CHAPTER 3
DATABASE SECURITY

"With the true view all the data harmonize, but with a false one the facts soon clash".

Aristotle
Ethics

"In order to talk about the semantics of database operations, we must have a model of "universe" that the database is supposed to represent.... The user view of the world is one table, with one column for each attribute of the database.... In addition we are given a set of integrity constraints which for the time being, we assume are functional dependencies. Later... we shall deal with multivalued dependencies."

A.V.Aho, C.Beeri, and J.D.Ullaman
The Theory of Joins in Relational Databases,

A paper based on this chapter and Chapter 1 entitled "Access Control Issues in Distributed Heterogeneous Database Management System" was presented at the International Conference on Information Technology, ICIT'90 at Kuala Lumpur in September 1990 and the same has been published in the proceedings of this conference entitled "IT for Competitive Advantage : Opportunities for Development" [GOY90b].

74
3.1 Introduction

As automation increases and our reliance on computer systems grows, it becomes important to ensure that the information entrusted to these systems is protected (See Figure 3.1). In fact, the increasing sophistication of users makes these systems vulnerable. A number of cases of misuse involving computers keep coming in newspapers and periodicals. Examples of misuse include the theft of money by interception of electronic-fund-transfer messages among banks, and access of confidential information for illegal purposes.

The increased usage of database to store large amounts of data has created new security problems. Typically a database contains data of various degrees of importance and levels of sensitivity. This data is shared among a wide variety of users with different responsibilities and privileges. It is therefore necessary to restrict users of the database to those portions of the total data that are necessary for their activities [MIR82]. The introduction of Distributed DBMSs has created more threats to database (DB) security.

3.2 Centralized DB Security Vs Distributed DB Security

In a centralized DBMS, it is important that a mechanism be provided to prevent the database from unauthorized disclosure, alteration and destruction.
FIG. 31: THE MAIN FACTORS IN COMPUTER SECURITY
Security and privacy issues in a DDBMS also depend on the reliability of communication links and the security of the distributed operating system. Hence it becomes much harder to verify. Despite its importance, there has been very little work done in the area of security and privacy in a distributed database environment.

In a DDBMS the following questions can be asked regarding security. With each question, we list several possible answers.

I) Where does the system store the access rules?

a) The system can store the access rules at the site where the data resides. That is, each local site retains the access rules for the data at that site. Other sites have no record of these access privileges.

b) The system can store the access rules in a global table so that all the sites of the system may access them.

c) The system can store the access rules at every site of the distributed system. Then each site has an identical copy of the access privileges.

II) How does each site control its data?

a) Each site has exclusive control of data which resides at that site. This is site autonomy [SEL83]. The revocation, granting, and
enforcement of access privileges of an object are local operations.

b) Partial degree of site autonomy. The access control enforcement may occur in some specified sites of the system (may include the site where the data is stored).

c) No site autonomy (each site can enforce access control over data which resides at any site). In this case, access control enforcement may occur at i) the site where the query is entered; ii) the site where the requested data is stored; iii) the site where the access rule resides; iv) a particular access control centre.

III) Which enforcement mechanisms are used?

a) Views (as used in System R [AST79]),
b) Query modifications (as used in INGRES [ST074]),
c) Others: such as predefined queries, filters etc.

IV) Of which level does the enforcement occur for each site?

a) Logical level: Internal, Conceptual or External,
b) Physical level.

In a centralized DBMS, questions I, and II, do not arise and only one choice can be adopted for questions III and IV. A DDBMS may adopt different combinations of strategies from I,II,III and IV. This may significantly
affect the operations of the system. Moreover, the different choices may have advantages and disadvantages. For example, when a security centre is used for enforcement, then that site may become a bottleneck. Hence it will degrade performance. On the other hand, if enforcement is done at the site where the query is entered, this will conflict with a requirement for site autonomy.

There are several types of access control policies [HOF77], [HSI79], [FER80], [DAT83]. We have discussed some common and important access control policies in Chapter 1. In this chapter, we will discuss prominent access control mechanisms used in existing DBMSs. In this chapter, we will also discuss an access control mechanism for DHDBMS proposed in [WAN87].

3.3 Identification and Authentication

The term security is used in a database context to mean the protection of the database against unauthorised disclosure, alterations or destruction. From this definition, it follows that the system must not allow any operations to be performed on the database unless the user is authorized for the operation concerned. For each user, therefore, the system must maintain a record - usually called the user profile - specifying the data objects the user is authorised to access and the operations the user is authorised to perform on those objects. Alternatively the
system must maintain a record for each object - an object profile - specifying the users that are allowed to access that object. Before accessing the database, these users, will have to identify themselves. Users will then go on to authenticate their identification.

