An analysis on Fine-grained Access Control in Databases

Prasanna M. Rathod
Student M.E. First Year H.V.P.M. COET, Amravati

Vrundan R. Parode
Student M.E. First Year H.V.P.M.COET, Amravati

R.R.Keole
Asst. Prof. H.V.P.M. COET, Amravati
ranjeetkeole@gmail.com

ABSTRACT

Today, there are countless terabytes of data processed by IT systems, and we store a measurable portion of that data in the relational database management systems (RDBMS). Access control in these IT systems, is one of the cornerstones of any Information Security Policy. The granularity of such access control can be on different levels, like on directories or folder level, database level, table level, and even on individual record (tuple) and data field level.

In this report, I survey various models and mechanisms of fine-grained access control in databases. Different models exist for providing access control at level finer than tables. This paper considers existing and upcoming theoretical models as well as models currently implemented in various database systems.

Keywords

RDBMS, IT System, Fine-grain access, Tuple

1.   INTRODUCTION

Access control is an integral part of databases and information systems. Granularity of access control refers to the size of individual data items which can be authorized to users. There are many scenarios that demand _ne-grained access control:

1. For an academic institution's database that stores information about student grades, it may be desired to allow students to see only their own grades. On the other hand, a professor should get access to all grades for a course she has taught.

2. For a bank, a customer should be able to query his/her account balance, and no one else's balance. At the same time, a teller should have read access to balances of all accounts but not the addresses of customers corresponding to these balances.

3. Who would consider opening production systems, such as order entry, inventory and customer support, to customers and partners without the ability to strictly limit data access?

4. A Web hosting company may wish to run the HR and Payroll business of other companies. Different companies want different personalization. Some want access to raw data to run business analysis reports that best suit their corporate standards.

As seen in above scenarios, there's need to apply security policies to data rather than through what means it is accessed. This report is structured as follows; Section 3 discusses ASP based SCM as a larger example. Section 4 explains how current generation systems use application-tier access control and it's drawbacks. Next, Section 5 touches upon the basic access schemes. Section 6 provides the recently introduced concept of Database-tier firewall. Section 7, discusses query modification based models including Oracle's VPD. Further

Section 8, discusses a model based on algebraic manipulation of views and extended Non- Truman model based on authorization views. Finally, the paper concludes with a summary and pointers to the references

2.   APPLICATION SERVICE PROVIDER BASED SUPPLY CHAIN MANAGEMENT

B2B partners usually form a chain for providing a service or manufacturing a product. The partners along the chain are either inventory buffer points or value adding points. This chain is referred to as a supply chain or supply web. The network of suppliers and consumers is modeled as a graph with nodes representing entities such as suppliers, manufacturers, wholesalers, retailers etc. A supplier-consumer relationship is represented as an arc from the supplier to the consumer. The graph is a multi-stage or multi-partite graph with each stage comprising entities of a given type. Goods or products flow from a node to one connected to it from supplier to consumer while orders and payments flow in the reverse direction. The graphs linking suppliers and customers for various products are represented and stored in tables in a RDBMS.

Business process support is assumed to be provided by an Application Service Provider. This is often employed when the client is a small or medium enterprise which lacks the resources or manpower to support such an application. In such a model, the ASP provides the database as a service; it hosts the shared database and allows for custom applications to be deployed. Thus, every application logging in the database as a particular supplier or customer should be able to get access only to data it is allowed to access.

Some access constraints expected to be posed by various customers and suppliers are as follows;

1. A node requires that its sales and inventory levels of item x should be visible to only its direct and indirect suppliers of item x and no one else.

2. A node requires that the orders placed by it of item x be visible to only its direct or indirect suppliers of item x. However, the price information should be visible only to its direct supplier of x.

One solution for the above situation is to develop custom applications for suppliers and customers and allow access to data using only these applications, where these applications internally make sure all security constants are met. That is, each application has access to all data but allows manipulation based on authorization of users. Of course, this implies no node can wish to use its own application to access data, all applications are to be provided by the ASP.

3. APPLICATION-TIER ACCESS CONTROL

Current generation Information systems bypass DB access control facilities and embed access control in application programs used to access the DB. For example, an application may authenticate the user using a password and then provide access to data user is authorized to manipulate.

