Proceedings of IEEE Symposium on Security and Privacy, Oakland, California, May 8-10, 1995

The Semantics and Expressive Power of the MLR Data Model

Fang Chen and Ravi S. Sandhu[*]
Center for Secure Information Systems
&
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444

Abstract

In this paper, we define the Multilevel Relational (MLR) data model for multilevel relations with element-level labeling. This model builds upon prior work of numerous authors in this area, and integrates ideas from a number of sources. A new data-based *semantics is given to the MLR data model which combines ideas from SeaView, belief-based semantics and LDV model, and has the advantages of both eliminating ambiguity and retaining upward information flow. The resulting model is simple, unambiguous and powerful. It has five integrity properties and five operation statements for manipulating multilevel relations. In order to support this integration, we introduce several new concepts as well as redefine several old ones. The expressive power of the MLR model is also discussed in this paper, and is compared with several other models. We also address some issues in converting the MLR model to tuple-level labeling, including both scheme mapping and operation interpretation.*

1 Introduction

There are many MLS relational data models in the literature, for example SeaView [3, 7], LDV [4], and those proposed by Sandhu-Jajodia [12, 13], Jajodia-Sandhu [5, 6], Smith-Winslett [14], etc. We do not believe that any of these models is right or wrong. To us, they are just different. We would like to reconcile these differences by unifying as much as we can, to establish a simple, unambiguous and powerful multilevel relational data model.

In this paper, we define the Multilevel Relational (MLR) data model for multilevel relations with element-level labeling. A new *data-based* semantics for both data and operations is also given. MLR is substantially based on the data model proposed by Sandhu and Jajodia [13]. However, we integrate many ideas from SeaView, LDV as well as belief-based semantics [14]. To complete this integration, several new concepts are introduced and several old ones are redefined.

The MLR data model is a simple, unambiguous and powerful model. It contains five integrity properties and five operation statements for manipulating multilevel relations. Moreover, it retains upward information flow and has no semantic ambiguity.

The expressive power of the MLR model is discussed in depth by comparing it with several other models. Also, by converting the scheme and operations to tuple-level labeling data model, we can see that the MLR data model can be grounded on most of the current commercial multilevel database products, as long as some tools are developed to support this conversion. Note that tuple-level labeling is a natural approach for DBMS vendors, because the tuple is the basic storage and retrieval unit in typical DBMS implementations.

We have developed proofs of soundness, completeness and security for the MLR model. These proofs respectively show that any of the provided operation statements will keep the database state legal (i.e., satisfying all integrity properties), every legal database state can be constructed, and the MLR data model is secure, in that all information flow is upwards in the security lattice. The proofs are omitted here because of lack of space, and are given in [2].

The rest of this paper is organized as follows. In section 2 we explain why we want to integrate several models to establish the MLR data model. In sections 3 and 4 we define the basic model and give the data semantics. The expressive power of the model

[*]The work of both authors is partially supported by the National Security Agency through contract MDA904-94-C-6119. We are indebted to Blaine Burnham, Dorothy Darnauer and Darrel Sell for their support and encouragement in making this work possible.

is discussed in section 5. Section 6 addresses the data manipulation operations of the model. Section 7 shows how these manipulation operations can be converted to tuple-level labeling model. In section 8, we summarize our major contributions.

2 Motivation

As mentioned earlier, MLR is substantially based on the data model proposed by Sandhu and Jajodia [13]. Many aspects of that model can be traced back to the SeaView work [3]. The most significant difference is the requirement that there can be at most one tuple in each access class for a given entity. This gives us the simplicity of tuple-level labeling, combined with the flexibility of element-level labeling. There are also several other subtle, but very important differences in the precise formulation of various properties.

However, there are still some problems left unsolved in the Sandhu-Jajodia model of [13]. Two major problems are semantic ambiguity and operational incompleteness. Also, the No Entity Polyinstantiation Integrity property of [13] may cause some downward information leakage.

To illustrate the semantic ambiguity problem, consider the following relation SOD(SHIP, OBJ, DEST) where SHIP is the primary key and the security classifications are assigned at the granularity of individual data elements. OBJ and DEST are abbreviations for OBJECTIVE and DESTINATIONS, respectively. TC is an abbreviation for TUPLE-CLASS. The label in the TC attribute applies to the entire tuple. ENT is an abbreviation for Enterprise.

SHIP		OBJ		DEST		TC
Ent	U	Spying	S	Talos	U	S
Ent	U	Exploration	U	Talos	U	U

What is the data accepted by TS-subjects? In the Sandhu-Jajodia model, absence of a TS-tuple means there is no additional data at this level. However, there are both U- and S-tuples existing in this relation. Do TS-subjects take the values from S-tuple or U-tuple? Is there any necessity to force them to choose the higher one? Are there situations in which a TS-subject should accept values from the U-tuple rather than the S-tuple?

Taking another more general example, let M_1 and M_2 be incomparable labels whose least upper bound is S and greatest lower bound is U. Consider

SHIP		OBJ		DEST		TC
Ent	U	Mining	M_1	Talos	U	M_1
Ent	U	Spying	M_2	Talos	U	M_2
Ent	U	Exploration	U	Talos	U	U

Which OBJ value is accepted by S-subjects? Mining or Spying or even Exploration? Again, is it necessary to force them to accept data from any specific level?

As for the operational incompleteness problem, again let M_1 and M_2 be incomparable labels whose least upper bound is S and greatest lower bound is U. How can a tuple whose individual classification attributes are at, say, U, M_1, and M_2 be instantiated by an S-subject? In other words, how can an S-subject add tuples like the following.

SHIP		OBJ		DEST		TC
Ent	U	Mining	M_1	Sirius	M_2	S

In order to solve these problems we need to integrate ideas from several models to establish the new MLR data model. There will be no semantic ambiguity, no operational incompleteness, and no downward information leakage in the MLR data model.

3 The Basic Model

First we formally define the basic model. Then we give a data interpretation to the model as a part of the data semantics. After that we discuss the practicability of the model.

3.1 Model Definition

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

Definition 3.1 [RELATION SCHEME] A multilevel *relation scheme* is denoted by $R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$, 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 domains of C_i are specified by a range $[L_i, H_i]$, $H_i \geq L_i$, which defines a sub-lattice of access classes ranging from L_i up to H_i. The domain of TC is $\cup_{i=1}^{n}([L_i, H])$, where H is system high. □

Definition 3.2 [RELATION INSTANCE] A *relation instance*, denoted by $r(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$, 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 $a_i \in D_i$ and $c_i \in [L_i, H_i]$ or $a_i = \text{null}$ and $c_i \in [L_i, H_i] \cup \text{null}$, and $tc \geq \text{lub}\{c_i \mid c_i \neq \text{null} : i = 1 \ldots n\}$. Here lub denotes the least upper bound. □

We assume that there is a user-specified *apparent primary key* AK consisting of a subset of the data attributes A_i. In general AK will consist of multiple attributes. In order to simplify our notation, we use A_1 as synonymous to AK, i.e., A_1 and AK both denote the apparent primary key. We also assume that the relation scheme is itself unclassified (or, more generally, classified at the greatest lower bound of L_i, $i = 1 \ldots n$). A tuple whose tuple class is c is said to be a c-tuple, while a subject whose clearance is c is said to be a c-subject. Similarly, each A_i can also be seen as a group of attributes having identical classification level.