The process of identification may involve supplying an User Identification number, or using machine readable identity cards or badges. Many terminals now available have special features to assist in this process, such as the ability to suppress the display of data being entered to preserve its secrecy, the ability to read specially designed cards or badges. Special terminals that allow identification by voiceprint or fingerprints have also been suggested (in such cases authentication as a separate step would not be necessary).

The process of authentication involves supplying information known only to the person the user has claimed to be in the identification procedure. This may be done by giving a password, or by answering some questions from the system [HOF77]. Often times, user authentication is handled outside of the database management system [LIN82].

3.4 Access Control Mechanisms for Centralized DBMSs

Several mechanisms have been proposed for authorizing access to objects stored in a centralized
database. One such mechanism, the use of passwords to authorize access, is quite inadequate in a database environment composed of dynamically changing sets of objects and users. While passwords may be acceptable mechanism for authentication of users, they are not very useful for access control. With this mechanism, we cannot implement name-dependent or content-dependent access control policies.

An access control mechanism basically performs three kinds of operations [WIL82]:

- checking the ability of SUBJECT to access an OBJECT in a requested mode,
- granting access rights on a OBJECT to a SUBJECT, and
- revoking access rights on an OBJECT from a SUBJECT.

Therefore, an access control mechanism should include following features:

- It should figure out, when processing a user's request, whether the user is authorized to access specified objects;
- It should be able to dynamically modify access rights of a user on specific objects; and
- For each object defined in the database, it should be able to identify the associated users who have specific access rights on this object.

3.4.1 CODASYL DBMSs Access Control

A typical CODASYL DBMS provides access control
through the schema and subschema of the data definition language. The data definition language defines access control security policies by using privacy locks for which keys must be specified to gain access. Privacy locks can be specified for each schema, subschema and record etc. The DBA is fully responsible for granting access rights by providing passwords to the potential users.

There are two major problems with the CODASYL approach to access control enforcement. First, it does not provide a fine enough granularity of protection for many applications. A user typically has access to either all or no occurrences of a record type or set type. There is no way to grant access to an arbitrary subset of occurrences of a record type. Second, this mechanism is incapable of enforcing anything but strictly content-independent access control policies. Content-dependent access control policies are unenforceable with just this mechanism.

3.4.2 Relational DBMSs Access Control

There are two common access control mechanisms which are used in relational DBMSs. These are (1) access control through view mechanism and (2) access control by query modification. Both these mechanisms can provide content-dependent access control and are discussed below.
3.4.2.1 Access Control through View Mechanism

The structure of data to be stored by a Database Management System is usually decided by a database administrator. Individual users and applications are generally interested in only subset of data stored in the database. Often, they wish to see this subset structured in a way which reflects their particular needs. Since it is not generally possible to structure a database so as to please all of its users, some mechanism is needed whereby each user can view the data according to his own requirements. The representation of data structure as seen by a user is referred to as an external schema; the view mechanism is a means by which a DBMS can support various external schemas.

Besides providing users with tailored views of data, the view mechanism contributes to [CHA75]:

1) Data independence: giving applications a logical view of data, thereby isolating them from data reorganization.

2) Data isolations: giving the application exactly the subset of data it needs, thereby minimizing error.

In relational databases, views are defined as the result of SELECT statement (in SQL) producing a result relation from one or more operand relations [CHA75]. The result relation gives to the user a new view of the database.
(or external schema in the ANSI/SPARC terminology), which is built through the standard query language. These views are objects of authorization, and in fact views are typically used for granting to users the capability of accessing selected information, instead of complete relation [GRI76]. Here is an example. Let us assume that there is a relation for parts.

\[ P(NO, NAME, COLOUR, WEIGHT, CITY) \]

and a view REDPARTS is defined over this relation as follows:

```
DEFINE VIEW REDPARTS AS:
   SELECT NO, NAME, WEIGHT, CITY
   FROM P
   WHERE COLOUR = 'Red'
```

When DEFINE VIEW is executed, no data retrieval is performed at that time, instead, the definition of the view is simply saved in the dictionary under the specified view name. When this view name is referenced in a query, only then data manipulation operations take place. For instance, let us see the effect of SELECT operation against REDPARTS.

```
SELECT NO, NAME, WEIGHT
   FROM REDPARTS
   WHERE CITY = 'Bombay'
```
This SELECT statement looks like a normal SELECT statement on a conventional base table. The system handles such operation by converting it into an equivalent operation on the underlying base table. In this example, the equivalent operation is

```
SELECT NO, NAME, WEIGHT
FROM P
WHERE CITY = 'Bombay'
AND COLOUR = 'Red'
```

