Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Data & Knowledge Engineering 48 (2004) 129–152
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

The semantics of an extended referential integrity for a multilevel secure relational data model

Sang-Won Lee [a,*,1], Yong-Han Kim [b], Hyoung-Joo Kim [c]

[a] *School of Information and Communication Engineering, SungKyunKwan University, Chunchun 300, Jangan-Gu, Suwon, 440-746 South Korea*
[b] *Oracle Korea, Seoul, South Korea*
[c] *Computer Science and Engineering, Seoul National University, Seoul, South Korea*

Received 21 August 2002; received in revised form 22 January 2003; accepted 26 May 2003

Abstract

To prevent information leakage in multilevel secure data models, the concept of polyinstantiation was inevitably introduced. Unfortunately, when it comes to references through foreign key in multilevel relational data models, the polyinstantiation causes referential ambiguities. To resolve this problem, this paper proposes an extended referential integrity semantics for a multilevel relational data model, Multilevel Secure Referential Integrity Semantics (MLS-RIS).

The MLS-RIS distinguishes foreign key into two types of references, i.e. value-based and entity-based reference. For each type, it defines the referential integrity to be held between two multilevel relations, and provides resolution rules for the referential ambiguities. In addition, the MLS-RIS specifies the semantics of referential actions of the SQL update operations so as to preserve the referential integrity.

Keywords: Security; Multilevel secure relational data model; Referential integrity; Polyinstantiation

* Corresponding author.
E-mail addresses: swlee@acm.org (S.-W. Lee), yonghan.kim@oracle.com (Y.-H. Kim), hjk@oopsla.snu.ac.kr (H.-J. Kim).

[1] This work was partially supported by grant no. R05-2003-000-11943-0 from Korea Science and Engineering Foundation.

0169-023X/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0169-023X(03)00122-8

1. Introduction

A database management system (DBMS) should have an access control mechanism ensuring that only authorized users can access the shared data according to a specific policy. The function of an access control mechanism can be abstracted as follows.

$$F(S, O, T) = \text{yes or no}$$

Each symbol of S, O, and T represents user, data, and access type (including read and write), respectively. When user S requires access to data O in T mode, the function F decides whether or not to allow the request.

There are two representative access control policies for database systems: (1) *discretionary access control* (DAC) and (2) *mandatory access control* (MAC). DAC controls the access to data on the basis of the modes of access privileges of users to data. It is called discretionary in the sense that the owner of data has complete discretion regarding granting/revoking of access privilege to his/her data. DAC was originally invented for System/RDBMS by Griffiths and Wade [13], and later revised by Fagin [10]. Currently, almost all commercial relational DBMSs (RDBMSs) support DAC.

Meanwhile, MAC controls the access to data on the basis of security labels assigned to users and data. It is mandatory in that the access to data is rigidly allowed 'only if' some condition (depending on the access type) between the security label of the user (also called *clearance*) and that of data (also called *classification level*) is satisfied. MAC was introduced to overcome the limitations of DAC such as Trojan Horse [3,26], and during the past decade there have been several efforts to build secure relational DBMSs, such as SeaView project [21], LDV model [31] and Jajodia–Sandhu model [17,19]. Some commercial RDBMSs such as Oracle [23,24] are now delivering products supporting MAC. Even though other major commercial DBMSs, such as IBM DB2 and MS SQL Server, do not support MAC, we foresee that they will also incorporate the functionality of MAC into their future releases, as the security requirements from mission critical applications get stronger than ever.

With regard to MAC, it is noteworthy that recently many real-time database applications, including military systems, have been paying much attention to the multilevel security paradigm [8,12,30]. In these areas where security enforcement is crucial to the success of the enterprise, the multilevel security paradigm is preferred in order to guarantee security within their rigid temporal requirements. More recently, the MAC approach inspired the Hippocratic Databases [1] which will rejuvenate the interest in database security and privacy.

Currently, other research efforts on database security include, to name a few, (1) extensions of current DAC policy for RDBMSs in order to directly support a variety of application security policies [4,5]; (2) developments of authorization models for object-oriented DBMSs using DAC [25] and MAC [32] and for federated databases [34] and (3) works on role-based access control (RBAC) model [27], emerging as a candidate for filling the gap between DAC and MAC. Refer to [3,26] for details.

In the remainder of this paper, we will use the term *multilevel security*, instead of the term *MAC*. MAC is called multilevel security in the sense that to each user and each data item, one of the multiple security labels can be assigned.

It should be noted that access control policies are not necessarily exclusive. Different policies can be combined to provide a more suitable protection system. Each policy represents a policy which allows a subset of all possible accesses. When the policies are combined, only the intersection of their accesses is allowed.

1.1. Our goals

For the last decade, there have been several efforts on the multilevel relational data model. However, many of them focused on the semantics within a single relation; (1) definition of relation scheme and relation instance, (2) integrity properties for a single relation, and (3) operational semantics of insert, delete, and update. Little attention has been paid to the relationships between two multilevel relations, such as referential integrity which is an essential semantic of the standard relational data model. Only a few works touched upon the definition of referential integrity in multilevel relational data models. However, these works do not address the issues considering the referential ambiguity in the multilevel relational data model.

Based on this fact, we propose an extended referential integrity semantic for a multilevel relational data model, called Multilevel Secure Referential Integrity Semantics (MLS-RIS). For this, we extend the referential integrity semantics of the SQL standard [16].

1.2. Paper outline

The next section gives an overview of the basic concepts: multilevel relational data model for single relation and the referential integrity in standard SQL. Section 3 defines the problem of referential ambiguity, reviews the previous solutions, and gives the basic ideas of MLS-RIS. Sections 4 and 5 gives the formal semantics of *valued-based reference* and *entity-based reference*, respectively. After comparing MLS-RIS with the related works in Section 6, we conclude the paper in Section 7.

2. Basic concepts

2.1. Multilevel secure data model

In the multilevel secure data model, each data object (called *object*) is assigned a security class, and each user (called *subject*) is assigned a clearance for a security class. We will call the class of an *object* or a *subject* A as its label and denote it as $L(A)$. The security classes in a multilevel secure data model can be organized into a lattice. In this paper, for the sake of simplicity, we will assume the linear sequence of security classes: $U(unclassified) < C(confidential) < S(secret) < TS(top\text{-}secret)$, instead of a lattice structure. The notation $L(B) < L(A)$ means that the security class of A is higher than that of B, and $L(B) \leqslant L(A)$ that the security class of A is equal to or higher than that of B.

Access control in multilevel secure data model is based on the Bell–LaPadula model [2], which imposes two properties on all reads and writes of database objects by users

1. *The Simple Security Property*. A user s is allowed to access an object o only if $L(s)$ is higher than or identical to $L(o)$, that is,

   $F(s, o, R) =$ yes and only if $L(s) \geqslant L(o)$

2. *The $*$-Property*. A user s is allowed a write access to an object o only if $L(s)$ is identical to or lower than $L(o)$, that is,

   $F(s, o, W) =$ yes and only if $L(s) \leqslant L(o)$

The goal of *The Simple Security Property* is to prevent a user with low clearance from accessing higher data (that is, *No Read-Up*), while the goal of *The $*$-Property*, as shown in Fig. 1, is to prevent a malicious user with a high clearance from passing classified data to a user cleared at a lower level (that is, *No Write-Down*).