Although widely used, this approach has several disadvantages:

1. Access control has to be checked at each user-interface. This increases the overall code size. Any change in the access control policy requires changing a large amount of code.

2. All security policies have to be implemented into each of the applications built on top of this data (e.g. OLTP and decision support applications using the same underlying data).

3. Given the large size of application code, it is possible to overlook loopholes that can be exploited to break through the security policies, e.g. improperly designed servlets.

4. Also, it is easy for application programmers to create trap-doors with malicious intent, since it is impossible to check every line of code in a very large application.

For the above reasons, _ne-grained access control should ideally be specified and enforced at the database level.

4. BACKGROUND ON ACCESS CONTROL MECHANISMS

Access control determines whether access to a resource is permitted. Permissions and authorization of users or processes are defined according to the policies of the business. An access control policy basically specifies a set of rules that describe the methods in which a client can access a server.

4.1 Access Control Matrix

An access control matrix is a simple mechanism for the storage of access control information. It is a table in which each row represents a subject (user), each column represents an object (the object can be a _le or a record etc.) and each entry is the set of access rights for that subject to that object. In general the access control matrix will be sparse, because most users will not have access rights to most objects. Every subject will, however, be mapped with every object (subject, object, rights).

This approach can provide very fine grained security control. The problem is the more fine grained the control becomes the more entries are required in the table. In a big system the table can quickly become very big and difficult to manage and slow to search.

4.2 Access Control Lists

A different approach is to use capabilities and access control lists. The first method is to specify only the objects that a user may access. This approach is called a capability. It can be seen as a token giving the possessor certain rights to an object. The capability can be stored with the subject.

A second method is to create a list that specifies which subjects can access an object, including their access rights. This approach is called an access control list (ACL). The ACL can be stored with the object or the resource.

These methods also have some problems: Firstly, it is very restrictive, because a user can only access the objects explicitly named in its capability, and object can only be accessed by a specified set of users in its ACL. Secondly, the management and administration of these approaches become impractical and near impossible; for example, in a big system with many objects it becomes impossible to update and check all the object's ACLs when one subject leaves the system.

4.3 MoFAC: Model for Fine-grained Access Control

MoFAC requires various queries to be executed to be pre-specified as transactions and thus is more suited for particular applications like banking, however still keeping the access control at DB-tier increases security and flexibility.

The basic unit of grouping records is called a Record Group(RG) which is a collection of record IDs and certain other information relevant to records belonging to that RG. The other information may be aspects like What transactions may be executed on records belonging to this RG; and What business roles may access the records belonging to the RG etc.

Various similar type of user needs are classified as role profiles eg: a generic 'Teller' profile at a bank. With each role a list of what transactions it can access is added. Further, Role Profile Assignment records are kept which describe which user can log on as under which profile and it's level, for example, X may be a manager , level 2.

As a user logs under a specific role, and requests a transaction to execute, the system checks whether the current role can execute the transaction or not, access may be denied at this point. If the required transaction can be accessed, then the transaction is checked for affected rows, using their IDs various RGs are listed, and each of that RG is checked for whether this Transaction under this profile can be executed. Only then the final execution takes place or the query is rejected.

The basic idea behind MoFAC, lies in the fact that access to different records in the same file can be treated differently through a subject wanting access to record possessing (offering) certain rights, and the resource (object) to which access is required, demanding certain rights. Only if the offered rights satisfy the demanded rights, is the access granted.

Although, by requiring a lot of pre-specifications, extra management is still needed. However the method more elegant than the brute force method of having an Access Control List for individual records. Also, addition & deletion of users and records is much easier than in simple models

4.4 View Based Security

Currently, authorization mechanisms in SQL permit access control at the level of complete tables or columns, or on views. It is also possible to create views for specific users, restricting access to data by granting rights only to certain views & tables for each user. These allow those users access to only selected tuples of a table. However, complex role based access control conditions are difficult to implement. In some cases view security can be bypassed (if users have access to base tables).

Also, administration of security policy becomes complex by views. When a security policy is added, changed, or removed, it's difficult to determine what exactly to do with each view. An administrator cannot tell whether, by changing security

policies through altering or dropping a view, he/she is breaking an application.

