Proceedings of National Conference on Recent trends in DBMS Concepts and Practice, Sponsored by AICTE, NMAMIT, Karnataka. 8-9 March-2002, pp.8-19, 2002
Archived in Dspace@nitr, http://dspace.nitrkl.ac.in/dspace

Secure Database Design, A comparison of Object database Vs relational database security at SQL level

Bibhu D. Sahoo
M. Balaram Prasad
Department of Computer Science Engineering & Applications
Regional Engineering College, Rourkela, PIN-769 008

ABSTRACT

The most commercially available database provides security facility for a single table. However the situation has been changed, as enterprises move from local database to information backbone composed of cooperating heterogeneous database and then direct sharing of information between enterprise on the Internet. A strict security measure at database level is essential for a secure database design. This paper describes the secure SQL features as propounded by Winslett et al[1] which is an extension of SQL[4,5,6] .The object oriented model are more complex than relational model, and object orientation is not based on a formal mathematical model like relational model. For this reason the model for secure object oriented database are complex than the relational secure database model. For a object-oriented database, separate security assumption is to be made about the object orientation model. A discussion has been presented about the different security aspect of object-oriented database and relational database and finally it compares the security measures of both the approaches. A brief discussion is presented on security and user authorization in SQL2. It also indicates various choices that one may implement while designing a secure database.

1. Introduction

The data stored in the database need to be protected from unauthorized access, malicious destruction or alternation and accidental introduction of inconsistency. Data base security usually refers to protection from malicious access , some times it use along with integrity because in practice, it is difficult to draw a dividing line between security and integrity. To protect the database, security measures are to be deployed at various levels. They forms a hierarchy as DBMS security measure, Network level security, OS security measure, both at physical and human level. As it is not possible to provide adequate security at the lower level it required to ensure strict high-level security measures at database level. A database is said to be secure if, (i)no subject is able to obtain information without authorization, (ii)No subject is able to modify information

without authorization, (iii)No mechanism exists whereby a subject authorized to obtain information can communicate that information to a subject not authorized to obtain it, and (iv) No subject is able to activate a method without authorization.

Current security proposals have been classified into two groups, those that offer "*discretionary*" access controls and those impose "*mandatory* " access control. The discretionary access control is very much similar to that of security available with the file system of the operating system and database. Mandatory security is the tightest type security at multilevel database.

1.1 Discretionary Security

Discretionary security measures include the security features associated with current-day file-system and database. The new security measures are essential because of development of cooperating heterogeneous database and information backbone in Internet. A new semantics is develop over the conventional query language to be called as secure query language. The necessity of the secure query language can be explained with following example.

Let a person is attempting to ascertain the value of attribute A of a tuple t in a relational database, but is not permitted to read the value. On attempted access, the DBMS might return an "access denied message", but this approach may conveyed too much information about the value of t.A. As a alternative, the system might plant a cover story to hide the value of attribute A, by replacing the actual value of t.A by a null value. However, this approach may divulge too much information .The user on seeing a null value for t.A may try innocently or maliciously to update t to replace the null by a concrete value. If the update request is rejected , then the fact that nulls really means "access denied" has been divulged. If the update request accepted then the DBMS has actually got to maintain two different value for t.A, so that the user will see the value for t.A that he or she expects, after the update has been committed. At that point the DBMS is storing multiple versions of reality, and the exact meaning of the data in the system is unclear.

1.2 Mandatory Security

Mandatory protection is based upon the policy defined by US department of defense[1985] and interpreted for computerized systems by Bell and LaPadulla[1974]. Under mandatory protection, objects(data items) are assigned a

security classifications and subjects (active process, users) are assigned a security clearance. The classifications and clearances are both taken from a common domain of access classes. These access classes known as *labels* or *levels* , which form a finite partially ordered set known as *security hierarchy* or simply the *hierarchy*. For example, labels Top Secret(*TS*), Secret(*S*), Confidential(*C*) and Unclassified(*U*) are widely used. For any two labels c_1 and c_2, if $c_1 > c_2$ in the partial order, then we say that c_1 is higher than or above c_2 . If $c_1 \geq c_2$, we say c_1 dominates c_2 .

The mandatory security model proposed by Bell-Lapadla imposes the “no read up, no write down” restrictions on access by subjects. subjects are only permitted to write to a level that dominates their own. These assumption sufficient to prevent subjects from passing information directly down ward through the security hierarchy.

