Proc. ACM SIGMOD International Conference on Management Data, Denver, Colarado, May 1991, pages 50-59.

TOWARD A MULTILEVEL SECURE RELATIONAL DATA MODEL

Sushil Jajodia *and Ravi Sandhu*
Center for Secure Information Systems
and
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444

Abstract

Although there are several efforts underway to build multilevel secure relational database management systems, there is no clear consensus regarding what a multilevel secure relational data model exactly is. In part this lack of consensus on fundamental issues reflects the subtleties involved in extending the classical (single-level) relational model to a multilevel environment. Our aim in this paper is to discuss the most fundamental aspects of the multilevel secure relational model. Specifically, we consider two requirements: entity integrity and update semantics. Our overall goal is to preserve as much as possible the simplicity and flexibility of the relational model without sacrificing security in the process.

1 INTRODUCTION

A large number of databases in the Department of Defense, the intelligence community and civilian government agencies contain data that are classified to have different security levels. All database users are also assigned security clearances. It is the responsibility of a *multilevel secure* database management system (DBMS) to assure that each user gains access—directly or indirectly—to only those data for which he has proper clearance. Private corporations also use security levels and clearances to ensure secrecy of sensitive information, although their procedures for assigning these are much less formal than in the government.

Most commercial DBMSs provide some form of data security by controlling modes of access privileges of users to data [3, 12]. These so-called *discretionary* access controls do not provide adequate mechanisms for preventing unauthorized disclosure of information. Therefore, commercial DBMSs are not suitable for use in multilevel environments. Multilevel systems require additional mechanisms for enforcing *mandatory* (or *nondiscretionary*) access controls [1].

As a result, there are several efforts underway to build multilevel secure relational DBMSs. These efforts are following the same path taken by object-oriented databases. On one hand, several database vendors (e.g, Oracle, Sybase, Trudata, to name a few) are busy building commercial products, and others (e.g, SRI [2, 10], SCTC [4]) are building research prototypes. On the other hand, there is no clear consensus regarding what a multilevel secure relational data model exactly is. This has led to continuing arguments about basic principles such as integrity requirements and update semantics. This lack of consensus on fundamental issues underscores the subtleties involved in extending the classical relational model to a multilevel environment. In absence of a strong theoretical framework it is unfortunate, but inevitable, that much of the argument on basic issues is unduly influenced by implementation details of specific projects.

Our aim in this paper is to discuss the most fundamental aspects of the multilevel secure relational model. It is our goal to be formal, analytical and objective—in the sense of implementation independent—in this exercise. We specifically consider two requirements.

- *Entity Integrity.* It is important to specify precisely all constraints that relations must satisfy since these constraints ensure that all instances in the database are meaningful. It is equally important to require only the minimal necessary constraints so as to allow as large a class of admissible instances as possible. In classical relational

*This work was partially supported by the U.S. Air Force, Rome Air Development Center through subcontract #C/UB-49;D.O.No.0042 of prime contract 5#F-30602-88-D-0026, Task B-O-3610 with CALSPAN-UB Research Center.

theory the essential constraints have been identified as entity integrity and referential integrity. In section 4 we consider the multilevel analog of entity integrity. We identify four core integrity properties which should be required of all multilevel relations. One of these is a generalization of the usual entity integrity requirement to a multilevel context, while the other three are new to multilevel relations. Our focus in this paper is on single relations and we do not consider multilevel referential integrity here.

- *Relation Updates.* Somewhat paradoxically, the understanding of update operations is crucial to achieving secrecy of information in multilevel systems. In section 5 we generalize the familiar INSERT, DELETE and UPDATE operations of SQL to a multilevel context. The main difference, with respect to the classical semantics of these operations, is that certain updates cannot be carried out by overwriting the data in place because doing so would result in leakage or destruction of secret information. This inescapable fact complicates the semantics of multilevel relations. Our goal here has been to preserve as much as possible the intuitive simplicity of these operations in classical relations without sacrificing security in the process.

The rest of this paper is organized as follows. The next section gives an overview of basic concepts of multilevel security. Section 3 reviews basic definitions for standard (single-level) relations followed by those for multilevel relations. Sections 4 and 5 discuss entity integrity and relation updates in a multilevel context as outlined above. The last section concludes the paper.

2 BASIC CONCEPTS

Below we give a brief description of the relevant multilevel security concepts. For a more detailed discussion, we refer the reader to [1] or [9].

The way in which a secure DBMS controls access to data is known as the system's *security policy.* In the context of multilevel databases, a secure DBMS must enforce a suitable interpretation of the mandatory access controls employed in manual systems. In the government and military sectors these controls are literally mandated by law. In the commercial world they are a matter of internal company policy. A well-accepted interpretation of these access controls for computerized systems was given by Bell and LaPadula. The Bell-LaPadula model was originally developed by analogy to manual systems in the military. An axiomatic derivation and generalization of the model were subsequently given by Denning.

