Update Semantics for Multilevel Relations*

Sushil Jajodia†*, Ravi Sandhu and Edgar Sibley*

Department of Information Systems
and Systems Engineering
George Mason University
Fairfax, VA 22030

Abstract

In this paper we give a formal operational semantics for update operations on multilevel relations, i.e., relations in which individual data elements are classified at different levels. For this purpose, the familiar INSERT, UPDATE and DELETE operations of SQL are suitably generalized to cope with polyinstantiation. We conjecture that these operations are consistent (or sound) in that all relations which can be constructed will satisfy the basic integrity properties required of multilevel relations. We also conjecture that the operations are complete in that every multilevel relation can be constructed by some sequence of these operations.

1 INTRODUCTION

In a multilevel world of classified information it is inherent that users with different clearances see different sets of facts. The presence of classified information inevitably leads to polyinstantiation, i.e., the simultaneous existence of data objects or attributes which are indistinguishable except for classification. Polyinstantiation arises at all levels of granularity and it will be as fine-grained as the elementary unit of classification. For example, there may be a secret starship called Enterprise in a database along with an unclassified starship which is also named Enterprise. This happens because it is not possible, in general, to prevent creation of the unclassified Enterprise without leaking the fact that a classified Enterprise already exists. At a finer grain, the unclassified Enterprise might have the unclassified destination of Talos while its secret destination is Rigel.

Polyinstantiation does complicate the meaning of multilevel relations relative to relations as ordinarily considered in a single level world. This is unfortunate since much of the appeal of the relational model is due to its intuitive simplicity and economy of concepts. Polyinstantiation is however inevitable and must be confronted [1, 2]. The best we can do is to give as simple a semantics for polyinstantiation as feasible. The semantics of polyinstantiation is reasonably straightforward so long as security classifications are applied to entire tuples ("rows") or attributes ("columns") of a relation. This level of granularity is however cumbersome and artificial for modeling the real world. When classifications are assigned at the level of individual data elements, the semantics of polyinstantiation turns out to be surprisingly subtle [3, 6, 7, 9, 11]. As work on this topic has progressed it has become increasingly evident that a formal consideration of update operations is necessary to fully articulate the meaning of polyinstantiation.

Our principal objective in this paper is to give a simple operational semantics for update operations on multilevel relations. In developing the semantics we are motivated by the following principles.

1. The update operations should be as close to standard SQL as possible.

2. An update should result in polyinstantiation only when absolutely required for closing signaling channels (or optionally for deliberately establishing cover stories). Moreover, the fewest possible tuples should be introduced in such cases.

In the latter requirement we deliberately use the term signaling channel rather than covert channel. A signaling channel is a means of information flow which is inherent in the model and will therefore occur in *every* implementation of the model. A covert channel on

*This research was supported (partially) by the Center for Excellence in Command, Control, Communications, and Intelligence at George Mason University. The Center's general research program is sponsored by the Virginia Center for Innovative Technology, MITRE Corporation, the Defense Communications Agency, CECOM, PRC/ATI, ASD (C3I), TRW, AFCEA, and AFCEA NOVA.

†Also with Secure Technology Center, The MITRE Corporation, 7525 Colshire Drive, McLean, VA 22102-3481.

TH0351-7/90/0000/0103$01.00 © 1990 IEEE

the other hand is a property of a specific implementation and not a property of the model. That is, even if the model is free of signaling channels, a specific implementation may well contain covert channels due to implementation quirks.

This paper is organised as follows. In Section 2, we begin by giving basic definitions related to multilevel relations, and then we state four integrity requirements which we feel must be satisfied by all multilevel relations. In Section 3, we discuss in detail the three update (insert, update, and delete) operations in the context of multilevel relations. Finally, the conclusion is given in Section 4.

2 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* R 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 instance* 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.

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 *candidate key* of a relation is a minimal set of attributes on which all other attributes are functionally dependent. It is minimal in the sense that no attribute can be discarded without destroying this property. It is guaranteed that a candidate key always exists, since in the absence of any functional dependencies it consists of the entire set of attributes. There can be more than one candidate key for a relation with a given collection of functional dependencies.

The *primary key* of a relation is one of its candidate keys which has been specifically designated as such. The primary key serves the purpose of selecting a specific tuple from a relation instance as well as of linking relations together. The standard relational model incorporates two application independent integrity rules, called *entity integrity* and *referential integrity*, respectively to ensure these purposes are properly served. Entity integrity in the standard relational model simply requires that no tuple in a relation instance can have null values for any of the primary key attributes. This property guarantees that each tuple will be uniquely identifiable. In this paper our focus is on single relations, so referential integrity is not relevant.

