View Security as the Basis for Data Warehouse Security

Arnon Rosenthal
The MITRE Corporation
Bedford, MA, USA
arnie@mitre.org

Edward Sciore
Boston College and MITRE
Chestnut Hill, MA, USA
sciore@bc.edu

Abstract

Access permissions in a data warehouse are currently managed in a separate world from the sources' policies. The consequences are inconsistencies, slow response to change, and wasted administrative work. We present a different approach, which treats the sources' exported tables and the warehouse as part of the same distributed database. Our main result is a way to control derived products by extending SQL grants rather than creating entirely new mechanisms. We provide a powerful, sound inference theory that derives permissions on warehouse tables (both materialized and virtual), making the system easier to administer and its applications more robust. We also propose a new permission construct suitable for views that filter data from mutually-suspicious parties.

1 Introduction

A key challenge for data warehouse security is how to manage the entire system coherently – from sources and their export tables, to warehouse stored tables (conventional and cubes) and views defined over the warehouse. Permissions on the warehouse must satisfy the restrictions of the data owners, and be updated quickly as those local concerns evolve. Yet the system cannot demand extensive administrator time, since there are too few people with both technical skills to understand derivation logic, and business skills to balance security versus accessibility.

Thus the critical problem is how to *automatically* coordinate the access rights of the warehouse with those of the sources. To do so, one must be able to infer access rights across subsystems, without infringing on their local autonomy. This problem has not been addressed in systems to date. As a result, the warehouse DBA (W-DBA) not only has to manually specify access rights on all warehouse data, the W-DBA must also be trusted by all sources to specify these rights appropriately.

The goal of this paper is to provide a theory that permits automated inference of many permissions for the warehouse, in a way that minimizes both the learning curve for administrators and the amount of new software that vendors would implement. We do not propose a separate theory that requires vendors to implement major new mechanisms, or administrators to learn and execute new tasks. Instead, our theory is a natural extension of the standard SQL grant/revoke model, to systems with redundant and derived data. Nearly all the capability comes from adapting general mechanisms for view security, and by exploiting *available* technology for generating equivalent queries.

1.1 Overview of Our Results

Our theory extends SQL in three ways.

First, we split the notion of "access permission on a table" into two separately-administered issues: who is allowed to access what information (*information permissions*), and who is allowed to access which physical tables (*physical permissions*). An information permission is the result of an enterprise-wide decision, and should be consistently applied to all the views, replicas and derivatives in the system.[1] In contrast, physical permissions are local, and need not be consistent.

For example, the decision that "Employee salary information is releasable to payroll clerks and cost analysts" is an information permission, whereas the decision that "Cost analysts are allowed to run queries on the warehouse" is a physical permission. The effect of separating these concerns is that each set of permissions may evolve independently without

Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2000)
Stockholm, Sweden, June 5-6, 2000
(M. Jeusfeld, H. Shu, M. Staudt, G. Vossen, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-28/

[1] Consistency means having an unambiguous statement for each table or tuple. We *do* support policies that are conditional based on data value.

invalidating the other, and each in isolation has very natural semantics.

Our second extension provides a powerful inference mechanism. In SQL, a user is allowed to execute a query Q if the user has permission on all tables mentioned in Q. We extend SQL so that a user can also execute Q if there is an equivalent query Q' for which the user has permission. That is, releasability depends on the result, not on the computation. We say that Q' is a *witness* for Q. This extension relies on the query rewriting capability of the database system to determine equivalence. It is not necessary for the system to have any particular level of rewrite capability – we can exploit whatever degree of rewriting a vendor can afford to provide. We therefore dare to hope that the results would be practical and attract vendors.

Third, we propose a new construct that broadens the creation of views over mutually suspicious organizations. Conventionally, a source can reduce risk by exporting a view that filters out sensitive data. In SQL, a view can be created (with Grant authority) only if there is a user that has Grant authority on all mentioned tables. We motivate and define an extension that allows the desensitizing view to be over several mutually-suspicious sources, where no person is trusted to have Grant permission over all of them.

