A FLEXIBLE FRAMEWORK FOR APPLYING DATA ACCESS AUTHORIZATION BUSINESS RULES

Leonardo Guerreiro Azevedo, Sergio Puntar, Raphael Thiago, Fernanda Baião and Claudia Cappelli
NP2Tec – Research and Practice Group in Information Technology
Federal University of the State of Rio de Janeiro (UNIRIO), Av Pasteur 458, Rio de Janeiro, Brazil

Keywords: Database Security, Data Access Authorization Business Rules, VPD, TPC-H Benchmark.

Abstract: This work proposes a flexible framework for managing and implementing data access authorization business rules on top of relational DBMSs, in an independent way for the applications accessing a database. The framework adopts the RBAC policy definition approach, and was implemented on Oracle DBMS. Therefore, data access security is managed by the data server layer in a centralized manner, rather than in each application that accesses data, and is enforced by the database server. Experimental tests were executed using the TPCH Benchmark workload, and the results indicate the effectiveness of our proposal.

1 INTRODUCTION

Business rules are policies which define and describe business logic. According to the taxonomy defined by BRG (2009), one type of business rules is authorization action assertion rule, or *authorization rule* for short, which restricts who is allowed to perform a certain action in an organization.

Authorization rules are gaining major importance in organizations, since they are central in most data access security initiatives. The definition and control of who may have access to each piece of information during the execution of a business process is vital to prevent frauds and to conduct controlling initiatives.

An authorization rule is composed by: (i) Data on which the rule is applied; (ii) The user (or profile or user group) whose access to the data is controlled by the rule; and (iii) the rule description.

Applications have tended to define their own security policies and enforce them at the client layer. However, this is not a properly solution when it comes to very frequent scenarios in which large companies are required to adhere to a whole new set of authorization rules defined by controlling initiatives derived by SOX (SOX, 2009) or other security demand. Therefore, there is a need of a flexible framework for the implementation of authorization rules on top of existing databases, which minimizes the changes required on the legacy applications that manipulate existing data.

This work proposes a flexible framework for managing and implementing authorization rules on top of relational DBMSs, in an independent way for the applications accessing the database. The framework adopts the RBAC (Role-based access control) policy definition approach, and was implemented on top of Oracle 10g DBMS using its VPD (Virtual Private Database) features. Experimental tests were performed on top of the TPCH Benchmark (TPC Council, 2008).

This work is divided as follows. Section 2 presents database security concepts. Section 3 presents the proposed framework. Section 4 details its implementation, while section 5 presents the experimental results. Section 6 and 7 presents related work, and conclusions and future works, respectively.

2 DATA SECURITY CONCEPTS

Data access control is one of the major components of database security, and has been the focus of several database security approaches provided by most DBMS vendors. Typically, those security mechanisms consist of defining users (a database-level identity to anyone connected to the database), profiles (group of users with similar privileges to database structures and data) and database sessions (which allows the DBMS to identify and control the sequence of commands performed by a single user

Guerreiro Azevedo L., Puntar S., Thiago R., Baião F. and Cappelli C. (2010).
A FLEXIBLE FRAMEWORK FOR APPLYING DATA ACCESS AUTHORIZATION BUSINESS RULES.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Databases and Information Systems Integration*, pages 275-280
DOI: 10.5220/0002909602750280

in each connection).

Murthy and Sedlar (2007) argue that those access control mechanisms, although simple and frequently used, present significant drawbacks. Firstly, the same underlying data may have to be accessed via many different applications and it is hard to reconcile different application security mechanisms. Secondly, many data warehousing and data mining tools require direct access to the database (via SQL) and it is impossible to enforce application level security during direct SQL access. Thirdly, the lack of a common security framework makes it very hard to administer the policies and increases the risk of security holes. Finally, there is a significant performance impact by always evaluating data access authorization rules at the application layer.

The mechanisms for access control can be classified into: DAC (Discricionary Access Control), MAC (Mandatory Access Control), both proposed by DoD (1983), and RBAC (Role-Based Access Control) (Ferraiolo and Khun, 1992).

DAC policies are based on the identity of the requestor and on access rules stating what requestors are (or are not) allowed. They can be implemented by an access matrix model which regulates the privileges that a subject can have on an object. An object can be a table, a view, a procedure or any other database object. Yang (2009) presents that DAC policies do not enforce any control on the flow of information, thus making it possible for processes to leak information to users not allowed to read it. SQL Server, MySQL, Oracle Database, DB2 and Sybase DBMSs support the implementation of DAC policies through access matrix models.