Moving on to a multilevel world, we follow the lead of SeaView in extending the standard relation model to define a *multilevel relation* R as consisting 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 $[\mathrm{lub}\{L_i : i = 1 \ldots n\}, \mathrm{lub}\{H_i : i = 1 \ldots n\}]$. □

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 $a_i =$ 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 also require that c_i be defined even if a_i is null, i.e., a classification attribute cannot be null. □

Since tc is computed from the other classification attributes, it is included or omitted as convenient. We use the notation $t[A_i]$ to mean the value of the A_i attribute in tuple t, and similarly for $t[C_i]$ and $t[TC]$.

Because a multilevel relation has different instances at different access classes it is inherently more complex than a standard relation. It is most important to understand what constitutes the full primary key of a multilevel relation. In a standard relation the definition

[‡]In many cases it is useful to have a group of *uniformly classified* data attributes. Our definition easily extends to such cases by treating each A_i as a group of data attributes. This extension requires straightforward, but tedious, modifications to the statement of our update semantics which are stated in this paper in terms of the A_i's being individual data attributes.

of candidate 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. Rather than trying to resolve this complex issue here, we follow the lead of SeaView and assume 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. Henceforth we understand the term primary key as synonymous with apparent primary key.

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. SeaView extends this property to multilevel relations as follows.

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

1. $A_i \in AK \Rightarrow t[A_i] \neq$ 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 extension 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. These requirements seem quite reasonable. Further intuitive justification for them is given in [1, 5].

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] Instance R_c of R satisfies null integrity if and only if both of the following conditions are true.

1. For all $t \in R_c$, $t[A_i] =$ null $\Rightarrow t[C_i] = t[C_{AK}]$, i.e., nulls are classified at the level of the key.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U

Figure 1: SOD_U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	S	S

Figure 2: SOD_S

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

Figure 3: Violation of Null Integrity

2. We say that tuple t ***subsumes*** tuple s if for every attribute A_i, either $t[A_i, C_i] = s[A_i, C_i]$ or $t[A_i] \neq$ null and $s[A_i] =$ 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.

Throughout this paper, we use the following example to provide the motivation and the illustrations of the main ideas. We consider a multilevel relation scheme SOD consisting of three data attributes Starship, Objective, and Destination, with Starship as the apparent primary key. We will also be using the standard security hierarchy with U (unclassified) < C (confidential) < S (secret).

A typical relation instance for SOD is given in Figure 1. 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 2 rather than the one given in Figure 3; since the first tuple in Figure 3 is subsumed by the second tuple.

The next property is concerned with consistency between relation instances at different access classes.

Property 3 [Inter-Instance Integrity] R satisfies inter-instance integrity if and only if for all states and 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 2 while a U-user will see the filtered instance given in Figure 1. It is evident that

$$\begin{aligned} \sigma(R_c, c) &= R_c \\ \sigma(\sigma(R_c, c'), c'') &= \sigma(R_c, c'') \text{ for } c > c' > c'' \end{aligned}$$

as one would expect from the intuitive notion of filtering.

Finally we have the following polyinstantiation integrity constraint which prohibits polyinstantiation within a single access class.

Property 4 [Polyinstantiation Integrity] R satisfies polyinstantiation integrity 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.

We regard property 4 as the formal definition of the informal notion of AK as the user-specified primary key. The effect of polyinstantiation integrity is to rule out instances such in Figure 4, where there are two values labeled U for the Objective attribute of the Enterprise. Note that for single level relations C_{AK} and C_i will be equal to the same constant value in all tuples. In this case property 4 amounts to saying $AK \rightarrow A_i$, which is precisely the definition of primary key in relational theory.

3 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. The effect of these operations, however, on multilevel relation instances is sometimes not as straightforward as in the case of standard (single-level) relations because of two factors: (1) star-property must be preserved which prevents any write downs, and (2) signaling channels must be avoided.

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

Figure 4: SOD_S

Let $R(A_1, C_1, \ldots, A_n, C_n, TC)$ be a multilevel relation scheme. In order to simplify the notation, we understand A_1 as equivalent to AK from now on, i.e., A_1 is the apparent primary key.

Consider a user logged on at access class c. For the sake of brevity we also refer to such a user as a c-user. 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.

