ISSN:0975-9646

A Fine Grained Access Control Model for Relational Databases

Neha Sehta[*], Dr. Suresh Jain[#]
[*]*Sr Lecturer, Department Of Information Technology*
S.D. Bansal College Of Technology, Indore, India
[#]*Director, KCB Technical Academy, Indore, India*

Abstract- **Nowadays most of the commercial work make use of relational database management systems (RDBMS) to store a substantial amount of their data. Accessing information over the Internet has become an essential requirement. Authorization mechanisms in SQL permit access control at the level of complete tables or columns, or on views, providing coarse granularity. There is no direct way to control which tuples can be accessed by which users. Fine Grained Access Control (FGAC) is one of the ways to ensure data security. As per the requirements the granularity of fine grained access control can be on directories or folder level, database level, table level, even on individual record (tuple) and data field level. Access control is the process of mediating every request to resources and data maintained by a system and determining whether the request should be granted or denied. The access control decision is enforced by a mechanism implementing regulations established by a security policy. Corresponding to different criteria for defining what should, and what should not be allowed, different access control policies can be defined. In this paper, a novel access control model is proposed, which provides fine grained access control to shared data to authorized users. In proposed implementation, we have created a set of meta-tables, to store the data that make up the security policies, registered users and their authorization information. This allows policies to be created (or changed) dynamically. The access permissions are stored in the form of quadruple <role, object, operation, policy>. Any change in policies doesn't affect the application program. Moreover multiple policies defined to regulate user access together are also supported, which facilitates smooth access to user having multiple credentials without having him to mention each of his credentials at the time he makes request.**

Keywords- Access Control Policy, Data Security, Fine Grained Access Control, RDBMS.

I. INTRODUCTION

Information is the most valuable asset for organizations. In our days information is stored in databases that become accessible from the Internet [17]. The information disclosure from such databases may have very serious impact on organization business. So new access control approaches for databases and especially for web databases have become a dire necessity. Thus appropriate access control methodology is to be used to provide access only to authorized person.

Access control is an integral part of databases and information systems [1]. Granularity of access control refers to the size of individual data items which can be authorized to users. There are many scenarios where we need some sort of FGAC:

- In an academic scenario, institution's database stores information about students, it may be desired to allow students to see only those tuples which store their own marks and/or fees details. On the other hand, a professor should be able to access all grades for a course she has taught.
- In banks, it is required that the customer should be able to query the balance only in his own account, and not others. Similarly a teller should enjoy read privileges to balances of accounts of customers but not other details.
- A company providing web services, may have assignments to carry out the HR and Payroll business of other companies.
- Electronic Health Records (EHR) containing confidential and sensitive information about the patients should permit the access to authorized users only.
- And many more…

Thus it is quite evident that security policies should be applied on data rather than through what means it is accessed. In this work we have proposed the access control system for an academic institute which provides access to information requested, if requestor abides by security policies.

The rest of this paper is organized as follows. Section 2 elaborates motivation behind the work, i.e. the problems which we generally face while working with web application which is using important and valuable data resources. Section 3 presents the related work in this area. Section 4 explains the proposed access control policy architecture and methodology. Section 5 concludes and describes some future work.

II. MOTIVATION

Current day database applications, with large numbers of users, sharing invaluable resources, require fine-grained access control mechanisms, at the level of individual tuple and column, not the entire relations/views. To control which parts of the data can be accessed by each user, fine-grained access control is often enforced in the application code, which has numerous drawbacks; these can be avoided by specifying/enforcing access control at the database level.

Currently, authorization mechanisms (using GRANT/ REVOKE) in SQL permit access control at the level of complete tables or columns, or on views. It is based on the System R authorization model; ownership administration with administration delegation.

GRANT privileges ON object TO users [WITH GRANT OPTIONS]

privileges = SELECT | INSERT | DELETE | . . .
object = table | attribute

REVOKE privileges ON object FROM users [CASCADE]

There is no direct way to specify fine-grained authorization to control which tuples can be accessed by which users. In theory, fine-grained access control at the level of individual tuples can be achieved by creating an access control list for each tuple. However this approach is not scalable, and would be totally impractical in systems with millions of tuples, and thousands or millions of users, since it would require millions of access control specifications to be provided (manually) by the administrator. As an alternative access control specifications based on user roles can be used.

