9-1995

An Algebra for Belief Persistence in Multilevel Security Databases

Tsz Shing Cheng
Iowa State University

Shashi K. Gadia
Iowa State University, gadia@iastate.edu

Follow this and additional works at: http://lib.dr.iastate.edu/cs_techreports

Part of the Information Security Commons, Systems Architecture Commons, and the Theory and Algorithms Commons

Recommended Citation
http://lib.dr.iastate.edu/cs_techreports/38

This Article is brought to you for free and open access by the Computer Science at Iowa State University Digital Repository. It has been accepted for inclusion in Computer Science Technical Reports by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
An Algebra for Belief Persistence in Multilevel Security Databases

Abstract
In a multilevel security environment, the security levels form a hierarchy which is generally assumed to be a lattice. A user can see not only its own information, but also information belonging to lower users. In a multilevel security database, different users have different beliefs (versions of information) about the same real world object. In this paper we present a relational model SecDB for multilevel security data. We also present an SQL-like language SecSQL for querying security information. For a given level, a tuple consists of all the differing beliefs about the same real world object. Therefore, the model provides a built-in coherence to different beliefs of the same real world object. For an operator to be well defined, its application should preserve beliefs and coherence. This persistence of belief and coherence is achieved through the concept of an anchor borrowed from an earlier work. On one hand (in addition to the usual database queries) SecSQL yields itself naturally to formulation of security related queries, yet on the other hand the algebraic operators yield natural identities which hold a good promise of algebraic optimization.

Disciplines
Information Security | Systems Architecture | Theory and Algorithms
An algebra for belief persistence in multilevel security databases

Tsz Shing Cheng and Shashi K. Gadia
Department of Computer Science
Iowa State University
Ames, IA 50011
{tcheng,gadia}@cs.iastate.edu; (515)294-4377
September 1995

Abstract. In a multilevel security environment, the security levels form a hierarchy which is generally assumed to be a lattice. A user can see not only its own information, but also information belonging to lower users. In a multilevel security database, different users have different beliefs (versions of information) about the same real world object. In this paper we present a relational model SecDB for multilevel security data. We also present an SQL-like language SecSQL for querying security information. For a given level, a tuple consists of all the differing beliefs about the same real world object. Therefore, the model provides a built-in coherence to different beliefs of the same real world object. For an operator to be well defined, its application should preserve beliefs and coherence. This persistence of belief and coherence is achieved through the concept of an anchor borrowed from an earlier work. On one hand (in addition to the usual database queries) SecSQL yields itself naturally to formulation of security related queries, yet on the other hand the algebraic operators yield natural identities which hold a good promise of algebraic optimization.

Index terms. Databases, security, database security, multilevel security, relational databases, beliefs, belief data, dimensional databases.
1. Introduction

In a multilevel security environment, the user levels form a hierarchy which is generally assumed to be a lattice. Throughout this paper we assume that we are given an arbitrary finite lattice $\Lambda$ of users. We use the terms user, security level, security class and user level interchangeably. For the sake of simplicity one may assume that there is only one user at a given level. In addition in multilevel security literature the term subject is also used for a user.

As our running example we consider the lattice $\{\alpha\beta, \alpha, \beta, \lambda\}$, as shown in Figure 1, together with the partial order $\leq$. It may help the reader to think of $\alpha\beta, \alpha, \beta, \lambda$ as $\{\alpha, \beta\}, \{\alpha\}, \{\beta\}, \emptyset$, respectively, and $\leq$ as $\subseteq$.

![Figure 1. A lattice $\Lambda$ of security levels](image)

Suppose $u_1 \leq u_2$, then we say $u_2$ dominates $u_1$, $u_1$ is below $u_2$ or $u_1$ is dominated by $u_2$; in addition in the context of $u_1$ we call $u_2$ a higher user, and similarly, in the context of $u_2$, we call $u_1$ a lower user. In the lattice of Figure 1, the user $\alpha\beta$ dominates all users $\alpha\beta, \alpha, \beta$ and $\lambda$. For a user $u$, the set of all users dominated by $u$ is denoted as $u^-$. E.g., $\alpha^- = \{\alpha, \lambda\}$.

An object space is a set of objects in the real world. In a multilevel security database, a user forms a belief about the object space: which objects exist in the real world, their identities and property values. In other words a user has a subjective view of the object space and this subjective view is called object space of the user. Information available to a user consists of the following:

- An object space which is a belief about which objects exist in the real world. Every object in an object space is identified by a key value, e.g. NAME and other property values. Each user has its own object space, which is independent of the object spaces of other users.
- All information available to lower users.
- Knowledge of which object in user’s object space is known to a lower user, possibly with different identity (key) and property values.

As an example suppose object space of $\alpha\beta$ is $\{John, Mary\}$, and object space of $\alpha$ is $\{Inga, John, Tom\}$, as shown in Figure 2. As stated above, the reader should keep in mind that the two object spaces are mutually independent. Clearly, $\alpha\beta$ believes that there are only two objects, whereas $\alpha$ believes there are three objects in the real world. In addition $\alpha\beta$ believes that John is known as Tom to $\alpha$, Mary is unknown to $\alpha$, and the objects known as Inga and John to $\alpha$ do not exist in the real world. The user $\alpha$ is not aware of the user $\alpha\beta$ or its object space.

To maintain confidentiality of information, the upper user should not have write access to a lower user, and a lower user should not have read access to the upper user. In a model for secure databases it is important that such access can not be created covertly by users through the facilities available in the model. In other words the model should avoid convert channels. For exam-
ple, consider a model where a user requesting a tuple insertion gets a system rejection because a tuple belonging to a higher user with the same key already exists in the relation. If this is permitted, a user can misuse it to send one bit of information to a user with lower user.

1.1. Dimensional data

Dimensional data is a term given by us for any mix of ordinary, temporal, spatial and belief data, including multilevel security data. Over last decade we have worked in dimensional databases, attempting to unify different forms of dimensional data in a seamless framework. We have published a series of papers in dimensional data in a variety of outlets. (See references.) We like to point out that we have consistently professed only one point of view, only one model, and only one language covering all forms of dimensional data, and that everything we have published in last decade is still our current view.

Appendix A gives a brief summary of some of our research in dimensional databases. It is mentioned how multilevel security will seamlessly integrate in our framework and readily avail the many advantages offered by our approach.

