An Access Authorization Model for Relational Databases Based on Algebraic Manipulation of View Definitions

Amihai Motro

Computer Science Department
University of Southern California
University Park, Los Angeles, CA 90089-0782

Abstract

We describe a new model of access authorization for relational databases. In this model access permissions are a form of database *knowledge*, from which access permissions that apply to specific requests are *inferred*. The basic principles of this model are: (1) Database access is specified in terms of views: a set of views is defined, and each user is granted permission to access one or more views. (2) Users direct queries at the actual database, not at any particular view. (3) When a request to access a view is presented to the database system, the system derives views of the request that are views of the views to which the user has access permission, and presents the user only with these views. The model represents the definitions of views in special "meta-relations", and extends standard algebraic operators to these relations. When a request is presented to the database system, it is performed *both* on the meta-relations, resulting in a "mask", and on the actual relations, resulting in an answer. The mask is then applied to the answer, yielding the data that may be delivered to the user. This answer is accompanied by statements describing the portions delivered.

1 Introduction

A *view* (or a *virtual relation*) is a relation that is defined in terms of the actual database relations. Most access authorization models for relational databases define access permissions by means of pairs: (*user*, *view*). Each such pair grants *user* permission to access *view*. When a query is submitted to the database system, the permissions table is consulted to determine whether to authorize the retrieval. Individual models differ substantially in their details. Two well-documented models are discussed below.

This work was supported in part by NSF Grant No. IRI-8609912 and by an Amoco Foundation Engineering Faculty Grant.

The authorization scheme of System R [1] allows permissions to be granted to both actual relations and views. It is a flexible scheme that incorporates useful features, such as allowing users to grant other users permissions that were granted to them. In our opinion, however, the mechanism has a serious limitation. Consider the situation of two actual relations A and B, and assume that we want to permit access to a particular view of A and B. We define this view V and grant access permission to V, but not to A or B. Authorization is now granted only to queries that access V. Queries that accesses A or B, will be rejected for lack of access permissions to these relations, even if the requests are *within the permissions* (as described in view V). Thus, V is not only a *statement* of the permissions, but the actual "access window", as well.

The authorization scheme of INGRES [5,2] is quite different. Queries are handled by a "query modification" algorithm. Essentially, the algorithm searches for permitted views whose attributes *contain* the attributes addressed by the query, and the qualifications placed on these attributes in the views are then *conjoined* with the qualification specified in the query. The algorithm is attractive because when a query exceeds its permissions, it delivers the data that are within the permissions. However, there are several limitations. First, permissions are granted only for actual relations or views of single relations, and it is not possible to grant permissions to views of several relations. Second, the algorithm does not handle rows and columns symmetrically. Consider relation A with attributes A_1, A_2 and A_3, and assume permission is granted to the tuples of A_1 and A_2 that satisfy a predicate P. A request to retrieve A_1 and A_2 would be reduced to the tuples of A_1 and A_2 that satisfy P. However, a request to retrieve A_1, A_2 and A_3 would be denied altogether, where one would expect that it would be reduced to tuples of A_1 and A_2. In addition, there are various cases where the algorithm actually delivers less than what the user is permitted to view.

In this paper we describe a new model of access authorization for relational databases. This model avoids the restrictions and limitations described above. In particular, views are not access windows, but statements of permissions (indeed, a form of database *knowledge*), from which access permissions that apply to individual access requests are *inferred*.

The basic principles of this model are:

- Database access is specified in terms of views: a set of views is defined, and each user is granted permission to access one or more views. Views are conjunctive relational calculus expressions involving any number of relations.
- Users direct queries at the actual database, not at any particular view.
- When a request to access a view is presented to the database system, the system derives views of the request that are views of the views to which the user has access permission, and presents the user only with these views.

In our approach, determining the access permissions for a given query is a specific case of the following more general problem: Given a query and set of database views that possess a particular property, what views of the answer possess this property?

In [4] we considered database views whose property is that they have guaranteed *integrity*. The problem then became: Given a query, what views of its answer have guaranteed integrity? Solving this problem enabled us to extend a database system so that every answer is accompanied by statements that define its integrity, resembling a *certification of quality*.

In this paper we consider database views whose property is that they are *permitted* to a particular user. The problem then becomes: Given a query, what views of its answer should be permitted to this user?

In each case, our solution is to represent the definitions of the given database views in special relations, using the concept of *meta-tuples*. A meta-tuple defines a subview (i.e., a selection and a projection) of a single relation, and several meta-tuples can be used together to define general views. All meta-tuples that define subviews of the same relation are stored together in one *meta-relation*, whose structure mirrors the actual relation. Standard algebraic operators (product, selection and projection) are extended to these meta-relations.

When a query is presented to the database system, it is performed *both* on the actual relations, resulting in an answer, and on the meta-relations, resulting in definitions of views of the answer that inherit the particular property of the given views.

In the case of access permissions, this property is that the views should be permitted to the user. Hence, they are taken as a "mask" that is applied to the answer, yielding the data that may be delivered to the user. This data is accompanied by statements describing the portions delivered.