Again this approach is not scalable with large numbers of users.

5. CONCEPT OF DB-TIER FIREWALL

The database tier firewall, as envisaged in [2], is a security mechanism that validates and then transforms an application generated SQL query to produce a query understood by the database and in keeping with the access restrictions implied by the security policy. For this purpose it uses abstract-to-logical schema mappings, internally created view definitions, security policy _les and information from the session context. To facilitate ease of programming, it offers the programmer a set of APIs that complement the provision for expressing queries in SQL.

The principal building blocks of the database tier firewall include:

1. Query Validator: It checks whether the input query submitted by the application is valid with respect to the abstract schema i.e. whether the table objects and attributes referred in the query are all contained in the abstract schema visible to the application developer. It also contains a parser which checks whether the given query conforms to standard SQL syntax.

2. Query Generator: The application program has the liberty of intimating its queries to the database in the form of API calls as published by the database firewall. Whenever the firewall receives such a request, it transforms the request into an appropriate query statement involving tables in the abstract schema and delivers the query to the query transformer.

3. Query Transformer: This block transforms a query on the abstract-schema into an equivalent query containing internally defined view names or tables in the logical-schema. The query transformer would then substitute one or more order tables in the logical schema for the single orders table in the abstract schema.

4. Query Filter: This is the most security-aware component in the firewall, performing operations such as appending a predicate to a query and projecting out only the columns that the current user is authorized to see. For this purpose, it consults the security policies administered by the security manager. In addition, it may also consult attributes in the session or application context. In effect, this block does both row-level and column-level filtering.

One of currently implemented solution for Query filtering or providing fine access control is Oracle Virtual Private Database, explored next.

6. QUERY MODIFICATION BASED MODELS

6.1 Oracle's Virtual Private Database

The Virtual Private Database (VPD) is the aggregation of server-enforced, fine-grained access control, together with a secure application context in the Oracle9i database server.

Fine-grained access control in VPD relies upon dynamic query modification to enforce security policies on the objects with which the policies are associated. Here, query refers to any selection from a table or view, including data access through a query for update, insert or delete statements, or a sub-query. From the user's point of view, the set of rows that he or she has access to only exists inside the table. Each user should only be allowed to do modifications on the rows that follow the security rules. Based upon these security rules, Oracle generates a predicate clause that transparently appends to the user s SQL statement. This concept is called Virtual Private Database.
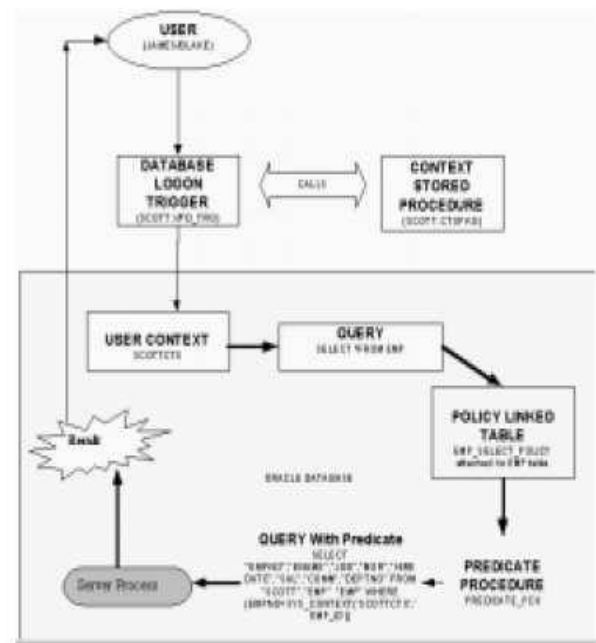
6.1.1   ARCHITECTURE OF ORACLE VPD

Figure 1: Architecture of Oracle VPD

As shown in Figure 1, whenever a user connects to the database, the database logon trigger fires, which calls the Context stored procedure to set the defined context.

Policy rules are defined according to the user context. After connection, when the user issues a query or DML statement against a policy-linked table, Oracle determines which policy needs to be linked according to the statement type. In other words, you can link various policies depending on the select, insert, update, or delete statement. The policy further indicates which function to call to implement security rules. If you have different policy rules for selecting, updating, inserting, or deleting data, you can create different functions to form a predicate.

