Access Control to Materialized Views: an Inference-Based Approach

Sarah Nait Bahloul
Advisors: Emmanuel Coquery, Mohand-Saïd Hacid
Université de Lyon, CNRS
Université Lyon 1, LIRIS, UMR5205, F-69622, France
sarah.nait-bahloul@liris.cnrs.fr

ABSTRACT

Ensuring security of data is one of the fundamental needs of people. In this context, issues related to confidentiality, integrity and availability of the data arise with a crucial importance, whether in economic, legal or medical domains. Standards covering fine-grained access control were proposed and adopted to control access to data through queries.
In this paper, we propose a novel approach to facilitate the administration of access control rules to ensure the confidentiality of data at the level of materialized views. Several techniques and models have been proposed to control access to databases, but to our knowledge the problem of automatically generating from access control rules defined over base relations the applicable access control rules needed to control materialized views is not investigated. We are investigating this problem by resorting to an adaptation of query rewriting techniques. We choose to express fine-grained access control through authorization views. This paper mainly discusses the problem of automatically ensuring confidentiality of materialized views based on basic access control rules, and identifies formal tools to tackle the problem.

1. INTRODUCTION

Data security is one of the major concerns in the data management community. A complete solution of data security must meet the following three requirements [3]:

- *Confidentiality refers to limiting information access and disclosure to authorized users and preventing access by or disclosure to unauthorized ones.*
- *Integrity refers to prevention of unauthorized and improper data modification.*
- *Availability of information resources; an information system must be accessible and operational in any time to a legitimate user.*

EDBT/ICDT PhD 2011, March 25, 2011, Uppsala, Sweden.

In very large systems such as data warehouses [18][16], or distributed systems [6], materialized views can be used for performance reasons or in order to duplicate or summarize data (e.g., for analysis purposes). In relational database management systems, a view is a virtual table representing the result of a query. A materialized view records the results returned by the query into a physical table. Thus, the user can perform the known access by querying the materialized view in the same manner as querying a database table: Selection, Projection, Join with other tables or materialized views. In this context, ensuring security at the materialized view level is as important as ensuring security on database tables.
Based on our previous work [9], we propose a novel approach that facilitate the administration of the access control rules to ensure the confidentiality of data at the level of materialized views. We compute new rules based on predefined access control rules defined over base relations.
In our approach, we consider fine-grained authorization policies that are defined and enforced in the database through **authorization views** [12]. Authorization views specify the accessible data, by projecting out of specific columns in addition to selecting rows. This framework allows fine-grained authorization at the cell level.
The rest of the paper is organized as follows: section 2 defines the problem. Section 3 discusses authorization views. Section 4 presents our approach. Section 5 presents the related work. We conclude in section 6.

2. PROBLEM STATEMENT AND MOTIVATION

The problem of access control to data has been investigated by the data management community. As a consequence, several techniques and models have been proposed to improve data security and ensure confidentiality [12][8][10][2].
Large systems like Data Warehouses or Distributed Database Systems use tables to store and manage their information and very often they resort to materialized views. In many settings, the views are materialized in order to optimize access. For instance, in Data Warehouses, they can be used to precompute and store complex aggregated data. In distributed computing, materialized views are used to replicate data at distributed sites and synchronize updates done at several sites. In these cases, the use of materialized view is as important as using a database. The question is then how to ensure data security at the level of a materialized view? So far, access control rules on materialized views are defined manually by an administrator by trying to comply with the

basic access control rules, i.e., the administrator builds new access control rules on a materialized view by taking into account those defined on base tables that are used to define the materialized view. In a system containing tens or hundreds of tables managed by hundreds of access control rules, it becomes impossible for administrators to deal with such huge number of rules and consider all the relevant ones. To cope with this problem, we propose to automatically generate a set of access control rules on the materialized view. Figure 1 summarizes our approach. The idea is the following: Given a set of base relations (with access control rules over those relations) and a view definition, synthesize a set of access control rules that will be attached to the view, such that querying the materialized view through those rules does not give more information than querying the database through the original access control rules.