The Bell-LaPadula model is stated in terms of *subjects* and *objects.* An object is a passive entity such as a data file, a record, or a field within a record. A subject is an active process that can request access to objects. Every object is assigned a classification, and every subject a clearance. Classifications and clearances are collectively referred to as *access classes* (or levels). An access class consists of two components: a hierarchical component (usually, Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U), in this order) together with a set of unordered categories (e.g., NATO, Nuclear, Army, etc.). Access classes are partially ordered in a lattice as follows: Given two access classes c_1 and c_2, $c_1 \geq c_2$ iff the hierarchical component of c_1 is greater than or equal to that of c_2 and the categories in c_1 include those in c_2.

Throughout this paper, we use the terms *high* and *low* to refer to two access classes such that the former is strictly higher than the latter in the partial order. Also if a user is logged on at an access class c, we refer to such a user as a c-user.

The Bell-LaPadula model imposes the following restrictions on all data accesses:

1. *The Simple Security Property.* A subject is allowed a read access to an object only if the former's clearance is identical to or higher (in the partial order) than the latter's classification.

2. *The $\star$-Property* (pronounced "the star property"). A subject is allowed a write access to an object only if the former's clearance is identical to or lower than the latter's classification.

It should be noted that these properties are necessary but not sufficient to allow the corresponding access. The sufficient conditions will include the usual discretionary access controls of commercial DBMSs.

As a consequence of these two restrictions, subjects having different clearances see different versions of a multilevel relation: A user having a clearance at an access class c sees only that data which lies at class c or below. As an example, consider the relation scheme SOD(Starship, Objective, Destination) where Starship is the primary key and the security classifications are assigned at the granularity of individual data elements. A user with Secret clearance will see the entire multilevel relation $\mathrm{SOD_S}$ shown in figure 1, while a user having Unclassified clearance will only see the filtered relation $\mathrm{SOD_U}$ shown in figure 2.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Voyager	U	Spying	S	Mars	S	S

Figure 1: $\mathrm{SOD_S}$

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Voyager	U	Null	U	Null	U	U

Figure 2: $\mathrm{SOD_U}$

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Voyager	U	Exploration	U	Talos	U	U
Voyager	U	Spying	S	Mars	S	S

Figure 3: $\mathrm{SOD_S}$

Starship		Objective		Destination		TC
Voyager	U	Exploration	S	Mars	S	S
Voyager	U	Spying	S	Mars	S	S

Figure 4: An illegal S-instance

It turns out that the system may not be secure even if it always enforces the two Bell-LaPadula restrictions correctly. A secure system must additionally guard against *covert* channels [8]. Covert channels provide indirect means by which information by subjects at high security classes can be passed down to subjects at lower security classes. To illustrate, consider once again the multilevel relation given in figure 1. Suppose that an U-user who sees the instance in figure 2 wishes to replace the second tuple of $\mathrm{SOD_U}$ by the tuple (Voyager, Exploration, Talos). From a purely database perspective, this update by the U-user will be rejected because the attribute Starship constitutes the primary key of $\mathrm{SOD_S}$. However, from the security viewpoint, this update cannot be rejected since doing so will be sufficient to compromise security. Since a Secret process can send one bit of information by either inserting or deleting a particular tuple at the Secret level, both Secret and Unclassified processes can cooperate to establish a covert channel. Thus, both tuples (Voyager, Spying, Mars) and (Voyager, Exploration, Talos) must somehow co-exist in $\mathrm{SOD_S}$, as in figure 3. This is called *polyinstantiation*: there are two or more tuples in a multilevel relation with the same primary key.

Polyinstantiation illustrates the intrinsic difficulty of extending the standard relational concepts to the multilevel world. Even the basic relational notion of a key does not have a straightforward extension to multilevel relations. Although polyinstantiation is inevitable in multilevel systems it must be carefully controlled so as to avoid confusion and ambiguity in the database. For instance the S-instance of figure 4 should not be allowed because it gives ambiguous information about the Voyager's objective at the S level. It is therefore most important to precisely identify the constraints required of all multilevel relations.

3 MULTILEVEL RELATIONS

We first review the basic concepts for the standard (single-level) relations, followed by those for multilevel relations. In the next section we will state four core integrity requirements that we feel must be satisfied by all multilevel relations.

The standard relational model is concerned with data without security classifications. Data are stored in relations which have well defined mathematical properties. Each relation has two parts as follows.

1. A state-invariant *relation scheme* $R(A_1, A_2, \ldots, A_n)$, where each A_i is an *attribute* over some *domain* D_i which is a set of values.

2. A state-dependent *relation* over R, which is a set of distinct *tuples* of the form $(a_1, a_2, \ldots, a_n)$ where each *element* a_i is a value in domain D_i.

