NIST Internal Report
NIST IR 8611

m-NGAC

Transcending Traditional Database Security Models

Gopi Katwala

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8611

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Internal Report
NIST IR 8611

m-NGAC

Transcending Traditional Database Security Models

Gopi Katwala
Computer Security Division
Information Technology Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8611

August 2026

U.S. Department of Commerce
Howard Lutnick, Secretary

National Institute of Standards and Technology
Arvind Raman, NIST Director and Under Secretary of Commerce for Standards and Technology

Certain equipment, instruments, software, or materials, commercial or non-commercial, are identified in this paper in order to specify the experimental procedure adequately. Such identification does not imply recommendation or endorsement of any product or service by NIST, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. Many NIST cybersecurity publications, other than the ones noted above, are available at https://csrc.nist.gov/publications.

NIST Technical Series Policies
Copyright, Use, and Licensing Statements
NIST Technical Series Publication Identifier Syntax

Publication History
Approved by the NIST Editorial Review Board on 2026-08-18

How to Cite this NIST Technical Series Publication
Katwala G (2026) m-NGAC: Transcending Traditional Database Security Models. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Internal Report (IR) NIST IR 8611. https://doi.org/10.6028/NIST.IR.8611

Author ORCID iD
Gopi Katwala: 0009-0008-1886-5113

Contact Information
ir8611-comments@nist.gov

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930

Additional Information
Additional information about this publication is available at https://csrc.nist.gov/pubs/ir/8611/final, including related content, potential updates, and document history.

All comments are subject to release under the Freedom of Information Act (FOIA).

Abstract

Securing sensitive data within databases is a paramount concern for modern applications. Traditional database access control mechanisms often fall short in providing the fine-grained control and centralized management required to mitigate evolving security threats. This paper examines the limitations of conventional database access control practices and introduces m-NGAC (embedded Next Generation Access Control), a novel system that embeds the ANSI/INCITS NGAC standard directly within the database. By enforcing access control at the level of individual column data right in the database, m-NGAC offers a significant advancement over traditional methods and ensures data security, regardless of the querying tool, including SQL editors. This paper details the architecture and functionality of m-NGAC and highlights its key advantages in achieving robust and generic solutions with fine-grained access control.

Keywords

ABAC; fine-grained access control; NDAC; NGAC; database security; RDBMS.

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems.

Note to Readers

The patent for this invention has been approved by the United States Patent and Trademark Office (USPTO). Proof-of-concept implementations have been developed for a couple of database vendors.

Table of Contents

List of Figures

1. Introduction

Databases store vast amounts of critical data and are the foundation of many applications. Ensuring the confidentiality and integrity of this data necessitates robust access control mechanisms. Native database access control systems that typically rely on **GRANT** statements provide basic control over tables and columns. However, these systems often lack the granularity to control access at the row and field (i.e., individual column data) levels. Consequently, finer-grained access control is often implemented outside the database at the application level, leading to inconsistencies and vulnerabilities. This report argues that embedded Next Generation Access Control (m-NGAC) offers a superior paradigm by embedding fine-grained access control directly within the database, thereby addressing the shortcomings of traditional approaches.

Traditional database access control models are primarily based on discretionary access control (DAC) and role-based access control (RBAC). While these mechanisms allow administrators to grant or revoke privileges on database objects, they are typically limited to coarse-grained control. In environments where data sensitivity varies across records or attributes, such coarse-grained mechanisms are insufficient. Although database systems may provide finer-grained enforcement mechanisms via schema-integrated constructs such as views and triggers, developers often implement additional access checks in application code, stored procedures, or middleware layers to meet application-specific security requirements. This decentralized approach can introduce inconsistencies across applications accessing the same database and increase the risk of security bypass through direct database queries or misconfigured application logic. Fine-grained access control mechanisms, such as those provided by the Next Generation Access Control (NGAC) model, offer a more flexible and policy-driven approach. By associating users, objects, and operations with attributes and enforcing centrally defined policies, NGAC enables precise control over access at multiple levels of granularity. The m-NGAC approach embeds these policy constructs directly within the database environment, ensuring that access decisions are consistently enforced regardless of how the data is accessed. This integration eliminates reliance on external application logic for security enforcement and provides a unified framework for managing complex access requirements in modern data-intensive systems.

2. Traditional Database Access Control

2.1. Access Control with GRANT/REVOKE

Traditional relational databases enforce security mainly through **Role-Based Access Control (RBAC)** and **Discretionary Access Control (DAC)**. RBAC manages permissions through roles that represent job functions, allowing administrators to assign privileges to roles and then assign users to those roles. This simplifies administration because privileges do not need to be managed for each user individually. DAC, on the other hand, allows the owner of a database object (such as a table) to grant or revoke permissions to other users. These permissions are typically controlled through SQL commands such as GRANT and REVOKE. For example:

```
CREATE ROLE analyst;
GRANT SELECT ON Employees TO analyst;

CREATE USER john IDENTIFIED BY 'password';
GRANT analyst TO john;
```

In addition to role and privilege management, some modern database systems implement **row-level security (RLS)** and **column-level security (CLS)** to provide more fine-grained control. RLS restricts which rows a user can access based on defined policies, while CLS limits access to specific columns that may contain sensitive data. For instance, a role might be allowed to view only selected columns in a table:

```
GRANT SELECT (EmpID, Name, Department)
ON Employees
TO analyst;
```

By combining roles, privilege management, and fine-grained policies, database systems implement comprehensive access control mechanisms that protect sensitive data while still enabling authorized users to perform their tasks. However, despite these capabilities, existing approaches still present several limitations in terms of flexibility, scalability, and policy management in complex data environments. These challenges motivate the need for improved access control mechanisms, which this research aims to address.