MAC policies, also known as label security, are based on mandated regulations determined by a central authority. The most common form of MAC is the multilevel security policy using classification of subjects and objects in the system. In multilevel mandatory policies, an access class is assigned to each object and subject. MAC policy controls flow of information, thus preventing leakages to unauthorized subjects. However, it does not address actions that users are allowed to execute over data (Ferraiolo and Khun, 1992).Most database vendors offer label security supporting features, such as Oracle, Sybase and Microsoft SQL Server.

MAC policies assume that each label is applied to the whole table row, thus preventing the definition of authorization rules on a subset of the row attributes. Besides, since a label is attached to each table row, if many policies are applied, management and maintenance costs may be high.

An RBAC access rule states which actions and subjects are allowed to users in a given role. RBAC is a common paradigm to ensure that users have sufficient rights to perform various operations (Fischer *et al.*, 2009), and is typically used by organizations to specify and enforce specific security policies in a way that maps naturally to the organization structure. Ferraiolo *et al.* (2001) propose a pattern for RBAC in order to consolidate different RBAC reference models, commercial products and research prototypes. Both SQL Server and Oracle databases support concepts of roles.

RBAC policies are more flexible than MAC ones, and therefore are the focus of our proposed framework. Nevertheless, defining and enforcing RBAC policies is not simple in real scenarios, since it requires a lot of effort and knowledge from the user responsible to create and assign rules and roles, usually a Database Administrator (DBA). Therefore, there is a need of a flexible and easy to use mechanism to aid database administrators in defining and managing RBAC policies to implement database security.

3 PROPOSED FRAMEWORK

This section presents a framework for managing and controlling authorization rules of applications on top of corporative databases. The framework is composed of two modules: (i) Authorization rule management (ARM) and (ii) Authorization rule execution (ARE). The authorization rule management module is responsible for creating, changing, viewing, composing, testing, and simulating rules. All authorization rules defined in this module are stored in a business rule database.

The authorization rule execution module follows the RBAC policy described in section 2, and is responsible for assuring that the previously defined rules are enforced during the execution of every application accessing the corporative database. In other words, the execution module controls all applications so that all data retrieved by them will surely adhere to the authorization rules stored in the business rule database.

To prevent changes in the source code of legacy applications, the ARE module should be implemented in an independent way for the applications accessing the DBMS.

Defining and executing an authorization rule are tasks performed in two distinct moments and, ideally, by two distinct teams. Typically, the ARM functions will be handled by business users, while ARE functions should be monitored by IT users.

Firstly, business users of the authorization rule definition team uses the ARM module to define and create the set of rules that reflects RBAC policies over corporative data. Through the ARM module, they may access the conceptual schema of the organization, which represents a business (high-level) definition of all the concepts and relationships stored in the database. To define a new authorization rule for a RBAC policy, the ARM user should define a profile (for example, *LocalManager*), choose the concept to be controlled (for example, *Order*), each concept attribute that will be of restricted access (for example, *totalValue*), along with its range interval (for example, *totalValue < 1,000.00*), and the operations that should be controlled (select, insert, update, delete) (Figure 1). The ARM module should therefore provide a graphical interface with usability concerns so as to enable business users to self-manage their authorization rules and RBAC policies.

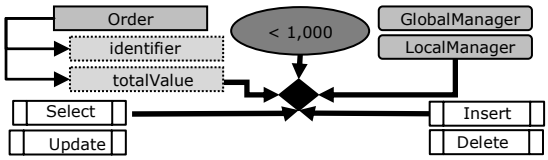

Figure 1: Architecture of ARM component.

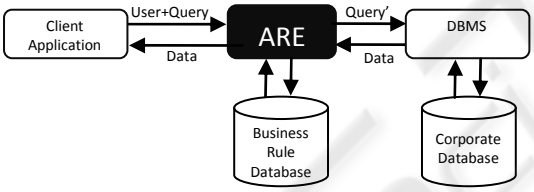

Figure 2: Architecture of ARE component.

In execution time (Figure 2), ARE module captures every connection from a client application to the corporative database, transforms it (say, "Query") into a modified one (say, "Query´") according to the authorization rules present in the business rule database for the connected user. The ARE module then sends Query´ to the DBMS and forwards its returned data to the client application.

3.1 Metamodel for Authorization Rules

The authorization rules are stored in the business rule database as follows (Figure 3).