Not all possible relations are meaningful in an application; only those that satisfy certain integrity constraints are considered valid.

Let X and Y denote sets of one or more of the attributes A_i in a relation scheme. We say Y is *functionally dependent* on X, written $X \rightarrow Y$, if and only if it is not possible to have two tuples with the same values for X but different values for Y. A *key* of a relation is a minimal set of attributes on which all other attributes are functionally dependent. The *primary key* of a relation is one of its keys which has been specifically designated as such.

Moving on to a multilevel world, a major issue is how access classes are assigned to data stored in relations. The proposals have ranged from assigning access class to relations, to individual tuples in a relation, to individual attributes of a relation, or to individual data elements of the tuples of a relation. In this paper, we will consider the general case and assign access class to individual data elements of a relation.

A *multilevel relation* consists of the following two parts.

Definition 1 [RELATION SCHEME] A state-invariant multilevel relation scheme

$$R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$$

where each A_i is a *data attribute* over domain D_i, each C_i is a *classification attribute* for A_i and TC is the *tuple-class* attribute. The domain of C_i is specified by a range $[L_i, H_i]$ which defines a sub-lattice of access classes ranging from L_i up to H_i. The domain of TC is $[\mathrm{lub}\{L_i : i = 1 \ldots n\}, \mathrm{lub}\{H_i : i = 1 \ldots n\}]$ (where lub denotes the least upper bound). □

Definition 2 [RELATION INSTANCES] A collection of state-dependent *relation instances*

$$R_c(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$$

one for each access class c in the given lattice. Each instance is a set of distinct tuples of the form $(a_1, c_1, a_2, c_2, \ldots, a_n, c_n, tc)$ where each $a_i \in D_i$ or $a_i = \text{null}$, $c \geq c_i$ and $tc = \mathrm{lub}\{c_i : i = 1 \ldots n\}$. Moreover, if a_i is not null then $c_i \in [L_i, H_i]$. We require that c_i be defined even if a_i is null, i.e., a classification attribute cannot be null. □

The multiple relation instances are, of course, related; each instance is intended to represent the version of reality appropriate for each access class. Roughly speaking, each element $t[A_i]$ in a tuple t is visible in instances at access class $t[C_i]$ or higher; $t[A_i]$ is replaced by a null value in an instance at a lower access class. We will give a more formal description using the filter function in the next section.

4 CORE INTEGRITY PROPERTIES

We next state four core integrity properties that must be satisfied by all multilevel relations.

Since a multilevel relation has different instances at different access classes, it is inherently more complex than a standard relation. In a standard relation the definition of keys is based on functional dependencies. In a multilevel setting the concept of functional dependencies is itself clouded because a relation instance is now a collection of sets of tuples rather than a single set of tuples.

We assume that there is a user specified primary key AK consisting of a subset of the data attributes A_i. This is called the *apparent primary key* of the multilevel relation scheme. We will return to the issue of what constitutes the primary key of a multilevel relation after we define the polyinstantiation integrity property.

In general AK will consist of multiple attributes. Entity integrity from the standard relational model prohibits null values for any of the attributes in AK. This property, taken from [2], extends to multilevel relations as follows.

Property 1 [Entity Integrity] Let AK be the apparent key of R. A multilevel relation R satisfies entity integrity if and only if for all instance R_c of R and $t \in R_c$

1. $A_i \in AK \Rightarrow t[A_i] \neq \text{null}$,
2. $A_i, A_j \in AK \Rightarrow t[C_i] = t[C_j]$, i.e., AK is uniformly classified, and
3. $A_i \notin AK \Rightarrow t[C_i] \geq t[C_{AK}]$ (where C_{AK} is defined to be the classification of the apparent key). □

The first requirement is an obvious carryover from the standard relational model and ensures that no tuple in R_c has a null value for any attribute in AK. The second requirement says that all AK attributes have the same classification in a tuple, i.e., they are either all U or all S and so on. This will ensure that AK is either entirely visible or entirely null at a specific access class c. The final requirement states that in any tuple the class of the non-AK attributes must dominate C_{AK}. This rules out the possibility of associating non-null attributes with a null primary key.

At this point it is important to clarify the semantics of null values. There are two major issues: (i) the classification of null values, and (ii) the subsumption of null values by non-null ones. Our requirements are respectively that null values be classified at the level of the key in the tuple, and that a null value is subsumed by a non-null value independent of the latter's classification. These two requirements are formally stated as follows.

