A SYMMETRICAL APPROACH TO GRANTING AND REVOKING ACCESS RIGHTS IN DATABASE MANAGEMENT SYSTEMS

David A. Goldberg
Palindrome Corp.
850 E. Diehl Rd.
Naperville, Illinois 60563

Ali Oorooji
Department of Computer Science
University of Central Florida
Orlando, Florida 32816

ABSTRACT
The security mechanism of a database management system must provide means for granting and revoking access to data in the database. In current systems, when an authorizer wants to grant some access to a user, he specifies the new access independent of the current access rights (i.e., independent of the current status of authorization). On the other hand, when an authorizer wants to revoke some access of a user, he must determine how to change the current status of authorization to reflect the revocation. If this is determined incorrectly, the authorization status becomes invalid, and a user may gain improper access to data. It would be preferable if revocation could be performed symmetrical to access granting, i.e., it would be preferable if the authorizer were able to simply specify the access that should be revoked and let the system determine what changes should be made to reflect the revocation. In such a system, the work required by the authorizer is minimized, as is the chance of error.

This paper discusses this method of revocation, which we refer to as independent revocation (since the revocation is specified independent of the current status of authorization). An actual implementation of an authorization system which provides such revocation is described. The applicability of independent revocation to other database management systems is also discussed.

1. INTRODUCTION
A database management system (DBMS) is responsible for ensuring that users are not able to view or modify any data in the database for which they are not authorized. This responsibility is carried out by the security mechanism of the DBMS. This mechanism is comprised of two components, an authorization component and an enforcement component [1].

The authorization component is responsible for allowing an authorizer to define which users have access to what data items. This component must provide a means for granting and revoking access, as well as displaying the current access rights. The enforcement component is responsible for ensuring that no user may access any data for which he/she is not authorized.

Both of these components are essential for the security of the database. If the enforcement component fails, a user may be able to access unauthorized data. If the authorization component fails, a user may obtain improper authorization, and thus access data improperly.

The authorization component must be more than functionally correct. It must also provide the authorizer with a flexible and easy-to-use system. If such a system is not provided, the authorizer may erroneously grant access which was not desired. One such source of errors occurs during revocation of access. In current systems, when an authorizer wants to revoke some access of a user, he must determine how to change the current status of authorization to reflect the revocation. If this is determined incorrectly, the authorization status becomes invalid, and a user may gain improper access to data.

It would be preferable if the authorizer were able to simply specify the access that should be revoked, and let the system determine what changes should be made to reflect the revocation. For example, an authorizer should be able to tell the system: "Fred should not be able to access the data of the employees in the toy department." If Fred currently did not have access to this data, the status of authorization would obviously be unaffected. However, if Fred did have such access, the system would alter its status, to reflect the revocation, automatically. In such a system, the work required by the authorizer is minimized, as is the chance of error.

This paper discusses this method of revocation, which we refer to as independent revocation (since the revocation is specified independent of the current status of authorization). Section 2 discusses, in more depth, the basic idea of independent revocation. Section 3 describes an actual implementation of an authorization system which provides such revocation. The applicability of independent revocation to other database management systems is discussed in Section 4. Finally, in Section 5, some conclusions and a summary are presented.

2. INDEPENDENT REVOCATION
One way of modeling access control in any security system is by means of an access matrix [2, 3]. The rows of the matrix represent the subjects, such as users of the system, and the columns of the matrix represent the objects in the system which are to be protected, such as files. The entries in the matrix specify the type of access a specific user has to a specific object. The entries are determined by the types of operations on the objects which are to be controlled.

An example of an access matrix is shown in Figure 1. In this case, user U1 has been given access to perform READ operations on object O2. User U2 has been given access to perform WRITE operations on objects O1 and O3. An empty entry in the matrix indicates the user has no access to the given object.

This model has been extended for modeling access control in database systems [4]. This extension appends a condition (or where clause) to each entry in the access matrix. The meaning of the condition is that the subject has the specified access to the object, providing the condition is satisfied. This allows modeling of security in database systems where it might be desirable to specify that a user has access to a data item dependent upon the content of that data item. Figure 2 shows an example of the extended access matrix. We see, for example, that Fred has READ access to the department data, but only for those departments on the second floor. An entry without a condition indicates that there is no condition which must be satisfied.

" The work reported in this paper was supported in part by Palindrome Corp.
It must be noted that such models of security do not specify the actual implementation of the matrix. In fact, if the matrix were implemented as such, it would probably be inefficient because the matrix tends to be sparse. One possible implementation is to store the access matrix as a table of access rules which are of the form:

(subject, object, access type, condition)

In this manner, only the non-empty entries of the access matrix are actually stored. Other methods of implementation have also been suggested [3].

