Rule Based Access Control for Granular Security in Databases

Mark A. Sherer
CPSC6126 Information Systems Assurance
Columbus State University
Columbus, GA USA

ABSTRACT – “Database security administration is one of the top priorities for today’s IT departments and it can be a cumbersome and frustrating task. As corporations become larger the granularity of the security becomes finer, more complex and unmanageable after a certain point. This paper examines a solution that will partition the database into fragments and use declarative rules to associate access constraints to the fragments to make access control manageable.”

INTRODUCTION
One of the main objectives of database security is to protect the data from unauthorized access while making it available to authorized users inside and outside the company. The data is typically accessed by the users in a variety of ways, from special purpose applications to ad-hoc queries. Because of the variety of behavior that the applications exhibit when accessing data the applications access rights are treated in different ways, some of them may be trusted while others are more restricted. The database itself typically uses a schema based approach where certain sets of operations are restricted to certain schema objects. The security policies of an organization may not be able to be implemented using the standard access control language of a DBMS so the enforcement of constraints is split between the DBMS and the applications. Being able to move all of the access constraints into the databases environment is a desirable goal for several reasons. It would simplify the job of the security administrator as well as the developers while providing more security by not allowing the applications to circumvent the database access constraints.

Using role based access control allows the security manager to group users, privileges and database objects. The solution being examined proposes to take advantage of this grouping by enhancing it with finer granularity constraints. To do this the database will be partitioned into fragments and use a set of declarative rules to specify the access constraints for them. The rule will specify what set of privileges are to be granted based on the users position in the company. The tables will be partitioned horizontally into fragments; the fragment could be a single tuple or the entire table. This will enable access constraint to be applied at the tuple level. The primary and foreign key relationships are used for inheritance of access constraints between fragments of related tables. The access control rules will be transformed into triggers that will be used to maintain the access constraints. The rest of this paper will be organized as follows: Section two will discuss how the database rules and triggers are used in conjunction with the access control module and table fragmentation to achieve the goal of access control. This will be followed by a brief discussion of related work, a proposal to enhance the solution and a brief summary.

TRIGGERS, RULES, FRAGMENTS AND ACCESS CONTROL
The focus of the access control being proposed is more applicable to maintenance of the information in the database than to general accessibility. The solution employs the use of access control rules that are transformed into database triggers. The reason for this is that when a database trigger is executed it has the ability to perform a check either before or after an action has been performed to determine if it is valid based on constraints contained within it and then roll back the transaction(s) if a constraint is violated. Another reason that triggers are used as the rules is because their granularity can be applied at the tuple level or a set level. The triggers in this case are created based on the access constraints using a rule compiler.

The model for access control uses the role concept in that individual’s rights are determined by their
position within the given organization. A role in this case is a named set of object privileges that are granted to users or to other roles. The role based access control is augmented with the use of rules that define the structure of the organization and the ownership of objects. Database operations are allowed if they comply with the role and the rules associated with it. The organization is modeled as a hierarchy of units with each user being assigned to a given unit.

The mapping of tuples is done by partitioning the tables into non overlapping fragments that are assigned to units. The members of the organizational unit have the privileges to modify the tuples. Database operations are allowed for a given transaction if all of the tuples involved belong to a fragment that is modifiable by the unit that the user belongs to. To ensure a well behaved rule set the access control is based on predefined fragment types that have a corresponding trigger implementation. The first fragment type consists of the entire table. All of the contents of the table will be modifiable by the unit to which the fragment is assigned. The second fragment type is attribute based; the values of the attribute dictate what unit the fragment is assigned to. All of the users assigned to the unit will be able to update the fragments that contain the values for that attribute. The third fragment type is foreign key based and perhaps the most powerful and useful of all. The rules are applied transitively by default to tables with foreign key fragmentation. The transitivity can become several levels deep and is only limited by the SQL compiler of the database being used. The transitivity can be broken if needed using the syntax for the rule language. Using the fragmentation in this manner simplifies the manner in which the rules are implemented and avoids repetition of rules on several tables. The fourth type, arbitrary access restrictions, is not really based on fragments of the database. The arbitrary access restrictions are coded by the programmer to perform special access controls that are compiled into the generated triggers. The example used, is if updates are not allowed on weekends, would allow a programmer to create executable SQL to prevent updates from occurring on the weekends. It is noted though that the syntax of this code when used as a rule is left up to the programmer to verify for correctness.

All of the information in regards to the hierarchy of units, who belongs to what unit, the fragmentation definitions, and the rules that are applied for the given units are stored inside the database. The triggers are generated and applied based on the rules and the roles associated to the units.

RELATED WORK
Cook and Gannholm applied for a patent in 2004 for their Rule Based Database Security System and Method. There are two main methods that are employed in the rule engine, the update manager and the access manager. The update manager is used to process insert, update, and delete requests. When the update manager receives a request it checks the request by applying the rules that have been defined and then either passes the request on to be processed or rejects the request and returns an error message. The access manager is used to process queries to retrieve data. When the access manager receives a request it applies object level access rules and modifies the request based on them. The request is then processed and the results are returned to the access manager which then applies field level access control rules to further filter the data before the results are sent to the user. Using this approach moves all of the management of security policies to the rule engine. Unfortunately there is no indication as to how robust this process is, so it is hard to determine if this approach is a good one or if it might be a bottleneck.

PROPOSAL
The solution uses fragmentation horizontality, but not vertically. I think that the solution could be improved if vertical fragmentation were added because there are certain tables where users should only be able to update part of a tuple. An example of this would be an HR table that contains employee information that could be updated by the user. If an employee has an address change it would make sense to me to allow them to update it. The reason for the vertical fragmentation is to prevent them from updating other information that may be included in the same tuple, such as their pay grade. With the use of vertical fragmentation this problem could be controlled.

CONCLUSION
The solution as proposed tends to be a bit complex in my opinion and doesn’t really alleviate the problem of complexity in applying granular security. The exception I would have to this would be based on the
use of the foreign key fragmentation concept; this could be a big advantage if a majority of the rules were based on this concept. The overall advantage to using this solution is that it does move all of the security control for updates into the DBMS being used and that would certainly be an advantage for any company. The solution was supposed to have increased performance by 50%. I have a hard time believing this because nothing in the database or the applications being used was supposed to have been changed except the addition of the tables to support the rules and the unit hierarchy, the table fragmentation, and all of the triggers used to enforce the access control. To me this would add a lot of overhead on the DBMS being used and probably decrease performance. The proposal to use vertical fragmentation as well as horizontal could prove to be resources intensive depending on the size of the database tables being fragmented and the location of these fragments. The vertical fragmentation as proposed may also have to actually occur outside of the realm of the use of the solution which may cause it to become even more complex. For these reasons it may not be feasible to use vertical fragmentation for some environments.

REFERENCES