Depending upon the policy rules defined in the predicate procedure, the resultant predicate is appended to the user's query. The server process now executes this predicated query and sends the results back to the user.

Thus the process of query modification is entirely transparent to the user.

6.1.2   LIMITATIONS OF VPD

Oracle VPD currently provides no mechanism to project out certain attributes i.e., no column level security is present.

It is difficult to write predicates involving cases of cross-ref, joins of tables etc., in complex cases the user may be

authorized to view certain data, but a correct query is transformed into an invalid one by attaching of predicate.

The user's query is relatively simple with the onus of ensuring secure access resting squarely with the author of the security policy. Writing policy functions corresponding to business policies is a lot of work.

6.2   Truman Models

The idea behind the Truman security model is to provide each user with a personal and restricted view of the complete database. User queries are modified transparently to make sure that the user does not get to see anything more than her view of the database. The returned answer is correct with respect to the restricted view, if we assume the database has no other data.

In the Truman model, the DBA defines a parameterized authorization view for each relation in the database. This view defines all that the user can access from this database relation. The user query is modified transparently by substituting each relation in the query by the corresponding parameterized view (the user can write queries on base relations in addition to the authorization views). Values of run-time parameters like user-id, times etc. are plugged in before the modified query is executed.

The parameterized view framework provides a more general way to express authorization policies than the technique of adding where clause predicates used in VPD, since it can additionally perform other actions such as hiding or falsifying specific attribute values which cannot be done by VPD (for example, if an authorization policy permits a student to see her grades tuple but only after the grade attribute has been modified as - all A and B grades to High, and all other grades to Low).

6.2.1   *QUERY MODIFICATION SCHEME OF INGRES*

As discussed in section 5, in INGRES queries are handled by a "query modification" algorithm. Essentially, the algorithm searches for permitted views whose attributes contain the attributes addressed by the query, and the qualifications placed on these attributes are then conjoined with the qualifications specified in the query. The algorithm is attractive because when a query exceeds it's permissions, it delivers the data that is within the permissions.

However, there are a few limitations. First, permissions are granted for actual relations or views of single relations, and it is not possible to grant permissions to views of several relations. Secondly, the algorithm does not handle rows & columns symmetrically. That is, if a request exceeds permissions in rows, rows within permissions are returned. However, if an extra column is asked for, the full query is denied rather than projecting out extra columns. Also, there are few cases in which the algorithm delivers less than what user is permitted to view.

6.3   Limitations Of Query Modification Approach

Following are the various limitations of existing query modification based models;

1. A major drawback of VPD and Truman models results from the fact that the query that is executed on the database is a transparent modification of the user query. This may cause inconsistencies between

what the user expects to see and what the system returns.

2. Due to queries being modified transparently, it may not return the results exactly expected by the user. For example, there may be tuples excluded from the result due to user not having authorization to access them, however the successful execution of query may lead the user to interpret that these are only tuples which exist leading to possibly wrong misinterpretation, which could be avoided if the query is rejected or user informed of access constraint not being met.

3. The rewritten query executed by the system may be different from the query posed by the user, and may have very different execution characteristics. For example, if each base relation is substituted by a view, and the views are complex (as they may be, if they express complex authorization policies), the rewritten query may be quite expensive to execute. In the case of VPD, equivalently, the conditions introduced in the where clause may have complex sub-queries, which may be expensive to execute. In the case of VPD, equivalently, the conditions introduced in the where clause may have complex sub-queries, which may be expensive to execute.

A user query and an authorization view used to replace a relation in the query may both contain the same test, leading to redundant tests. If the test involves a join, the Truman-modified query may also contain redundant joins. Removal of redundant joins is an extra task for the query optimizer, and if not removed, the redundant joins would result in wasted execution time.

7.   NON-TRUMAN & OTHER MODELS

Under the Non-Truman model, the query is subjected to a validity test, failing which, the query is rejected and the user is notified about this (this can be handled like an exception by the user application). If the query passes the test, it is allowed to execute normally, without modification.