2. Taxonomy for secure relational databases

2.1 Semantics for secure relational database

Semantics for secure relational database[1] is defined initially for ordinary(single level) relational database. The approach is extended to incorporate secure case. Let D_1, ..., D_d represents a finite set of domains, where each consists of a set of values. The relational database schema defined as

$\{R_1(A_{11}:D_{11}, \ldots, A_{1n1} : D_{1n1}), \ldots, R_m(A_{m1}:D_{m1}, \ldots, A_{mnm} : D_{mnm})\}$

where each R_i is the name of a relation; A_{ij} is the name of a unique attribute of relation R_i; and D_{ij} is the domain of attribute A_{ij} (that is, one of D_1, ..., D_d). A relational database instance corresponding to this schema is an assignment of a finite subset $D_{ia} \times \ldots \times D_{in}$ to each relation R_i of the schema. Some times the term relational database used to refer to the combination of a schema and an instance. A database schema may also includes the information about integrity constraint that the database must satisfy, such as key and referential integrity constraints. These constraints can be described using an extension of the query language or by a sopecial-pupose construction.

To design a secure database a set of features are to be added to the databases schema, to describe relevant facts about security. That a schema has to describe security hierarchy of the application domain, and to say What the security level is of the information in the current databases .The domains are

extended ad called *labels*, whose values are all the different security class names in the security hierarchy. As partial ordered relation holds on *Labels*, hence $l \geq l'$ iff l dominates l'. Then two new unary relation defined as Anyone(label :Labels) and Self(label : Labels). Self gives the level of the subjects whose complete beliefs are contained in the database. Hence in ordinary relation, all database's user have same security clearance l and then instance for self must be { $<l>$ }.

2.2 Formalizing Multilevel Secure Relational Databases

The multi level secure (MLS) relational database is a set of ordinary relational database Most security proposals for MLS relational database have utilized syntactic integrity properties to control problems that arise in the presence of very tight security. Here we consider MLS database is a set of ordinary relational database with one database for each label in security hierarchy. The database all shares a common schema and all database tagged with its label. Additionally there is a binary relationship between dated base in the interpretation, which holds exactly when the label of the first database dominates the second , according to security hierarchy. From the property of security hierarchy , it follows immediately that the binary relationship is reflexive symmetric and transitive.

The label on a database indicates the label of the subject who believes that the contents of the database describe the state of the world accurately. The information in the database may have come from sources at many different levels, and thus may have different security classifications. For example, an S subject may have agree with some U beliefs, such as an unclassified list of pin number. although the pin code have security classification U, both U and S subjects believe that the pin code information is correct and both will include the pin code in the database of beliefs at their level.

The database tagged with a particular label contains the total *beliefs* of the subject having that label about the state of the world reflected in the schema. The term *belief* is used because the subjects with different labels may make different statement about the value of the same attributer for the same entity. The binary relation ship between database in an interpretation are given in figure 1(b), which shows subjects and databases for three linearly ordered levels : S, C and U. A subject believes only the contents of the database at its own level, as represented by solid arrow in figure 1(b) from a circle(subject) to a

box (the database) at the same level. The subject of each level see what they and the subject of each lower label believe, as represented by dotted arrows from subjects at one level to databases at lower levels. A subject may see many tupels that it doesn't itself believe. The information about which databases are visible from which levels is embedded in *self* and *anyone*. Formally an MLS database consists of a relational schema, as defined above, plus an interpretation

$$I = \bigcup_{l \in Labels} I_l$$

where each I_l is an ordinary relational database over S with label *l*, it can be called as I_l is the databases at level *l* or the interpretation at level *l*.

