14

# SECURITY ADMINISTRATION FOR FEDERATIONS, WAREHOUSES, AND OTHER DERIVED DATA

Arnon Rosenthal, Edward Sciore and Vinti Doshi

**Abstract** Security administration is harder in databases that have multiple layers of derived data, such as federations, warehouses, or systems with many views. Metadata (e.g., security requirements) expressed at each layer must be visible and understood at the other layer. We describe several use cases in which layers negotiate to reconcile their business requirements. The sources must grant enough privileges for the derived layer to support the applications; the derived layer must enforce enough restrictions so that the sources' concerns are met; and the relationship between the privileges at source and derived layer must be visible and auditable.

The guiding principle is that a security policy should primarily govern information; controls over source tables and views from which the information can be obtained are intended to implement such policies. We require a kind of global consistency of policy, based on what information owners assert about tables and views. Deviations are primarily a local affair, and must occur within safe bounds. Our theory examines view definitions, and includes query rewrite rules, differing granularities, and permissions granted on views. Finally, we identify open problems for researchers and tool vendors.

**Keywords:** Data warehouses, federations, security administration

## 1. INTRODUCTION

Federations and warehouses have multiple layers of schemas [4]; they are examples of *multi-layer* database systems. Data originates in a *source* layer. The *derived* layer consists of information derived from the sources. The system should have an integrated security policy, so that each granule of data is protected consistently, whether obtained through the source or derived layer.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35508-5_22

V. Atluri et al. (eds.), *Research Advances in Database and Information Systems Security*

Creating such a policy requires negotiations between administrators at different layers.

Our main contribution is to provide a unified model for administering many derived-data architectures, in a way that exploits the power of today's SQL DBMSs. Especially by exploiting the power to rewrite queries, we can provide wider access without compromising security. We present motivating scenarios, and sketch the necessary theory, methodologies and automated tool Our main contribution is to provide a unified model for administering many derived-data architectures, in a way that exploits the power of today's SQL DBMSs. Especially by exploiting the power to rewrite queries, we can provide wider access without compromising security. We present motivating scenarios, and sketch the necessary theory, methodologies and automated tool support.

## 1.1 THE NEED FOR COLLABORATION

The layers in a multi-layer database are often administered separately, which makes security administration harder than in a centralized system. Administrative activities at one layer may affect other layers, and thus need to be visible and understood at these layers. For example, if a source layer grants or revokes the access permissions on one of its tables, it must communicate this to derived layers that use that table.1 Similarly, if a derived layer wishes to give a particular subject (e.g., user, role, group) access to one of its tables, it must request permission from the (source layer) owners of information used to derived the table, in terms of each owner's schemas.

The administrators of each layer must be able to negotiate with each other, in order to match security and business requirements. In particular, negotiation aims at:

- Protecting information, as its owners require.
- Giving sufficient access permissions so applications can accomplish their tasks.
- Enforcing the agreed-upon access policies as efficiently and reliably as possible.
- Allowing administrators at different layers to understand each other's concerns, without needing to understand each other's schema and the derivation logic.

## 1.2 SOURCE VERSUS DERIVED PERMISSIONS

Administrators need to see all relevant security metadata, whether initially provided in terms of their "native" schema or a foreign schema. To enable this

visibility, negotiation support must include a translation step. We sketch the theory here, and revisit in more detail in section 3.

Current multi-layer systems do not really connect the source-layer and derived-layer permissions. For example in a federation, a middleware product (as the owner of federation views) may be given unlimited Read access to sources, with Grant option. This approach places great burdens and trust on the federation administrator, but provides neither guidance nor tools. In particular, there is no formal basis for source administrators to judge whether the permissions that the federation administrator grants are appropriate.

Our approach is based on three premises:

1. Ownership and access controls are fundamentally about *information*, not physical artifacts (source tables) or interfaces (views). That is, protection must be global. This implies that access controls asserted at one schema must be *propagated* to the other schemas.

2. A table's permissions are the sum of two sources: permissions explicitly asserted on the table, plus permissions *inferred* from permissions on other tables. If you have Read permission on tables X and Y, you get permission on any view you can compute from X and Y. Conversely, one gets read permission for a view V if there is *any* query that computes V from tables to which you have access.

3. Administrators of implementation units (source tables, views, replicas) may reduce the permissions allowed by (2), but not expand them. These restrictions often stem from a desire to control physical resources. To protect response times or generate revenues, one database might access only by known subscribers. To reduce hacker threats, another might allow access only by the enterprise's employees.