1.2. Polyinstantiation in multilevel security and our works

Polyinstantiation refers to the ability of a database model to incorporate different beliefs about the same object in the real world. A simplifying assumption that is prevalent in multilevel security literature is the unikey assumption. The unikey assumption says that identity of an object at all levels is the same. We term polyinstantiation under this assumption as u-polyinstantiation. The full form of polyinstantiation termed key-polyinstantiation would allow the freedom to use different keys at different levels for the same object. A key under key-polyinstantiation is termed polykey.

As shown in Figure 3(a), unikey is typically supported in multilevel security literature by having multiple tuples for the same real world object. In our model unikey is illustrated in Figure 3(b) and 3(c); in each case a single tuple corresponds to an object in the real world. In the first

---

1. “Unikey assumption” is our term to explain nature of polyinstantiation prevalent in multilevel security literature, and also in our works in dimensional databases.
case the tuple is in temporal databases, and in the second case it is a tuple in multilevel security. Clearly, whereas in multilevel security literature u-polyinstantiation is built at the relation level, in our model it is built at the tuple level.  

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Dept</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>50K</td>
<td>Toys</td>
<td>αβ</td>
</tr>
<tr>
<td>John</td>
<td>50K</td>
<td>Shoes</td>
<td>α</td>
</tr>
<tr>
<td>John</td>
<td>80K</td>
<td>Toys</td>
<td>β</td>
</tr>
<tr>
<td>John</td>
<td>90K</td>
<td>PCs</td>
<td>λ</td>
</tr>
</tbody>
</table>

(a) The usual (U-)polyinstantiation in multilevel security literature

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[51,54] 80K</td>
<td>[60,63] Shoes</td>
</tr>
<tr>
<td></td>
<td>[60,75] 90K</td>
<td>[64,75] PCs</td>
</tr>
</tbody>
</table>

(b) U-polyinstantiation in our model of temporal data

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>{αβ,α,β,λ} John</td>
<td>{αβ,α} 50K</td>
<td>{αβ,β} Toys</td>
</tr>
<tr>
<td></td>
<td>{β} 80K</td>
<td>{α} Shoes</td>
</tr>
<tr>
<td></td>
<td>{λ} 90K</td>
<td>{λ} PCs</td>
</tr>
</tbody>
</table>

(c) U-polyinstantiation applied to our model

Figure 3. Polyinstantiation in existing works in multilevel security

[GY88], which articulated the concept of unikeys, is an interesting work in dimensional data. Unikeys work well for any mix of ordinary, temporal, and spatial data, and to this mix of dimensional data the framework of [GY88] applies readily. Although [GY88] was presented in the context of temporal databases, the exercise of applying it to different forms of dimensional data is so straightforward, that we will refer to it as a generic work in dimensional data.

Although polykeys are the defining characteristic of belief data, to the best of our knowledge only unikeys have been considered in the multilevel security literature. Therefore [GY88] also readily applies to multilevel security with u-polyinstantiation. As remarked above, unikeys in multilevel security literature are incorporated at the relation level, but in our case at tuple level. Because of this reason, [GY88] provides a cleaner framework to multilevel security databases. The advantage of [GY88] over existing models in multilevel security is the ease of querying.  

2. Note that this would also apply to many works in temporal and spatial databases, although the concept of unikey is well articulated in our works.

3. See Examples 1 and 2 in Section 2. There we compare querying in [GY88] and [WSQ94]. A reader is encouraged to try these and other queries in model of their choice, and in [GY88] to verify our claims.
1.3. Supporting polykeys

As stated above, polykeys are defining characteristic of belief data, including multilevel security data. Whereas to the best of knowledge polykeys have not been considered in multilevel security literature, we have extensively studied polykeys in evolutionary beliefs [GB89]. In evolutionary belief data, a user forms different beliefs about a fixed momentary state of a real world object at different times. Obtaining a model supporting polykeys, as cleanly as [GY88] supports unikeys, is quite non-trivial. We will discuss polykeys in Section 3, where we will also motivate our model for multilevel security.

Key-polyinstantiation is a delicate concept and demands a careful consideration. Without proper support for key-polyinstantiation the SQL-like framework can become unreliable, user unfriendly and inefficient. The lack of reliability arises from the fact that the correct values and incorrect values have to coexist in a belief database, and the incorrect values can assume the role of keys destroying the identity of objects. It can be shown that even simple algebraic identities such as commutativity of selection with union will cease to hold without proper support of key-polyinstantiation. Due to lack of natural identities, the language can become more imperative and less declarative. This is shown for multilevel security data in Section 3. Besides making the query language less natural, the lack of identities also reduce opportunities for algebraic optimization. Theses problems are contrary to the rich spirit of the relational model, and can bring the database modeling of belief data to the brink of failure irrespective of the choice of the database paradigm.

In [GB89], key-polyinstantiation is supported by adding an anchor to a dimensional value. An anchor is essentially “the correct” belief. For evolutionary objects, the current knowledge of name, say John, can be used as an anchor. Adding an anchor is like having your cake and it too. On one hand we have polykeys, because all differing beliefs about the key are still present. On the other hand the anchor is a unique value, and it amounts to having a unikey. Once we have unikeys, belief data starts to behave like spatial and temporal data.

1.4. Our objective

Our objective is to give a model supporting key-polyinstantiation in multilevel security databases that works without loosing reliability, user friendliness and opportunities for algebraic optimization².

1.5. Our approach and its novelty

This paper may be viewed as an exercise in extending [GY88] to polykeys. In this goal [GB89] guides us. However a reader should note that evolutionary beliefs and multilevel security data are far from being isomorphic, and a model for one does not readily work for the other. In other words, the work in this paper is novel, and it goes over and beyond [GY88,GB89].

The rest of the paper is organized as follows. In Section 2 we discuss other works and discuss why [GY88] provides a superior framework for querying multilevel security data. In Section 3, we discuss the nature of polykeys and motivate our model for multilevel security. In Section 4, we give formal details for the model. In Section 5 we present a query language for this model.

---

² We have a clean and comprehensive framework for optimization in dimensional databases. But that topic is beyond the scope of this paper.
We conclude in Section 6. Appendix A gives a summary of some of our research in dimensional databases.

