Proc. IEEE Symposium on Research in Security and Privacy
Oakland, California, May 1990, pages 104-105

POLYINSTANTIATION INTEGRITY IN MULTILEVEL RELATIONS

Sushil Jajodia[*][†] *and Ravi Sandhu*[†]

Department of Information Systems and Systems Engineering
George Mason University, Fairfax, VA 22030-4444

ABSTRACT. Polyinstantiation integrity (PI) as defined in the SeaView multilevel relational data model consists of a functional dependency component and a multivalued dependency component. We show that the latter component rules out many practically useful relations and is therefore unduly restrictive. This leads us to propose that PI be defined to consist only of the functional dependency component. For this revised definition of PI, we formulate and prove correct a lossless decomposition of multilevel relations into single level ones with recovery based on the natural join operation.

1 INTRODUCTION

In a multilevel world as we raise a user's clearance new facts will emerge. Conversely as we lower a user's clearance some facts will get hidden. It is therefore inherent that users at different levels see different versions of reality. This naturally complicates the meaning of multilevel relations relative to the meaning of relations as ordinarily considered in a single level world.

Various approaches to extending the standard relational model to deal with multilevel relations have been proposed. A major issue is how access classes are assigned to data stored in relations. The proposals have ranged from assigning access class

[*] The work of Sushil Jajodia was supported by the U.S. Air Force, Rome Air Development Center through subcontract # RI-64155X of prime contract # F30602-88-D-0028, Task B-9-3622 with Univ. of Dayton.

[†] Both authors are also with the Center of Excellence for Command, Control, Communications, and Intelligence at George Mason University. The Center receives general support via grants from the Virginia Center for Innovative Technology, MITRE, the Defense Communications Agency, and PRC/ATI.

to relations (as in [10]), assigning access classes to individual tuples in a relation (as in [9]), or assigning access classes to individual attributes of a relation (as in [11]).

The most ambitious and exciting proposal has come from the SeaView (Secure Data Views) project to assign security classifications to individual data elements of the tuples of a relation. This project began as a joint effort by SRI International and Gemini Computers with the goal of designing and prototyping a multilevel secure relational database management system that satisfies the Trusted Computer System Evaluation Criteria for Class A1 [7]. SeaView researchers have considerably advanced the state of the art in multilevel database security and the project itself has moved to a prototype implementation phase using GEMSOS as the underlying TCB along with the ORACLE relational DBMS [14]. SeaView has been extensively described [4, 5, 6, 13, 14, 15, 16, 17, 18, for instance].

Perhaps the most significant contribution of SeaView is the realization that multilevel relations at the logical level can be decomposed into single level base relations which are then physically stored in the database. Completely transparent to users, multilevel relations can be reconstructed from these base relations on user demand. The practical advantages of being able to decompose and store a multilevel real relation by a collection of single level base relations are obvious. In particular the TCB can enforce mandatory controls with respect to the single level base relations which allows the DBMS to execute largely as an untrusted application on the TCB.

Another crucial contribution of SeaView is the formulation of polyinstantiation as a fundamental property of multilevel relations. Much as most programmers are more comfortable with sequential programs and face considerable difficulty in dealing with concurrent programs, we believe polyinstantiation makes multilevel relations that much more difficult to deal with than standard single level relations. We are sympathetic with Denning's observation [6] that "polyinstantiation is an intrinsic *problem* of multilevel systems." We would state the concern a little differently to say that "polyinstantiation is an intrinsic *phenomenon* of multilevel systems."

Our objective in this paper is to identify some properties of polyinstantiation that have been overlooked or oversimplified in the SeaView work. Our major contribution is to show that polyinstantiation integrity (PI) as defined in SeaView is needlessly restrictive. Specifically SeaView defines PI by a functional dependency component and a multivalued dependency component. We show that the latter component can be dropped while still preserving the very desirable property that multilevel relations can be stored as single level base relations. We also demonstrate that SeaView's multivalued dependency component of PI prohibits many relation instances which are of obvious practical value.

Our contribution is a positive one in that we not only point out shortcomings in SeaView's current definitions but also show how these can be fixed. Among the most encouraging aspects of our work is that the recovery of multilevel relations from single level base relations is accomplished by using the natural join operator rather than the outer joins proposed in SeaView. As a result we avoid the theoretical complications and pitfalls which arise with outer joins.

The paper is organized as follows. Section 2 presents examples of several relations which are intuitively very reasonable yet are ruled out by SeaView. Section 3 formalizes the informal discussion of section 2. It also reviews the standard relational model and SeaView to the extent required for our objective. Section 4 formulates and prove correct a lossless decomposition of multilevel relations into single level ones with our revised definition of PI. The proofs turn out to be quite subtle underscoring the need for careful analysis in this regard. Section 5 concludes the paper with a discussion of future work that remains to be done.

The reader should note that we have relied on the SeaView final report [16] as the definitive description. Some aspects of SeaView have since been changed or clarified, as a result of an earlier draft of this paper and our conversations with Teresa Lunt. Nevertheless the modifications we have seen [17, 18] do not resolve all the problems we point out in this paper.