These premises guarantee that the access permissions in a database are consistent. That is, if a user has access to information in one part of the database, the user has access to that information everywhere in the database. Section 3.5 discusses a mechanism for superimposing local restrictions. This approach differs from earlier work [1, 2], that took preservation of administrator autonomy as the cornerstone. While any security system must permit autonomy to be preserved, we contend that power-sharing does not provide a suitable bases for rationalizing system-wide security. We believe that SQL Grant provides more flexibility than "Export" permissions, and can preserve autonomy where needed.

## 1.3 ASSUMPTIONS AND SIMPLIFICATIONS

Our work does not address the usual problems of integrating distributed heterogeneous data, nor deal with all aspects of security for an enterprise database.

We assume that a metadata management environment (denoted *MME*), provides the illusion of several kinds of uniformity:

- *Schemas, metadata, and distributed databases*: The source and view schemas and their metadata are represented in a repository. All schemas and views are in the same language (e.g., SQL).
- *Permission types*: MME defines a uniform set of permission types (e.g., Create, Read, Update, Delete), across all layers.
- *Subjects to whom privileges are granted*: MME manages grants of access permissions to subjects (e.g., users or roles) that are globally known across layers. Authentication, management of role membership, and subject globalization (i.e., specifying which global subject corresponds to each DBMS or operating system subject) are outside our scope.

Some further assumptions simplify our prose. We believe that each of them could easily be removed, as indicated:

- We speak in terms of only two layers - a source layer and a derived layer - so each administrator sees just one "foreign" layer. (Actually, the derivation graph can be arbitrary.)
- We assume that view definitions are readable by all users (rather than being protected objects).As a consequence, ownership of a view is insignificant - anybody can define an equivalent view. The real issue is who is permitted to grant access to the view's information. (As in SQL, we may wish to let some users access a filtered version of source data.) We assume that anyone withGrant(Read ) permission on a view's underlying tables can Grant Read on the view.
- We ignore delays in propagating data and metadata updates.

Federations and warehouses bring together a great deal of information, increasing the *aggregation vulnerabilities*: Information that in isolation was not sensitive can become sensitive when made accessible together, e.g., via a federation [3]. However, the proposed solutions to this problem for centralized systems seem to provide modest extra security, at very high cost in run-time and administration (e.g., quadratic growth). We therefore do not address this issue.

## 2. NEGOTIATION SCENARIOS

The layers in a multi-layer database may be administered separately, with conflicting goals. This section illustrates the kinds of negotiations that must be

performed, and implicitly the kinds of capabilities a metadata coordination tool must provide. Its research contribution is to identify, from the overwhelming set of real requirements, scenarios that point at useful, feasible facilities for permission negotiation and translation. In particular, we try to answer:

- *What sort of negotiations do the layers' administrators need to perform?*
- *What sorts of requests do they need to send to each other?* There appear to be a mix of commands, proposals, notifications, and comparisons.
- *How are requests for permissions translated to the recipients' schemas?* Each party needs to know what has been said, in terms of its own schema.

Many tasks involve a set of permissions against the requestor's schema, a target schema, a recipient for the permissions (translated to refer to the target schema), and a desired action. The action may be a command to install the translated permissions ("Enforce these permissions at your layer."), a proposal ("I would like you to cause these permissions to be true."), a description of what the layer is doing ("For your information: Here are the permissions I enforce."), or a hypothetical display ("If I granted these permissions, how would it look on the target schema?")

The scenarios below present more detail. The following 2-layer Hospital database provides a running example. The source layer contains the two base tables:
PATIENT (Patient_Id, Insurance_Co)
PROCEDURE (Patient_Id, Procedure_Performed, Doctor, Bill_Amount, Date)

The view layer is a data warehouse containing the derived tables
DOCTOR_ACTIVITY (Doctor, Procedure_Performed, Month, Year)
INSURANCE (Insurance_Co, Month, Year, Total_Billed)

The first (materialized) view is a selection from PROCEDURE; the second joins PATIENT and PROCEDURE, and then groups by Insurance_Co and (Month, Year) from Date, and totals the Bill_Amount. The derivations may involve complex fusion and scrubbing logic.

## 2.1 BOTTOM-UP: CONFORMING TO IMPLIED PERMISSIONS

In the first scenario, the warehouse handles user requests without referring back to sources, and hence checks all permissions itself. The source layer keeps the warehouse informed about what access permissions to enforce.