There are two subtle differences in these definitions, relative to [13]. Firstly, the domain of TC is different from that in [13], which is [lub{ $L_i : i = 1 \ldots n$}, lub{ $H_i : i = 1 \ldots n$}]. Secondly, the model of [13] does not allow classification attributes to be null. These changes are primarily for the INSERT semantics (section 6.2), because now

SHIP		OBJ		DEST		TC
Ent	U	Exploration	U	null	null	U

can be inserted into the relation even in case that the domain of the classification attribute for DEST needs to be limited to [S, TS].

Also, in MLR every relation has only one relation instance at any time. Subjects at different levels have different views of the instance. Previous models have defined a relation as having a different relation instance at each level. This modification is simply for convenience of semantic description.

Same as [13], we define $tc \geq \text{lub}\{c_i \mid c_i \neq \text{null} : i = 1 \ldots n\}$, by which

SHIP		OBJ		DEST		TC
Ent	U	Exploration	U	Talos	U	S
Ent	U	Exploration	U	Talos	U	U

is allowed and has different meanings from

SHIP		OBJ		DEST		TC
Ent	U	Exploration	S	Talos	S	S
Ent	U	Exploration	U	Talos	U	U

In fact, the former one says S-subjects borrow from U-subjects the data currently owned by U-subjects, and the data could be changed by U-subjects; whereas the latter one means S-subjects have their own data for OBJ and DEST, which could not be changed by U-subjects, and the value equivalence in OBJ and DEST is just by coincidence. The data interpretation is fully discussed in section 3.2.

The following definition is directly taken from the standard relational data model.

Definition 3.3 [DATABASE STATE] A *database* is a collection of relations. A *database state* is a collection of all relation instances of a database at a particular time. □

3.2 Data Interpretation

The intuitive ideas of our *data-based* semantics are as follows.

1. The data accepted by subjects at one level consist of two parts: the data owned by them and the data borrowed from lower-level subjects. The latter one can be changed by the lower-level subjects who are owning them.

2. The data a subject can see are those accepted by subjects at its level or at the levels below it.

3. For an entity, c-tuple contains all the data accepted (either owned or borrowed) by c-subjects. Absence of a c-tuple means the existence of the entity is not accepted by c-subjects.

These ideas come from combining belief-based semantics and LDV model. In sections 4 and 6 we will see how these ideas lead to a complete semantics of a data model.

We now give a formal description of the above intuitive ideas. For all instances $r(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$ and for all tuples $t \in r$, the data are interpreted as follows.

1. Apparent Primary Key A_1 and its Classification Attribute C_1
   - $t[A_1, C_1]$ identifies the entity and also gives the class level of the entity.
   - $t[C_1] = c_1$ means the entity is created by a c_1-subject and can only be deleted by c_1-subjects.

2. Tuple-Class Attribute TC
   - $t[TC] = tc$ with $t[C_1] = c_1$ means that
     - t is added by a tc-subject and all data in t are accepted by tc-subjects.
     - t can only be seen by subjects with level $c' \geq tc$. In other words, all a c'-subject can see are tuples t' with $t'[TC] \leq c'$.

- t can be deleted either by tc-subjects, or by c_1-subjects in cases such as the entire entity is deleted.

- When $t[TC] = t[C_1]$, t is the *base tuple* of the entity, which means all tuples $t' \in r$ such that $t'[A_1, C_1] = t[A_1, C_1]$ are based on t, and t can only be deleted when the entire entity is to be deleted.

3. Data Attribute A_k and Classification Attribute C_k $(2 \leq k \leq n)$

   - $t[A_k, C_k]$ with $t[C_k] = c_k$ and $t[TC] = tc$ indicates that
     - the data $t[A_k]$ accepted by tc-subjects is currently owned by c_k-subjects.
     - $t[A_k, C_k]$ can be maintained either by c_k-subjects or by tc-subjects.
   - When $t[C_k] < t[TC]$, $t[A_k] \neq$ null is borrowed from the $t'[A_k]$ of t' which has $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] = t'[C_k] = t[C_k]$, and is subject to change when $t'[A_k, C_k]$ is changed or t' is deleted.

4. Null Value

   - $t[A_k, C_k]$ =[null, c_k] $(c_k < tc)$ means for attribute A_k, tc-subjects are expecting to borrow data owned by c_k-subjects, however, no data is currently owned by them.
   - Both $t[A_k, C_k]$ =[null, null] and $t[A_k, C_k]$ =[null, tc] means for A_k no data is available at level tc. The [null, null] case applies when $tc \notin [L_k, H_k]$.

3.3 Practicability

As we have seen, the data-based semantics takes the following ideas from belief-based semantics: For an entity, absence of a c-tuple means the entity is not accepted by c-subjects. In other words, in order to reference an entity, c-subjects should add a c-tuple of the entity into the relation first, even though all the data accepted by c-subjects are belonging to the subjects below c. Actually, this is the crucial point to eliminate semantic ambiguity.

This requirement would be quite acceptable if 95% entities had different data at different levels. Unfortunately, often only, say, 5% entities will have secret data at levels higher than unclassified; which means for the other 95% entities, high level tuples just repeat unclassified data. If there are m levels higher than unclassified, in order to reference unclassified data at every level, all these data are logically repeated m times. A naive implementation would not just waste space but also waste labor time, because the repetition is explicitly done by subjects at each level.

By this consideration, we are forced to think about some practical issues of the physical model under this reasonable logical model. Fortunately there are several techniques that can be used to deal with these problems.

To avoid wasting space we can physically (not logically) expand the tuple-class attribute TC to be a tuple-class attribute set, containing several levels whose data are exactly the same in all $A_1, C_1, \ldots, A_n, C_n$. Hence, instead of keep several repeated tuples in the database, we can physically just keep one tuple and indicate all levels that accept it in the TC set. For example, $(a_1, c_1, \ldots, a_n, c_n, \{tc_1, \ldots, tc_m\})$ can stand for $(a_1, c_1, \ldots, a_n, c_n, tc_1), \ldots, (a_1, c_1, \ldots, a_n, c_n, tc_m)$.