2.2. Limitations of Traditional Database Access Control

Current database practices for access control face several significant limitations, as highlighted in the context of the problem m-NGAC aims to solve:

- **Lack of fine-grained control:** In most cases, traditional GRANT statements operate at the table and column levels and fail to provide control over individual rows or specific data fields within a column. This necessitates workarounds for sensitive data, often implemented at the application level.
- **Decentralized control:** When access control is implemented at the application level, each application that accesses the same data needs to build and maintain its own separate

access control logic. This means that there is no centralized point of control, which increases complexity and the potential for inconsistencies and security gaps.

- **Bypass through direct database connections:** Users can often bypass application-level access controls by directly querying the database using direct database connections (like SQL editors, vendor's SQL CLI), thereby exposing unprotected data.
- **Limited granularity in vendor solutions:** While some database vendors have introduced features like row-level security, they often lack field-level access control.
- **Limited proprietary solutions:** Although proprietary solutions to this problem exist and improve with each release, there is no generic solution.

These limitations underscore the need for a more comprehensive and integrated approach to database access control, which m-NGAC aims to provide.

3. m-NGAC: Embedded Fine-Grained Access Control

m-NGAC addresses these limitations by embedding the ANSI/INCITS Next Generation Access Control (NGAC) standard directly within the database environment. Instead of relying solely on external applications to enforce security policies, m-NGAC integrates the NGAC policy framework into the database layer itself. This integration allows the database to natively understand and enforce complex authorization relationships among users, objects, and policies. As a result, access control decisions are no longer dependent on the behavior or correctness of individual client applications; they are consistently applied at the data source where the information resides.

This is achieved by implementing NGAC algorithms (APIs) in SQL, enabling the database to evaluate authorization rules dynamically during query processing. Before a query is executed, these SQL-based APIs are invoked to determine whether the requesting user has the appropriate permissions for the specific data being accessed. Queries submitted through any interface, including command-line clients, applications, reporting tools, and administrative utilities—are intercepted through database-resident enforcement mechanisms such as views, triggers, or policy functions that invoke the SQL APIs during query execution. Because these checks are performed inside the database engine, the enforcement mechanism applies uniformly across all access paths, including web applications, reporting tools, administrative utilities, and direct database connections. This design significantly reduces the risk of bypassing access control through alternate access methods.

Furthermore, embedding NGAC logic within the database enables much finer levels of control than traditional privilege systems. Policies can be applied at granular levels such as rows, individual columns, or even specific fields, allowing organizations to precisely define who can access which pieces of information. This fine-grained enforcement improves data confidentiality and supports complex policy requirements often found in enterprise, regulatory, or multi-tenant environments. By centralizing access control within the database itself, m-NGAC ensures consistent policy enforcement, simplifies security management, and strengthens the overall protection of sensitive data.

3.1. Next Generation Access Control (NGAC)

NGAC is an ANSI/INCITS standard that was developed by NIST. It uses attribute-based, policy-driven access control that models users, operations, and resources as a graph to provide flexible and dynamic security. It offers scalable and fine-grained control over data and systems by decoupling policy enforcement from the application layer.

NGAC is an advanced Attribute-Based Access Control (ABAC) framework that distinguishes itself through its embedded, graph-based architecture. NGAC models access decision data as a directed acyclic graph, representing users, resources, attributes, and policies as interconnected nodes and relationships.

Key aspects of NIST NGAC include:

- **Attribute-based approach:** NGAC allows access-control policies to be defined based on the attributes of users, operations, and resources, thus enabling more granular control than traditional roles.
- **Graph-based model:** Access-control decisions are based on data modeled as a graph, which allows for flexible and dynamic policies.
- **Dynamic and context-aware:** NGAC supports real-time policy changes based on context (e.g., time, location, user behavior), which allows for fine-grained, policy-driven access control.
- **Performance:** The graph-based approach provides linear scalability, enabling efficient policy evaluation and execution as the environment grows.
- **Management flexibility:** The graph-based model supports flexible policy management by simplifying policy definition, modification, and extension.
- **Applications:** In addition to controlling access to databases, NGAC can also enforce policy-based access control across cloud applications and other systems.