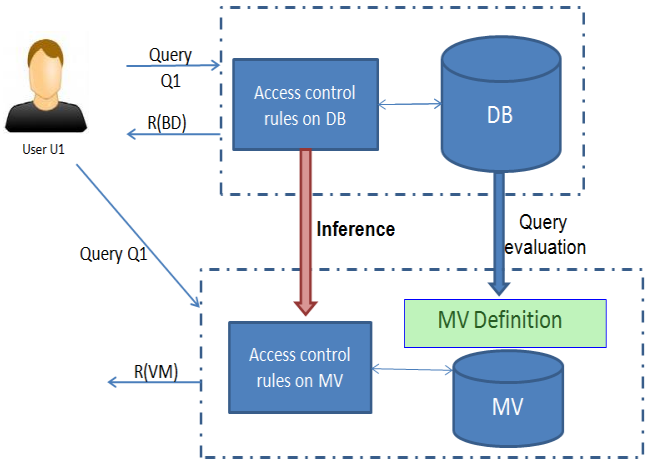

Figure 1: Which subset of access control rules on DB can ensure confidentiality at the materialized view level in such a way that, when a user requests the materialized view (s)he will not get more information than if (s)he requests the base relations

3. DATALOG FOR AUTHORIZATION

Note that we are not designing a new security model. We make use of existing rule-based access control models. We need a method that takes a set of access control rules, and computes exactly the data that is accessible by the user. For this purpose, we use authorization views.

Authorization views are a well known database techniques that provides fine-grained access control. They are implemented as part of a database management system. Authorization views are logical tables that specify exactly the accessible data, either drawn from a single table or from multiple tables. For each access control rule, an authorization view that stores the accessible data corresponding to the rule is defined.

In [19], the authors use the authorization views in order to test if a query posed on a database is valid by determining whether it can be completely rewritten using the authorization views. In our proposal, we use the authorization views in a different way. Indeed, the user has only right to query the authorization views. Hence, the aim of our work is to automatically generate a new set of authorization views to ensure confidentiality of data stored at the materialized view level based on the basic authorization views.

An authorization view can be a traditional relational view or a parameterized view [10][12]. A parameterized authorization view is a SQL view definition which makes use of parameters like user-id, time, user-location etc. These views provide a rule-based framework, where one view definition applies across several users. This allows the administrator to avoid encoding the same policies for each user.

In this paper, we consider two types of access control rules that might exist to support data access at the level of tables: authorizations views defined on a single table and those defined on multiple tables.

We propose the use of Datalog as a formal framework for expressing the access control rules [1].

We assume the existence of three types of symbols: variables, constants and predicates names. $p(t_1, ..., t_n)$ is a literal where p is a predicate name with arity n and each t_i for $1 \leq i \leq$n is either a constant or a variable. We call the sequence $(t_1, ..., t_n)$ a tuple with arity n. A rule is a statement of the form $p(u) \leftarrow q_1(u_1), ..., q_n(u_n)$, where p and each q_i for $1 \leq i \leq$n are relation names and $u_1, ..., u_n$ are tuples of appropriate arities. Each variable occurring in u must occur in at least one of $u_1, ..., u_n$. We call $p(u)$ the head of the rule, and $q_1(u_1), ..., q_n(u_n)$ sub-goals in the body of the rule. Rules may also contain sub-goals whose predicates are arithmetic comparisons $<, \leq, =$. In this case, we require that if a variable x appears in a sub-goal of a comparison predicate, then x must also appear in an ordinary sub-goal.

A Datalog program is a finite set of Datalog rules[1].

The logical sentence associated with the Datalog rule $p(u) \leftarrow q_1(u_1), ..., q_n(u_n)$ is:

$$\forall x_1 ... x_n (p(u_1) \leftarrow q_1(u_1) \wedge ... \wedge q_n(u_n)).$$

where $x_1 ... x_n$ are the variables occurring in the rule and the symbol $\leftarrow$ is the standard logical implication.

Now, we recall the notion of deductive database. We limit our discussion to facts and deductive rules. The fact that a tuple $(t_1, ..., t_n)$ is a member of the relation R is expressed by the literal $R(t_1, ..., t_n)$. Such facts define the extensional database, EDB, and such a relation R is called an extensional (stored) relation. The intentional database, IDB, contains deductive rules. Relations defined by deductive rules are called intentional relations [5] [1]. For example,

Info-Doc (Id-D, Dname, Dfname, Dspecialty) $\leftarrow$
doctor (Id-D, Dname, Dfname, Dadr, Dphone, Dspecialty, Dsalary)

is a rule that defines the (intentional) Info-Doc relation in terms of the (extensional) doctor relation.

In general, a rule $S \leftarrow R_1, ..., R_n$ defines a relation S in terms of the relation $R_1, ..., R_n$. In relational database terminology, view definitions are expressed as deductive rules. The schema of a program Datalog P, denoted $sch(P)$, is the union of $edb(P)$ and $idb(P)$. The semantics of a Datalog program P is defined in [1].

