introduces the view concept and the main reference that implemented it in GDB. Section 4 shows the recent solutions that implemented AC in graph databases. In Section 5, we discuss the research directions in the area. Finally, Section 6 concludes the paper.

Figure 2 The structure of the paper


Main Sections Structure
Graph Pattern Matching Algorithms
Sub-graph Isomorphism
Graph Simulation
Dual Simulation
Strong Simulation
Triple Simulation
Views and Pattern Queries
View Definition and Types
Answering Pattern Queries Using Views
Access Control Enforcement Mechanisms
Engine Level
Pre-processing
Pre-processing with Query Rewriting
Pre-processing without Query Rewriting
Hybrid Level
Post-processing

2 Graph pattern matching

GPM is a critical concept in the graph database since it is the key to retrieving graph data regardless of the used algorithm. Having this section at the beginning comes from the need to pave the way for the remaining sections since there are related and fundamental terms to be defined, such as data graph (see Definition 1) and pattern queries (see Definition 2) (Halevy, 2001).

Definition 1 (data graph): A data graph is a directed graph $G = (V, E, L)$ where V and E represent a set of nodes, and edges, respectively. L is a function when applied to a node v, it returns its labels $L(v)$.

Definition 2 (pattern query): A pattern query is a directed graph $Q = (V_p, E_p, f_v)$ where V_p and E_p represent a set of pattern nodes and edges, respectively. f_v is a function when applied to a pattern node u, it returns its labels $f_v(u)$.

There are a variety of algorithms that perform GPM. In this paper, five algorithms will be described: subgraph isomorphism, graph simulation, dual simulation, strong simulation, and triple simulation.

2.1 Subgraph isomorphism

Subgraph isomorphism is one of the earliest GPM algorithms. In this algorithm, the subgraph G_s of G matches Q if a bijective function f from Q to G_s exists and satisfies the following:

- Each pattern node $u \in Q$ has a corresponding node in G_s such that u and $f(u)$ have an identical label.
- The edge (u, u') in Q has a match in G_s if and only if $(f(u), f(u'))$ is an edge there.

This algorithm succeeded in preserving the child and parent relationships (i.e., duality). Thus, it preserves the structure of the query pattern. However, sometimes it fails to capture sensible matches due to its restrictions. The size of the matches retrieved is considered exponential which is a drawback. Furthermore, it is an NP-complete problem (Ma et al., 2011).

2.2 Graph simulation

Graph simulation is proposed to overcome the NP-complete problem of subgraph isomorphism (Milner, 1989). In this algorithm, G matches Q if a binary relation $S \subseteq V_p \times V$ exists, where S is a set of matched pairs of nodes, and V_p and V represent a group of pattern nodes and nodes, respectively. In this binary relation:

- For every node $u \in V_p$ there is a corresponding node $v \in V$ so that $(u, v) \in S$.
- For every pair $(u, v) \in S$:

a matched labels $f_v(u) = L(v)$

b every edge $e = (u, u')$ in E_p has a corresponding edge (v, v') in E, so that $(u', v') \in S$.

Although this algorithm succeeded in solving the problem of subgraph isomorphism by reducing the complexity to a quadratic time, it generates results with too large size, and with a structure that may differ from the pattern query. Furthermore, it is incapable of preserving the parent relationship (Ma et al., 2011).

2.3 Dual simulation

As the name suggests, this algorithm extends the graph simulation by adding a condition that maintains both the child and parent relationships (i.e., duality). In this algorithm, G matches Q if a binary relation $S_D \subseteq V_p \times V$ exists, where S is a set of matched pairs of nodes. In this binary relation:

- For every node $u \in V_p$ there is a corresponding node $v \in V$ so that $(u, v) \in S_D$.
- For every pair $(u, v) \in S_D$:

  a matched labels $f_v(u) = L(v)$

  b every edge $e = (u, u')$ in E_p has a corresponding edge (v, v') in E, so that $(u', v') \in S_D$

  c every edge $e = (u'', u)$ in E_p has a corresponding edge (v'', v) in E, so that $(u'', v'') \in S_D$.

All the points listed above are similar to graph simulation except the last one which preserves the parent relationship.

2.4 Strong simulation

Ma et al. (2011) proposed the strong simulation algorithm to overcome the limitations of graph simulation and its extensions that fail to preserve the structure of the pattern query. Furthermore, it aims to bound the number of the retrieved matches and the size of each matched subgraph by proposing the locality notion. Indeed, the definition of locality relies on two terms: distance and diameter.

- *Distance:* is the shortest length of the undirected path from nodes n to n' in G, denoted $dis(n, n')$.
- *Diameter:* is the longest shortest path of all pairs of nodes in G, denoted $d_G = max(dis(n, n'))$.

In this algorithm, G matches Q, if there is a subgraph G_s of G with a node n at the centre such that:

- For every node n' in G_s, $dis(n, n') \leq d_Q$.
- Q matches G_s via dual simulation with the maximum match relation S.

Strong simulation has the same complexity time as graph simulation extensions which is cubic-time.

2.5 Triple simulation

Strong simulation succeeded in preserving duality and in enforcing the locality notion. However, if we have two pattern graphs $Q_1 = A \rightarrow B$ and $Q_2 = B \leftarrow A \rightarrow B$, where A and B are labelled nodes. Intuitively, the data node that meets A in Q_1 must have a minimum of one child labelled B, whereas in Q_2 it must have a minimum of two child nodes labelled with B. But applying strong simulation returns only $A \rightarrow B$ as a match for both, thus it failed to make the distinction between Q_1 and Q_2. Hence, it does not preserve the topological constraint named the label-repetition (LR) constraint in which two or more nodes are having the same label. Indeed, subgraph isomorphism can preserve this constraint, but as mentioned earlier it is an NP-complete problem.

To overcome this problem, Mahfoud (2018) proposed a graph simulation extension called triple simulation that is executed in quartic time. It supports the duality and locality notions, and it preserves the LR constraint. When the LR constraint is defined over a node n in the pattern graph Q with label l. This means that

1 there are at least two children (resp. parents) labelled with l

2 any match v of u in the data graph G must have at least two children (resp. parents) that match the children (resp. parents) of n.

In this algorithm, G matches Q, if there is a binary relation $S_T \subseteq V_p \times V$, so that:

- Every node $u \in V_p$ has a corresponding node $v \in V$ such that $(u, v) \in S_T$.
- For every $(u, v) \in S_T$, $f_v(u) = L(v)$.
- For every $(u, v) \in S_T$ and for all edges $(u, u_1), ..., (u, u_n) \in E_p$, there exists a minimum of n distinct children of v, $(v, v_1), ..., (v, v_n) \in E$, such that: $(u_1, v_1), ..., (u_n, v_n) \in S_T$.
- For every $(u, v) \in S_T$ and for all edges $(u_1, u), ..., (u_n, u) \in E_p$, there exists a minimum of n distinct parents of v, $(v_1, v), ..., (v_n, v) \in E$, such that: $(u_1, v_1), ..., (u_n, v_n) \in S_T$.

To satisfy the LR constraints, there is an essential condition based on the use of the bipartite graph. In this graph, there are two sets of nodes S_1 that contains the children (resp. parents) of n in Q that are concerned by the LR constraint, and S_2 that contains the children (resp. parents) of v in G that potentially matches n. The LR constraint is satisfied if and only if there is a complete match over the bipartite graph, i.e., each node in S_1 has a match in S_2. Note that to check the LR constraints of children and parents of n, there will be two bipartite graphs.

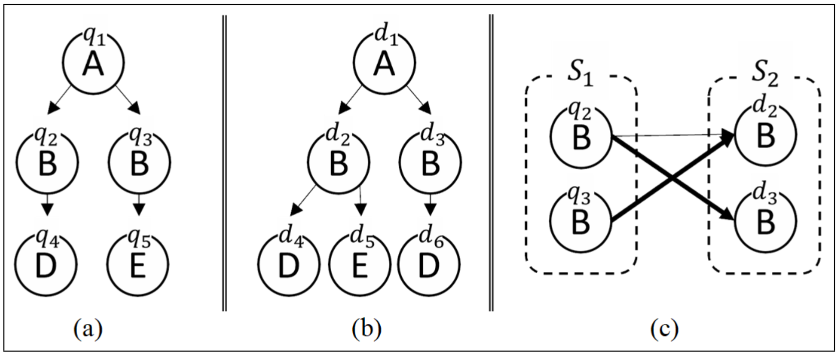
Example 1 illustrates how to enforce the LR constraint using the bipartite graph.

Example 1: Consider the pattern query $Q = (V_q, E_q)$ and the data graph $G = (V, E)$ depicted in Figures 3(a) and 3(b). The LR constraint is defined over the node q_1 which means that the potential match of q_1 must have at least two children with label B.