2.2. Multilevel relational data model

A multilevel relational data model is no more than an extended relational model to guarantee the two properties of the Bell–LaPadula model. This section reviews the multilevel relation data model we will assume in this paper: (1) the basic definitions of multilevel relations, (2) the concept of polyinstantiation, which is inevitable in multilevel security, and (3) the four core integrity properties. We assume the readers are familiar with the standard relational model.

2.2.1. Multilevel relations

Even though we assume that the readers are familiar with basic concepts of standard relational model, we quickly review the basic definitions of relational model before moving on to multilevel relation world. The main construct for representing data in the standard relational model is a relation. A relation consists of a *relation schema* and a *relation instance*. The schema specifies the relation's name, the name of each attribute (or column), and the domain of each attribute. A relation schema is denoted as $R(A_1, \ldots, A_n)$, where each A_i is an attribute. An instance of a relation is a set of tuples, in which each tuple has the same number of attributes as the relation schema. Each tuple has the form of $(a_1, a_2, \ldots, a_n)$. A relation instance can be thought of as a table. The term relation instance is often abbreviated to just relation, where there is no confusion with the schema of the relation.

In analogy to the definition of relation in standard relational model, a multilevel relation consists of two parts: (1) the state-invariant relation scheme and (2) a collection of state-depen-

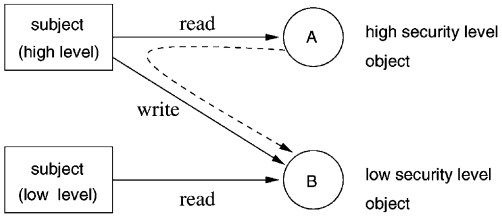

Fig. 1. Illegal information flow in multilevel secure data model.

SHIP		MISSION		DEST		TC
Pathfinder	U	Exploration	U	Mars	U	U
Pathfinder	U	Nuclear Test	S	Mars	U	S
Pathfinder	C	Exploration	C	Sun	C	C
Cassini	TS	Exploration	TS	Saturn	TS	TS

Fig. 2. A multilevel relation *SMD*.

SHIP		MISSION		DEST		TC
Pathfinder	U	Exploration	U	Mars	U	U
Pathfinder	C	Exploration	C	Sun	C	C

Fig. 3. A multilevel relation instance SMD_C.

dent relation instances. *A multilevel relation schema* is denoted as $R(A_1, C_1, \ldots, A_n, C_n, TC)$, where each A_i is a *data attribute*, each C_i is a *classification attribute* for A_i, and TC is the *tuple-class attribute* [9,17,28]. We assume the primary data attribute consists of only one data attribute and denote it as PK. Its corresponding classification attribute is denoted as C_{PK}.

For each access class c in the given classification level, a multilevel relation has a corresponding *relation instance*, R_c. In other words, for a given multilevel relation schema, there exist as many multilevel relation instances as access classes. For a same multilevel relation, users with different access classes see a different collection of rows—that is, different multilevel relation instances according to their access classes. In this respect, the readers can regard the concept of multilevel relation instances as a special kind of views. Each multilevel relation instance R_c has a set of distinct tuples of the following form.

$$R_c(a_1, c_1, \ldots, a_n, c_n, tc)$$

Given a tuple t, $t[A_i]$ and $t[C_i]$ $(i = 1, \ldots, n)$ represents the data value of data attribute A_i, classification level of C_i, respectively, and $t[TC]$ represents tuple-class TC of tuple t. The tuple-class $t[TC]$ is equal to $lub\{t[C_1], t[C_2], \ldots, t[C_n]\}$, where *lub* denotes the least upper bound.

Fig. 2 shows a multilevel relation *SMD* which will be used in the rest of this paper. *SMD* has three data attributes, SHIP (spaceship name), MISSION (mission), and DEST (destination). Besides data attributes, it has the tuple class attribute TC also. Fig. 2 does not explicitly model the classification of each attribute. Instead, the security class of each attribute is shown right to its data value. In fact, the table in Fig. 2 is the multilevel relation instance SMD_{TS}. According to *The Simple Security Property* of the Bell–LaPadula model, a multilevel relation should be differently viewed to the different users depending on their clearances. For instance, a user with C clearance will see the filtered relation instance SMD_C as shown in Fig. 3, while a TS user will see the entire relation of Fig. 2. [2] The relation instance SMD_C is a set of tuples with the tuple-class less than or identical to C.

[2] The tables SMD_{TS} and SMD_C do not need to exist as a separate table in disk. Using the techniques such as [19], we can decompose the contents of tables into several physical relations, and can dynamically generate a appropriate logical relation according to a user clearance.

2.2.2. Polyinstantiation

Polyinstantiation refers to the simultaneous existence of multiple data with the same value, but with different classification level. In general, polyinstantiation is assumed to be an inevitable part of the multilevel security paradigm [20]. There are two types of polyinstantiation in a multilevel relational data model; *polyinstantiated tuple* (or *polyinstantiated entity*) and *polyinstantiated attribute*. The polyinstantiated attribute is used to model a same real-world entity, an attribute of which has different values at different classification levels. For example, the first two tuples in Fig. 2 model polyinstantiated attributes of the data attribute `MISSION` of the same `Pathfinder` entity; that is, while the mission of the ship `Pathfinder` is known as `Nuclear Test` by *TS*-users, with users having lower clearance its mission is known just as `Exploration`.

The polyinstantiated tuple was introduced to model two or more different real-world entities with the same primary key value. For instance, suppose that a user with low clearance asks to insert a tuple with the same primary key as an existing tuple at a higher level. The DBMS should not reject this insertion, because otherwise it would inform the low-level user of the existence of a higher-level tuple with the same key value, thereby resulting in information leakage. The third tuple with key value `Pathfinder` in Fig. 2, represents a different entity from the entity modeled by the first two tuples with key value `Pathfinder`. This type of polyinstantiated tuples is called as *required polyinstantiation* [18].

There is another type of polyinstantiated tuple, which is contrary to the required polyinstantiation. Suppose that a user asks to insert a tuple with the same primary key as an existing tuple at a lower level. In this case, the user already knows that there is a lower class tuple with the same key value, and moreover there is no information leakage even if we allow the insertion. It is a valid option to reject the insertion, but this is a denial-of-service [3]. We call this type of polyinstantiated tuples *optional polyinstantiation*. In this paper, we assume both types of polyinstantiated tuples.

In summary, polyinstantiated tuples were introduced to model the different real-world entities, while polyinstantiated attributes were introduced to represent the different values of a data attribute of the same entity at different classification levels. In a multilevel relational data models, the concept of polyinstantiation is inevitably introduced to guarantee the $*$-property of the Bell–LaPadula model. For more details about polyinstantiation, refer to [3,18,20].

2.2.3. Integrity properties

Many of the multilevel relational data models includes integrity properties that each multilevel relation should satisfy [9,17,28]. These integrity properties are required from three characteristics of the multilevel relational data model. First, since the multilevel relational data model itself is an extension of the standard relational data model, the integrity properties required in relational data model should also be satisfied in the multilevel relational data models. Second, because, for a relation scheme, there are multiple relation instances according to the security level, several new integrity properties about inter-instance relationships should be introduced. Finally, the concept of polyinstantiation requires another type of integrity properties.

It is beyond the scope of our work to introduce a new set of integrity properties for a single multilevel relation, so we assume in this paper the integrity properties of [28] which we think are the simplest and the most advanced. In this section, we will briefly review the integrity properties defined in [28].

SHIP		MISSION		DEST		TC
Pathfinder	U	Exploration	U	Mars	U	U
Pathfinder	U	Nuclear Test	S	Jupiter	U	S

Fig. 4. Violation of polyinstantiation integrity: an example.