Example: The following rules define a hospital database (extensional relations) and the authorizations views (intentional relations).

Hospital database:
service (Snum, Sname, Hospital, Dept, Director).
doctor (IdD, Dname, Dfname, Dadr, Dphone, Dspecialty, Dsalary).
nurse (IdI, Snum, Nname, Nfname, Nadr, Nphone, Nsalary).
patient (IdP, Snum, Pname, Pfname, Disease).
care (IdD, idP).

Authorization views:
av_1 *(IdD, Dname, Dfname, Dspecialty)←*
doctor (IdD, Dname, Dfname, Dadr, Dphone, Dspecialty, Dsalary).

av_2 *(IdI, Snum, Nname, Nfname) ←*
nurse (IdI, Snum, Nname, Nfname, Nadr, Nphone, Nsalary).

av_3 *(IdP, Snum, Pname, Pfname, Disease)←*
patient (IdP, Snum, Pname, Pfname, Disease), care (IdD, IdP), IdD = User.

We allow all users to view the last name, the first name and the speciality of doctors in the hospital, but not personal and salary information. In the same manner, we do not allow users to view the salary information of nurses. This is accomplished by defining the two first rules in the example. We also allow all doctors to view information of their patients. This is accomplished by defining the third rule. In this rule, we define a parameterized authorization view which makes use of the parameter user-id.

Formally, we describe our problem as follows. Let $\mathcal{R}$ be a set of extensional[1] relations and $\mathcal{AV}_\mathcal{R}$ be a set of intentional relations defined in terms of the extensional relations $\mathcal{R}$. $\mathcal{AV}_\mathcal{R}$ represents the set of authorization views, that is, tuples that are members of $\mathcal{AV}_\mathcal{R}$ are tuples that are allowed to be accessed by users issuing queries over extensional relations $\mathcal{R}$.
Now, suppose we have new intentional relations $\mathcal{MV}$ (defined in terms of the extensional relations $\mathcal{R}$). The user querying $\mathcal{MV}$ relations may get additional tuples than when (s)he queries the $\mathcal{AV}_\mathcal{R}$ relations. The aim of our work is to define a set of intentional relations $\mathcal{AV}_\mathcal{MV}$ defined in terms of the intentional relations $\mathcal{MV}$. $\mathcal{AV}_\mathcal{MV}$ represents the set of authorization views such that for each intentional relation $\mathcal{Q}$ defined in terms of $\mathcal{AV}_\mathcal{MV}$, there exists an intentional relation $\mathcal{Q}'$ defined in terms of $\mathcal{MV}$ and an intentional relation $\mathcal{Q}''$ defined in terms of $\mathcal{AV}_\mathcal{AV}$
such that $\mathcal{Q} \sqsubseteq \mathcal{Q}'$ and $\mathcal{Q} \sqsubseteq \mathcal{Q}''$. In other words, for any database instance $\mathbf{I}$, $\mathbf{I}(\mathcal{Q})$ is a subset of $\mathbf{I}(\mathcal{Q}')$ and a subset of $\mathbf{I}(\mathcal{Q}'')$.

4. CONTRIBUTION

The aim of our work is to facilitate the specification and derivation of access control rules that should be attached to materialized views. We mainly consider conjunctive queries in the preliminary work.

For each (intentional) relation mv in $\mathcal{MV}$, the first step is about authorization views selection. The access control rules involving data that are present in mv are considered relevant, hence, they are selected. The aim of the first step is to determine what data are accessible in the mv relation. i.e. rewrite the query using only the set of authorization views $\mathcal{AV}_\mathcal{R}$.

[1]Note that our proposal can easily be extended to allow the use of intentional relations in R.

In order to implement the first step, we build on the Bucket algorithm [11] for query rewriting algorithms in data integration. This algorithm assumes that *all views to be considered to rewrite the query can be reduced consistently, if we handle each sub-goal (relations used to define the query) separately.* We make use of this idea in our rewriting framework.
The first step of the traditional bucket algorithm consists of creating a bucket for each sub-goal except the sub-goal of a comparison predicate in the query. Each entry of the bucket is the head of a view (in our case, the view represents the authorization view). Each entry must satisfy the following conditions:

(a) one of the sub-goal of the view must be mapped to a sub-goal of the query.

(b) If a head variable of the query appears in the query sub-goal, it must also appear in the head of the view.