We note that this paper focuses on the method for expressing and storing views, and on the method by which permissions on individual access requests are inferred. It does not address other important aspects of authorization.

The remainder of this paper is organized as follows. Section 2 defines the language with which access permissions are expressed, and Section 3 describes how access permissions are stored. Section 4 defines the algebraic manipulations of access permissions, and Section 5 demonstrates how these manipulations are employed to yield permission masks for individual access requests. Section 6 concludes with a brief discussion of further refinements of the model. This representation and manipulation of views was first introduced in [4], and has been modified for the particular problem at hand.

2 Expressing Access Permissions

We assume the following definition of a relational database [3]. A *relation scheme* R is a finite set of *attributes* $A_1, \ldots, A_m$. With each attribute A_i a set of values D_i, called the *domain* of A_i, is associated (domains are nonempty, finite or countably infinite sets). A *relation* on the relation scheme R is a subset of the product of the domains associated with the attributes of R. A *database scheme* $\mathcal{R}$ is a set of relation schemes $R_1, \ldots, R_n$. A database instance $\mathbf{D}$ of the database scheme $\mathcal{R}$ is a set of relations $R_1(\mathbf{D}), \ldots, R_n(\mathbf{D})$, where each $R_i(\mathbf{D})$ is a relation on the relation scheme R_i.

A *view* V is an expression in the relation schemes of $\mathcal{R}$ that defines a new relation scheme, and for each database instance $\mathbf{D}$ defines a unique relation on this scheme denoted $V(\mathbf{D})$. The concept of views plays a fundamental role in most access schemes for relational databases. Queries are simply requests to access particular views, and access permissions are usually specified in terms of views. In this paper we consider views that are defined by *conjunctive relational calculus expressions* [6]. Using domain relational calculus, expressions from this family have the form:

$$\{a_1, \ldots, a_n \mid (\exists b_1) \ldots (\exists b_k)\, \psi_1 \wedge \ldots \wedge \psi_m\}$$

Where the ψ's may be of two kinds:

1. **membership:** $(c_1, \ldots, c_p) \in R$, where R is a database relation (of arity p), and the c's are either a's or b's or constants.
2. **comparative:** $d_1\, \theta\, d_2$, where d_1 is either an a or

a b, d_2 is either an a or a b or a constant, and θ is a comparator (e.g., $<, \leq, >, \geq, =, \neq$).

In particular, each a and each b must appear at least once among the c's.

We shall refer to such views as *conjunctive views.* While this family is a strict subset of the relational calculus, it is a powerful subset. The family of conjunctive relational calculus expressions is precisely the family of relational algebra expressions with the operations *product*, *selection* and *projection* (where the selection expressions are conjunctive). The attributes that participate in the selection predicate will be called *selection attributes*, and the attributes that participate in the projection will be called *projection attributes.* Often, we shall define views that include the selection attributes in the projection attributes. The advantage of such views will be evident later.

As an example, consider a database with the following relation schemes:

$$\begin{aligned}\text{EMPLOYEE} &= (\text{NAME}, \text{TITLE}, \text{SALARY})\\ \text{PROJECT} &= (\text{NUMBER}, \text{SPONSOR}, \text{BUDGET})\\ \text{ASSIGNMENT} &= (\text{E\_NAME}, \text{P\_NO})\end{aligned}$$

The view SAE (salary of all employees) presents the names and salaries of all employees:

$$\text{SAE} = \{a_1, a_2 \mid (\exists b_1)(a_1, b_1, a_2) \in \text{EMPLOYEE}\}$$

The view PSA (projects sponsored by Acme) presents all the attributes of projects sponsored by Acme:

$$\begin{aligned}\text{PSA} = \{&a_1, a_2, a_3 \mid (a_1, a_2, a_3) \in \text{PROJECT} \wedge\\ &a_2 = \text{Acme}\}\end{aligned}$$

The view ELP (employees of large projects) presents the names and titles of employees assigned to projects with budgets of at least \$250,000:

$$\begin{aligned}\text{ELP} = \{&a_1, a_2, a_3, a_4 \mid (\exists b_1)(\exists b_2)\\ &(a_1, a_2, b_1) \in \text{EMPLOYEE} \wedge\\ &(a_3, b_2, a_4) \in \text{PROJECT} \wedge\\ &(a_1, a_3) \in \text{ASSIGNMENT} \wedge\\ &a_4 \geq 250,000\}\end{aligned}$$

Note that this view is defined to include, in addition to the names and the titles, also the project numbers and the budgets. Finally, the view EST (employees with same title) presents pairs of names of employees that have the same title, along with that title:

$$\begin{aligned}\text{EST} = \{&a_1, a_2, a_3 \mid (\exists b_1)(\exists b_2)\\ &(a_1, a_2, b_1) \in \text{EMPLOYEE} \wedge\\ &(a_3, a_2, b_2) \in \text{EMPLOYEE}\}\end{aligned}$$