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

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 his actions cannot impact any $R_{c'<c}$, from the user's point of view 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 $R_{c'>c}$. In general this may require the insertion, update or deletion of some tuples in $R_{c'>c}$ whose tuple class strictly dominates c. Moreover there may be several different ways to do this while maintaining inter-instance integrity. This fact complicates the semantics of insert, update and delete operations; underscoring the need for a formal definition.

It is important to realize that the general behavior outlined above is a necessary consequence of simple-security, the star-property and inter-instance integrity. The precise articulation of this behavior is given on a statement-by-statement basis in the rest of this section.

In all cases operations performed by a c-user[§] on R_c have no effect on any $R_{c'<c}$ or $R_{c'\sim c}$. The direct effect of the operation is on R_c. However each operation also indirectly effects every $R_{c'>c}$. The latter effect is only partly determined by the core integrity properties of section 2 leaving room for at least the two different interpretations identified in section 3.2.3.

3.1 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]...)]
VALUES    (a_i[, a_j]...)
```

In this notation the rectangular parenthesis denote optional items and the "..." signifies repetition. If the list of attributes in omitted, it is assumed that all the data attributes in R_c are specified. Moreover, note that 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, $t[A_k] = \text{null}$ 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 $R_{c'>c}$.

Thus, the insert statement works in a straightforward manner. A c-user can insert a tuple t in R_c if R_c does not already have a tuple with the same apparent primary key value and key class as t. In the inserted tuple, the access classes of all data attributes as well as the tuple class are set to c.

To illustrate, suppose a U-user wishes to insert a second tuple to the SOD instance given in Figure 5. He does so by executing the following insert statement.

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

[§]Strictly speaking in all cases we should be saying c-subject rather than 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.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

Figure 5: $\text{SOD}_\text{U} = \text{SOD}_\text{S}$

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

Figure 6: SOD_U

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

Figure 7: SOD_S

Starship		Objective		Destination		TC
Enterprise	S	Spying	S	Rigel	S	S

Figure 8: SOD_S

As a result of the above insert statement, the U-instance of SOD will become as shown in Figure 6. 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 5.

```
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 7. The two tuples in in Figure 7 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. We believe that whether or not optional polyinstantiation occurs is best specified as a property of the relation by the Database Administrator or perhaps as part of the INSERT statement. The prohibition of optional polyinstantiation as an integral part of a data model is in our opinion needlessly restrictive.

Finally, we illustrate the situation where polyinstantiation is required to close signaling channels. Consider the SOD_S instance given in Figure 8. U-users see an empty instance SOD_S. 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 SOD_S from Figure 8 to Figure 7.

Note that we have shown two different scenarios for arriving at the SOD_S instance of Figure 7, one based on optional polyinstantiation and the other on required polyinstantiation. This theme of optional versus required polyinstantiation occurs repeatedly through our discussion. As we have demonstrated above the net result of optional polyinstantiation can be achieved by required polyinstantiation. We must therefore give a sensible semantics to the net result independent of whether it was reached by optional or by required polyinstantiation.

3.2 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. 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. As we see it, there is one and only one reason why we must polyinstantiate: to prevent signaling channels or establish cover stories; otherwise, we must not polyinstantiate!

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

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

Here, s_k is a scaler 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. In particular if $t[C_k] < c$ the star-property prevents us from actually updating $t[A_k]$ in place, since this would amount to a write down. We must instead keep both values of A_k.¶ This is achieved by creating a new tuple t' in R_c which is identical to t except for such attributes A_k in the UPDATE statement. As discussed earlier the effect of the update must also be propagated up to $R_{c'>c}$ in a consistent manner.

3.2.1 Examples of UPDATE Operations

We now illustrate the semantics of UPDATE by giving several examples. Following this we will give the formal definitions and more examples.

Consider the SOD instances given in Figures 9 and 10. Suppose the U-user makes the following update to SOD_U shown in Figure 9.

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

The changes to SOD_U in Figure 9 and SOD_S in Figure 10 are shown in Figures 11 and 12 respectively. Note that in 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 signaling channels or severely limiting the expressive capability of the database. Also note that the two tuples for the Enterprise in Figure 12 refer to the same real-world entity unlike the two tuples of Figure 7 which refer to two distinct entities.

Next, suppose starting with the instance SOD_S shown in Figure 12 a S-user invokes the following update.