As mentioned in Section 1, any database security system (or any other security system) must have some means of authorization. Such an authorization component would normally provide three major functions. First, there must be a facility to grant access to objects in the database, i.e., add entries into the access matrix. Second, an authorization should be given some means of viewing the current status of the access matrix. Third, there should be some way to revoke access (or some of the access) a user has been given.

In those existing database systems which represent the access matrix in the form of access rules, revocation is performed by specifying the access rule to be removed from the system. For example, in the INGRES system [5, 6], access rules are stored in a directory relation called "protection" [4]. Revocation is performed by executing the "destroy" command which specifies that the access rule with a given identifier should be removed. For example,

\[
\text{destroy permit employee 3}
\]

would remove the access rule pertaining to the employee relation, with the identifier 3.

Such a system provides a basic means of revocation, but is limited in the flexibility provided to the authorizer. Suppose, for example, the status of authorization was as shown in Figure 3. Suppose it was desired to restrict Fred from seeing those employees who work in the Toy department. To perform this revocation, the authorizer would have to delete three different access rules, and perform grant operations to specify the new form of the access matrix, indicating the fact that Fred no longer has access to the Toy department employees.

It would be preferable to provide the authorizer with the ability to specify revocation independent of specific access rules stored in the system. For example, the authorizer should be able to say "Revoke Fred's access to the employees of the Toy department", and let the system determine what changes, if any, should be made to the access rules. Such a facility would reduce the amount of work required by the authorizer. Instead of having to examine and modify the existing status, the authorizer simply specifies the access to be revoked. Additionally, the possibility of introducing errors into the status of authorization, by incorrectly reformulating the access rules, is eliminated.

Depending on the current status of authorization, and depending on the revocation being performed, various results can occur. If the user was not previously granted the access which is being revoked, the access matrix would obviously not be affected. On the other hand, if the result is to actually revoke some access previously granted, the access matrix would be altered to reflect the revocation. Such modification might simply be the deletion of an entry in the matrix (the elimination of an access rule). However, in some instances, entries must be modified in some way to properly reflect the revocation. For example, a revocation which is dependent upon the context of data (such as the example with Fred above) would require modification of the associated condition for one or more entries in the access matrix. The various situations the system must handle are discussed more thoroughly in the next section, where we discuss an actual implementation of an authorization system which provides independent revocation.

3. AN IMPLEMENTATION OF INDEPENDENT REVOCATION

An authorization system which provides a form of independent revocation has been implemented [1] in a prototype database management system known as RRDS (Relational Replicated Database System) [7]. This section first discusses the basic structure of RRDS, and then describes the implementation of the authorization system. The enforcement component of the RRDS security mechanism, which utilizes the idea of query modification [8], is not included in this paper (the reader may refer to [1] for details of the enforcement component).

3.1. The Structure of RRDS

The structure of RRDS, a backend database system based upon the relational data model, is shown in Figure 4. The system consists of a controller, a broadcast bus, and a number of identical processing elements referred to as replicated computers (RCs). The system is designed to provide fast operations on large databases via multiprocessing. Since the RCs are identical, including the software on them, the system is easily extendible.

The controller is responsible for communicating with the host and parsing the queries. It is designed to perform as little function as
The security system of RRDS is considered a "closed" system. That is, if no specification has been given to allow a user access to a given item in the database, it is assumed that the user has no access to that item. Thus, neglecting to specify access will not compromise the security of the database.

3.2.1. The ALLOW Command

The ALLOW command grants a user access to a specified set of data items in the database. The format of this command is:

```
ALLOW <Access_Type> <Attribute_List> ON <Relation>
[WHERE <Specifiers>]
TO <User_Id>
```

The possible values of Access_Type are: RETRIEVE, which covers SELECT, PROJECT, and two-relation queries; INSERT for the INSERT query; UPDATE for the UPDATE query; DELETE for the DELETE query; and AGGREGATE for the aggregate queries. Attribute_List is either a list of attributes belonging to Relation, or the keyword "ALL," which specifies all attributes belonging to Relation. User_Id is the integer identifier of the user who is being granted access to Relation. The user issuing the command must be the owner of the specified relation or the ALLOW is not performed (i.e., it is rejected).

The ALLOW command allows specification at the granularity of a relation, or as fine as the columns and rows of a relation. For example, suppose the following relation is stored in the database:

```
EMP (Name, Dept, Salary)
```

and the owner of the EMP relation issued:

```
ALLOW RETRIEVE ALL ON EMP
TO 35
```

This specifies that user #35 is granted access to perform any of the retrieve operations on the EMP relation. The user is allowed access to all columns and all rows of the relation. To specify that a user has access only to certain rows of the relation, a where clause may be used:

```
ALLOW RETRIEVE ALL ON EMP
WHERE (Salary < 40000)
TO 35
```

In this case, only those employees earning less than $40,000 may be retrieved (by user #35).