Users are grouped into profiles, where each User may be assigned to several Profiles, and each Profile groups several Users. The self-relationship of Profile entity allows the representation of a Profile hierarchy. For example, *Sales Manager of America and Asia* and *Sales Manager of Europe* profiles may be specializations of the *Sales Manager* profile. This denotes that authorization rules assigned to the more general profile (*SalesManager*) should also be enforced to all its descendents. Additionally, descendent profiles may determine specific values for range predicates. The *Sales Manager* profile may be restricted to access *Orders* based on its *originatedFromContinent* attribute; the *Sales Manager of America and Asia* specifies the specific (list of) values of the controlled attribute that the profile is allowed to access (*"(America, Asia)"*), thus enabling the RBAC policy to enforce access only to *Orders* where "*originatedFromContinent* in *"(America, Asia)"*).

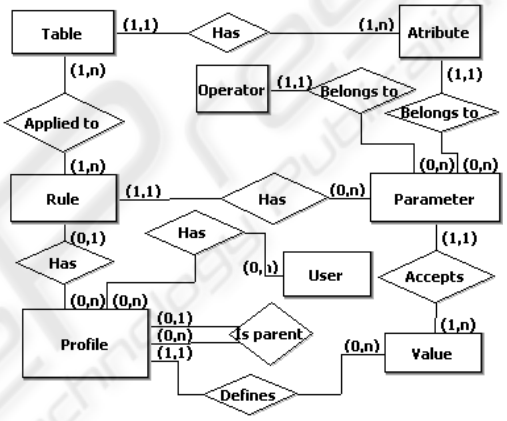

Figure 3: Framework model.

Rule statements are stored as predicates (in Struct Query Language (SQL) format) in table "`Rule`". The predicate is to restrict user access returning a subset of a table. So, there is a relationship "`Rule × Table`". The same query can be used to define an authorization rule over different tables, and each `Table` may have more than one rule statement.

A `Parameter` denotes an expression of the form "`Attribute operator value`" in the corresponding entities (`Attribute`, `Operator` and `Value`). `Attribute` is a database attribute with restricted access. `Operator` is a binary operator (including "=", "<>", ">", ">=", "<", "<=", AND, "OR" and "IN"). `Value` is specific values (or values lists) within the attribute domain. It is used to delimit value ranges of predicate Parameters. It is represented by the entity `Parameter` which relates the entities `Attribute × Operator × Value`. The relationship `Table × Attribute` indicates to which `Table` the

Attribute belongs. This relationship may appear controversial, and it denotes a high-level representation of the database schema (tables and attributes) to which the authorization rules are applied and that are referred as predicates' parameters in a rule definition.

The same user can have more than one Profile related to the same Table. For example, the user has the profile *Manager* and *Customer* which are related to *Orders*. So, when executing the authorization rule, more than one rule (SQL for restriction) will be returned, and they must be composed in order to restrict access. This composition can be executed using OR or AND operators. Using OR operator the composition is for permissive rules, since the user will have access to the union of subsets of restrict table. For example, the subset corresponding to *Manager* profile and the subset corresponding to *Customer* profile. However, using AND operator, it is a restrictive composition, since the user will have access to the intersection of the subset tables corresponding to each rule. The proposed framework can be used to handle permissive or restrictive rules, and it is upon the organization to decide which composition operator to use. Our tests applied the OR operator.

Therefore, rules created using ARM module must be stored. During authorization rule execution, ARE reads rules which are related to user's profiles and executes the restrictions.

4 IMPLEMENTATION ARCHITECTURE AND DETAILS

In this work, we implemented the ARE proposal on top of Virtual Private Database (VPD), which also follows the RBAC approach.

4.1 Virtual Private Database (VPD)

VPD enables define authorization rules for tables, views and synonyms. When a user tries to access (directly or indirectly) a table, view, or synonym protected by a VPD policy (implemented in a PL/SQL function), the server dynamically changes the user command. This creates a predicate (condition for the WHERE clause) returned by the function implementing the security policy. Policies can be set for SELECT, INSERT, UPDATE, INDEX, and DELETE commands.

For example, a user makes the following query in a table *emp*: "SELECT name, ssn FROM emp;". Considering that table *emp* has a protecting policy, where each user can view only their own information, then authorization VPD function returns the following predicate *ssn = 'my ssn'* and the command is rewritten as "SELECT name, ssn FROM emp WHERE ssn='*my ssn*';".

VPD is not simple to use in real scenarios. It requires a lot of effort and knowledge from the user responsible to create rules, usually a Database Administrator (DBA). For example, for each rule, a function should be implemented to return the rule predicate. SQL, PL/SQL and the database structure must be very well understood.

4.2 Generic Authorization Function

This work proposes a generic function for authorization rules. This function must be applied to all secured tables, and accordingly to information stored in tables of the framework model, it will return the predicate for table access restriction.