¶This is an example of optional polyinstantiation so another sensible alternative is to reject the update. As argued earlier the rejection of optional polyinstantiation should not be hard-wired into the data model. We reiterate and emphasize the point that even if we insist on always rejecting optional polyinstantiation we must still cope with required polyinstantiation. For UPDATE required polyinstantiation arises due to updates by c-users of null values in tuples with $t[A_1] = c$.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U

Figure 9: SOD_U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	S	S

Figure 10: SOD_S

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

Figure 11: SOD_U

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

Figure 12: SOD_S

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

Figure 13: SOD_S

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

Figure 14: SOD_S

Starship		Objective		Destination		TC
Enterprise	U	Spying	U	Talos	U	U

Figure 15: SOD_U

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

Figure 16: SOD_S

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

In this case, the SOD_S will change to the instance given in Figure 13, *not* to the instance given in Figure 14. This follows from our underlying philosophy: we need to polyinstantiate to either close a signaling channel or provide a cover story.

Next, suppose a U-user makes the following update to the relation shown in Figure 11. (Assume S-users see the instance given in Figure 12.)

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

As a consequence of the above update, not only SOD_U will change from the relation in Figure 11 to the one in Figure 15, but SOD_S will also change from the relation in Figure 12 to the one in Figure 16. Thus, polyinstantiation integrity is preserved in instances at different security levels. Note in particular how the secret tuple in Figure 12 has changed to the secret tuple in figure 16 due to an update by a U-user.

3.2.2 Effect of UPDATE 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 SOD_U from Figure 9 to Figure 11 and then to Figure 15 illustrates this semantics. The formal definition of the tuple t' obtained by replacement semantics is straightforward as follows.

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

Secondly to avoid signaling 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 2 must also be preserved. To be concrete consider our earlier example of the update of SODs from Figure 12 to Figure 13. The WHERE clause of the UPDATE statement picks up the second tuple in Figure 12 which by replacement semantics gives us the second tuple in Figure 13. In this case the unclassified Exploration value of the Objective attribute continues to be available in the first tuple of Figure 13 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 10. Prior to the update U-users see the instance in Figure 9 and therefore must continue to do so after the update. To achieve this SOD_S changes from Figure 10 to Figure 17. The first tuple in Figure 17 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 18 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 17.

We now give a formal definition of the t'' tuple introduced to close the signaling channel. From the preceding discussion it might appear that in the definition one has to consider tuples other than the tuple t which is being updated. Fortunately this complication can be avoided because the t'' tuple will be subsumed by existing tuples whenever appropriate. The t'' tuple is defined as follows.

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

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

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

Figure 17: SOD_S

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

Figure 18: SOD_S

3.2.3 Effect of UPDATE 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. The effect of the update operation is again best explained by focusing on a particular tuple t in S.

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

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

   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.

2. To maintain inter-instance integrity we need at the minimum to insert t' and t'' (if it exists) in $R_{c'>c}$.

The first requirement is an absolute one and must be rigidly enforced by the DBMS. The second requirement is, however, a weaker one in that inter-instance integrity only stipulates what minimum action is required. We can however 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.

In short the core integrity properties do not uniquely determine how an update by a c-user to R_c should be reflected in updates to $R_{c'>c}$. There are at least two reasonable approaches to resolving this issue, both of which should be available as options.

1. *Minimal propagation*: introduce only the minimum necessary to maintain inter-instance integrity, i.e., put t' and t'' (if it exists) in each $R_{c'>c}$ and nothing else.

2. *Interpreted propagation*: introduce exactly those tuples in $R_{c'>c}$ dictated by the update statement in question. For this purpose consider the set

$$Q = \{q \in R_{c'>c} : q[A_1, C_1] = t[A_1, C_1] \wedge q \text{ satisfies } p\}$$

where p is the predicate in the WHERE clause of the UPDATE statement. For each q insert the following tuple in $R_{c'>c}$

$$q'[A_k, C_k] = \begin{cases} q[A_k, C_k] & A_k \notin \text{SET clause} \\ < s_k, c > & A_k \in \text{SET clause} \end{cases}$$

To illustrate the difference between the minimal and interpreted propagation rules, assume that SOD_U and SOD_C are identical as shown in Figure 19 while SOD_S is as shown in Figure 20. Suppose now that a C-user makes the following update to SOD.

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

As a consequence of the above update SOD_C will change to the relation given in Figure 21. (SOD_U remains unchanged as in Figure 19.) The exact change to SOD_S depends on the propagation rule. Under the minimal propagation rule, SOD_S will change from Figure 20 to Figure 22, while under the interpreted propagation rule, the relation in Figure 23 will result. The basic difference is that with minimal propagation the newly inserted confidential data element is not associated with any secret data whereas with interpreted propagation it is.