Saving labor time can be achieved by allowing subjects, or the database administrator, to set some defaults. For example, c-subjects can set defaults such as for all entities in R, all data owned by c'-subjects $(c' < c)$ are accepted. This is to say, whenever a c'-subject insert an entity into R, a c-tuple of the entity will be automatically created and all data of the c-tuple are borrowed from c'-subjects (possibly, we can just put c into the TC set of the c'-tuple). For single level relations, i.e., for any i, j such that $1 \leq i, j \leq n$, $L_i = L_j = H_i = H_j$, this approach could be very useful. For example, a single level relation could be used to keep common knowledge such as the city name, location, area, climate, etc, which can be used by subjects at any level.

It is easy to see that there are many variations of these two basic approaches. Physical issues may also depend on the storage strategies used to construct the DBMS. Since the MLR model is a logical data model, further discussion about physical issues is outside the scope of this paper. Our objective here is to simply make the feasibility argument sketched out above.

4 Integrity Properties

There are five integrity properties in the MLR data model, in which Entity Integrity and Foreign Key Integrity are taken from the original SeaView model; Polyinstantiation Integrity and Referential Integrity are significantly redefined by us; and Data-Borrow Integrity is newly introduced.

In particular, the Polyinstantiation Integrity given here is much more general than that of either SeaView or Sandhu-Jajodia model, in that it takes care of both entity polyinstantiation and element polyinstantiation.

4.1 Entity Integrity

The Entity Integrity property was first proposed by SeaView, and has stayed unchanged in most work since then.

Property 1 [Entity Integrity (EI)] Let AK be the apparent primary key of R. An instance r of a multilevel relation R satisfies entity integrity if and only if for all $t \in r$

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 primary key). □

The first requirement was exactly the definition of entity integrity from the standard relational model, and ensures that no tuple in r has a null value for any attribute in AK. The second requirement says that all attributes in AK have the same classification in a tuple. 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.

4.2 Polyinstantiation Integrity

Polyinstantiation Integrity has been required by many models. The definition given here is much more general than previous ones. It is the first one that concerns both entity polyinstantiation and element polyinstantiation in a unified manner.

Property 2 [Polyinstantiation Integrity (PI)] An instance r of a multilevel relation R satisfies polyinstantiation integrity if and only if for $1 \leq i \leq n$,

1. $A_1, TC \rightarrow C_i$
2. $A_1, C_1, C_i \rightarrow A_i$. □

The second requirement is the original polyinstantiation integrity required by both SeaView and Sandhu-Jajodia model, which says the real primary key of the relation is $A_1, C_1, C_2, \ldots, C_n$. The first requirement is derived from the following two (as well as the second one above).

- $A_1, TC \rightarrow C_1$;
- $A_1, C_1, TC \rightarrow A_i, C_i$.

Here, the latter is called tuple-class polyinstantiation integrity in [10][1], which says that every entity in a relation can have at most one tuple for every access class; whereas the former is newly introduced and is to be discussed in detail.

In fact, this new property can also be called as entity polyinstantiation integrity. The intuitive idea of this property is that there could be several entities in a relation with the same AK value, but subjects at any security level can accept at most one entity with that AK value. For example,

SHIP		OBJ		DEST		TC
Ent	U	Spying	TS	Talos	U	TS
Ent	S	Mining	S	Rigel	S	S
Ent	U	Exploration	U	Talos	U	U

is allowed, because at each level, either TS, S or U, subjects only accept one entity with AK value being Ent. However,

SHIP		OBJ		DEST		TC
Ent	S	Spying	S	Rigel	S	S
Ent	U	Mining	S	Talos	U	S
Ent	U	Exploration	U	Talos	U	U

is not allowed, because there are two entities with the same AK value Ent, (Ent, S) and (Ent, U), accepted at level S. S-subjects could choose either of them, but not both, to avoid semantic confusion.

This is very different from the No Entity Polyinstantiation Integrity of [13]. In our case, there can be no downward information leakage, because entity polyinstantiation is allowed across security levels, and therefore no insertion will be rejected due to existing entity polyinstantiation at higher level. Also, there is no ambiguity, because no entity polyinstantiation is allowed in what is accepted by subjects at any particular security level.

4.3 Data-Borrow Integrity

The newly introduced Data-Borrow Integrity is a key property of our data-based semantics. Allowing data-borrow ensures that the MLR data model can retain upward information flow. Changes to data at a lower level can be automatically propagated to higher levels. The integrity property can be expressed as follows.

[1] It has been misexpressed at many places as $A_1, C_1, TC \rightarrow A_i$, which is obviously too weak.

Property 3 [Data-Borrow Integrity (DBI)] An instance r of a multilevel relation R satisfies data-borrow integrity if and only if for all $t \in r$ and $1 \leq i \leq n$, if $t[A_i] \neq$ null $\wedge$ $t[C_i] < t[TC]$, there exists $t' \in r$ such that $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] = t'[C_i] = t[C_i] \wedge t'[A_i] = t[A_i]$. □

This is based on the following idea of data-based semantics: c-tuple contains all the data accepted (but not necessarily owned) by c-subjects; absence of a c-tuple means that to c-subjects the entity does not exist.

Consider the following relation instance

SHIP		OBJ		DEST		TC
Ent	U	Exploration	U	Rigel	S	S
Ent	U	Exploration	U	Talos	U	U

SHIP		OBJ		DEST		TC
Ent	U	Exploration	U	Rigel	S	S

The former one satisfies DBI but the later one does not. Here DBI requires that the U-tuple, which is the source of the data Ent and Exploration, must exist. This is because absence of a U-tuple means that to U-subjects the entity Ent does not exist; which implies that the data once owned by U-subjects is invalid now and, of course, can no longer be used by S-subjects.

Note that DBI is independent from PI. For example, the second instance above does not satisfy DBI but yet satisfies PI.

4.4 Foreign Key Integrity

The Foreign Key Integrity is also proposed by SeaView, and is another very stable property.

Property 4 [Foreign Key Integrity (FKI)] Let FK be a foreign key of the referencing relation R. An instance r of a multilevel relation R satisfies foreign key integrity if and only if for all $t \in r$

1. Either $(\forall A_i \in FK)[t[A_i] =$ null$]$ or $(\forall A_i \in FK)[t[A_i] \neq$ null$]$.

2. $A_i, A_j \in FK \Rightarrow t[C_i] = t[C_j]$ (i.e., FK is uniformly classified). □

The first part of this property arises from standard relations. The motivations for the second part of this property are similar to those for the uniform classification of apparent primary keys in EI.