Integrity property 1 (*Entity integrity*). *A relation instance R_c of a multilevel relation R satisfies entity integrity if and only if the following requirements hold for all $t \in R_c$.* [3]

1. $t[PK] \neq null$,
2. $A_i \neq PK \rightarrow t[C_i] \geqslant t[C_{PK}]$.

The first requirement is exactly the same as the entity integrity of the standard relational model, and the second one says that in any tuple of R_c the class of non-key attributes must dominate C_{PK}.

Integrity property 2 (*Inter-instance integrity*). *A multilevel relation R satisfies inter-instance integrity if and only if the following requirement holds*

1. $\forall c' \leqslant c$, $R_{c'} = \{t \in R_c \mid t[TC] \leqslant c'\}$.

Relation instance $R_{c'}(c' \leqslant c)$ consists of tuples t from R_c, where $t[TC]$ is lower than or equal to c'. For instance, a user with *TS* level sees the relation instance of Fig. 2, while a *C*-level user sees the relation instance of Fig. 3, which consists of tuples whose *tuple-class* are *U* or *C*.

Integrity property 3 (*Polyinstantiation integrity*). *A multilevel relation R satisfies polyinstantiation integrity if and only if the following functional dependency holds for all relation instance R_cs of R.*

1. $PK, C_{PK}, C_i \rightarrow A_i$ for all $i = 1, \ldots, n$.

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. The intuitive meaning of this integrity property is that all the tuples modeling polyinstantiated attributes of the same real-world entity should have the same data value of a data attribute A_i if their classification levels are identical. Fig. 4 shows an example of violation of this polyinstantiation integrity. Two tuples in Fig. 4 model the same entity `Pathfinder`, but they have different `DEST` values even though their C_{DEST}s are identical.

Integrity property 4 (*PI-tuple-class integrity*). *A multilevel relation R satisfies tuple-class polyinstantiation integrity if and only if the following requirement holds for all relation instance R_cs of R.*

1. $PK, C_{PK}, TC \rightarrow A_i$ for all $A_i \notin PK$.

[3] In [28], they assume that, when the primary key consists of more than one attribute, the security class of all the attributes are same. Since, in this paper, we assume a single-attributed primary key, we omit this condition.

SHIP		MISSION		DEST		TC
Pathfinder	U	Exploration	U	Mars	U	U
Pathfinder	U	Nuclear Test	S	Mars	U	S
Pathfinder	U	Exploration	U	Saturn	S	S

Fig. 5. Violation of PI-tuple-class integrity: an example.

This integrity property implies that, for a given classification level, there can be only one tuple modeling a real-world entity. For instance, consider the relation instance in Fig. 5. This relation instance violates PI-tuple-class integrity, since the last two tuples model the same `Pathfinder`, but at the same time their tuple-classes are equal, that is, there exist two tuples modeling an entity at the same classification level.

At this point, we should note that polyinstantiation integrity and PI-tuple-class integrity are orthogonal to each other. The relation instance in Fig. 5 violating PI-tuple-class integrity satisfies polyinstantiation integrity. In contrast, the relation instance in Fig. 4 violating polyinstantiation integrity, satisfies the PI-tuple-class integrity.

2.3. Referential integrity in the standard relational data model

Referential integrity, in addition to entity integrity, is an essential component of the relational data model. The relational database standards such as SQL2 [15] and SQL3 [16], specify the referential integrity in detail, including its definition and the semantics of run-time referential actions. For detailed explanations about the referential integrity in SQL3 and its support in commercial DBMSs, see a recent excellent survey paper [33].

Simply stated, referential integrity says that a value that appears in one relation (R_C) for a given set of attributes ($R_C.FK$) also should appear for primary key ($R_P.PK$) in another relation (R_P). R_P and R_C are called 'parent relation' and 'child relation', respectively, and $R_C.FK$ and $R_P.PK$ are called 'foreign key' and 'primary key', respectively. A foreign key can have null value when there is no matching value in the parent relation or when its value is unknown. For a tuple t_{R_P} in a parent relation R_P, there may exist several tuples t_{R_C} in child relation R_C, which references t_{R_P}. The tuple t_{R_P} is called parent tuple, and each t_{R_C} child tuple.

Referential integrity 1 (*referential integrity in SQL standard*). For every tuple $t_{R_C} \in R_C$, where $t_{R_C}[FK] \neq null$, there should exist a tuple t_{R_P} in R_P, where $t_{R_P}[PK]$ is equal to $t_{R_C}[FK]$.

The following syntax is used, as a part of child relation definition, to declare a foreign key in SQL3 [16]. It should be noted that the syntax also provides the ways to declare the referential actions of the foreign key with regarding to update and delete operations in the parent relation.

```
FOREIGN KEY [(<referencing columns>)]
  REFERENCES [PENDANT] <table name> [(<referenced columns>)]
  [MATCH FULL | MATCH PARTIAL]
  [ON UPDATE {CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION }]
  [ON DELETE {CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION }]
```

`<referencing columns>` declares foreign key $R_C.FK$ of R_C, and `<table name>` represents parent relation R_P. In SQL3, the optional `<referenced columns>` can designate the candidate

Table 1
The actions to guarantee referential integrity

	R_P			R_C		
	Insertion	Deletion	Updates	Insertion	Deletion	Updates
CASCADE	o	?	?	–	o	–
RESTRICT	o	?	?	–	o	–
SET NULL	o	?	?	–	o	–

o = to always satisfy the referential integrity.
– = to check referential integrity at run-time.
? = to require referential actions at run-time.

key of R_P, but in this paper we assume the referenced columns to be the primary key of the parent relation. The last two statements are used to define the referential actions of the foreign key; what is the effect of the update or delete operation in R_P on the matching tuples in R_C? For instance, when a tuple t_{R_P} is deleted from R_P, the `CASCADE` option specifies that a DBMS should also delete all its child tuples t_{R_C} from R_C. In the case of `SET NULL`, a DBMS should also nullify the foreign key value of all its child tuples in R_C. The `RESTRICT` option declares that, when a tuple t_{R_P} is deleted from R_P or its primary key value is changed, the deletion (or update) operation is not allowed if there exists its child tuple(s) t_{R_C} in R_C. We do not consider the `SET DEFAULT`, `NO ACTION` options and the `PENDANT`, `MATCH` keywords. See [16,33] for detailed explanations of their semantics.

Table 1 shows the scope of the referential integrity semantics to be covered in this paper: that is, the three options for the referential actions of foreign key, `CASCADE`, `RESTRICT`, `SET NULL` for three operations, insert, delete, and update in R_P and R_C, respectively. In Table 1, the element with o always satisfies the referential integrity; for example, insertion to the relation R_P does not violate referential integrity in any case. With the element with −, it should be checked whether the corresponding foreign key satisfies the referential integrity. Finally, the element with ? requires referential actions at run-time.

3. Referential ambiguity and its solutions

3.1. Referential ambiguity

Due to polyinstantiation, as described in the previous section, more than one tuple with the same primary key value can simultaneously exist in a multilevel relation. When it comes to referential integrity, this fact brings about the problem of referential ambiguity in a multilevel relational data model [11]. In Fig. 6, for instance, what *Pathfinder* in R_P does the tuple `(Kennedy, U, Pathfinder, S, S)` in R_C refer to? The first two `Pathfinder` tuples are polyinstantiated to model a *U*-level `Pathfinder`. The last `Pathfinder` is a polyinstantiated tuple that models another `Pathfinder` different from the *U*-level `Pathfinder`.