Hence d_1 is the potential match. To inspect the LR constraint, the bipartite graph is depicted in Figure 3(c). The set S_1 contains the children of q_1 that are concerned by the LR constraint, while S_2 contains the children of d_1 that are the potential matches. Figure 3(c) illustrates that there is a complete match over the bipartite graph, thus the data graph G satisfies the LR constraint of Q.

However, the triple simulation might be infeasible when applied to massive graphs such as Facebook, where active users exceed one billion each month (Pang and Zhang, 2015). Thus, as future work, the author intends to extend it with some optimisation techniques.

Figure 3 Example of the constraint inspection using a bipartite graph, (a) pattern query Q (b) data graph G (c) bipartite graph BG



2.6 Summary of GPM algorithms

Table 2 summarises the previously mentioned algorithms based on the following criteria:

- *Denotation:* The algorithm symbol.
- *Complexity:* The time required for an algorithm to perform its task from the first step to the last one.
- *Number of matches:* From G to Q_s in terms of nodes and edges.
- *Result structure:* The style of the result and how much it is similar to the pattern query.
- *Duality:* To maintain the child and parent relationships.
- *Locality:* The matches are within a subgraph with a specific radius (Ma et al., 2011).
- *Label repetition (LR) constraint:* A topological constraint where two or more nodes have the same label (Mahfoud, 2018).

Summarising the graph pattern matching algorithms in Table 2, we can see that subgraph isomorphism (Ma et al., 2011) has the highest complexity and the largest retrieved number of matching subgraphs. However, it supports all duality, locality, and LR constraints similar to triple simulation (Mahfoud, 2018). These constraints are not supported by the graph simulation (Ma et al., 2011) which reduces the number of matches and the time complexity of NP-complete in subgraph isomorphism to quadratic time. The dual simulation (Ma et al., 2011) has a higher complexity than graph simulation while retaining duality. Among all the algorithms, strong simulation (Ma et al., 2011) and triple simulation are the only ones that successfully recover the result with the exact structure of the query pattern.

Table 2 Comparison of GPM algorithms

	Subgraph isomorphism (Ma et al., 2011)	*Graph simulation (Ma et al., 2011)*	*Dual simulation (Ma et al., 2011)*	*Strong simulation (Ma et al., 2011)*	*Triple simulation (Mahfoud, 2018)*
Denotation	$G_s \prec_{iso} Q$	$Q \prec G$	$Q \prec_D G$	$Q \prec_D^L G$	$Q \prec_T G$
Complexity	NP-complete problem	Quadratic time	Cubic-time	Cubic-time	Quartic
Number of matches	Exponential matched subgraphs	Single match relation but too large	Single match relation but too large	Linear the number of V bounds the number of matched subgraphs, and each subgraph is bounded by a diameter	-
Result structure	Preserves the structure of the query pattern, but sometimes fails to capture sensible matches due to its restrictions	May differs from the query pattern	May differs from the query pattern	Preserves the structure of the query pattern	Preserves the structure of the query pattern
Duality	✓	×	✓	✓	✓
Locality	✓	×	×	✓	✓
LR constraint	✓	×	×	×	✓

3 Views and pattern queries

In this section, we introduce the view notion and its types. Furthermore, we explain one of the key references in this area that used views for answering pattern queries.

3.1 View definition and types

The notion of views has shown its strengths and value in relational databases (Halevy, 2001). Views act like a window that shows part of the scene. So, if we have a massive database, and there are multiple queries issued frequently over this database, then this is considered time-consuming because at each time the queries are computed over the original big database. Instead of this, we can accelerate the computation process by creating views that represent the frequently queried data (da Trindade et al., 2020). Creating a view is done by running the desired query over the underlying database, then retrieving the result. We distinguish between two types of views: virtual and physical views, based on the way of dealing with the result.

A virtual view (or just view) means the result is not stored in the database. This requires re-computing the query over the underlying database each time the view is referenced (Gupta and Mumick, 1999). Though this affects the performance, in this type of view, there is no need to worry about view maintenance; because the data is always up-to-date.

On the other hand, in the physical view (i.e., materialised view), the data is physically stored in the database. So, once a query is issued over this view the result is retrieved directly from the materialised view (MV) without accessing the underlying database. Thus, this type of view improves the performance and speeds up the result retrieving process (Gutiérrez et al., 1994). However, it affects the storage space and requires periodic maintenance since whenever a change happens to the underlying database, it will not be automatically reflected on the MV. Indeed, the view here is a separate copy of the base data. Table 3 represents a comparison between both types of views based on the following criteria:

- *Naming:* A synonym name.
- *The result:* Whether it is stored in the database or not.
- *Performance:* The speed at which the data is retrieved.
- *Up-to-date:* Does the view always have updated data?
- *Storage:* For storing the view.

Table 3 Comparison of virtual and physical views

	Virtual view	*Physical view*
Naming	View	Materialised view (MV)
The result	Not stored in the database (not pre-computed)	Stored physically in the database (pre-computed)
Performance	Slow	High
Up-to-date	Always	Requires a periodic maintenance
Storage	No need	Affects the storage space

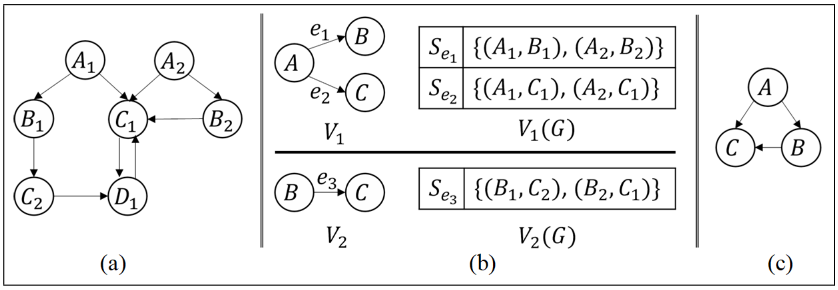
The following section demonstrates how views are used in GDB for answering pattern queries.

3.2 Answering pattern queries using views

In light of the above benefits of views, GDBs have also adopted the view notion. However, it is still in its early stages (Fan et al., 2016). In this section, we introduce Fan et al. (2016) which has a significant contribution to this area. In a nutshell, this paper addresses the problem of using views in answering graph pattern queries. The authors show that a pattern query Q_s can be answered using a set of views $\mathcal{V} = \{V_1, ..., V_n\}$ via graph simulation (see Section 2), if and only if Q_s is contained in $\mathcal{V}$ as illustrated in Example 2. This led them to extend the conventional notion of query containment used in relational DB, to be pattern containment.

Example 2: Figures 4(a) and 4(c) represents an example of a graph G and a pattern query Q_s (see Definitions 1–2, Section 2). Here we need to define what is the view definition and its extension. View (V) is defined as a pattern query, and the group of match sets (S_{e_i}) for all edges (e_i) in this view is named as a view extension $V(G)$, see Figure 4(b).

Figure 4 Using views in answering pattern queries, (a) data graph G (b) view definition V and view extension $V(G)$ (c) pattern query Q_s


A_1
A_2
B_1
C_1
B_2
C_2
D_1
(a)
e_1
A
B
e_2
C
V_1
S_{e_1} $\{(A_1, B_1), (A_2, B_2)\}$
S_{e_2} $\{(A_1, C_1), (A_2, C_1)\}$
$V_1(G)$
B
e_3
C
V_2
S_{e_3} $\{(B_1, C_2), (B_2, C_1)\}$
$V_2(G)$
(b)
A
C
B
(c)

Now, to check the containment, one of the three algorithms can be used: contain, minimal containment, and minimum containment. Moreover, the last two algorithms help in selecting views for answering Q_s. All of them will be described in what follows.

- *Contain algorithm:* Given a pattern query Q_s with an edge set E_p and a set of view definitions $\mathcal{V}$, the algorithm computes for each view definition V a view match $M_V^{Q_s}$ from V to Q_s where the view match $M_V^{Q_s}$ is the union of S_{e_V} for all e_V in V. If we find that the union of all $M_V^{Q_s} = E_p$, then the boolean output will be true for $Q_s \subseteq \mathcal{V}$. Otherwise, it is false.

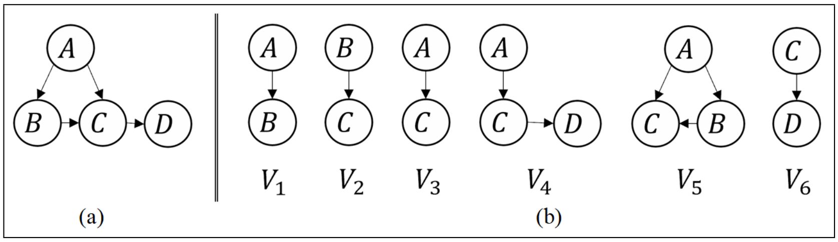
  Example 3: Consider the pattern query Q_s and view definitions $V_1 - V_6$ illustrated in Figures 5(a) and 5(b), respectively. Then the view match $M_V^{Q_s}$ values from $\mathcal{V}$ to Q_s are shown in Table 4. Such values indicate that Q_s is contained in $\mathcal{V}$, thus the output of the algorithm is true.