Additional benefits of NGAC include:

- **Centralized policy enforcement with distributed enforcement:** NGAC centralizes policy administration while supporting distributed enforcement, enabling consistent policy application across diverse systems and environments.
- **Reduced development efforts:** Developers can avoid implementing access control at the application level, saving time and resources.
- **Enhanced data security:** Fine-grained access control is applied at the database level, thus securing data where it resides.

The NGAC graph shown in Figure 1 represents a sample policy graph corresponding to a Role-Based Access Control (RBAC) policy class. In this graph, users are associated with roles, and permissions are assigned to those roles, thereby reflecting the traditional RBAC authorization model within the NGAC framework. A key advantage of NGAC is that it is not limited to a single policy model. In this sample policy, RBAC is a policy class. Multiple policy classes can coexist within the same NGAC graph, each represented by its own policy subgraph while sharing the same underlying users, attributes, and protected resources. For example, alongside the RBAC policy subgraph, additional subgraphs may implement attribute-based, relationship-based, or organization-specific access control policies. During authorization evaluation, NGAC considers the relevant policy classes collectively, enabling flexible and unified enforcement of diverse access control requirements without duplicating resource attributes or restructuring the underlying data model.

Figure 1 demonstrates an example of a simple policy in the NGAC graph.

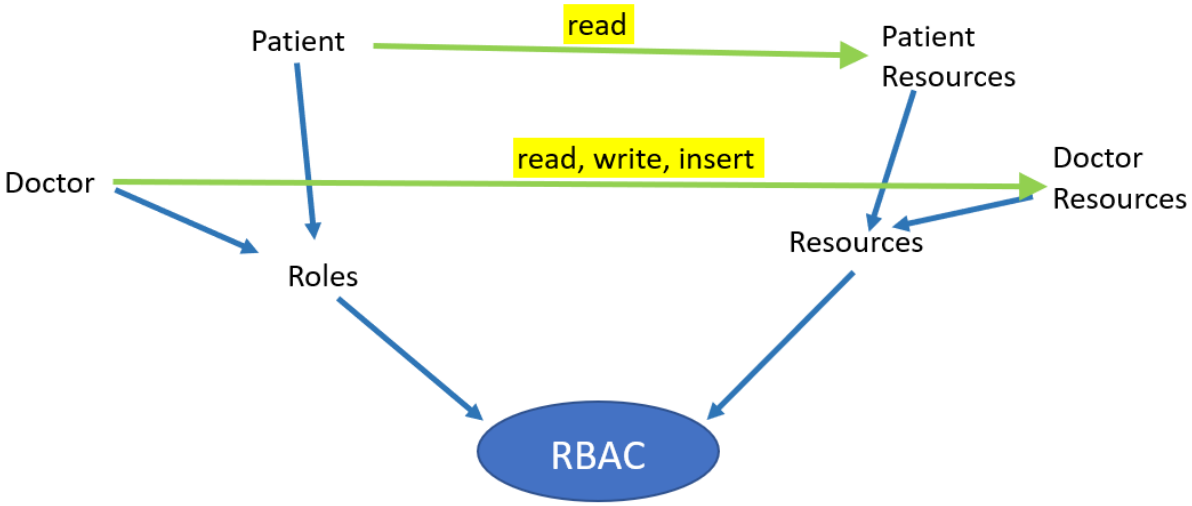

Fig. 1. NGAC policy in directed acyclic graph

3.2. m-NGAC Architecture

The **m-NGAC architecture** comprises three interacting components that collectively enforce fine-grained access control within the database environment. These components work together to intercept database operations, evaluate access policies, and enforce decisions before data is returned to the requesting user or application. Figure 2 illustrates the basic architecture with these components.