Property 2 [Null Integrity] A multilevel relation R satisfies null integrity if and only if for each instance R_c of R both of the following conditions are true.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U

Figure 5: SOD_U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	S	S

Figure 6: SOD_S

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U
Enterprise	U	Exploration	U	Rigel	S	S

Figure 7: Violation of Null Integrity

1. For all $t \in R_c$, $t[A_i] = \text{null} \Rightarrow t[C_i] = t[C_{AK}]$, i.e., nulls are classified at the level of the key.
2. Let us say that tuple t *subsumes* tuple s if for every attribute A_i, either (a) $t[A_i, C_i] = s[A_i, C_i]$ or (b) $t[A_i] \neq \text{null}$ and $s[A_i] = \text{null}$. Our second requirement is that R_c is subsumption free in the sense that it does not contain two distinct tuples such that one subsumes the other. □

We will henceforth assume that all computed relations are made subsumption free by exhaustive elimination of subsumed tuples. The null integrity requirement was first identified in [5].

Consider the relation instance for SOD given in figure 5. The motivation behind the null integrity property is that if a S-user updates the destination of Enterprise to be Rigel, he or she will see the instance given in figure 6 rather than the one given in figure 7; since the first tuple in figure 7 is subsumed by the second tuple.

The multiple relation instances at different access classes are of course related. Each instance at an access class c is intended to represent the version of reality appropriate for the access class c. The next property is concerned with consistency between the different relation instances. The need for such a property was identified in [2]. However the formulations of [2] were incorrect. The correct formulation was first given in [5] and adopted by SeaView researchers in [10].

Property 3 [Inter-Instance Integrity] R satisfies inter-instance integrity if and only if for all $c' \leq c$ we have $R_{c'} = \sigma(R_c, c')$ where the *filter function* σ produces the c'-instance $R_{c'}$ from R_c as follows:

1. For every tuple $t \in R_c$ such that $t[C_{AK}] \leq c'$ there is a tuple $t' \in R_{c'}$ with $t'[AK, C_{AK}] = t[AK, C_{AK}]$ and for $A_i \notin AK$

$$t'[A_i, C_i] = \begin{cases} t[A_i, C_i] & \text{if } t[C_i] \leq c' \\ <\text{null}, t[C_{AK}]> & \text{otherwise} \end{cases}$$

2. There are no tuples in $R_{c'}$ other than those derived by the above rule.
3. The end result is made subsumption free by exhaustive elimination of subsumed tuples. □

The filter function maps a multilevel relation to different instances, one for each descending access class in the security lattice. Filtering limits each user to that portion of the multilevel relation for which he or she is cleared. Thus, for example, a S-user will see the entire relation given in figure 6 while a U-user will see the filtered instance given in figure 5. It is evident that $\sigma(R_c, c) = R_c$, and $\sigma(\sigma(R_c, c'), c'') = \sigma(R_c, c'')$ for $c > c' > c''$; as one would expect from the intuitive notion of filtering.

We are now ready to state our fourth and final requirement. In a standard relation there cannot be two tuples with the same primary key. In a multilevel relation we will similarly expect that there cannot be two tuples with the same *apparent primary key*. However, as we observed in section 2, secrecy considerations compel us to allow multiple tuples with the same apparent primary key.

Property 4 [Polyinstantiation Integrity] R satisfies polyinstantiation integrity (PI) if and only if for every R_c we have for all $A_i : AK, C_{AK}, C_i \rightarrow A_i$. □

This property stipulates that the user-specified apparent key AK, in conjunction with the classification attributes C_{AK} and C_i, functionally determines the value of the A_i attribute. Thus, PI allows the instance in figure 3 while ruling out the S-instance of figure 4.

Property 4 implicitly defines what is meant by the primary key in a multilevel relation. The primary key of a multilevel relation is $AK \cup C_{AK} \cup C_R$ (where AK is the set of data attributes constituting the user specified primary key, C_{AK} is the classification attribute for data attributes in AK and C_R is the set of classification attributes for data attributes not in AK), since from PI it follows that the functional dependency $AK \cup C_{AK} \cup C_R \rightarrow A_R$ holds (where A_R denotes the set of all attributes that are not in AK). Note that for single-level relations C_{AK} and C_R will be equal to the same constant value in all tuples. Therefore, in this case, PI amounts to saying that $AK \rightarrow A_R$, which is precisely the definition of the primary key in relational theory.

Property 4 was originally proposed in [2]. However it was coupled with an additional multivalued dependency requirement $AK, C_{AK} \rightarrow\rightarrow A_i, C_i$ to be satisfied by every instance. There are unpleasant consequences of this multivalued dependency, as pointed out in [5]. Thereafter our position has been that polyinstantiation integrity should only require the functional dependency stated in property 4.

5 THE UPDATE OPERATIONS

In this section, we discuss in detail the three update (insert, update, and delete) operations. We keep the syntax for these operations identical to the standard SQL.

Let $R(A_1, C_1, \ldots, A_n, C_n, TC)$ be a multilevel relation scheme. In order to simplify the notation, we use A_1 instead of AK to denote the apparent primary key.