Whenever a grant or revoke command occurs at the source, the metadata management environment transmits the command to the warehouse, and translates the source's command to an appropriate action on the appropriate view tables. For example, a grant on PROCEDURE translates to a grant on DOCTOR_ACTIVITY and, if there is an existing corresponding grant on PATIENT, a grant on INSURANCE.

Automated translation is crucial, both to reduce administrator workload and to keep the system secure in a dynamic environment. When a source revokes a permission, the warehouse should almost immediately reflect the revocation. A message expressed in terms of source tables will be unintelligible to installer scripts at the warehouse DBMS. Even human administrators will react slowly and unreliably when presented with changes in terms of another layer's tables.

Our example views are simpler than the general case, but we believe such simplicity occurs often in real life. Even complex derivations often return some attributes that were simply pulled from source tables. For the remaining cases, we contend that an administrator's assistant need not be complete to be useful. In fact, it may be very helpful for researchers to point out and formalize easy cases, where there can be substantial benefit at little cost.

## 2.2 TOP-DOWN: PROPOSING NEW PERMISSIONS

Suppose the derived layer administrator proposes an increase in her users' permissions. For example, suppose cost analysts need to be able to read the INSURANCE table. The derived layer administrator sends a message to the source layer requesting this change.

The system executes this message as follows. The first step is to compare the desired and existing permissions on the derived tables, so that only the additional (delta) permissions will be requested. There should be less work in judging the delta, and we avoid redundant grants that invite confusion when privileges are revoked. In the example, suppose costAnalyst already has access to PROCEDURE, and requests access to view INSURANCE. When the request is translated to the sources, one would ask for a new Grant only on PATIENT. In systems with column permissions, one might determine that Analyst already had permission to read the first three INSURANCE columns, so the downward translation consists of a request for access only to INSURANCE.Billed_Amount.

It may be useful for the source to grant a wider set of permissions than were requested. For example, suppose the derived layer administrator requests that cost analysts be given access to: Select Procedure_Performed, Bill_Amount From PATIENT Where Year > 1996.

The source administrator determines that this information is suitable for cost analysts to see, and therefore is willing to create a view for it. However, the administrator realizes that other fields are equally releasable (e.g., Doctor, Date), as are pre-1996 records. Furthermore, it is likely that cost analysts will later want to see other fields or selection criteria. Hence, the source administrator chooses to define and grant Read access to the following wider view: Select Procedure_Performed, Bill_Amount, Doctor, Date From PATIENT. Expanding the view contravenes the principle of least privilege, but may reduce the administrator's workload substantially.

In the above scenarios, the source had ultimate control over permissions. Our approach also accommodates scenarios where the security officer works in terms of the view schema. For example, often a conceptual object (e.g., HOSPITAL) is partitioned and denormalized for efficiency, and expressed as a view. The physical tables represent no natural application unit. For administering information security, the HOSPITAL view may be a more natural venue.

## 2.3 COMPARISON: CHECK THE CONSISTENCY OF THE TWO LAYERS

If all layers faithfully imposed the information owners' permissions, the permissions would all be consistent. This is not how systems are administered today. For example, suppose information owners use just the source schema to express permissions, and consider a warehouse that enforces permissions on queries against the (derived) warehouse schema. Auditors may want to check whether all permissions granted on the warehouse can be inferred from the source permissions. If not, there may be a serious security violation. Today, such checks are so troublesome that they are rarely done. With automated comparisons, one could do a nightly consistency check on critical information.

Pointers to Additional Material: A mockup demonstration of elementary negotiation and translation is available on the web pages [7]. It also shows the rules for translating permissions (and some other kinds of metadata) to refer to the other layer's schema, and the management of these translation rules. A paper describing more sophisticated negotiation support is also available there. Frameworks for building and extending such tools are discussed in [5, 6].

# 3. HANDLING ACCESS CONTROL METADATA

The scenarios above illustrated database administrator's requirements in a multi-layer database, especially for negotiations. This section focuses on the model of permissions, and their translation (i.e., inference) across layers. To encourage vendors to move the results into product, we seek simple imple-

mentation but broad applicability (e.g., federations, warehouses, and ordinary views).

## 3.1 PRELIMINARIES

We treat warehouse or federation tables as views, rather than introducing new concepts of similarity, composition, and so forth [Cas97]. Views are subject to the same Grant privileges as views within a single DBMS. Data redundancy is specified by means of integrity constraints. Rather than provide a separate theory for data redundancy, we rely on integrity constraints. Access permissions are expressed as an access predicate associated with each source or view table.