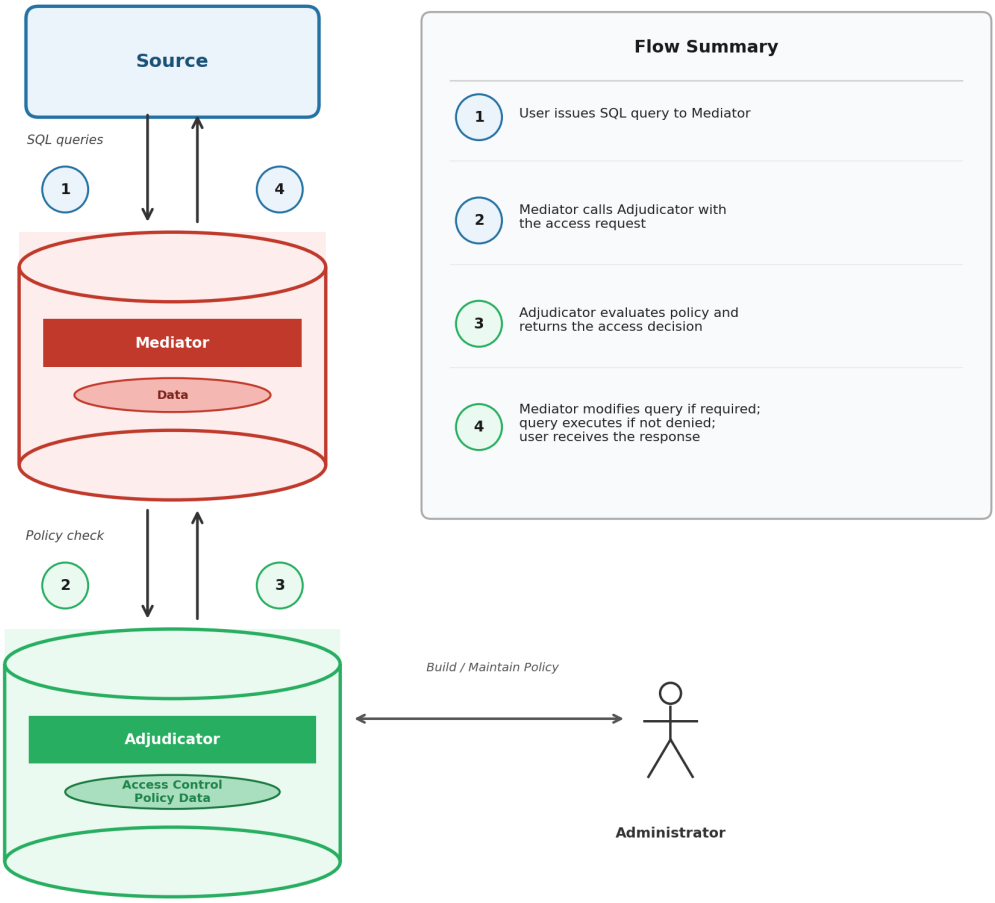

Fig. 2. m-NGAC architecture

Source:

The Source represents the user, application, or external system that issues database queries. These queries may originate from client applications, analytics tools, administrative interfaces, or automated services. From the database's perspective, the Source behaves like any standard client issuing SQL statements such as SELECT, INSERT, UPDATE, or DELETE. However, unlike traditional database architectures where queries are executed directly against the data tables, in the m-NGAC architecture all requests are mediated before reaching the protected data.

Mediator:

The Mediator acts as an intermediary layer that intercepts and processes all data access requests originating from the Source. It is implemented within the resource schema, which is the database schema containing the protected data. Rather than being an integral component of the DBMS engine itself, the Mediator is realized using database-resident objects and logic within the schema, allowing it to leverage native DBMS

functionality without requiring modifications to the underlying database engine. The Mediator is responsible for intercepting incoming queries, identifying the user issuing the request, and determining which database objects and fields are being accessed. By operating within the resource schema, the Mediator ensures that all interactions with the protected data are routed through a controlled enforcement point. This enables the system to enforce fine-grained access policies without requiring modifications to external applications.

Adjudicator:

The Adjudicator is the component responsible for storing access control policies and evaluating them to determine whether a particular operation should be permitted. It resides in a separate **NGAC policy schema**, which may exist within the same database instance as the protected data or be deployed externally, such as on a local service or a cloud-based policy store. The Adjudicator evaluates requests based on the NGAC policy model, which associates users, objects, and operations through attributes and policy relationships. To make efficient authorization decisions, the Adjudicator maintains Access Control Lists (ACLs) derived from the NGAC policy graph and optimized for high-performance access checks. Rather than traversing the policy graph for every request, authorization decisions can be obtained through direct ACL lookups, significantly reducing evaluation overhead. Because ACLs represent a materialized view of the current policy state, they must be regenerated or updated periodically, or whenever policy changes occur, to ensure that authorization decisions accurately reflect the latest user, object, and attribute assignments. Based on the evaluated policies and ACL information, the Adjudicator computes and returns an access decision indicating whether the requested action is allowed or denied.

3.3. Transaction Flow

The interaction among these components during a database transaction proceeds as follows:

1. **Query Submission**
   The Source submits a query to the database. Instead of executing directly against the data tables, the request is first intercepted by the Mediator.
2. **Request Analysis and Policy Inquiry**
   The Mediator analyzes the incoming query to determine the requesting user's identity, the tables involved, and the specific fields being accessed. It extracts this information and forwards it to the Adjudicator for evaluation.
3. **Policy Evaluation**
   The Adjudicator evaluates the request using the NGAC policy stored in the schema using NGAC access functions. These policies determine whether the requesting user has permission to perform the requested operation on the specified data elements. The Adjudicator then returns an access decision to the Mediator.

4. **Query Modification and Result Filtering (SELECT operations)**
   For SELECT queries, the Mediator enforces the access decision by modifying the query. Only the fields permitted by policy are retrieved for the user. For fields where access is denied, the Mediator returns a configurable placeholder value (e.g., **"Protected"**) rather than the actual data. Importantly, the **original WHERE clause of the query remains unchanged**, ensuring that the query's filtering logic remains intact while still enforcing column-level protections.
5. **Controlled Execution of Data Modification Operations**
   For INSERT, UPDATE, and DELETE operations, the Mediator uses database triggers or enforcement mechanisms to verify permissions with the Adjudicator before allowing the operation to proceed. If the Adjudicator confirms that the user has the necessary privileges, the transaction is executed normally. Otherwise, the Mediator denies the request and prevents the modification, thereby preserving the integrity and confidentiality of the protected data.

