Enhancing Database Access Control by Facilitating Non-Key Related Cover Stories

ABSTRACT

Mandatory access control policies are applicable in environments that contain large amounts of sensitive information with very strict access and security requirements. Database systems that implement mandatory access control are referred to in the literature as Multi-Level Secure (MLS) systems. In this paper we summarize the security benefits and features of such systems and we present improvements to the underlying logic of MLS databases. We describe the necessary model changes and the consequent relational algebra modifications, which are required in order to facilitate the proposed improvements. We also discuss the implementation and performance of a system based on the described concepts.

1. INTRODUCTION

Database access control ensures that, once a user enters a database environment, all accesses to database objects occur only according to the models and rules fixed by protection policies. In general, there are two approaches to enforcing access control policies: discretionary and mandatory.

Discretionary access control is based on granting and revoking privileges for the usage of data objects. The privileges are granted (or revoked) to every user separately. Discretionary access control policies allow access rights to be propagated from one user to another. Discretionary access control is a standard feature of all contemporary RDBMS (Relational Database Management Systems) software tools, and it is used as a primary access control measure for most commercial applications.

Mandatory access control is applicable in systems containing large amounts of extremely sensitive information, with very strict access and security requirements (e.g. military, governmental agencies, airlines, etc.), in environments where the users are grouped into clearances and data are grouped by their classifications. In a mandatory access control policy, access to data is determined solely by the user’s and data object’s membership in security classes. The systems that implement mandatory access control policies are known as multilevel secure (MLS) systems.

In MLS relational databases multiple records on various security levels can depict the same real-world entity. For such records non-key attributes can have different values at different security levels. Providing information to users at lower security levels that is different from the information stored at higher security levels is called a cover story. Cover stories provide a mechanism to protect information that should only be known to users at higher security levels from users at lower levels. Until recently, every MLS model required the key attributes to have the same value at all security levels. This requirement, excluded the possibility of users at different security levels from seeing different values for the key attributes, even though there are applications for which it may be necessary to provide a cover story for the key attributes (in order to mask the value of an identifier of a depicted object to
users at lower security levels.) In our recent paper [12] we identified this shortcoming as “the cover story dependence on the value of a user-defined key” (referred to as “the key loophole”) and we proposed an approach that provides a practical solution to this problem. In this paper we present the model changes and the consequent relational algebra modifications, which are required in order to facilitate the proposed improvements. We also describe the implementation of the system based on the proposed solution and we demonstrate the performance feasibility of such solution.

2. MLS

MLS models [3] [7] [10] [11] [14] [17] [18] [19] are based on the classification of the system elements, where classifications are expressed by security levels. Data objects have security levels and users have clearance levels. The security levels of objects are also known as security labels. A security label can contain one security level or a list of levels [6]. As an example, we can have three possible classifications S-Secret, C-Classified, and U-Unclassified, where S is a higher classification than C and U, and C is a higher classification than U. A security (or clearance) level l1 dominates another level l2 (stated as l1 ≥ l2), if l1 is higher than or on the same level as l2 in the partial (or total) order of security levels. For example, S ≥ C ≥ U. According to the Bell-LaPadula [1] simple property, a subject (user) can read a certain object (data) only if the subject’s clearance level dominates the object’s security level. In other words, a subject cannot read an object at a higher or incomparable security level than the subject. A second restriction on multilevel secure databases [1] is the *-property, which states that all writes take place at the subject’s security level or higher.

2.1 Models

Many MLS relational database models have been proposed and early work in MLS relational databases focused on the semantics and the relational algebra for such models. The SeaView model [3] [4] was the first formal MLS secure relational database designed to provide mandatory security protection. The Sea View model extended the concept of a database relation to include the security labels. A relation that is extended with security classifications is called a multilevel relation. The Jajodia-Sandhu model [8] [10] was derived from the SeaView model. It was shown in [8] that the SeaView model can result in the proliferation of tuples on updates and the Jajodia-Sandhu model addresses this shortcoming. The Smith-Winslett model [18] [19] was the first model to extensively address the semantics of an MLS database. The MLR model [14] [15] is substantially based on the Sandhu-Jajodia model, and also integrates the belief-based semantics of the Smith-Winslett model.