To specify that a user has access only to certain columns of the relation, a subset of the attributes of the relation may be specified:
```
ALLOW AGGREGATE (Salary) ON EMP
TO 35
```

This grants user #35 the ability to perform aggregate operations on the Salary field of EMP, but access to no other field is given.

RRDS security also provides context-dependent security [4]. It is possible to specify that columns of a relation may be accessed but not at the same time. For example, the commands:
```
ALLOW RETRIEVE (Name) ON EMP
TO 35
```
```
ALLOW RETRIEVE (Salary) ON EMP
TO 35
```

specify that user #35 may RETRIEVE (PROJECT) the Name and Salary columns of EMP, but not at the same time (i.e., user #35 should not be able to determine the salary of an employee). To allow attributes to be viewed together, they must appear together in the same ALLOW, as in:
```
ALLOW RETRIEVE (Name, Salary) ON EMP
TO 35
```

3.2.2. The DISPLAY ALLOW Command

To see the current status of authorization, a user may use the DISPLAY ALLOW command. This command shows the allows currently stored in RRDS. The format of the command is:

```
DISPLAY ALLOW
```

It is possible so that it will not become a bottleneck in the system. The broadcast bus is responsible for communication between the controller and the RCs. Each of the RCs has a portion of the database, determined by a partitioning strategy, stored in its dedicated disk drives. The RCs are responsible for the processing of the queries.

RRDS supports all of the operations that are available in the relational algebra, including one-relation operations and two-relation operations. In addition, the system supports several aggregate functions: count, sum, min, max, and average. The list of operations in RRDS are shown in Table 1.

Table 1: The RRDS Operations

<table>
<thead>
<tr>
<th>One-Relation Operations</th>
<th>Two-Relation Operations</th>
<th>Aggregate Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>Union</td>
<td>Max</td>
</tr>
<tr>
<td>Project</td>
<td>Difference</td>
<td>Min</td>
</tr>
<tr>
<td>Insert</td>
<td>Intersection</td>
<td>Sum</td>
</tr>
<tr>
<td>Delete</td>
<td>Join</td>
<td>Count</td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td>Average</td>
</tr>
<tr>
<td>Aggregate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.2. Implementation of the Authorization Component in RRDS

As described before, the authorization component of a security system allows the database administrator to control access to the items in the database. For RRDS, the items are relations and the columns and rows of those relations. RRDS provides four commands for controlling access to the relations. The ALLOW, DISPLAY ALLOW, REMOVE ALLOW, and DENY commands are provided for adding, displaying, and modifying the access rights of users. Each of these commands are described in this section.

Authorization commands are received from the host computer by the controller. Each command (as with any other message from the host) is associated with a user id number identifying the user issuing the command. The host is responsible for authenticating the user id, i.e., ensuring that the specified user id is correct. The controller parses the command and sends it to the RCs for processing. Upon completion of the request, each RC will send an appropriate message to the controller, which will pass the appropriate information to the host.
DISPLAY ALLOW
[ON <Relation>]
[FOR <User_Id>]

A specific relation and/or user may be specified, or the allows for all relations and/or all users may be requested. For example, the command

DISPLAY ALLOW ON EMP

will return all the allows pertaining to the EMP relation (assuming the user issuing the command is the owner of the EMP relation). The output might be:

4:
ALLOW DELETE (Name, Dept, Salary) ON EMP
TO 27
7:
ALLOW UPDATE (Name, Salary) ON EMP
WHERE ((Dept = Math) OR (Dept = Computer Science))
TO 3

Each relevant allow is displayed, along with an identifying index. The index can be used to identify an allow in a subsequent REMOVE ALLOW command (see the next section).

If a relation is not specified, the allows for all relations (the user executing the command is the owner of) are displayed. Note that a user may not necessarily be able to see the allows pertaining to himself (except when he is the owner of the relation, in which case he has complete access to the relation).

3.2.3. The REMOVE ALLOW Command

The REMOVE ALLOW command may be used to revoke a stored allow. This is a dependent form of revocation which the RRDS security mechanism provides. The format of the command is:

REMOVE ALLOW <index>

This command is to be used in conjunction with the DISPLAY ALLOW command, from which the index of the desired allow may be determined. For example,

REMOVE ALLOW 7

would remove the stored allow with index 7. The allow is removed only if the user issuing the command is the owner of the relation referenced by the allow.