Table 4 View match values $M_V^{Q_s}$ from the set of view definitions $\mathcal{V}$ to the pattern query Q_s

V_i	$M_{V_i}^{Q_s}$
V_1	$\{(A, B)\}$
V_2	$\{(B, C)\}$
V_3	$\{(A, C)\}$
V_4	$\{(A, C), (C, D)\}$
V_5	$\{(A, C), (A, B), (B, C)\}$
V_6	$\{(C, D)\}$

The reversion of the $M_V^{Q_s}$ relation is called a mapping λ. It works from Q_s to $\mathcal{V}$. So that, for each edge $e_p \in E_p, \lambda(e_p)$ is the set of edges e' in $\mathcal{V}$. This mapping λ will be used as an input for the algorithm MatchJoin that will be explained in the coming few lines.

Figure 5 Containment verification of pattern query Q_s in the set of view definitions $\mathcal{V}$, (a) pattern query Q_s (b) view definitions $\mathcal{V}$



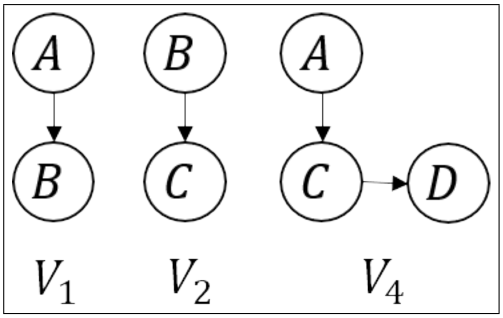
- *Minimal containment algorithm:* This algorithm aims to find the minimal subset $\mathcal{V}'$ of $\mathcal{V}$ that contains Q_s such that:

  a $Q_s \subseteq \mathcal{V}'$

  b for any subset $\mathcal{V}''$ of $\mathcal{V}'$, $Q_s \nsubseteq \mathcal{V}''$.

  Example 4: Consider again Q_s and $\mathcal{V}$ shown in Figure 5. The output of applying the minimal algorithm is a subset $\mathcal{V}' = \{V_1, V_2, V_4\}$, as shown in Figure 6.

Figure 6 The result of applying the minimal algorithm on pattern query Q_s and the set of view definitions $\mathcal{V}$



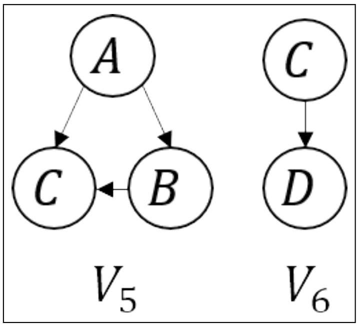
- *Minimum containment algorithm:* This algorithm aims to find the minimum subset $\mathcal{V}'$ of $\mathcal{V}$ that contains Q_s such that:

  a $Q_s \subseteq \mathcal{V}'$

  b for any subset $\mathcal{V}''$ of $\mathcal{V}'$, if $Q_s \subseteq \mathcal{V}''$, then card($\mathcal{V}'$) $\leq$ card($\mathcal{V}''$).

  Example 5: Applying the minimum algorithm on Q_s and $\mathcal{V}$ depicted in Figure 5 results in two views only V_5 and V_6 as shown in Figure 7.

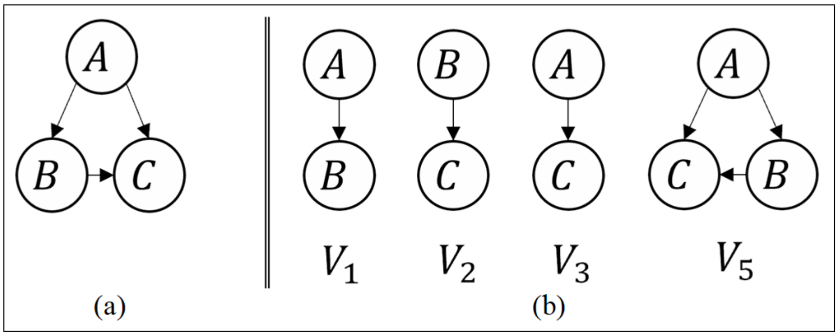
Figure 7 The result of applying the minimum algorithm on pattern query Q_s and the set of view definitions $\mathcal{V}$



- *Maximal algorithm:* In all of the above-mentioned algorithms, Q_s is contained in $\mathcal{V}$, thus Q_s can be answered using the views. However, if Q_s is not contained in $\mathcal{V}$, then no answer is retrieved. To overcome this problem, when Q_s is not contained in $\mathcal{V}$, the maximal algorithm will look for the largest part Q'_s of Q_s that is contained in $\mathcal{V}$. As a result, a rewritten query Q'_s is produced which can have an approximate answer using $\mathcal{V}$.

  Example 6: Given Q_s in Figure 5(a) and $\mathcal{V}$ in Figure 8(b) defined, the containment algorithm will return false. Therefore, the maximal algorithm will rewrite Q_s to be Q'_s, as shown in Figure 8(a). Thus, Q'_s is now contained in $\mathcal{V}$ and can have an approximate answer.

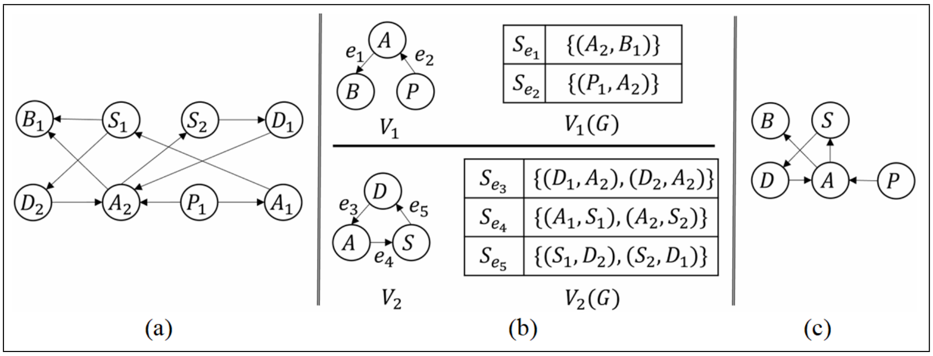
Figure 8 Pattern query rewriting for containment enforcement in the set of view definitions $\mathcal{V}$, (a) a rewritten Q'_s (b) view definitions $\mathcal{V}$



- *MatchJoin algorithm:* It is the algorithm responsible for computing $Q_s(G)$ by joining the views $V_i(G)$ resulting from the mapping λ.

  Example 7: Consider the data graph G, set of view definitions $\mathcal{V} = \{V_1, V_2\}$ with their extensions $\mathcal{V}(G) = \{V_1(G), V_2(G)\}$, and pattern query Q_s depicted in Figure 9. First, a mapping λ from Q_s to $\mathcal{V}$ will verify that $Q_s \subseteq \mathcal{V}$ by mapping (A, B), (P, A) to e_1, e_2 in V_1, respectively; and (D, A), (A, S), (S, D) to e_3, e_4, e_5 in V_2, respectively. Then, MatchJoin merges view matches guided by , and removes invalid matches for edges in Q_s such as (A_1, S_1) from S_{e_4}. This will lead to the removal of (S_1, D_2) from S_{e_5}, and (D_2, A_2) from S_{e_3}. Finally, MatchJoin returns the matches shown in Table 5 as the final result $Q_s(G)$.

Figure 9 Answering pattern queries using views, (a) data graph G (b) view definition V and view extension $V(G)$ (c) pattern query Q_s



S_{e_1}	$\{(A_2, B_1)\}$
S_{e_2}	$\{(P_1, A_2)\}$

S_{e_3}	$\{(D_1, A_2), (D_2, A_2)\}$
S_{e_4}	$\{(A_1, S_1), (A_2, S_2)\}$
S_{e_5}	$\{(S_1, D_2), (S_2, D_1)\}$

Table 5 The result of using the MatchJoin algorithm to evaluate the pattern query Q_s using views

Edge	*Matches*
(P, A)	$\{(P_1, A_2)\}$
(D, A)	$\{(D_1, A_2)\}$
(S, D)	$\{(S_2, D_1)\}$
(A, B)	$\{(A_2, B_1)\}$
(A, S)	$\{(A_2, S_2)\}$

- *Matching algorithms summary:* Table 6 provides a brief description along with the algorithm complexity for all of the aforementioned algorithms.