The new statement can now be executed in the usual way. This conversion is done by (in effect) merging the SELECT issued by the user with the view definition stored in the dictionary. From the dictionary, the system knows that references to the view REDPARTS are really references to base table P; it also knows that any retrieval from REDPARTS must be further qualified by the WHERE condition

```
"COLOUR = 'Red'"
```

In creating the internal representation of a view, names in the view definition statement are bound to be specific objects they reference, namely tables and other views. A view is therefore logically dependent on the continued existence of all objects that it references. If an object is dropped or substantially changed, then the views referencing these objects must be invalidated.
In addition, since at view usage time the operations performed on the view will be translated into operations on the underlying objects, the view must have the necessary privileges on those objects. The privileges are granted to the view by the view definer. The view definer must therefore have the necessary privileges on the objects referenced in the view definition to be able to define the view. This allows users who do not have privileges on the objects, to access those objects through the view, according to protection requirements expressed by the view. If all privileges on an underlying object are revoked from the view definer, the view is no longer valid. The granting and revocation of authorization on views is discussed in [GRI76]. This approach of access control through view mechanism is implemented in System R [AST79].

3.4.2.2 Access Control by Query Modification

A second method of access control is by means of query modification. In this approach presented in [ST074] and [ST076], access control is achieved dynamically by the modification of a user request into a form which ensures no access violation. This approach has been implemented for a relational DBMS INGRES [DAT87]. The end-user interacts with INGRES via a high level non-procedural language (QUEL or SQL) which allows the statement of very complex access restrictions. In this case, access control is achieved through the query language. The Data Base
Administrator (DBA) expresses the desired constraints as additional qualifiers and restrictions in a query language (QUEL or SQL) which will be AND-ed to a query from a user. The resulting interaction \([(\text{query}) \text{ AND } (\text{restrictions})]\) is compiled into a series of simple interactions which are executed without any further concern for data security.

The access control predicates are stated in QUEL by means of the DEFINE PERMIT (in SQL it is done through CREATE PERMIT) statement. Here is a simple example for a relation \(P\) defined in Section 3.1:

```
DEFINE PERMIT RETRIEVE ON P TO RAM
WHERE P. CITY = 'Bombay'
```

(user RAM is allowed to see Bombay parts only). Such constraints are stored in a system catalogue and are enforced by query modification. In other words, user requests are modified before execution in such a way as to guarantee that they cannot possibly violate any access constraints. For example, suppose user RAM is allowed to see parts stored in Bombay only (as above), and suppose user RAM issued the request:

```
RETRIEVE (P.NO, P.WEIGHT)
WHERE P.COLOUR = 'Red'
```
INGRES will automatically modify the query to the form:

```
RETRIEVE (P.NO, P.WEIGHT)
WHERE P.COLOUR = 'Red'
AND P.CITY = 'Bombay'
```

And of course this modified query cannot possibly violate the access constraint.

As suggested above, the query modification process just outlined is actually identical to the technique used to implement views. The main difference between access control through view mechanism in System R and query modification in INGRES is that in System R, views are treated like base tables for GRANT and REVOKE operations where in INGRES, DEFINE PERMIT operation is allowed only on base tables and not on views. Therefore in INGRES, access control mechanism cannot be achieved by view mechanism because there is no authorization procedure on views.

### 3.5 Access Control Mechanism for Distributed Homogeneous DBMSs

The concept of view mechanism of System R is extended [BER88] to R*[LIN85], [WIL86] for access control. R* is a prototype distributed homogeneous database management system. It's component DBMSs are built using System R architecture. We have already discussed that system R is based on the relational model. The extension of
views in R*, preserves the appearance of views as virtual tables. Views are derived from one or more relations stored at different sites, using the relation operators (project, join etc.) [CER84]. In R* terminology, views without associated authorization properties are called shorthand views, their use being mainly to provide an easier interface to users. Those views which are used for providing authorization are called Protection Views. As we are discussing access control in this thesis, we are mainly concerned with protection views. Three examples for defining views in R* (using SQL) are shown in Figure 3.3. These views have been defined over the three relations stored at three different sites (Figure 3.2).