3.2.4. The DISALLOW Command

The DISALLOW command is the independent form of revocation provided by the RRDS security mechanism. It gives the authority the ability to revoke access for a user by specifying the set of data that the user should not be allowed to access. The format of the command is similar to that of the ALLOW command:

DISALLOW <Access_Type> <Attribute_List> ON <Relation> [WHERE <Specifier>]
TO <User_Id>

Unlike the REMOVE ALLOW command, a specific stored allow is not specified for revocation. Instead, the relevant allows are modified such that they do not permit the access which is being revoked by the DISALLOW. Only the allows which specify the same user, relation, and access type as in the DISALLOW command are affected. The keyword ALL may be used in place of Attribute_List as in the ALLOW command.

For example, if the stored allows were as follows:

1:
ALLOW RETRIEVE (Name, Dept) ON EMP
WHERE (Name > Jones)
TO 35
2:
ALLOW UPDATE (Name, Dept, Salary) ON EMP
WHERE (Salary < 40000)
TO 35
3:
ALLOW AGGREGATE (Salary) ON EMP
TO 35

and suppose the owner of the EMP relation issued

DISALLOW RETRIEVE (Name, Dept) ON EMP
TO 35

This revokes any RETRIEVE access previously given for user #35 to the Name or Dept columns of the EMP relation. As a result, allow #1 is removed from the system. Allows #2 and #3 are unaffected.

If, instead, the command had been

DISALLOW RETRIEVE (Dept) ON EMP
TO 35

then user #35 would still continue to have the original access to the Name columns, but would lose access to the Dept column. Allow #1 would be replaced by

ALLOW RETRIEVE (Name) ON EMP
WHERE (Name > Jones)
TO 35

It is possible to revoke access to a subset of the rows of a relation with the DISALLOW command. For example,

DISALLOW RETRIEVE (Name, Dept) ON EMP
WHERE (Dept = Math)
TO 35

states that user #35 should not have access to the Name and Dept columns of EMP when the employee belongs to the math department. The result of this command is to replace allow #1 (in the original set of allows) with

ALLOW RETRIEVE (Name, Dept) ON EMP
WHERE (Name > Jones AND Dept <> Math)
TO 35

The where clause of the allow has been modified so that the user is not allowed to access the math department records.

In some cases, an existing allow is replaced by two new allows as a result of a DISALLOW command. For example,

DISALLOW RETRIEVE (Dept) ON EMP
WHERE (Salary > 50000)
TO 35

results in allow #1 (in the original set of allows) being replaced with

ALLOW RETRIEVE (Name) ON EMP
WHERE (Name > Jones)
TO 35
ALLOW RETRIEVE (Name, Dept) ON EMP
WHERE (Name > Jones AND Dept <> 50000)
TO 35

The first of these allows indicates that access to the Name column remains unaffected by the DISALLOW. The second of the allows indicates that when accessing the Dept column, the salary of the employee must be greater than $50,000, which is what the DISALLOW specifies. The Name attribute must be included in the attribute list of the second allow, otherwise access would be revoked which should not be (the user would no longer be able to access both Name and Dept in the same query).

In some cases, when a where clause of an allow is modified due to a DISALLOW command, some simplification is possible. For example, consider the following DISALLOW command:

DISALLOW UPDATE ALL ON EMP
WHERE (Salary < 50000)
TO 35

This causes allow #2 to be replaced with

ALLOW UPDATE (Name, Dept, Salary) ON EMP
WHERE (Salary < 40000 AND Salary >= 55000)
TO 35

This allow clearly allows nothing since the where clause can never be true. The where clause of a modified allow is automatically simplified before being stored by the system. In this case, the simplification results in an allow with a FALSE where clause (which provides no access) and therefore the allow doesn't need to be stored. Thus, the DISALLOW has the effect of eliminating allow #2 from the system.
Simplification is discussed in more detail in the next section.

3.2.4.1. Simplification of a Where Clause

When the where clause in an existing allow is modified as a result of a DISALLOW command, it is simplified by the system, if possible. This is done for two reasons. First, the effect of the new allow becomes more clear to an authorizer when viewing the allow stored in RRDS (with a DISPLAY ALLOW command). Secondly, it improves system efficiency in the execution of user queries after query modification is performed.