2 MULTILEVEL RELATIONS

The purpose of this section is to argue on intuitive grounds, by means of a simple example, that SeaView's definition of polyinstantiation integrity is too restrictive. We define a single relation and consider several different instances of this relation as enumerated in table 3. Each instance has a realistic and useful interpretation. A general data model should certainly allow any one of the eight instances to occur if this is deemed appropriate in an application. However SeaView allows only two combinations of these eight instances within a single relation scheme. Specifically a SeaView relation can accommodate either instances 1, 2, 3, 8 or 1, 4 within a single scheme.

We begin by considering an ordinary relation which has three attributes, Starship, Objective and Destination with Starship being the key. For each starship there is at most one tuple in this relation giving us that starship's unique objective and unique destination. For example, the tuple <Enterprise, Exploration, Talos> denotes that the starship Enterprise has set out to explore Talos. We say that this entire tuple gives us the mission of the Enterprise.

Next consider a multilevel relation which attempts to represent the same information, i.e., the objective and destination of starships, but in a multilevel world where some facts are classified. Assume there are just two levels, U for unclassified and S for secret. Following SeaView we now have a classification attribute associated with each of the original attributes of the above mentioned ordinary relation. To simplify the example let us say the Starship attribute is always unclassified. So the classification range of the Starship attribute has lower and upper bounds of U. On the other hand let the classification range of the Objective and Destination attributes have a lower bound of U and upper bound of S. This allows a starship to have a secret objective and/or a secret destination. Let us call the resulting relation SOD with the scheme summarized in table 1. The Tuple-Class attribute, abbreviated as TC, gives us the classification of the entire tuple. TC is a redundant attribute whose value is the least upper bound of the attribute classifications for the individual attributes in a tuple. The range of TC is derived in an obvious way from the classification ranges of the individual attributes.

The apparent primary key of SOD is specified as Starship. Intuitively this means that if only unclassified data is stored in SOD then Starship would be the actual primary key of the relation. Similarly, if only secret data is stored in the Objective and Destination attributes Starship would be the actual primary key. On the other hand if a mix of secret and unclassified data is stored in these attributes the actual primary key of SOD is Starship along with the attribute classifications. This point gets to the crux of the question we are addressing in this paper and is formally stated in the next section. The intuition can be appreciated by considering instance 8 of table 3. This instance contains four tuples for the starship Enterprise. What makes each tuple distinct is the classification of the Objective and Destination attributes.

An instance of SOD will contain different tuples at different levels. So we distinguish between the U-instance of SOD, visible to unclassified users, and the S-instance, visible to secret users. The inter-instance property of SeaView roughly speaking requires that the S-instance be a superset of the U-instance. This is most reasonable since increasing a user's clearance should keep all previously visible information intact and perhaps add some new facts visible only at the higher level. To be concrete consider the U-instance of SOD given in table 2. It contains exactly one tuple telling us that, as far as unclassified users are concerned, the starship Enterprise has set out to explore Talos. In table 3 we enumerate eight different S-instances of SOD, all of which are consistent with the U-instance of table 2. Their common property is that the single tuple of the U-instance appears in all eight S-instances. We regard each tuple in an instance of SOD as defining a mission for the starship in question. A U-instance of SOD allows only one mission per starship. S-instances on the other hand allow up to four missions per starship, three of which are secret and one unclassified.

We now demonstrate there is a practically useful and intuitively reasonable interpretation for each of the eight S-instances of table 3. Consider each S-instance in turn as follows.

1. *The S-instance is identical to the U-instance.* There is therefore no secret aspect to the Enterprise. This is the simplest case and needs little explanation.

In each of the next three cases there is a single tuple in the S-instance in addition to the tuple of the U-instance. This secret tuple defines a secret mission for the Enterprise in addition to its unclassified mission.

2. *The S-instance reveals the secret mission to be spying on Talos.* Presumably the unclassified exploration mission to Talos is a cover story to hide the secret spying mission. To maintain the integrity of the cover story, the Enterprise will probably expend resources on exploring Talos. Conceivably the bulk of its resources might be devoted to useful exploration of Talos with the secret spying mission added on as a low profile, low marginal cost and opportunistic effort. We obviously cannot resolve this issue without further knowledge about the real situation, such as a competent user might have. The main point is that the Enterprise does have two distinct missions: the unclassified one of exploring Talos and the secret one of spying there.

3. *The S-instance reveals the secret mission to be exploration of Rigel.* This case is very similar to the previous one in that only one attribute has a secret value. Clearly the desire to explore Rigel under cover of exploring Talos is a realistic one, not only in the national security arena but also in a competitive commercial context.

4. *The S-instance reveals the secret mission to be spying on Rigel.* This case is similar to the previous two in that there is only one secret mission. It is different in that the objective and destination of the secret mission are now both classified.