Figure 3 illustrates a more scalable and robust architectural design in which the Adjudicator and the associated access control policies are maintained centrally within a dedicated database. In this architecture, the centralized Adjudicator serves as the authoritative decision engine, evaluating access requests in accordance with the defined policy rules. Instead of distributing policy logic across multiple databases or applications, all access control policies are stored, managed, and enforced from this central point. This approach simplifies policy administration, ensures uniform enforcement of security rules, and reduces the risk of inconsistencies that might arise when policies are implemented separately across systems.

In this model, multiple resource databases—each storing application or organizational data—can connect to the central Adjudicator to obtain access control decisions. When a user or application submits a query to a resource database, the database consults the Adjudicator before executing the query. The Adjudicator evaluates the request against the centrally stored policies and returns a decision indicating whether the requested access should be permitted or denied. Because all participating databases rely on the same decision authority, organizations can maintain consistent access control policies across diverse data sources, even when the databases are distributed across different systems, environments, or organizational units.

This centralized architecture also improves scalability and maintainability. New databases can be integrated into the system simply by connecting them to the central Adjudicator, without requiring each database to maintain its own independent policy implementation. Similarly, policy updates can be made in one place and immediately apply to all connected databases. Such a design is particularly beneficial in large enterprise environments where multiple databases store related or sensitive information and where consistent enforcement of fine-grained access control policies is critical for security, compliance, and governance. Figure 3 illustrates the extended architecture of securing multiple databases with central policy.

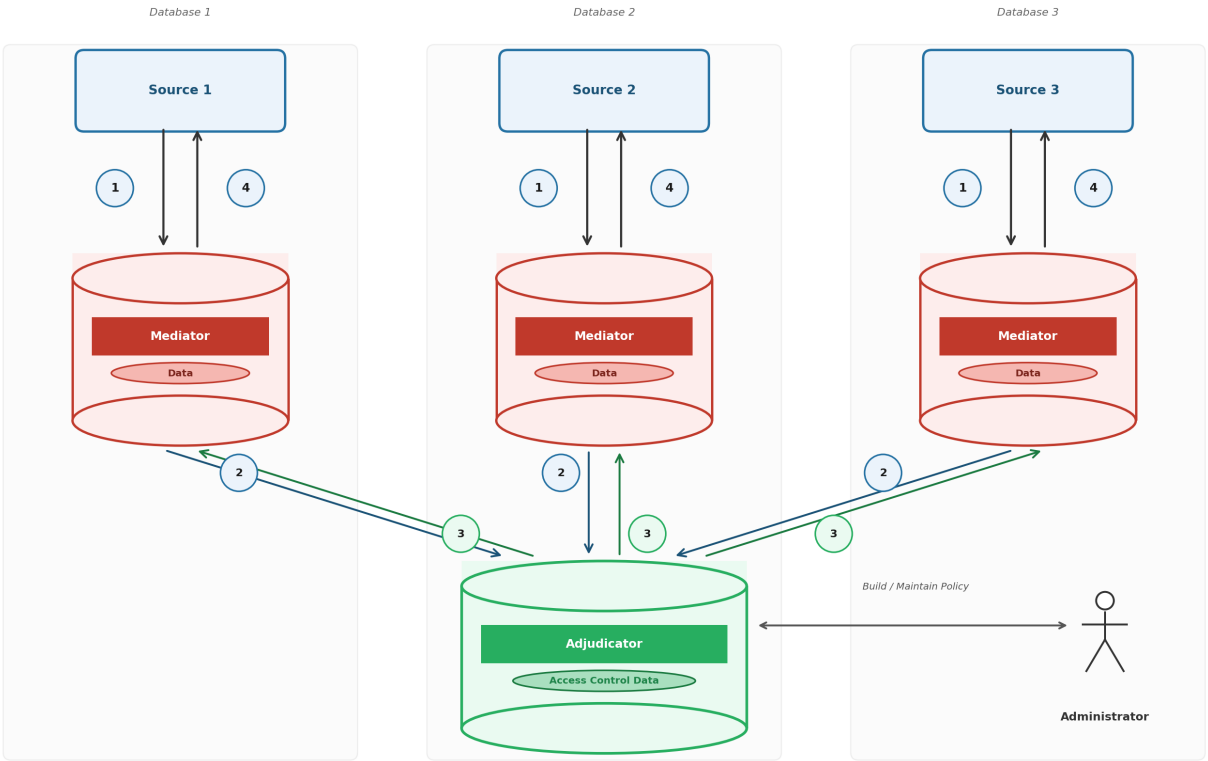

Fig. 3. m-NGAC with multiple databases

3.4. Building Policy

Constructing an access control policy with m-NGAC involves four key steps:

1. Importing the resource schema tables and data into the NGAC schema as object attributes that are organized according to the policy hierarchy. Primary keys are also imported as objects. Only data that requires protection is imported.
2. Creating database users that correspond to application users and granting them basic access using the database's native GRANT statements.
3. Creating users and attributes within the NGAC policy based on the user attribute hierarchy.
4. Building the final policy by defining permissions and denials in the NGAC policy using associations and prohibitions.