2. Existing works in multilevel security and [GY88]

To the best of our knowledge, existing works in multilevel security only support unikeys. Some of the papers [JS90, JS91, DLS87, HOT91] have proposed syntactic integrity rules to avoid or interpret polyinstantiation. [CS95] is a relatively more recent paper in this area, and it seems to draw ideas from numerous sources including [DLS88, LDS90, HOT91, JS90, JS91, SW92]. That paper proposes a model and incorporates several semantic requirements which are added as constraints. These constraints become necessary in existing models (including [CS95]) because of the scheme and object fragmentation. In our model most of these constraints become redundant, or reduce to the classical model.

[PMP94] gives a model for integrating temporal and multilevel security data. Such integration is much tighter and naturally built at attribute level in our dimensional model.

The MLS model [SW92]

As illustrated in Figure 4, the model in [SW92] adds two attributes KC and TC (called key classification and tuple classification, respectively) to a classical relation. KC and TC values are levels, such that $KC \leq TC$. A $\langle$Name, Dept, Salary, KC, TC$\rangle$ tuple expresses that the information $\langle$Name, Dept, Salary$\rangle$ belongs to KC and TC concurs with it. Both tuples in the figure represent the belief $\langle$Tom, 10K, Shoes$\rangle$ of $\alpha$. The TC-values $\alpha$ and $\alpha\beta$ in the two tuples say that $\alpha$ and $\alpha\beta$ concur with this belief. In the context of the second tuple we also say that $\alpha\beta$ draws the information $\langle$Tom, 10K, Shoes$\rangle$ belonging to $\alpha$. Information drawing is also built in our model to be presented in this paper.

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Dept</th>
<th>KC</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>100K</td>
<td>Shoes</td>
<td>$\alpha$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>Tom</td>
<td>100K</td>
<td>Shoes</td>
<td>$\alpha$</td>
<td>$\alpha\beta$</td>
</tr>
</tbody>
</table>

Figure 4. Tuples in a relation in [SW92]

[GQ95] gives ramifications of information drawing under update operators. (The concept of information drawing is mentioned above.) Those issues are only of peripheral importance to this paper, which mainly concentrates on querying. In our model the issues articulated in [GQ95] either do not arise, or have natural solutions. [BKS95] provides a deductive framework for multilevel security. A discussion of that work is beyond the scope of this paper.

A model by Winslett, Smith and Qian [WSQ94]

[WSQ94] gives a model where labeling of beliefs is done in a somewhat complicated way. On one hand the thesis of that work seems to be subsumed by [GY88], and on the other hand [GY88] gives a cleaner formulation because of its ability to capture u-polyinstantiation at the tuple level. Below we consider a couple of examples drawn from [WSQ94], adopted to our emp relation. The
reader may informally think of Self and Anyone as two syntactic relations: at a given level L, Self evaluates to the singleton \{L\}, and Anyone evaluates to the set of all levels, L and below, denoted as \(L^-\) in our notation. The point of these examples is to invite a reader to compare the complexity of queries in [WSQ94] and our queries, had we used the framework of [GY88].

**Example 1.** Consider the query *list all names everyone believes to exist*. This query is expressed as follows in algebras of [WSQ94] and [GY88] type model. Note that Self and Anyone in [WSQ94] query have been explained above. \(U\) in our query denotes the set of all users.

\[
\begin{align*}
[WSQ94] & \quad \Pi_{\text{NAME}}(\text{emp}) - \Pi_{\text{NAME}}(\Pi_{\text{NAME}}(\text{emp}) \times \text{Anyone} - ((\Pi_{\text{NAME}}(\text{emp}) \times \text{Self}) \downarrow \text{Anyone})) \\
[GY88] & \quad \Pi_{\text{NAME}} \sigma(\text{emp}, \llbracket\text{NAME}\rrbracket = U, )
\end{align*}
\]

**Example 2.** The query *list employee names believed at my level but no lower level below me*, is expressed in [WSQ94] and our SQL as follows.

\[
\begin{align*}
[WSQ94]: & \quad \text{select NAME} \\
& \quad \text{from emp} \\
& \quad \text{where NAME not in (select NAME} \\
& \quad \text{from emp} \\
& \quad \text{believed by ( select LEVEL} \\
& \quad \text{from Anyone} \\
& \quad \text{where LEVEL not in ( select } \\
& \quad \text{from self)))}
\end{align*}
\]

\[
\begin{align*}
[GY88] & \quad \text{select NAME} \\
& \quad \text{from emp e}^5 \\
& \quad \text{where } \llbracket e \rrbracket = \text{me}
\end{align*}
\]

It should be clear from above examples that [GY88] provides a user-friendly framework for multilevel security. For further confirmation of our claims, a reader is invited to try some queries in any existing model for multilevel security and in [GY88] like framework.

### 3. Key-polyinstantiation

As stated before unikeys are inadequate for belief data. Polykeys are the defining characteristic of belief data where identity of a given real world object may vary from belief to belief. The concept of key-polyinstantiation for belief data was formulated by us in [GB89] in the context of evolutionary beliefs. Key-polyinstantiation also arises in multilevel security. The polykeys in these two cases of belief share some similarities, but they are not isomorphic; therefore, a model for one does not automatically extend to the other.

#### 3.1. Evolutionary beliefs and key-polyinstantiation.

A person known as John today may be considered a case of mistaken identity in future, and the same person may be believed to be Tom. Yet, at another point in future one may realize that there was a misunderstanding, and the person’s name was in fact John. There is no bound on how many times such misunderstandings may occur. Thus for the same object in the real world, our knowledge of its name changes with time. Clearly we have key-polyinstantiation here. A model for evolutionary data with key-polyinstantiation was given by us in [GB89]. Figure 5 shows a key-

5. Note “emp e” creates alias e of the emp relation; this is not a cross product of emp and e
polyinstantiated tuple in evolutionary beliefs.

### 3.2. Key-polyinstantiation in multilevel security data.

Key-polyinstantiation also arises in multilevel security databases (see Figure 2). An object known as John to one user may be known as Tom to another user. Although through our work in [GB89] we know how to deal with key-polyinstantiation in evolutionary beliefs, it turns out that in multilevel security keys are nested. This is illustrated in the following example, which essentially runs through the rest of this section (Section 3).