For clarity, definitions of views may be specified with an equivalent **view** statement, as in the following example that defines the view ELP:

view ELP (EMPLOYEE.NAME, EMPLOYEE.TITLE,
  PROJECT.NUMBER, PROJECT.BUDGET)
where EMPLOYEE.NAME = ASSIGNMENT.E_NAME
  and PROJECT.NUMBER = ASSIGNMENT.P_NO
  and PROJECT.BUDGET $\geq$ 250,000

The next example, which defines the view EST, shows how to handle cases where several membership subformulas reference the same relation:

view EST (EMPLOYEE:1.NAME, EMPLOYEE:2.NAME,
  EMPLOYEE:1.TITLE)
where EMPLOYEE:1.TITLE = EMPLOYEE:2.TITLE

As mentioned earlier, views are used both in queries and in the specification of access permissions. Permissions are granted with a **permit** statement, as in the following example to grant Klein permission to access the view EST:

permit EST **to** KLEIN

Queries are expressed with a **retrieve** statement, as in the following query for the names and titles of employees assigned to projects sponsored by Acme:

retrieve (EMPLOYEE.NAME, EMPLOYEE.TITLE)
where EMPLOYEE.NAME = ASSIGNMENT.E_NAME
  and ASSIGNMENT.P_NO = PROJECT.NUMBER
  and PROJECT.SPONSOR = Acme

Finally, given a view, we define every view derived from it by a selection and a projection, as its *subview.* In particular, every view is its own subview.

3 Storing Access Permissions

Access permissions are stored in new relations that are added to the database. For each database relation R a *meta-relation* R' is added. The scheme of R' is similar to the scheme of R, but with an additional attribute called VIEW. Also, two auxiliary relations are defined: COMPARISON = (VIEW,X, COMPARE, Y) and PERMISSION = (USER,VIEW). In the previous example, the database is extended with the following relations:

$$\begin{aligned}\text{EMPLOYEE}' &= (\text{VIEW}, \text{NAME}, \text{TITLE}, \text{SALARY})\\ \text{PROJECT}' &= (\text{VIEW}, \text{NUMBER}, \text{SPONSOR}, \text{BUDGET})\\ \text{ASSIGNMENT}' &= (\text{VIEW}, \text{E\_NAME}, \text{P\_NO})\\ \text{COMPARISON} &= (\text{VIEW}, \text{X}, \text{COMPARE}, \text{Y})\\ \text{PERMISSION} &= (\text{USER}, \text{VIEW})\end{aligned}$$

Relations EMPLOYEE$'$, PROJECT$'$ and ASSIGNMENT$'$ will be used to store membership subformulas of views. Comparative subformulas will be stored in relation **COMPARISON**. Relation **PERMISSION will be used to associate users** with views.

Consider a view V,

$$V = \{a_1, \ldots, a_n \mid (\exists b_1) \ldots (\exists b_m)\, \psi_1 \wedge \ldots \wedge \psi_k\}$$

A subformula ψ of the kind $(c_1, \ldots, c_p) \in R$ is first modified so that the c's that are a's are suffixed with $*$, and the c's that are variables (i.e., a's or b's) that appear only once in the whole expression are replaced with $\sqcup$ (blank). Hence, each component of the modified subformula is either a constant (a value), or a variable, or a blank, and each may be suffixed by $*$. This *meta-tuple* is prefixed with V (the name of the view), and is stored in the meta-relation R'. A subformula ψ of the kind $d_1\, \theta\, d_2$, where θ is not $=$, is transformed to the tuple (V, d_1, θ, d_2) and is stored in the auxiliary relation COMPARISON. If θ is $=$, then all occurrences of d_1 in the other subformulas are substituted with d_2. Finally, for each user that is granted permission to access this view, a tuple is inserted into the auxiliary relation PERMISSION with the appropriate user name and view name.

This representation of views in relations recalls the representation of queries in QBE [7]. As an example, Figure 1 shows an instance of the example database extended with access permissions. For convenience of presentation, each pair of relations R, R' is shown as a Figuresingle contiguous table. There are four views: SAE (salary of all employees), ELP (employees of large projects), EST (employees with same title), and PSA (projects sponsored by Acme). User Brown is permitted to access SAE, PSA and EST, and user Klein is permitted to access ELP and EST.

Note that each individual meta-tuple may be regarded as defining a subview of the corresponding relation. The constants and variables specify the selection condition, and the $*$'s specify the projected attributes. For example, the meta-tuple (PSA, $*$, Acme$*$, $*$) stored in PROJECT$'$ specifies a selection of all tuples of relation PROJECT for which SPONSOR = Acme, and a projection of NUMBER, SPONSOR and BUDGET. And the meta-tuple (ELP, x_1*, $*$, $\sqcup$) stored in EMPLOYEE$'$ specifies a selection of all tuples of relation EMPLOYEE for which NAME = x_1, and a projection of NAME and TITLE. (A variable shared by another meta-tuple, such as x_1 in the view ELP in the meta-relation EMPLOYEE$'$, specifies a selection condition which is satisfied by any value from a set of values defined elsewhere.)