(c) If the query has a comparison predicate sub-goal , then any view with a comparison predicate with the same variable is acceptable if its comparison predicate is consistent with the comparison predicate of the query.

(d) If a sub-goal of the query is mapped to more than one sub-goal of a particular view, then the head of this rule appears multiple times in the bucket of that sub-goal.

For each sub-goal (here an extensional relation) of mv, the algorithm determines the views that are attached to it. For this, we modify the conditions mentioned above. We do not take into account the condition (b). Indeed, this condition removes some relevant authorization views. For example, assume the query (1) shown below (materialized view definition). It makes a copy of table r. The rule (2) states an authorization view, which says that the user accessing tuples in mv has right to access only values of x, y attributes:

$mv(x,y,z) \leftarrow r(x,y,z)$ (1)

$av(x,y) \leftarrow r(x,y,z)$ (2)

If we apply the original bucket algorithm, the authorization view will be considered as irrelevant. Indeed, the attribute 'z' appears in the sub-goal of the query and it is also in the head of the query, then it must also be in the head of the view. This is not the case. If we do not take it as relevant, this means that the user has no rights to access to any tuple in mv. It is too strict, since one can project out 'z' by generating the appropriate authorization view on mv. In contrast, if we take it as relevant, this means that the user has only access to values of x,y attributes of the materialized view; which corresponds to the basic conditions for access.
After the selection step, the algorithm generates a set of query rewritings $\mathcal{RW}$ by combining the selected authorizations views, one from each sub-goal. These rewritings represent data that are accessible in mv.

The second step of our algorithm is to define $\mathcal{AV}_\mathcal{MV}$. We recall that the purpose of our work is to define the set of $\mathcal{AV}_\mathcal{MV}$, authorization views defined in terms of $\mathcal{MV}$ relations. Users wishing to query $\mathcal{MV}$ relations, are allowed only to query $\mathcal{AV}_\mathcal{MV}$. For this, we re-apply the original query rewriting algorithm on the computed $\mathcal{RW}$ in first step. i.e. rewrite each rw in $\mathcal{RW}$ using only the set of $\mathcal{MV}$ relations. This step is important, because if we apply only the first step, we will have a set of rewritings that represent (certainly) the accessible data but if the user wants access

to tuples in $\mathcal{MV}$ relations, (s)he must use the basic authorization views. This is not our goal. By applying the second step, we compute the set of authorization views defined in terms of $\mathcal{MV}$ relations. Hence, access to the $\mathcal{MV}$ relations no longer need access to $\mathcal{AV}_\mathcal{R}$ and $\mathcal{R}$ relations.

The idea behind using the bucket algorithm is that it exploits the predicates in the query to significantly prune the number of candidate conjunctive rewritings that need to be considered. Other query rewriting algorithms were proposed. For example, the MiniCon algorithm [11] allows to efficiently perform selection of relevant views in the first step. Nevertheless, we decided to make use of the Bucket algorithm for its simplicity and we will consider the use of the MiniCon algorithm in future works.
To give an intuitive idea about the approach, let us consider two examples.

Case 1: authorization views defined on a single table

Consider the following schema $r(a, b, c)$, $s(d, e)$, $p(f)$ and the corresponding set of authorization views:

$av_1(a,b,c) \leftarrow r(a,b,c),\ Cond(c)$
$av_2(d) \leftarrow s(d,e)$
$av_3(f) \leftarrow p(f),\ Cond(f)$
$av_4(b,c) \leftarrow r(a,b,c)$

The set of authorization views represents access control rules defined on a single table. $Cond(x)$ is a condition constraining the attribute x.
Assume the following intentional relation mv_1 (materialized view definition):

$mv_1(z) \leftarrow r(x,y,z),\ s(x,w)$

In the first step, the algorithm selects the authorization views that are attached to the relations involved in mv_1. For this, the algorithm creates a bucket for each sub-goal g in mv_1. The bucket for a sub-goal g contains the views that include sub-goals that can be mapped to g in a rewriting [11].

$r(x, y, z)$	$s(x, w)$
$av_1(x, y, z)$ $av_4(y, z)$	$av_2(x, w)$

In the second step of the query rewriting algorithm, the bucket algorithm considers conjunctive query rewritings, each one consisting of one conjunct from each relation. In the example, the combination of av_4 and av_2 fails. Indeed, if we use av_4, it will be not possible to apply the join between $r(x, y, z)$ and $s(x, w)$: av_4 projects out of the join attribute x. In this example, there is only one possible rewriting. It corresponds to the accessible data by the user that comply with the base rules.