4.5 Referential Integrity

Referential Integrity appears both in SeaView and in Sandhu-Jajodia model. The main issue with referential integrity is to avoid semantic ambiguity, as discussed at length in [13]. The definition given here is similar to the original SeaView definition and the Sandhu-Jajodia definition. However, ambiguity is eliminated by our data-based semantics as follows.

Property 5 [Referential Integrity (RI)] Let FK be a foreign key of the referencing relation R_1. Let R_2 be the referenced relation, with apparent primary key AK. Instances r_1 of R_1 and r_2 of R_2 satisfy referential integrity if and only if for all $t_1 \in r_1$ such that $t_1[FK] \neq$ null, there exists $t_2 \in r_2$ such that $t_1[FK] = t_2[AK] \wedge t_1[TC] = t_2[TC] \wedge t_1[C_{FK}] \geq t_2[C_{AK}]$. □

In standard relations, the referential integrity property precludes the possibility of dangling references. In other words a non-null foreign key must have a matching tuple in the referenced relation. The requirement $t_1[C_{FK}] \geq t_2[C_{AK}]$ is added by SeaView to only allow downward references. In our definition, we require $t_1[TC] = t_2[TC]$ as well, which means for any level c, c-tuples can only reference c-tuples. This follows naturally from our data-based semantics: c-tuple contains all the data accepted by c-subjects, absence of a c-tuple means to c-subjects the entity does not exist.

Now let us consider the two examples described in [13] as an impasse between referential ambiguity and modeling power.

In the first example, references between relation instances SOD and CS

SHIP		OBJ		DEST		TC
Ent	U	Exploration	U	Talos	U	U
Ent	S	Spying	S	Rigel	S	S

CAPTAIN		SHIP		TC
Kirk	U	null	U	U
Kirk	U	Ent	S	S

no longer have any ambiguity, because the S-tuple of CS only references the S-tuple of SOD. In previous models the S-tuple of CS has been interpreted as referencing both the U- and S-tuples of SOD, resulting in referential ambiguity.

As the second example, references between relation instances SOD and CS

SHIP		OBJ		DEST		TC
Ent	U	Exploration	U	Talos	U	U
Ent	U	Spying	S	Rigel	S	S

CAPTAIN		SHIP		TC
Kirk	U	null	U	U
Kirk	U	Ent	S	S

are also allowed. Again, the S-tuple of CS references the S-tuple of SOD without referential ambiguity.

5 Expressive Power

In this section we compare the expressive power of the MLR data model with respect to several other data models. This is done by giving ER (Entity-Relationship) style diagrams to show how entities, tuples and elements are related in different models.

We also give a decomposition for mapping a single MLR relation into multiple relations with tuple-level labeling. This not only shows another way to compare data models, but is also important in building MLR on current commercial multilevel database products. The decomposition is discussed in further detail in section 7.

We address the tuple-level labeling model first, because it is the simplest data model and is used as the base for our comparisons.

5.1 The Tuple-Level Labeling Model

The tuple-level labeling model has simple schemes as follows

A_1	A_2	...	A_n	TC

The expressive power of the tuple-labeling data model can be shown as Figure 1. Every entity is identified by A_1 and can have at most one tuple for each classification level. Each tuple has its class level recorded in TC, and consists of $n-1$ elements associated with A_1. The value of every element is kept in A_k $(2 \leq k \leq n)$.

Note that there is no entity polyinstantiation in the tuple-level labeling model, because here an entity is identified only by A_1. Fundamentally, tuple-level labeling can directly provide either element polyinstantiation or entity polyinstantiation, but not both. An alternate interpretation can be shown as Figure 2, in which every entity, identified by A_1 and C_1, can only have one tuple. It is obviously more useful to opt for element polyinstantiation, because this allows entities whose attributes are labeled at different classes; whereas entity polyinstantiation would require all attributes of an entity to be uniformly classified.

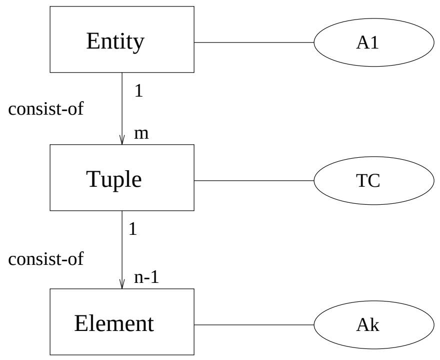

Figure 1: The expressive power of the tuple-level labeling data model

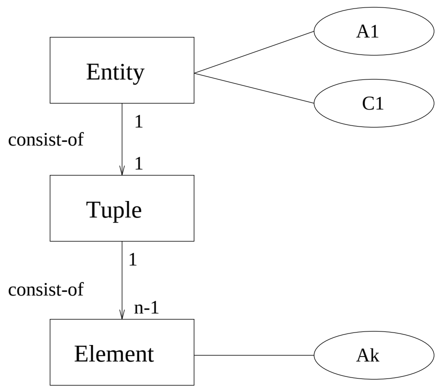

Figure 2: An alternate interpretation of the tuple-level labeling data model

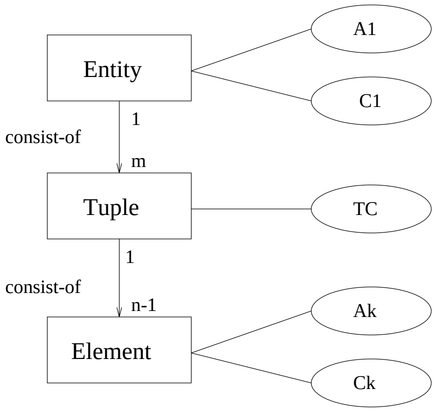

Figure 3: The expressive power of the MLR data model

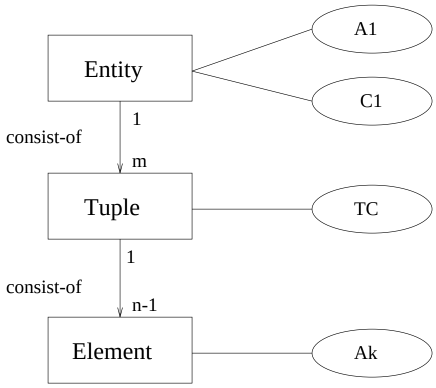

Figure 4: The expressive power of the semi-tuple-level labeling data model

5.2 The MLR Model

The expressive power of the MLR data model can be shown as Figure 3. Here an entity is identified by A_1 and C_1, and may have at most one tuple for each classification level. Each tuple has its class level recorded in TC, and consists of $n-1$ elements. The value and owner's level of every element are kept in A_k and C_k $(2 \leq k \leq n)$. It is clear that entity polyinstantiation is allowed here, because each entity is identified by both A_1 and C_1.