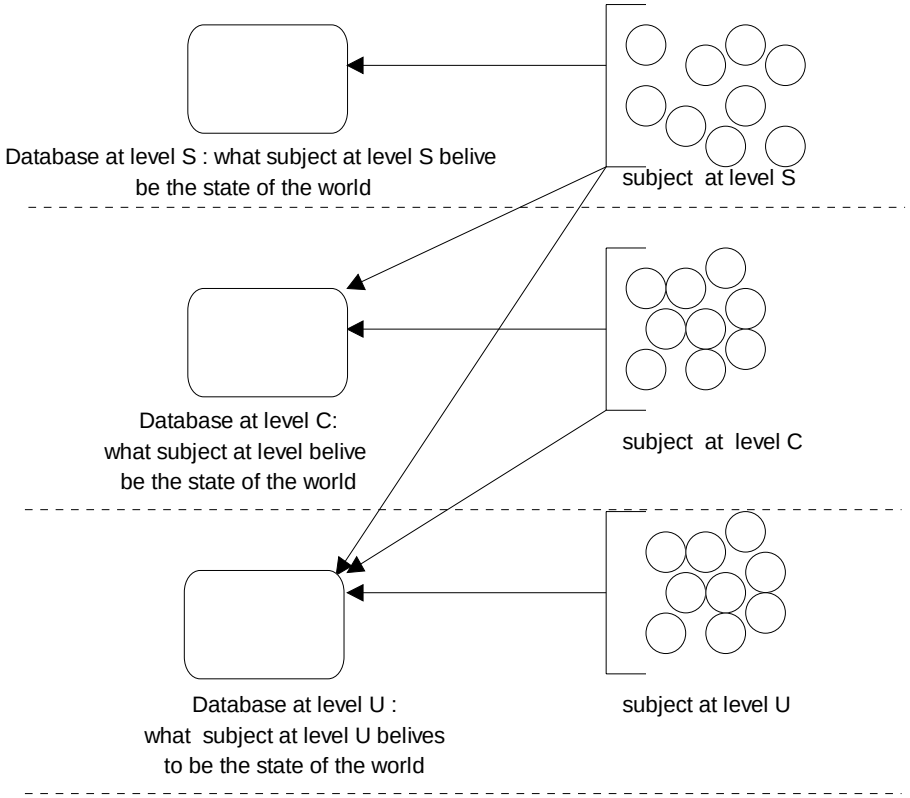

Figure 1(b). Relationship between subjects and databases of different levels

2.3 Formal query language and Secure Relational Algebra Syntax and Semantics

The secure relational algebra presented here is a formal query language for use with database interpretations. Syntactically, secure relational algebra is ordinary relational algebra[3,5,6], plus an additional symbol **B** , which can be thought of as meaning "believes" , which is applicable to any other formal query language, including logic based query language.

The Inductive definition proposed by Winslett et al[1] defines a secure relational algebra expression . Let R is a query expression , with the constant relations containing the m tupelos $\{ < C_{11}, ..., C_{an}>\}, ..., <C_{m1}, ..., C_{mn}>\}$. If E_1 and E_2 are query expression the then following rules holds :

(1) *Cartesian Product.* (E_1 5 E_2);

(2) *Union*. ($E_1 \cup E_2$), where both have arity n and have the same underlying domain for their *i*th attribute, $1 \leq i \leq n$;

(3) *Difference*. (E_1 - E_2), where E_1 and E_2 both have arity n and have the sane underlying domain for their *i*th attribute, $1 \leq i \leq n$;

(4) *Projection*. ($E_1[A_1, ..., A_n]$), where A_i is an unambiguous reference to an attribute;

(5) *Selection*. ($E_1[\varnothing]$), where $\varnothing$ is a selection condition (defined below);

(6) Level Shift. (**B**[E_1] E_2), where E_1 is a query expression whose result is a unary relation over *Labels*.

The items defined from (1) to (5) is derived from the usual definition of relational algebra [6] . The quantity **B**[E_1] E_2 poses the question contained in E_2 to the database levels determined by E_1. Using an inductive definition the selection condition $\varnothing$ occurring in E[$\varnothing$] can be , as usual, any of the following :

(1) *(t_1 **op** t_2),* where ***op*** is any operation of =, <, >, #, ≥, ≤, and each of t_1 and t_2 is a constant or an unambiguous reference to an attribute of E.

(2) ($\varnothing_1$ ^ $\varnothing_2$)

(3) ($\varnothing_1$ v $\varnothing_2$)

(4) $\neg\varnothing$

Reducing them to ordinary relational algebra carries out interpretation of query expression. The interpretation of a query expression E at level *l* (written | E |$_l$) is a relation defined as follows

- **Base Cases** | R |$_l$ is the instance of R at level *l.*
- **Cartesian product**: | ($E_1 \times E_2$)|$_l$ = | E_1 |$_l \times$ | E_2 |$_l$
- **Union** : | ($E_1 \cup E_2$)|$_l$ = | E_1 |$_l \cup$ | E_2 |$_l$
- **Difference** : | (E_1 - E_2)|$_l$ = | E_1 |$_l$ - | E_2 |$_l$
- **Projection** : | (E_1 [A_1, ..., A_n]) |$_l$ = | E_1 |$_l$[A_1, ..., A_n]
- **Selection** : | (E_1 [$\varnothing$])|$_l$ = | E_1 |$_l$ [$\varnothing$]

- **Level Shift** :

$$(\mathbf{B}[E_1]\ E_2)|_l = \bigcup_{\langle l' \rangle \in |E1 \cap Anyone|_l} |E_2|_l$$

There is no security labels satisfying E_1 and are dominated by l When $\langle l' \rangle \in |E1 \cap Anyone|_l$ is the empty set.