Each of the three preceding cases presents a distinctly different secret mission—secretly spying on Talos, secretly exploring Rigel and secretly spying on Rigel. These three secret missions do share the common property that exploring Talos is an acceptable unclassified cover story. The next three cases present situations where two of these three secret missions are concurrently in progress.

5. *The S-instance reveals two secret missions—to explore Rigel and to spy on Rigel.* Both secret missions are concerned with Rigel. Whether the principal one is to explore it or spy there or the two missions are equally important, cannot be ascertained without further information. The secret exploration of Rigel may simply be a convenient damage control story, should the secret destination of the Enterprise be leaked. Conversely, spying on Rigel may be an opportunistic and relatively unimportant add on to its secret exploration.

6. *The S-instance reveals two secret missions—to spy on Talos and to spy on Rigel.* This is similar to the previous case and once again we cannot a priori decide which, if any, is the principal secret mission.

7. *The S-instance reveals two secret missions—to spy on Talos and to explore Rigel.* This may appear strange at first, but it is perfectly proper. For instance, there may be no life-forms on Rigel worth spying on while there are indications of vast quantities of Uranium. This S-instance does point out problems with simple

rules such as "give the value with the highest classification for each attribute." Such a rule would manufacture the secret mission of spying on Rigel which does not exist in the relation.

As the reader may have guessed by now our final S-instance specifies that the three secret missions identified in instances 2, 3 and 4 are all concurrently in progress.

8. *The S-instance reveals three secret missions—to spy on Talos, to explore Rigel and to spy on Rigel.* As before, without further information and knowledge, we cannot say very much about the relation of these three secret missions to one another. All we know is that they share the same cover story of exploring Talos.

Now consider that a SeaView relation can accommodate either instances 1, 2, 3, 8 or 1, 4 of table 3 within a single scheme. In the former event our position is simply that S-instances 5, 6 and 7 are as meaningful as instance 8, and therefore should not be ruled out by fiat. SeaView however does exactly this since the multivalued dependency component of its definition of polyinstantiation integrity is not satisfied by instances 5, 6 and 7. In the latter event, which admits instances 1 and 4, SeaView requires the Objective and Destination attributes to be uniformly classified (i.e., either both are classified U or both S). We find it particularly troublesome that instance 4 cannot be accommodated without such a major assumption. After all the whole point of SeaView is to have element level classification. Requiring uniform classification in order to accommodate instance 4 amounts to defeating this basic objective.

In absence of the uniform classification constraint, consider that a secret user attempts to go from S-instance 1 to S-instance 4 by inserting the secret tuple specifying the secret mission of spying on Rigel. SeaView will interpret this as a request to go from S-instance 1 to S-instance 8, thereby manufacturing two bogus missions for the Enterprise. Unfortunately, the higher the clearance of a user the greater is the potential for such bogus information being retrieved from the database. It is easy to see that, in the worst case, the number of spurious tuples materialized by SeaView grows at the rate of $|\text{security-lattice}|^{k}$ where k is the number of non-key attributes in the relation. For instance, table 4 shows a TS-instance of a relation similar to SOD, except that it has a range of four security levels for the Objective and Destination attributes. The particular TS-instance shown there describes 4 missions for the Enterprise, one each at the unclassified, confidential, secret and top-secret levels. Seaview will require that this information be represented by the 16 missions shown in table 5. Users with clearance U, C, S and TS will respectively see 1, 4, 9 and 16 missions with the SeaView approach rather than the 1, 2, 3 and 4 missions they would respectively see with our proposal.

3 POLYINSTANTIATION INTEGRITY

Our objective in this section is to formalize the concepts discussed intuitively thus far. Until we arrive at the critical step of defining polyinstantiation integrity, this amounts to a review of SeaView's definitions. Our review is necessarily limited to the issues of concern in this paper and is not intended to be a complete description of either the relational model or SeaView.

The standard relational model [1, 2, 3] 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.

1. 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 as before 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 $[\text{lub}\{L_i\}, \text{lub}\{H_i\}]$.

2. A collection of state-dependent *relation instances* R_c, 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 $a_i \in D_i$, $c \geq c_i$ and $tc = \text{lub}\{c_i\}$. Moreover, if a_i is not null then $c_i \in [L_i, H_i]$. We require that c_i be defined even if a_i is null, i.e., a classification attribute cannot be null. In such cases c_i is equal to the classification of the apparent primary key (see below). Since tc is computed from the other classification attributes we will include it or omit it as convenient.

These multiple relation instances are, of course, intended to represent the version of reality appropriate for each access class. SeaView defines an inter-instance integrity property to ensure this objective is achieved. Roughly speaking as we go up to higher classes a tuple t that was already visible at a lower class should continue to be visible, although null elements of t may be replaced by non-null values in the higher-class instances. Furthermore a tuple should be visible at the lowest class allowed by its classification attributes. More precisely SeaView defines the following property. (This property has since been modified [17] in response to the problems pointed out in the next section. The new SeaView property is essentially the same as our property 5.)