Implementing our approach would improve existing systems in several ways. We meet the goals of notable previous proposals [Cap97, Cas97, Jo94], while using fewer new mechanisms and providing greater flexibility. For example, our model allows authority to pass across system boundaries, and supports multiple layers (e.g. of warehouses and data marts) without requiring new constructs. Our model also aids the system administrator, by automatically inferring many *Read* and *Grant-Read* privileges from the warehouse view definition, and by providing rules that determine the allowable queries on data cubes (and rollups). The authority to grant onward is provided by normal view inference of *Grant-Read*. The model also helps with views defined over warehouse tables – it emphasizes layers of views rather than system boundaries.

1.2 Scope Limitations

Warehouse security entails many issues that are not addressed here. We believe that industrial progress is most likely if we design a robust, useful module for security issues of derived data, rather than partial solutions for a broad range of requirements. The derived data facility would be complemented by modules that handle issues such as query rewrite, role management, separation of duty, and inference attacks.

We offload the problem of managing user sets to a role/group management module. Several previous models include authentication and authorization within their models (e.g., [Jo94, Cas97]). However, recent industrial trends tend to pull these capabilities toward a middleware credentials infrastructure, and hence beyond the control of database researchers.

Because warehouses emphasize read operations, we focus on the operations *Read* and *Grant-Read*. As remarked in [Cas97], a similar inference theory works for update, but is outside the scope of this paper.

2 Basic Definitions

A *permission* is a 4-tuple (subject, operation, object, mode). A *subject* is an abstract user, representing an individual, a group, role, process, etc. We say subject *s* has been granted a permission if it is granted either explicitly to *s*, or to an ancestor in a group or role hierarchy. In principle, each subject is uniquely named and is valid in all systems (though explicit grants to individuals may be rare outside their home system).[2]

An *object* (also called a *table*) belongs to some schema, in some system, either a source or a warehouse. It can be a relation or OLAP cube (including, trivially, an individual cell), and can be either materialized or virtual. We refer to a table exported by a source as a *source table.*

Our set of allowable *operations* is the same as in SQL. In this paper we focus on the operations *Read* and *Grant-Read*. If subject *s* is granted a *Read* permission on table *T*, then *s* is allowed to access all of *T*. If *s* is granted a *Grant-Read* permission on *T*, then *s* is allowed to grant a *Read* or *s* permission on *T* to any subject.

The *mode* of a permission is either *information* or *physical*, and will be discussed in Section 3.

We treat each warehouse table as a view (materialized or virtual) over the tables exported by sources or in the warehouse. A *view* is defined by an SQL query, and may include user-defined functions. If *V* is a view, we write its defining query as Q_V. The *inputs* to a query Q are the objects mentioned in it; to show the inputs, we denote the query as $Q(T_1, \ldots, T_n)$. Q can be a black box (either opaque code or secret), but we assume that every query's input set is known.

3 Information and Physical Permissions

The mode of a permission specifies that it is either an *information permission* or a *physical permission.* A subject *s* is allowed to access a table only if it has both information and physical permissions on that table.

[2] This paper addresses what permissions should *exist*. Partitioning by physical system is permitted in the physical implementation, which we do not address.

A permission *(s, op, T, "information")* indicates that the *information* in *T* should be accessible to *s*, for operation *op*. It concerns releasability of knowledge, not access to the physical container *T*. Information policies apply globally, spanning source systems and warehouse tables. They are unaffected by creation of redundant copies or new interfaces – the policy on a table would be unaffected by a decision to create a mirror table or an extract at another site.

For example, a medical system may contain several information policy assertions concerning when TreatmentCost-values can be releasable to certain subjects. One assertion might be "to auditors". Others might be: "In New York, to the role BillingClerk"; and "in Florida, to the role InsuranceAdjuster". These assertions correspond to information permissions, and refer to all copies of the information.

A *physical permission* authorizes an execution strategy to use a single physical resource (usually a materialized table). Whereas information permissions are global, physical permissions allow local autonomy. By withholding physical permissions, a resource owner can limit workload, allow only users who have paid a fee, or allow only corporate employees. We suspect that physical permissions will typically be administered in coarse granules, e.g., to give a user physical access to all tables on a particular platform. It seems best to administer in terms of database schemas, so as to allow finer granularity (e.g., to exclude blobs), and to use groups and roles defined in the DBMS.

Our long-term vision is to allow a security administrator to see a giant distributed database that spans many platforms. This database would be managed using Grant/Revoke, with implied permissions as described below. (Grant and Revoke would be extended by a parameter that distinguishes <information | physical> mode).