Consider a user logged on at access class c. Now a c-user directly sees and interacts with the c-instance R_c. From the viewpoint of this user the remaining instances of R can be categorized into three cases: those strictly dominated by c, those that strictly dominate c and those incomparable with c. The following notation is useful for ease of reference to these three cases.

$$
\begin{array}{lcl}
R_{c'<c} & \equiv & R_{c'}, \text{ such that } c' < c \\
R_{c'>c} & \equiv & R_{c'}, \text{ such that } c' > c \\
R_{c'\sim c} & \equiv & R_{c'}, \text{ such that } c' \text{ incomparable with } c
\end{array}
$$

Security considerations, and in particular the $\star$-property, dictate that a c-user cannot insert, update, or delete a tuple, directly or indirectly (as a side-effect) in any $R_{c'<c}$ or $R_{c'\sim c}$. Since actions of a c-user cannot impact any $R_{c'<c}$, the effect of insertion, update or deletion must be confined to those tuples in R_c with tuple class equal to c. Because of the inter-instance property these changes must be properly reflected in the instances $R_{c'>c}$. The latter effect is only partly determined by the core integrity properties of section 4 leaving room for different interpretations (see [4, 5, 7, 11, 13]).

Strictly speaking in all cases we should speak of operations being performed by a c-subject (or c-process) rather than a c-user. It is however easier to intuitively consider the semantics by visualizing a human being interactively carrying out these operations. The semantics do apply equally well to processes operating on behalf of a user, whether interactive or not.

5.1 The INSERT Statement

The INSERT statement executed by a c-user has the following general form, where the c is implicitly determined by the the user's login class.

INSERT	
INTO	$R_c[(A_i[, A_j] \ldots)]$
VALUES	$(a_i[, a_j] \ldots)$

In this notation the rectangular parenthesis denote optional items and the "..." signifies repetition. If the list of attributes is omitted, it is assumed that all the data attributes in R_c are specified. Only data attributes A_i can be explicitly given values. The classification attributes C_i are all implicitly given the value c.

Let t be the tuple such that $t[A_k] = a_k$ if A_k is included in the attributes list in the insert statement, $t[A_k] =$ null if A_k is not in the list, and $t[C_l] = c$ for $1 \leq l \leq n$. The insertion is permitted if and only if:

1. $t[A_1]$ does not contain any nulls.
2. For all $u \in R_c : u[A_1] \neq t[A_1]$.

If so, the tuple t is inserted into R_c and by side effect into all $R_{c'>c}$. This is moreover the only side effect visible in any $R_{c'>c}$.

To illustrate, suppose a U-user wishes to insert a second tuple to the SOD instance given in figure 8. He or she does so by executing the following insert statement.

INSERT	
INTO	SOD
VALUES	('Voyager', 'Exploration', 'Mars')

As a result of the above insert statement, the U-instance of SOD will become as shown in figure 9. This insertion is straightforward and identical to what happens in single-level relations.

On the other hand suppose a S-user wishes to insert the following tuple into the SOD instance of figure 8.

INSERT	
INTO	SOD
VALUES	('Enterprise', 'Spying', 'Rigel')

In this case we can either reject the insert or accept it and allow two tuples with the same apparent key Enterprise to coexist as shown in figure 10. The two tuples in in figure 10 are regarded as pertaining to two distinct entities. We call such situations as *optional polyinstantiations.* Insertion of the secret tuple is not required for closing signaling channels. It is secure to reject such insertions.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

Figure 8: $\mathrm{SOD_U} = \mathrm{SOD_S}$

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Voyager	U	Exploration	U	Mars	U	U

Figure 9: $\mathrm{SOD_U}$

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	S	Spying	S	Rigel	S	S

Figure 10: $\mathrm{SOD_S}$

Starship		Objective		Destination		TC
Enterprise	S	Spying	S	Rigel	S	S

Figure 11: $\mathrm{SOD_S}$

Finally, we illustrate the situation where polyinstantiation is required to close signaling channels. Consider the $\mathrm{SOD_S}$ instance given in figure 11. U-users see an empty instance $\mathrm{SOD_U}$. Suppose a U-user executes the following INSERT statement.

INSERT
INTO SOD
VALUES ('Enterprise', 'Exploration', 'Talos')

This insertion cannot be rejected on the grounds that a tuple with apparent key Enterprise has previously been inserted by a S-user. Doing so would establish a signaling channel from S to U. Therefore by security considerations we are compelled to allow insertion of this tuple. In such cases we say we have *required polyinstantiation*. The effect of this insertion by a U-user is to change $\mathrm{SOD_S}$ from figure 11 to figure 10.