The foreign key value of the tuple `(Kennedy, U, Pathfinder, S, S)` in Fig. 6 is said to have "referential ambiguity", and the three `Pathfinder` tuple in R_P is said to "cause" the ambiguity. Until now, several approaches have been taken to address this problem of referential ambiguity,

CAPTAIN		SHIP		TC
Kennedy	U	Apollo	U	U
Kennedy	U	Pathfinder	S	S

(a)

SHIP		MISSION		DEST		TC
Apollo	U	Exploration	U	Moon	U	U
Pathfinder	U	Exploration	U	Mars	U	U
Pathfinder	U	Nuclear Test	S	Mars	U	S
Pathfinder	C	Exploration	C	Sun	C	C

(b)

Fig. 6. Referential ambiguity: an example. (a) R_C, relation *CS* and (b) R_P, relation *SMD*.

but we think they are incomplete and less systematic. In the rest of this section, we review those approaches and give the basic idea of our solution.

3.2. Previous solutions

Researchers recognized the problem of referential ambiguity in multilevel relational data models, and proposed some approaches to the problem [21,28,29]. These approaches can be classified as follows.

- Avoidance of ambiguity
  - MLR model [29]. Prevents the occurrence of referential ambiguity in a multilevel relational data model.
  - Sandhu–Jajodia model [28]. Prevents referential ambiguity during the database design phase.
- Resolution of ambiguity
  - SeaView model [21]. Allows referential ambiguity, but provides a resolution policy to choose one tuple.

However, these approaches have several disadvantages. First, the approach of ambiguity avoidance seems to be too restrictive. Second, the approach taken in SeaView model is inflexible, since users may require another resolution policy different from the specific one supported by the model. Finally (and most importantly), all the previous approaches focused on the definition of referential integrity and on the resolution of referential ambiguity, but no work has been done on the dynamic referential actions as in the standard relational data model. For instance, they did not consider the effect of deleting the tuple `(Pathfinder, C, Exploration, C, Sun, C, C)` from the table in Fig. 6(b) on the tuple `(Kennedy, U, Applo, U, U)` in Fig. 6(a).

3.3. MLS-RIS approach

In this section, we propose a systematic approach, MLS-RIS, to the referential ambiguity problem, which removes the limitations in the previous approaches. In particular, MLS-RIS extends the semantics of referential actions in SQL standard, considering the referential ambiguity.

For this purpose, MLS-RIS extends the syntax of foreign key declaration as follows.

```
FOREIGN KEY [(<referencing columns>)]
  [VALUE(default)|ENTITY]
  REFERENCES [PENDANT] <table name> [(<referenced columns>)]
  [MATCH FULL | MATCH PARTIAL]
  [ON UPDATE {CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION}]
  [ON DELETE {CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION}]
```

The additional part is an option `[VALUE|ENTITY]` before keyword `REFERENCES`, which allows users to choose the type of reference of the foreign key; that is, value-based reference or entity-based reference (value-based reference is the default option). In the remainder of this section, we will give an informal description of each reference type.

3.3.1. Value-based references

As described above, the source of referential ambiguity is tuples having the same key value, which were introduced to model either polyinstantiated tuples or attributes.

With value-based references, if more than one tuple matching the foreign key value of a child tuple exist in parent relation, the child tuple is bound semantically to the parent tuple with the highest C_{PK}. For instance, the tuple `(Kennedy, U, Pathfinder, S, S)` in Fig. 6(a) is bound to parent tuple `(Pathfinder, C, Exploration, C, Sun, C, C)` in Fig. 6(b).

However, readers should note that a value-based reference chooses a tuple with highest C_{PK}, rather than highest tuple class *TC*. In Fig. 6(b), the tuple class of the second `Pathfinder` tuple is higher than that of the last `Pathfinder` tuple, but the last `Pathfinder` tuple is bound to the child tuple `(Kennedy, U, Pathfinder, S, S)`.

We choose this semantic because we think it is more natural for a foreign key value to refer to a tuple modeling the highest level entity, rather than a tuple with highest tuple class.

Now let us consider the case that the tuple `(Pathfinder, C, Exploration, C, Mars, C, C)` is deleted from the table in Fig. 6(b). What about the child tuple bound to this tuple? Should the child tuple be also deleted from the parent table? Or should the child tuple be bound to another parent tuple with matching value. The child tuple `(Kennedy, U, Pathfinder, S, S)` is newly bound to the parent tuple `(Pathfinder, U, Nuclear Test, S, Mars, U, S)`.

3.3.2. Entity-based references

In contrast to value-based references where a parent tuple is dynamically bound to a child tuple based on the change in parent relation, the entity-based reference statically determines a parent entity to be bound to a child tuple, when the child tuple is created.

In order to encode the information of binding parent tuple, a foreign key declared as `ENTITY REFERENCE`, in addition to its class attribute C_{FK}, has 'Referenced Attribute Classification' RC_{FK}.

Fig. 7 illustrates entity-based references. The tuple `(Kennedy, U, Pathfinder, S, C, S)` statically refers to *C*-level tuple `(Pathfinder, C, Exploration, C, Sun, C, C)` of parent relation, since the RC_{SHIP} of the child tuple is *C*.

CAPTAIN		SHIP			TC
		(FK	C_{FK}	RC_{FK})	
Kennedy	U	Apollo	U	U	U
Kennedy	U	Pathfinder	S	C	S

Fig. 7. Entity-based references: an example R_C, relation *CS*.

When a parent tuple is deleted, its child tuples, in contrast to value-based references, are cascadedly deleted under entity-based references. For example, if we delete the parent tuple `(Pathfinder, C, Exploration, C, Sun, C, C)` from the relation *SMD*, its child tuple `(Kennedy, U, Pathfinder, S, C, S)` is also deleted from the relation *CS*.

With entity-based references, the referenced attribute classification statically determines the parent entity. However, it should be noted that, among the tuples modeling polyinstantiated attributes of an entity, the binding parent tuple is dynamically determined like value-based references. For instance, if a new tuple `(Pathfinder, C, Exploration, C, Moon, S, S)` is inserted in relation *SMD* of Fig. 6 (in order to model the polyinstantiated attributes of the tuple `(Pathfinder, C, Exploration, C, Sun, C, C)`), the child `(Kennedy, U, Pathfinder, S, C, S)` refers semantically to the new tuple.

3.4. Comments

Now we are to argue that both value-based references and entity-based references can be reasonable approaches to the referential ambiguity, in particular with regard to polyinstantiation.

Let us consider the case of polyinstantiated attributes. Usually, tuples for modeling polyinstantiated attributes of a real-world entity are intended to provide a plausible value with each classification level, thus preventing a low-level user from knowing about the value at a higher level. From this fact, it is reasonable for both value-based and entity-based references to be dynamically bound to the highest level tuples among those tuples for polyinstantiated attributes tuples. Similarly, the deletion of a tuple for polyinstantiated attributes means that the plausible value is not necessary any longer, so it is semantically correct for a child tuple to refer to the next highest level tuple modeling the same entity.

Next, let us turn to polyinstantiated tuples, which are to model more than one real-world entity with the same primary key value. We believe that in a given classification level, a name, by default, is used to designate the highest entity among the ones that are visible to the level. This is the underlying assumption why we adopt value-based references. Thus, if it is necessary to model exceptional cases where foreign key values refer to other tuples rather than the default highest level entity, database designers should declare the foreign key as `ENTITY REFERENCE`.