For a web application, end users don't have database user ids; they are all mapped to the same database user id. The users of web applications can be casual users and their number is not limited. The browser does not directly connect to the database, but instead transfer a request to a web server. The server processes the request and if required it performs a transaction to the database

III. RELATED WORK

The known methods for providing protection in databases is to create users in database, assigning them unique user_id, password. Using such identity, the authorization provided by Grant / Revoke mechanism.

Fine-grained access control was first introduced as a part of the access control system in INGRES by Stonebraker and Wong (1974), which was implemented by query modification technology [2]. The basic idea of query modification is that before being processed, user queries are transparently modified to ensure that users can access only what they are authorized to access (Bertino et al., 2005; Wang et al., 2007). Views are used to specify and store access permission for users[4]. When a user submits a query, DBMS first finds all views whose attributes include the attributes of the issued query, and then add the predicates of these views to the predicates of the original query to form a new modified query, which will be carried out.

Oracle virtual private database (VPD) [15] also uses query modification to implement FGAC (Oracle Corporation, 2005). VPD supports FGAC through functions written as stored procedures which are associated with a relation. When a user accesses the relation, the function is triggered to return predicates, and the database rewrites the SQL statement submitted by the user to include these predicates. For providing enhanced access control, in addition to row level access control, column-level VPD has been added to Oracle to provide column-level access control, which in turn associates functions with columns.

Chaudhuri et al. (2007) also extended SQL language to support fine-grained authorization by predicated grants[3]. Not only the column- and cell-level authorizations, but also the authorizations for function/procedure execution were supported. Moreover, they designed query defined user groups and authorization groups to simplify the administration of authorizations.

Hippocratic databases [14] are another available solution to controlling database access. These databases were designed in order to take responsibility for the privacy of the data they contain (Agrawal, Kiernan, Srikant, and Xu, 2002) and are applicable to data-sensitive fields such as healthcare (Agrawal, Kini, LeFevre, Wang, Xu, and Zhou, 2004) and finance (Agrawal, Asonov, Bayardo, Grandison, and Johnson, 2005). IBM's Hippocratic Database (HDB) is one implementation that provides database security by enforcing security policies, auditing usage, and allowing data to be shared amongst databases (Agrawal et al., 2005).

IV. PROPOSED WORK

We put forward a framework for enforcing Role Based Access Control[7,8,9] at the database level rather than the application level using dynamic query rewriting. The proposed framework takes a user submitted SQL statement and dynamically rewrites it according to security policies defined. The query rewriter adds rules in the form of WHERE clauses to create a new SQL statement, that is then submitted to the RDBMS. This framework is DBMS vendor independent and allows for attribute-level granularity, where an attribute is the intersection of a row and column.

To implement proposed framework, we have created a set of meta-tables in which to store the data that make up the security policies, registered users and their authorization information. This allows for joins, necessary for policies to be created dynamically. The access permissions are stored in the form of quadruple <role, object, operation, policy> associated with a table join id as shown in table I given below.

TABLE I
ACCESS PERMISSION TABLE

Role	Object	Operation	Policy_id	Join_id
Student	Fees_detail	Select	P1	J1

A. System Overview

The proposed system mainly comprises of four modules:

1) Authentication and Authorization Module: This module basically authenticates user and extracts user authorization information. User authorization information contains various roles user possesses and the attributes and their values specific to particular role.

2) Query Interface: This module presents GUI to build query by selecting data source and applying filters.

3) Policy Engine: This module takes user authorization information and query build by user as input and based on policies stored in policy store generates partial WHERE clause of query to be submitted to the data source. Policy is mapping between data item and corresponding credentials required for those items. Our system works only on closed policy [5]. A *closed* policy permits specification of only positive authorizations and allows only those accesses that are explicitly authorized. In contrast, an *open* policy permits specification of only negative authorizations and allows only those accesses that are not explicitly denied.

4) Query Modifier And Generator: This module combines predicate generated by policy engine to the filter conditions given by user. Thus query given by user is appended with modified WHERE clause.