  After describing the algorithms used to answer Q_s using a set of views $\mathcal{V}$, it is worth mentioning that the complexity of computing Q_s over the big graph G (i.e., no views) is $O(|Q_s|^2 + |Q_s||G| + |G|^2)$ time, in the size of $|G|$ which is very large. Table 7 illustrates the differences between answering Q_s using G and $\mathcal{V}(G)$ based on the following criteria:

  a *Complexity:* The time required to answer Q_s.

  b *Size:* The number of nodes and edges.

  c *Accessing G:* The need to access the underlying graph.

  d *Cost (in terms of time):* More time means more cost.

  e *Speed:* Traversal speed.

Table 6 Summary of matching algorithms

Algorithm	*Description*	*Complexity*
Mapping λ	For every edge e_p of Q_s, $\lambda(e_p)$ is a set of edges e' from the view definitions in $\mathcal{V}$.	Computed within the complexity of the containment algorithms.
Contain	Checks the containment of Q_s in $\mathcal{V}$. $Q_s \subseteq \mathcal{V}$.	$O(card(\mathcal{V})\|Q_s\|^2 + \|\mathcal{V}\|^2 + \|Q_s\|\|\mathcal{V}\|)$ time. Contain + computing mapping λ from Q_s to $\mathcal{V}$.
Minimal containment	Finds the minimal subset $\mathcal{V}'$ of $\mathcal{V}$ that contains Q_s: 1 $Q_s \subseteq \mathcal{V}'$ 2 for any subset $\mathcal{V}''$ of $\mathcal{V}'$,$Q_s \nsubseteq \mathcal{V}''$.	$O(card(\mathcal{V})\|Q_s\|^2 + \|\mathcal{V}\|^2 + \|Q_s\|\|\mathcal{V}\|)$ time. Minimal + computing mapping λ from Q_s to $\mathcal{V}'$.
Minimum containment	Finds a subset $\mathcal{V}'$ of $\mathcal{V}$ such that: 1 $Q_s \subseteq \mathcal{V}'$ 2 for any subset $\mathcal{V}''$ of $\mathcal{V}'$, if $Q_s \subseteq \mathcal{V}''$, then $card(\mathcal{V}') \leq card(\mathcal{V}'')$.	NP-complete and APX- hard but it is approximable within $O(\log \|E_p\|)$ in $O(card(\mathcal{V})\|Q_s\|^2 + \|\mathcal{V}\|^2 + \|Q_s\|\|\mathcal{V}\| + (\|Q_s\| \cdot card(\mathcal{V}))^{3/2})$ time.
Maximal	Identifies a maximal part Q_s' of Q_s that can be approximately answered by using $\mathcal{V}$.	$O(card(\mathcal{V})\|Q_s\|^2 + \|\mathcal{V}\|^2 + \|Q_s\|\|\mathcal{V}\|)$ time.
MatchJoin	Computes $Q_s(G)$ by joining views $V_i(G)$ as guided by λ.	$O(\|Q_s\|\|\mathcal{V}(G)\| + \|\mathcal{V}(G)\|^2)$ time.

Notes: $|Q_s|$: The number of all nodes.
$|\mathcal{V}|$: The total $|V|$ in $\mathcal{V}$.
$card(\mathcal{V})$: The number of view definitions V in the set of view definitions $\mathcal{V}$.
$|E_p|$: The number of edges in Q_s.

Table 7 Comparison of answering pattern query Q_s using graph G and the set of view extensions $\mathcal{V}(G)$

Criteria	*Using the original graph G (i.e., no views)*	*Using views $\mathcal{V}(G)$*
Complexity	$O(\|Q_s\|^2 + \|Q_s\|\|G\| + \|G\|^2)$ time	$O(\|Q_s\|\|\mathcal{V}(G)\| + \|\mathcal{V}(G)\|^2)$ time (quadratic time)
Size	$\|G\|$ is large	$\|\mathcal{V}(G)\|$ is much smaller
Accessing G	Mandatory	No need
Cost (in terms of time)	Costly	Cost-effective
Speed	Slow	Faster

By referring to Table 7, we can observe that using views in answering pattern queries yields prompt answers, and offers superior memory management compared to relying solely on the original graph.

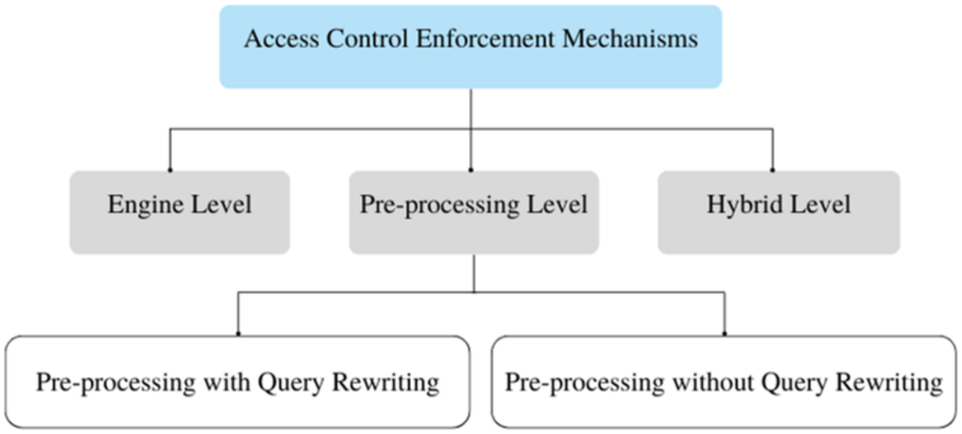
4 Access control enforcement mechanisms

In this section, we present the recent solutions for enforcing AC over the data graph. These solutions vary between research projects, commercial solutions, and patents. Moreover, they differ in terms of the AC model used.

The three traditional AC models adopted by some solutions are the identity-based model (IBAC), the role-based model (RBAC), and the attribute-based model (ABAC). In IBAC, each object has an ACL that specifies the identity of the authorised subjects. So, accessing a secured object depends on having a match between the subject identity and the identity placed in the ACL. Thus, the permissions here need to be managed on an individual basis (Hu et al., 2014).

In RBAC, the admin assigns a set of privileges to a subject based on his/her role. These privileges specify the allowed actions to be performed on specific objects. It is a predetermined process (Hu et al., 2014). For instance, a user with an advisor role can add and drop courses for a student under his/her supervision.

Figure 10 The state-of-the-art solutions taxonomy (see online version for colours)


Access Control Enforcement Mechanisms
Engine Level
Pre-processing Level
Hybrid Level
Pre-processing with Query Rewriting
Pre-processing without Query Rewriting

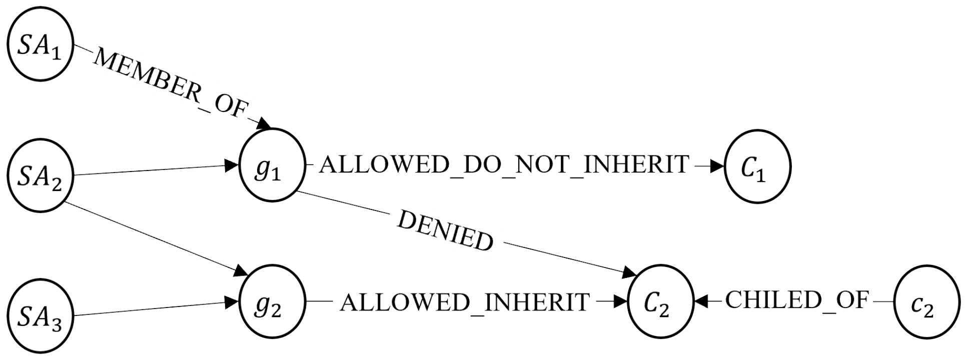
In ABAC, access is permitted or denied based on the assigned attributes of the subject, object, and context. Besides the policies that the admin set are based on these attributes (Hu et al., 2014).

In the following, we classify the state-of-the-art solutions based on the AC enforcement mechanism into engine-level (runtime), pre-processing, and hybrid (i.e., based on views and pre-processing) models, see Figure 10.

4.1 Engine-level (runtime) access control

Bastani (n.d.) proposed the use of fine-grained relationships (FGR) to represent the access control rule (ACR), such as ALLOWED_DO_NOT_INHERIT, DENIED, ALLOWED_INHERIT. They found that this type of relationship is almost twice faster than their counterparts in coarse relationships. The equivalent representation in the coarse relationship will be a property named permission followed by two boolean attributes allowed and inherit. Figure 11 illustrates how the model works, so if a security administrator group (g_i) is connected to a company (C_i) with an ALLOWED_INHERIT relationship, ALLOWED means that the admin (SA_i) is allowed to manage that company. INHERIT means all children of the same company (c_i) inherit this permission. In this model, creating a node that takes as input all the subjects sharing the same security permissions reduces the graph complexity. However, we think having a relationship with DENIED permission is equivalent to dropping its edge. So, it is unnecessary.