**Example 3.** Consider the lattice of users \{αβ, α, β, λ\} as in Figure 1. Recall that each of the four users has an object space which is independent of the object spaces of the other users. The entire belief of αβ about an object John is shown in Figure 6(a). Explanation of Figure 6(a) follows.

First, we concentrate on part (i) of Figure 6(a). Each edge from a higher user to lower user expresses a belief of the higher user about how an object known to it is known to the lower user. For example, the edge from αβ:John to α:Tom expresses αβ’s belief that the object known to it as John is known to α as Tom. User α is not even aware of the existence of user αβ, let alone the above relationship. However, α is aware of the user λ, and as shown in the figure, α believes that the object known to it as Tom is known to λ as Hari. The important observation we make now is that αβ does not believe the belief of α, and αβ thinks that the object known to it as John is known as Ron to λ and not as Hari. The reader is highly encouraged to examine all the ramifications of the complexity of beliefs inherent in multilevel security data. It should also be clear to the reader that whereas polykey in evolutionary beliefs have a flat structure, the polykeys in multilevel beliefs are nested.

Now we consider part (ii) of Figure 6(a) which captures the salary of the object known as John to αβ. The backward arrow from the higher user αβ to β says that the salary at level αβ is drawn from the salary at β. In part (iii) of Figure 6(a) we see transitive drawing.

### 3.3. Operators can destroy polykeys

As stated before, key-polyinstantiation for evolutionary beliefs was introduced in [GB89]. As key-polyinstantiation is the fundamental characteristic of belief data, [GB89] is an interesting work in belief data. Key-polyinstantiation is a delicate concept and demands a careful consideration. Without proper support for key-polyinstantiation the SQL-like framework can become

---

6. No term was given to key polyinstantiation in that paper.

7. In this paper we do not go through all the different ramifications about drawing of information. The reader is invited to give reasons as to why this transitivity makes sense.
(i) Object named John

(ii) John’s salary

(iii) John’s department

(a) Information about John available to $\alpha \beta$

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>${\alpha \beta, \beta}$ John</td>
<td>${\alpha}$</td>
<td>${\alpha}$</td>
</tr>
<tr>
<td>${\alpha}$ Tom</td>
<td>${\alpha}$</td>
<td>${\alpha}$</td>
</tr>
<tr>
<td>${\lambda}$ Ron</td>
<td>${\alpha}$</td>
<td>${\alpha}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>${\alpha}$ Tom</td>
<td>${\alpha}$</td>
<td>${\alpha}$</td>
</tr>
<tr>
<td>${\lambda}$ Hari</td>
<td>${\alpha}$</td>
<td>${\alpha}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>${\beta}$ John</td>
<td>${\beta}$</td>
<td>${\beta}$</td>
</tr>
<tr>
<td>${\lambda}$ Ron</td>
<td>${\beta}$</td>
<td>${\beta}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>${\lambda}$ Hari</td>
<td>${\lambda}$</td>
<td>${\lambda}$</td>
</tr>
<tr>
<td>${\lambda}$ Ron</td>
<td>${\lambda}$</td>
<td>${\lambda}$</td>
</tr>
</tbody>
</table>

(b) Our storage model for polyinstantiation of the object John

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>${\alpha \beta, \beta}$ John</td>
<td>${\alpha \beta}$</td>
<td>${\alpha \beta, \alpha}$</td>
</tr>
<tr>
<td>${\alpha}$ Tom</td>
<td>${\alpha}$</td>
<td>${\beta}$</td>
</tr>
<tr>
<td>${\lambda}$ Ron</td>
<td>${\lambda}$</td>
<td>${\lambda}$</td>
</tr>
</tbody>
</table>

(c) The query content of the first tuple from (b)

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha \beta$ John</td>
<td>${\alpha \beta}$</td>
<td>${\alpha \beta, \alpha}$</td>
</tr>
<tr>
<td>${\alpha}$ Tom</td>
<td>${\alpha}$</td>
<td>${\beta}$</td>
</tr>
<tr>
<td>${\lambda}$ Ron</td>
<td>${\lambda}$</td>
<td>${\lambda}$</td>
</tr>
</tbody>
</table>

(d) The tuple of (c) after anchoring

Figure 6. An object named John known to $\alpha \beta$
unreliable, user unfriendly and inefficient. The lack of reliability arises from the fact that the correct values and incorrect values have to coexist in a belief database, and the incorrect values can assume the role of keys destroying the identity of objects. It can be shown that even simple algebraic identities such as commutativity of selection with union will cease to hold without proper support of u-polyinstantiation. Due to lack of natural identities, the language can become more imperative and less declarative. This is shown for multilevel security data later in this section. Besides making the query language less natural, the lack of identities also reduce opportunities for algebraic optimization. From our point of view, these problems are contrary to the rich spirit of the relational model, and can bring the database modeling of belief data to the brink of failure irrespective of the choice of the database paradigm.

In [GB89], key-polyinstantiation is supported by adding an anchor to a dimensional value. An anchor is essentially “the correct” belief. For evolutionary objects, the current knowledge of name, say John, can be used as an anchor. Adding an anchor is like having your cake and it too. On one hand we have polykeys, because all differing beliefs about the key are still present. On the other hand the anchor is a unique value, and it amounts to having a unikey. Once we have unikeys, belief data starts to behave like spatial and temporal data.

Figure 6(b) shows the tuples which capture the information content of Figure 6(a). These tuples represent the storage structure for the information contained in them. Figure 6(c) shows the full query contents of (only) the first tuple from Figure 6(b). Note that in this paper we incorporate key-polyinstantiation at the relation level and not at the tuple level.

3.4. Anchors

To illustrate anchors, we will use the Name attribute value in the tuple of Figure 6(c) as our running example. For ease of reference it is also shown in Figure 7(a).

![Figure 7](image)

(a) Value without anchor  
(b) Value with anchor

Figure 7. The NAME value of Figure 6(c) with and without anchor

The purpose of an anchor is to provide a unikey at the relation level. In case of the belief in Figure 7(a), the unikey would be \(\langle \alpha \beta \rangle \text{ John}\): which says “this is \(\alpha \beta\)’s belief about the object John in its \(\alpha \beta\)’s object space”. This unikey should serve us well, because it is capable of distinguishing this belief from a belief of user \(\alpha\) about an object John in \(\alpha\)’s object space. An anchor is nothing but this unikey; Figure 7(b) shows belief value of Figure 7(a) after anchoring.