When executing the authorization rule, the framework receives the *user* and the *query*, and executes the following steps considering data stored in tables of the framework (Figure 3):

- From the *user* in table User, it discovers the Profiles the user has.
- From the *query* the user wants to execute, it discovers which Tables are considered by the query (tables in "from" clauses).
- From Profiles and Tables, it identifies which rules must be applied to each table (stored in Profile SQL).
- From the Profile and Rule, it is identified which predicates (Attribute × Operator × Value) must be used by the rules to filter data.
- Finally, a predicate to restrict the queried table is returned and the query submitted by the user (say Query) is transformed to a restricted query (say Query') which is used to return only the data the user is allowed to access.

5 EXPERIMENTAL RESULTS

We evaluated our implementation using the TPC-H Benchmark scenario (TPC Council, 2008). The TPC-H is a benchmark specification of broad industry-wide relevance that simulates a scenario of a representative decision support application.

This scenario has a realistic context. It consists of a database schema description and a suite of business-oriented, ad-hoc OLAP queries. The TPC-H Benchmark was chosen to show the effectiveness and generality of our proposed framework, since it examines large volumes of data, executes queries with a high degree of complexity and answers critical business questions. TPC-H model includes Suppliers and Customers, and Nation and Region were they are located. Suppliers supply items (table Part), and table Partsupp stores relationship between Part and Supplier. Customers make orders (table Order) of line items (table LineItem) from suppliers.

In the experimental tests, some TPC-H Benchmark queries were chosen to cover several user access patterns on the database, and used to evaluate the following operations: simple queries, queries with subqueries, queries with group by and queries with having. Besides, we evaluated the framework for the insert, update, and delete operations, and for trigger. Some tests descriptions are presented to illustrate the proposal.

Q1 – Local supplier volume. Lists the revenue volume done through local suppliers.

Q2 – Pricing summary report. Reports the amount of business billed, shipped, and returned.

For the tests, we have added an attribute (named N_HEMISPHERE) to the NATION table.

T1. Authorization rule designed to Q2 query: Restricts the access of Customers to the line itens from Mozambique, India or Russia nations.

For this example, we created a hierarchy of profiles *Customer* > *Customer of Mozambique, India or Russia*. The predicate of Figure 4 was created as a `Rule` related to Customer profile and `Table LINEITEM`. The "?user" is replaced by the user login. A `Parameter` was created for `Attribute` "n_name" of `Table` "Nation" using `Operator` "IN". Related to this `Parameter` and profile *Customer of Mozambique, India or Russia*, `Values` *Mozambique, India, Russia* were created.

```
l_orderkey in (
  select o_orderkey
  from orders, customer, nation
  where lower(c_name) = lower(?user))
    and c_custkey = o_custkey
    and n_nationkey = c_nationkey
    and
```

Figure 4: Predicate for profile Customer.

After profile configuration, query Q2 was executed by a user without restrictions, and a user with *Customer of Mozambique, India or Russia* profile. For the first user, all data was returned. However, for the second user, the generic function generated the right predicate, restricting accessed data to tuples from *Mozambique, India* and *Russia*. For the insert operation test, we tried to insert data in a restricted table in a range that do not belong to the user profile. In this case, the predicate is computed and used to evaluate if the user still have access to data after the insertion. If it can access data, then the insertion is performed, otherwise insertion is not executed. The same reasoning was used in update and delete tests. So, data that do not belong to subset corresponding to user profile was not updated or deleted as well.

The trigger test executed on top of `NATION` table. A profile was created restricting user to access suppliers only of his nation. Table Nation did not have restrictions. A user from Brazil trying to update the field comment of nation Germany succeeded; however, suppliers of Germany were not updated. When the same user updated the comment of Brazil, all suppliers of Brazil were updated.

6 RELATED DATABASE SECURITY APPROACHES

Fischer *et al* (2009) argue that traditional RBAC approaches does not easily express parameterized security requirements. They proposed a generalized RBAC model (called Object-sensitive RBAC – ORBAC) to solve this expressiveness limitations of RBAC by allowing roles to be parameterized by properties of the business objects being manipulated. ORBAC is applicable in scenarios where the accessing application is programmed in an OO language, which is not the case in several legacy systems; our approach may be applied independent of the application programming language.