At this point, let us intuitively explain the differences of value-based and entity-based reference. The polyinstantiated tuples represent several versions of the same primary key values. In other words, they are different versions with the same name. Among these versions, value-based reference dynamically chooses the referenced version, while entity-based reference statically determines the referenced version.

Since the reference type of child tuples is determined at the time of the foreign key declaration, every tuple in a child relation follows the same policy regarding referential ambiguity; that is, the granularity of foreign key reference types is relation, instead of tuple. An alternative to this ap-

proach is to select either policy for each tuple when the tuple is created. However, one disadvantage of this alternative is that users should be conscious about the policy of every individual tuple they manipulate. In this respect, the reference type declaration at the granularity of relation taken by our MLS-RIS approach does not force users to be concerned about the reference type of each child tuple, but it is flexible enough to handle real-world requirements about multilevel security.

4. Value-based MLS-RIS

In this and next section, we will give a formal semantic of MLS-RIS, including definitions of referential integrity, resolution rules for referential ambiguity, and the semantics of extended referential actions for the multilevel relational data model, in value-based references and entity-based references, respectively.

4.1. Referential integrity in value-based reference

The following integrity property defines the referential integrity between two multilevel relations, R_P and R_C, in value-based references.

Referential integrity 2 (*MLS-RIS referential integrity* 1). For every tuple $t \in R_C$, where $t[FK] \neq null$, there should exist a parent tuple q in R_P.

1. $t[FK] = q[PK]$
2. $t[C_{FK}] \geqslant q[C_{PK}]$

The first requirement is same as the standard relational data model. The second requirement says, in agreement with the Bell–LaPadula model, that there exist at least one visible tuple at the classification level $t[C_{FK}]$.

4.2. Resolution rule for referential ambiguity in value-based reference

The following rule determines to which parent tuple the child tuple binds, when there exist more than one key matching tuples in parent relation.

Referential ambiguity resolution rule 1 (*value-based reference*). For each tuple $t \in R_C$ with referential ambiguity, its corresponding parent tuple q_P is determined according to the following rules.

1. For every tuples $q \in R_P$ such that $t[FK] = q[PK]$ and $t[C_{FK}] \geqslant q[TC]$, choose the one with the highest $q[C_{PK}]$.
2. If there exist more than one tuple satisfying the above rule, choose the one with highest $q[TC]$.

The first rule, intuitively, chooses the highest level entity among entities with matching key values which are visible at the classification level $t[C_{FK}]$. In other words, the first rule selects,

CAPTAIN		SHIP		TC
Kennedy	U	Apollo	U	U
Kennedy	U	Pathfinder	S	S

(a)

SHIP		MISSION		DEST		TC
Apollo	U	Exploration	U	Moon	U	U
Pathfinder	U	Exploration	U	Mars	U	U
Pathfinder	U	Nuclear Test	S	Mars	U	S

(b)

Fig. 8. The effects of a parent tuple deletion on referential relationships: (a) R_C and (b) R_P.

among the tuples modeling polyinstantiated entities, the highest one. The second rule chooses, if there exist more than one tuple modeling polyinstantiated attributes for an entity, the tuple with highest tuple class as q_P. The q_P resolved by the above rules is unique since, according to the PI-tuple-class integrity, there can be only one tuple modeling polyinstantiated attributes in a classification level.

The above resolution rule assumes that users should take responsibility for establishing the correct binding between a child tuple and a parent tuple; that is, users should understand the effects of changes in a parent relation such as insertion, updates and deletion, on the referential relationship of child tuples, and should take some actions, if necessary, to manage the referential integrity, For instance, when the tuple `(Pathfinder, C, Exploration, C, Sun, C, C)` is deleted from parent relation R_P in Fig. 6, the child tuple `(Kennedy, U, Pathfinder, S, S)` refers to a new parent tuple `(Pathfinder, U, Nuclear Test, S, Mars, U, S)`, as shown in Fig. 8. Conversely, when the tuple `(Pathfinder, C, Exploration, C, Sun, C, C)` is inserted into R_P in Fig. 8, the tuple `(Kennedy, Pathfinder)` binds to this new tuple.

4.3. Referential actions of MLS-RIS in value-based references

In this section, based on the referential integrity 2 and resolution rule 1, we describe the semantics of the various update operations in child relation and parent relation. The update operations are based on the classification of Table 1 in Section 2.

4.3.1. Insertion (update) in child relation R_C

As shown in Table 1, against every insert and update operations in R_C, whether a corresponding parent tuple exists in R_P should be checked.

MLS-RIS rule 1 (*insertion* (*update*) *of tuple t in child relation* R_C). For every tuple t being inserted (updated) in R_C, where $t[FK] \neq null$, there should exist a parent tuple satisfying the referential integrity 2. If the matching tuple q does not exist, this operation fails.

If there exist more than one parent tuple satisfying the referential integrity 2, the ambiguity is resolved according to the ambiguity resolution rule 1.

As noted in Table 1, the tuple deletion in R_C, like in the standard relational data model, does not violate referential integrity in any case, therefore requires no referential integrity check.

4.3.2. Insertion in parent relation R_P

In SQL standards, tuple insertion in R_P has no effect on referential integrity. But, tuple insertion in R_P in value-based references may change the referential relationship of some child tuples, according to the ambiguity resolution rule 1.

MLS-RIS rule 2 (*insertion of tuple t in* R_P)

- $\exists t' \in R_P$, $t'[PK] = t[PK] \Rightarrow$ apply the resolution rule 1 to t's all child tuples $q \in R_C$.
- In other case, no changes in the referential relationship between R_P and R_C.

4.3.3. Deletion in parent relation R_P

In this section, we will describe the semantics of tuple deletion in value-based references for each foreign key declaration option, `CASCADE`, `RESTRICT`, and `SET NULL`.

MLS-RIS rule 3 (*deletion of tuple t in* R_P (`CASCADE`))

- $\exists t' \in R_P$, $(t'[PK] = t[PK]) \wedge (t'[C_{PK}] \leqslant t[C_{PK}]) \wedge (t'[TC] < t[TC]) \Rightarrow$ apply the resolution rule 1 to t's all child tuples $q \in R_C$.
- In other case (that is, t is the last tuple with the primary key value $t[PK]$), delete all the tuples $q \in R_C$, $q[FK] = t[PK]$.

For example, when the tuple `(Pathfinder, C, Nuclear Test, C, Mars, C, C)` is deleted from R_P in Fig. 8, its child tuple `(Kennedy, U, Pathfinder, S, S)` in R_C, instead of being cascadedly deleted, changes its parent tuple to `(Pathfinder, U, Exploration, U, Mars, U, U)` as shown in Fig. 9.

Now consider another example of tuple deletion under value-based reference. When the tuple `(Apollo, U, Exploration, U, Moon, U, U)` is deleted from R_P in Fig. 9, its child tuple `(Kennedy, U, Apollo, U, U)` is also deleted, since the parent relation does not have any tuple with `Apollo` as its primary key. Note that when the child tuple `(Kennedy, U, Apollo, U, U)` is deleted, another tuple `(Kennedy, U, Pathfinder, S, S)` modeling polyinstantiated attributes of the entity `Kennedy` is also deleted from the child relation. So, the deletion of parent tuple `(Apollo, U, Exploration, U, Moon, U, U)` from Fig. 9 results in the status of Fig. 10.

