CHAPTER 4

ACCESS CONTROL IN A DHDBMS

"Within a short span it is likely that individuals will be able to connect their own private computers with generally available networks in order to consult and communicate with large databases. The ability for government agencies (income tax, social securities, etc.), banks, credit card companies and health care organizations to connect their systems and thereby share their data, poses a serious threat to individual privacy.

R.P. Van de Riet and M.L. Kersten
A Module Definition Facility for Access Control in Communicating Data Base Systems [VAN82].

"Who sees the variety and not the unity, wanders on from death to death."

Katha Upanishad

A paper based on this chapter and earlier chapters, entitled "Access Control in Distributed Heterogeneous Database Management Systems" has been communicated for the publication in the International Journal, Computers and Security [GOY91].
4.1 Introduction

In this chapter, we will discuss and propose an access control mechanism for a general DHDBMS. This access control mechanism is based on a generalized DHDBMS architecture discussed in chapter 2. First of all, let us discuss a basic model for database access control.

4.2 Basic Access Control Model

Models of database access control have grown out of earlier work on protection in operating systems. One of the most influential protection models was developed by Lampson [LAM74]. The basis of this model is the access rule, which specifies the types of access a subject can have for an object. In the context of operating systems, objects are entities, known to the operating system, to which access must be controlled, such as main memory pages, programs, auxiliary storage devices, and files. Subjects are the entities that request access to objects, usually a process-domain pair, a process being a program in execution, and the domain the environment in which the process is executing. Access types might be EXECUTE, ALLOCATE, READ. The set of all access rules can conveniently be thought of as forming an access matrix \( A \), in which columns \( o_1, o_2, \ldots, o_n \) represent objects, and rows \( s_1, s_2, \ldots, s_m \) represent subjects. The entry \( A[s_i, o_j] \) contains a list of access types \( t_1, t_2, \ldots \), which specifies the access privileges held by subject \( s_i \) for object \( o_j \). The objects accessible by
a subject, together with the mode of access, are sometimes termed the 'capabilities' of the subject. This model assumes that all attempted accesses to an object are intercepted and checked by a controlling process.

For controlling database access, [FER80] suggested similar concepts of access rules and access matrix. However, [FER80] defined objects as a set of data item occurrences, subjects as a set of potential users and access types as a set of operations such as READ, UPDATE, ADD and DELETE (the particular set depends on the data model).

It is important to note that the database access matrix is more static than the operating system protection matrix. It is modified explicitly only by an authorizer who wishes to specify a new access rule or revoke an old one. Figure 4.1 shows part of an access matrix that represents the rules governing access to the EMPLOYEE relation. The attributes of the EMPLOYEE relation are EMP-NAME, PERS-NO, ADDRESS, TEL-NO, and SALARY. From the figure we see that the personnel manager has unrestricted access (indicated by the ALL entry) to all attributes, while the administration clerk has READ access to all attributes except SALARY. A null entry implies that no access is allowed to that object. This access matrix is capable of modelling name-dependent (content-independent) access control down to any level of granularity supported by the DBMS. To represent access rules that are content-dependent, [FER80] extended the model
<table>
<thead>
<tr>
<th>OBJECT</th>
<th>EMP-NAME</th>
<th>PERS.NO</th>
<th>ADDRESS</th>
<th>TEL-NO</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBJECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PERSONNEL</th>
<th>ALL</th>
<th>ALL</th>
<th>ALL</th>
<th>ALL</th>
<th>ALL</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MANAGER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADMIN-CLERK</td>
<td>READ</td>
<td>READ</td>
<td>READ</td>
<td>READ</td>
<td>READ</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.1: Access Matrix
so that the access rule contains a predicate, p. The predicate, p may place some constraints on the access rule (such as allowing access only if a set of conditions are true).