Property 1 [Inter-Instance Integrity] A collection of relation instances for a given multilevel relation scheme satisfies the inter-instance integrity property if and only if $t \in R_c$ implies

1. $\forall c' > c \; \exists t' \in R_{c'}$ such that
   (a) $t[A_i] \neq \text{null} \Rightarrow t'[A_i, C_i] = t[A_i, C_i]$
   (b) $t[A_i] = \text{null} \Rightarrow t'[C_i] \geq t[C_i]$
2. $t \in R_{t[TC]}$. □

The notation $t[A_i]$ denotes the value of the A_i attribute in tuple t. More generally a set of attributes in the rectangular parenthesis following t is a "sub-tuple" of t with the values of these attributes.

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 of candidate keys is based on that of 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 2 [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.* Define C_{AK} to be the classification of the apparent key, i.e., $t[C_i] = t[C_{AK}]$ for all $A_i \in AK$.
3. $A_i \notin AK \Rightarrow t[C_i] \geq t[C_{AK}]$. □

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 [4, 8]. Hereafter we assume all multilevel relations satisfy property 2.

We are now ready to address polyinstantiation. Polyinstantiation arises in several different ways. In a standard relation there cannot be two tuples with the same primary key. In a multilevel relation we will similarly expect that there cannot be two tuples with the same *full primary key.* However, the user specified primary key is only the apparent primary key and secrecy considerations compel us to allow multiple tuples with the same apparent primary key. For instance, consider the scheme of table 1 with the classification range of its apparent key (Starship) modified to be [U,S]. Table 6 shows an S-instance of this relation with the Enterprise polyinstantiated at the U and S levels. Polyinstantiation is unavoidable because existence of the secret tuple with Enterprise/S cannot prevent insertion of the unclassified tuple with Enterprise/U without introducing a covert channel. We saw several examples of polyinstantiation for non-AK attributes in the previous section. Here again polyinstantiation serves to close covert channels. It moreover fulfills the real need for cover stories.

Although polyinstantiation is inevitable its effect must be controlled. SeaView proposes to do so by means of a new application independent integrity property, called polyinstantiation integrity. This property actually has two distinct parts. Since our objective is to argue against the second part we will review the SeaView definition in two pieces. The first part consists of a functional dependency component whose effect is to prohibit polyinstantiation within the same access class.

Property 3 [Polyinstantiation Integrity] Let AK be the apparent key of R. R satisfies polyinstantiation integrity if and only if for every R_c

$$AK, C_{AK}, C_i \rightarrow A_i \qquad \Box$$

This property allows all the S-instances of SOD shown in table 3 while ruling out the S-instance of table 7. This S-instance is clearly improper in light of the fact that the user has specified Starship as the apparent primary key.

The second part of SeaView's definition consists of a multivalued dependency component stated as follows.

Property 4 [MVD Polyinstantiation Integrity] Let AK be the apparent key of R. R satisfies MVD polyinstantiation integrity if and only if for every instance R_c, for all $A_i \notin AK$

$$AK, C_{AK} \rightarrow\rightarrow A_i, C_i \qquad \Box$$

As we demonstrated in the previous section this prohibits coexistence of the eight S-instances of table 3 within a single relation scheme. We also saw that with this requirement the four missions of table 4 can be stored only by materializing the 16 missions of table 5, or by the major restriction that Objective and Destination be uniformly classified. Moreover the SeaView literature does not give any intuitive reason for requiring this property and does not exhibit any example of an undesirable situation ruled out by it.

4 LOSSLESS DECOMPOSITION

In this section we develop and prove correct a lossless decomposition of a multilevel relation into single level ones. There are numerous pitfalls in this endevour as evident from a close examination of the SeaView decomposition. As we have pointed out at some length in section 2 the SeaView decomposition fails to be lossless for many instances which are intuitively reasonable and practically useful. But this is at least clearly formalized in SeaView. Unfortunately the proposed SeaView decomposition has not been stated or analyzed with the same rigor devoted to other aspects of SeaView. As a result there are many subtle and non-trivial issues which have been overlooked.

In analyzing the SeaView decomposition we find two fundamental problems which it is our goal to correct here.

1. Seaview does not base its decomposition on the full primary key of a multilevel relation but rather bases it on the apparent primary key.
2. Seaview does not store null values explicitly but attempts to pick them up by outer joins.

The concept of the full primary key is recognized in the SeaView literature. Indeed it is implicit in the definition of polyinstantiation integrity. The full primary key is $AK \cup C_{AK} \cup C_R$ where AK is the set of data attributes constituting the user specified primary key, C_{AK} is the classification attribute for data attributes in AK and C_R is the set of classification attributes for data attributes not in AK. Note that every classification attribute is part of the full primary key. We base our decomposition on the full primary key to guarantee that the recovery is lossless. Moreover we store null values in our decomposition explicitly. This has the great benefit of allowing us to recover the multilevel relation instances by using natural joins. Natural joins have a simpler behavior than outer joins. We understand that the outer join may be needed if we allow arbitrary insertions and deletions in the decomposed relations [14]. However we strongly urge they not be used without careful rigorous analysis. The results of this paper make it clear that such analysis is non-trivial and subtle even with the natural join.