CAPTAIN		SHIP		TC
Kennedy	U	Apollo	U	U
Kennedy	U	Pathfinder	S	S

(a)

SHIP		MISSION		DEST		TC
Apollo	U	Exploration	U	Moon	U	U
Pathfinder	U	Exploration	U	Mars	U	U

(b)

Fig. 9. Tuple deletion in R_P changes in referential relationship: (a) R_C and (b) R_P.

CAPTAIN	SHIP	TC

(a)

SHIP	MISSION	DEST	TC
Pathfinder U	Exploration U	Mars U	U

(b)

Fig. 10. Cascaded tuple deletions: (a) R_C and (b) R_P.

MLS-RIS rule 4 (*deletion of tuple t in* R_P (`RESTRICT`))

- $\exists t' \in R_P,\ (t'[PK] = t[PK]) \wedge (t'[C_{PK}] \leqslant t[C_{PK}]) \wedge (t'[TC] < t[TC]) \Rightarrow$ apply the resolution rule 1 to t's all child tuples $q \in R_C$.
- In other case, reject the deletion.

MLS-RIS rule 5 (*deletion of tuple t in* R_P (`SET NULL`))

- $\exists t' \in R_P,\ (t'[PK] = t[PK]) \wedge (t'[C_{PK}] \leqslant t[C_{PK}]) \wedge (t'[TC] < t[TC]) \Rightarrow$ apply the resolution rule 1 to t's all child tuples $q \in R_C$.
- In other case, set $q[FK]$ to `null` for t's all child tuple $q \in R_C$.

Please note that under the option of `RESTRICT` or `SET NULL`, the child tuple in R_C (semantically) changes its reference to the parent table; that is, from the deleted tuple t_{R_P} to its polyinstantiated tuple with lower class (if any).

4.3.4. Primary key update in parent relation R_P

Previous works on multilevel relational data models did not deal with updates of the primary key of a multilevel relation. In this section, we will assume that the primary key update of a tuple is allowed only at the classification level where it was created, and this update will also change all the primary key values of tuples modeling polyinstantiated attributes in order to satisfy the polyinstantiation integrity of Section 2. And we denote the old and new value of the primary key of tuple t as $t[PK_{old}]$ and $t[PK_{new}]$, respectively.

MLS-RIS rule 6 (*primary key update of tuple t in* R_P (`CASCADE`))

- $\exists t' \in R_P,\ (t'[PK] = t[PK_{old}]) \wedge (t'[C_{PK}] \leqslant t[C_{PK}]) \wedge (t'[TC] < t[TC]) \Rightarrow$ apply the resolution rule 1 to t's all child tuples $q \in R_C$.
- In other case, set $q[FK]$ to $t[PK_{new}]$ for t's all child tuple $q \in R_C$.

MLS-RIS rule 7 (*primary key update of tuple t in* R_P (`RESTRICT`))

- $\exists t' \in R_P,\ (t'[PK] = t[PK_{old}]) \wedge (t'[C_{PK}] \leqslant t[C_{PK}]) \wedge (t'[TC] < t[TC]) \Rightarrow$ apply the resolution rule 1 to t's all child tuples $q \in R_C$.
- In other case, reject the update.

MLS-RIS rule 8 (*primary key update of tuple t in* R_P (`SET NULL`))

- $\exists t' \in R_P,\ (t'[PK] = t[PK_{old}]) \wedge (t'[C_{PK}] \leqslant t[C_{PK}]) \wedge (t'[TC] < t[TC]) \Rightarrow$ apply the resolution rule 1 to t's all child tuples $q \in R_C$.
- In other case, set $q[FK]$ to `null` for t's all child tuple $q \in R_C$.

5. Entity-based MLS-RIS

Continuing the previous section, this section will give a formal semantic of entity-based MLS-RIS, including the definition of referential integrity, a resolution rule for referential ambiguity, and the semantics of extended referential actions.

As noted in Section 3, a child tuple in entity-based references is statically bound to its parent tuple at the time of its creation, and the binding information is encoded in the referenced attribute classification RC_{FK}. All the following descriptions in this section are based on this simple concept. The semantics of entity-based MLS-RIS is very similar to that of the standard SQL, except for the effects of this concept.

5.1. Referential integrity in entity-based reference

The following integrity property defines the referential integrity between two multilevel relations, R_P and R_C, in entity-based references.

Referential integrity 3 (*MLS-RIS referential integrity* 2). For every tuple $t \in R_C$, where $t[FK] \neq null$, there should exist a parent tuple q in R_P.

1. $t[FK] = q[PK]$,
2. $t[C_{FK}] \geqslant q[C_{PK}]$,
3. $t[RC_{FK}] = q[C_{PK}]$.

The first two conditions are the same as in the referential integrity 2 for value-based reference. The last condition requires that for a child tuple t, there must exist a parent tuple q with the security class of its primary key same as the 'referenced attribute class' of t.

5.2. Resolution rule for referential ambiguity in entity-based references

However, there may exist more than one parent tuple in R_P satisfying the MLS-RIS referential integrity rule 3. In this case, the following rule determines to which parent tuple the child tuple should be bound.

Referential ambiguity resolution rule 2 (*entity-based reference*). For each tuple $t \in R_C$ having referential ambiguity in entity-base reference, its parent tuple is chosen as follows.

1. Among the tuples $q \in R_P$ such that $t[FK] = q[PK]$, $t[C_{FK}] \geqslant q[C_{PK}]$ and $t[RC_{FK}] = q[C_{PK}]$, choose the one with the highest tuple classification.

5.3. Referential actions of MLS-RIS in entity-based references

In this section, based on the referential integrity 3 and the ambiguity resolution rule 2, we describe the semantics of update operations in child and parent relation. The update operations are based on the classification of Table 1 in Section 2.

5.3.1. Updates in child relation R_C

Like value-based reference, for every insertion and update operations in R_C, whether a corresponding parent tuple exists in R_P should be checked.

MLS-RIS rule 9 (*insertion* (*update*) *of tuple t in* R_C). For every tuple t being inserted (updated) in R_C, where $t[FK] \neq null$, there should exist a parent tuple satisfying the referential integrity 3. If the matching tuple q does not exist, this operation fails.

If there exist more than one parent tuple satisfying referential integrity 3, the ambiguity is resolved according to ambiguity resolution rule 2. Same as in value-based reference, the tuple deletion in R_C does not violate referential integrity in any case, therefore requiring no referential integrity checking.

5.3.2. Insertion in parent relation R_P

In the previous section, we showed that tuple insertion in R_P under the semantics of value-based reference, unlike the standard SQL, may change the referential relationship of some child tuples. Similarly, under the semantics of entity-based reference, tuple insertion in R_P may change the referential relationship of some child tuples, according to the referential ambiguity resolution rule 2.

MLS-RIS rule 10 (*insertion of tuple t in* R_P)

- $\exists t' \in R_P$, $(t'[PK] = t[PK]) \wedge (t'[C_{PK}] = t[C_{PK}]) \wedge (t'[TC] \leqslant t[TC])$ (that is, t' is a lower level tuple modeling the same entity with tuple t) $\Rightarrow$ apply the resolution rule 2 to t's all child tuples $q \in R_C$.
- In other case, no changes in the referential relationship between R_P and R_C.

Unlike value-based reference, even when a tuple `(Pathfinder, C, Exploration, C, Sun, C, C)` is inserted in R_P in Fig. 11, the tuple `(Kennedy, U, Pathfinder, S, U, S)` still refers to the U-level `Pathfinder`.