When converting the MLR data model to tuple-labeling data model, attributes Eid and El_2, $\ldots$, El_n could be introduced, which identify, respectively, an entity and an element in the MLR model and are transparent to *all* subjects.

A_1	Eid	C_1

Eid	El_2	$\ldots$	El_n	TC

El_k	A_k	C_k

This is to say, every A_1 can be mapped to at most one Eid at each classification level indicated by C_1; every Eid can have at most one set of elements El_2, $\ldots$, El_n at each level indicated by TC; every element El_k contains A_k and C_k.

This decomposition is simple and good for static expression. As for dynamic operations, the following decomposition is more convenient

A_1	Eid	C_1

Eid	El_2	$\ldots$	El_n	TC

El_k	Eid	A_k	C_k

which means element El_k is uniquely determined by Eid, A_k and C_k, therefore El_k can be added and deleted freely in cases such as Eid is to be added or deleted. See section 7 for further detail.

5.3 The Semi-Tuple-Level Labeling Model

What we called the semi-tuple-level labeling data model has schemes as follows,

A_1	C_1	A_2	$\ldots$	A_n	TC

This is exactly the same as the model provided by Smith and Winslett [14]. The expressive power of the semi-tuple-level labeling data model can be shown as Figure 4. Both entity and element polyinstantiation are allowed here, but there is no data borrow.

The semi-tuple-level labeling model can also be seen as coming from restricting the MLR model in such a way that no C_k $(2 \leq k \leq n)$ is allowed. By this restriction, the model would no longer take care of the sources of element data, and at the same time no upward information flow by data borrow would exist. When converting MLR to tuple-level labeling data model under these assumptions, we will get:

A_1	Eid	C_1

Eid	A_2	$\ldots$	A_n	TC

El_k's are no longer needed.

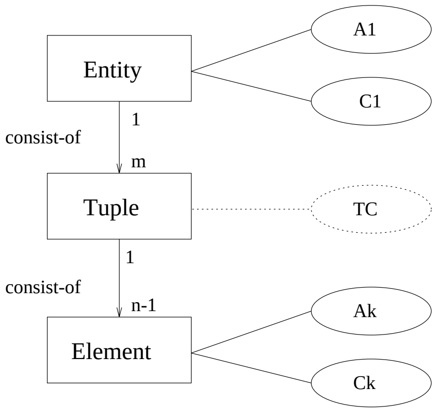

Figure 5: The expressive power of the SeaView data model

5.4 The SeaView Model

Now let us compare MLR with SeaView. Although the SeaView model is not grounded on belief-based semantics, we can still establish some relationship between SeaView and MLR. The expressive power of the SeaView data model can be shown as Figure 5. The TC in SeaView is redundant and can be calculated from $C_1, \ldots, C_n$. Hence, it is attached by a dotted line in the diagram.

Qian and Lunt [9] decompose a SeaView relation to several tuple-level labeling relations as follows

A_1	C_1

A_1	A_k	C_k

There is an implicit assumption in this decomposition: no entity polyinstantiation, i.e. $A_1 \rightarrow C_1$.[2] Here, Eid is not needed, because A_1 can identify the entity. El_k's are not needed either, because the relation including TC in the MLR decomposition can, to some extent, be omitted.

What SeaView can do to counter the semantic ambiguity problem addressed in section 2 is very limited. The model-theoretic semantics of [8] is slightly different from the SeaView's original "fact-based" semantics [9], in that it integrates some ideas from the belief-based semantics to overcome semantic ambiguity. However, as long as believability is equated to visibility, it is quite possible that either believability is maximized or visibility is minimized. To differentiate believability from visibility, operational semantics and performance may become two problems.

[2] If entity polyinstantiation was allowed, A_1 alone could not be used to connect two or more tuple-level labeling relations during the recovery process.

5.5 Discussion

As an example, let us consider an MLR instance

SHIP		OBJ		DEST		TC
Ent	U	Mining	S	Talos	U	TS
Ent	U	Mining	S	Rigel	S	S
Ent	U	Exploration	U	Talos	U	U

In this case, all data accepted by TS-subjects are borrowed from lower-level subjects. Furthermore, TS-subjects take DEST from U-subjects instead of S-subjects, and the data Talos can be changed by U-subjects. Neither SeaView nor semi-tuple-labeling model can express this case, because it concerns data-borrow. In SeaView, TS-subjects should accept S-tuple if they do not have their own data. In semi-tuple-labeling model, TS-subjects can only accept the data owned by themselves.

So far, we can see that the MLR model is a very powerful multilevel relational data model, which can be used as a unified data model to support general MLS database design.

6 Manipulation

There are five manipulation statements in the MLR data model. Four of them are the conventional SQL statements of INSERT, DELETE, SELECT and UPDATE. The fifth statement is UPLEVEL and is new to MLR. The UPLEVEL statement is introduced to solve the operational incompleteness problem addressed in section 2. Compared to the Sandhu-Jajodia model of [12], we have redefined the semantics of the four standard SQL statements, and have replaced PUPDATE by UPLEVEL. The SELECT statement given here is similar to that of Smith-Winslett [14], and our intent is to provide a friendly interface upward-compatible to the standard SQL.

We first give several examples to show how these statements are used. After that, we present formal syntax and semantics for all these manipulation statements. The syntax is similar to that in [5], and the explanation given here concentrates on data-borrow and operation propagation.

We are not going to discuss performance in detail, because it depends on physical strategies. However, these operations should not present a performance problem if we can use an entity, instead of a tuple, as the basic storage and retrieval unit. As discussed in section 3.3, the size of an entity will not be too large, in comparison to that of a tuple.

6.1 Examples

Consider the following relation instance

SHIP		OBJ		DEST		TC
Ent	U	Mining	M_1	Talos	U	M_1
Ent	U	Exploration	U	Sirius	M_2	M_2
Ent	U	Exploration	U	Talos	U	U

to which an S-subject applies the following UPLEVEL command

UPLEVEL SOD
GET OBJ FROM M_1, DEST FROM M_2
WHERE SHIP = "Ent"

giving us the following result

SHIP		OBJ		DEST		TC
Ent	U	Mining	M_1	Sirius	M_2	S
Ent	U	Mining	M_1	Talos	U	M_1
Ent	U	Exploration	U	Sirius	M_2	M_2
Ent	U	Exploration	U	Talos	U	U

An S-tuple is added into SOD, whose OBJ and DEST values are, respectively, borrowed from the data owned by M_1-subjects and M_2-subjects.

Next, suppose an S-subject executes the following UPDATE statement

UPDATE SOD
SET DEST = "Rigel"
WHERE SHIP = "Ent"

the result is

SHIP		OBJ		DEST		TC
Ent	U	Mining	M_1	Rigel	S	S
Ent	U	Mining	M_1	Talos	U	M_1
Ent	U	Exploration	U	Sirius	M_2	M_2
Ent	U	Exploration	U	Talos	U	U

Only the S-tuple is changed.