EMPLOYEE

VIEW	NAME	TITLE	SALARY
	Jones	manager	26,000
	Smith	technician	22,000
	Brown	engineer	32,000
SAE	$*$		$*$
ELP	x_1*	$*$	
EST	$*$	x_4*	
EST	$*$	x_4*	

PROJECT

VIEW	NUMBER	SPONSOR	BUDGET
	bq-45	Acme	300,000
	sv-72	Apex	450,000
	vg-13	Summit	150,000
PSA	$*$	Acme$*$	$*$
ELP	x_2*		x_3*

ASSIGNMENT

VIEW	E_NAME	P_NO
	Jones	bq-45
	Smith	bq-45
	Jones	sv-72
	Brown	sv-72
	Smith	vg-13
	Brown	vg-13
ELP	x_1*	x_2*

COMPARISON

VIEW	X	COMPARE	Y
ELP	x_3	$\geq$	250,000

PERMISSION

USER	VIEW
Brown	SAE
Brown	PSA
Brown	EST
Klein	ELP
Klein	EST

Figure 1: Database Extended with Access Permissions

4 Manipulating Access Permissions

Consider user U who is accessing database $R_1, \ldots, R_n$. Assume that meta-relations $R'_1, \ldots, R'_n$ and COMPARISON include definitions for views $V_1, \ldots, V_m$, and assume that relation PERMISSION includes permissions for U to access $V_1, \ldots, V_m$.

Granting user U permission to access $V_1, \ldots, V_m$ implies that permission is also granted to access any *view* of $V_1, \ldots, V_m$. Therefore, any request by U to access a view of $R_1, \ldots, R_n$ should be authorized, if the requested view is also a view of $V_1, \ldots, V_m$.

Assume now that user U requests to access view Q (i.e., user U submits query Q). Q should be authorized if it is also a view of $V_1, \ldots, V_m$. In addition, any subview of Q which is also a view of $V_1, \ldots, V_m$ may be authorized.

Consequently, given a request to access view Q of $R_1, \ldots, R_n$, we are interested in the subviews of Q that are also views of $V_1, \ldots, V_m$. In particular, we want to find out whether Q itself is also a view of $V_1, \ldots, V_m$.

As an example, recall that Klein was granted permission to access the view ELP, which presents the names and titles of employees that are assigned to projects with budgets exceeding \$250,000. If Klein submits a query to list the names of employees that are assigned to projects with budgets exceeding \$500,000, then the query should be authorized, since it is a view of ELP. If Klein submits a query to list the names and *salaries* of employees that are assigned to projects with budgets exceeding \$500,000, then only the subview of the request which is a view of ELP should be authorized: the names of employees that are assigned to projects with budgets exceeding \$500,000.

We describe a method that discovers subviews of a given query that should be authorized. Basically, this method *generates* views of $V_1, \ldots, V_m$ that are subviews of Q, by manipulating the definitions of $V_1, \ldots, V_m$ algebraically. These manipulations mirror those that are necessary to implement Q. In effect, we *generalize* the standard product, selection and projection operations to manipulate also relations of view definitions.

This method is illustrated by the commutative diagram shown in Figure 2. The solid lines describe the current situation: the views R' define the permissions on the database relations R, and the virtual relation A is derived from R to answer query Q. The dashed lines describe our method: query processing is extended to manipulate also R' to yield the views A' that define the permissions on the answer A.

$$\begin{tikzcd}
R' \arrow[r] \arrow[d, dashed, "Q"] & R \arrow[d, "Q"] \\
A' \arrow[r, dashed] & A
\end{tikzcd}$$

Figure 2: Extending Query Processing to Manipulate Access Permissions

For simplicity, in the remainder of this section we shall assume that attribute VIEW had already been removed from the meta-relations $R'_1, \ldots, R'_n$.

4.1 Meta-Relation Operations

Definition 1: Assume that R' and S' are meta-relations that define, respectively, views of R and S. The product of R' and S', denoted $R' \times S'$, is defined as follows. For every pair r and s of meta-tuples from R' and S', respectively,

$$\begin{aligned} r &= (a_1, \ldots, a_m) \\ s &= (b_1, \ldots, b_n) \end{aligned}$$

$R' \times S'$ includes the meta-tuple:

$$q = (a_1, \ldots, a_m, b_1, \ldots, b_n)$$

Proposition 1: Let $\mathbf{D}$ be an instance of this database, and let $r(\mathbf{D})$, $s(\mathbf{D})$ and $q(\mathbf{D})$ denote, respectively, the relations defined by r, s and q. Then $q(\mathbf{D}) = r(\mathbf{D}) \times s(\mathbf{D})$.

Proof: Let λ and μ denote, respectively, the selection predicates of r and s, and let α and β denote, respectively, the projected attributes of r and s. Then $r(\mathbf{D})$, $s(\mathbf{D})$ and $q(\mathbf{D})$ can be expressed as the following product-selection-projection expressions:

$$\begin{aligned} r(\mathbf{D}) &= \pi_\alpha \sigma_\lambda(R(\mathbf{D})) \\ s(\mathbf{D}) &= \pi_\beta \sigma_\mu(S(\mathbf{D})) \\ q(\mathbf{D}) &= \pi_{\alpha \cup \beta} \sigma_{\lambda \wedge \mu}(R(\mathbf{D}) \times S(\mathbf{D})) \end{aligned}$$

We observe that $\pi_{\alpha \cup \beta} \sigma_{\lambda \wedge \mu}(R(\mathbf{D}) \times S(\mathbf{D})) = \pi_\alpha \sigma_\lambda(R(\mathbf{D})) \times \pi_\beta \sigma_\mu(S(\mathbf{D}))$.