5.3.3. Deletion in parent relation R_P

In this section, we describe the semantics of tuple deletion in entity-based reference for each foreign key declaration option, `CASCADE`, `RESTRICT`, and `SET NULL`.

MLS-RIS rule 11 (*deletion of tuple t in* R_P (`CASCADE`)). All child tuples q satisfying the following conditions must be deleted from R_C.

1. $q[FK] = t[PK]$,
2. $q[RC_{FK}] = t[C_{PK}]$.

CAPTAIN		SHIP			TC
Kennedy	U	Apollo	U	U	U
Kennedy	U	Pathfinder	S	U	S

(a)

SHIP		MISSION		DEST		TC
Apollo	U	Exploration	U	Moon	U	U
Pathfinder	U	Exploration	U	Mars	U	U
Pathfinder	U	Nuclear Test	S	Mars	U	S

(b)

Fig. 11. Tuple insertion in R_P: changes in referential relationship. (a) R_C and (b) R_P.

MLS-RIS rule 12 (*deletion of tuple t in* R_P (`RESTRICT`)) When there exists any tuple q in R_C satisfying the following conditions, this operation fails.

1. $q[FK] = t[PK]$,
2. $q[RC_{FK}] = t[C_{PK}]$.

MLS-RIS rule 13 (*deletion of tuple t in* R_P (`SET NULL`)). The foreign key of all the child tuples q satisfying the following conditions must be set to *null*.

1. $q[FK] = t[PK]$,
2. $q[RC_{FK}] = t[C_{PK}]$.

5.3.4. Primary key updates in parent relation R_P

As in the previous section, we denote the old and new values of the primary key of tuple t as $t[PK_{\text{old}}]$ and $t[PK_{\text{new}}]$, respectively.

MLS-RIS rule 14 (*primary key update of tuple t in* R_P (`CASCADE`)). The foreign key of all the child tuples q satisfying the following conditions must be changed to $t[PK_{\text{new}}]$.

1. $q[FK] = t[PK_{\text{old}}]$,
2. $q[RC_{FK}] = t[C_{PK}]$.

MLS-RIS rule 15 (*primary key update of tuple t in* R_P (`RESTRICT`)). When there exists any tuple q in R_C, satisfying the following conditions, this operation fails.

1. $t'[FK] = t[PK_{\text{old}}]$,
2. $t'[RC_{FK}] = t[RC_{PK}]$.

MLS-RIS rule 16 (*primary key update of tuple t in* R_P (`SET NULL`)). The foreign key of all the child tuples q satisfying the following conditions must be set to *null*.

1. $t'[FK] = t[PK]$,
2. $t'[RC_{FK}] = t[RC_{PK}]$.

6. Related works

There have been a few works on referential integrity in multilevel relational data models, including the SeaView model [9,21], Sandhu–Jajodia model [28] and the MLR model [29]. However, these works were confined to the definition of the properties of referential integrity, without paying any attention to the referential actions at run-time. In this section, we will summarize these referential integrities and compare them with our MLS-RIS model.

The SeaView model originally proposed the following definition of referential integrity for multilevel relations [9].

Referential integrity 4 (*SeaView* 1). Two relations, R_P and R_C, satisfy referential integrity if and only if for each tuple t in R_C where $t[FK] \neq null$, there exists tuple q in R_P such that $q[PK] = t[FK]$ and $q[C_{PK}] \preceq t[C_{FK}]$.

Unfortunately, as pointed out in [11], this definition causes referential ambiguity. Thus, Lunt et al. [21] changed the above referential integrity property to eliminate the possibility of referential ambiguity, as follows.

Referential integrity 5 (*SeaView* 2). Two relations, R_P and R_C, satisfy referential integrity if and only if for each tuple t in R_C where $t[FK] \neq null$, there exists tuple q in R_P such that $q[PK] = t[FK]$ and $q[C_{PK}] = t[C_{FK}]$.

Though this definition removes the problem of referential ambiguity, it incurs another problem; that is, the modeling power of multilevel relations. For example, it is semantically correct for an S-level child tuple to refer to a parent tuple with U-level primary key value, but the above definition do not allow this kind reference.

Based on this recognition, Sandhu and Jajodia suggested another definition of referential integrity for a multilevel relational data model [28], which slightly changes the original referential ambiguity of SeaView model, as follows.

Referential integrity 6 (*Sandhu–Jajodia*). Two relations, R_P and R_C, satisfy referential integrity if and only if for each tuple t in R_C where $t[FK] \neq null$, there exists tuple q in R_P such that $q[PK] = t[FK]$, $q[C_{PK}] \leqslant t[C_{FK}]$ and $q[TC] \leqslant t[C_{FK}]$.

This definition itself, however, does not overcome the limitation of the SeaView referential integrity. For this definition of referential integrity to be valid, polyinstantiated tuples in multilevel relations must not be allowed in the database design phase; when a multilevel relation is created, the user should partition the domain of key attributes and assign each partition to a specific security class. Therefore, the multilevel relational data model assumed in [28] does not allow the relation such as R_C in Fig. 6 where two or more tuples with different security classes have the same key value. We think this is another type of restrictions on modeling power. Furthermore, there still exists referential ambiguity due to polyinstantiated attributes [29].

Note that the definition of referential integrity 6 is same as that of our MLS-RIS. But, the difference is in that we, instead of restricting the multilevel relation schema, provide the referential

ambiguity resolution rules. Our approach can solve both referential ambiguity and modeling power.

Recently, Sandhu et al. suggested the MLR data model [29] which overcomes the limitations of the previous models. The MLR model is, as far as we know, the first work that comprehensively approaches the issues of referential integrity in multilevel relational model, including operational semantics of update operations. In this respect, this model is closest to our MLS-RIS. However, our MLS-RIS model is different from the MLR model in the way to resolve referential ambiguity. Specifically, our MLS-RIS model approaches this problem by introducing value-based and entity-based reference, while the MLR model is based on the concept of *data-based semantics* and *data-borrow integrity*.

In commercial side, Oracle is, as far as we know, the only product supporting multilevel relational model [23,24]. But they do not consider the concept of polyinstantiation and thus do not encounter the referential ambiguity issues. In this respect, they support very limited form of multilevel relational model.

In summary, *MLS-RIS*, compared to the previous works on referential integrity in multilevel relational models, has two unique characteristics. First, to address the problem of referential ambiguity, it proposes two kinds of reference types, value-based and entity-based reference, and for each type of references, defines the referential integrity and introduces a resolution rule. Consequently, MLS-RIS allows users to flexibly take a policy about foreign key declaration, without curtailing the modeling power of multilevel relations. Secondly, the *MLS-RIS* originally specifies the semantics of referential actions of the SQL update operations, including insert, delete, and update, so as to preserve the referential integrity for each type of references.

7. Conclusion

We believe that it is not the right approach either to avoid referential ambiguity [21] or to limit the modeling power of multilevel relational data models [28]. Moreover, in order to complete the work on foreign key in multilevel relational data model, it is essential to define the semantics of referential actions, as in the standard SQL. In this paper, considering these facts, we proposed the *MLS-RIS*, an extended referential integrity semantic for a multilevel secure relational data model. To deal with referential ambiguity resulting from polyinstantiation, *MLS-RIS* distinguishes two types of foreign key references, and for each type, defines referential integrity and proposes a resolution rule. Based on this framework, *MLS-RIS* defines the semantics of referential actions for various data manipulation operations, including insert, delete, and update.