It was shown that all of the aforementioned models can present users with some information that is difficult to interpret [13]. Consequently, we have developed the Belief-Consistent MLS (BCMLS) model [11], which
addressed those concerns by including the semantics for an unambiguous interpretation of all data presented to the users.

To illustrate the main features of an MLS database we will use the following example scenario, where the underlying MLS database is based on the BCMLS model.

Example 1.

National Airline keeps track of its passengers in a relational MLS database (Figure 1). The airline classifies its database users into three clearance categories: U, C, and S, which determine the sensitivity of information they are allowed to see. Every passenger of every flight must be accounted for, on every clearance level. However, the correct passenger’s type and ticket pricing information may have to be hidden from some security levels. All the information about passengers Mike Smith and Bob Johnson is available for all three clearance levels. However, the information about passenger Sue McCoy is more sensitive. The subjects on the S level correctly see both her ticket pricing information and the fact that she is an Air Marshal. The subjects on the C level see her correct ticket pricing information, but the fact that she is an Air Marshal is masked by a cover story. The subjects on the U level are given a cover story for both her passenger type and ticket pricing information.

Within a table, each attribute is accompanied by the security label that can contain more than one letter. The first letter in the label indicates the security level on which the value of the attribute was created. Such a level is called the primary level of that attribute. Labeled information is always believed to be true by the users whose clearance is equivalent to the security level indicated by the primary level of the label. The letters that follow the first letter of the label indicate the security levels where users from those levels do have a belief about labeled information, but the labeled information was not created at that level. Such levels are called secondary levels. Each security level in the label must dominate the level to its left. Letters that are not preceded by the “-” symbol indicate the secondary levels where the information is believed to be true. The letters following the “-” symbol indicate the secondary levels where the information is believed to be false. In addition to labeling each attribute with a security label, a tuple as a whole is also labeled by a security label, which is depicted by the TC column. The tuple is visible on a certain level only if the TC label contains the label for that level. Also, not every part of the label is visible to every user. Only the parts of the label that depict the user’s levels or levels below are visible.

<table>
<thead>
<tr>
<th>Passenger Name</th>
<th>Seat Assg.</th>
<th>Type</th>
<th>Ticket Pricing</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mike Smith</td>
<td>First</td>
<td>Regular</td>
<td>Paid Ticket</td>
<td>UCS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Passenger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob Johnson</td>
<td>Coach</td>
<td>Crew in</td>
<td></td>
<td>UCS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transfer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sue McCoy</td>
<td>Coach</td>
<td>Regular</td>
<td></td>
<td>U-CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Passenger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sue McCoy</td>
<td>Coach</td>
<td>Regular</td>
<td></td>
<td>C-S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Passenger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sue McCoy</td>
<td>Coach</td>
<td>Air</td>
<td>No Charge</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Marshal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1

Figure 2 shows how the table shown in Figure 1 would be seen by the users from three different levels.
As was mentioned in Section 1, in MLS relations multiple tuples can exist at different security levels representing contradictory information about the same entity. Assume a user is at security level \( c \). If a lower level tuple with a \( TC < c \), represents the same entity as some other higher level tuple, where \( TC = c \), the lower level tuple is interpreted by a higher level user as a false tuple that represents a cover story \([5]\) \([16]\) for the entity represented by the higher level tuple. A belief held by \( c \)-level users that a lower level tuple is a cover story tuple is derived from the fact that there exists a \( c \)-level tuple that represents the same entity as the cover story tuple. Every user on the higher-level \( c \) has the following belief about the cover story lower-level tuple: “Some attribute values of this lower level tuple incorrectly represent a real-world entity.” In Figure 2 (S-view), S level users see the second and third tuples as a cover story of the fourth tuple. Cover stories have been used in MLS models for non-key attributes only. None of the existing models has considered a cover story involving a key attribute. In Section 3 we will show how the usage of cover stories can be expanded to involve key attributes.