8. Figure 6(d). Note that we can represent all the different tuples in Figure 6(d) by using path expressions such as \(\alpha \beta, \beta, \lambda\) the points in the dimensional space, but this gives rise to some other complications. This may be treated in our future publications.

9. We note that in evolutionary beliefs, a user is concerned with its own beliefs at different times. Therefore \(\langle \text{John} \rangle\) would suffice as the unikey.
3.5. Anchors make algebraic operators meaningful

The information in belief data can be fragile when algebraic operators are applied. Now we discuss how anchors can be used to protect the identity of objects from destruction due to algebraic operators. We illustrate the use of anchors through an example. We informally consider a selection of the form \( \sigma(r, \mu) \). The selection retrieves the functional restriction of relation \( r \) to domain \( \mu \).

Next consider the following algebraic identity.

\[
\sigma(r, \mu) \cup \sigma(r, \nu) = \sigma(r, \mu \cup \nu)
\]

Intuitively a user expects this algebraic identity to hold. First we use the unanchored value in Figure 7(a) and carry on a naive computations and show that it will fail to yield the algebraic identity. Then we use the anchored value in Figure 7(b), and use it as a keying mechanism and show how the algebraic identity is obtained. We will use the following values for \( r, \mu \) and \( \nu \):

- For \( r \) we will use the single object of Figure 7. (In other words \( r \) is a single attribute single tuple relation.)
- \( \mu = \{\alpha, \beta\} \)
- \( \nu = \{\beta, \lambda\} \)

**Naive computation.** In this case \( r \) is the un-anchored object of Figure 7(a). The naive computations are shown in Figure 8(a). Starting with \( r \), we compute \( \sigma(r, \mu), \sigma(r, \nu), \sigma(r, \mu) \cup \sigma(r, \nu) \) and \( \sigma(r, \mu \cup \nu) \). From parts (iii) and (iv) of Figure 8(a), it is clear that \( \sigma(r, \mu) \cup \sigma(r, \nu) \) and \( \sigma(r, \mu \cup \nu) \) are not equal; they don't even have the same number of tuples!

![Figure 8(a)](image)

(a) Without anchoring \( \sigma(r, \mu) \cup \sigma(r, \nu) \neq \sigma(r, \mu \cup \nu) \)

![Figure 8(b)](image)

(b) With anchoring \( \sigma(r, \mu) \cup \sigma(r, \nu) = \sigma(r, \mu \cup \nu) \)

*Figure 8. Persistence of identities with anchors*

(Note that \( \mu = \{\alpha, \beta\} \), and \( \nu = \{\beta, \lambda\} \))
Computation with anchors. In this case \( r \) is the anchored object of Figure 7(b). Figure 8(b) shows computations with anchors. With anchoring, the results of \( \sigma(r, \mu) \), \( \sigma(r, \nu) \), \( \sigma(r, \mu) \cup \sigma(r, \nu) \) and \( \sigma(r, \mu \cup \nu) \) are also shown. Note that when the system computes union (in this case \( \sigma(r, \mu) \cup \sigma(r, \nu) \)), the collapse of two tuples is with respect to their anchors. This permits the restoring of an object (belief) in a single tuple after it is decomposed into several tuples. It is clear that Figure 8(b)(iii) shows the result of \( \sigma(r, \mu) \cup \sigma(r, \nu) \) as well as \( \sigma(r, \mu \cup \nu) \).

3.6. Anchoring is done by the system

Our goal is to design a query language which is as natural for a user as possible. In order to achieve this we favor passing the complexities of belief data on to the system. Anchors help us in meeting this non-trivial objective. We note that anchoring is done automatically by the system before executing a query. The user makes no effort in the underlying process to carry out the algebraic computations.

3.7. Partitioned associative navigation

In databases the construct \( A \theta B \) is of fundamental importance. In the conventional 1nf approach, the attribute values \( A \) and \( B \) are atomic, therefore, \( A \theta B \) evaluates to TRUE or FALSE. In our dimensional models, \( A \) and \( B \) are functions, and \( A(p) \theta B(p) \) is TRUE at some points \( p \) and FALSE at others. Therefore, the appropriate counterpart of \( A \theta B \) in our dimensional models is \( \llbracket A \theta B \rrbracket \), which computes the set of points \( p \) where \( A(p) \theta B(p) \) holds. In other words, in dimensional data, \( \llbracket A \theta B \rrbracket \) is a domain expression and not a boolean expression.

Though in our existing dimensional models the above definition of \( \llbracket A \theta B \rrbracket \) is satisfactory, in security data we need to reexamine this in the context of anchors. We could have the following two options:

- Define \( \llbracket A \theta B \rrbracket \) as above, i.e. \( \{ p : A(p) \theta B(p) \text{ holds} \} \)
- Define \( \llbracket A \theta B \rrbracket \) more conservatively as \( \{ p : \text{beliefs } A \text{ and } B \text{ belong to the same user and } A(p) \theta B(p) \text{ holds} \} \)

The second choice is more natural for multilevel security; it keeps the object spaces belonging to different users partitioned as associative navigation is done. Perhaps the best justification for such a choice can be given in terms of natural join. For instance consider Example 8 where the natural join (equi-join to be exact) \( \llbracket \text{emp.DEPT} = \text{management.DEPT} \rrbracket \) is used. It is clear that the partitioned computation would yield a natural result.

In the next section we give formal details for our model. The model keeps object spaces of different levels partitioned. Such partitioning is build from ground up, applying it to domains, values, tuples and associative navigation.

4. The model SecDB: Formal details

In this section we give the formal details for our model. As stated at the end of the previous section, the model keeps object spaces of different levels partitioned. Such partitioning is build from ground up.

A security element \( \mu \) is a subset of the \( \Lambda \), the lattice of all users. A security element allows us to represent a user’s knowledge about a real world object. Clearly, the set of all security elements is closed under union, intersection and complementation. These closure properties lay a foundation
for uniform handling of “or”, “and” and “not” of natural languages in a query language [GN93].