In the future, we will investigate the problem of defining the global semantics of referential actions within *MLS-RIS*. In standard SQL, while the local behavior of a referential action, i.e. the semantics confined to only the parent and child relation, is intuitive and easy to understand, it has been an issue to determine the combined effects of a set of referential actions, i.e. their global semantics [6,7,14,22]. Several characteristics of multilevel relational data models, including polyinstantiation will complicate the situation.

Acknowledgements

We want to thank the two anonymous reviewers for their helpful and constructive comments.

References

[1] R. Agrawal, J. Kiernan, R. Srikant, Y. Xu, Hippocratic databases, in: Proceedings of the International Conference on Very Large Data Bases, 2002, pp. 143–154.
[2] D. Bell, L. LaPadula, Secure computer systems: unified exposition and multics interpretation, Technical Report MTIS AD-A023588, 1975.
[3] E. Bertino, S. Jajodia, P. Samariti, Database security: research and practice, Information Systems 20 (7) (1995) 537–556.
[4] E. Bertino, S. Jajodia, P. Samariti, A flexible authorization mechanism for relational database management systems, ACM Transaction on Office Information Systems 17 (2) (1998) 101–140.
[5] E. Bertino, P. Samariti, S. Jajodia, An extended authorization model for relational databases, IEEE Transaction on Knowledge and Data Engineering 9 (1) (1997) 85–100.
[6] B. Ludascher, W. May, G. Lausen. Referential actions as logical rules, in: Proceedings of the ACM International Conference on PODS, 1997, pp. 217–227.
[7] R. Cochrane, H. Pirahesh, N. Mattos, Integrating triggers and declarative constraints in SQL database systems. in: Proceedings of the International Conference on Very Large Data Bases, 1996, pp. 567–578.
[8] R. David, S. Son, R. Mukkamala, Supporting security requirements in multilevel real-time databases, in: Proceedings of IEEE Symposium on Research in Security and Privacy, 1995.
[9] D.E. Denning, T.F. Lunt, et al., The SeaView security model, in: Proceedings of IEEE Symposium on Research in Security and Privacy, 1988.
[10] R. Fagin, On an authorization mechanism, ACM Transaction on Database Systems 3 (3) (1978) 310–319.
[11] G.E. Gajnak, Some results from the entity-relationship multilevel secure DBMS project, in: Aerospace Computer Security Applications Conference, 1989.
[12] B. George, J. Haritsa, Secure transaction processing in firm real-time database systems, in: Proceedings of the ACM International Conference on Management of Data, 1997 pp. 462–473.
[13] P.G. Griffiths, B. Wade, An authorization mechanism for a relational database system, ACM Transaction on Database Systems 1 (3) (1976) 242–255.
[14] B.M. Horowitz, A run-time execution model for referential integrity maintenance, in: Proceedings of the International Conference on Data Engineering, 1992, pp. 548–556.
[15] J. Melton, A.R. Simon, Understanding The New SQL: A Complete Guide, Morgan Kaufmann, Los Altos, CA, 1993.
[16] J. Melton, A.R. Simon, J. Gray, SQL: 1999—Understanding Relational Language Components, Morgan Kaufmann Publishers, Los Altos, CA, 2001.
[17] S. Jajodia, R. Sandhu, Polyinstantiation integrity in multilevel relations, in: Proceedings of IEEE Symposium on Research in Security and Privacy, 1990, pp. 104–115.
[18] S. Jajodia, R. Sandhu, Update semantics for multilevel relations, in: Proceedings of IEEE Symposium on Research in Security and Privacy, 1990, pp. 103–112.
[19] S. Jajodia, R. Sandhu, A novel decomposition of multilevel relations into single-level relations, in: Proceedings of IEEE Symposium on Research in Security and Privacy, 1991, pp. 300–315.
[20] T.F. Lunt, Polyinstantiation: an inevitable part of a multilevel world, in: Proceedings of the Fourth Workshop on the Foundations of Computer Security, 1991, pp. 239–240.
[21] T.F. Lunt, D.E. Denning, et al., The SeaView security model, IEEE Transaction on Software Engineering 16 (6) (1990) 593–607.

[22] V.M. Markowitz, Safe referential integrity structures in relational databases, in: Proceedings of the International Conference on Very Large Data Bases, 1991, pp. 123–132.
[23] Oracle Corp. Oracle Label Security Administrators' Guide, Release 2(9.2). Oracle Online Manual (Available from <http://otn.oracle.com/docs/products/oracle9i/doc-library/release2/network.920/a96578.pdf>), March 2002.
[24] Oracle Corp. Oracle9i Label Security. Oracle Technical White Paper (also Available from <http://otn.oracle.com/deploy/security/ols/pdf/OLS9iR2.TWP.pdf>), January 2002.
[25] F. Rabitti, E. Bertino, W. Kim, D. Woelk, A model of authorization for next-generation database systems, ACM Transaction on Database Systems 16 (1) (1991) 88–131.
[26] R. Sandhu, Relational database access control, in: Handbook of Information Security Management, Auerbach Publisher, Philadelphia, PA, 1994, pp. 145–160.
[27] R. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, Role-based access control models, IEEE Computer 29 (2) (1996) 38–47.
[28] R. Sandhu and S. Jajodia, Referential integrity in multilevel secure databases, in: Proceedings of 16th NIST-NCSC National Computer Security Conference, 1993, pp. 39–52.
[29] R.S. Sandhu, F. Chen, The multilevel relational (MLR) data model, ACM Transactions on Information and Systems Security 1 (1) (1998) 93–132.
[30] S. Son, B. Thuraisingham, Towards a multilevel secure database management system for real-time applications, in: Proceedings of IEEE Workshop on Real-Time Applications, 1993.
[31] P.D. Stachour, B. Thuraisingham, Design of LDV: a multilevel secure relational database management system, IEEE Transaction on Knowledge and Data Engineering 2 (2) (1990) 190–209.
[32] M. Thuraisingham, Mandatory security in object-oriented database systems, in: Proceedings of International Conference in Object-Oriented Programming: Systems, Languages, and Applications, 1989, pp. 203–210.
[33] C. Turker, M. Gertz, Semantic integrity support in SQL:1999 and commercial (Object-) relational database management systems, The VLDB Journal 10 (4) (2001) 241–269.
[34] W. Eßmayr, F. Kastner, G. Pernul, S. Preishuber, A M. Tjoa, Authorization and access control in IRO-DB, in: Proceedings of the International Conference on Data Engineering, 1996, pp. 40–47.

Sang-Won Lee has been a full time lecturer in Sungkyunkwan University (SKKU), Korea since 2002. He received his BS, MS and Ph.D. degrees in computer engineering from Seoul National University, Korea, in 1991, 1994 and 1999, respectively. Before joining SKKU, he had been a BK21 Contract Professor in Ewha Womans University, Korea, and a technical staff at Oracle, Korea. Dr. Lee is interested in the areas of database security, query optimization, performance tuning, and XML database.

Yong-Han Kim is a technical staff at Oracle, Korea since 1997. He received MS degree in computer engineering from Seoul National University, Korea, in 1997 and BS degree in computer science from Ajou University, Korea, in 1995. Mr. Kim is interested in the areas of business intelligence, data warehouse, and enterprise user security management.

Hyoung-Joo Kim is a professor in the School of Computer Science Engineering at Seoul National University since 1991. He received his BS degree in computer engineering from Seoul National University, Seoul, Korea, in 1982 and his MS and Ph.D. in computer science from University of Texas at Austin in 1985 and 1988, respectively. His current research interests include XML data management, object-relational DBMS, HCI, and computer-aided software engineering.