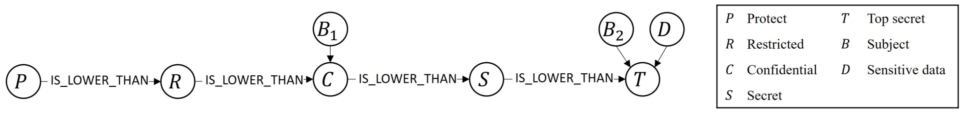
Figure 11 Engine-level access control model with fine-grained relationships



Similar to Bastani (n.d.), the model in Bramley (2015) supports the FGR. However, the core idea is about having classification levels as nodes such as protect (P), restricted (R), confidential (C), secret (S), and top-secret (T) nodes. Both nodes the subject (B) who has access permission and the object (D) that need to be secured must have a relationship with the security nodes. The classification levels are linked in a hierarchical manner using a relationship called IS_LOWER_THAN. This arrangement allows the user to access the objects having the same security level or lower. For instance, in Figure 12, the subject (B_2) can access object (D). While the subject (B_1) cannot access the same object (D) due to its security level.

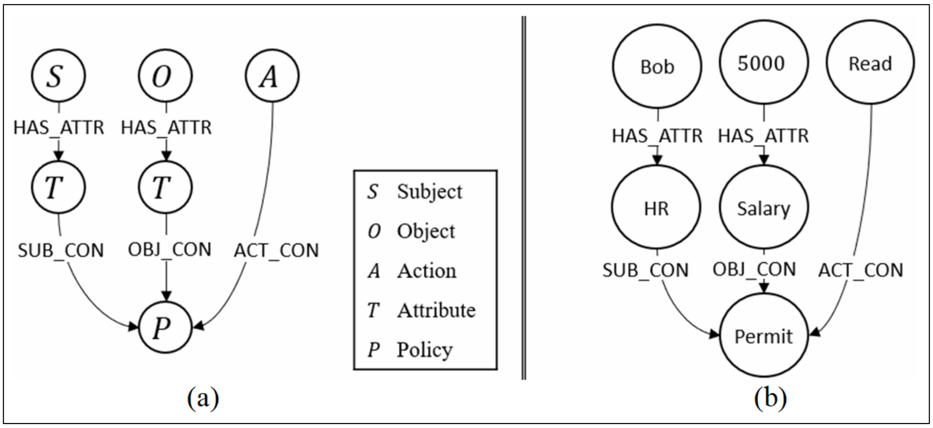
Regarding the performance, the suggested hierarchical classification has a negative impact because it adds the number of hops to check the access authority.

Figure 12 AC model with classification levels as nodes



Ahmadi and Small (2019) proposed a graph model that shows how ABAC policies can be expressed and evaluated using graphs. The main components of any AC policy are subject (S), object (O), and action (A). The subject is the user who asks to perform a specific action (i.e., operation) over the object (i.e., sensitive data). Here, they are named primitives. Each primitive can be described and connected to one or more attribute nodes (T) through a relationship named HAS_ATTR. Now, to build up the policies within the graph, the authors used a policy node (P) with two properties either permit or deny. Furthermore, to connect between the primitives/attributes and the policy node, there are three kinds of relationships SUB_CON, OBJ_CON, and ACT_CON which represent the access conditions based on the subject, object, and action, respectively. Figure 13(a) illustrates the ABAC graph model. Note that if there is an access policy that permits/denies both read and write actions, then both primitive nodes can be linked to one node labelled with 'full access' through the HAS_ATTR relationship. Now, let us demonstrate the model with an example, assume there is an access policy that says: human resources (HR) employees can read employees' salaries. Thus, the graph will be the same as Figure 13(b).

Figure 13 Graph model implementation of attribute-based access control, (a) the general graph model of ABAC (b) an example of ABAC graph model



Dhia (2013) proposed an AC model that targets the online social network (OSN) where users share their data (e.g., personal information, photos, contacts, etc.) with other users in the network. The proper graph model for OSN is the directed property graph where nodes and edges can have attributes (e.g., age, gender, etc.). The author built the model based on two things:

1 the owner privacy preferences (i.e., AC rules)

2 the reachability constraints.

Reachability is a well-known problem in GDB. Having a reachability query means looking for a path that allows u to reach v, where u and v are nodes in graph G (Jin et al., 2010). The author considered a specific type of reachability namely *distance and reachability queries with constraints* that considers the edge labels, distance, direction, etc.

Figure 14 shows the main components of the AC model. The definition of the subject, the action, and the object were mentioned above. The remaining components are defined as follows:

- The owner: The user who owns the object.
- The access rules (ARs): The ACP is based on the owner's privacy preferences and the reachability constraints.
- The reference monitor: A software module that takes the subject request and the AC rules as input to make the access decision whether permit or deny.

The AR is composed of the constraints that must be satisfied in order to permit the subject access. The AR syntax is as follows:

$$AR = (u, r, P, C)$$

where u is the owner of the resource r. C is the list of constraints over the requester attributes (i.e., age > 20). $P = p_1, ..., p_n$ represents the list of constraints over the path between the requester and the owner. Each p_i is defined as triple $p_i = (l, dir, I)$ where l is the edge label in $\sum$, $dir \in \{\leftarrow, \rightarrow, \Leftrightarrow\}$ is the direction of p_i, and $I = (\min, \max)$ represents the minimum and the maximum length of p_i as a pair of integer values.

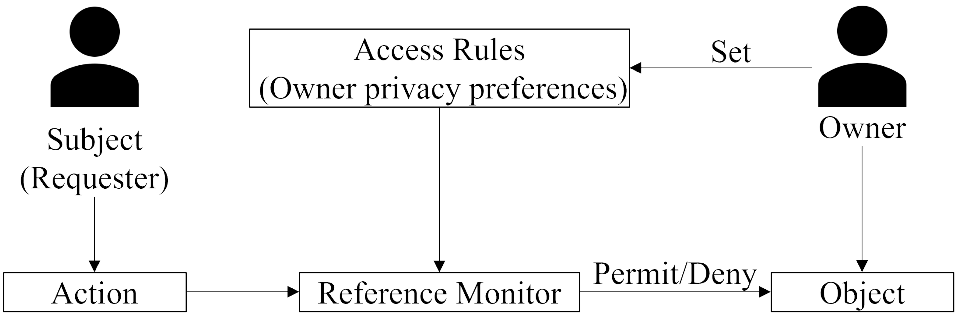
Let's demonstrate an example from the OSN subgraph depicted in Figure 15. Suppose Bill wants to post an advertisement (ad) on his social media account, and just his direct friends living in the USA are allowed to see the post. In this case, the AR will be written as the following:

$$AR_1 = (Bill, ad, ('friend', \rightarrow, (1, 1)), [location = USA])$$

Based on this AR, Bob is the only one allowed to access Bill's post. If Bill wants to make the ad post available to the friends of his friends (i.e., indirect relationship), then the revised AR will be

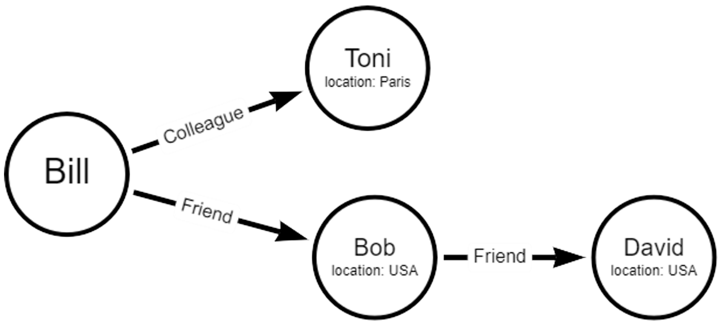
$$AR_1 = (Bill, ad, ('friend', \rightarrow, (1, 2)), [location = USA])$$

Figure 14 Reachability-based access control model



In this work, the AC enforcement is made on the fly when the subject requests a resource. Moreover, the ACRs here differ from Bastani (n.d.), Bramley (2015) and Ahmadi and Small (2019), they are not defined as nodes or edges in the original graph.

Figure 15 An OSN subgraph example



As a drawback, all the solutions under this category still answer the queries by traversing the original graph (i.e., no views), which has a massive size. Thus, it is time-consuming.