$rw_1(x,y,z,w) \leftarrow av_1(x,y,z),\ av_2(x,w)$

The second step of our algorithm is to compute the new authorization view based on the rewriting rw_1 by applying the rewriting query algorithm. i.e. rewrite rw_1 in terms of mv_1. For this, we first must expand rw_1. The expansion consists in replacing the authorization views by their definitions:

$rw_1(x,y,z,w) \leftarrow r(x,y,z),\ s(x,w), Cond(z).$

The new authorization view is a relation defined in terms of mv_1 and has the following definition:

$av_{mv}(z) \leftarrow mv_1(z),\ Cond(z)$

Case 2: authorization views defined over several tables

Continuing with our example, assuming we have, in addition, the following authorization view defined over two tables:

$av_5(d) \leftarrow s(d,e),\ p(e)$

After applying the first step, i.e. rewrite mv_2 in terms of av_i we have in addition to the rewriting rw_1, a second rewriting:

$rw_2(x,y,z) \leftarrow av_1(x,y,z),\ av_5(x).$

when applying the second step, i.e. rewriting rw_2 in terms of mv_1, at the creation of buckets for each sub-goal in rw_2, there is no sub-goal in mv_1 that can be unified with $p(e)$. Indeed, after applying expansion, one obtains:

$rw_2(x,y,z) \leftarrow r(x,y,z), s(x,w), p(w), Cond(z).$

In this example, there is only one intentional relation: mv_1 in $\mathcal{MV}$. In other words, to ensure the confidentiality of mv_1 data we must access to the relation p and make a join. This is possible only if there is another intentional relation mv_i in $\mathcal{MV}$ that contains p in its definition. If the relation is not accessible or the access to the relation is very expensive; then we cannot apply the access control rule.

5. RELATED WORK

Rosenthal and Sciore [13][14] have considered the problem of how to automatically coordinate the access rights of the warehouse with those of sources. The authors proposed a theory that allows automated inference of many permissions for the warehouse by a natural extension of the standard SQL grant/revoke model, to systems with redundant and derived data. They split the notion of 'access permission' on a table into two issues [13]: information permissions, specifying who is allowed to access what information, and physical permissions which specify who is allowed to access which physical tables. The second contribution of the authors is about providing an inference mechanism based on witness notion. 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.
The authors also defined an extension of the witness notion by including the use of views. In other words, the user can accesses only to a part of a table represented by a view; Hence, the user has clearance to view the values of T that contribute to the view (information permission) and (s)he is allowed to execute a query that physically accesses T, but only for the purpose of computing the view (physical permission). For instance, to define access permission to a materialized view MV, the inference mechanism for permissions tries to derive a query Q and views V_i equivalent to MV, such that for each table t_i mentioned in Q, either

- the permission to access to t_i has been explicitly granted, or
- the user has access to values in t_i that contribute to V_i, where Q is equivalent to V_i.

Note that the notion of equivalence is very important in that paper. Suppose now, we define a materialized view mv_1 based on the join of two tables t_1 and t_2. We also define two views v_1, v_2 defining what tuples the user has right to access in t_1 and t_2 respectively. To determine if the user has right to access mv_1, we must find an equivalent query Q to mv_1 that uses only t_1 and t_2 that contributes to v_1 and v_2 respectively. The proposed framework will conclude that the user has not right to access mv_1 if the inference mechanism does not find an equivalent query even if the user has right to access a part of mv_1. However, in our approach, we propose a more flexible model. Indeed, by inferring the set of authorizations views to control access to the materialized view, we allow users to access even a part of the materialized view.
To summarize, the framework proposed by the authors determine only if a user has right to access a derived table (based on explicit permission) but our proposal goes further by determining which part the user has right to access in the derived table.
Note also that the authors stated the inference rules at a high level. The properties of this inference system and the efficient evaluation algorithm were not investigated and remain an open research issue.
In [7], the authors have built on [9] to provide a way to select access control rules to be attached to materialized view definitions based on access control rules over base relations. They resort to the basic form of the bucket algorithm which does not allow to derive all relevant access control rules. Another limitation of this work is that since they only deal with selection of rules, the framework remains strongly dependent of the base relations. That is, the body of the derived rules involves base relations only. In our work, we synthesize new rules from existing rules where the body of the new rules makes reference to materialized views.