5.2 The UPDATE statement

Our interpretation of the semantics of an update command is close to the one in the standard relational model: An update command is used to change values in tuples that are already present in a relation.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U

Figure 12: $\mathrm{SOD_U}$

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	S	S

Figure 13: $\mathrm{SOD_S}$

UPDATE is a set level operator; i.e., all tuples in the relation which satisfy the predicate in the update statement are to be updated (provided the resulting relation satisfies polyinstantiation integrity). Since we are dealing with multilevel relations, we may have to polyinstantiate some tuples. However, addition of tuples due to polyinstantiation is to be minimized to the extent possible.

The UPDATE statement executed by a c-user has the following general form.

UPDATE R_c
SET $A_i = s_i [, A_j = s_j] \ldots$
[WHERE p]

Here, s_k is a scalar expression, and p is a predicate expression which identifies those tuples in R_c that are to be modified. The predicate p may include conditions involving the classification attributes, in addition to the usual case of data attributes. The assignments in the SET clause, however, can only involve the data attributes. The corresponding classification attributes are implicitly determined to be c.

The intent of the UPDATE operation is to modify $t[A_k]$ to s_k in those tuples t in R_c that satisfy the given predicate p. In multilevel relations, however, we have to implement the intent slightly differently in order to prevent illegal information flows.

5.2.1 Examples of UPDATE Operations

Consider the SOD instances given in figures 12 and 13. Suppose the U-user makes the following update to $\mathrm{SOD_U}$ shown in figure 12.

UPDATE SOD
SET Destination = Talos
WHERE Starship = 'Enterprise'

The changes to $\mathrm{SOD_U}$ in figure 12 and $\mathrm{SOD_S}$ in figure 13 are shown in figures 14 and 15 respectively.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

Figure 14: $\mathrm{SOD_U}$

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	U	Exploration	U	Rigel	S	S

Figure 15: $\mathrm{SOD_S}$

Note that in $\mathrm{SOD_S}$ the Destination attribute for the Enterprise is now polyinstantiated. This is an example of required polyinstantiation which cannot be completely eliminated without introducing covert channels or severely limiting the expressive capability of the database.

Next, suppose starting with the instance $\mathrm{SOD_S}$ of figure 15 a S-user invokes the following update.

```
UPDATE   SOD
SET      Objective = Spying
WHERE    Starship = 'Enterprise' AND
         Destination = 'Rigel'
```

In this case, $\mathrm{SOD_S}$ will change to the instance given in figure 16, *not* to the instance given in figure 17. That is the UPDATE is interpreted as applying only to the second tuple in figure 15 but not to the first tuple. The S-user can go from figure 15 to figure 17 by issuing the following update.

```
UPDATE   SOD
SET      Objective = Spying
WHERE    Starship = 'Enterprise'
```

This UPDATE is interpreted as applying to both tuples of figure 15. The first two tuples of figure 17 result due to polyintantiation of the first tuple of figure 15. The third tuple of figure 17 results due to the normal replacement update of the second tuple of figure 15.

Next, suppose a U-user makes the following update to the relation shown in figure 14. (Assume S-users see the instance given in figure 15.)

```
UPDATE   SOD
SET      Objective = Spying
WHERE    Starship = 'Enterprise'
```

As a consequence of the above update, not only $\mathrm{SOD_U}$ will change from the relation in figure 14 to the one in

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	U	Spying	S	Rigel	S	S

Figure 16: $\mathrm{SOD_S}$

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	U	Spying	S	Talos	U	S
Enterprise	U	Spying	S	Rigel	S	S

Figure 17: $\mathrm{SOD_S}$

figure 18, but $\mathrm{SOD_S}$ will also change from the relation in figure 15 to the one in figure 19. Thus, polyinstantiation integrity is preserved in instances at different security levels. Note in particular how the secret tuple in figure 15 has changed to the secret tuple in figure 19 due to an update by a U-user.

5.2.2 Effect at the User's Access Class

We now formalize and further develop the ideas sketched out above. First consider the effect of an update operation by a c-user on R_c. Let

$$S = \{t \in R_c : t \text{ satisfies the predicate } p\}$$

We describe the effect of the UPDATE operation by considering each tuple $t \in S$ in turn. The net effect is obtained as the cumulative effect of updating each tuple in turn. The UPDATE operation will succeed if and only if at every step in this process polyinstantiation integrity is maintained. Otherwise the entire UPDATE operation is rejected and no tuples are changed. In other words UPDATE has an all-or-nothing integrity failure semantics.