It will be difficult technically and politically to extend DBMS capabilities to cope with new modes, and to notify each other of the need to Grant and Revoke derived permissions. We suspect progress will be faster if security management is part of a systems manager (e.g., Tivoli) that communicates with each relevant DBMS. Permission changes would be replicated to the metadata manager, and it would replicate derived changes to the other systems.

In such an architecture, granting an ordinary SQL permission in a DBMS would create both information and physical permissions in the metadata manager. The metadata manager would determine derived permissions. The SQL permission *(s, operation, object)* would be granted on a derived table (typically in the warehouse) if and only if the subject has both permissions *(s, operation, object, "information")* and *(s, operation, object, "physical")*. Each Grant or Revoke would (conceptually) cause all the permissions to be recomputed. It will be challenging to implement these semantics efficiently and reliably among distributed heterogeneous DBMSs. The alternative is to implement them by human administrators.

4 Permission Inference

A permission is *explicit* if it is granted directly by an authorized grantor. When the information for a table *T* is derivable as $Q(T_1, \dots T_n)$, we may be able to infer additional permissions. This section presents a theory for determining the implied permissions in effect at a particular schema (typically the warehouse).

We want the rationale for permission inference to be understandable (even by nontechnical managers) and to be easy to automate. In earlier papers we relied on inference rules, whose justification required considerable argument. In this section we propose a simpler "witness" semantics; inference rules are implementation aids derived to track witnesses.

4.1 A Theory of Witnesses

Informally, a subject should have permission to execute a query if and only if the query can be expressed in terms of tables (base or view) for which the user has explicit permission.

Formally, let *T* be a table (base or view). We have the following definitions:

- A query *Q* is *equivalent* to *T* if the output of *Q* always contains the same tuples as *T*.

- A permission *(s, op, T, mode)* is *implied* if there exists an equivalent query $Q(T_1, \dots, T_n)$ such that each permission $(s, op, T_i, mode)$ has been granted *explicitly*. Query *Q* is called the *witness* for the implied permission.

The inference mechanism for permissions is the same for both physical and information permissions. However, the interpretation of the implied permission is different, depending on the mode.

An implied information permission *(s, read, T)* means that the information in *T* is releasable to *s*. In effect, a subject need not care whether an information permission is explicit or implied, nor whether *T* is a materialized view, a view computed over several sources, or a base table.

An implied *physical* permission on *T* asserts that there exists at least one way to compute *T* for which the physical permissions are available. If *T* were a materialized table, however, the subject would *not* have physical access to *T* – the subject (or the database query processor) would still need to use the tables from the witness query.

4.2 Query Rewrite Techniques

Our definition of implied permission requires that the system be able to find a witness query equivalent to *T*. The theory of query equivalence is mature. There have been numerous research articles, and it is heavily exploited in query optimizers. Exploring the details would be a distraction from the security issues, and from creating a derived data security module. Instead, we explore the benefits of exploiting three particular rewriting strategies. A larger set of motivating examples and a discussion of the issues involved in using the query processor's rewriter appear in [Ro99b].

- *View Substitution:* View substitution replaces a mention of a view in a query by the view's definition. We can infer that if a subject *s* has the necessary permissions on the source tables, then *s* also has permission on the view. In this case, the view is simply an alternate interface to the same data. (In contrast, the current SQL standard requires that all view users obtain explicit grants from the view owner.)

- *Semantic Query Optimization:* If the user queries a view *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. For example, suppose *V* is the join of two tables; then typically a query on *V* is possible only if the user has permission on both tables. However, if the join is on the key of one table and referential integrity holds, then a query that accesses only fields from the foreign-key table need not perform the join. Thus, the subject issuing the query would not need permission on *V*, but just the one table.

- *Rewrite in terms of other views*: Subjects are often given access to information through views when they do not have permissions on the base tables. A query mentioning base tables may therefore be equivalent to one mentioning only views. In such a case, the system can infer that the subject issuing the query has permission to execute it.

Our model does not require any particular degree of query rewrite, nor do we propose algorithms in this paper. Below, we elaborate on two issues: why push rewrite to another module, and why accept incomplete enumeration of rewrites.

First, DBMS vendors have historically given metadata management a low priority. If the purpose of query rewrite is to improve metadata (e.g., access permissions), they are unlikely to spend the substantial sums needed to do it well (e.g., to exploit constraints, to extend it to new operators, or to employ materialized views). However, they *have* been willing to spend the money in pursuit of performance gains. Therefore if we want to get the most from query equivalence, we need to hook into the query processing module.