From the above examples, it is clear that a view can be defined over several tables stored at different sites. Such views can be defined and stored at any site depending upon user requirements. In R*, views are treated like base tables as is the case in System R. Now let us discuss the issue of storing the access rules. There are two key questions: Where to store (and apply) access rules, and where to define access rules.

In R*, a table may be created at any site [LIN82]. This site is known as the birth site. It may be stored at some other site, which is referred as table store site. The birth site of a table plays a central role since it maintains the information about the actual store site for
Database at Site A

EMPLOYEE (EMP#, NAME, SALARY, ADDRESS, MANAGER, DEPARTMENT)

Database at Site B

PLANNING-TRIP (EMP#, DESTINATION, COST, DATE, APPROVAL)

Database at Site C

DEPARTMENT (DEP#, NAME, LOCATION, BUDGET)

Figure 3.2: Database Example for R*
Example 1

CREATE VIEW VEMP (VEMP#, VNAME, VSALARY)
    AS SELECT EMP#, NAME, SALARY FROM EMPLOYEE

Example 2

CREATE VIEW SAL (A-SAL, DNO)
    AS SELECT AVG (SALARY), DEPARTMENT FROM EMPLOYEE
    GROUP BY DEPARTMENT

Example 3

CREATE VIEW EMP-LOC (E-NAME, E-LOCATION)
    AS SELECT NAME, LOCATION FROM EMPLOYEE, DEPARTMENT
    WHERE DEPARTMENT = DEPT#

Figure 3.3: Examples of view definition in R*
that table. This allows tables to migrate to different sites, without affecting the existing application programs. The table store site is incharge of storing and managing all catalogue information about the tables it stores. Among this information are the access rules, which define the set of local and non-local users authorized to access the tables stored at that site. In other words, authorization information is distributed together with the data to which it refers. If a table migrates to another site, the authorization information related to this table is moved to the table's new store site. Because the access rules are stored with the data; grant and revoke of access privileges on locally stored data are local operations. The participation of other sites is not required to perform such requests. Therefore, site autonomy is preserved as there are no global authorization tables.

The second question concerns where to define access rules; in other words, where access rights can be granted. In R*, a grant for access rights can be initiated from any site, but it is automatically shipped to the table store site, where it will be performed. Since R* only stores access rules at the site storing the data, the rules can only be perverted during the definition by a system programmer at the store site, or some one with the ability to interfere with the communication system. The granting, revocation and checking of authorization privileges in R* is similar to system R. The details can be seen in [BER88].

92
In this section, we have discussed the access control mechanism developed for a prototype distributed homogeneous DBMS R*. The component DBMSs of R* are built using System R· architecture. System R is based on the relational model. We have already discussed in Section 1.11 that there is no distributed homogeneous system available today whose component DBMSs are based on either the network or hierarchical model.

The access control mechanism discussed for R* is not applicable in a DHDBMS because of the following two reasons:

1) The component DBMS in a DHDBMS may have been built using different data models.

2) A DHDBMS uses Global Data Model concept for defining Global Conceptual Schema. There is no such concept in R*. (In System R, a view may be defined over several relations (tables) stored at one site, where in R*, a view may be defined over several relations (tables) stored at different sites.)

3.6 Access Control in a DHDBMS: [WAN87] proposal

In chapter 1, we have discussed access control issues in a DHDBMS. [WAN87] has studied content-dependent access control problem in a DHDBMS and proposed a solution. In this section we discuss the proposed solution of [WAN87].
The DHDBMS architecture used in [WAN87] is based on the ER model and its global conceptual schema is a set of entities and relationships among the entities. This architecture is shown in Figure 2.2. We have discussed in Section 3.4.2.1 that content-dependent access control can be implemented in a Centralized Relational DBMS through view mechanism. [WAN87] has suggested access control in a DHDBMS through the use of view mechanism. However, the view materialization technique suggested by [WAN87] is different than the one discussed in Section 3.4.2.1. As per [WAN87], the global conceptual schema has an access control mechanism that checks the access rights of the user as recorded in a global catalogue for each global database object referenced in a query. If the user has no rights for accessing a global database object, the request is rejected. If the user has access privileges on a global database object, then this query is allowed to be processed. In such cases, an execution plan is generated and subqueries are distributed to different sites for accessing/processing local database objects corresponding to global database object. As per site autonomy norms, if a user wants an access to a local database object in a component DBMS through a subquery, then his access authorization on that local database object will have to be checked by the access control mechanism of that component DBMS. If he has access privilege on this local database object, only then his subquery will be allowed to be processed at the component DBMS site.
Let us take a case for illustrating the access control problem in a DHDBMS. For example, suppose that at global conceptual schema level, user A is the owner of the global data object CIRCUIT with four attributes NAME, DESIGNER, DATE and LAYOUT. It means, A is owner of corresponding local data objects in component DBMSs. Suppose that he wants to allow user B to be able to access this data only for circuits created before 1986. User A can define a global external view (using NQUEL syntax):