### 2.2 Covert Channels and Polyoinstantiation

The two aforementioned Bell-LaPadula properties prevent the direct flow of information from objects and/or subjects at a higher security clearance level to subjects at a lower level, and are the basis for all MLS models. However, a system may not be secure even if it always enforces the two Bell-LaPadula properties. There may
exist a covert channel, which allows for an indirect flow of information from a higher level user to a lower level
user. For example, suppose a lower level user wishes to insert a tuple that already exists in the database at a
higher level of security (e.g. inserting record ‘James Bond’ into the Spy table on the U level, when there is
already a record ‘James Bond’ on the S level in the same table). If this insert is rejected by the system, the lower
level user will know that there already exists the same tuple at a higher level (i.e. that there is a higher security-
level spy James Bond that U level users are unaware of). This indirect flow of information from higher to lower
security levels can occur in other ways. For instance, the concurrent execution of transactions results in
contention for data objects. If the results from a lower security level transaction are delayed when there is a
higher-level security transaction, then the lower security level user can determine there are transactions at higher
levels, and may even be able to infer information from the length of the delay.

In the relational model, two tuples must not exist in a relation with the same values for the primary key
attribute, but requiring this constraint to hold in multilevel relations may result in a covert channel. In order to
avoid covert channels in MLS data models, subjects with different classifications are allowed to operate on the
same relations, through the use of polyinstantiation [9] [13]. The term polyinstantiation refers to the
simultaneous existence of multiple tuples with the same primary key, where such tuples are distinguished by their
classifications [13]. Because of that, the user specified primary key in the MLS environment is called the
apparent key. Polyinstantiation is illustrated in Figure 1 where there are 3 tuples representing Sue McCoy.
There are two kinds of polyinstantiation that can occur within an MLS relation: attribute polyinstantiation and
entity polyinstantiation.

Attribute polyinstantiation occurs when two tuples representing the same entity have different values
associated with the same attribute (e.g. two different ticket pricing entries for Sue McCoy). Cover stories utilize
the concept of attribute polyinstantiation. They are often used to deceive lower level users about the nature of the
sensitive information (as illustrated by the example in Figure 1). Entity polyinstantiation occurs when two tuples
have the same primary key and different classifications associated with the primary keys. In the previously
presented James Bond scenario, the lower-level user would have to be allowed to insert a James Bond tuple into
the table (in order to prevent a covert channel). That would lead to entity polyinstantiation, due to the existence
of two different James Bond’s (e.g. James Bond U and James Bond S) in the relation. Since a U-level does not
know about the existence of Bond S, it is assumed that Bond U is a different entity, unless an S-level user
indicates differently.

3. THE KEY-LOOPHOLE AND NON-KEY RELATED COVER STORIES

Within MLS models the link between a tuple and its corresponding lower-level cover story tuple is the matching
value of their entity identifier. For example, in the BCMLS model the entity identifier is composed of the user-
defined key and the primary level (pl) of its classification attribute: K + pl(KC). Figure 1 shows that the third, fourth, and fifth tuples share the same entity identifier ‘Sue McCoy, U’ (note that pl(UCS) = U). In this case, the entity identifier value is used to identify, on the S level, the third and fourth tuples as cover stories for the fifth tuple. The same entity identifier identifies on the C level the third tuple as the cover story of the fourth tuple.

Existing MLS models use a value-based approach for defining the entity identifier (e.g. stored values ‘Sue McCoy, U’ were used to indicate that the third, fourth, and fifth tuple refer to the same entity), which limits the scope of their usage. In order to portray this limitation we will slightly alter Example 1, introduced earlier in this paper.

Example 1 (Altered)

As in the first version of this example, all the information about passengers Mike Smith and Bob Johnson is available for all three security/clearance levels. However, the information about passenger Sue McCoy is more sensitive. The subjects on the S level are allowed to see correctly her seat assignment, passenger type, ticket pricing, and name. The subjects on the C level are allowed to see her seat assignment and ticket pricing, but her passenger type and her name should be masked by a cover story passenger type and a cover story name. The subjects on the U level can see her correct seat assignment, and should be given a cover story for her passenger type, ticket pricing and her name.