Second, our benchmark is to do better than others have, and to allow only justifiable permissions. Given that current systems perform no inference, even the implementation of the simple view substitution strategy would be a significant improvement. The more strategies implemented, the more powerful the system. However, "completeness" is a minor goal. In fact, a complete set of equivalents is impossible due to incomplete declarations of constraints and operator rewrite rules, and because the general rewrite problem is undecidable. Moreover, even a complete set of equivalents may give suboptimal permissions; only a health care expert would know that "Patients where Age>21" deserves more permissions than Patients.

4.3 Administering a Warehouse

Our witness theory specifies which information permissions are to be automatically inferred and declared on warehouse tables. This section examines the consequent benefits for warehouse administration that arise from it.

4.3.1 Modes of Administration

Standard SQL infers view privileges only when a view is defined – the view definer receives the intersection of her privileges on the input tables. However, the view definer must then explicitly specify all other permissions on the view. Our model infers information permissions for other users, so the W-DBA need not do so. Beyond this, normal "grant option" semantics apply. That is, if a source administrator owns all the tables that underlie a warehouse view, she may administer the view's permissions directly, grant "grant option" to a warehouse administrator, or combine the modes.

[Cap97, Cas97] propose a rich security model for tightly coupled federated databases. Many of the facilities apply also to warehouses. Derived data security entwines with models of data integration, entity and object data models, and some aspects of group management with negative permissions. They require a new sort of specification, to tell whether federation administrators can delete or supplement permissions inferred from the sources. They also proposed features in areas that we do not address, such as top-down administration, and determining where authorizations may be checked.

These papers motivated many of our requirements, but the complexity seems likely to deter major commercial implementation. In areas where the models overlap, ours seems to have fewer new constructs. Yet we can simulate the modes of local autonomy proposed in [Cap97, Cas97], and additional ones besides.

One such mode has source administrators make *all* decisions; Read or Update permissions are then inferred to the view. This resembles our inference (and includes Update). However, it does not appear to infer privileges to views defined over the federation/warehouse base tables, and does not allow an upper-tier view (e.g., a statistical rollup) to receive additional permissions from the underlying source administrator(s). Our model allows all this. The W-DBA gives physical permission in advance to All-Employees, but receives no Grant-Read privileges, and hence does nothing more. Going beyond the previous work, in our model, an administrator *s* who possesses Grant-Read information permissions on all of $T_1, \ldots, T_n$, possesses it on $V(T_1, \ldots, T_n)$. Even though *T* is in the warehouse and *s* is associated with a source system, she automatically has the right to grant additional information permissions on *T*.

Another proposed mode is to have joint signoff, where the W-DBA must approve each permission on a warehouse table. This can be achieved in our model by having the W-DBA grant the physical permissions selectively. Finally, there is a mode where the W-DBA actually grants information permissions; our model handles this by asking each source administrator to grant Grant-Read on exported tables to the W-DBA.

4.3.2 Deriving OLAP Permission Rules

Witness theory works to guide the proper access permissions for OLAP [Pr00]. Consider a cube having several hierarchical dimensions. Suppose the source declares that subject has permissions on a view defined by some rollup level on each dimension. Then in our model, a query is permissible only when a witness can be found, i.e., if it can be derived from views at the permitted rollup level. It will not be permissible to access the warehouse data at a finer granularity than the rollups, even if the filtered data is later rolled up.

This is in contrast with some warehouse systems [UPa00], which allow the query to execute, but surreptitiously replace the user's query by a query to a subcube for which the user has access. We consider this unsatisfactory, too likely to lead to bad decisions. It would be far preferable to *suggest* to the user one or more queries for which they do have sufficient access.

4.4 Computing Local Permissions

When a query is issued to a warehouse, one wants to test permissions there, rather than refer the issue to the source databases. That means that the system needs to populate permission tables on the warehouse.[3] Our theory of witnesses allows us to determine whether a subject is entitled to access a table *T*. However, the warehouse needs to know the set of all subjects authorized to access *T*, i.e., the set of users for which witnesses can be found. This section shows how it is possible to compute that set.