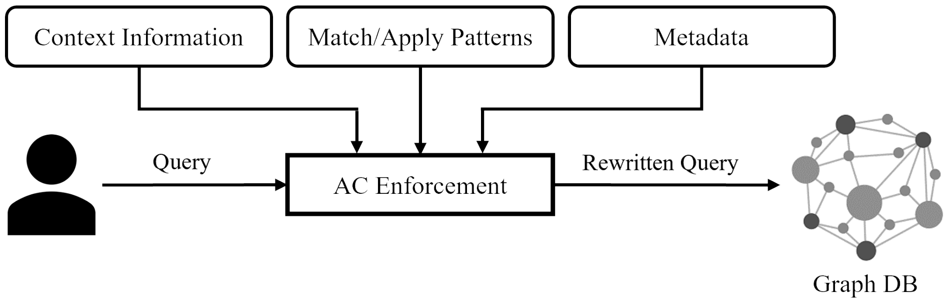
4.2 Pre-processing level access control

The main technique used in this category is query rewriting. Such a technique allows reformulating the original query by either removing unauthorised parts (Thimma et al., 2013), adding security parts (Yalamanchi et al., 2012), or even making the query refers to views (Fan et al., 2016). The following two subsections classify the proposed pre-processing-based solutions based on whether they use rewriting or not.

4.2.1 Pre-processing with query rewriting

Yalamanchi et al. (2012) invented a technique that focuses on controlling access to a specific part of the resource description framework (RDF) data model, instances of classes or properties. RDF is recommended by W3C. It is used to represent metadata as a set of triples. A triple is composed of three parts: subject, object, and predicate (or property) (Kirrane et al., 2020). The subject and object are represented as nodes, while the property is represented as a link between these nodes to describe their relationship (Angles and Gutierrez, 2008). The technique proposed in Yalamanchi et al. (2012) can be applied to any graph model.

Figure 16 An access control model for graph databases



The security policy is composed of three main elements: the graph metadata, the access constraints that restrict access to the instance data, and the session context information, which could be used to support the enforcement of dynamic access constraints based on the runtime values, see Figure 16. The constraint in the security policy is represented as a pair of a match pattern and an apply pattern. Both are in the compiled form. The match pattern, as the name indicates, is the pattern that matches the query in terms of resources. The apply pattern is appended to the query during the query rewriting process as security conditions. Sometimes we need to restrict access to a property instance based on a specific value of the property's object. The FILTER clause in the SPARQL query language can handle this. The following steps describe how the system works in brief:

1 The user issues a query.

2 The AC enforcement receives the query.

3 It looks for the corresponding metadata, then for the corresponding match pattern.

4 It uses the apply pattern paired with the match pattern and the session context information in the query rewriting process.

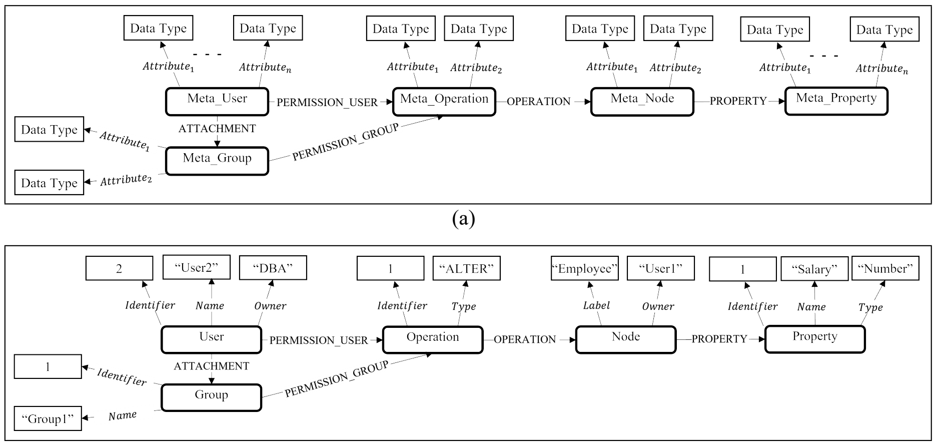
5 It retrieves the data that the user is authorised to access.

4.2.2 Pre-processing without query rewriting

Morgado et al. (2018) proposed a security model that facilitates and guides the development of graph-based applications. The model is based on meta-data and supports both data definition language (DDL), e.g., create and alter, and data manipulation language (DML), e.g., insert, update, and delete. The model is composed of five types of meta-data: meta_user, meta_group, meta_operation, meta_node, and meta_property, see Figure 17(a). Each one of them is described through a set of attributes.

Figure 17(b) shows an instance of the model where the DBA creates the account of User2 which makes him the owner. User2 has ALTER permission to modify the employee salary. When the plugin layer (AC) where the model is placed receives a commit transaction from User2, it will check whether the required operation is permitted or not. In the former case, it will execute the commit, while in the latter, it will roll-back the transaction.

Figure 17 A security model for access control in graph databases, (a) the general AC model for GDBMS (b) an instance of the model



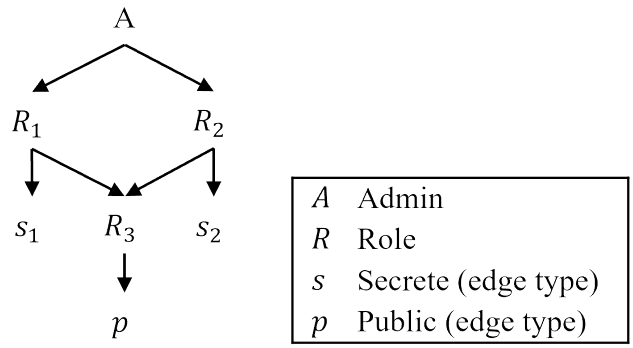
The model was implemented using RDF but is applicable to other graph models, such as property graph. Moreover, it supports the concept of group permissions similar to Bastani (n.d.). The authors were focusing on the feasibility of the model, rather than the performance, therefore, they implement it at the plugin layer offered by Neo4j, which works as an intermediate layer between the querying system and GDB.

4.3 Hybrid level access control

In this section, we will explain the papers that combine both categories the view-based and pre-processing for the AC enforcement.

Views and pre-processing without rewriting are found in the work of Akkiraju et al. (2016) and Hosseinzadeh Kassani et al. (2020). In Akkiraju et al. (2016), they invented an AC technique that works at the edge level. The edge will be associated with one or more facts. The fact is composed of a subject, predicate, and object. The identification of an edge is based on two parameters (source node ID, target node ID). Moreover, the edge will have a type, a pointer to a node, and an access control list (ACL) properties (e.g., type: hasSkill, ACL: ALL). ACL identifies the roles of authorised users, who are permitted to access the fact. Furthermore, a lattice is created for mapping the roles with the edge types. The roles are arranged in a hierarchical order based on their access privileges. So, $roleR_1 \geq roleR_2$, this formula is true when the role R_1 has privileges more than or equal to the role R_2, see Figure 18.

Figure 18 A lattice for mapping the roles with the edge types



There are three approaches to populate the value of the ACL property:

1 Populate ACL based on the edge type wherein the edges with the same type will have the same ACL.

2 Override the ACL by writing a query that retrieves the required edges to update their ACL.

3 ACL is assigned to derived edges as the group of all authorised users of the edges passed during the derivation process.

Enforcing AC is done through the following two steps:

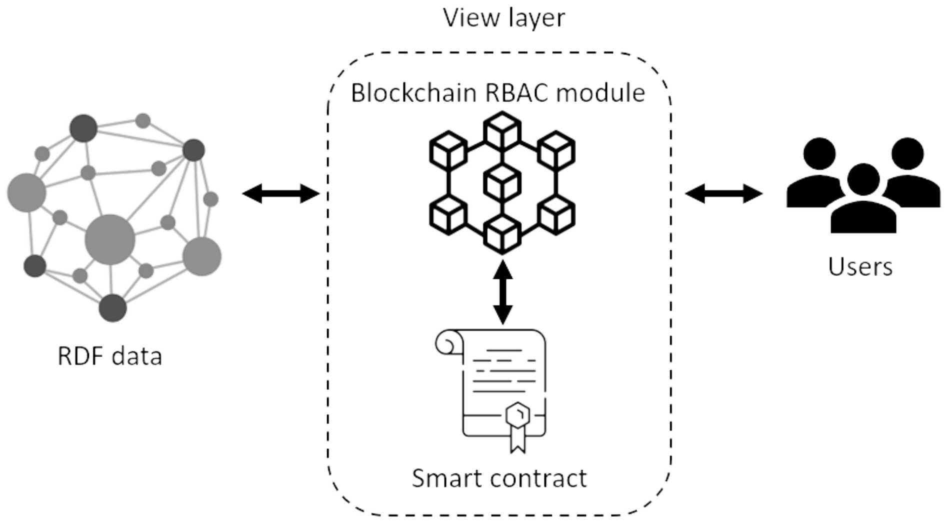
1 Breakdown the big graph into sub-graphs (i.e., views) by applying the edge access control (EAC) enforcement algorithm. Each sub-graph reflects the edge type associated with the user role. Then, based on the query issuer role unionise the sub-graphs related to this role and roles with fewer privileges.