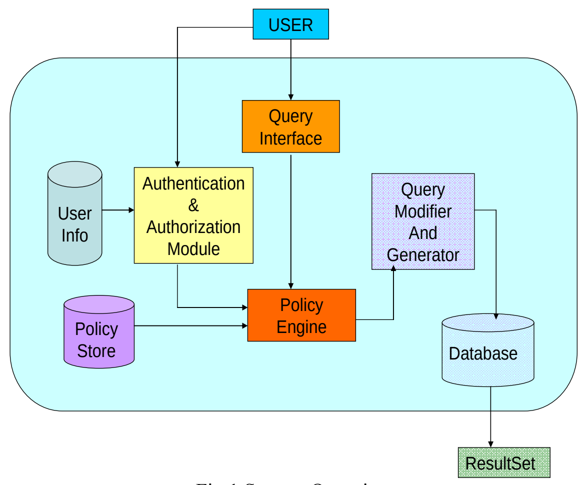

Fig 1 System Overview

B. Database Design

1) Database to store user authentication information:- User id and password is used to authenticate user.

2) Database to store user authorization information: Figure 2 shows the ER diagram to show how user authorization information is stored.

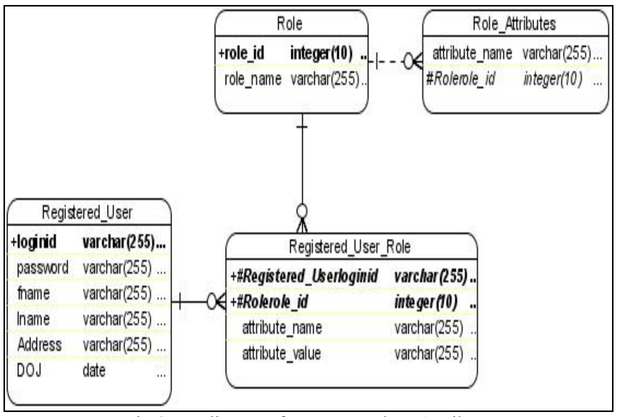

Fig 2 ER diagram for User- Role – Attribute

3) Database to store Access Control Policies: Access Control Policies defines high-level rules according to which access control must be regulated. Policies define what a user group can access and the filtering condition applied to requested database. A fact that every Boolean condition can be reduced to Conjunctive Normal Form (CNF) is used here for representing the boolean condition [6]. CNF is defined as ANDing of various disjunctions where disjunction is an ORing of variables. A general CNF can be represented as

(A or B) AND (C) AND (D or E or F) AND ……

A policy is defined as ANDing of boolean conditions and that boolean condition can further be defined as ORing of boolean conditions which is stored in database with the help of three tables as shown below.

TABLE II
POLICY TABLE

Policy_id	Condit_id
P1	C1
P1	C2
P1	C3
P2	C1
P2	C4
P3	C1

TABLE III
CONDITION TABLE

Condit_id	Description id
C1	D1
C1	D2
C2	D3
C2	D4
C3	D5
C4	D6

TABLE IV
CONDITION DESCRIPTION TABLE

desc_id	Opd1	ROp	Opd2
D1	Fees_detail.enrollment_no	=	Enrollment_no

From the table II above:
Policy P1 → C1 Λ C2 Λ C3
Policy P2 → C1 Λ C4
Policy P3 → C1

After replacing C1, C2, C3, C4 using table III
Policy P1 → (D1 V D2) Λ (D3 V D4) Λ D5
Policy P2 → (D1 V D2) Λ (D6)
Policy P3 → (D1)

Each description of boolean condition can be defined as shown in table IV.

Example 1:

Policy 1: - A Student can access only his fees detail

C1. User should be a valid student

C2. Can only see tuples which have his enrollment no as attribute

C3. Can request only in working hours i.e. on or after 10:00AM

C4. Can request only in working hours i.e. on or before 5:00PM

Based on access defined on role, data source and privilege, policy is extracted as P1 (from table I). Here Policy 1 is defined as (from database defining policies)
P1 → C1 Λ C2 Λ C3 Λ C4.