Definition 2: Assume that R' is a meta-relation that defines views of R. Let λ denote a primitive selection predicate (i.e., either $A'_i \, \theta \, c$, or $A'_i \, \theta \, A'_j$). The selection from R' by predicate λ, denoted $\sigma_\lambda(R')$, is defined as follows. Consider first the case $\lambda = A_i \, \theta \, c$, and let r be a meta-tuple from R',

$$r = (a_1, \ldots, a_i, \ldots, a_m)$$

Denote by μ the selection predicate expressed by a_i [1]. If a_i is suffixed by $*$, then $\sigma_\lambda(R')$ includes the meta-tuple:

$$q = (a_1, \ldots, a'_i, \ldots, a_m)$$

where a'_i represents $\lambda \wedge \mu$. Consider now the case $\lambda = A_i \, \theta \, A_i$, and let r be a meta-tuple from R',

$$r = (a_1, \ldots, a_i, \ldots, a_j, \ldots, a_m)$$

Denote by μ the selection predicate expressed by a_i and a_j. If a_i and a_j are both suffixed by $*$, then $\sigma_\lambda(R')$ includes the meta-tuple:

$$q = (a_1, \ldots, a'_i, \ldots, a'_j, \ldots, a_m)$$

where a'_i and a'_j represent $\lambda \wedge \mu$.

Proposition 2: Let $\mathbf{D}$ be an instance of this database, and let $r(\mathbf{D})$ and $q(\mathbf{D})$ denote, respectively, the relations defined by r and q. Then $q(\mathbf{D}) = \sigma_\lambda r(\mathbf{D})$.

Proof: Let α denote the projected attributes of r. Then

[1] If a_i is blank, then μ is true.

$r(\mathbf{D})$ and $q(\mathbf{D})$ can be expressed as the following selection-projection expressions:

$$\begin{aligned} r(\mathbf{D}) &= \pi_\alpha \sigma_\mu(R(\mathbf{D})) \\ q(\mathbf{D}) &= \pi_\alpha \sigma_{\mu\wedge\lambda}(R(\mathbf{D})) \end{aligned}$$

We observe that if the predicate λ is on attributes in α, then $\pi_\alpha \sigma_{\mu\wedge\lambda}(R(\mathbf{D})) = \sigma_\lambda \pi_\alpha \sigma_\mu(R(\mathbf{D}))$.

Definition 3: Assume that R' is a meta-relation that defines views of R. The projection of R' that removes its i'th attribute, denoted $\pi_{R-A_i}(R')$, is defined as follows. For every meta-tuple r from R',

$$r = (a_1, \ldots, a_m)$$

If a_i is $\sqcup$ (possibly suffixed with $*$), then $\pi_{R-A_i}(R')$ includes the meta-tuple:

$$q = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_m)$$

Proposition 3: Let $\mathbf{D}$ be an instance of this database, and let $r(\mathbf{D})$ and $q(\mathbf{D})$ denote, respectively, the relations defined by the meta-tuples r and q. Then $q(\mathbf{D}) = \pi_{R-A_i}(r(\mathbf{D}))$ [2].

Proof: Let λ denote the selection predicate of r, let α denote the projected attributes of r, and let $\beta = R - A_i$. Then $r(\mathbf{D})$ and $q(\mathbf{D})$ can be expressed as the following selection-projection expressions:

$$\begin{aligned} r(\mathbf{D}) &= \pi_\alpha \sigma_\lambda(R(\mathbf{D})) \\ q(\mathbf{D}) &= \pi_\alpha \sigma_\lambda \pi_\beta(R(\mathbf{D})) \end{aligned}$$

We observe that if the i'th attribute of R does not participate in the predicate λ, then $\pi_\alpha \sigma_\lambda \pi_\beta(R(\mathbf{D})) = \pi_\beta \pi_\alpha \sigma_\lambda$ $(R(\mathbf{D}))$.

Briefly, the selection and projection operations behave as follows: selection requires the attributes it selects to be in the projection attributes of the meta-tuple; and projection requires the attribute it removes not to be in the selection attributes of the meta-tuple.

Propositions 1, 2 and 3 are summarized in the following theorem:

Theorem: Assume user U is accessing database R_1, $\ldots, R_n$. Assume the meta-relations $R'_1, \ldots, R'_n$ and COMPARISON include definitions for views $V_1, \ldots, V_m$, and assume that relation PERMISSION includes permissions for U to access $V_1, \ldots, V_m$. Let Q be a conjunctive query against this database. Let S be the relational algebra expression that implements Q. Let S' be the relational algebra expression obtained from S by substituting every reference to R with a reference to R'. S operates on the relations to yield the answer A. S' operates on the meta-relations to yield the meta-answer A'. Then, the meta-tuples in A' define views of A that are also views of $V_1, \ldots, V_m$.