4.1 Filter Function

We begin by formulating a new inter-instance integrity property to eliminate an ambiguity in the SeaView definitions. (As noted earlier, SeaView has since been modified to incorporate our property [17].) To be specific consider the TS-relation R_1 and the S-relation R_2 shown in tables 8 and 9 respectively. These have often been used as examples in the SeaView literature [4, 13, 16] where they are described respectively as the TS- and S-instances of the same relation. The SeaView definitions actually permit a second S-instance, R_3 shown in table 10 which is consistent with the TS-instance R_1. In particular R_1 satisfies the inter-instance property with respect to both R_2 and R_3.

Our inter-instance property is based on the definition of a *filter function* which maps a multilevel relation to different instances, one for each descending access class in the security lattice. The filter function limits each user to that portion of the multilevel relation for which he or she holds a clearance.

Definition 1 [Filter Function] Given the c-instance R_c of a multilevel relation the filter function σ produces the c'-instance $R_{c'} = \sigma(R_c, c')$ for $c' \leq c$. A tuple $t' \in R_{c'}$ if and only if t' can be derived from some $t \in R_c$ as follows:

$$\begin{array}{lcl} t'[AK, C_{AK}] & = & t[AK, C_{AK}] \\ \text{and for } i \notin AK & & \\ t'[A_i, C_i] & = & \left\{ \begin{array}{ll} t[A_i, C_i] & \text{if } t[C_i] \leq c' \\ <\text{null}, t[C_{AK}]> & \text{otherwise} \end{array} \right. \end{array}$$

□

For example, $\sigma(R_1, S)$ for the TS-relation R_1 as shown in table 8, gives us the S-relation R_2 of table 9.

It is important to clarify the semantics of null values in this context. We say t *subsumes* s if for every attribute A_i, either $t[A_i, C_i] = s[A_i, C_i]$ or $s[A_i] =$ null and $t[A_i] \neq$ null. That is t and s agree everywhere except possibly for some attributes where s has a null value and t a non-null value with the same classification. R_c is said to be *subsumption free* if it does not contain two tuples such that one subsumes the other. Subsumption of null values is required for example to have σ produce the U-instance of table 2 from the S-instances 2 through 8 of table 3. In the sequel, we assume that all multilevel relation instances and their decompositions are made subsumption free by exhaustive elimination of subsumed tuples.

The following properties of σ are easily verified.

1. $\sigma(R_c, c) = R_c$
2. For $c'' < c' < c$, $\sigma(\sigma(R_c, c'), c'') = \sigma(R_c, c'')$

The first property states that filtering a relation instance at its own level has no effect. The second states that filtering twice successively at descending levels has the same effect as filtering directly to the second level. Both properties are natural ones to expect of a filter function.

The filter function describes how the various instances R_c of a relation scheme are related as follows.

Property 5 [Inter-Instance Property] Let R_c and $R_{c'}$ be two relation instances of a given relation scheme R such that $c' < c$. Then we have that $\sigma(R_c, c') = R_{c'}$. □

It is easy to see that this requirement implies the inter-instance property of SeaView (i.e, property 1). The converse is of course not true as demonstrated by tables 8, 9 and 10.

4.2 Additional Concepts and Notation

Before describing the decompositions we need to define some notation. We will consider multilevel relations of the form $R(A_1,\ C_1,\ A_2,\ C_2,\ \ldots,\ A_n,\ C_n)$. Each A_i represents a collection of attributes that are uniformly classified (i.e., in any tuple of R_c, the values for attributes in A_i have identical security classification), and C_i is the classification attribute for A_i. We require that the A_i's be mutually disjoint. The allowable range for each C_i is denoted by $[L_i,\ H_i]$ where L_i denotes the lowest possible access class and H_i is the highest possible access class for A_i. Let $H = \text{lub}[H_i]$. The instance R_H is the one where all tuples of R are visible. The c-instance R_c at access class $c \leq H$ is therefore $\sigma(R_H, c)$.

We assume the attribute group A_1 is the user specified primary key, i.e., $A_1 = AK$. Note that entity integrity requires every tuple in R_c be A_1-total and that A_1 be uniformly classified. Let X be a subset of the data and/or classification attributes of a relation R. The notation $R[X]$ denotes the projection of R on the attributes in X. Finally, let $L = L_1$. L is the lowest access class at which tuples of R can be visible.

The nature of functional dependencies with null values needs clarification. Let X and Y be subsets of $A_1, \ldots, A_n$. A tuple t is X-total if it has no null value for attributes in X. We say the *null-valued functional dependency* (NFD) $X \rightarrow Y$ is satisfied by G if for all X-total tuples $t, t' \in G$ such that $t[X] = t'[X]$, we have that $t[Y] = t'[Y]$. Note that $t[Y]$ and $t'[Y]$ may contain nulls, and nulls are equal only to other nulls. Henceforth we understand the term functional dependency to mean NFD.

