Proc. 4th RADC Workshop on Multilevel Database Security, Rhode Island, April 1991.

ENFORCING PRIMARY KEY REQUIREMENTS IN MULTILEVEL RELATIONS

Sushil Jajodia and Ravi S. Sandhu

Center for Secure Information Systems
and
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444

1 INTRODUCTION

The notion of a primary key is considered a fundamental concept in the classical (single-level) relational model. For example, it forms the basis for several normal forms and is used when the database schema is designed. The primary key is used to maintain integrity of relations. It is also used for storage and retrieval purposes.

Unfortunately, the concept of a primary key does not extend to multilevel relations in a straightforward way because of two factors: (a) the $\star$-property must be preserved which prevents any write downs, and (b) signaling channels must be avoided. These security considerations have lead to the notion of *polyinstantiation* in multilevel relations [2].

Polyinstantiation comes in several different flavors [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. There are significant differences between these approaches and debate continues about the correct definition of polyinstantiation and its operational semantics. However, in each case polyinstantiation fails to preserve the basic requirement of a primary key that there be one and only one tuple per primary key value in a relation. Polyinstantiation forces a relation to contain multiple tuples with the same primary key, distinguishable by their classifications or by non-primary key attribute values.

Since polyinstantiation significantly complicates the semantics of multilevel relations (particularly for high users), recently some solutions have appeared which attempt to do away with polyinstantiation completely [1, 13, 14]. In this paper, we take another step along this direction, and examine ways to preserve primary key requirements in multilevel relations. Of course, any solution we give will have to be secure and free of denial-of-service problem.

The organization of the remainder of this paper is as follows. In section 2 we briefly review the notion of primary key in classical (single-level) relations. In section 3 we show how polyinstantiation arises in multilevel relations. In section 4 we explore alternatives to polyinstantiation that help us enforce primary key requirements in multilevel relations. Finally, the conclusion is given in section 5.

2 PRIMARY KEY IN SINGLE-LEVEL RELATIONS

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

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

Figure 1: $\mathrm{SOD_S}$

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

Figure 2: $\mathrm{SOD_U}$

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

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

The notion of primary key is central to the relation model. Before we define it, however, we need to give a definition for a candidate key:

We say $X \subseteq \{A_1, A_2, \ldots, A_n\}$ is a *candidate key* of R if any relation r for R at all times satisfies the following two properties:

Uniqueness Property. The relation r does not contain two distinct tuples with the same values for X.

Minimality Property. No proper subset Y of X satisfies this uniqueness property.

A *primary key* of a relation scheme R is a candidate key of R. It is possible that a relation scheme has more than one candidate key, in which case one of the candidate keys must be chosen and designated as the primary key.

3 POLYINSTANTIATION

While the notion of a primary key is simple and well understood for classical (single-level) relations, it does not have a straightforward extension to multilevel relations. To illustrate, consider the relation scheme SOD(Starship, Objective, Destination) where Starship is the primary key and the security classifications are assigned at the granularity of individual data elements. Suppose the Secret and Unclassified views of SOD are as shown in figures 1 and 2, respectively.

Suppose that an U-user* who sees the instance in figure 2 wishes to insert a second tuple (Voyager, Exploration, Talos) to $\mathrm{SOD_U}$. If we were to enforce the primary key requirement, this insertion by the U-user will be rejected (since it conflicts with an existing tuple in $\mathrm{SOD_S}$). However, since this rejection will create a signaling channel, both tuples (Voyager, Spying, Mars) and (Voyager, Exploration, Talos) are allowed to co-exist in $\mathrm{SOD_S}$, as in figure 3, in violation of the uniqueness requirement. This is one type of polyinstantiation, called *entity polyinstantiation*: A relation contains

*Strictly speaking we should be saying subject rather than user.

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

Figure 3: SOD_S

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

Figure 4: SOD_S

two or more tuples with the same primary key values, but having different access class values for the primary key.

There is another form of polyinstantiation, called *element polyinstantiation*. With element polyinstantiation, a relation contains two or more tuples with identical primary key and the associated access class values, but having different non-primary key values, as shown in the relation in figure 4. The objectives and destinations of the starship Enterprise are different for U- and S-users.

Both entity and element polyinstantiation can occur in basically two different ways:

1. A high user attempts to update data which conflicts with the existing low data. Since overwriting the low data in place will result in a downward signaling channel, the tuple is polyinstantiated

2. An opposite situation occurs where a low user attempts to insert data which conflicts with existing high data. Since rejecting the update is not a viable option because it establishes a downward signaling channel, the updated tuple is polyinstantiated to reflect the low update.