3.5. Enforcement Mechanisms

m-NGAC employs different mechanisms for enforcing access control depending on the type of **Data Manipulation Language (DML)** operation being performed. Because read and data-modifying operations have different security implications, the architecture employs specialized

enforcement techniques to ensure confidentiality and data integrity. These mechanisms are implemented directly within the database using structured SQL constructs, enabling consistent application of access control decisions regardless of how the database is accessed.

- **SELECT:**

  Access control for SELECT queries is enforced through database views that function as the **Mediator layer** between users and the protected data tables. Instead of allowing users to query the base tables directly, queries are routed through these mediator views. The views incorporate logic that invokes the Adjudicator to determine whether the requesting user is authorized to view specific data elements. Based on the Adjudicator's access decision, the view dynamically determines which information to return to the user. If the user has permission to access a particular column or field, the actual data value is returned. If access is denied, the system substitutes a configurable masked value (e.g., “Protected”) for the actual data. This approach allows the database to enforce fine-grained visibility controls at the row and column level while preserving the structure and semantics of the original query. Importantly, the filtering conditions specified in the query—such as those in the WHERE clause—remain unchanged, ensuring that the query logic continues to function correctly while sensitive data remains protected.

- **UPDATE, INSERT, DELETE:**

  For data modification operations such as UPDATE, INSERT, and DELETE, m-NGAC employs **database triggers** that act as the Mediator, enforcing access control before the operation is executed. When a user attempts to modify data, the trigger intercepts the request. It communicates with the Adjudicator to verify whether the requested operation is permitted according to the defined NGAC policies. Only if the Adjudicator confirms that the user has the necessary privileges does the database proceed with the operation. Unlike read operations, partial authorization is not permitted for these commands to maintain data consistency and integrity. If the user does not have full authorization for the requested modification, the entire transaction is denied.

  Additionally, when an INSERT or DELETE operation is successfully authorized and executed, m-NGAC automatically updates the corresponding policy relationships within the NGAC policy store. Specifically, the system may create or remove associated NGAC object attributes and policy relationships to reflect the addition or removal of protected data objects. This ensures that the access control model remains synchronized with the database's current state and that newly inserted or deleted records are properly incorporated into the policy structure. By tightly integrating policy updates with data modification operations, m-NGAC maintains a consistent and accurate mapping between protected resources and the policies governing their access.

4. Comparison of m-NGAC with other Access Control mechanisms

4.1. Comparison of General Features of m-NGAC With Traditional Database Access Control

Figure 4 summarizes the key differences between traditional database access control (as generally described in the source) and m-NGAC.

Features	Traditional Databases	m-NGAC
Authentication	Some databases support multiple authentication mechanisms such as user-based authentication or integration with external identity systems.	Supports user-based authentication integrated with policy-driven access control.
Role-Based Access Control (RBAC)	Most databases support RBAC through roles and privilege assignments.	Supports RBAC as part of the NGAC policy framework.
Discretionary Access Control (DAC)	Commonly supported through GRANT and REVOKE privileges controlled by object owners.	Supported within NGAC policies along with other access control models.
Mandatory Access Control (MAC)	Some databases do not support MAC or provide only limited implementations.	Fully supports MAC through attribute-based policy rules.
Row-Level Security (RLS)	Some databases support row-level security, while others lack native support or require custom implementation.	Provides row-level access control through policy-based enforcement.
Column-Level Security (CLS)	Some databases support column-level restrictions, but support varies and may require additional configuration.	Supports column-level control through policy attributes.
Field-Level Security (FLS)	Many databases do not provide true field-level security or require complex workarounds.	Supports fine-grained control down to the individual field level.
Fine-Grained Access Control (FGAC)	Some databases provide limited or complex implementations of fine-grained access control.	Built-in fine-grained access control using attribute-based policies.
Granularity of Control	Access control is typically limited to tables, views, or columns; fine-grained control may be unavailable or complex to implement.	Provides fine-grained control at row, column, and field levels through policy attributes.
Centralized Policy Management	Access policies are often decentralized, with enforcement partly implemented in applications.	Policies are centrally defined and enforced within the database through the NGAC framework.
Policy Enforcement Point	Enforcement may occur partly in the database and partly in application logic.	Enforcement is embedded directly within the database schema and policy engine.
Policy Management Mechanism	Policies are usually managed through privilege statements such as GRANT and REVOKE or through database-specific features.	Policies are managed through NGAC constructs such as objects, attributes, and associations.
Decentralized Access Logic	Access control logic is often implemented across multiple applications, leading to inconsistent enforcement.	Centralized policy logic ensures consistent enforcement regardless of application or access method.
Protection Against Direct SQL Access	Users with direct database access or SQL editors may bypass application-level controls if database policies are incomplete.	Access policies are enforced inside the database, preventing bypass even when using SQL editors or direct queries.
SQL Injection Protection	Protection varies; some databases offer limited or moderate built-in protections and rely on application safeguards.	Stronger protection due to integrated policy enforcement within the database environment.
Complexity of Policy Management	Advanced access control mechanisms may be complex to configure and maintain.	Policy management is structured and manageable through standardized NGAC constructs.
Vendor Feature Limitations	Some databases offer limited or inflexible fine-prained security features that depend on proprietary implementations.	Provides flexible and configurable fine-grained security based on attribute-driven policies.
Portability and Standardization	Security implementations are often proprietary and differ across database systems.	Based on the ANSI/INCITS NGAC standard, enabling portability and extensibility access platfcoms.