Therefore, the basic data model proposed by [FER80] for database security represents access control information by a tuple with four major components (S, O, T, P). S is set of users (subjects), O is a set of database objects, T is a set of access types, and P is a set of binary predicates. A tuple (s, o, t, p) in (S, O, T, P) says that a user s is eligible to do operations of type t to the database object o if the predicate p is true. This model is general enough to represent the security requirements for most DBMSs. Moreover, this model includes content-dependent security policies. Figure 4.2, shows a simple example of an access rule that gives the payroll clerk READ access to the EMPLOYEE relation for those employees who earn less than Rs. 40000.

4.3 Extensions to the Basic Access Control Model

By defining the access rule (s, o, t, p) some important requirements of authorization have been left unexpressed. [HSI79] proposed a model for database protection languages. Based on this model, [FER80] extended the basic access control model by introducing two more components of the access rules. These rules are used for
<table>
<thead>
<tr>
<th>USER</th>
<th>OBJECT</th>
<th>ACCESS-TYPE</th>
<th>PREDICATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>o</td>
<td>t</td>
<td>p</td>
</tr>
</tbody>
</table>

PAYROLL EMPLOYEE READ  SALARY<40000
CLERK

Figure 4.2: An Access Rule
validating authorization and requests.

One requirement is for control over the set of access rules. The basic model does not allow for some important policies about who may write access rules. One such policy permits only the authorizer who wrote the rule to change it. For this purpose, the access rule specifies the authorizer, \( a \), so that the rule becomes \( (a, s, o, t, p) \). The model must also cover important policies for delegation of rights. By a right we mean a certain kind of access to an object; a right is the \( (o, t, p) \) of the access rule. A subject \( s_1 \) who holds the right \( (o, t, p) \) may be allowed to delegate that right to another subject \( s_2 \); this delegation is equivalent to inserting a new access rule \( (s_1, s_2, o_1, t_1, p_1) \). Since a portion of the rule is "copied", a copy flag, \( f \), is added to the access rule specifying whether \( s_2 \) is (in turn) allowed to delegate the access right. So the extended access rule is \( (a, s, o, t, f, p) \).

For describing access control mechanism in a DHDBMS, we will assume that the facility to delegate access rights recursively is not available. We will assume that access rights are granted only by the DBA to the users. This is done purely to keep the discussions simple. However, the access control mechanism can be similarly extended with this facility. Therefore the access rule is represented as \( (a, s, o, t, p) \).
Also, because our access control mechanism is based on view mechanism, and as a view can be defined by using an arbitrary query (which itself can contain predicate part using 'WHERE........'), there is no requirement for a predicate in the access rule. Therefore, we will represent the access rule by a tuple \((a, s, o, t)\). Here \(o\) represents a data object or view. Figure 4.3, shows a simple example of access rule that gives user B, a READ access to the view V. It also indicates that user A (DBA) is authorizer of this access rule.

### 4.4 Views as Protection Objects

There are several advantages in using views for access control. Defining a view to represent the subset of a database which a user is authorized to access is conceptually easy to understand. They provide a convenient way to express and evaluate the predicates needed to specify content-dependent policies.

In general, a view in a DHDBMS is application-oriented. That is, database view is created for a particular application and authorized for use by those users who need to run that application. A view is a virtual object. We will use it for content-dependent access control in a DHDBMS. A view is created over the global conceptual schema (using global data objects). Depending on the system, the syntactic specification of a view may be defined.
Figure 4.3: An Access Rule for DHDBMS

<table>
<thead>
<tr>
<th>AUTHORIZER</th>
<th>USER</th>
<th>OBJECT</th>
<th>ACCESS-TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>s</td>
<td>o</td>
<td>t</td>
</tr>
</tbody>
</table>

User A (DBA) User B V READ
in various ways [CLA85]. For example, a view may be defined with the following construct [CHA75], [AST76], [WIL81]:

\[
\text{DEFINE VIEW VIEW-NAME<TARGET-LIST> WHERE Qualification-clauses.}
\]

VIEW-NAME is the name of the view. TARGET-LIST denotes the subset of attributes from the global database to be included in the view, and qualification-clauses identifies which elements on what condition are to be included in the view. In other words, qualification-clauses is a boolean predicate. A view is identified by the VIEW-NAME. A user must be granted the right to use a view before he can do so. Only an authorized user of a global database object can define a view on that object (which may be another view). The definer of the view (the view owner) can grant access to the view to other users.