A u-element is a subset of \( u^- \). An example of an \( \alpha\beta \)-element is \( \{\alpha\beta, \alpha, \lambda\} \). The purpose of this security element is to allow the user \( \alpha\beta \), to assemble a belief about a real world object known to \( \alpha\beta \) (ownself), \( \alpha \) and \( \lambda \). The u-elements are closed under union, intersection. The u-elements are also closed under complementation when the complementation is computed with respect to \( u^- \).

A u-assignment to an attribute A is a function \( \xi \) from a u-element into \( \text{dom}(A) \), the domain of A. For example, \( \langle \{\alpha\beta, \alpha\} \text{ Jack} , \{\beta\} \text{ John} , \{\lambda\} \text{ Tom} \rangle \) is an \( \alpha\beta \)-assignment to the attribute NAME.

Whereas, in the conventional 1nf approach, an attribute value in existing multilevel security models capture information pertaining to only one level of a belief, in our model we assemble the entire information about a belief in a (single) attribute value. The non-1nf nature of our tuples makes a substantial simplification in user queries [GN93]. The domain of \( \xi \) is denoted as \( [\xi] \).

For example, \( [\langle \{\alpha\beta, \alpha\} \text{ Jack} , \{\beta\} \text{ John} , \{\lambda\} \text{ Tom} \rangle] = \{\alpha\beta, \alpha, \beta, \lambda\} \).

Now we formalize the concept of anchor. First we discuss anchored domains, and then anchored (attribute) values. The purpose of anchors is to provide persistence to identity of an object while it undergoes an algebraic operation. Our definitions also keep operands partitioned at various levels. The level \( L \) is indicated by the prefix “L:”.

4.1. Anchored security elements

We assume that we have an attribute A in mind, and \( \text{dom}(A) \) denotes domain of A. An anchored u-element is an ordered pair \( (\{u\}, \mu) \), where \( \mu \) is a subset of \( u^- \). For ease of readability, we denote a u-element \( (\{u\}, \mu) \) as \( u:\mu \).

Example 4. \( \alpha\beta: \{\alpha\beta, \alpha, \lambda\} \) is an anchored \( \alpha\beta \)-element; its anchor is \( \alpha\beta \) and its domain is \( \{\alpha\beta, \alpha, \lambda\} \). Another example of an anchored element is \( \beta: \{\lambda\} \), but \( \beta: \{\alpha\} \) is not an anchored element, because \( \alpha \) is not below \( \beta \).

Now, we define union, intersection and complementation of anchored u-elements for a fixed value of \( u \). Basically, for a fixed anchor \( u \), these operations are performed on the domains, with the \( u^- \) as the universe. For example,

\[
\begin{align*}
\alpha\beta: \{\alpha, \lambda\} \cup \alpha\beta: \{\beta, \lambda\} & = \alpha\beta: \{\alpha, \beta, \lambda\} \\
\alpha\beta: \{\alpha, \lambda\} \cap \alpha\beta: \{\beta, \lambda\} & = \alpha\beta: \{\lambda\} \\
\alpha\beta: \{\alpha, \lambda\} - \alpha\beta: \{\beta, \lambda\} & = \alpha\beta: \{\alpha\} \\
- \alpha\beta: \{\alpha, \lambda\} & = \alpha\beta: \{\alpha\beta, \beta\} \\
- \alpha: \{\alpha, \lambda\} & = \alpha: \emptyset
\end{align*}
\]

It is clear that the above definitions provide each user \( u \), a space of anchored elements which are closed under union, intersection and complementation. However, this definition is not an end in itself. It is not our intention to have different relations for different users. A relation \( r \) is shared by all users and we want to be able to compute its domain.

An anchored domain \( \mu \) is a (finite) set of anchored u-elements, such that for a given user \( u \), there is at most one u-element in \( \mu \). This definition forces different u-elements for the same value of \( u \) to be coalesced together. The union, intersection and complementation among anchored domains is defined naturally.

Example 5. The following is an example show how the user \( \alpha\beta \) may perform union, intersection and complementation among anchored domains.
Example 6. The user β is unaware of αβ and α. Therefore, the user β will see the following part of the last identity in the previous example.

{ β : { λ } } = { β : { λ } , λ : { λ } } 

It is easily verified that union, intersection and complementation satisfy properties of a boolean algebra at every step of the definition. As all our definitions in dimensional databases, these definitions are designed to arrive at a query language which is easy to use.

The last definition will allow us to compute [[r]], the domain of a relation. The operator [[r]] is of fundamental importance in dimensional databases. It allows us to capture the concept of “when” in a temporal database, “where” in a spatial database, and “when and where” in a spatio-temporal database. In those databases, the definition of [[r]] is very simple. [[r]] plays a fundamental role in the syntax of our SQL-like algebraic query language. On one hand it makes the syntax highly recursive and natural, and on the other hand it allows users to express “selections” from natural languages as selections instead of joins in our query language. In multilevel security, the definition of [[r]] is complex compared to other forms of dimensional data. It allows us to capture “who” in a secure manner. We remark that the said complexity in the definition of [[r]] is not passed on to the user, but rather it is absorbed by the system.

4.2. Anchored attribute values

A u-anchor over A, or simply an anchor is a function from a singleton {u} to dom(A). An anchored u-assignment with the anchored u-element u:μ as its domain, is a pair of functions (ξu:ξv), denoted ξu:ξv, where ξu is a u-anchor, and ξv is a function from u− into dom(A) such that if ξv is defined at u, it has to agree with ξu. The domain of ξu:ξv, denoted [[ξu:ξv]] is defined as the anchored domain [[ξu]]:[[ξv]]. We revisit our example from Section 3.4.

Example 7. The attribute value ⟨{αβ,β} John, {α} Tom, {λ} Ron⟩ belonging to user αβ after anchoring becomes ⟨αβ John: (αβ,β) John, {α} Tom, {λ} Ron⟩. Figure 7(b) shows a tabular representation of this anchored attribute value. In addition the domain of the anchored value is αβ: {αβ,α,β,λ}.

4.3. Associative navigation

Motivation for this has already been given at the end of previous section. Here we merely restate that [[AβB]] is defined as {p: beliefs A and B belong to the same user and A(p)θB(p) holds}. The requirement “beliefs A and B belong to the same user” keeps the object spaces partitioned.