Fig. 4. Comparison of databases' native access control with m-NGAC

4.2. Fine-Grained Database Access Control Mechanisms

Over the years, several companies have attempted to implement fine-grained access control in databases. Most of these efforts have relied on third-party tools that intercept SQL queries

outside the database. These tools typically modify SQL statements before execution to enforce policy-based access control.

4.2.1. Database-Integrated Fine-Grained Security

Some database systems provide fine-grained access control mechanisms tightly integrated with the database engine. These mechanisms enforce security by dynamically modifying user queries before execution. When a user issues a SQL statement against a table, view, or synonym, the database automatically appends security predicates (additional filtering conditions) to the query. As a result, access control policies are enforced directly within the database, allowing administrators to restrict data visibility and modification at the row and column levels based on factors such as user identity, application context, or session attributes.

This approach is typically implemented by associating security policies with database objects through stored procedures or database functions that return predicate conditions. When a query is executed, the database transparently attaches the returned predicate to the SQL statement, ensuring that only authorized data is accessible. Because enforcement occurs within the database engine, it provides a centralized security mechanism that reduces reliance on application-level access control logic and lowers the risk of policy bypass via direct database connections. These systems often support both row-level and field-level security and may include additional capabilities such as dynamic data masking or redaction. However, such implementations are often **vendor-specific and proprietary**, limiting their portability across different database platforms.

4.2.2. NDAC

The closest prior art to a universal solution is **N**ext Generation **D**atabase **A**ccess **C**ontrol **(NDAC)**.

NDAC's method begins by automatically generating composite objects in the form of object attributes from a database schema. It uses NGAC as its authorization engine to manage policies Administration Point (PAP) and compute authorization responses via the Policy Decision Point (PDP). While NDAC aims to be a universal solution, excerpts indicate that it operates between the application and the database, parsing SQL queries to modify them based on the defined policies.

However, a newer approach called m-NGAC (modular NGAC) introduces a breakthrough in fine-grained, policy-enforced, database-independent access control. A significant difference highlighted when comparing it to m-NGAC is that NDAC does not generate access control lists (ACLs) and can be bypassed by direct database connections.

Furthermore, NDAC enforces access control by modifying the WHERE clause of the original SQL query. This approach can be fragile because it alters application-generated query logic and may introduce unintended interactions with existing query conditions. The following section discusses the limitations and potential drawbacks of modifying the original WHERE clause for policy enforcement.

4.2.2.1. Issues with WHERE Clause Augmentation

NDAC enforces access control by modifying the WHERE clause of application-generated SQL queries. While this approach enables policy enforcement without requiring changes to the underlying database, it introduces several technical challenges and limitations.

- **SQL Parsing Complexity and Fragility.**
  SQL is not a single standardized language in practice; major database platforms such as MySQL, PostgreSQL, Oracle, and SQL Server each support dialect-specific syntax and features. A policy enforcement mechanism that parses and rewrites SQL statements must correctly handle a wide range of constructs, including nested subqueries, common table expressions (CTEs), window functions, and vendor-specific extensions. Incomplete or incorrect parsing can result in malformed queries, enforcement failures, or unintended security gaps.
- **Operator Precedence and Semantic Correctness.**
  Modifying an existing WHERE clause requires careful preservation of the query's original logical semantics. For example, appending an access-control predicate to a query containing complex combinations of AND and OR operators can change the evaluation order if the original conditions are not properly parenthesized. Ensuring semantic correctness across all query forms increases implementation complexity and introduces opportunities for errors that may affect either security or application functionality.
- **Impact on Application Behavior.**
  Applications are often designed with assumptions about the structure and size of query result sets. Injecting additional filtering conditions can alter row counts, affect pagination behavior, and change the results returned to application logic. Such changes may introduce unexpected behavior, requiring developers to understand and account for policy-driven modifications that occur outside the application's control.
- **Potential Effects on Query Optimization.**
  Database query optimizers generate execution plans based on the predicates present in a query. Additional access-control predicates may alter these execution plans, potentially affecting index selection and query performance. In some cases, the injected conditions can prevent the optimizer from using otherwise effective indexes, resulting in less efficient execution strategies such as full table scans. As policy complexity grows, the resulting queries may become more difficult for the optimizer to process efficiently, leading to increased execution costs and reduced scalability.
- **Auditability and** Traceability Challenges.
  When SQL statements are modified between the application and the database, the query executed by the database differs from the query originally issued by the application. This separation can complicate auditing, troubleshooting, and forensic analysis because application logs may not accurately reflect the statements ultimately executed by the database engine.