2.4 Secure SQL

This is an simple extension of SQL though the features available with SQL are sufficient to incorporate MLS database. However an explicitly extension to SQL is needed for pragmatic considerations. It is too hard for a user to write ordinary SQL queries that have an unambiguous interpretation. Also the **B** operation is not simple enough for users to grasp easily. So the extended SQL is clamed *secure SQL* [proposed by Winslett et al[1] . This permits the appearance of IN and >ANY, as well as EXISTS with a constant relation as an argument, as well as a sub-query[8].

3. Secure object-oriented databases

3.1 Taxonomy for secure object-oriented databases

An object O is a set of facets (methods, instances variable etc.). If m is a method of O, it represented as m∈ O ; if *v* represents a fact then *v*∈ O. When a system consists of a set of objects, that set is represented by U. In particular *v* is the name corresponding to the variable and m is the signature of the corresponding method and O is additionally contains a unique identifier that distinguishes it from all other objects in the system U. Let C be the set of all classes. For any class c∈ C , the set of super-class of c is denoted as sup(c). In the case of single inheritance, sup(c) will consists of a single element. For any object O ∈ U , the class of O denoted as class(O). The assumptions are summarized as following axioms:

AXIOM 1. *If an object o is in the system, then the class of O is also in the system; formally*

$$O \in U \rightarrow class(O) \in C$$

AXIOM 2. *Any facet x of an object o ∈ U is also a facet of the class of O, formally*

$$x \in O \rightarrow x \in class(O)$$

AXIOM 3. *If a class c has a facet x, any instance O of c will also have the facet x; formally*

$(x \in c) \Lambda(\exists O \in U)[c = class(o)] \rightarrow x \in O$

AXIOM 4. *If a class c is in the system, then all the super-class of c are also in the system; formally*

$c \in C \rightarrow (\forall d \in sup(c))[d \in C]$

AXIOM 5. *For any class $c \in C$, if x is a facet of a super-class of c, then x is a facet of c(it is either inherited or redefined), formally*

$d \in sup(c) \Lambda x \in d \rightarrow x \in c$

3.2. Design Parameters

The design parameters for secure object oriented data base are grouped into three categories that include eight design parameters as (i) Labeling Semantics : Underlying Model and Protection Interpretation, (ii) Structural Labeling :Protectable Entities, Label Instantiation, and Relationship Restrictions, and (iii) Dynamic Labeling : Authorization Flow, sensitivity Flow, and Information flow restriction.

3.2.1 Labeling Semantics

The labeling semantics refers to the assignment of security category to an item. In the case of a subject a clearance is usually assigned. In the case of an entity a sensitivity or classification is usually assigned. Two aspects deserve consideration under the heading labeling semantics:

X1.1 The model on which labeling is based.

X1.2 Exactly what is protected if an item is labeled.

X1.1 Underlying Model

The model on which labeling is based falls in one of three broad categories or a combination of these categories

Explicit levels : Sensitivity levels are assigned to entities and clearance levels to subjects. These levels are normally integers. Rules determine when a subject may access an entity; often a subject may read an entity if the subject's clearance level dominates the entity's sensitivity level. In general , the levels need not be integers- as long as the $\geq$ relation is defined for some labels associated

with the subjects and the entities. In many models the same label acts as an indication of an item's clearance when viewed as a subject and its
its sensitivity when viewed as an entity.

Access control lists: ACL s is lists associated with entities, containing the identification of subjects that are authorized to access the entity. Extension of ACLs has been proposed that do not only contain the identity of authorized assessors, but also the path through which such request has to flow.

Capabilities: Capabilities are the non-forgettable identifiers possessed by subjects. Such a capability is similar to a key for a padlock. Where a padlock is a subject that will be allowed to access a protected entity only if it presents an acceptable capability.

The combination of the first to approaches is popular. Entities are classified using a sensitivity level and a category. Only subjects with a proper clearance level and belonging to the specified category are allowed to access the entity. Classifications thus form an particular ordered lattice. However most models based on this combination ignore the category aspect of the classification and only address the (fully ordered)classification levels when developing the model. A subject considered here may be an object (including human objects using the database), a combination of objects, etc. For a subjects (i.e. elements of S) that may authorized to access an entity ***e*** with following possibility:

- An object with a clearance level that dominates the sensitivity level of ***e.***
- An object in possession of a capability to access ***e.***
- An object listed in the access control list of ***e.***
- An acceptable access path defined by access control list via which a request may reach ***e.***

X1.2 Protection Interpretation

This deals with exactly what is to be protected if an item is labeled. Some times an attempt is made to protect the fact that an item exists, while other protect the contents of an item . Related to this the three *dimension of protection* where

1. The data is classified
2. The fact that the data exists is classified
3. The rule for classifying is itself classified.