However, if an M_1-subject issues

UPDATE SOD
SET OBJ = "Spying"
WHERE SHIP = "Ent"

the relation instance will be

SHIP		OBJ		DEST		TC
Ent	U	Spying	M_1	Rigel	S	S
Ent	U	Spying	M_1	Talos	U	M_1
Ent	U	Exploration	U	Sirius	M_2	M_2
Ent	U	Exploration	U	Talos	U	U

The M_1-tuple is changed, and the change is propagated to S-tuple. This is because the OBJ value accepted by S-subjects is currently owned by M_1-subjects.

Furthermore, a DELETE statement from an M_1-subject

DELETE
FROM SOD
WHERE SHIP = "Ent"

will change the relation instance to

SHIP		OBJ		DEST		TC
Ent	U	null	M_1	Rigel	S	S
Ent	U	Exploration	U	Sirius	M_2	M_2
Ent	U	Exploration	U	Talos	U	U

The M_1-tuple is deleted and the OBJ value in S-tuple is set to null since no data is currently owned by M_1-subjects.

If the M_1-subject issues an UPLEVEL instead of the DELETE,

UPLEVEL SOD
GET OBJ FROM U, DEST FROM U
WHERE SHIP = "Ent"

the result will be

SHIP		OBJ		DEST		TC
Ent	U	null	M_1	Rigel	S	S
Ent	U	Exploration	U	Talos	U	M_1
Ent	U	Exploration	U	Sirius	M_2	M_2
Ent	U	Exploration	U	Talos	U	U

The M_1-tuple is changed and all its values are borrowed from U-subjects. The OBJ value of the S-tuple is null because there is no OBJ data currently owned by M_1-subjects.

Note that there is upward propagation of changes in UPLEVEL, DELETE and UPDATE statements.

To the instance above, an M_2-subject issuing

SELECT *
FROM SOD

will get the following result

SHIP	OBJ	DEST
Ent	Exploration	Sirius

i.e., the tuple at level M_2 with neither classification attribute nor tuple-class attribute. This looks like a traditional SELECT statement applied to a traditional relation, consisting of all data in the M_2-tuple. If the issued statement was

SELECT $*\%$
FROM SOD

the result would be

SHIP		OBJ		DEST		TC
Ent	U	Exploration	U	Sirius	M_2	M_2

all data attributes, classification attributes as well as a tuple-class attribute are included.

Finally, let us consider the two examples from [13] mentioned in section 4.5. To both of them, if a TS-subject issues the following SELECT statement

SELECT CS.CAPTAIN, CS.CAPTAIN%,
SOD.DEST, SOD.DEST%,
SOD.TC
FROM CS, SOD
WHERE CS.SHIP=SOD.SHIP
AT S

where CAPTAIN% and DEST% stand for the classification attributes of CAPTAIN and DEST respectively, the results returned to the TS-subject are the same

CAPTAIN		DEST		TC
Kirk	U	Rigel	S	S

This is because in both cases the S-tuple of CS only joins with the S-tuple of SOD.

This concludes our examples and we now give the formal definition and operational semantics of the five data manipulation statements.

6.2 The INSERT Statement

The INSERT statement executed by a c-subject has the following general form:

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

where R is a relation name; A_i, A_j, ... are data attribute names, $1 \leq i, j, \ldots \leq n$; a_i, a_j, ... are data values for A_i, A_j, ... respectively. Value specified must be from appropriate domains. (In this paper, as a syntax description we use [ ] to stand for option and ... for repetition.)

Each INSERT operation can insert at most one tuple into the relation R. The inserted tuple t is constructed as follows: for $1 \leq k \leq n$,

1. if A_k is in the attribute list of INTO clause, $t[A_k, C_k] = (a_k, c)$;
2. if A_k is not in the attribute list of INTO clause,
   (a) if $c \in [L_k, H_k]$, $t[A_k, C_k] = (\text{null}, c)$;
   (b) if $c \notin [L_k, H_k]$, $t[A_k, C_k] = (\text{null}, \text{null})$;

Also, $t[TC] = c$.

The insertion is permitted if and only if:

1. There is no $t' \in r$ such that $t'[A_1] = a_1 \wedge t'[TC] = c$; and
2. The resulting database state satisfies EI, FKI and RI.

If so, the tuple t is inserted into r. Otherwise the operation is rejected and the original database state is left unchanged.

6.3 The DELETE Statement

The DELETE statement executed by a c-subject has the following general form:

DELETE
FROM R
[WHERE p]

where R is a relation name; p is a predicate expression which may include conditions involving the classification attributes, in addition to the usual case of data attributes.

Only tuples $t \in r$ with $t[TC] = c$ will be taken into the calculation of p. For those tuples $t \in r$ that satisfy the predicate p, r will be changed as follows:

1. t will be deleted;
2. if $t[C_1] = c$, all $t' \in r$ with $t'[A_1, C_1] = t[A_1, C_1]$ $\wedge$ $t'[TC] > c$ will be deleted from r;[3]
3. if $t[C_1] < c$, for $t' \in r$ with $t'[A_1, C_1] = t[A_1, C_1]$ $\wedge$ $t'[TC] > c \wedge t'[C_k] = c$, $t'[A_k]$ is set to null.

[3] Some improvement are possible, such as instead of deleting higher-level data or setting higher-level data to null, the system could just "freeze" and mark them, show subjects at these levels some warnings and let these subjects fix them. Those are important in practice and should be explicitly added to the semantics of an implementation.

The DELETE statement is successful if at level c the resulting database state satisfies RI. Otherwise the operation is rejected and the original database state is left unchanged.

In case that RI is not satisfied at levels c' ($c' > c$),[4] for the relation R_1 with relation instance r_1 containing the referencing tuple t_1 and with apparent primary key AK_1 and the foreign key FK_1,

1. if $FK_1 \cap AK_1 = \emptyset$, there are two steps to be done,
   (a) if $t_1[C_{FK_1}] = c'$, t_1 is set as $t_1[FK_1] =$ null, and for $t'_1 \in r_1$ with $t'_1[AK_1, C_{AK_1}] = t_1[AK_1, C_{AK_1}] \wedge t'_1[TC] > c' \wedge t'[C_{FK_1}] = c'$, $t'_1[FK_1]$ is set to null;
   (b) if $t_1[C_{FK_1}] < c'$ and $t_1[FK_1]$ has not been set to null in step (a), t_1 is set as $t_1[FK_1, C_{FK_1}] =$ (null,c');
2. if $FK_1 \cap AK_1 \neq \emptyset$, t_1 (at level c') should also be deleted, which appears as cascading deletions.