As discussed in previous section, we assume that access rights are not granted recursively and therefore access rights are granted only by DBA. We can also assume that all the access privileges which are permitted on global data objects are available to the global DBA, because it is he who will either access these database objects himself or will give access rights on these objects to other users. Also, as the global conceptual schema is formed by integrating local conceptual schema, the global DBA will have corresponding access privileges on local data objects.
in component DBMSs. The same thing can be explained in a different way. Let us assume that a global DBA has access privileges on local data objects in component DBMSs and therefore he defines on a global conceptual schema by integrating local conceptual schemas of these component DBMSs. So this global DBA will have all the access privileges on the corresponding global data objects of this global conceptual schema.

4.5 View Authorization

As discussed above, the global DBA may create a view at the global level (over global conceptual schema) and he is fully and solely authorized to perform actions upon it. He may share this view with an user by issuing a command similar to GRANT command [GRI76] to give various access privileges on that view to this user. An access rule is entered into the Global Security Catalogue to record the fact that a particular user is authorized to use the view. This approach is similar to the approach used for view authorization in System R [AST79].

Similarly, the DBA who has granted an access privilege may subsequently withdraw it, by issuing a command similar to REVOKE command [GRI76]. By this action, the access rule corresponding to this revocation is deleted from the Global Security Catalogue. Subsequently, the revoked privilege on the named view is denied to the revokee.
4.6 View Replacement

We will assume that a global catalogue contains all the information about each view; namely, the view-name, target-list, qualification-clauses and global-data-object-name over which the view is defined. At the time of query processing, the view name is replaced by the global data object name and qualification clauses of the view materialization process then proceeds as part of normal query processing.

Figure 4.4 briefly describes the view materialization algorithm as used in System R [GRI76], [AST79]. The algorithm can be applied in the Global Functional System at a particular site where a query has been submitted for processing. As discussed in Section 2.6.1, this operation modifies the global query into an intermediate query based on the global conceptual schema. This intermediate query is broken into a set of subqueries to be processed at the various sites of the system.

4.7 Proposed Access Control Mechanism

It is necessary to provide content-dependent access control in the global model since some of the local systems are not capable of enforcing such policies, and content-dependent policies may depend on data at multiple sites. Similar reason is there for functional access
Step 1: Check the query against the view table in the Global catalogue to see if any views are referenced.

Step 2: Retrieve any required view definitions from the view table.

Step 3: Determine from each view definition the name of the global data object in the Global Functional System over which the view is defined.

Step 4: Replace each view name in the query by its global data object name.

Step 5: Add the qualification clauses from the view definitions to the qualification clauses of the query.

Figure 4.4: View Materialization Algorithm
control and therefore this control is also provided in the global model. First we will explain content-dependent access control mechanism and later on we will extend this mechanism to cover functional access control.

4.7.1 Content-dependent Access Control Mechanism

We will use the traditional method of view mechanism to provide content-dependent access control in DHDBMS. We now present the details of the content-dependent access control mechanism. To fully understand it, it is necessary to discuss view creation, granting of views, and processing of views in a query.

4.7.1.1 View Creation

When a view V is created in the Global Functional System, the following operations are necessary:

1. Save the view definition in a global catalogue

2. Insert an access rule in the Global Security Catalogue for the user A (DBA) who defines the view giving him full privileges for the view.

4.7.1.2 View Granting

When a view is granted to a user (say B), it is necessary that an access rule (A, B, V, t) is entered into
the Global Security Catalogue in the Global Functional System to record the fact that user B is now authorized to have access type t on the view V. It also records the information that A is the authorizer of this grant.