Here, First method based on algebraic manipulation of view definitions is described. Then, a model based on notions of conditional & unconditional validity of queries is discussed

7.1   Model Based On Algebraic Manipulations of View Definitions

This model allows the administrator to describe the access permissions using views that are defined by conjunctive relational calculus expressions 1. The basic principles of this model are:

1. Database access is specified in terms of views: a set of views is defined, and each user is granted permission to access one or more views.

2. Users direct queries at the actual database, not at any particular view.

3. When a request is presented to the database system, it is performed both on the meta-relations (holding definitions of views), resulting in a mask, and on actual relations, resulting in an answer.

4. The mask is then applied to the answer, yielding the data that may be delivered to the user. This reply may be accompanied by statements describing the portions delivered.

However, the model has a few limitations in that firstly, it handles only specific types of retrieval queries. It does not handle views with disjunctions and aggregate functions. Secondly, extending the model is difficult as by allowing partial answers, set difference and aggregation can result in wrong answers. Finally, the algorithm yields only permitted views (masks) that can be expressed with attributes requested. Views expressed using additional attributes are not handled.

7.2 Non-Truman Model Based On Validity Notion of Queries

A fine-grained access model is based on authorization an view that allows authorization transparent querying; that is, user queries can be phrased in terms of the database relations, and are valid if they can be answered using only the information contained in these authorization views.

7.2.1 KEY FEATURES OF THE MODEL

1. Access control is specified using Authorization views. A parameterized authorization view is an SQL view definition which makes use of parameters like user-id, time, user-location some of which may be taken out from the session context. The DBA can create several authorization views, one for each access policy and any of those views can testify for the validity of the user's query. The model works within the basic SQL framework and does not require the DBA to encode policies using a separate rule language.

2. Queries to allow to be written in an authorization-transparent manner, that is, queries can be written against the database relations without having to refer to the authorization views. Given a user query (phrased in terms of database relations or views), the system checks if the query is valid, that is, it can be answered using the information available in the authorization views that are accessible to the user. If found to be valid, the query is allowed to execute as originally specified, without any modification, otherwise it is rejected.

7.2.2 QUERY VALIDITY AND INFERENCE RULES

A user query q can be answered using the information contained in the authorization views available to the user if there is a query q' using only the authorization views that is equivalent to q, i.e., the two queries give the same result on all database states. Such queries q are classified as unconditionally valid.

The certain queries can be answered using the information contained in a set of authorization views, even if they cannot be rewritten using the views. They characterize such class of queries, called conditionally valid queries, that can be answered using the information contained in a set of views in a given database state.

A set of powerful inference rules which can be used to infer the unconditional and conditional validity of queries are given

in the model and also mechanism to efficiently check the validity of a query by incorporating these rules into a query optimizer are provided.

Unlike the Truman model, the Non-Truman model avoids the pitfalls of the query modification approach and allows a great deal of flexibility in authorization, such as authorization of aggregate results.

However, the inferencing mechanisms described are quite complex to implement. Also, in some cases the algorithm provided may not be able to infer validity of some unconditionally valid queries.

8. CONCLUSION

This paper highlighted the approaches taken to provide _ne-grained access control to data in Relational DBMS. There are simple to implement basic schemes which fail as the size of data & users grow. Fine-grained control, on tuple or field level increases management function significantly, and also makes it more complex for these basic schemes.

The currently implemented market solution, Oracle's VPD leaves much to be desired. The other query modification approaches are also not complete and do not handle all cases. Further, performance issues in query modification approaches may also of concern to large applications.

On the other hand, methods which modify(or mask) results rather than modifying the query handle only a subset of operations and language desired while the complexity of complete implementation of validity based models is quite high.

Thus, while a lot of applications requiring _ne-grained access control are emerging and can gain much from a db-tier access model, no one model is still in place and more work can be done in the areas of implementing generic models as well as customizing existing methods for various applications.

10. REFERENCES

[1] The virtual private database in oracle9ir2: An oracle technical white paper. http://otn.oracle.com/deploy/security/oracle9ir2/pdf/vpd9ir2twp.pdf.

[2] Santosh Dwivedi, Bernard Menezes, and Ashish Singh. Database access control for e-business a case study. In 11th International Conference on Management of Data COMAD (to be presented), 2005.