In case of access protection , users may know that a variable or method exists, but will get “access denied” error message when they try to activate a method or read or modify an instance variable without authorization. The use of access protection presents a possible convert channel, a highly cleared subject may create an object under certain circumstances, and an un-cleared subject may observe the fact that this object has been created. Hence the following definition :

Definition *: In an existence-protected model the fact that a labeled exists is hidden from unauthorized subjects.*

In an existence-protected model, if the existence of one entity implies the existence of second entity, then the sensitivity of the first must be at least as high as second. For access protected model it defines as;

Definition *: In an access-protected model, an unauthorized subject is not allowed to access a protected entity; ”not allowed to access” means that:*

- Any unauthorized messages sent to a protected object will fail.
- Any unauthorized message sent to a protected method will fail
- Any method attempting to access (read or write) an instance variable illegally will fail.

Some methods are only aimed at preventing authorized subject from obtaining information from a protected entities. In an existing -protected model , an object may receive a message from an unauthorized subject and activate the corresponding method . In this method it has three features;

- sends a message to an object that does not exist as far as the original subject is concerned,
- sends a message to another method that does not exist as far as the original subject is concerned
- accesses a variable that does not exist as far as the original subject is concerned.

It addition to this other protection models of interest are :

⇒ Use existence protection for objects, but hide classes totally from all subjects(except the data base system itself, which needs to access it to create instances of the classes). Thus no subject can gain direct information from the class and , from there, infer information about the instances of the class

⇒ Use access protection for classes (and therefore do not attempt to hide the structure of objects), but do hide the fact that an instance exists from a subject not authorized to access the instance.

3.2.2 Structural Labeling

Structural labeling deals with the influence of the structure of the data on the labeling of entities. The object-oriented model has a rich variety of entities with relationships between such entities. For example, an object is an instantiation of a class; an object may be an aggregation or composition of other objects; objects contain variables and methods; etc. These entities and relationships describe the structure of an object oriented data model. The three-aspect aspect in this concern is :

- (X2.1) Which entities may be labeled? Possibilities include objects, classes, methods, and instance variables.
- (X2.2) How and when are entities labeled?
- (X2.3) Does the model place restriction on the labeling of related entities ?

X2.1 Protectable Entities

A model for a secure object-oriented database must specify which entities may be protected. The Protectable entities may be objects, methods, instance variables, classes, class methods, class variables, etc. If only objects are allowed to be loaded, the whole object has the same sensitivity; this type of object is refereed as a *single-level-object*. If portions of an object(i.e. methods and instance variables) may be labeled individually, it provides finer granularity from a security viewpoint. Because the sensitivity of portions of such an object may be different, this type of object is refereed as a *multilevel object*[2].

X2.2 Label Instantiation

An object-oriented system is a dynamic system :objects are instantiated and destroyed continually. In order to compromise security, newly created objects must be protected immediately. The initial sensitivity of an entity reflects the inheritance of the entity. In particular one has to predetermined which subjects will be allowed to invoke a method of an object. Normal database activities will have no influence on the sensitivity of this method. Similarly, the inherent sensitivities of the instance variables of such an object may be predetermined reflecting the sensitivity of the value of such a variable or the sensitivity of the relationship between the object and the contents of that

variable. Three primary possibilities exist for determining the initial sensitivity of an object.

- The class must be labeled, and the label(s) specified for the class must apply for all instances of the class.
- Every object(and possibly its variables and methods) must be explicitly labeled when or after the object is instantiated
- Constraints may be specified-i.e., separate(logic) rules that determine the sensitivity of a newly instantiated object and then ensure that the entity is sensitivity labeled immediately.

A combination is also allowed with default labels derive from the class and individual labels given after instantiation where the default labels do not suffer.

X2.3 Relationship Restrictions

The labeling restrictions of the entities leads to the consideration of various relationships, classified as

- ***Aggregation:*** The relationships that exist between an object and its facets(name, instance variables, methods)
- ***Instantiation :*** The relationships that exist between a class and its instances.
- ***Inheritance:*** The relationships that exist between a class and its sub classes.
- Composition: The relationship between objects that are combined into a larger object.
- ***Association:*** The relationships for objects that exist in order to associate two or more other objects.
- ***Data structure membership:*** The relationships between a data structure (such as list) and a member of data structure; also relationships among members themselves.