Vimercati *et al*. (2008) present that significant amount of research has recently focused on the problem of processing distributed queries under access restrictions, based on the concept of access pattern and chase and data dependencies. Calì and Martinenghi (2008) define the access pattern as an approach to mark each attribute of a relation/view as "i" (input) or "o" (output). The relation/view can only be accessed if constants can bind the input attributes. Data from output attributes are returned. This approach restricts access to data, but handles only attributes of relations, and not range of attribute values nor relationships between relations. In our proposal, both issues are handled. Besides, supply input attributes to execute the access policy is not

feasible because of the many ways data can be accessed, and different operations executed.

Vimercati *et al*. (2008) present that the chase process exploits a specific data structure, called tableau, to represent a query or a relation. It is usually adopted to study and identify functional dependencies within a relation schema, to check if a decomposition is lossy or lossless, to evaluate if the result of a query q_i is contained in the result of another query q_j (or vice versa) without explicitly computing the queries. When the verification returns false, the user receive no data, and the application must be change to comply with the rule. In RBAC approach, query is rewritten in order to return only data user has access. No error is returned, and the user receives only data he has access to. They propose a graph model approach to model authorization rule, database schema and queries, using authorization compositions and coloring the graph. This approach has the following drawbacks: it handles authorization for read operations (queries) and not write operations; it does not handle authorization on specific tuples of tables; it does not handle cyclic schemas, so it requires to remove all cycle from existing schemas which can be very expensive, and not feasible in practice. Medium and large companies usually cannot change their database to comply with this requirement.

7 CONCLUSIONS

Data access security is an important issue for enterprises. Authorization rules are traditionally implemented into IT applications, which define their own security policies and enforce them at the client layer. However, if a rule change, all applications that implemented the rule must be updated. So, it is a very complex problem in a scenario with lot o legacy systems.

In order to improve this environment, there are solutions for authorization control on top of databases, such as Discretionary access control (DAC), Mandatory access control (MAC), and Role-based access control (RBAC). However, such implementations are difficult to manage, thus requiring skilled professionals.

In this work, we presented a flexible and easy to use framework for managing and controlling authorization rules of applications on top of corporative databases. The framework has two components (i) Authorization rule management (ARM) and (ii) Authorization rule execution (ARE). ARE component was implemented using Virtual Private Database (VPD) in Oracle, and evaluated using TPC-H Benchmark queries and data. The results showed the effectiveness of the proposal. Further experiments are being conducted, beyond the scope of this work, addressing the performance impact of our proposal.

As future work, we point the implementation of ARM and the evaluation of the ARE in a real scenario. For the first, we are evaluating if existing Business Rule Management System comply with ARM requirements. For the second, we are executing experimental tests in real scenarios.

REFERENCES

BRG, 2009. The Business Rules Group. http://www.businessrulesgroup.org/home-brg.shtml.

Calì, A., Martinenghi, D. 2008. Querying data under access limitations. In ICDE 2008, Cancun.

DoD, 1983, Trusted Computer Security Evaluation Criteria. Department of Defense, DoD 5200.28-STD.

Ferraiolo, D., Khun, D. 1992, Role-Based Access Control. In: 15th Nat´l Computer Security Conf, pp. 554-563.

Ferraiolo, D.F., Sandhu, R., *et al.*, 2001, Proposed NIST standard for role-based access control. ACM Transactions on Information and System Security 4 (3), pp. 224-274.

Fischer et al., 2009, Fine-Grained Access Control with Object-Sensitive Roles, In: Drossopoulou (Ed.): ECOOP 2009, LNCS 5653, pp. 173–194

Murthy, R., Sedlar, E., 2007. Flexible and efficient access control in oracle. In *ACM SIGMOD 2007*, pp. 973-980, Beijing.

ORACLE. 2003. Oracle Label Security Administrator's Guide. *Oracle Corporation*. http://download.oracle.com/docs/cd/B14117_01/network.101/b10774.pdf.

ORACLE, 2008. Oracle Database Security Guide, Oracle RDBMS 10gR2. *Oracle Corporation*. http://download.oracle.com/docs/cd/B19306_01/network.102/b14266.pdf.

SOX, 2009. Sarbanes-Oxley: Financial and Accounting Disclosure Information. http://www.sarbanes-oxley.com/section.php?level=1&pub_id=SOA-Manual

TPCH, 2008. TPC Benchmark H Standard Specification Revision 2.8.0. *Transaction Processing Performance Council*. http://www.tpc.org/tpch/spec/tpch2.8.0.pdf.

Vimercati, S., Foresti, S. *et al.*, P. 2008. Controlled information sharing in collaborative distributed query processing. In *Proc. of ICDCS 2008*, Beijing.

Yang, L. 2009. *Teaching database security and auditing*. ACM SIGCSE 1(1), pp. 241—245.