6.4 The SELECT Statement

The SELECT statement executed by a c-subject has the following general form:

$$\begin{array}{ll} \text{SELECT} & B_1[, B_2] \ldots \\ \text{FROM} & R_1[, R_2] \ldots \\ \text{[WHERE} & p] \\ \text{[AT} & c_1[, c_2] \ldots] \end{array}$$

where B_1, B_2, ... are attribute names, either data attribute or classification attribute or tuple-class attribute (Wildcards are available, $*$ for all data attributes, % for all classification attributes and tuple-class attribute, $*$% for all attributes); R_1, R_2, ... are relation names; p is a predicate expression which may include conditions involving the classification attributes, in addition to the usual case of data attributes; c_1, c_2, ... are values of classification levels (There is wildcard $*$ standing for all levels lower than or equal to c). Value specified must be from appropriate domains.

Only those tuples $t \in r_1, r_2, \ldots$ that have $t[TC]$ being c if there is no AT clause or otherwise included in AT clause will be taken into the calculation of p. If there are more than one relation included in FROM clause, the p is implicitly substituted by $p \wedge (R_1.TC = R_2.TC = \ldots)$.

For those tuples t satisfying p, the data of t for those attributes listed in SELECT clause will be included in the result.

6.5 The UPDATE Statement

The UPDATE statement executed by a c-subject has the following general form:

$$\begin{array}{ll} \text{UPDATE} & R \\ \text{SET} & A_i = s_i[, A_j = s_j] \ldots \\ \text{[WHERE} & p] \end{array}$$

where R is a relation name; A_i, A_j, ... are data attribute names, $1 \leq i, j, \ldots \leq n$; s_i, s_j, ... are scalar expression for A_i, A_j, ... respectively; p is a predicate expression which may include conditions involving the classification attributes, in addition to the usual case of data attributes. Value specified must be from appropriate domains.

Only tuples $t \in r$ with $t[TC] = c$ will be taken into the calculation of p. For those tuples $t \in r$ that satisfy the predicate p, r will be updated as follows:

1. if some attribute of A_1 is in SET clause,
   (a) if $t[C_1] = c$, all tuples $t' \in r$ that have $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c$ will be deleted[5];
   (b) if $t[C_1] < c$,
      i. for $2 \leq k \leq n$, if A_k is not in SET clause and $t[C_k] < c$, $t[A_k, C_k] =$ (null,c);
      ii. for tuples $t' \in r$ with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_k] = c$, $t'[A_k] =$ null;
   (c) for $1 \leq k \leq n$, if A_k is in SET clause, $t[A_k, C_k] = (|s_k|, c)$;
2. if no attribute of A_1 is in SET clause, for $2 \leq k \leq n$, if A_k is in SET clause,
   (a) $t[A_k, C_k] = (|s_k|, c)$;
   (b) for tuples $t' \in r$ with $t'[A_1, C_1] = t[A_1, C_1] \wedge t'[TC] > c \wedge t'[C_k] = c$, $t'[A_k] = |s_k|$.

The UPDATE operation is successful if and only if:

1. In case that some attribute of A_1 is in SET clause, there is no $t' \in r$ such that for the resulting $t[A_1]$, $t'[A_1] = t[A_1] \wedge t'[TC] = c$; and

[4] Adopting different policies in RI checking, at level c and at levels c' ($c' > c$), is because the DELETE statement issued by a c-subject should not be rejected due to violation of RI at levels c', otherwise there would be downward information leakage. Same thing happens with the UPDATE statement.

[5] Changing A_1 means changing the entity. Lower level subjects should not have the privilege of having higher level subjects to accept any new entity beyond their willingness to do so by UPLEVEL statements.

2. The resulting database state satisfies EI, FKI and RI (at level c).

Otherwise the operation is rejected and the original database state is left unchanged.

In case that RI is not satisfied at levels c' ($c' > c$), for the relation R_1 with relation instance r_1 containing the referencing tuple t_1 and with apparent primary key AK_1 and the foreign key FK_1,

1. if $FK_1 \cap AK_1 = \emptyset$, there are two steps to be done,
   (a) if $t_1[C_{FK_1}] = c'$, t_1 is set as $t_1[FK_1] =$ null, and for $t'_1 \in r_1$ with $t'_1[AK_1, C_{AK_1}] = t_1[AK_1, C_{AK_1}] \wedge t'_1[TC] > c' \wedge t'[C_{FK_1}] = c'$, $t'_1[FK_1]$ is set to null;
   (b) if $t_1[C_{FK_1}] < c'$ and $t_1[FK_1]$ has not been set to null in step (a), t_1 is set as $t_1[FK_1, C_{FK_1}] = (\text{null}, c')$;
2. if $FK_1 \cap AK_1 \neq \emptyset$, t_1 (at level c') should also be deleted, which appears as cascading deletions.

6.6 The UPLEVEL Statement

The UPLEVEL statement executed by a c-subject has the following general form:

UPLEVEL	R
GET	A_i FROM $c_i[, A_j$ FROM $c_j] \ldots$
[WHERE	p]

where R is a relation name; A_i, A_j, ... are data attribute names, $2 \leq i, j, \ldots \leq n$; c_i, c_j, ... are values of classification levels for A_i, A_j, ... respectively; p is a predicate expression which may include conditions involving the classification attributes and tuple-class attributes, in addition to the usual case of data attributes. Value specified must be from appropriate domains.

Only tuples $t \in r$ with $t[TC] \leq c$ will be taken into the calculation of p. For every entity that has tuples $t' \in r$ satisfying the predicate p, a c-tuple t will be constructed as follows:

1. $t[A_1, C_1] = t'[A_1, C_1]$;
2. for $2 \leq k \leq n$,
   (a) if A_k is in GET clause,
      i. if there is a tuple t'' with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] = t''[C_k] = c_k$, set $t[A_k, C_k] = t''[A_k, C_k]$;
      ii. if there is no tuple t'' with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] = t''[C_k] = c_k$, set $t[A_k, C_k] = (\text{null}, c_k)$;
   (b) if A_k is not in GET clause,
      i. if $c \in [L_k, H_k]$, $t[A_k, C_k] = (\text{null}, c)$;
      ii. if $c \notin [L_k, H_k]$, $t[A_k, C_k] = (\text{null}, \text{null})$.

After that,

1. if there is a tuple t'' with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] = c$,
   (a) replace t'' with t;
   (b) for any tuple t''' and any $2 \leq k \leq n$ such that $t'''[A_1, C_1] = t[A_1, C_1] \wedge t'''[TC] > c \wedge t'''[C_k] = c$, if $t[A_k, C_k] \neq t''[A_k, C_k]$, set t''' as $t'''[A_k] =$null.
2. if there is no tuple t'' with $t''[A_1, C_1] = t[A_1, C_1] \wedge t''[TC] = c$, add t into r.