No existing MLS model is capable of properly handling this scenario. The reason for it is the cover story dependence on the value of a user-defined key. Figure 3 illustrates the situation. If we simply tried to change the name of the passenger Sue McCoy to a different name (Jane Clark) on the C and U levels, we will be faced with the following problem. The user on the S level would know that there is no passenger named Sue McCoy, but at the same time the S user would have no way of knowing that Jane Clark is a cover story for Sue McCoy. Instead, the S level user would treat all records relating to Jane Clark as so-called mirage tuples, which represent a non-existing entity (when every attribute of a tuple is labeled as false on a certain level, a user from that level considers that tuple to be a mirage tuple [11].) This can cause problems in situations when an S level user has to communicate with lower level users. For example the S level user would be unaware that C level users are aware of the passenger Sue McCoy (they simply know her under a different name) and her ticket pricing.

<table>
<thead>
<tr>
<th>Passenger Name</th>
<th>Seat Assg.</th>
<th>Type</th>
<th>Ticket Pricing</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mike Smith</td>
<td>UCS</td>
<td>First</td>
<td>Regular Passenger</td>
<td>Paid Ticket UCS</td>
</tr>
<tr>
<td>Bob Johnson</td>
<td>UCS</td>
<td>Coach</td>
<td>Crew in Transfer</td>
<td>No Charge UCS</td>
</tr>
<tr>
<td>Jane Clark</td>
<td>UC-S</td>
<td>Coach</td>
<td>Regular Passenger</td>
<td>Paid Ticket U-CS</td>
</tr>
<tr>
<td>Jane Clark</td>
<td>UC-S</td>
<td>Coach</td>
<td>Regular Passenger</td>
<td>No Charge CS</td>
</tr>
<tr>
<td>Sue McCoy</td>
<td>S</td>
<td>Coach</td>
<td>Air Marshal</td>
<td>No Charge S</td>
</tr>
</tbody>
</table>

Figure 3
We call the inability of the existing MLS models to connect a tuple that represents a certain entity on a particular security level to a lower-level cover story tuple that has different key attribute value but represents the same entity: the key loophole.

In [12] we identified the key loophole problem and we introduced a change in the way the entity identifier is defined. We proposed a system defined entity identifier (SEID), whose value would remain hidden to all users on all security levels and would be used only internally by the MLS DBMS. Here we illustrate how this new concept would be used to properly handle the situation depicted in the Example 1. This is shown in Figure 4.

```
NATIONAL AIRLINES FLIGHT 1234 TABLE

<table>
<thead>
<tr>
<th>SEID</th>
<th>Passenger Name</th>
<th>Seat Assg.</th>
<th>Type</th>
<th>Ticket Pricing</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Mike Smith</td>
<td>First</td>
<td>Regular Passenger</td>
<td>Paid Ticket</td>
<td>UCS</td>
</tr>
<tr>
<td>2222</td>
<td>Bob Johnson</td>
<td>Coach</td>
<td>Crew in Transfer</td>
<td>No Charge</td>
<td>UCS</td>
</tr>
<tr>
<td>3333</td>
<td>Jane Clark</td>
<td>Coach</td>
<td>Regular Passenger</td>
<td>Paid Ticket</td>
<td>U-CS</td>
</tr>
<tr>
<td>3333</td>
<td>Jane Clark</td>
<td>Coach</td>
<td>Regular Passenger</td>
<td>No Charge</td>
<td>C-S</td>
</tr>
<tr>
<td>3333</td>
<td>Sue McCoy</td>
<td>Coach</td>
<td>Air Marshal</td>
<td>No Charge</td>
<td>S</td>
</tr>
</tbody>
</table>
```