It remains to consider the effect of UPDATE on each tuple $t \in S$. There are two components to this effect. Firstly, tuple t is replaced by tuple t' which is identical to t except for those data attributes which are assigned new values in the SET clause. This is the familiar replacement semantics of UPDATE in a single-level world. In terms of our earlier examples the update of $\mathrm{SOD_U}$ from figure 12 to figure 14 and then to figure 18 illustrates this semantics. The formal definition of the tuple t' obtained by replacement semantics is straightforward as follows.

$$t'[A_k, C_k] = \begin{cases} t[A_k, C_k] & A_k \notin \text{SET clause} \\ <s_k, c> & A_k \in \text{SET clause} \end{cases}$$

Starship		Objective		Destination		TC
Enterprise	U	Spying	U	Talos	U	U

Figure 18: $\mathrm{SOD_U}$

Starship		Objective		Destination		TC
Enterprise	U	Spying	U	Talos	U	U
Enterprise	U	Spying	U	Rigel	S	S

Figure 19: $\mathrm{SOD_S}$

Secondly to avoid covert channels, we may need to introduce an additional tuple t'' to hide the effects of the replacement of t by t' from users at levels below c (c is the level of the user executing the UPDATE). This will occur whenever there is some attribute A_k in the SET clause with $t[C_k] < c$. The idea is that the original value of $t[A_k]$ with classification $t[C_k]$ is preserved in t''. At the same time the core integrity properties of section 3 must also be preserved. To be concrete consider our earlier example of the update of $\mathrm{SOD_S}$ from figure 15 to figure 16. The WHERE clause of the UPDATE statement picks up the second tuple in figure 15 which by replacement semantics gives us the second tuple in figure 16. In this case the unclassified Exploration value of the Objective attribute continues to be available in the first tuple of figure 16 and we need not introduce an additional tuple to hide the effect of this update from U-users. On the other hand suppose the same UPDATE statement, viz.,

```
UPDATE   SOD
SET      Objective = Spying
WHERE    Starship = 'Enterprise' AND
         Destination = 'Rigel'
```

was executed by a S-user in context of figure 13. Prior to the update U-users see the instance in figure 12 and therefore must continue to do so after the update. To achieve this $\mathrm{SOD_S}$ changes from figure 13 to figure 20. The first tuple in figure 20 is the tuple t' dictated by the usual replacement semantics. The second tuple is the t'' tuple introduced to hide the effect of the update from U-users and maintain inter-instance integrity. It should be noted that figure 21 also achieves these two goals. However it does so at the cost of a spurious association between Rigel and Exploration which is avoided in figure 20.

We now give a formal definition of the t'' tuple introduced to close the covert channel. It is defined as follows.

Starship		Objective		Destination		TC
Enterprise	U	Spying	S	Rigel	S	S
Enterprise	U	Exploration	U	null	U	U

Figure 20: $\mathrm{SOD_S}$

Starship		Objective		Destination		TC
Enterprise	U	Spying	S	Rigel	S	S
Enterprise	U	Exploration	U	Rigel	S	S

Figure 21: $\mathrm{SOD_S}$

$$t''[A_k, C_k] = \begin{cases} t[A_k, C_k] & t[C_k] < c \\ <\text{null}, t[A_1]> & t[C_k] = c \end{cases}$$

To summarize each tuple $t \in S$ is replaced by t' and possibly in addition by t'' (if t'' exists). The update is successful if the resulting relation satisfies polyinstantiation integrity. Otherwise the update is rejected and the original relation is left unchanged.

5.2.3 Effect Above the User's Access Class

Next consider the effect of the update operation on $R_{c'>c}$. This of course assumes that the update operation on R_c was successful. Unfortunately, the core integrity properties do not uniquely determine how an update by a c-user to R_c should be reflected in updates to $R_{c'>c}$. Several different options have been proposed [4, 7, 10, 11]. In this paper we will adopt the *minimal propagation rule* [7] which introduces exactly those tuples in $R_{c'>c}$ that are needed to preserve inter-instance property, i.e., put t' and t'' (if t'' exists and survives subsumption) in each $R_{c'>c}$ and nothing else.

Formally, the effect of the update operation is again best explained by focusing on a particular tuple t in S. Let A_k be an attribute in the SET clause such that: (i) $t[C_k] = c$ and (ii) $t[A_k] = x$ where x is non-null. That is the c-user is actually changing a non-null value of $t[A_k]$ at his own level to s_k. Now consider $R_{c'>c}$. Due to polyinstantiation there may be several tuples u in $R_{c'>c}$ which have the same apparent primary key as t (i.e., $u[A_1, C_1] = t[A_1, C_1]$) and match t in the A_k and C_k attributes (i.e., $u[A_k, C_k] = t[A_k, C_k]$). To maintain polyinstantiation integrity (i.e., property 4 of section 3) we must therefore change the value of $u[A_k]$ from x to s_k. This requirement is formally stated as follows.

1. For every $A_k \in$ SET clause with $t[A_k] \neq$ null let

$$U = \{u \in R_{c'>c} : \begin{array}{l} u[A_1, C_1] = t[A_1, C_1] \wedge \\ u[A_k, C_k] = t[A_k, C_k] \end{array} \}$$

Polyinstantiation integrity dictates that we replace every $u \in U$ by u' identical to u except for

$$u'[A_k, C_k] = < s_k, c >$$

This rule applies cumulatively for different A_k's in the SET clause.