The basic idea is that the set of users authorized to execute a query (not considering equivalence) is the intersection of the user sets of its inputs. The user set of a view can thus be determined by taking the union of these individual user sets for each equivalent query found.

Given *T*, the system first computes all queries equivalent to *T*, structured as a directed graph in which each subexpression appears only once. One can traverse the graph, computing permissions for each table in that graph from the permissions on its predecessors. If the graph is acyclic, this can be done once, bottom up.

It is possible to have cycles in derivability (e.g., a whole table from horizontal or vertical partitions). We have developed algorithms that appear to handle the general case with similar order of complexity, but a larger constant factor. The general case can also be handled by translating the above recursions into Datalog (one clause per view definition), and using semi-Naïve evaluation [Ul89].

5 Within-View Permissions

A warehouse's derivation is likely to draw from many parts of an organization, or even from multiple organizations. For example, a drug-effects warehouse might draw information from many hospitals. With the current SQL model, each organization must grant a warehouse administrator *Read* (and *Grant-Read*) permissions on their exported data – only then can someone define and administer a warehouse table view over the combined information. But what happens if no person enjoys such universal trust? And how would 500 hospitals or 50000 doctors' offices negotiate to choose such a paragon?

Our approach is to exploit *multi-table*[4] views that combine information in a way that makes it less sensitive. A source can stipulate that its exported data can be used only for computing the view. The SQL

[3] To maximize rewrite possibilities, the warehouse could also consider rewrites in terms of source tables, while using its own copy of source permissions.

[4] If V is computed only from table T, the definition is uninteresting -- one could just grant select to s on V.

syntax from the administrator of *one* of the view's input tables might become:

grant select to s on T *within V*

That is, subject *s* can access *T*, but only from within view *V*.

This model still requires trust that the warehouse computes the view correctly, and makes no other use of the information. One must also trust the other sites not to subvert the filtering effects of the view, as illustrated in Example 1. But it will require malice and skill rather than carelessness to subvert the software.

5.1 Examples

We give three examples where the warehouse computes a view that is less sensitive, in a way that could not be achieved by single-source views.

Example 1 – Totals over a large set

The warehouse supports financial studies of hospitals, by providing COST data about hospital patients, in a cube with dimensions Treatment, Age, and State (i.e., where each entry is a statewide average). Some hospitals might be willing to release data only if it is hidden within Statewide totals. The warehouse defines a view V to compute the cube, and each hospital issues:

grant read on table MyHospital.COST within V

Elaborating this example, the warehouse might contain other tables with more detailed information, from hospitals that are willing to provide it. Permissions on that more detailed information allows that information to be used for computing *V*. Hospitals in New York City might feel there is sufficient anonymity in being included in citywide totals; they could define *V'* that totals by city. *V* is derivable from *V'*. A few hospitals that particularly trust the researchers, or whose cost information is public, may grant access to the raw COST data.

In this example, the hospitals trust that the warehouse software will compute *V* (or *V'*) and discard the raw inputs. They also trust that the other hospitals will not spoof or reveal information in ways that enable individual hospital results to be detected.[5]

Example 2 – Peer-to-Peer Intersection

Consider tables that track people entering the country on international flights (denoted ENTRANT), and people who are wanted by the police (WANTED). There is a procedure "match" which examines rows in the two tables and decides whether the entering person fits the description of a person who is being sought.

Each airport's authorities want access to view *V*, defined by:

```
select * from ENTRANT, WANTED
where match(ENTRANT, WANTED)
and ENTRANT.Airport = $MY_Airport
```

The border authorities should not have access to all arrest warrants, and the police searchers should not know about all citizens' travels. So neither will give the other's representative Read authority on their entire table. However, they both are willing to give read access within V to the head agent at each airport.

Example 3 – Intersection of child and parent

Suppose that a hospital has two tables:

PATIENT[P#, Age,…] and
SURGERY[P#, ProcedureDone, Date, …]

PATIENT is the parent table (every surgery has a valid patient), but most patients have no surgery records.

A researcher *s* studying surgery on the old might create the view

```
select * from PATIENT, SURGERY
where PATIENT.Age > 80
and PATIENT.P# = SURGERY.P#
```

In current systems, *s* would need access on both tables in order to create (and access) the view. In our model, it is only necessary for the administrator of each table to give u permission on the table within the view.