Figure 4

The SEID column contains the new system defined entity identifier. If an S level user requests all information about Diva Megastar, the fourth tuple along with the cover story second and third tuples would be displayed. The S level user would now be aware of the fact Jane Clark’s records are cover–stories about Sue McCoy’s given to the lower level users. An interface to an MLS application can now bundle each tuple with its related cover stories, even if the cover stories are not related via a key value (Figure 5). We call the cover stories that are not related through a matching value of key attributes – Non Key-related Cover Stories (NKCS).

```
NATIONAL AIRLINES FLIGHT 1234 TABLE

<table>
<thead>
<tr>
<th>Passenger Name</th>
<th>Seat Assg.</th>
<th>Type</th>
<th>Ticket Pricing</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mike Smith</td>
<td>First</td>
<td>Regular Passenger</td>
<td>Paid Ticket</td>
<td>UCS</td>
</tr>
<tr>
<td>Bob Johnson</td>
<td>Coach</td>
<td>Crew in Transfer</td>
<td>No Charge</td>
<td>UCS</td>
</tr>
<tr>
<td>Jane Clark</td>
<td>Coach</td>
<td>Regular Passenger</td>
<td>Paid Ticket</td>
<td>U-CS</td>
</tr>
<tr>
<td>Jane Clark</td>
<td>Coach</td>
<td>Regular Passenger</td>
<td>No Charge</td>
<td>C-S</td>
</tr>
<tr>
<td>Sue McCoy</td>
<td>Coach</td>
<td>Air Marshal</td>
<td>No Charge</td>
<td>S</td>
</tr>
</tbody>
</table>

Next Record … … … …
Its Cover Stories … … … …
```

Figure 5

4. MODEL CHANGES

As we will show in this paper, the proposed solution to the key loophole problem by using an SEID requires considerable technical changes within MLS models and associated relational algebras. At the same time, the
improvements gained in the robustness of the extended model have far reaching implications for its practical applicability, and therefore, warrant the effort required to make the changes.

4.1. Properties

We first describe how the introduction of the new entity identifier approach changes the properties of the BCMLS model. We are using the BCMLS model as a representative of MLS models, and the changes described here would not be significantly different for other contemporary MLS models. We presented the original properties of the BCMLS model in [11]. We give an abbreviated version of these properties in Figure 6.

Key Properties of the Belief-Consistent MLS Data Model

LABELS: In the BCMLS Model, bcl[L, H] indicates, for the set of totally-ordered security levels ranging form the lowest level security L to the highest level security H, a set of possible security labels (belief-consistent labels) available. For example, in the environment with two security levels U and C where C dominates U (U ≤ C), the set of possible security labels is bcl[U, C] = {U, UC, U-C, C}. Function pl(c) where c ∈ bcl[L,H], extracts a primary level from the belief-consistent label c. For example, pl(UC)=U, pl(U)=U, and pl(C) = C.

RELATION SCHEME: A multilevel relation scheme is denoted by R(K, KC, A1, C1,...,Am, Cm, TC) where K is the data primary key attribute(s), KC is the classification attribute of K, each Ai is a non-key data attribute over domain Di, each Ci is the classification attribute for corresponding Ai, and TC is the tuple classification attribute. The domain of KC, TC, and Cj is the set of possible belief-consistent labels bcl[L,H].

RELATION INSTANCE: A relation instance, denoted by r(K, KC, A1, C1,...,Am, Cm, TC), is a set of distinct tuples of the form (k, a1, a2,...,am, c1,...,cm, tc) where each k, ai ∈ Di, and kc, ci ∈ bcl[L,H], and tc is a set of labels defined as follows: for every security level l in the range [L,H]

Figure 6.
In the remainder of section we show how the original model properties are changed in order to accommodate the system defined entity identifier.

**Relation Schema and Relation Instance**
The relation schema $R(K, KC, A_1, C_1,...,A_n, C_n, TC)$, shown in Figure 6, is now expanded to account for the new entity identifier and it is denoted by $R(SEID, K, KC, A_1, C_1,...,A_n, C_n, TC)$. Consequently, a relation instance is now denoted by $r(SEID, K, KC, A_1, C_1,...,A_n, C_n, TC)$, and it represents a set of distinct tuples of the form $(seid, k, kc, a_1, c_1,...,a_n, c_n, tc)$.

**Entity Integrity**
The entity integrity property (shown in Figure 6) ensures that no key attribute can contain null values, no key classification can contain null values, and primary level of a classification of the non-key attribute (denoted as $pl(t[C_i])$ ) must dominate the primary level of the classification of the key attribute (denoted as $pl(t[KC])$ ). This property will remain the same, with the addition of the following condition:

4. $t[SEID] \neq \text{null}$

This ensures that every tuple has a system entity identifier assigned to it.