In RRDS, where clauses are specified in disjunctive normal form, i.e., a where clause condition is a disjunction of the form:

\[ C_1 \text{ OR } C_2 \text{ OR } C_3 \text{ OR } ... \]

where \( C_i \) is a conjunction of the form:

\[ P_1 \text{ AND } P_2 \text{ AND } P_3 \text{ AND } ... \]

where \( P_i \) is a predicate such as \( \text{Salary} < 5000 \). Thus, a where clause is of the form:

\[ (P \text{ AND } P \text{ AND } ...) \text{ OR } (P \text{ AND } P \text{ AND } ...) \text{ OR } ... \]

There are three types of simplification performed by the security system. First, a conjunction is removed from a where clause if any two predicates in the conjunction are contradictory (this is referred to as a “conflict”). Second, a predicate in a conjunction may be redundant, in which case it is removed from the conjunction (this is referred to as a “reduction”). Last, two predicates in a conjunction may be equivalent to a third predicate, in which case the third predicate replaces the original two predicates in the conjunction (this is referred to as a “substitution”).

Simplification is performed by comparing pairs of predicates within each conjunction of a where clause. If a conflict is detected, or a reduction or substitution is possible, the proper action is taken. Each of these forms of simplification is now discussed in turn.

A conflict occurs if two predicates in a conjunction contradict each other. Consider the conjunction

\[ \text{Name} = \text{Jones AND Dept = Math AND Name < Jones} \]

This conjunction can never be evaluated as TRUE, since the first and third predicates contradict one another. Thus, this conjunction may be removed from a where clause without affecting its truth value. If the conjunction removed from the where clause is the only conjunction in the where clause, the entire where clause becomes FALSE.

A reduction can be performed when a predicate is redundant in a conjunction. Consider the conjunction

\[ \text{Salary} > 20000 \text{ AND Salary} >= 30000 \]

The first predicate in this conjunction is redundant, since removing it does not affect the truth value of the conjunction. Any value of Salary which satisfies the second predicate will satisfy the first predicate. That is, the second predicate implies the first predicate. Thus, the first predicate can be removed from the conjunction, resulting in

\[ \text{Salary} >= 30000 \]

A substitution can be performed when two predicates in a conjunction are equivalent to a third predicate. Consider the conjunction

\[ \text{Salary} >= 150000 \text{ AND Salary} <= 150000 \]

The two predicates in this conjunction can be replaced by

\[ \text{Salary} = 150000 \]

without affecting the truth value of the conjunction.

There are various situations indicating a conflict, or that a reduction or substitution is possible in a conjunction. Given two predicates of the form:

\[ \text{Predicate}_1: \text{Attribute}_1 \text{ OP}_1 v_1 \]
\[ \text{Predicate}_2: \text{Attribute}_2 \text{ OP}_2 v_2 \]

in the same conjunction (\( \text{OP}_1 \) and \( \text{OP}_2 \) are relational operators; \( v_1 \) and \( v_2 \) are constant values; and \( \text{Attribute}_1 \) is the same as \( \text{Attribute}_2 \)), Table 2 summarizes the possible simplifications.

3.2.4.2. Details of the DISALLOW Command

When an allow is affected by a disallow (when it is for the same user, relation, and access type as the disallow), it is replaced by up to two new allow specifications. One new allow is created to ensure that the user retains access to the columns and rows of the relation which are unaffected by the disallow. This allow is created with the same attribute list and where clause as the original allow, but with the attributes specified in the disallow attribute list removed from the attribute list of the new allow. If the attribute list of this new allow is empty (i.e., all attributes of the original allow are affected by the disallow), the new allow is not stored.

A second allow is created to reflect the new access to the columns and rows of the relation, i.e., the original access minus the access revoked by the disallow. This new allow primarily reflects access to the columns which are affected by the disallow. The where clause of this allow is set to be the conjunction of the where clause of the original allow with the negation of the disallow where clause (an allow or disallow having no where clause is considered to have a TRUE where clause). The implementation of this is complicated by the fact that an RRDS where clause must be in disjunctive normal form, thus the new where clause must be converted back into disjunctive normal form. The resulting where clause is simplified, and if it simplifies to FALSE the new allow is not stored. The attribute list of this allow is set to the entire attribute list of the original allow. This is done because otherwise the user's ability to access columns not affected by the disallow, together with columns affected by the disallow (but still accessible) would be lost.