This requirement is an absolute one and must be rigidly enforced by the DBMS. The next requirement is imposed by the inter-instance integrity property of section 4.

2. To maintain inter-instance integrity we insert t' and t'' (if it exists and survives subsumption) in $R_{c'>c}$.

This second requirement is a weaker one than the first, in that inter-instance integrity only stipulates what minimum action is required. We can insert a number of additional tuples v in $R_{c'>c}$ with $v[A_1, C_1] = t'[A_1, C_1]$ so long as the core integrity properties are not violated. In particular if t' subsumes the tuple in $\sigma(\{v\}, c)$ inter-instance integrity is still maintained. Minimal propagation makes the simplest assumption in this case, i.e., only t' and t'' are inserted in $R_{c'>c}$ and nothing else is done.

5.3 The DELETE statement

The DELETE statement has the following general form:

DELETE
FROM R_c
[WHERE p]

Here, p is a predicate expression which helps identify those tuples in R_c that are to be deleted. The intent of the DELETE operation is to delete those tuples t in R_c that satisfy the given predicate. But in view of the $\star$-property only those tuples t that additionally satisfy $t[TC] = c$ are deleted from R_c. In order to maintain inter-instance integrity polyinstantiated tuples are also deleted from $R_{c'>c}$.

In particular, if $t[C_1] = c$, then any polyinstantiated tuples in $R_{c'>c}$ will be deleted from $R_{c'>c}$, and so the entity that t represents will completely disappear from the multilevel relation. On the other hand with $t[C_1] < c$ the entity will continue to exist in $R_{t[C_1]}$ and in $R_{c'>t[C_1]}$.

6 FUTURE WORK

In this paper, we have examined the entity integrity requirement and the semantics of various update operations in the context of multilevel relations. These concepts were suitably generalized to deal with polyinstantiation.

This paper is part of an effort to develop better understanding of the interactions between multilevel security and the relational model. This paper points out clearly how subtle this interaction can be.

We believe that there is much more interesting work that remains to be done in this area (see, for example, [6]). In particular we would like to give a complete and formal set of principles that can help with design and implementation of multilevel secure relational DBMSs. Initial steps have been taken in this direction in the present paper, but more remains to be done.

References

[1] Dorothy E. Denning, *Cryptography and Data Security.* Addison-Wesley, Reading, Mass., (1982).

[2] Dorothy E. Denning, Teresa F. Lunt, Roger R. Schell, Mark Heckman, and William R. Shockley, "A multilevel relational data model." *Proc. IEEE Symposium on Security and Privacy*, 220-234 (1987).

[3] Patricia P. Griffiths and Bradford W. Wade, "An authorization mechanism for a relational database system." *ACM Trans. on Database Systems*, (1)3:242-255, (September 1976).

[4] J. T. Haigh, R. C. O'Brien, and D. J. Thomsen, "The LDV Secure Relational DBMS Model." *Database Security IV: Status and Prospects*, Jajodia, S. and Landwehr, C. (editors), North-Holland, 1991, to appear.

[5] Sushil Jajodia and Ravi Sandhu, "Polyinstantiation integrity in multilevel relations." *Proc. IEEE Symposium on Security and Privacy*, 104-115 (May 1990).

[6] Sushil Jajodia and Ravi Sandhu, "Database security: Current status and key issues," *ACM SIGMOD Record,* (19)4:123-126 (December 1990).

[7] Sushil Jajodia, Ravi Sandhu, and Edgar Sibley, "Update semantics of multilevel relations."

Proc. 6th Annual Computer Security Applications Conf., December 1990, pages 103-112.

[8] B. W. Lampson, "A note on the confinement problem." *CACM,* (16)10:613-615, (October 1973).

[9] Carl E. Landwehr, "Formal models for computer security." *ACM Computing Surveys,* (13)3:247-278 (September 1981).

[10] Teresa F. Lunt, Dorothy E. Denning, Roger R. Schell, Mark Heckman, and William R. Shockley, "The SeaView security model." *IEEE Transactions on Software Engineering*, 16(6):593-607 (1990).

[11] Teresa F. Lunt and Donovan Hsieh, "Update semantics for a multilevel relational database." *Database Security IV: Status and Prospects*, Jajodia, S. and Landwehr, C. (editors), North-Holland, 1991, to appear.

[12] Fausto Rabitti, Darrell Woelk, and Won Kim, "A model of authorization for object-oriented and semantic databases." *Proc Conf. on Extending Database Technology*, 231-250, (March 1988).

[13] Ravi Sandhu, Sushil Jajodia, and Teresa Lunt, "A new polyinstantiation integrity constraint for multilevel relations." *Proc. IEEE Workshop on Computer Security Foundations*, 159-165 (June 1990).