The relationship restrictions may be divided into compulsory and additional restrictions. Compulsory restriction restrictions are those restrictions that a model must enforce as a result of design choices made elsewhere or as a result of the inherent object-oriented structure. Additional relationship restrictions are other restrictions a model may prescribe because they simplify the model or have some other benefit.

3.2.3 Dynamic Labeling

Dynamic labeling activity outlined by the grammar given below

$\Sigma \rightarrow M$

$M \rightarrow a_i\ T$

$T \rightarrow MT \mid \in \mid r$

Here Σ represents the primary accessor, in other words the object that sends the original message to the database. The non-terminal M represents a message; the production rule $\Sigma \rightarrow M$ models the message sent by the primary assessor.

The a_i represents an object; The a message causes a method to be activated for active object . The list of activities such an active method performs is represented by T. The production $M \rightarrow a_i\ T$ indicates that a specific method (a_i) is activated on receipt of a message after which the method executes a list of activities T.

The production $T \rightarrow MT$ represents the case where a list of activities T consists of sending a message, before executing some more activities; the production $T \rightarrow r$ represents the activity that terminates execution of active method and sends a reply to the calling method, while the production $T \rightarrow \in$ represents the activity that terminates execution of the active method and returns control to its calling method without sending a replay to calling method. The authorization flow deals with the question whether and how the clearance of a subject is influenced by the method activation.

Information flow deals with the flow of sensitive information through the system and more particularly the restriction that a model may enforce to ensure that such information does not flow to some where it is less protected.

Under *dynamic labeling* the measure aspects considered are;

- Message act on behalf of a subject and therefore the clearance of the message depends on the subject.
- Message also carry information -this information may be sensitive, requiring labels.
- If some of the sensitivity information contained in message is stored in variables of the receiving object, it must be ensured that an unauthorized subject cannot now access the information in this object. This can be ensured either by labeling the object or variable with a suitable label by disallowing information to be saved if the existing labels are not suitable.

4. Security Specification in SQL(SQL-92)

4.1. Security and user Authorization

The SQL-92 standard specifies a primitive authorization mechanism for the database schema, such that the only owner of the schema can carry out any modification to the schema. Thus schema modification, such as creating or deleting relations, adding or dropping attributes of relations and adding or dropping indices are only executed by the owner of the schema. SQL2 postulates the existence of authorization ID's, that are essentially user-names. It also provides an authorization ID, "PUBLIC" that can include any users. Like UNIX file system there are three kind of privileges: read, write and execute, SQL2 defines six types of privilege on databases . These privileges are : SELECT, INSERT, DELETE, UPDATE, REFERENCES, and USAGE.

The first four of these apply to a relation, which may be either a base table or a view. As their names imply that they give the holder of the privilege right to query the relation, insert into the relation, delete from the relation, and update tuples of the relation, respectively. A module containing an SQL statement cannot be executed without the privilege appropriate to that statement; e.g., select-from-where statement requires the SELECT privilege on every table(database) it accesses. The REFERENCE privilege is the right to refer to the relation in an integrity constraint. A constraint can not be checked unless the schema in which the constraint appears has the REFERENCES privilege on all data involved in the constraint. The USAGE privilege on a domain, or on several other kind of Schema elements other than relations and assertions is the right to use that element in one's own declarations. The three privileges - INSERT, DELETE, and REFERENCES may also be given a single attribute as an argument. In that case, the privilege refers to the mentioned attribute only. Several privileges, each mentioning one attribute, may be held; in that way one can authorize access to any subset of the
columns of a relation.

4.2. Creating privilege

There are two aspect to the awarding privileges: how they are created initially, and how they are passed from user to user. The initialization of privilege is carried out as follows in order to establish an ownership using SQL2. The detail steps are described as follows:

1. When a schema is created, it and all the tables and other schema elements in it are assumed owned by the user who created it. Thus the user has all possible privileges on elements of the schema.
2. When a session is initiated by a CONNECT statement, there is an opportunity to indicate the user with a USER clause.
3. When a module is created, there is an option to give it an owner by using an AUTHORIZATION clause. An user can become the owner of the module by including the following clause in to the module creation statement.

4.3 The Privilege-Checking Process

Any SQL operation has two parties (1) The database elements upon which the operation is performed and (2)The agent that causes the operation.