Formally, the expression *Grant(T, p)* denotes that if predicate p is satisfied, one can access table T.[1] A table can have any number of such access permissions with "OR" semantics, e.g., from separate SQL Grant operations. A granted permission can be revoked. We exclude other negative authorizations, because they greatly complicate the semantics of Grant and inference.

We separate a table's permissions into

- Information permissions: Here, the information's owner wishes to impose controls on all routes through which a particular kind of information (e.g., patients' names) can be accessed. From this information policy, one derives permissions to be allowed on each table.

- Physical resource permissions: Permissions that are associated with concrete processing resources (e.g., physical tables, interfaces with executable code). These allow administrators to deal with local requirements for system security, performance, or payment.

To access information from a table, one needs both the information and the physical resource permissions. Administrators working with derived tables will need answers about both, e.g., "Are cost-analysts allowed to read patient names" and "Will some system that possesses patient names allow me to run my queries?" The two properties are likely to be administered separately. Information permissions are discussed in sections 3.1-3.3, and physical resources in 3.4. An administrator who insists on local autonomy can simply refuse to Grant outsiders physical permissions on their resources.

[1] To simplify notation, we omit "subject" and "operation" as explicit arguments. Until the last part of section 3.2, our examples focus on Read.

## 3.2 INFERRING ACCESS PERMISSIONS

Given a (base or derived) table T, the following rules define the inferred permissions associated with T.

```
Direct_perms(T) = OR{ p — Grant(T, p) was asserted }
All_perms(T) = OR{ Direct_perms(T), Inferred_perms(T) }
Inferred_perms(T) = OR{ Query_perms(Qi) — T can be computed
                                          by query Qi }
Query_perms(Q) = AND{ All_perms (Ti) — query Q mentions table Ti }
```

The first two rules are straightforward. The first rule defines direct permissions as the OR of predicates for individual Grants (from section 3.1). The second defines permissions to be the OR of direct assertions or inferred ones.

The last two rules go together, to say that you get access permission on a table T if you have (direct or inferred) permission on all tables mentioned by some query that calculates T. To illustrate, consider a typical case of inferring permissions on a view table V defined by query Q(T1,...,Tk). Then V (by definition) can be computed by Q. Assuming no other query computes V, the rules state that

Inferred_perms(V) = AND{All_perms(T1),...,All_perms(Tk)}.

In other words, anyone who has access permissions on Q can be inferred to have the same permissions on V. Note that if the query processor detects that a stored table is derivable, its derivations may be used.

Our approach also handles update operations on views, whose implementation is expressed as a sequence of source-table operations. In all cases, the view update is replaced by a sequence of database operations (i.e., a procedure) that carries out the desired update. We can easily adapt the inference rules to such situations, by replacing "T can be computed by query Qi" with "T can be implemented by a procedure Pi". It can also be adapted to handle grants of Grant privilege, i.e., Grant (Grant, operation, Table).

Note that the generated operations need not all be the same flavor as the original. For example, a request to delete from a join view V = R1 join R2 join R3 might translate to "Delete from R1; Delete from R2; Read R3 to check integrity, and Abort if violated". The access permission for the view update is the AND of permissions for the two Deletions and the Read.

## 3.3 EXPLOITING REWRITES

The inference rules of Section 3.2 define the allowable access permissions on a table T by referring to the set of all queries that can compute T. But this

set may be neither robust (e.g., new types of integrity constraints will require more powerful inference), nor feasible to enumerate.

We propose to search all rewrites that our software (preferably the query processor) can generate; this is a subset of all queries that might be included in Inferred_Perms. We are content to do better than current systems, even if we reject some requests that are provably legitimate. That is, our approach is sound, but not complete. Pragmatically, vendors are unlikely to invest in sophisticated inference for security alone. Query processors are a critical, large (¿ 100K lines of code) well-funded part of DBMS implementations. If we can adapt their rewrite code, the security subsystem can improve as the query optimizer improves.

This section explores the use of query rewrite in security semantics. Of course, systems are free to implement any subset of these mechanisms.

**3.3.1 View Substitution.** View substitution is the process of replacing a mention of a view in a query by the view's definition. For example, the following query, which mentions the view INSURANCE, can be rewritten to use only source tables. select Insurance_Co, Total_Billed from INSURANCE where Year = 1999

To users who have the necessary permissions on the source tables, the view is simply an alternate interface with no security ramifications. In contrast, the current SQL standard requires that view users obtain explicit grants from the view owner.