2 Processing the issued query by traversing the graph starting from any node. At each hop, a check is made to ensure whether the following data is allowed to be seen by the user or not. To do this, the path query is compared with the union sub-graph from step 1. If all edges of the path query have a match in the sub-graph, then the query is executed, otherwise, it cannot be processed (i.e., no rewriting).

Hosseinzadeh Kassani et al. (2020) proposed a view-layer architecture that secures the RDF data against unauthorised access by using blockchain technology. Blockchain is one of the distributed ledger technology (DLT) that stores accurate transactions in blocks and make them immutable and indestructible. Furthermore, it supports the use of a smart contract that contains a pre-defined set of rules. This smart contract is distributed and automatically executed once an event occurs. The blockchain model in this architecture follows the role-based access control (RBAC) approach to secure sensitive data. So, there will be a rule in the smart contract that will check the role of the query-issuer to decide whether the user is allowed to access the data or not.

Figure 19 illustrates how the model works. First, the view-layer will receive a query from the user. Note that the user is querying a view of the data, not the original data. Then, the blockchain RBAC module will check the privileges of the user's role and make a decision based on the smart contact response. If the user is authorised, then the query will be executed and the information of the user and the transaction will be added to the block, otherwise, the user is denied.

Figure 19 Securing graph data using views and blockchain



Both Akkiraju et al. (2016) and Hosseinzadeh Kassani et al. (2020) are relying on RBAC which lacks the support of context conditions.

Unlike Akkiraju et al. (2016) and Hosseinzadeh Kassani et al. (2020), Thimma et al. (2013) combine the views with query rewriting. This research is targeting the XML graph. XML is mainly produced to exchange data between web-based applications. Its graph representation consists of labelled nodes, connected via edges in a tree-like structure (Angles and Gutierrez, 2008). The authors adopted a simple AC model, in which its rules come in the form of a tuple with four parts:

$$R = Subject\ (role),\ Object\ (XML\ node),\ Action\ (e.g.,\ read\ and\ write),$$
$$Sign\ (+,\ -\ (i.e.,\ permit\ or\ deny)$$

In most researches, the views are classified on a per-role basis, which causes a high redundancy that affects the storage. This redundancy is caused by having multiple roles with the same views. Therefore, the authors make the classification of sub-views on a per-rule basis, which supports the proposed notion of sub-views sharing. Query rewriting in this model is used in case of having negative rules to eliminate the parts of the query that breach the rules.

The limitation of this solution is that it supports only the tree-like structure graph.

In Table 8, we summarise the pros and cons of all the above-mentioned solutions that enforce AC over graph database.

Table 8 Pros and cons of state-of-the-art access control solutions

Ref.	*Ref. type*	*Category*	*Pros*	*Cons*
Bastani (n.d.)	Commercial solution	Engine level	• A fine-grained relationship makes the traversing faster. • Assigning the users to groups, reduces the redundancy of permissions. • Inheritance permission simplifies the enforcement process.	• Requires traversing the original graph. • Large overhead on the engine. • Using the DENIED relationship is equal to removing the edge.
Bramley (2015)	Commercial solution	Engine level	• A fine-grained relationship makes the traversing faster.	• Requires traversing the original graph. • Large overhead on the engine. • Requires checking multiple classification levels (i.e., more hops).
Ahmadi and Small (2019)	Research project	Engine level	• A fine-grained relationship makes the traversing faster. • Supports the use of context information.	• Requires traversing the original graph. • Large overhead on the engine.
Dhia (2013)	Research project	Engine level	• Fine-grained ACR. • Access decision is made on the fly. • Attributes are involved in the ACR. • The ACR combines both user privacy preferences and the reachability constraints.	• Requires traversing the original graph. • Large overhead on the engine. • Targets only OSN graph.
Morgado et al. (2018)	Research project	Pre-processing (no rewriting)	• Works with various graph models.	• Low performance. • Not supporting the query rewriting technique.
Yalamanchi et al. (2012)	Patent	Pre-processing (rewriting)	• It works with any graph model. • Fine-grained ACR. • Supports context-based AC. • The named groups (i.e., set of constraints per role) facilitate AC enforcement.	• The query is evaluated against the original graph.

Table 8 Pros and cons of state-of-the-art access control solutions (continued)

Ref.	*Ref. type*	*Category*	*Pros*	*Cons*
Akkiraju et al. (2016)	Patent	View-based and pre-processing (no rewriting)	• Creates sub-graphs per secret type. • Not traversing the entire original graph. • The AC management computation can be applied on a copy of the graph that contains only the edge type and ACL information.	• Relies on RBAC which ignores the context environment conditions. • Not supporting the query rewriting technique.
Hosseinzadeh Kassani et al. (2020)	Research project	View-based and pre-processing (no rewriting)	• It works with any graph model. • Not traversing the entire original graph. • Combining the views and the blockchain technologies adds strength to the solution.	• Relies on RBAC which ignores the context environment conditions. • Not supporting the query rewriting technique.
Thimma et al. (2013)	Research project	View-based and pre-processing (rewriting)	• Traversing views, not the entire graph. • Define sub-view based on ACR instead of roles. • Allow sub-views sharing. • Reduce the views redundancy.	• Only supports the XML graph model. • The sub-view notion is easy to implement in a tree-like structure.

4.4 Post-processing

Following this technique means the user query will be computed first, then the retrieved data will be evaluated against the AC policies to remove unauthorised parts if exists, then the final result will be displayed. Although this category was mentioned by Thimma et al. (2013), there was no reference supporting it and we have been unable to locate one during our research.

Though supporting the use of contextual information will strengthen any security model, we can notice in Table 9 that only (Ahmadi and Small, 2019; Yalamanchi et al., 2012) considered contextual information. This highlights a deficiency in this area. Furthermore, most of the recent solutions except (Akkiraju et al., 2016; Hosseinzadeh Kassani et al., 2020; Thimma et al., 2013) rely on traversing the original graph instead of the views. This approach is time-consuming and inefficient. Moreover, the RBAC model is the most commonly used model in current state-of-the-art techniques. However, ABAC is more efficient and is only used in Bramley (2015), Ahmadi and Small (2019), and Dhia (2013).

Table 9 shows a comparison between all solutions we mentioned earlier. The criteria used are listed below:

- *Graph type:* Which type of graph the model supports, whether property graph, RDF graph, or tree-like structure graph.

- *Context:* Context information, e.g., time and the user's role.
- *Metadata:* The data of the data, helps in instantiating and referencing data.
- *Subject in ACR as a node:* The subject specified in the ACR is represented as a node in the graph. Some solutions deal with the subject as the query issuer only without being depicted in the graph.
- *Group permissions:* Having a node denoted as group that takes as input all subjects sharing the same access permissions.
- *Views:* Traversing views instead of the big graph.
- *AC model:* The type of the AC model such as IBAC, RBAC, or ABAC. The AC model is not necessarily explicitly mentioned in the research.

Table 9 Comparison of the state-of-the-art solutions

Ref.	*Graph type*	*Context*	*Metadata*	*Subject in ACR as a node*	*Group permissions*	*Views*	*AC model*
Bastani (n.d.)	Labelled graph	✕	✕	✓	✓	✕	RBAC
Bramley (2015)	Directed property graph	✕	✕	✓	✕	✕	ABAC
Ahmadi and Small (2019)	Directed property graph	✓	✕	✓	✕	✕	ABAC
Dhia (2013)	Directed property graph	✕	✕	✕	✕	✕	ABAC
Morgado et al. (2018)	RDF and others	✕	✓	✓	✓	✕	IBAC
Yalamanchi et al. (2012)	RDF and others	✓	✓	-	✓	✕	RBAC
Akkiraju et al. (2016)	Property graph	✕	✕	✕	✕	✓	RBAC
Hosseinzadeh Kassani et al. (2020)	RDF and others	✕	✕	-	-	✓	RBAC
Thimma et al. (2013)	Tree-like graph	✕	✕	✕	✕	✓	RBAC

5 Discussion and future research directions

Access control (AC) is crucial in graph databases, just as it is in any other type of database. Graph databases often store sensitive information, such as customer data and financial records, making it essential to ensure that only authorised individuals can access and manipulate such data. AC in graph databases ensures that users have the appropriate permissions to read, write, or delete data, and that these permissions are granted based on their role or level of access.

AC for graph databases is challenging for several reasons. Firstly, graph databases store data in a highly interconnected manner, which makes it difficult to define and enforce AC at a granular level. Secondly, graph databases often support complex querying capabilities, which can make it challenging to define rules for AC that are both effective and efficient. Thirdly, graph databases often require collaboration between multiple users or organisations, which can create difficulties in defining and enforcing AC policies across different entities.

Based on the limitations identified in the reviewed solutions that applied AC in graph databases, we recommend several directions for future research.