The privilege available to the agent derive from a particular authorization ID called the current authorization ID. Which is either (a) The *module authorization ID*, if the module that the agent is executing has an authorization ID, (b) the session authorization ID. One can execute the SQL operation only if the current authorization ID possesses all the privilege need to carry out the operation. The various principles of privilege checking process are:

- The *needed privileges* are always available if the data is owned by the same user as the user whose ID is the current authorization ID.
- The *needed privileges* are available if the user whose ID is the current authorization ID has been granted those privileges by the owner of the data or if the privilege have been granted to user PUBLIC.
- Executing a module owned by the owner of the data, or by someone who has been granted privileges on the data, makes the needed privileges available.
- Executing publicly available modules during a session whose authorization ID is that of a user with needed privileges is another way to execute operation legally.

The form of authorization statement is : AUTHORIZATION <*authorization ID*>.

4.4 Granting Privilege

SQL2 provides a GRANT statement to allow one user to give a privilege to another. In this process the first user retains the privilege granted, hence the GRANT can be thought of as “copy a privilege”.

The format of a grant statement consists is as follows :

GRANT<*privilage list*> ON <*database element*> TO <*user-list*>

Where;

- The keyword GRANT, ON and TO
- A list of one or more privileges such as SELECT/INSERT
- A database element are typically a relation, either a base table or a view(projection)
- A list of one or more users(authorization ID's)

The granting of privilege can be explained by following example. Assume that initially, the database administrator grants update authorization on a database "lone" to users U_1 ,U_2, and U_3. In turn the users U_1 ,U_2, and U_3 pass on these authorization to other users. The passing of authorization from one user to another can be represented by an authorization graph. The nodes of this graph are the users. An edge $U_i \rightarrow U_j$ is included in the graph if user U_i grants update authorization on loan to U_j . The root of the graph is the database administrator(DBA). A sample graph is given in Figure 2.

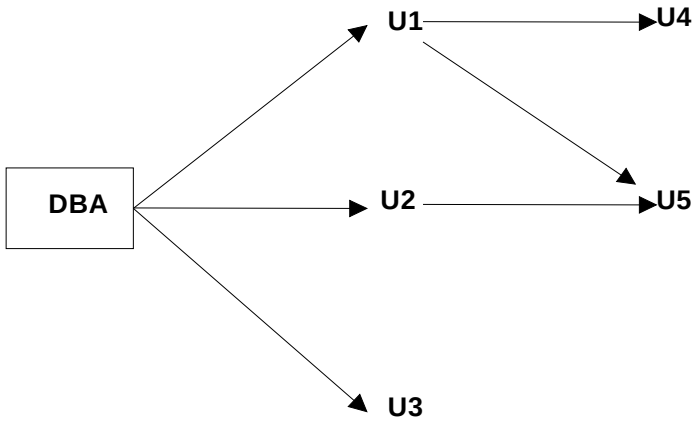

Figure 2. Authorization grants graph.

A user has an authorization if and only if there is a path from root of the authorization graph(namely , the node corresponding the database administrator) down to the node representing the user. In the above example if DBA decides to revoke the authorization of user U_1 then authorization of U_4 is to be revoked as U_4 granted authorization by U_1.

4.5 Revoking Privilege

A granted privilege can be revoked at any time. The revoking of privilege required to cascade, in the sense that revoking a privilege with the grant option that has been passed on to other user may require those privileges to be revoked too. The simplest form of revoked statements is as follows.

REVOKE <*privilege list*> **ON** <*database element*> **FROM** <*user list*>

Where *User list* includes one or more users authorization ID, *privilege list* includes a list of one or more privileges and *database element* are typically a relation, either a base table or a view(projection) . In addition to this the revoke statement syntax may include the following;

- The statement can end with the word CASCADE. If so, then when the specified privileges are revoked, one can revoke any privilege that were granted only because of the revoked privilege.
- The statement can instead end with RESTRICT, which means that the revoke statement cannot be executed if the cascading rules described in the revoked privileges having been passed on to others.
- It is permissible to replace REVOKE by REVOKE GRANT OPTION FOR, in which case the privileges themselves remain, but the option to grant them to other is removed.

An attempt to defeat authorization revocation is explained in figure 3. Though U_5 is granted authorization by U_1 , it will not be revoked as it also granted authorization by U_2 . As shown in Figure 3(a) a pair of user might attempt to defeat the rules for revocation of authorization by granting authorization to each other. If the database administrator revoke authorization from U_2, U_2 retains authorization through U_3 as given in figure-3(b). If authorization is revoked subsequently from U_3, U_3 appears to retain authorization through U_2, as shown in figure-3(c). However, when the database administrator revoke authorization from U_3, the edge from U_3 to U_2 and from U_2 to U_3 are no longer part of a path starting with database administrator.