Range of C is CIRCUIT
Define 85-CIRCUIT (NAME, DESIGNER, LAYOUT)
Where C.DATE = "12/31/85"

A can grant 85-CIRCUIT to user B and this information about grant is stored in a global catalogue. User B can now use 85-CIRCUIT to access CIRCUIT names, designers, and layouts, but only for circuits with a creation date before 1986. View 85-CIRCUIT is defined only at the global functional model and hence is unknown to component DBMSs, since we assume that each component DBMS has independently operating access control system. Thus when user B references view 85-CIRCUIT in a query, the global data model will allow this query to be processed as user B has required access privilege over the view 85-CIRCUIT. During query processing operation, this query is modified in an intermediate query in which view name 85-CIRCUIT is replaced by global database object name.
Circuit with qualification clauses. Then this intermediate query is decomposed in single-site subqueries formed over local database objects. Then an execution plan is generated and the subqueries are distributed to component DBMS sites for processing. When such a subquery tries to reference a local database object in a component DBMS, it is denied access by the access control system of the component DBMS, as user B does not have access privilege on these local database objects. Thus, this traditional algorithm of view materialization produces an inconsistency between the global and local access control systems since the local system is unable to record the fact that user B has been granted access to part of local database objects as defined by the view.

To overcome this problem, [WAN87] has proposed the following procedure.

1) Whenever a view is created in the global functional model, the following operations are performed.

a) Save the view definition in the global catalogue and insert access rules for the user (say A) who defines the view giving him full privileges of the view.

b) In the appropriate DBMSs, create a record type (if component DBMS is hierarchical or network), or relation (if component DBMS is relational) with fields corresponding to the attributes in the
target list of view definition. This record type or relation is used to store the materialized view during the processing of a query that references the view. (In this way the proposed solution of \cite{WAN87} creates a view in component DBMSs, especially in the case of network and hierarchical DBMSs.)

c) Request the appropriate component DBMS to permit the user who defined the view to write the record type or relation created above.

2) When a view is granted to a user (say B), then the following operations are performed:

a) An access rule is entered into the global catalogue of the global functional model to record the fact that user B is now authorized to use the view.

b) The component DBMS where the view will be materialized is requested to grant to user B the right to read the record type or relation that was created (in 1(b)) to hold the materialized view. This is done by sending a message to the DBAs of component DBMSs, asking them to issue the required grant.

3) When a user issues a query that references a view, the query is processed by the global functional model with the view being handled as if it were an ordinary
database object (i.e. view name is not replaced by data object name and qualification clause). The user therefore must have an access rule allowing him to use the view or the query processing will not continue. The query is parsed, decomposed, and an execution plan is generated to process the query (but view name is not replaced). The subqueries are then sent to the site that are responsible for processing the view.

When a site receives a subquery for processing, the first thing it does is check to see if it contains a reference to a view. If it does, then the following operations are performed:

a) Generate a subquery for the owner of the view to materialize the view (write the data from the materialized view into the record type or relation created to hold it).

b) Generate a subquery for the user to read data from the materialized original subquery.

c) Once the processing for the subquery is completed, a final subquery is generated to delete the data in the materialized view (i.e. in the record type or relations that contains the materialized view). This subquery is executed with the privileges of the view creator.

[WAN87] states that if this modified procedure of view materialization is used then the problem of
inconsistency in global and local access control systems will not arise and in this way access control in a DHDBMS can be implemented. The access control mechanism suggested in [WAN87] is applicable in a DHDBMS whose component DBMSs are either relational or CODASYL (i.e. network or hierarchical) DBMSs.

3.7 Summary

Database security is an important area. In this chapter, we discussed different aspects of database security. We described database security issues for centralised and distributed DBMSs. We discussed identification and authentication procedures. We also discussed access control mechanisms used in centralized DBMSs. During our literature survey, we found that although a lot of work has been done in database security area in general, but little work has been done in the area of access control in DHDBMS. We have already discussed the access control mechanism used for R*. We also discussed the access control mechanism proposed by [WAN87] for a DHDBMS. Some work related to access control in DDBMS is discussed in [WO079],[BUS82],[KER82],[LEV82], but none of these address the enforcement of access control for a general purpose DDBMS.