We require one final property regarding the semantics of null values. The reason for this property can be appreciated by considering the two S-instances shown in table 11. Here M$_1$ and M$_2$ are two incomparable labels dominated by S. Instance 2 does not reduce to instance 1 by the subsumption rules we have defined. At the same time R_{M_1} and R_{M_2} are identical with respect to either S-instance. Since no additional data is revealed at the S level, one would expect the lower level information at M$_1$ and M$_2$ to uniquely determine the higher level instance at S. We resolve this ambiguity in favor of instance 1 by imposing the following property, which invalidates instance 2.

Property 6 [Null Integrity] R satisfies null integrity if and only if for every instance R_c, we have for all $t, t' \in R_c$ such that $t[AK, C_{AK}] = t'[AK, C_{AK}]$:

$$t[A_i] = \text{null} \Leftrightarrow t'[A_i] = \text{null},\ \forall A_i \notin AK \qquad \square$$

In words, two polyinstantiated tuples for the same entity must either both be null or both non-null in any given non-key attribute, independent of the access class.

4.3 The Replicated Decomposition

We now present our decomposition and recovery algorithms along with proofs of correctness. It is convenient to do this in two steps. Our first decomposition is a highly redundant one in that a stored relation at access class c has data for all access classes dominated by c. This amounts to replication of low data at higher classes. This decomposition is relatively straightforward to state and prove correct. The second decomposition is an optimized one in that a stored relation at access class c has only that data which by security considerations cannot be stored below c. This decomposition can be further optimized in several ways. For the sake of simplicity and clarity we have chosen to state it so the proof will be analogous in outline to that of our first decomposition.

The replicated decomposition stores a multilevel relation as a collection of primary key groups relations and attribute group relations defined as follows.

Definition 2 [Primary Key Group Relations] Given the H-instance R_H of a multilevel relation, construct $Q_{1,c}$ for $c \in [L, H]$ as follows:

$$Q_{1,c} = \sigma(R_H, c)[A_1, C_1, C_2, \ldots, C_n] \qquad \Box$$

Definition 3 [Attribute Group Relations] Given the H-instance R_H of a multilevel relation, construct $Q_{i,c}$ for $1 < i \leq n$ and $c \in [L, H]$ as follows:

$$Q_{i,c} = \sigma(R_H, c)[A_1, C_1, A_i, C_i] \qquad \Box$$

It is obvious that this decomposition has considerable redundancy. It is almost, but not quite precisely, the case that $Q_{i,c'} \subseteq Q_{i,c}$ for $c' < c$. The difference is that null values in a tuple of $\sigma(R_H, c')$ may become non-null in $\sigma(R_H, c)$. Nevertheless it is clear there is a great deal of redundancy. Each $Q_{i,c}$ only has data elements at or below c so $Q_{i,c}$ can be stored in a storage object with label c. The reader might wonder why we do not store $\sigma(R_H, c)$. The point of course is that this is not suggested as a practical decomposition but is rather a convenient theoretical device for understanding the nature of a lossless decomposition of R_H.

The formula to recover the c-instance of the multilevel relation is remarkably simple.

Definition 4 [Recovery Formula] Recover instance R_c from this decomposition as follows, where $\bowtie$ is the natural join:

$$R_c = Q_{1,c} \bowtie Q_{2,c} \bowtie \ldots \bowtie Q_{n,c} \qquad \Box$$

To prove the correctness of these decomposition and recovery formulas we use the following result, which is a slight generalization of a well-known theorem of Risannen [20] to allow for null values. We say that a relation is X-total if all its tuples are X-total, i.e., null values are not allowed for attributes in the set X.

Lemma 1 Let $G(X, Y, Z)$ be an X-total general relation over the mutually disjoint sets of attributes X, Y and Z. Then

$$X \rightarrow Y \vee X \rightarrow Z \Rightarrow G = G[XY] \bowtie G[XZ] \qquad \square$$

Proof: Since G is X-total, it is easy to show that $G \subseteq G[XY] \bowtie G[XZ]$. To establish the reverse inclusion consider tuple $t \in G[XY] \bowtie G[XZ]$. So there exist tuples $u \in G[XY]$, and $v \in G[XZ]$ such that $t[X] = u[X] = v[X]$, $t[Y] = u[Y]$, and $t[Z] = v[Z]$. Since $G[XY]$ and $G[XZ]$ are projections of G, there must be tuples $t', t'' \in G$ such that $t'[XY] = u[XY]$ and $t''[XZ] = v[XZ]$. Since $X \rightarrow Y$ or $X \rightarrow Z$ holds in G it follows that $t'[Y] = t''[Y]$ or $t'[Z] = t''[Z]$. In the former case $t = t''$ and in the latter case $t = t'$, so $t \in G$. □

The converse of this lemma is also true but its proof would be a digression from our main objective so we omit it. Correctness of the recovery formula follows by repeated application of lemma 1.