Since each of these new allows may or may not be stored, zero, one, or two allows may replace the original allow, following a disallow. Let us illustrate this process with an example. Suppose the following allow is currently stored:

\[ \text{ALLOW RETRIEVE (Name, Salary, Dept) ON EMP} \]
\[ \text{WHERE} \]
\[ \text{Name} = \text{Jones AND Dept = Math; OR} \]
\[ \text{Name} = \text{Smith} \text{ OR} \]
\[ \text{Salary} > 20000 \text{ AND Salary} < 50000 \]
\[ \text{TO 35} \]

and the following disallow is performed:

\[ \text{DISALLOW RETRIEVE (Salary) ON EMP} \]
\[ \text{WHERE} \]
\[ \text{Name} = \text{Smith} \text{ OR} \]
\[ \text{Salary} > 35000 \]
\[ \text{TO 35} \]

The disallow causes the original allow to be replaced by two new allows. First, an allow is created for the columns in the original allow unaffected by the disallow:

\[ \text{ALLOW RETRIEVE (Name, Dept) ON EMP} \]
\[ \text{WHERE} \]
\[ \text{Name} = \text{Jones AND Dept = Math; OR} \]
\[ \text{Name} = \text{Smith} \text{ OR} \]
\[ \text{Salary} > 20000 \text{ AND Salary} < 50000 \]
\[ \text{TO 35} \]

This allow indicates that the access to the Name and Dept columns of EMP is unchanged. Since the attribute list of this new allow is not empty (there are some columns unaffected by the disallow), the allow is stored.

Second, an allow is created for the columns in the original allow which are affected by the disallow. At first, the where clause (of this new allow) is set to be the conjunction of the where clause of the original allow with the negation of the disallow where clause:

\[ \text{ALLOW RETRIEVE (Name, Salary, Dept) ON EMP} \]
\[ \text{WHERE} \]
\[ ((\text{Name} = \text{Jones AND Dept = Math}) \text{ OR} \]
\[ \text{Name} = \text{Smith} \text{ OR} \]
\[ \text{Salary} > 20000 \text{ AND Salary} < 50000)) \]
\[ \text{AND NOT (Name = Smith) OR (Salary > 35000))} \]
\[ \text{TO 35} \]
Table 2: The Simplification for the AND Logical Operator

<table>
<thead>
<tr>
<th>OP₁</th>
<th>=</th>
<th>&lt;&gt;</th>
<th>&gt;</th>
<th>&gt;=</th>
<th>&lt;</th>
<th>&lt;=</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>v₁ &lt; v₂ : C</td>
<td>v₁ &gt; v₂ : C</td>
<td>v₁ = v₂ : R (*)</td>
<td>v₁ = v₂ : R (P₁)</td>
<td>v₁ = v₂ : R (P₂)</td>
<td>v₁ = v₂ : R (P₃)</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>v₁ &lt; v₂ : R (P₁)</td>
<td>v₁ &gt; v₂ : N</td>
<td>v₁ &lt; v₂ : N</td>
<td>v₁ &lt; v₂ : R (P₁)</td>
<td>v₁ &lt; v₂ : R (P₂)</td>
<td>v₁ &lt; v₂ : R (P₃)</td>
</tr>
<tr>
<td>&gt;</td>
<td>v₁ &lt; v₂ : R (P₁)</td>
<td>v₁ = v₂ : C</td>
<td>v₁ = v₂ : R (P₁)</td>
<td>v₁ = v₂ : C</td>
<td>v₁ = v₂ : C</td>
<td>v₁ = v₂ : C</td>
</tr>
<tr>
<td>&gt;=</td>
<td>v₁ &lt; v₂ : R (P₁)</td>
<td>v₁ = v₂ : C</td>
<td>v₁ = v₂ : R (P₁)</td>
<td>v₁ = v₂ : C</td>
<td>v₁ = v₂ : C</td>
<td>v₁ = v₂ : C</td>
</tr>
<tr>
<td>&lt;</td>
<td>v₁ &lt; v₂ : C</td>
<td>v₁ &lt; v₂ : R (P₁)</td>
<td>v₁ &lt; v₂ : R (P₂)</td>
<td>v₁ &lt; v₂ : N</td>
<td>v₁ &lt; v₂ : N</td>
<td>v₁ &lt; v₂ : N</td>
</tr>
<tr>
<td>&lt;=</td>
<td>v₁ &lt; v₂ : C</td>
<td>v₁ &lt; v₂ : R (P₁)</td>
<td>v₁ &lt; v₂ : R (P₂)</td>
<td>v₁ &lt; v₂ : N</td>
<td>v₁ &lt; v₂ : N</td>
<td>v₁ &lt; v₂ : N</td>
</tr>
</tbody>
</table>

Notation used in the table:

- C: Conflict
- N: None
- R: Reduce
- S: Substitution

We first apply the negation in the where clause to get:

WHERE
(Name = Jones AND Dept = Math) OR
(Name = Smith) OR
(Salary > 20000 AND Salary < 50000) AND
(Name <= Smith AND Salary <= 35000)

After converting back to disjunctive normal form (by distributing the last expression into the three conjunctions), the where clause becomes:

WHERE
(Name = Jones AND Dept = Math AND Name <= Smith AND Salary <= 35000) OR
(Name = Smith AND Name <= Smith AND Salary <= 35000) OR
(Salary > 20000 AND Salary < 50000 AND Name <= Smith AND Salary <= 35000)

The simplification of the where clause produces:

WHERE
(Name = Jones AND Dept = Math AND Salary <= 35000) OR
(Salary > 20000 AND Name <= Smith AND Salary <= 35000)

The where clause did not simplify to FALSE, so the resultant allow is stored.