Further these conditions are described as D1, D2, D3, and D4. Based on the roles the user possesses and attributes associated above conditions could be framed as in the format shown in table IV

Desc_id	Opd1	ROp	Opd2
D1	year(sysdate()-6)	<=	year_of_ addmission
D2	fees_detail.enrollment_no	=	enrollment_no
D3	hour(now())	>=	10
D4	hour(now())	<=	17

Now when a student access fees_detail by the query

SELECT * FROM fees_detail

Based on access permissions defined on role Student(Table I), data source query is modified as

SELECT * FROM student natural join fees_detail WHERE ((((fees_detail.enrollment_no='IT071028')) and ((year(sysdate())-6<='2007')) and ((hour(now()) >= 10)) and ((hour(now()) <= 17)))

This modified query is fired to database and result set obtained is according to the authorizations defined.

V. CONCLUSION

The major contributions are summarized as follows:

1. We can represent access control condition in Conjunctive Normal Form. Policy could be breakdown to atomic level (variable) to store in database.
2. We implement the FGAC model in relational database as a part of DBMS. Any change in policies doesn't affect the application.
3. Policies can be designed for any underlying database.
4. If the requestor possesses multiple credentials, multiple policies could be combined and information is provided for which he is authorized. Moreover filters could be applied as per the requirements.

The system developed is capable of defining access control rules using user roles, attributes specific to roles and their values and providing fine grained access control which current SQL authorization does not. Currently the system allows a user to only view the information i.e. the user can perform only the SELECT operation but in future the system can be expanded to provide the functionality of insertion, updation and deletion. The user can be allowed to perform filtering operation on the information accessible to him.

The system developed only supports positive authorizations; Positive authorizations state those accesses that are to be allowed. FGAC using Query Modification approach definitely creates complex queries, which surely degrades the performance. Performance evaluation and analysis can also be done on large database in future.

REFERENCES

[1] Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In Proceedings of the 2004 ACM SIGMOD international conference on Management of data. ACM Press, 2004.

[2] Stonebraker, M., Wong, E., 1974. Access Control in a relational Database Management System by Query Modification. Proc. ACM Annual Conf., p.180-186. [doi:10. 1145/800182.810400]

[3] Surajit Chaudhuri, Tanmoy Dutta, and S. Sudarshan, "Fine Grained Authorization Through Predicated Grants

[4] Motro. An access authorization model for relational databases based on algebraic manipulation of view definitions. In ICDE, 1989.

[5] E. Bertino, S. Jajodia, and P. Samarati, "A Flexible Authorization Mechanism for Relational Data Management Systems," in *ACM Transactions on Information Systems*, April, 1999

[6] Nirmal Dagdee, Ruchi Vijaywargiya: "Role based hybrid access control Methodology for shared Electronic Health Records".

[7] Ravi S. Sandhu, Edward J. Coynek, Hal L. Feinsteink and Charles E. Youmank: "Role-Based Access Control Models", IEEE Computer, Volume 29, Number 2, pages 38-47 , 1996

[8] Sabrina De Capitanidi Vimercati, Pierangela Samarati : New Directions in Access control, www.spdp.dti.unimi.it/papers/nato.pdf, 2002

[9] Pierangela Samarati1 and Sabrina De Capitani di Vimercati, "Access Control: Policies, Models, and Mechanisms".

[10] Wei Zhou and Christoph Meinel, "Implement role based access control with attribute certificates"

[11] Jie SHI, Hong ZHU, "A fine-grained access control model for relational databases"

[12] Bertino, E., Sandhu, R., 2005. Database security-concepts, approaches, and challenges. IEEE Trans. Depend. Secur. Comput., 2(1):2-19. [doi:10.1109/TDSC.2005.9]

[13] Bertino, E., Samarati, P., Jajodia, S., 1997. An extended authorization model for relational database. IEEE Trans. Knowl. Data Eng., 9(1):85-101. [doi:10.1109/69.567051]

[14] Agrawal, R., Kiernan, J., Srikant, R., Xu, Y., 2002. Hippocratic Databases. Proc. Very Large Data Bases, p.563- 574.

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

[16] J.D. Ullman. Principles of Database Systems. Galgotia Publications, 2nd edition, 2001.

[17] Alex Roichman, Ehud Gudes. Fine-grained Access Control to Web Databases. SACMAT'07, June 20–22, 2007.