We recognize that sometimes executing a view query QV(T1, ...,Tn) adds knowledge, making the result more sensitive than its inputs.[2] We cannot solve the entire aggregation problem, but can protect knowledge embodied in the view definition. This requires no change to our model - one simply protects the view text and (possibly with looser permissions) its executable.

**3.3.2 Constraint-based Simplifications.** If the user queries a derived table V, some source data that underlies V may be irrelevant to the query result. Query processors routinely exploit integrity constraints and relational algebra to rewrite queries in a simpler form. In terms of our inference rules, a query needs only permissions for the data it accesses (using the simplified expression). Such inference can substantially increase the set of view queries that users are allowed to execute.

[2]The other side of the aggregation problem is to prevent two items from being revealed together. Commercial DBMSs offer no protection on this score, nor are they likely to. A user is always free to type in a query that retrieves the information directly from source tables, in one or multiple queries.

The following example illustrates the benefit. Suppose the source database has a foreign key constraint on Patient_Id (i.e., every record in PROCEDURE has a unique corresponding PATIENT). Suppose also that the derived layer contains a table that is the join of the two source tables, as in:

Create view MED_INFO as Select p.*,r.* From PATIENT p, PROCEDURE r Where p.Patient_Id = r.Patient_Id

Suppose a user issues the following query Q1 on the derived table:

Select Patient_Id, Procedure_Performed From MED_INFO

A straightforward evaluation of Q1 requires permissions on both PATIENT and PROCEDURE, or on the entire view MED_INFO. But note that the selected attributes all derive from PROCEDURE, and (by the foreign key constraint) no PROCEDURE records drop out of the join. A query processor can rewrite Q1 to access only PROCEDURE, so permissions on PATIENT are unnecessary.

**3.3.3 Rewrite in terms of other views.** A source layer can use views to provide redundant interfaces to its data. The same information might be available through a source table and through a view, and both may be available for queries. This means that there can be many ways to rewrite a query in equivalent form. In particular, many databases (e.g., warehouses) include materialized views, and query optimizers are beginning speed execution by rewriting queries to use them [8]. These techniques can be used to find additional access permissions.

For a simple example of such query rewrites, consider query Q2 again. In an effort to tighten security, one might revoke external users' permissions for PROCEDURE. Instead, the source administrator could define a view PUB_PROCEDURE containing the non-confidential information from PROCEDURE, and grant access to it. Create view PUB_PROCEDURE as

Select Patient_Id, Doctor, Date From PROCEDURE

This view can then be used to rewrite Q2 as:

Select Insurance_Co
From PATIENT
Where Patient_Id in (select Patient_Id from PUB_PROCEDURE)

This rewritten form is used for permission checking only, not for the execution plan. The system can guarantee that the user will see only information she is allowed to see.

## 3.4 PHYSICAL RESOURCE PERMISSIONS

Thus far, we have been concerned about permissions on information, applicable wherever the information resides. We anticipate that most enterprises will allow physical system owners to further restrict who may use their resources. Motivations include financial (only those who pay), workload management, and security (high security machines that contain copies of public information do not invite the public in).

Previous researchers (e.g., [1, 2]) have proposed models that focus on preserving administrators' autonomy. The resulting model assumes that administrative domains split across system boundaries. But the resulting model adds several concepts not present in SQL (e.g., distinct layers, export), requires still more machinery for redundant data, and does not apply within a single DBMS. We found it more natural to treat each table as a Grant-able physical resource.

We allow administrators to specify *physical resource permissions* on stored tables. If desired, multiple machines might maintain partial copies of the same data, for different user sets. Physical resource permissions determine whether the execution strategy may use a physical resource.

The declaration *GrantPhys(T,p)* specifies that physical resource permission p is enabled on stored table T (a source or a materialized view). The execution strategy must be expressed in terms of stored tables for which the user received such explicit allowances. Inference calculates whether some rewrite can provide sufficient physical resources.

Physical resource permissions are inferred using rules analogous to those of Section 3.2. Thus each table T (stored or view) has two inferred sets of permissions. Its information permissions specify who is allowed to access T; its local resource permissions specify who will be physically able to calculate T, based on enabled access to underlying tables. A user can access a table only if she has both kinds of permissions on it.

# 4. CONCLUSIONS AND OPEN QUESTIONS / RESEARCH ISSUES

## 4.1 WHAT WE HAVE ACCOMPLISHED

Several important classes of systems (federations, warehouses, conceptual views) can be described as "multi-layer databases". Current metadata management tools offer little help in coordinating any but the most basic metadata (e.g., tables and their attributes). As a consequence, metadata is frequently absent or inconsistent, causing security vulnerabilities on one hand, and on the other, inadequate accessibility.