[2] In general, we define $\pi_\alpha(R)$ as a projection on those attributes in α that are in R. Thus, if attribute A_i had already been removed, a projection on $R - A_i$ has no effect.

The theorem guarantees that meta-tuples in A' define subviews of A that are views of $V_1, \ldots, V_m$ (and therefore user U is authorized to access them). However, some meta-tuples may still contain references to meta-tuples outside A', and are therefore not expressible entirely within A'. Such subviews are avoided if S' is modified so that all products are performed first, and their result is pruned to retain only those meta-tuples that do not contain references to other meta-tuples. Also, as we explain below, it is advantageous to perform selections before projections. Altogether, S' is transformed to a sequence of products, followed by selections, and ending with projections. This simple strategy for implementing conjunctive queries is not necessarily optimal. However, we note that the optimality is not so essential for meta-relations, because they are relatively small. For the actual relations, where optimality is essential, a different strategy may be implemented.

4.2 Refinements

The theorem guarantees that the method for generating subviews is *sound*, but it does not guarantee that it is *complete*. That is, this method generates subviews of the result that should indeed be authorized, but does not necessarily generate all such subviews. A method that would guarantee completeness would undoubtedly be of a different complexity altogether. Yet, with several simple refinements, it can be improved to generate additional desirable subviews. Three such refinements are sketched below.

When an attribute of R' is removed, all the meta-tuples that restrict this attribute (with a variable or a constant) are discarded. Consequently, some views may be lost. For example, assume that Q is a product of R and S, followed by a projection that removes all the attributes of S. Obviously, Q is equivalent to R, and A' should retain all the meta-tuples of R'. However, these meta-tuples may be discarded by the projection, if they contain restrictions in the attributes contributed by S'. To handle this situation we may extend the product of meta-relations to include also these two tuples:

$$\begin{aligned} q_1 &= (a_1, \ldots, a_m, \sqcup, \ldots, \sqcup) \\ q_2 &= (\sqcup, \ldots, \sqcup, b_1, \ldots, b_n) \end{aligned}$$

These tuples define all previous subviews of R and S as subviews of the product of R and S.

The selection operation offers additional refinement opportunities. As defined, this operation requires conjuncting μ, the predicate expressed in the meta-tuple, with λ, the predicate expressed in the query. However, as all the tuples in the resulting relation satisfy λ, the expression $\mu \wedge \lambda$ is simply μ. Therefore, it appears that a simpler definition of the selection operation may be provided, which simply retains all meta-tuples without any modification. On the other hand, this simpler definition often would not generate the best definitions of views, nor would it detect views that are indeed irrelevant. As an example, assume a meta-tuple that defines the projects whose budgets are between \$300,000 and \$600,000, and consider the following four queries that select the projects whose budgets are (1) between \$200,000 and \$400,000, (2) between \$200,000 and \$700,000, (3) between \$400,000 and \$500,000, and (4) under \$300,000. In each case, the given view (projects whose budgets are between \$300,000 and \$600,000) could be retained as a view of the projects selected. However, it would be more desirable to handle this selection on a case by case basis, as follows. In the first query, modify the given view to define the projects whose budgets are between \$300,000 and \$400,000; in the second query, retain the given view without any modification; in the third query, modify the given view so it does not restrict the budget at all; and, in the fourth query, discard the given view altogether. In general, we observe four different cases: If λ *implies* μ, the meta-tuple is selected and the corresponding field is cleared (i.e., the variable or the constant is replaced by $\sqcup$); if μ *implies* λ, the meta-tuple is selected without any modification; if λ and μ are *contradictory*, the meta-tuple is discarded; otherwise, the meta-tuple is selected, and is modified to represent $\mu \wedge \lambda$. Clearing selection predicates ensures that more meta-tuples will "survive" future projections. Determining the appropriate case for given μ and λ may require consulting relation COMPARISON, and, possibly, modifying it. While for most views and queries this task is quite simple, if an implementation chooses not to determine the case for predicates of certain form, then in those cases the relevant meta-tuple must not be selected. Note that the only other time where relation COMPARISON is used, is when the views in A' are described to the user.

Finally, with a simple procedure, it is often possible to infer additional subviews from subviews that are stored in the same meta-relation. For example, assume that EMPLOYEE$'$ has two meta-tuples: $(*, *, \sqcup)$ and $(*, \sqcup, *)$. A query that selects *both* TITLE and SALARY, will not select any of these meta-tuples. However, it can be shown that in this case $(*, *, *)$ is also a view of EMPLOYEE which should be authorized, and this meta-tuple would have been selected by the same query. Let r and s be meta-tuples in relation R' that do not belong to the same view. Assume that the subviews defined by r and s can participate in a *lossless join* (for example, both subviews include the key of this relation). We define their *self-join* with a meta-tuple q, as follows: c_i is the disjunction of the subviews defined in a_i and b_i, and it is suffixed by $*$ if both a_i or b_i are suffixed by $*$. It can be shown that self-joins are subviews of R which should be authorized. Note that self-joins need not be generated for every query; once generated, they should be stored with the original view definitions, until these definitions are modified.