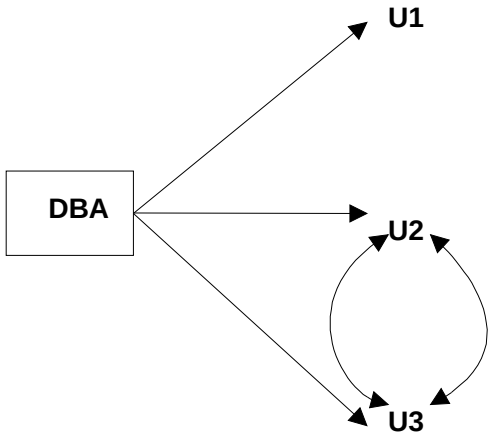

Fig. 3(a)

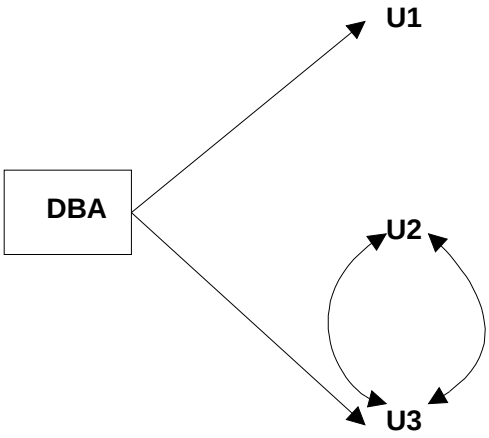

Fig. 3(b)

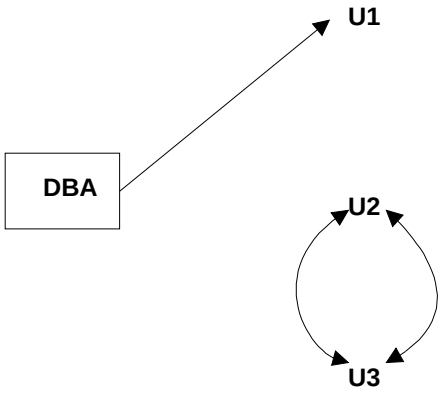

Fig. 3(c)

Figure 3. Attempt to defeat authorization revocation

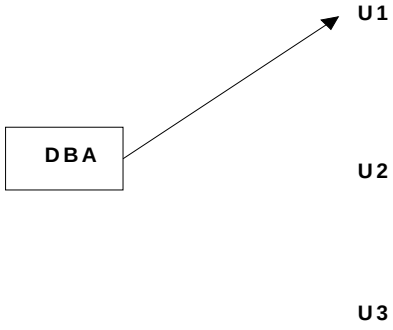

Figure 4. Authorization graph

It is required that all edges in an authorization graph be part of some path originating with the database administrator. Deleting such edge the resulting authorization graph is obtained as shown in figure 4.

5. Conclusion

Absolute protection of the database from malicious abuse is not possible, but the cost to the perpetrator can be made sufficiently high to deter most, if

not all, attempts to access the database without proper authority. The recent features available in SQL2 provide protection as well as privilege granting capabilities. A user who has been granted some form of authority may allowed to pass on this authority to other users. The data can be encrypted along with the authorization features will provide a better protection for sensitive data. However the problem of establishment of security measures between secure database is wide open, which an secure databases attempting to share the common data.

References

1.Winslett Marianne, Smith Kenneth and Qian Xiaolei, "Formal Query Languages for secure Relational Database", ACM Transaction on Database Systems, Vol. 19., No. 4, December 1994, pp. 626-662.

2. Olivier Martin S. and Solms Sebastiaan H. Von, "A Taxonomy for Secure Object-Oriented Databases", ACM Transaction on Database Systems, Vol. 19., No. 1, March 1994, pp. 3-46.

3. Desai, Bipin C., "An Introduction to Data Base systems", Galgotia Publications Pvt. Ltd., 1994.

4. Polyzois, Christos A. and Garcia-Molina, "Evaluation of Remote Backup algorithms for Transaction-Processing systems", ACM Transaction on Database Systems, Vol. 19., No. 3, September 1994, pp. 423-449.

5. Silberschatz Avi, Korth Hank, and Sudarshan S., "Data Base system Concepts", Mc-Graw Hill International Ed., 1996.

6. Ullman, J. D., "Principles of Database Systems", Galgotia Publications (P) Ltd, New Delhi(1991).

7. Majumdar Arun K., and Bhattacharya P.,"Database Management Systems", Tata-Mc-Graw Hill Publication Co. Ltd., 1997.