4.7.1.3 View Processing

When user B issues a query which references view V, then first of all access rule checking is performed to see whether user B is authorized to refer view V. If access rule checking fails, then query processing will not continue. Otherwise the first five steps of the algorithm in Figure 4.4 for the Global Functional System are performed as it is. This operation gives a modified query, which is referred as an intermediate query. This intermediate query is then parsed, and decomposed into a set of subqueries to be processed at the various sites of the system. An execution plan is generated to process these subqueries and these subqueries are distributed to the designated sites with the help of the Communication Network System. These subqueries are distributed to various sites after replacing the User Identification Number and other identification information of user B by the User Identification Number and corresponding information of user A (i.e. global DBA) in the subqueries. (It means that user B impersonates as global DBA. This is done because component DBMSs recognize the global DBA as an authorized user of the local data objects and they do not recognize global user B as user B is not an
authorized user of these objects as per local access control mechanism).

The Local Database Interface accepts a subquery (corresponding to that site) from the Communication Network System and creates a transaction or site query for the Local Functional System to process it. The Local Functional system performs the required operations to process this transaction or site query and returns the result back to the Local Database Interface. The Local Database Interface passes the result to the Communication Network System for transmission to the destination site and kills the user process it created. The Global Functional System at the destination site integrates the results received from various sites and passes the final result to user B.

It should be noted that if User Identification Number etc. of user B is replaced by corresponding information of the global DBA, then the subqueries will be able to access the component databases. In this way, the inconsistency problem (discussed in section 1.15) in access control between global and local data models is overcome.

We have discussed in Section 1.14.4, that content-dependent access control policies base a decision of whether to allow access or not on the values of data in the database. For example, a manager may be allowed to see an employee's personnel record only if he is the Manager of
that employee. In this case some data will have to be retrieved to determine whether the user is authorized to access the required data. Also, the data required to make an access decision may not reside in the same local database from which actual data is to be accessed. We have discussed this example in section 1.15 and posed an access control problem for such type of cases. This problem can be solved by using above access control mechanism. To demonstrate it, let us assume an employee record relation E at site X as

\[
E \left( \text{EMP-NO, EMP-NAME, AGE, DEPTT, SALARY} \right)
\]

and Employee-Manager relation EM at site Y as

\[
EM \left( \text{EMP-NO, EMP-NAME, MGR-NAME} \right)
\]

Now, to provide access to the employees' personnel records only to its manager, a view can be defined as follows.

\[
\text{DEFINE VIEW EMP-SAL} \left( E.\text{EMP-NAME, E.SALARY} \right) \\
\quad \text{WHERE EM.MGR-NAME} = \text{USERNAME} \\
\quad \text{AND EM.EMP-NO} = E.\text{EMP-NO}
\]

and then a query

\[
\text{RETRIEVE} \left( \text{EMP-SAL.ALL} \right)
\]

will give salary of only those employees for whom this user is the manager. The value of USERNAME is the DHDBMS-known name for the user executing the statement that invokes the
USERNAME reference. For example, if user Sharma issues the statement

RETRIEVE (EMP-SAL.ALL)

then the DHDBMS will effectively convert the statement into

RETRIEVE EMP.SAL(E.EMP-NO, E.EMP-NAME, E.SALARY)

WHERE EM.MGR-NAME = "Sharma"
AND EM.EMP-NO = E.EMP-NO

This query will be processed as discussed earlier. We have already discussed that subqueries will be entered in component DBMSs with user identification of DBA, so there will be no access control problem. A subquery will go to site Y and access all EM.EMP-NOs for whom Sharma is the Manager. Then another subquery will go to site X and will access all relevant information of these employees. In this way the access control problem discussed in section 1.15 can be overcome. It should be noted that for simplicity, we have assumed relational databases at site X and Y. We have also used the local data object names to define the view. However, in real case, a global conceptual schema has to be defined by integrating local conceptual schemas of DBMSs at sites X and Y and a view will have to be defined on the conceptual schema. It would require mappings. Also, the database at sites X and Y may be based on different data models.
4.7.2 Functional Access Control

We have discussed functional access control policy in Section 1.14.6. As per this policy, a user is allowed to get only summary and he is not allowed to access underlying detailed data. We have also posed a problem in Section 1.15, that without accessing underlying data, how can a user get a summary. If a view can be defined to calculate summary then by using proposed access control mechanism, this problem can be overcome. Because as per this access control mechanism, the global DBA will access underlying data, calculate summary as per view definition and finally give summary to the user. Although, many DBMSs support the facility to define views for statistical information like average, variance etc., it may be difficult to define a view for calculating summaries in general. The main reason for this may be the requirement of many steps to calculate a summary. It may be difficult to put so many steps in a view definition. Therefore, we propose the following approach.