4 PRIMARY KEY IN MULTILEVEL RELATIONS

In this section, we explore how we can enforce primary key requirements in multilevel relations without creating a downward signaling channel in the process.

4.1 A Simple Solution

There is a completely obvious way to preserve primary key requirements in multilevel relations.

1. Whenever a high user makes an update which violates the uniqueness requirement, we simply refuse that update.

2. Whenever a low user makes a change with conflicts with the uniqueness requirement, the conflicting high data is overwritten in place by the low data.

Returning to the example of the previous section, when the U-user inserts the second tuple (Voyager, Exploration, Talos) to SOD_U shown in figure 2, the conflicting second tuple in SOD_S in

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

Figure 5: $SOD_S = SOD_U$

figure 1 is substituted by the newly inserted tuple. As a result both U- and S- users see the instance shown in figure 5. On the other hand, suppose a S-user wants to insert the tuple (Voyager, Spying, Mars) to SOD_S in figure 5. This update is simply refused. Thus, in both cases primary key values in SOD uniquely identify the tuples in the multilevel relations.

It is not difficult to see that this simple solution preserves the uniqueness requirement in multilevel relations. This solution is secure in the sense of secrecy and information flow. It is our view that while this solution may be acceptable in some specific situations, it is clearly unacceptable as a general solution; it can lead to serious denial-of-service and integrity problems. Therefore, we now look for other alternatives which do not suffer from these problems.

4.2 Dealing with Entity Polyinstantiation

4.2.1 Single Access Class for the Primary key

A multilevel relation is created by using a data definition statement, similar to the following statement:[†]

```
CREATE TABLE SOD ( Starship     CHAR(15) NOT NULL [U:S],
                   Objective    CHAR(15) {U, TS},
                   Destination  CHAR(20) [U:TS],
                   Primary Key (Starship ) );
```

Here the domain of the access class of the primary key Starship has been specified as a range with a lower bound of U and an upper bound of S. As we saw in the previous section, this leads to entity polyinstantiation. Thus, one simple way of eliminating entity polyinstantiation is to have the domain of the access class of the primary key consist of a single element.

Thus if we create the SOD relation as follows, SOD will not have any entity polyinstantiation.

```
CREATE TABLE SOD ( Starship     CHAR(15) NOT NULL {U}
                   Objective    CHAR(15) {U, TS},
                   Destination  CHAR(20) [U:TS],
                   Primary Key (Starship ) );
```

It is possible that in some situation names of some starships must remain Top Secret, in such a case we can use the following solution.

4.2.2 Partitioning the Domain of the Primary Key

Another way to eliminate entity polyinstantiation is to partition the domain of the primary key among the various access classes possible for the primary key. For our example, we can introduce a

[†]The notation [L:H] specifies a range of security classes with lower bound L and upper bound H. The notation {X,Y,Z} enumerates the allowed values for the security class as one of X, Y or Z.

new attribute, called Starship#. Whenever a new tuple is inserted, we enforce the requirement that all the Starships numbered between 1 and 1,000 will be unclassified, those numbered between 1,001 and 2,000 will be confidential, and so on.

In SQL-like language, the SOD schema should be created as follows:

```
CREATE TABLE SOD
  ( Starship#    SMALL INTEGER NOT NULL [U:TS]
    Starship     CHAR(15) NOT NULL [U:TS]
    Objective    CHAR(15) {U, TS},
    Destination  CHAR(20) [U:TS],
    Primary Key (Starship# ),
    CHECK (User Access class = 'U'  AND Starship# BETWEEN 1    AND 1000),
    CHECK (User Access class = 'C'  AND Starship# BETWEEN 1001 AND 2000),
    CHECK (User Access class = 'S'  AND Starship# BETWEEN 2001 AND 3000),
    CHECK (User Access class = 'TS' AND Starship# BETWEEN 3001 AND 4000) );
```

4.3 Dealing with Element Polyinstantiation

It is possible to eliminate element polyinstantiation securely without sacrificing either integrity or availability. We show how this is done for the SOD example. For complete details, we refer the reader to [13]. This solution by Sandhu and Jajodia meets the following requirements.

1. There are no downward signaling channels.
2. The simple security and the $\star$-properties is enforced for all subjects, i.e., no trusted code can be used.
3. There are no temporary inconsistencies.
4. There is no denial of data entry service to high users.

Consider once again the following relation SOD where Starship is the primary key. We assume that the Starship attribute is always unclassified, so there is no entity polyinstantiation.