This example can also be used to illustrate the benefits of automatic inference. Suppose a research user *s* with access to PATIENT and SURGERY periodically runs a reporting application that executes the above view. Now suppose new regulations are enacted that forbid u from seeing children's data, i.e., *s* is given access only to the view *V'*, defined by:

```
select * From PATIENT where Age>21
```

In current systems, the reporting application would fail. With query rewrite, the query could instead execute over the (non-materialized) view *V'*. Details appear in [Ros99b].

5.2 Semantics for Within-View Permissions

Intuitively, a subject *s* is able to access view *V* if *s* has access to each input table, *within V*. Formally, we have:

- A *within-view permission* is a 5-tuple *(subject, operation, object, mode, view).*

The meaning of within-view permission *P = (s, read, T, m, V)* is as follows. If *P* is an information permission, then *s* has clearance to view the values of *T* that contribute to *V*. If *P* is a physical permission, then *s* is

[5] Confidentiality would be lost if all but one hospital in a state filled its COST table with \$0 for each entry. would presumably have serious operational effects. Another attack, also unlikely, would be for all the others to reveal their true costs.

allowed to execute a query that physically accesses *T*, but only for the purpose of computing *V*.

One can straightforwardly extend the witness semantics of Section 4.1 as follows.

A permission *(s, op, T, mode)* is an *implied permission* if there exists a query *Q* and views $\{V_i\}$ equivalent to *T*, such that for each object T_i mentioned in *Q*, either

- the permission $(s, op, T_i, mode)$ has been explicitly granted, or
- the within-view permission $(s, op, T_i, mode, V_i)$ has been explicitly granted, where *Q* is equivalent to V_i.

Query *Q* is called the *witness* for the implied permission.

This definition allows us to model each of the examples in Section 3.1.

A consequence of the above definition is that a user who receives within-view permissions on the inputs of view *V* gets (by inference) the permission on *V*. Thus, users are able to access a view even when nobody is trusted to grant on all underlying tables. A user can even be a view administrator with the ability to grant access to *V*, if the within-view permissions are for the *Grant-Read* operation.

We have studied examples that require more complex within-view permissions, in which multiple witnesses need to be coordinated. However, the extra complexity of the permission mode makes them less useful in practical situations, and we therefore relegate their development to a future paper.

6 Summary and Future Work

6.1 Contributions

The greatest strengths of our approach are simplicity, architecture and location independence, and compatibility with relational technology. Our strongest result is the discovery that some pleasantly simple concepts – separation of physical permissions, and the "witness"-based theory for permissions on derived data – can turn a difficult, messy problem into one that can be addressed with few technical complications. We obtained a consistent, defensible theory for inferring new permissions from those which were explicitly asserted. At the same time, we built upon rather than replaced existing relational technology.

More specifically, our approach provides the following benefits:

- We can determine whether permissions on sources' exported tables justify granting a user access to the warehouse table. The same mechanism is helpful for views defined above the warehouse base tables.
- The theory of witnesses for individual users underpins a recursion that can efficiently determine the permissions to be granted on each stored warehouse table. When source permissions change, the inference can be redone, completely automatically. An efficient "delta" inference process appears feasible for Grants.
- There is no need to customize the security inference rules to handle OLAP or extract/transform/load operators. Instead, one can use algebraic and constraint-based rewrite rules provided by the query optimizer.
- We showed that local autonomy can be preserved (using physical permissions) without sacrificing the inference of information permissions. Capabilities proposed in other research [Cap97, Cas97] were subsumed.
- "Within-view" permissions enable additional collaboration among mutually suspicious organizations.

6.2 Open Problems

This section describes several open problems in easing the administration of warehouse access permissions. First we describe pragmatic issues, and then theoretical ones.

Perhaps the biggest pragmatic issue is creation of a tool based on the theory, so user experience could be observed. An html mockup of an environment for managing metadata across layers in a multi-tier system is available at [Ro99a]. The mockup was useful in identifying requirements, and holes in the theory. But we lack the resources (especially grad student labor) to create and test a real tool. Also, our tool explored only the semantics. It will be necessary to embed it in a systems management environment that coordinates permissions with each relevant DBMS.