Theorem 1 The recovery formula of definition 4 is correct.

Proof: From the definitions of $Q_{1,c}$, $Q_{i,c}$ and σ it is obvious that the recovery formula can be rewritten as

$$\begin{aligned} R_c &= R_c[A_1, C_1, C_2, \ldots, C_n] \bowtie \\ &\quad R_c[A_1, C_1, A_2, C_2] \bowtie \ldots \bowtie R_c[A_1, C_1, A_n, C_n] \end{aligned}$$

We prove this by repeated application of lemma 1. For the first application let

$$\begin{aligned} X &= \{A_1, C_1, C_2\} \\ Y &= \{A_2\} \\ Z &= \{A_3, C_3, \ldots, A_n, C_n\} \end{aligned}$$

Note that R_c is X-total due to entity integrity and non-null classification, and that $X \rightarrow Y$ due to polyinstantiation integrity. So by lemma 1 we have

$$\begin{aligned} R_c &= R_c[A_1, C_1, A_2, C_2] \bowtie \\ &\quad R_c[A_1, C_1, C_2, A_3, C_3 \ldots, A_n, C_n] \end{aligned}$$

Now consider the second term on the right hand side and again for purpose of applying lemma 1 let

$$\begin{aligned} X &= \{A_1, C_1, C_3\} \\ Y &= \{A_3\} \\ Z &= \{C_2, A_4, C_4, \ldots, A_n, C_n\} \end{aligned}$$

Again note that R_c is X-total due to entity integrity and non-null classification, and that $X \rightarrow Y$ due to polyinstantiation integrity. So now by lemma 1 we have

$$\begin{aligned} R_c &= R_c[A_1, C_1, A_2, C_2] \bowtie R_c[A_1, C_1, A_3, C_3] \bowtie \\ & \quad R_c[A_1, C_1, C_2, C_3, A_4, C_4 \ldots, A_n, C_n] \end{aligned}$$

The theorem follows by $n \Leftrightarrow 1$ applications of this procedure and finally using the commutative and associative properties of the natural join to rearrange the resulting rightmost term to be the leftmost one. □

4.4 An Optimized Decomposition

In this section we define an optimized decomposition as follows.

Definition 5 [Primary Key Group Relations] Given R_H construct $D_{1,c}(A_1, C_1, C_2, \ldots, C_n)$ for $c \in [L, H]$ as follows: for every tuple $t \in R_H$ if either $t[C_1] = c$ or $t[C_1] < c \wedge (\exists j) t[C_j] = c$ then insert t' in $D_{1,c}$ with

$$\begin{array}{lcl} t'[A_1, C_1] & = & t[A_1, C_1] \\ t'[C_i] & = & \left\{ \begin{array}{ll} t[C_i] & \text{if } t[C_i] \leq c \\ t[C_1] & \text{otherwise} \end{array} \right. \quad \text{for } 1 < i \leq n \end{array}$$

□

Definition 6 [Attribute Group Relations] Given R_H construct $D_{i,c}(A_1, C_1, A_i, C_i)$ for $1 < i \leq n$ and $c \in [L, H]$ as follows: for every tuple $t \in R_H$ such that either $t[C_1] = c$ or $t[C_1] < c \wedge t[C_i] = c$ insert t' in $D_{i,c}$ with

$$\begin{array}{lcl} t'[A_1, C_1] & = & t[A_1, C_1] \\ t'[A_i, C_i] & = & \left\{ \begin{array}{ll} t[A_i, C_i] & \text{if } t[C_i] = c \\ < \text{null}, t[C_1] > & \text{if } t[C_i] > c \end{array} \right. \end{array}$$

□

It is evident that each $D_{i,c}$ has tuples for which at least one classification attribute is c. Such tuples cannot be stored at an access class less than c. In this sense the decomposition is optimal. An example of the decomposition is given in table 12 for the TS-relation instance of table 8.

The recovery algorithm is now more complicated than for the replicated decomposition. For the sake of clarity and simplicity in our proofs we have resisted the urge to make numerous obvious optimizations. Our primary goal is to enable a rigorous proof of correctness.

Definition 7 [Recovery Algorithm] Recover instance R_c from this decomposition as follows:

1. $P_{i,c} = \bigcup_{c' \leq c} D_{i,c'}$.
2. Let $R_c = P_{1,c} \bowtie P_{2,c} \bowtie \ldots \bowtie P_{n,c}$. □

An example of this algorithm applied to the decomposition of table 12 is shown in table 13. The example shows recovery of both the S- and TS-instances which are exactly R_2 of table 9 and R_1 of table 8 respectively. The reader may wish to verify that the algorithm applied to the decomposition of the TS-relation of table 4 will materialize exactly 1, 2, 3 and 4 tuples respectively in recovering the U-, C-, S- and TS-instances.

The proof of correctness is based on the following observations.

1. $P_{1,c} = \bigcup_{c' \leq c} Q_{1,c}$.