5.1 Views and pre-processing (with rewriting)

Most of the recent solutions fall under the categories of engine-level and pre-processing level. As a result, they typically require traversing the entire original graph to evaluate the query and retrieve authorised data. This can be a time-consuming process.

Certain solutions integrate both pre-processing and views approaches. While they do allow for the use of views, meaning that there is no requirement to traverse the entire original graph, many of these solutions do not support the query rewriting approach. This means that if a complete answer cannot be provided, there is no option for an approximate answer either.

To enhance the performance and efficiency of the solution, it is suggested to combine the techniques of views and pre-processing (with rewriting) in AC enforcement. This approach leverages the strengths of both techniques and yields better results. Using views will minimize the time needed to traverse the graph (Fan et al., 2016), while rewriting the user pattern query will increase user satisfaction by retrieving an approximate answer rather than no answer.

5.2 Unified AC enforcement mechanism

There are several graph models available, including RDF, property graph, and tree-like structure. However, some solutions are designed to work exclusively with a specific type of graph, making the AC model incompatible with others. For this reason, it is advisable to ensure that support for various graph types is taken into account when constructing a solution. Doing so will increase the overall usability and versatility of the solution.

5.3 Context conditions

It manifests in various aspects such as time, roles, and environment. Overlooking such a fact while designing an AC model results in limited and rigid AC scenarios. For instance, academic advisors in universities have the authority to add or drop courses for a limited period. In such a situation, time becomes a critical factor that must be considered when defining the AC rules.

Considering contextual conditions is essential for security models across all areas. As such, research has shown that implementing context conditions in cloud computing (Alagar and Wan, 2018) and IoT (Schuster et al., 2018) can enhance security measures. For instance, it will allow for various and unlimited AC scenarios. However, recent analysis of solutions in graph databases revealed that only two (Ahmadi and Small, 2019) and Yalamanchi et al. (2012) support context information by fetching data such as user roles during sessions. This highlights a shortage in graph AC models that take context conditions into account.

6 Conclusions

The increase in storage capacity and advancements in technology have resulted in the generation and exchange of a vast amount of data that traditional database management systems are incapable of processing. To address this challenge, a plethora of NoSQL databases have emerged, designed specifically to manage BD more efficiently. These databases come in various forms, ranging from document and key-value to graph databases. In this survey, our focus was on graph databases, as they are the most commonly used type in many social media applications to capture the relationships between users. There are different topics under the umbrella of graph databases, such as security, pattern matching algorithms, and views. These topics are attracting more and more attention from researchers. However, none of the existing investigations have fully addressed them. Therefore, to bridge the gap, we analysed the proposed pattern matching algorithms for answering the queries. Additionally, we discussed views in graph databases. Additionally, we looked at solutions to solve the security problem in graph databases. We have classified these solutions based on the AC enforcement mechanisms they use. Our survey makes a significant contribution by providing a starting point for a better understanding of the state-of-the-art solutions in this area. As described in this survey, recent solutions lack integration between views and preprocessing with rewriting, which decreases user satisfaction. Additionally, most of them do not support the use of contextual information. We therefore proposed valuable insights and clear research directions for the development of future solutions.

References

Ahmadi, H. and Small, D. (2019) *Graph Model Implementation of Attribute-based Access Control Policies*, p.20, arXiv preprint arXiv:1909.09904.

Akkiraju, R.K.T., Mukherjee, D., Nakamura, T., Qiao, M. and Jr, H.R.S. (2016) *Edge Access Control in Querying Facts Stored in Graph Databases* [online] https://patents.google.com/patent/US20160203327A1/en (accessed 5 August 2022).

Alagar, V. and Wan, K. (2018) 'Uniform service description and contextual access control for trustworthy cloud computing', in *2018 International Conference on Cloud Computing, Big Data and Blockchain (ICCBB)*, pp.1–7, ISSN: 7281-1277.

Angles, R. and Gutierrez, C. (2008) 'Survey of graph database models', Vol. 40, No. 1, pp.1:1–1:39, https://doi.org/10.1145/1322432.1322433.

Bastani, K. (n.d.) *Entitlements and Access Control – Neo4j GraphGist* [online] https://neo4j.com/graphgist/entitlements-and-access-control (accessed 15 September 2022).

Bramley, R. (2015) *Attribute-based Access Control with a Graph Database* [online] https://leanjavaengineering.wordpress.com/2015/04/13/attribute-based-access-control-with-a-graph-database/ (accessed 10 September 2022).

Colombo, P. and Ferrari, E. (2018) 'Access control in the era of big data: state of the art and research directions', in *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies, SACMAT '18*, Association for Computing Machinery, pp.185–192, https://doi.org/10.1145/3205977.3205998.

da Trindade, J.M.F., Karanasos, K., Curino, C., Madden, S. and Shun, J. (2020) 'Kaskade: graph views for efficient graph analytics', in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, IEEE, pp.193–204, https://ieeexplore.ieee.org/document/9101351/.

Dhia, I.B. (2013) ‘Large-scale data management in real-world graphs’, *Data Structures and Algorithms [cs.DS]*, Télécom ParisTech.

Fan, W., Wang, X. and Wu, Y. (2016) ‘Answering pattern queries using views’, Vol. 28, No. 2, pp.326–341, http://ieeexplore.ieee.org/document/7101284/.

Gupta, A. and Mumick, I.S. (1999) *Maintenance of Materialized Views: Problems, Techniques, and Applications*, pp.145–157, MIT Press, Cambridge, MA, USA.

Gutiérrez, A., Pucheral, P., Steffen, H. and Thévenin, J-M. (1994) ‘Database graph views: a practical model to manage persistent graphs’, in *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB ‘94*, Morgan Kaufmann Publishers Inc., pp.391–402.

Halevy, A.Y. (2001) ‘Answering queries using views: a survey’, Vol. 10, No. 4, pp.270–294, https://doi.org/10.1007/s007780100054.

Hosseinzadeh Kassani, S., Schneider, K.A. and Deters, R. (2020) ‘Leveraging protection and efficiency of query answering in heterogenous RDF data using blockchain’, in Alhajj, R., Moshirpour, M. and Far, B. (Eds.): *Data Management and Analysis: Case Studies in Education, Healthcare and Beyond, Studies in Big Data*, pp.1–15, Springer International Publishing, https://doi.org/10.1007/978-3-030-32587-9_1

Hu, V.C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R. and Scarfone, K. (2014) *Guide to Attribute Based Access Control (ABAC) Definition and Considerations* [online] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-162.pdf.

Jin, R., Hong, H., Wang, H., Ruan, N. and Xiang, Y. (2010) ‘Computing label-constraint reachability in graph databases’, in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ACM, pp.123–134, https://dl.acm.org/doi/10.1145/1807167.1807183.

Kirrane, S., Mileo, A., Polleres, A. and Decker, S. (2020) *Query Based Access Control for Linked Data*.

Ma, S., Cao, Y., Fan, W., Huai, J. and Wo, T. (2011) ‘Capturing topology in graph pattern matching’, Vol. 5, No. 4, pp.310–321, https://doi.org/10.14778/2095686.2095690.

Mahfoud, H. (2018) ‘Graph pattern matching preserving label-repetition constraints’, in Abdelwahed, E.H., Bellatreche, L., Golfarelli, M., Méry, D. and Ordonez, C. (Eds.): *Model and Data Engineering, Lecture Notes in Computer Science*, Springer International Publishing, pp.268–281.

Milner, R. (1989) *Communication and Concurrency*, Prentice-Hall, Inc., USA.

Morgado, C., Busichia Baioco, G., Basso, T. and Moraes, R. (2018) ‘A security model for access control in graph-oriented databases’, in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp.135–142.

Pang, J. and Zhang, Y. (2015) ‘A new access control scheme for facebook-style social networks’, Vol. 54, pp.44–59, https://www.sciencedirect.com/science/article/pii/S0167404815000632.

Patil, N., Kiran, P., Kavya, N. and Patel, K. (2018) ‘A survey on graph database management techniques for huge unstructured data’, *International Journal of Electrical and Computer Engineering*, Vol. 81, pp.1140–1149.

Schuster, R., Shmatikov, V. and Tromer, E. (2018) ‘Situational access control in the internet of things’, in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS ‘18*, Association for Computing Machinery, pp.1056–1073, https://doi.org/10.1145/3243734.3243817.

Thimma, M., Tsui, T.K. and Luo, B. (2013) ‘HyXAC: a hybrid approach for XML access control’, in *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies, SACMAT ‘13*, Association for Computing Machinery, pp.113–124, https://doi.org/10.1145/2462410.2462424.

Yalamanchi, A., Banerjee, J. and Das, S. (2012) *Access Control for Graph Data* [online] https://patents.google.com/patent/US8250048B2/en?q=access+control+grapg&oq=access+control+for+grapg (accessed 5 September 2022).