A second pragmatic issue is how to produce a query rewriter. It is infeasible to keep up with new query operators and constraint constructs. Fortunately, DBMSs already have a component that does such inference rather well, and in which big money is invested – the query optimizer. The key issue, partly technical and partly political, is whether query processors will export their rewrite functionality.[6]

[6] In fact, warehouse query processors increasingly rewrite queries to exploit materialized views. However, some modifications will be needed, to avoid exploring multiple witnesses over the same input set (e.g., for different join orders), and to avoid cost-based pruning.

Further inference capabilities would be available if the extract/transform/load operations were describable as SQL expressions (with new SQL operators such as *coalesce columns*, and user-defined functions). Currently, the E/T/L views tend to be described in code, though some products (such as the Microsoft Repository [Be99]) will identify the input objects.

Next, we believe that sources and warehouse users need to negotiate permissions in terms of roles they both understand. Without this, inference is of limited use. Role and group management is a difficult task. We hope to see rapid progress in this area, perhaps driven by task and workflow models. One also needs to see if knowledge of data derivation can help in defining such roles.

Finally, we need to understand the benefits of implementing each feature. For example, how many users would want views that filter across several sources, or cycles in view derivability? Which kinds of rewrites are most effective in giving users additional justified permissions?

Several theoretical problems remain open. We need efficient algorithms to infer permissions. We want to get a calculus of access predicates, to compute in advance what warehouse permissions should be granted. (Currently, for each request, one seeks a witness). Also, [Ro00] proposed to attach limitations to grant-option permissions (e.g., to allow onward grants only within the recipient's department). Such limitation predicates deserve to be added to the inference theory. Next, to extend the theory to federations and ordinary views, one needs to handle permissions for update operations, and user-defined operations (e.g., Hire-Employee).

Finally, research is needed to support permission requests that originate at the top (as in "make my view have the following inferred permissions") [Cas97]. If permission inference is seen as defining a view, top-down asks to update the view. As with conventional view updates, there are often multiple ways to meet the user's intent. Generation of those alternatives needs to be guided by a theory, and automated by tools.

6.3 Conclusion

The proposed facilities can provide significant benefits for administering distributed databases and views. At the same time, we minimized complexity for DBMS implementers, by building over rather than replacing SQL facilities.

We contend that a "derived data access permissions" module deserves implementation today with RDBMSs, especially distributed RDBMSs. For the short term, it would seem appropriate to distinguish physical and information permissions, and to grant access based on whatever rewrites are easy to implement. More sophisticated rewrites seem quite useful to employ predefined views or eliminate unneeded tables, but will be costly to implement.

Finally, a message to builders of query processors: Make the query rewrite facilities available for other purposes. Rewrites are invaluable at inferring access permissions and other metadata, and yet are too costly to implement just for metadata purposes.

7 References

[Be99] P. Bernstein, et. al., "Microsoft Repository Version 2 and the Open Information Model," *Information Systems 24(2),* 1999.

[Cap97] S. De Capitani di Vimercati, P. Samarati, Authorization Specification and Enforcement in Federated Database Systems, *Journal of Computer Security*, vol. 5, n. 2, 1997, pp. 155-188.

[Cas97] S. Castano, S. De Capitani di Vimercati, M.G. Fugini, Automated Derivation of Global Authorizations for Database Federations, *Journal of Computer Security,* vol. 5, n. 4, 1997, pp. 271-301.

[Jo94] D. Jonscher, K. Dittrich, "An Approach for Building Secure Database Federations", *VLDB,* Santiago, Chile, 1994.

[Pr00] T. Priebe, G, Pernul, "Data Warehouse Security in GOAL", submitted for publication.

[Ro99a] A. Rosenthal, E. Sciore, G. Gengo, "Demonstration of Multi-Tier Metadata Management", at www.mitre.org/resources/centers/it/staffpages/arnie/

[Ro99b] A. Rosenthal, E. Sciore, V. Doshi, "Security Administration for Federations, Warehouses, and other Derived Data", *IFIP 11.3 Working Conference on Database Security,* Seattle 1999. (Kluwer, 2000).

[Ro00] A. Rosenthal, E. Sciore, "Extending SQL's Grant and Revoke Operations", *IFIP Workshop on Database Security,* Amsterdam, August 2000

[Ul89] J. Ullman, *Principles of Database and Knowledge-Base Systems, Vol. 2,* Computer Science Press, Rockville MD, 1989.

[UPa00] Instructions from U. Pennsylvania data warehouse
http://www.upenn.edu/computing/da/dw/security.html