4.7.2.1 Authorization of Application Programs

In addition to being authorized to perform operations on data objects, a user can be authorized to run application programs on the system. An application program contains statements in some programming language plus database requests. The execution of an application program makes zero or more requests to the database. One can also conceive of application programs as those which are
installed in the DHDBMS and then run repeatedly, possibly by many users. Such programs can be written by the global DBA, and invoked by users. In such cases, the global DBA is authorized to perform different operations (like edit, read, delete, run etc.) on the programs he writes, but the users are authorized to run those programs.

By using this approach, a program can be written by the global DBA to find summary etc. Then a grant to run this program can be given to a user by the global DBA. In this situation, user will only be able to execute this program and get the end result. As the user is not authorized to do any other operation on this program, he cannot get anything other than the end result. In this way, a user can get a summary without accessing underlying data.

4.7.2.2 Application Program Authorization

An application program, undergoes two stages; installation and invocation. These stages may be under the control of different users, with different authorization privileges. The installation and invocation procedures of application programs are discussed below.

The application programmer defines and installs programs. Installing an application program consists of an application programmer entering it into the catalogues, storing the program in a safe place, and authorizing one or
more users to use it. The installation component of the system checks the installer's authorization against the set of DHDBMS requests in the program. Installation is successful if and only if the Installer is authorized to perform each request. A successful installation results in the installer obtaining the RUN privilege on the application program. For our discussions, we assume that installer is no one else but the global DBA.

4.7.2.3 The RUN privilege

One reason for defining application programs is to encapsulate objects so that other may use them without violating the integrity of the constituent objects. In just the same way, we used views to authorize (content-dependent) portions of data without fully authorizing use of underlying data objects. The application program is a mechanism for authorizing certain specific sequences of actions without authorizing excercise of the constituent actions themselves. Installed application programs become authorized objects in the database, and as we saw above, the creator or the installer of an object becomes fully and solely authorized to perform all actions on the object. In the case of an application program, the only privilege on application program is RUN, the ability to invoke the program.

After installation, the installer may grant the RUN privilege on the program if and only if he is authorized
to grant all the privileges required to perform the DHDBMS requests contained in the program. Revocation of the RUN privilege on an application program proceeds as for revocation of an access privilege on a data object or view.

When an application program is installed, a record is kept of the data objects it referenced, and the authorization of the installer on those data objects. If any of these data objects are dropped, or privileges on those tables are revoked from the installer of the program, then the program is invalidated. This invalidation also causes all the RUN privileges on that application program to be revoked.

It should be noted here that the access rule \((a, s, o, t)\) defined earlier for database objects and views can be used for application programs also. Now \(o\) represents a database object or a view or an application program and access type for an application program is RUN. Figure 4.5 shows three examples of access rules which give access rights of READ, RUN and READ to users B, C and D respectively on View V, Program P and Data Object K.

4.8 Summary

In this chapter, we have defined an access control model. Based on this model, we have proposed an access
<table>
<thead>
<tr>
<th>AUTHORIZER</th>
<th>USER</th>
<th>OBJECT</th>
<th>ACCESS-TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>s</td>
<td>o</td>
<td>t</td>
</tr>
</tbody>
</table>

User A          User B  V  READ
(DBA)

User A          User C  P  RUN

User A          User D  K  READ

Figure 4.5: Access Rules on View, Program and Data Object
control mechanism for a DHDBMS. The proposed access control mechanism supports content-dependent and functional access control policies. It has already been discussed that content-independent access control policy is a special case of content-dependent access control policy.

We have used view mechanism for content-dependent and functional access control. We have also suggested use of application program authorization technique for functional access control.

This proposed access control mechanism is to be built in the Global Functional System. It can be applied in a DHDBMS without affecting its query processing mechanism.