**Base Tuple Integrity Property**
The first two conditions of the original base tuple integrity property (shown in Figure 6) establish the key attribute value and its classification as the entity identifier. The third and fourth conditions ensure that for every entity depicted in the relation there will be a base tuple $t_b$ with no null values for any attribute and equal primary level of the classification for each attribute. In the new model, the first two conditions will be replaced with the following single condition:

1. $t[SEID] = t_b[SEID]$

which ensures that all the tuples that are referring to the same entity share the same system entity identifier.

The change in the definition of the entity identifier will not cause changes in the definitions of the Polyinstantiation Integrity Property (which ensures that only one tuple with a particular value of the user-defined
key attribute can originate on one security level), the Referential Integrity Property (which ensures that a foreign key on each security level references an existing value in another table that is true on the same security level), and the Foreign Key Property (which ensures that the security classifications of each part of the composite foreign key are the same.)

4.2 Relational Algebra

In addition to the above described model property changes, the new definition of the entity identifier also requires changes in the relational algebra. We introduce the new concept of query result entity equivalence as a basis for the relational algebra of the SEID based model. This concept ensures that, for each record that satisfies the condition of the query, the result includes all other records that refer to the same entity if they satisfy all parts of the query condition that do not involve the key value.

This concept is necessary in order to recognize and include Non-Key Related Cover Story (NKCS) in query results. For example, consider the select operation:

$$\sigma_{\Phi}(R)$$

where $\sigma$ is the select operator, $\Phi$ is the select condition and $R$ is the MLS relation on which the select operation is being applied. The select condition $\Phi$ has the following form:

$$\Phi = clause (boolean\_op\ clause)^*$$

where * means zero or more, boolean\_op is AND, OR, and NOT, and

$$clause := E_i \ op \ E_j \mid E_i \ op \ a \mid E_i \ L \ b \ (boolean\_op\ L\ b)^* \mid TC \ L\ b \ (boolean\_op\ L\ b)^*.$$  

where $E_i$ represents a value attribute (key or non-key) from $R$, $a$ is a constant, and $op$ is one of the comparison operators ($<, =, >, \leq, \geq, \text{ or } \neq$). $TC$ is the tuple classification label. $L$ is a single label representing a security level (e.g. U, C, or S), and $b$ is a belief held by that level (e.g. true or false).

As an example, suppose an S-level user issues the following select operation of the relation shown in Figure 1, to choose all tuples referring to the passenger Sue McCoy that show her correct ticket pricing:

$$\sigma_{\text{Passenger Name} = 'Sue McCoy' \ \text{AND} \ \text{Ticket Pricing} S \ \text{true} \ \text{(NATIONAL AIRLINES FLIGHT 1234 TABLE)}}.$$

The result is:

<table>
<thead>
<tr>
<th>Sue McCoy</th>
<th>UCS</th>
<th>Coach</th>
<th>UCS</th>
<th>Regular Passenger</th>
<th>UC-S</th>
<th>No Charge</th>
<th>CS</th>
<th>C-S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue McCoy</td>
<td>UCS</td>
<td>Coach</td>
<td>UCS</td>
<td>Air Marshal</td>
<td>S</td>
<td>No Charge</td>
<td>CS</td>
<td>S</td>
</tr>
</tbody>
</table>

10
However, the same query issued on the table shown in Figure 4 (where cover stories are not related through the value of the key) would result in:

| Sue McCoy  | S  | Coach | UCS | Air Marshal | S  | No Charge | CS | S |

thus depriving S users of the knowledge that C level users, who are also aware of the correct pricing for the sought passenger, know this passenger under a different name (see table in Figure 5).

In order to accommodate NKCS (and therefore eliminate the information availability problem illustrated by this example) we redefine the select operation as follows:

$$\sigma' \Phi (R)$$

where $\Phi$ is the select condition that has the form $\Phi = \text{clause} \ (\text{boolean\_op \ clause})^*$, $R$ is the MLS relation on which the select operation is being applied, and $\sigma'$ is the newly defined select operator where

if

$$\text{clause} = K \ \text{op} \ E_i \ | \ K \ \text{op} \ a$$

where $K$ represents the key value attribute from $R$, and $E_i$ represents a value attribute (key or non-key) from $R$

then

$$\sigma'\text{clause}(R) = \sigma\text{SEID in } (\pi\text{SEID}(\sigma\text{clause}(R))) \ (R)$$

else (for all other clauses)

$$\sigma'\text{clause}(R) = \sigma\text{ clause} \ (R)$$

where $\pi$ and $\sigma$ are the regular BCMLS relational algebra project and select operations and in is the set membership boolean operator.