Our scenarios illustrated practical requirements for managing security metadata in such environments. We showed the need for communicating access permissions between different layers in the database. We also illustrated the need for comparison capabilities, both for ordinary administration and for auditing. The scenarios further illustrated extra permissions granted when a view has filtered or summarized information.

We then sketched a technical foundation for coordinating access permissions. The key innovations that helped us simplify the model were to:

- Base the theory on query rewrite, rather than building a more complex theory of permission inference.

- Specify the global information permissions separately from local resource controls. Each has a strong inference rule, and the desired behavior is obtained as their intersection.

A series of examples showed how query rewrites permitted additional access. We then used the same theory to handle non-Read operations, and other granularities.

The theory in the paper is neither startling nor complex. Paradoxically, this is one of the key strengths of the paper. The security administration problem for warehouses and federations is broad and murky. We have identified a problem whose solution can give significant practical benefit, and can be implemented using relatively simple theory.

## 4.2 OPEN RESEARCH PROBLEMS

The multi-layer paradigm opens many questions for security and other metadata research.

First, we need a strategy for dealing with different permission granularities. Some organizations or DBMSs may permit multiple granularities simultaneously, while others will insist on a favored one.

Second, SQL's access controls on views are too inflexible to apply to federations. They require that each view definer be trusted by owners of source tables, and be willing to act as security administrator. And they don't allow the separate administration of local physical resources. As much as possible, we would like to see a federation or data warehousing system as an integrated database. Since SQL dominates relational databases, we want to integrate our approach with SQL security capabilities. At the same time, one ought to integrate controls on where authentication, access control, and execution are trusted to take place.

Third, the essence of our approach is to use some simple principles to propagate security metadata between the layers. We believe that it would be fruitful to apply a similar approach to other forms of metadata. For non-security issues, the chief problems are to devise:

- theory for propagating other kinds of metadata (e.g., data pedigree, credibility, precision, timeliness), through a wide set of query operators (e.g., outerjoin, pivot);
- a component framework that allows administrators to incrementally insert and customize the ever-increasing set of propagation rules [6].

Finally, we have been concerned with semantics, not with algorithmic efficiency or software interfaces. Perhaps the biggest technical systems challenge is to exploit rather than rebuild query rewrite technology, in generating propagation rules. First, query rewrite techniques require that derivations be understandable, e.g., be in SQL and definitely not in Java. Second, we want to be able to submit a query for rewrite but not execution.

## Acknowledgments

Scott Renner has provided many insightful comments throughout this project. Gary Gengo and Tom Lee provided informed close readings of the text.

## References

[1] Castano, S., De Capitani di Vimercati, S. and Fugini, M.G. (1997). Automated derivation of global authorizations for database federations, *Journal of Computer Security*, **5(4)**, pp. 271-301.

[2] De Capitani di Vimercati, S. and Samarati, P. (1997). Authorization specification and enforcement in federated database systems," *Journal of Computer Security*, **5(2)**, pp. 155-188.

[3] Jajodia, S. and Meadows, C. (1995). Inference problems in multilevel secure database management systems. *Information Security: An Integrated Collection of Essays* (eds. M. Abrams *et al.*), IEEE Computer Society Press, pp. 570–584.

[4] Oszu, T. and Valduriez, P. (1998). *Principles of Distributed Database Systems.* Prentice Hall.

[5] Rosenthal, A. and Sciore, E. (1998). Propagating integrity information among interrelated databases. *Proceedings of IFIP 11.6 Workshop on Data Integrity and Control.* http://www.cs.bc.edu/s̃ciore/papers/IFIP98.doc.

[6] Rosenthal, A. and Sciore, E. (1999). First-class views: A key to user-centered computing, *ACM SIGMOD Record.* http://www.cs.bc.edu/sciore/papers/fcviews.doc

[7] Rosenthal, A., Sciore, E. and Gengo, G., (n.d.). *Demonstration of Multi-Layer Metadata Management* (html, gzipped) and *Warehouse Metadata Tools: Something More Is Needed,* http://www.cs.bc.edu/sciore/papers/demo.zip and http://www.cs.bc.edu/sciore/papers/mdtools.doc.

[8] Srivastava, J., *et al.* (1996). Answering queries with aggregation using views. *VLDB*, pp. 318-329.