In summary, WHERE clause modification enforces security by rewriting application-generated queries rather than by controlling the data exposed by the database itself. While functional, this approach introduces complexity in query processing, increases the risk of semantic errors, and may affect performance, maintainability, and auditability.

4.2.2.2. Commonalities Between NDAC and m-NGAC

- Both are based on ANSI/INCITS NGAC, a standardized ABAC model.
- Both approaches use automatically generated object attributes to represent schema and row data. These attributes are then used to express access control rules and enforce policies accordingly.

4.2.2.3. Key Differences

- Implementation: m-NGAC is embedded directly in the database schema, while NDAC operates between the application and the database.
- SQL Parsing: NDAC parses SQL to modify it based on policies, which can become complex due to the diversity in SQL syntax. In contrast, m-NGAC uses database views to transparently enforce policies without altering the original WHERE clause.
- ACLs: NDAC does not generate ACLs. m-NGAC builds ACLs to facilitate policy decisions and enforces access using SQL functions, ACLs, views, and triggers.
- Bypass Risk: Direct database connections (e.g., MySQL Workbench, SQL Developer) can bypass NDAC policies. m-NGAC, however, enforces policies within the database, making it resistant even to system-level access, including that of root users.
- Query Modification: NDAC (and similar systems) modifies the WHERE clause of the SQL query to enforce policies. m-NGAC preserves the WHERE clause, modifies only the selected columns, and returns ‘PROTECTED’ in place of restricted data.

Figure 5 shows a side-by-side comparison between the two.

Feature	NDAC	m-NGAC
Access Control Model	NGAC (ABAC-based)	NGAC (ABAC-based)
Location of Enforcement	Between client and database	Within the database schema
SQL Query Handling	Parses and rewrites SQL queries	Uses views and triggers, no parsing needed
WHERE Clause Modification	Indirectly modified to enforce policy	Unchanged, preserved as is
Restricted Column Access handling	Modifies query, retrieves data and masks the values	Doesn't retrieve protected data, returns 'PROTECTED' for restricted columns
Use of ACLs	Does not produce ACLs	Builds and uses ACLs
Protection Against direct database connections	No, SQL editors can bypass it	Yes, enforced inside the DB
Portability Across Databases	No, custom per database	Yes, database-independent (with minimal customization effort)
Vendor Lock-In	Not applicable	No, open and flexible
Protection from Root/System Users	No, can be bypassed	Yes, via policy

Fig. 5. Comparison between NDAC and m-NGAC

Overall, **m-NGAC** is the **first generic, non-proprietary solution** that enforces fine-grained policies within the database itself, is **compatible with any database engine** (with minimal adjustments), and can protect **data, even from privileged users and direct database access**.

5. Conclusion

m-NGAC represents a significant evolution in database access control by embedding the NGAC standard's power and flexibility directly within the database environment. This approach overcomes the inherent limitations of traditional access control methods by offering **fine-grained control down to the field level, a centralized policy management point**, and **robust protection against circumvention via SQL editors**. By not requiring SQL parsing or modification of the WHERE clause, m-NGAC offers a unique and efficient solution for securing sensitive data across various SQL database systems. As organizations grapple with increasingly complex data security requirements, m-NGAC provides a compelling, generic framework for achieving comprehensive, reliable database access control.

Appendix A. Bibliography

- Ferraiolo D, Chandramouli R, Kuhn DR (2016) Next Generation Access Control (NGAC): Policy Machine-Based Authorization Framework. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Internal Report (IR) NIST IR 7987. https://doi.org/10.6028/NIST.IR.7987
- ISO/IEC 10181-3:1996. Information Technology – Open Systems Interconnection – Security Frameworks for Open Systems: Access Control Framework. International Organization for Standardization
- ANSI/INCITS 499-2018. Information technology - Next Generation Access Control - Functional Architecture (NGAC). International Committee for Information Technology Standards (INCITS)
- Oracle Corporation (2022) Oracle Database Security Guide, 19c. Available at https://docs.oracle.com/en/database/oracle/oracle-database/19/dbseg/index.html
- Microsoft Corporation (2023) SQL Server Security Documentation. Available at https://learn.microsoft.com/en-us/sql/relational-databases/security
- MySQL AB/Oracle (2023) MySQL 8.0 Reference Manual: Access Control and Account Management. Available at https://dev.mysql.com/doc/refman/8.0/en/access-control.html
- PostgreSQL Global Development Group (2023) PostgreSQL Documentation: Row-Level Security. Available at https://www.postgresql.org/docs/current/ddl-rowsecurity.html
- Ferraiolo D, Gavrila S, Katwala G, Roberts J (2017). Next generation access control system and process for controlling database access (U.S. Patent Application No. US20170024572A1). United States Patent and Trademark Office. Available at https://patentimages.storage.googleapis.com/f8/36/75/4a9f62bec49834/US20170024572A1.pdf