In certain cases, the processing of an original allow (with respect to a disallow) can result in two new allows where the first new allow is covered by the second new allow. This means that the first new allow is redundant. For example, if the original allow is:

ALLOW RETRIEVE (Name, Salary, Dept) ON EMP WHERE (Name = Jones)
TO 35

and the following disallow is performed:

DISALLOW RETRIEVE (Salary) ON EMP WHERE (Name = Smith)
TO 35

This produces:

ALLOW RETRIEVE (Name, Dept) ON EMP WHERE (Name = Jones)
TO 35

and

ALLOW RETRIEVE (Name, Salary, Dept) ON EMP WHERE (Name = Jones AND Name <= Smith)
TO 35

The second of these simplifies to

ALLOW RETRIEVE (Name, Salary, Dept) ON EMP WHERE (Name = Jones)
TO 35

which makes the first new allow redundant. Currently, RRDS security performs no checks for redundancy when storing an allow, so both allows would be stored. It should, however, be noted that this redundancy resulted from disallowing an access that was not previously allowed. Thus, such redundancies may not occur very frequently.

This section has described the RRDS security mechanism which provides a form of independent revocation. The application of independent revocation to other database systems and security implementations are discussed in the next section.
4. INDEPENDENT REVOCATION IN OTHER SYSTEMS

In the previous section we discussed a particular implementation of independent revocation in a DBMS. In this section we discuss the applicability of independent revocation to database management systems in general. Other systems might implement authorization in a manner substantially different from that of RRDS. Two factors which may affect how independent revocation is applicable to a given system are: first, the representation of the access matrix and, second, the specification of the condition within the extended access matrix.

4.1. The Representation of the Access Matrix

The representation of the access matrix can vastly affect the implementation of independent revocation. A system which represents the access matrix in the form of access rules can implement independent revocation in a manner analogous to RRDS. For this reason, INGRES [5, 6] is a system which could provide independent revocation. Other similar methods, such as access lists and capability lists, have been suggested as methods for implementing the access matrix [3]. These methods allow implementation of independent revocation in a manner essentially equivalent to a system with access rules.

An example of a system which implements the access matrix in a different manner is that of System R [9]. System R is a relational system which implements authorization in terms of views. A view can be defined as a subset of the columns or rows of a relation, or even data from more than one relation. After views have been defined, a user may be granted access to a view and perhaps subsequently have that access revoked. A user either has complete access to a view or does not have any access to the view, and may have access to more than one view.

Independent revocation in System R would first require the system to check which views are affected by the revocation. The affected views would then be copied and modified to reflect the revocation. This modification would be done in a manner similar to the modification of an access rule in RRDS. The user would then lose access to the original view and gain access to the newly created view. A new view is necessary due to the fact that other users, not affected by the revocation, may have access to the original view.

Other systems might implement the access matrix in other ways. For example, independent revocation is not possible in a system where a user has access to a set of data in the database based upon a password. In such a system, revocation would be performed by changing the password, at which time any user not given the new password would effectively have had his or her access revoked. Independent revocation (for that user) of a specified user is not possible, since it is not even possible to determine which users have access to what data.

4.2. The Specification of the Where Clause Condition

The second factor affecting the implementation of independent revocation is how the condition within the extended access matrix is specified. This factor affects the simplification process required after modifying the condition. Some systems might not even allow such a condition to be specified, in which case simplification is not necessary. RRDS allows the condition in a specific format, that of disjunctive normal form. This allows the system to simplify the condition without a great deal of difficulty.

A similar, and as commonly-used, form to that of disjunctive normal form (DNF) is conjunctive normal form (CNF). The format for a where clause condition in DNF has been described in Section 3. A condition in CNF is a conjunction of the form:

\[ D_1 \text{ AND } D_2 \text{ AND } D_3 \text{ AND } \ldots \]

where \( D_i \) is a disjunction of the form:

\[ P_1 \text{ OR } P_2 \text{ OR } P_3 \text{ OR } \ldots \]

where \( P_i \) is a predicate. Thus, a condition is of the form:

\[ (P \text{ OR } P \text{ OR } \ldots) \text{ AND } (P \text{ OR } P \text{ OR } \ldots) \text{ AND } \ldots \]

It should be noted that any arbitrary condition can be expressed in DNF and CNF. Furthermore, if a condition is expressed in non-DNF and non-CNF format, it can be translated to DNF and CNF format.