5 The Authorization Process

In the previous section we described a method that discovers subviews of a given query to which the user has access permissions. In this section we demonstrate the authorization process based on this method.

Example 1: Assume Brown submits a request to retrieve the names and sponsors of large projects:

retrieve (PROJECT.NUMBER, PROJECT.SPONSOR)
where PROJECT.BUDGET $\geq$ 250,000

This query may be implemented with the following sequence of algebraic operations:

1. A $\leftarrow \sigma_{(\text{BUDGET} \geq 250{,}000)}(\text{PROJECT})$
2. A $\leftarrow \pi_{\text{NUMBER,SPONSOR}}(\text{A})$

First, PROJECT$'$ is pruned to include only tuples of views that Brown is permitted to access, and that are defined in this relation in their entirety:

PROJECT$'$

VIEW	NUMBER	SPONSOR	BUDGET
PSA	*	Acme*	*

Now, the same operations that are applied to the database relations are applied to their meta-relation counterparts:

1. A$' \leftarrow \sigma_{(\text{BUDGET} \geq 250{,}000)}(\text{PROJECT}')$
2. A$' \leftarrow \pi_{\text{NUMBER,SPONSOR}}(\text{A}')$

The selection retains the only view tuple unmodified, and the final projection yields:

A$'$

NUMBER	SPONSOR
*	Acme*

Recall that Brown was entitled to access the projects sponsored by Acme, and requested to access the numbers and sponsors of all projects. The above mask indicates that Brown is restricted to projects sponsored by Acme. Thus, all numbers and sponsors of projects not sponsored by Acme will be masked, and the following view definition will inform the user that permission exists only for SPONSOR = Acme:

permit (NUMBER, SPONSOR)
where SPONSOR = Acme

Example 2: Assume Klein submits a request to retrieve the names and salaries of engineers assigned to very large projects:

retrieve (EMPLOYEE.NAME, EMPLOYEE.SALARY)
where EMPLOYEE.TITLE = engineer
  and EMPLOYEE.NAME = ASSIGNMENT.E_NAME
  and ASSIGNMENT.P_NO = PROJECT.NUMBER
  and PROJECT.BUDGET $\geq$ 300,000

This query may be implemented with the following sequence of algebraic operations:

1. A $\leftarrow$ EMPLOYEE $\times$ ASSIGNMENT $\times$ PROJECT
2. A $\leftarrow \sigma_{(\text{TITLE=engineer})\wedge(\text{NAME=E\_NAME})\wedge(\text{NUMBER=P\_NO})\wedge(\text{BUDGET}\geq 300{,}000)}(\text{A})$
3. A $\leftarrow \pi_{\text{NAME,SALARY}}(\text{A})$

First, the meta-relations are pruned to include only tuples of views that Klein is permitted to access, and that are defined in these relations in their entirety:

EMPLOYEE′

VIEW	NAME	TITLE	SALARY
ELP	x_1*	*	
EST	*	x_4*	
EST	*	x_4*	

PROJECT′

VIEW	NUMBER	SPONSOR	BUDGET
ELP	x_2*		x_3*

ASSIGNMENT′

VIEW	E_NAME	P_NO
ELP	x_1*	x_2*

COMPARISON

VIEW	X	COMPARE	Y
ELP	x_3	$\geq$	300,000

PERMISSION

USER	VIEW
Klein	ELP
Klein	EST

Now, the same operations that are applied to the database relations are applied to their meta-relations counterparts:

1. A′ $\leftarrow$ EMPLOYEE′ $\times$ ASSIGNMENT′ $\times$ PROJECT′
2. A′ $\leftarrow \sigma_{(\text{TITLE=engineer})\wedge(\text{NAME=E\_NAME})\wedge(\text{NUMBER=P\_NO})\wedge(\text{BUDGET}\geq 250{,}000)}(\text{A}')$
3. A′ $\leftarrow \pi_{\text{NAME,SALARY}}(\text{A}')$

The result of the product after replications are removed is [3]:

A′

NAM	TIT	SAL	E_NA	P_NO	NUM	SPO	BUD
x_1*	*		x_1*	x_2*	x_2*		x_3*
x_1*	*		x_1*	x_2*			
x_1*	*				x_2*		x_3*
x_1*	*						
*	x_4*		x_1*	x_2*	x_2*		x_3*
*	x_4*		x_1*	x_2*			
*	x_4*				x_2*		x_3*
*	x_4*						
			x_1*	x_2*	x_2*		x_3*
			x_1*	x_2*			
					x_2*		x_3*

The selection retains only the first view tuple, and clears its variables:

A′

NAM	TIT	SAL	E_NA	P_NO	NUM	SPO	BUD
*	*						

Finally, after the projection:

A′

NAME	SALARY
*	

Recall that Klein was entitled to access the names and titles of employees of large projects, and the names of employees with the same title. He requested to access the names and salaries of engineers assigned to very large projects. The above mask indicates that Klein is permitted to access their names, but not their salaries. Thus, the values of salaries in the two column result will be masked, and the following view definition will inform the user that permission exists only for attribute NAME:

permit (NAME)

Example 3: Assume Brown submits a request to retrieve the names and salaries of employees with the same title:

retrieve (EMPLOYEE:1.NAME, EMPLOYEE:1.SALARY,
  EMPLOYEE:2.NAME, EMPLOYEE:2.SALARY)
where EMPLOYEE:1.TITLE = EMPLOYEE:2.TITLE

This query may be implemented with the following sequence of algebraic operations [4]:

1. A $\leftarrow$ EMPLOYEE $\times$ EMPLOYEE
2. A $\leftarrow \sigma_{(\text{TITLE:1=TITLE:2})}(\text{A})$
3. A $\leftarrow \pi_{\text{NAME:1,SALARY:1,NAME:2,SALARY:2}}(\text{A})$

First, the relevant meta-relations are pruned to include only tuples of views that Brown is permitted to

[3]For brevity, the VIEW attributes are omitted from intermediate results (they are not used in selections, and the final projections remove them anyway).

[4]When a relation has several attributes named A, then A : i denotes the i'th appearance of A.

access, and that are defined in these relations in their entirety:

EMPLOYEE′

VIEW	NAME	TITLE	SALARY
SAE	*		*
EST	*	x_4*	
EST	*	x_4*	

PERMISSION

USER	VIEW
Brown	SAE
Brown	PSA
Brown	EST

The first tuple in EMPLOYEE′ may be combined either with the second or with the third, yielding:

EMPLOYEE′

VIEW	NAME	TITLE	SALARY
EST, SAE	*	x_4*	*
EST, SAE	*	x_4*	*

Now, the same operations that are applied to the database relations are applied to their meta-relations counterparts:

1. $A' \longleftarrow$ EMPLOYEE′ × EMPLOYEE′
2. $A' \longleftarrow \sigma_{(\text{TITLE:1=TITLE:2})}(A')$
3. $A' \longleftarrow \pi_{\text{NAME:1,SALARY:1,NAME:2,SALARY:2}}(A')$

The result of the product after replications are removed is:

A′

NAM:1	TIT:1	SAL:1	NAM:2	TIT:2	SAL:2
*	x_4*	*	*	x_4*	*
*	x_4*	*			
			*	x_4*	*

The selection retains only the first view tuple, and clears its variables:

A′

NAM:1	TIT:1	SAL:1	NAM:2	TIT:2	SAL:2
*		*	*		*

Finally, after the projection:

A′

NAME:1	SALARY:1	NAME:2	SALARY:2
*	*	*	*

Recall that Brown was entitled to access the names of employees with the same title, and the salaries of all employees. He requested the names and salaries of employees with the same title. The above mask indicates that Brown is permitted to access the entire answer. This answer will be delivered without any accompanying **permit** statements.

6 Conclusion

We presented a new model for access authorization to relational databases. Work is underway on extensions in three areas. (1) Currently, the model incorporates only *retrieval* permissions. We see no difficulty in extending it to incorporate *update* permissions, such as *insert, delete* and *modify*[5]. (2) Another restriction of our model is that currently it handles only conjunctive views and queries. This restriction can be relaxed in several ways. For example, the current methods can be extended to handle views with *disjunctions* and views with *aggregate functions.* Finally, (3) the algorithm yields only permitted views (masks) that can be expressed with the attributes requested. It should be possible to extend our methods to deliver views that are expressed with additional attributes.

Work is also underway on a database "front-end" interface that will implement our methods and enable experimentation. The user will define access authorization with **permit** statements, and the system will insert automatically the appropriate meta-tuples into the meta-relations. In response to a **retrieve** statement, the user will receive a derived relation, whose structure corresponds to the request but whose tuples include only permitted values, and a set of inferred **permit** statements describing the portion delivered. Thus, the meta-relations and the meta-tuple notation would be completely transparent, with all user-system communication done with customary query language statements.

References

[1] P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems,* 1(3):242–255, Sep. 1976.

[2] *SunINGRES Manual Set.* Sun Microsystems, Mountain View, California, Release 5.0, 1987.

[3] D. Maier. *The Theory of Relational Databases.* Computer Science Press, Rockville, Maryland, 1983.

[4] A. Motro. *Integrity = Validity + Completeness.* Technical Report, Department of Computer Science, University of Southern California, Dec. 1987.

[5] M. Stonebraker and E. Wong. Access control in a relational database management system by query modification. In *Proceedings of ACM Annual Conference,* pages 180–186, 1974.

[6] J. D. Ullman. *Principles of Database Systems.* Computer Science Press, Rockville, Maryland, 1982.

[7] M. Zloof. Query-by-Example: a database language. *IBM Systems Journal,* 16(4):324–343, Dec. 1977.

[5]Note that the problem of *propagating* view updates and modifications to the actual relations remains, in general, unsolvable.