This definition ensures entity equivalence of the query result. For example, the query Q1 executed on the table shown in Figure 4 using the new select statement will select the records:

<table>
<thead>
<tr>
<th>Jane Clark</th>
<th>UC-S</th>
<th>Coach</th>
<th>UCS</th>
<th>Regular Passenger</th>
<th>UC-S</th>
<th>No Charge</th>
<th>CS</th>
<th>C-S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue McCoy</td>
<td>S</td>
<td>Coach</td>
<td>UCS</td>
<td>Air Marshal</td>
<td>S</td>
<td>No Charge</td>
<td>CS</td>
<td>S</td>
</tr>
</tbody>
</table>

due to the fact that, even though they have different key attribute values, both records refer to the same entity (i.e. they are entity equivalent). An interface, such as the one illustrated by Figure 5, would ensure that the user clearly recognizes entity equivalent records.
5. Implementation and Performance

As a proof of concept, we implemented a prototype of our proposed model using Oracle 9i RDBMS (running on an Intel Pentium III Zeon dual processor, 1.8 Ghz Mhz, 1 GB RAM, Windows 2000 server). Using this prototype, we investigated how the SEID extension of the model affects performance.

In order to run queries efficiently in the new extended system, we had to develop a feasible way of implementing the SEID concept. The naïve approach would be to use system generated SEID values and implement the query language statements as a direct translation of the new relational algebra. However, this approach would result in a markedly slower system as compared to the existing MLS systems, and the benefits of the new approach would be offset by inferior performance. For example, consider the simple ‘select’ queries with a selection condition involving a key value, that are based on the \( \sigma^\Phi(R) \) operation (as defined in Section 4). These queries would correctly find more records than regular MLS queries (i.e. NKCS would be included in the result.) When implemented with the naïve approach, however, their running times would be orders of magnitude slower than those of comparable regular MLS queries (which would, albeit have less complete results). The main culprit for the slowdown is not the additional results, but the combination of the ‘in’ operator and the nested query used in the definition of the selection operation. This combination requires a series of comparison operations for each record in the table, which adds extra amount of time to each query, proportional to the size of the table.

In order to reduce the performance cost, we implemented the result-equivalent versions of relational algebra operations, which do not use nested queries in order to detect NKCS. Our strategy is consistent with the implementation of an object identifier (OID) in object-oriented database systems, in which performance is improved by generating OIDs in a manner to speed object lookup [2].

Our solution is to assign to the SEID, for each highest-level record that depicts a particular entity, a value that is the result of applying a numeric (encoding) function to the value of its primary key. Consequently, within a selection condition, to depict:

\[ K \text{ op } X \]

instead of using:

\[ \text{SEID in (nested query)} \]

we use
SEID op encode(X)

With this approach we connect all NKCS and include them in the result without degradation in performance. Note that, for each X the result of encode(X) will be computed only once regardless of the number of SEID involved in the comparison. Thus, the effect on the overall running time is negligible.

Using our prototype, we have conducted a series of experiments and compared the performance of a regular MLS system and the new extended system on MLS tables, ranging in size from one hundred thousand to one million records, by varying the share of NKCS (as a percentage of all cover stories). The results indicated no significant difference in performance between a regular and extended system.

6. Conclusion

The key-loophole presents a major inefficiency of existing MLS models, which restricts their use in practical applications. The solution to this problem is to develop a new MLS model that uses system-defined entity identifiers. In order to enable an implementation of the new model based on the concept of system-defined entity identifiers, we made the necessary changes to the basic MLS properties and we developed the new relational algebra. We also implemented the proposed model in a prototype application and investigated the performance issues. We found that implementing the new approach can be accomplished without creating a performance overhead.

REFERENCES