As with disjunctive normal form, CNF conditions can be simplified in a straightforward manner. Table 3 summarizes the possible actions when simplifying a disjunction within a where clause condition in CNF (analogous to Table 2 for disjunctive normal form). In this case, one new form of simplification is seen, that which exists for tautology. This occurs when a disjunction is constantly true. In a where clause condition in CNF, a constantly true disjunction can be removed from the expression without affecting its validity. In the special case that it is the last disjunction, the expression becomes true.

Other systems might allow the condition to be specified in a more general fashion. For example, a system such as INGRES allows the condition to be specified in unrestricted form. Such a system requires a more general simplification algorithm. It might be necessary to limit the types of simplification performed for efficiency reasons. Alternatively, simplification could be eliminated altogether, but would result in a condition which is harder to read and understand by an authorizer, and would result in less efficiency in storage of the access matrix and processing of user queries.

5. SUMMARY AND CONCLUSIONS

In this paper we have presented the concept of independent revocation. We described the advantages of allowing an authorizer to specify revocation independent of the current status of authorization. Some of the aspects relating to the implementation of a system providing independent revocation were discussed.

First, we discussed revocation in terms of formal models of authorization. The concept of an access matrix was introduced, and extended to allow for the specification of a condition for database systems. Then we considered the general idea of independent revocation in terms of this extended access matrix.

Second, we presented an actual implementation of a system which provides independent revocation. The system, RRDS, provides a DISALLOW command which gives the authorizer the capability to specify a data a user should not be allowed to access. The major aspects of the implementation of the DISALLOW command were discussed, including the simplification of the where clause of the access rules following their modification.

Last, we explored the applicability of independent revocation to database systems in general. We discussed the two major factors affecting the implementation. We concluded that independent revocation is applicable to a variety of systems, including some major systems currently in existence.

REFERENCES


Table 3: The Simplification for the OR Logical Operator

<table>
<thead>
<tr>
<th>OP1</th>
<th>T</th>
<th>N</th>
<th>R</th>
<th>S</th>
<th>R</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>= v1 &lt; v2 : N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 &lt; v2 : R (*)</td>
<td></td>
</tr>
<tr>
<td>v1 = v2 : R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 = v2 : R (*)</td>
<td></td>
</tr>
<tr>
<td>v1 &gt; v2 : N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 &gt; v2 : R (P1)</td>
<td></td>
</tr>
<tr>
<td>v1 &gt; v2 : T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 &gt; v2 : T</td>
<td></td>
</tr>
<tr>
<td>v1 = v2 : S (\Rightarrow)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 = v2 : R (P1)</td>
<td></td>
</tr>
<tr>
<td>v1 &gt; v2 : R (P2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 &gt; v2 : R (P2)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; v1 &lt; v2 : R (P2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 &lt; v2 : R (P2)</td>
<td></td>
</tr>
<tr>
<td>v1 = v2 : R (P2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 = v2 : R (P2)</td>
<td></td>
</tr>
<tr>
<td>v1 &gt; v2 : R (P1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 &gt; v2 : R (P1)</td>
<td></td>
</tr>
<tr>
<td>v1 &gt; v2 : T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 &gt; v2 : T</td>
<td></td>
</tr>
<tr>
<td>v1 = v2 : R (P1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 = v2 : R (P1)</td>
<td></td>
</tr>
<tr>
<td>v1 &gt; v2 : R (P2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 &gt; v2 : R (P2)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; v1 &lt; v2 : N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 &lt; v2 : N</td>
<td></td>
</tr>
<tr>
<td>v1 = v2 : S (\Rightarrow)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 = v2 : S (\Rightarrow)</td>
<td></td>
</tr>
<tr>
<td>v1 &gt; v2 : T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 &gt; v2 : T</td>
<td></td>
</tr>
<tr>
<td>v1 = v2 : T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 = v2 : T</td>
<td></td>
</tr>
<tr>
<td>v1 &gt; v2 : T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v1 &gt; v2 : T</td>
<td></td>
</tr>
</tbody>
</table>

Notation used in the table:

- **T**: True (Tautology always true)
- **N**: None (No simplification is possible)
- **R**: Reduce (Remove the predicate indicated in parentheses; (*) indicates either predicate may be removed)
- **S**: Substitution (Remove either predicate and change the relational operator in the remaining predicate to the operator indicated in parentheses)

---