4.4. Tuples

A u-tuple, or simply a tuple is a concatenation of u-assignments whose security domains are the same. The assumption that all u-assignments in a tuple have the same domain is called the homo-
geneity assumption [GN93]. Note that in all the existing models, security level is used as an attribute, and thus tuples in these models are a priori homogeneous. Thus, the homogeneity assumption does not limit our modeling power when compared to the existing approaches. Figure 6(d) shows example of an employee tuple over NAME SALARY DEPT, with NAME as its key. This tuple belongs to the user αβ. The tuple represents a single real world object, known as John to the users αβ and β, as Tom to α, and as Ron to λ.

4.5. Relations

A relation r over a scheme R, with K⊆R as its key, is a finite set of non-empty homogeneous tuples such that for a given level, no two tuples can have the same key.

5. The query language SecSQL

In this section we give a brief introduction to SecSQL, the query language for our model. Expressions in SecSQL can be divided into three mutually exclusive groups: domain expressions which evaluate to security elements, boolean expressions which evaluate to TRUE and FALSE, and relational expressions which evaluate to security relations. The most interesting operator in SecSQL is its SQL-like select statement. The select statement draws its simplicity and power through associative navigation made possible by domain expressions and boolean expressions. In this section we primarily concentrate upon the SQL-like select statement of SecSQL and giving several examples.

- [[AθB]] is a domain expression which extracts users where A and B are in θ-relationship. For example, if A is the security assignment <αβ a1: {αβ} a1, {α} a2, {λ} a3>, and B is <αβ a1: {αβ} a1, {α} a2, {β} a3>, then [[A=B]] evaluates to αβ:{αβ,α}.
- If e is a relational expression, then [[e]] is a domain expression, whose value is the union of domains of tuples in the relation computed by e. For example, [[emp]] is {αβ:{αβ,α,β,λ}, α:{α,λ}, β:{β}, λ:{λ}}. This construct is a source of powerful nesting among SecSQL expressions. It can also be used by itself as a query.
- [[AθB]] and [[e]] are atomic domain expressions, they evaluate to security elements. If µ and ν are domain expressions, then so are µ∪ν, µ∩ν, µ−ν and ¬µ.
- If µ and ν are domain expressions, then µ⊆ν is a boolean expression.
- We define AθB to be an abbreviation of the boolean expression [[AθB]] ≠ ∅.
- If f and g are boolean expressions then so are f∨g, f∧g and ¬f.

The select statement

The select statement of SecSQL is the most interesting one. It is similar to the select statement in SQL. The main difference is the addition of the “restricted to µ” clause which allows the retrieval of a tuple to be restricted to the domain specified by the domain expression µ. In SecSQL it has the following form.

```
select      X: K
restricted_to µ
from        r1, r2, ..., rn
where        f
```
5.1. Query examples

We assume a database called PersonnelDB. The database consists of two relations: emp (NAME SALARY DEPT) and management (DEPT MANAGER). We assume that NAME is the key of emp, and DEPT is the key of management. We also assume that DEPT and MANAGER functionally determine each other. Note that in our model we do not need any additional attributes beyond those in a classical relation, yet our modeling power extends the existing models in multilevel security. Not all data is visible to every user. A user at level u can query all tuples at level u and below.

We use ME to represent the level of the user. Therefore, ME will be substituted by the user (level) who submits the query. Also recall that \( u^- \) denotes the set of all users below u (including u).

Example 8. (For user \( \alpha \beta \)) Give managers of all employees in all object spaces.

\[
\text{select } \text{NAME}, \text{emp.DEPT}, \text{management.MANAGER} \\
\text{from } \text{emp, management} \\
\text{restricted_to } [\text{emp.DEPT} = \text{management.DEPT}] \\
\]

This is basically a natural join of emp and management, except that the attribute SALARY is not projected. The user complexity of the query is comparable to a classical query. 10

Example 9. (For any user) Give managers of those employees known only to me.

\[
\text{select } \text{NAME}, \text{management.MANAGER} \\
\text{from } \text{emp, management} \\
\text{restricted_to } [\text{emp.DEPT} = \text{management.DEPT}] \\
\text{where } [\text{NAME}] = \text{ME} : \{ \text{ME} \} \\
\]

Example 10. (For user \( \geq \alpha \)) Give the identities believed by \( \alpha \) of objects in user \( \alpha \)'s object space.

\[
\text{select } \text{NAME} \\
\text{restricted_to } \alpha : \{ \alpha \} \\
\text{from } \text{emp} \\
\]

Example 11. (For user \( \geq \alpha \)) Give all identities of objects in user \( \alpha \)'s object space.

\[
\text{select } \text{NAME} \\
\text{restricted_to } \alpha : \alpha^- \\
\text{from } \text{emp} \\
\]

Example 12. (For any user) Give the identities of object “John” in my object space.

\[
\text{select } \text{NAME} \\
\text{from } \text{emp} \\
\text{where } \text{ME} : \text{ME}^- \cap [\text{NAME}=\text{John}] \neq \emptyset \\
\]

Example 13. (For any user) Give all objects in my object space which have the identity “John” at any level.

10. Note that the conservative choice for \([\text{A} \theta \text{B}], [\text{emp.DEPT} = \text{management.DEPT}]\) in this case (see 2nd bullet in Section 4.3) manages to keep the object spaces of all users partitioned yielding a satisfactory result.
select * from emp
where ME: {ME} \subseteq \{\text{NAME}\} \text{ and } \{\text{NAME}=\text{John}\} \neq \emptyset

**Example 14.** (For any user) *Give the security points at which the identity “John” exists.*

\[[\text{select } * \text{ restricted to } \{\text{name=John}\} \text{ from emp}]\]

**Example 15.** (For any user) *Give the identities of objects in the whole object spaces, restricting to the security points at which the identity “John” exists in some object space.* Note that the query of Example 14 is nested here.

\[
\text{select NAME restricted_to } [\text{select } * \text{ restricted_to } \{\text{Name}=\text{John}\} \text{ from emp}] \text{ from emp}
\]

**Example 16.** (For any user) *Give the identities of the employees known only to my level.*

\[
\text{select } * \text{ from emp where } \{\text{NAME}\} = \text{ME:ME}
\]

**Example 17.** (For any user) *Give the identities of the employees known to every level.*

\[
\text{select } * \text{ from emp where } \{\text{NAME}\} = \text{ME:ME}^\sim
\]