Starship	Objective	Destination	TC
Enterprise U	Exploration U	Talos U	U

Consider the following scenario. Suppose a S-user attempts to modify the destination of the Enterprise to be Rigel. We cannot polyinstantiate even temporarily, so we must reject this update. There is no denial-of-service to the S-user since the S-user can obtain service as follows.

Step 1. The S-user first logs in as a U-subject and marks the destination of the Enterprise as restricted giving us the following relation.

Starship	Objective	Destination	TC
Enterprise U	Exploration U	restricted U	U

The meaning of restricted is that this field can no longer be updated by a U-user. U-users can therefore infer that the true value of Enterprise's destination is classified at some level not dominated by U.

Step 2. The S-user then logs in as a S-subject and enters the destination of the Enterprise as Rigel giving us the following relations at the U and S levels, respectively.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	S	S

One can argue that step 2 introduces an signaling channel. Fortunately, this is not a particularly harmful channel. Here a trusted S-user is in the loop who presumably will ensure that the channel is not exercised wantonly, but rather that this inference is permitted only when the real world situation is actually so. Such a channel with trusted humans in the loop can be exercised only by Trojan Horses that are capable of manipulating the real world. This entails the manipulation of real trusted people making real decisions and not merely the manipulation of bits in a database.

We refer the reader to [13] for additional examples and complete details.

5 CONCLUSION

In this paper we have shown that there are ways to enforce primary key requirements in multilevel relations. The methods we have listed eliminate problems that arise from polyinstantiation completely. These methods may be eminently suitable in many applications.

Yet we wish to remind the reader that there are situations where polyinstantiation is desirable. There is a real need for cover stories in the multilevel world, and polyinstantiation provides a simple way of satisfying this need. Moreover, we envision applications (particularly in an intelligence environment) where information is coming from different sources, bearing different classification. The information may sometimes be contradictory; however, it must be stored in the database. An analyst can make sense out of the confusing mess. It is desirable to have polyinstantiation for such situations.

References

[1] Rae K. Burns, "Referential Secrecy." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1990, pages 133-142.

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

[3] J. T. Haigh, R. C. O'Brien, and D. J. Thomsen, "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.

[4] Sushil Jajodia and Ravi S. Sandhu, "Polyinstantiation integrity in multilevel relations." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1990, pages 104-115.

[5] Sushil Jajodia and Ravi S. Sandhu, "A formal framework for single level decomposition of multilevel relations." *Proc. IEEE Workshop on Computer Security Foundations*, Franconia, New Hampshire, June 1990, pages 152-158.

[6] Sushil Jajodia and Ravi S. Sandhu, "Polyinstantiation integrity in multilevel relations revisited." *Database Security IV: Status and Prospects*, S. Jajodia and C. E. Landwehr (editors), North-Holland, 1991, pages 297-307.

[7] Sushil Jajodia, Ravi S. Sandhu, and Edgar Sibley, "Update semantics of multilevel relations." *Proc. 6th Annual Computer Security Applications Conf.*, December 1990, pages 103-112.

[8] Sushil Jajodia and Ravi S. Sandhu, "A novel decomposition of multilevel relations into single-level relations." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1991, pages 300-313.

[9] Sushil Jajodia and Ravi S. Sandhu, "Toward a multilevel secure relational data model," *Proc. ACM SIGMOD Int'l. Conf. on Management of Data,* Denver, Colorado, May 29-31, 1991, pp. 50-59.

[10] Teresa F. Lunt, Dorothy E. Denning, Roger R. Schell, Mark Heckman, and William R. Shockley, "The SeaView security model." *IEEE Transactions on Software Engineering*, Vol. 16, No. 6, June 1990, pages 593-607.

[11] Teresa F. Lunt and Donovan Hsieh, "Update semantics for a multilevel relational database." *Database Security IV: Status and Prospects*, S. Jajodia and C. E. Landwehr, (editors), North-Holland, 1991, pages 281-296.

[12] Ravi S. Sandhu, Sushil Jajodia, and Teresa Lunt, "A new polyinstantiation integrity constraint for multilevel relations." *Proc. IEEE Workshop on Computer Security Foundations*, Franconia, New Hampshire, June 1990, pages 159-165.

[13] Ravi S. Sandhu and Sushil Jajodia, "Honest databases that can keep secrets," *Proc. 14th NIST-NCSC National Computer Security Conference*, Washington, D.C., October 1991, To appear.

[14] S. R. Wiseman, "On the Problem of Security in Data Bases." In *Database Security III: Status and Prospects*, (Spooner, D.L. and Landwehr, C.E., editors), North-Holland, 1990 pages 143-150.