3.3 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. All tuples t in

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

Figure 19: $SOD_U = SOD_C$

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

Figure 20: SOD_S

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	U	Spying	C	Talos	U	C

Figure 21: Updated SOD_C

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

Figure 22: Updated SOD_S by Minimal Propagation

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

Figure 23: Updated SOD_S by Interpreted Propagation

R_c that satisfy the predicate p are deleted. If $t[C_1] = c$, then any polyinstantiated tuples in $R_{c'>c}$ will be deleted from $R_{c'>c}$ and the entity will completely disappear from the multilevel relation. On the other hand with $t[C_1] < c$ the entity will continue to exist in some $R_{c'<c}$ and possibly in R_c itself. At the same time to maintain inter-instance integrity we may need to delete some polyinstantiated tuples from $R_{c'>c}$. The precise set of tuples which need to be deleted is open to different interpretations. As for the UPDATE operation we can do so in a minimal or interpreted manner. The details are omitted for now.

4 CONCLUSION

In this paper we have examined the semantics of various update operations in the context of multilevel relations. To this end, the familiar INSERT, UPDATE and DELETE operations were suitably generalized to deal with polyinstantiation.

In terms of future work, we intend to consider the issue of implementing these update semantics in a kernelized DBMS. The existing decomposition and recovery algorithms [3, 6, 8, 9, 10] do not exhibit the update semantics proposed in this paper. Thus, they need to suitably modified.

It also remains to be shown that our update semantics are consistent (or sound) and complete. Consistency requires that all relations which can be constructed will satisfy the basic integrity properties required of multilevel relations. Completeness requires that any multilevel relation instance satisfying the four core integrity properties (given in Section 2) can be realized by some sequence of update operations. We conjecture that this is indeed the case, particularly in regard to the interpreted propagation rule. The minimal propagation rule we know to be incomplete.

Acknowledgement

We are indebted to John Campbell, Joe Giordano and Howard Stainer for their support and encouragement making this work possible. The opinions expressed in this paper are of course our own and should not be taken to represent the views of these individuals.

References

[1] Denning, D.E., Lunt, T.F., Schell, R.R., Heckman, M., and Shockley, W.R. "A Multilevel Relational Data Model." *IEEE Symposium on Security and Privacy*, 220-234 (1987).

[2] Denning D.E. "Lessons Learned from Modeling a Secure Multilevel Relational Database System." In Landwehr, C.E. (Editor) *Database Security: Status and Prospects*, North-Holland, 35-43 (1988).

[3] Denning, D.E., Lunt, T.F., Schell, R.R., Shockley, W.R. and Heckman, M. "The SeaView Security Model." *IEEE Symposium on Security and Privacy*, 218-233 (1988).

[4] Department of Defense National Computer Security Center. *Department of Defense Trusted Computer Systems Evaluation Criteria.* DoD 5200.28-STD, (1985).

[5] Gajnak, G.E. "Some Results from the Entity-Relationship Multilevel Secure DBMS Project." *Aerospace Computer Security Applications Conference*, 66-71 (1988).

[6] Jajodia, S. and Sandhu, R.S. "Polyinstantiation Integrity in Multilevel Relations." *IEEE Symposium on Security and Privacy*, Oakland, California, 104-115 (1990).

[7] Jajodia, S. and Sandhu, R.S. "A Formal Framework for Single Level Decomposition of Multilevel Relations." *IEEE Workshop on Computer Security Foundations*, Franconia, NH, 152-158 (1990).

[8] Jajodia, S. and Sandhu, R.S. "Polyinstantiation Integrity in Multilevel Relations Revisited." *IFIP WG11.3 Workshop on Database Security*, Halifax, U.K. (1990).

[9] Lunt, T.F., Denning, D.E., Schell, R.R., Heckman, M. and Shockley, W.R. "The SeaView Security Model." *IEEE Transactions on Software Engineering* 16(6):593-607 (1990).

[10] Lunt, T. and Hsieh, D. "Update Semantics for a Multilevel Relational Database." *IFIP WG11.3 Workshop on Database Security*, Halifax, U.K. (1990).

[11] Sandhu, R.S., Jajodia, S. and Lunt, T. "A New Polyinstantiation Integrity Constraint for Multilevel Relations." *IEEE Workshop on Computer Security Foundations*, Franconia, NH, 159-165 (1990).