The UPLEVEL operation is successful if and only if the resulting database state satisfies PI, FKI and RI. Otherwise the operation is rejected and the original database state is left unchanged.

7 Operational Behavior of Tuple-Level Decomposition

As mentioned in section 5.2, an MLR relation $R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$ can be decomposed to several tuple-level labeling relations

1. $R^1(A_1, Eid, C_1)$
2. $R^2(Eid, El_2, \ldots, El_n, TC)$
3. $R^{3k}(El_k, Eid, A_k, C_k)$

Now as a feasibility discussion, the operations issued by a c-subject in the MLR data model can be interpreted as follows,

- INSERT
  1. generate a new Eid and some new El_k;
  2. insert tuples into R^1, R^2 and some R^{3k}.
- SELECT
  1. select attributes from R^1, R^2 and R^{3k}, using Eid and El_k as connection keys.
- UPLEVEL
  1. select Eid from R^1, R^2 and R^{3k}, using Eid and El_k as connection keys;
  2. for the selected Eid that has no c-tuple in R^2, insert a tuple into R^2;

3. for the selected Eid that has c-tuple in R^2, update R^2 to change this c-tuple as well as propagate the change to higher level tuples.

- UPDATE
  1. select Eid and El_k from R^1, R^2 and R^{3k}, using Eid and El_k as connection keys;
  2. if A_1 is to be changed, for $C_1 < c$, generate a new Eid and insert a tuple into R^1, change R^2 and R^3 for both update and propagation; for $C_1 = c$, update R^1 as well as delete all higher level tuples of the entity in R^1, R^2 and R^{3k} for propagation.
  3. for the selected El_k that has $C_k = c$, update R^{3k} to change A_k;
  4. for the selected El_k that has $C_k < c$, generate a new El_k and insert a tuple into R^{3k} as well as update R^2 to propagate the change;
- DELETE
  1. select Eid and El_k from R^1, R^2 and R^{3k}, using Eid and El_k as connection keys;
  2. for the selected Eid that has $C_1 < c$, delete relative tuples in R^2 and R^{3k} as well as update R^2 to propagate these changes;
  3. for the selected Eid that has $C_1 = c$, delete all tuples in R^1, R^2 and R^{3k}, that contain the Eid.

Here the relation R can be seen as an updatable view based on R^1, R^2 and R^{3k}. However, both scheme mapping and operation interpretation as well as integrity checking require some tools to be developed. For example, we need tools to

1. generate and maintain Eid and El_k;
2. update and delete some specific tuples with classification levels higher than the level of the subject issuing the operation;
3. interpret some details of these operation statements.

Fortunately, all these tools could work in a straightforward way. The only thing that we should take care of is the performance. Joining R^1, R^2 and R^{3k} to reconstruct R is, to some extent, unavoidable in any model conversion as long as decomposition is used. What remains is data-borrow/operation-propagation, which should not present a performance problem if the interpretation tool could be implemented in the way that the (MLR) entity is treated as an access unit, i.e. one entity at a time.

8 Conclusion

Our major contributions in this paper are:

1. Establish the MLR data model with data-based semantics by unifying ideas from a number of other models.
2. Analyze the expressive power of the MLR data model, indicating that the MLR data model can be used as a unified data model to support general MLS database design.
3. Describe how to convert the MLR data model to tuple-level labeling ones, which shows a practical way to build the model on the current commercial multilevel database products.

Particularly, our redefined Polyinstantiation Integrity and Referential Integrity as well as our newly introduced UPLEVEL statement and Data-Borrow Integrity strongly support the fact that the MLR model is a simple, unambiguous and powerful data model.

Finally, we reiterate that the MLR model is sound, complete and free of downward information flows. Proofs of these properties cannot be provided here for lack of space, but are given in [2].

References

[1] Bell, D.E. and LaPadula, L.J. "Secure Computer Systems: Unified Exposition and Multics Interpretation." MTR-2997, MITRE (1975).

[2] Chen, F. and Sandhu, R.S. "The Multilevel Relational (MLR) Data Model." Technical Report, ISSE-TR-95-101, George Mason Univ (1995).

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

[4] Haigh, J.T., O'Brien, R.C. and Thomsen, D.J. "The LDV Secure Relational DBMS Model." *Database Security IV: Status and Prospects*, S. Jajodia and C. E. Landwehr (editors), North-Holland, 1991, pages 265-279.

[5] Jajodia, S. , Sandhu, R.S., and Sibley E. "Update Semantics of Multilevel Relations." *Proc. 6th Annual Computer Security Applications Conf.*, Tucson, AZ, December 1990, pages 103-112.

[6] Jajodia, S. and Sandhu, R.S. "A Novel Decomposition of Multilevel Relations Into Single-Level Relations." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1991, pages 300-313.

[7] 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).

[8] Qian, X. "A model-theoretic semantics of the multilevel relational model." *Advances in Database Technology – EDBT'94, Lecture Notes in Computer Science 779*, Jarke, M., Bubenko, J. and Jeffery, K. (editors), Springer-Verlag, pages 201-214, 1994.

[9] Qian, X. and Lunt, T.F. "Tuple-level vs. element-level classification." *Database Security IV: Status and Prospects*, Thuraisingham, B.M. and Landwehr, C. (editors), North-Holland, pages 301-315, 1993.

[10] Sandhu, R.S., Jajodia, S. and Lunt, T. "A New Polyinstantiation Integrity Constraint for Multilevel Relations." *Proc. IEEE Workshop on Computer Security Foundations*, Franconia, New Hampshire, June 1990, pages 159-165.

[11] Sandhu, R.S. and Jajodia, S. "Honest Databases That Can Keep Secrets." *14th NIST-NCSC National Computer Security Conference*, Washington, D.C., October 1991, pages 267-282.

[12] Sandhu, R.S. and Jajodia, S. "Polyinstantiation for Cover Stories." *Proc. European Symposium on Research in Computer Security*, Toulouse, France, November 1992, pages 307-328. Published as *Lecture Notes in Computer Science, Vol 648, Computer Security—ESORICS92* (Deswarte, Y., Eizenberg, G., and Quisquater, J.-J., editors), Springer-Verlag, 1992.

[13] Sandhu, R.S. and Jajodia, S. "Referential Integrity in Multilevel Secure Databases." *16th National Computer Security Conference*, Baltimore, MD, Sept. 20-23, 1993, pages 39-52.

[14] Smith, K. and Winslett, M. "Entity Modeling in the MLS Relational Model." *Proc. of the 18th VLDB Conference*, Vancouver, British Columbia, Canada, 1992, pages 199-210.