2. For $1 < i \leq n$, $P_{i,c} = Q_{i,c}$.

The proof then follows from theorem 1 and the fact that the additional tuples in $P_{1,c}$ as compared with $Q_{1,c}$ do not contribute any new tuples to the computation of R_c. The details of establishing these properties are quite intricate and non-trivial. They are omitted here for lack of space.

5 CONCLUSION

In summary, we propose that the MVD component of polyinstantiation integrity in SeaView be dropped to allow for a larger class of multilevel relations than currently permitted. We have shown that the additional relations allowed by doing this are practically useful. We have also shown that the very desirable result of SeaView that multilevel relations can be stored as single level base relations continues to be true, and can in fact be achieved by using natural joins for recovery.

In terms of future work much remains to be done. The efficiency of the recovery algorithm is clearly crucial to the query response time. It is therefore important to consider further optimizations to our recovery algorithm. Our immediate goal in this paper has been not so much to find a better, perhaps optimal, recovery algorithm, but to provide a conceptual framework for dealing with multilevel real relations as a collection of single-level base relations.

Since we decompose a multilevel real relation as a collection of single-level base relations, it remains to show that an update to a multilevel relation can be correctly translated into equivalent updates to base relations, and conversely. This will provide a formal basis for the updatability of multilevel relations *vis-a-vis* base relations. A formal consideration of updates is also necessary to show that the data model does not contain covert channels.

We are also examining a completely different approach of decomposing multilevel real relations into single level base relations. The solution involves associating a unique tuple identifier with each tuple in a multilevel real relation. The approach is promising and works correctly for the cases we have examined, although a thorough analysis is yet to be performed.

Acknowledgement

We are indebted to John Campbell, Joe Giordano, and Sylvan Pinsky for their support and encouragement, making this work possible. We also wish to thank Teresa Lunt and Gary Smith for their helpful comments and conversations. 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] Codd, E.F. "A Relational Model of Data for Large Shared Data Banks." *Communications of ACM* 13(6): (1970).

[2] Codd, E.F. "Extending the Relational Database Model to Capture More Meaning." *ACM Transactions on Database Systems* 4(4): (1979).

[3] Date, C.J. *An Introduction to Database Systems.* Volume I, Addison-Wesley, fourth edition (1986).

[4] 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).

[5] 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).

[6] Denning D.E. "Lessons Learned from Modeling a Secure Multilevel Relational Database System." In [12], 35-43 (1988).

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

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

[9] Garvey C. "Multilevel Data Storage Design." TRW Defense Systems Group (1986).

[10] Grohn M.J. "A Model of a Protected Data Management System." Technical Report ESD-TR-76-289, I.P. Sharp Associates Ltd., (1976).

[11] Hinke T.H. and Schaefer M. "Secure Data Management System." Technical Report RADC-TR-75-266, System Development Corporation (1975).

[12] Landwehr, C.E. (Editor) *Database Security: Status and Prospects.* North-Holland (1988).

[13] Lunt, T.F., Denning, D.E., Schell, R.R. Heckman, M. and Shockley, W.R. "Element-Level Classification with A1 Assurance." *Computers & Security*, Feb. 1988.

[14] Lunt, T.F., Schell, R.R., Shockley, W.R., Heckman, M. and Warren, D. "A Near-Term Design for the SeaView Multilevel Database System." *IEEE Symposium on Security and Privacy*, 234-244 (1988).

[15] Lunt, T.F. and Whitehurst, R.A. "The SeaView Formal Top Level Specifications." SRI Project 1143, A007: Final Report, Volume 3A (1989).

[16] Lunt, T.F., Denning, D.E., Schell, R.R. Heckman, M. and Shockley, W.R. "Secure Distributed Data Views. Volume 2: The SeaView Formal Security Policy Model." SRI-CSL-88-15 (1989).

[17] Lunt, T.F., Denning, D.E., Schell, R.R., Heckman, M. and Shockley, W.R. "The SeaView Security Model." *IEEE Transactions on Software Engineering*, to appear.

[18] Lunt, T.F. and Hsieh, D. "The SeaView Secure Database System: Some Critical Issues." Unpublished manuscript, SRI International (1989).

[19] Maier, D. *The Theory of Relational Databases.* Computer Science Press (1983).

[20] Rissanen, J. "Independent Components of Relations." *ACM Transactions on Database Systems* 2(4):317-325 (1977).

Table 1: A scheme for the multilevel relation SOD.

Table 2: A U-instance of SOD.

Table 3: Some S-instances of SOD consistent with the U-instance of table 2.

Table 4: A TS-instance of SOD with 4 missions.

Table 5: The SeaView materialization with 16 missions.

Table 6: An S-instance of SOD with 2 starships.

Table 7: An S-instance of SOD violating polyinstantiation integrity.

Table 8: Relation R_1.

Table 9: Relation R_2.

Table 10: Relation R_3.

Table 11: Relation R_3.

Table 12: Decomposition of relation R_1.

Table 13: Recovery of the S- and TS- instances.