**Example 18.** (For any user \(\geq \alpha\)) *Give all objects, in my object space, of which the names believed by level \(\alpha\) and \(\lambda\) are the same.* Note that \(|\text{NAME}({\text{ME}^\alpha})|\) retrieves the name after stripping the security point \(\text{ME}^\alpha\), \(|\cdot|\) is a “read only operation” being used in a comparison here. In our model a value cannot be written after stripping its security point, except in a terminal situation such as printing a report for human consumption.

\[
\text{select } * \text{ from emp where } |\text{NAME}((\text{ME}:\alpha))| = |\text{NAME}((\text{ME}:\lambda))|
\]

6. Conclusion

It should be clear from our query examples that we have successfully incorporated polykeys in our model for multilevel security data. In particular a reader is welcome to examine query in Example 8, and think through all that is implicit in it and the central fact that the complexity of the query is comparable to the classical case. Operators in our model for multilevel security turn out to be more robust than their counterpart in evolutionary beliefs [GB89]. A discussion of this point is beyond the scope of this paper. The techniques used in [GB89] gave us some ideas but they needed substantial changes before they became applicable to multilevel security.

The major strength of our approach is the ease of query for a user, and in addition the fact that the user never has to worry about cooking up information, e.g. inadvertently splicing a belief of a user to a different user.
We feel that this paper provides the most powerful and most user-friendly framework of all existing approaches to multilevel security. Imposition of constraints in the existing security database models seems to be necessary because of schema and object fragmentation. Our model avoids such fragmentation: our relational schemes are same as those in classical relational databases (we do not need attributes such as “TC, tuple classification”), and our tuples are richer in information content. Because of these reasons, most of the constraints appearing in the security literature are built-in our model. A discussion of this issue is beyond the scope of this paper.

Our approach in dimensional data has lead to seamless integration of ordinary, spatial, temporal and multilevel security data. A summary of the advantages offered by our approach is given in Appendix A; multilevel security could benefit from these advantages.

The prevailing folk wisdom in databases recognizes two important constructors in databases: tuple constructor and set constructor. We feel that there is much to be gained by adding the dimension constructor to this list. The various forms of seamlessness, uniformity and scalability of our SQL like language represent a promise to industry and users for a smooth transition from classical databases to temporal, spatial and security databases with oids and type hierarchy.

REFERENCES


11. Most of these constraints are cataloged (and imposed) in [CS95].
References to our works related to this paper


Features of our dimensional model: the big picture where this work fits

We have presented a model and an SQL like language to query and update belief information for a multilevel security database. The model has been developed within the framework of our ongoing research in dimensional data [NG92], [CGN93], [GPN92], [CG94], [BG93], [Ga88], [GY88], [GV85], [GB89], [Ch95], [GN93]. Under the banner of dimensional data we have unified ordinary, temporal, spatial and belief data. Considerable research has already been done in dimensional databases. Following is a list of some of the features of dimensional databases which need to be studied to see how well these features continue to hold when security is added as a dimension under our approach.

- **Dimensional seamlessness and query reuse.** Our SQL like query languages have so far remained essentially independent of the choice of the dimensional data. The seamless integration of the different forms of dimensional data is achieved through the concept of dimension alignment, which allows a query on a form of dimensional data (e.g., a temporal query) to be used literally without any change in higher forms of dimensional data (e.g., spatio-temporal) [CGN93]. We expect this feature to extend to security data; thus e.g. one should be able to use a security query for temporal data in a security-spatio-temporal database without any changes.

- **Boolean seamlessness.** The constructs “or,” “and” and “not” of natural languages are incorporated symmetrically in our SQL. Such a seamlessness is not possible in existing models of dimensional data because in these models the dimensional domain such as security level is used as a column in a relation [GN93].

- **Fewer joins.** In our model the number of joins one needs are comparable to those in classical databases. In existing models of temporal, spatial and security databases the database representation of a real world object is often fragmented into a potentially unbounded number of fragments. This makes those query languages more difficult for users; before a query can be formulated, unnecessary joins have to be performed in order to paste different fragments together. In this context, the construct [[r]] of our model is very useful. It allows a relational expression to be turned into a domain expression, which can be nested in a relational or boolean expression. This construct is natural as well as powerful. It allows natural language selection to be realized as selections in our algebra [GN93]. The construct is also very efficient to implement.

- **Algebraic query optimization.** Because of fewer joins the queries in our algebra tend to be more efficient. But in addition, dimensional data give rise to interesting algebraic identities which provide a foundation for algebraic optimization [NG92]. This framework needs to be extended with the addition of security as a dimension.

- **Integrity of security information.** If the security level can be replaced by a different security level (e.g. replacing u1 John by u2 John), we would compromise the integrity of security information. This cannot happen in our model because a value is permanently glued to a security level, and the two cannot be separated. However, it can happen in existing models where security level is used as a column in a relation.

- **Object ids, type hierarchy and belief consistency.** In recent work we have extended our relation framework for dimensional data to incorporate type hierarchy and oids without sacrificing
other features [CGN93,Ch95]. In case of security model, this has additional potential for us to give a cleaner treatment for information drawing from lower users.

• **Pattern matching languages for application development.** Our SQL like languages are interesting and fairly powerful, but they do not eliminate the need for embedding SQL in a lower level language such as C++. We have given powerful pattern matching languages for temporal and spatial databases [CG94]. The pattern matching languages are tailor designed to deal with linguistic peculiarities of different forms of dimensional data and substantially reduce the need for embedding SQL in C++. A pattern matching language can be viewed as a generalization of [[A@B]] and be seamlessly integrated with SQL for dimensional data. Such a language needs to be developed for security data from scratch.

• **Incomplete information.** We have given a model for incomplete information for temporal databases [BG92]. We expect the model to extend to security all forms of dimensional data, including security data.

• **Compatibility with classical databases.** For classical user of a classical database, the dimensional space degenerates into a single point. In case of security data, this point is \( \lambda \). When there is only one point in a dimensional space, the construct [[A@B]] reduces to A@B and the distinction between the restricted_to and where clauses disappears. Therefore, for such a user, the restricted_to clauses can be suppressed, and a dimensional algebra can be made to look like classical algebra [BG93]. In this way we achieve the full compatibility with the classical databases.