6. CONCLUSION

In the case of large organizations, the management of thousands of datasets is very common. Ensuring data confidentiality in the presence of materialized views is also important. In this work, we discussed ingredients for an automated method to derive access control rules for materialized views by selecting the appropriate access control rules that are attached to underlying base tables. We adapted the bucket query rewriting algorithm.
An algorithm that enforces fine-grained access control policies on MV should be [17]:

- Sound: Generated views should returns only correct information. Let $R(MV)$ the result of answering the query Q when there are no access control rules. Because the set of access control rules restricts access to MV, $R_{AV}(MV)$ (the result of answering the query when there are access control rules) may returns less information than $R(MV)$ does, but it should not return wrong information.

- Secure: It requires that the returned answer does not leak information not allowed by the policy. Let $R_{AV}(T)$ the result of answering the query Q on original tables when there are access control rules. If a tuple is in $R_{AV}(MV)$ then it should be in $R_{AV}(T)$.

- Maximum: Generated views should returns as much information as possible, while satisfying the first two properties.

Hence, we are investigating the soundness, security and maximality of our algorithm. To do this, we are working on the proprieties of the query rewriting algorithm.
As mentioned above, we have discussed only conjunctive queries. In large systems, (e.g., data warehouses), materialized views can be used to precompute and store aggregated data (e.g., sum of sales). This framework should also be extended to accommodate materialized views with aggregate functions. For this purpose, we will consider algorithms for rewriting aggregate queries using views [15].
For this preliminary investigation, we illustrated the approach by resorting to the bucket algorithm as a query rewriting algorithm. However, it has some limits; after the creation of buckets, the algorithm considers conjunctive rewriting for each element of the Cartesian product of the buckets, and checks whether it is contained or can be made to be contained in the query. The checking is done by looking for a containment mapping between the query and the rewriting, which is an NP-complete problem [4]. We also plan to study other query rewriting algorithms that can be more efficient than the bucket algorithm. For instance, the MiniCon algorithm [11] considers the join predicates in the query to construct a MiniCon Despcription (MCD). Because of the way MCD works, the MiniCon algorithm does not require containment checks in the second phase, giving it an additional speed-up compared to the bucket algorithm. Future work includes also implementing and testing the proposed approach.

Acknowledgments

This work is partially supported by the Rhône-Alpes Region, Cluster ISLE (Informatique, Signal, Logiciel Embarqué).

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] M. M. Astrahan, H. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson. System r: Relational approach to database management. *ACM Transactions on Database Systems*, 1:97–137, 1976.

[3] E. Bertino and R. Sandhu. Database security-concepts, approaches, and challenges. *IEEE Trans. Dependable Secur. Comput.*, 2(1):2–19, 2005.

[4] D. Calvanese, D. Lembo, and M. Lenzerini. Survey on methods for query rewriting and query answering using views, 2001.

[5] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Trans. Database Syst.*, 15(2):162–207, 1990.

[6] L. W. F. Chaves, E. Buchmann, F. Hueske, and K. Böhm. Towards materialized view selection for distributed databases. In *EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology*, pages 1088–1099, New York, NY, USA, 2009. ACM.

[7] A. Cuzzocrea, M.-S. Hacid, and N. Grillo. Effectively and efficiently selecting access control rules on materialized views over relational databases. In *Proceedings of the Fourteenth International Database Engineering & Applications Symposium*, IDEAS '10, pages 225–235, New York, NY, USA, 2010. ACM.

[8] A. Motro. An access authorization model for relational databases based on algebraic manipulation of view definitions. In *Proceedings of the Fifth International Conference on Data Engineering*, pages 339–347, Washington, DC, USA, 1989. IEEE Computer Society.

[9] S. Nait-Bahloul. Inference of security policies on materialized views. rapport de master 2 recherche. http://liris.cnrs.fr/∼snaitbah/wiki, 2009.

[10] L. E. Olson, C. A. Gunter, and P. Madhusudan. A formal framework for reflective database access control policies. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 289–298, New York, NY, USA, 2008. ACM.

[11] R. Pottinger and A. Y. Levy. A scalable algorithm for answering queries using views. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 484–495, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[12] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 551–562, New York, NY, USA, 2004. ACM.

[13] A. Rosenthal and E. Sciore. View security as the basis for data warehouse security. In *CAiSE Workshop on Design and Management of Data Warehouses*, pages 5–6, 2000.

[14] A. Rosenthal and E. Sciore. Administering permissions for distributed data: Factoring and automated inference. In *In Proc. of IFIP WG11.3 Conf*, 2001.

[15] D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy. Answering queries with aggregation using views. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 318–329, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[16] D. Theodoratos and T. Sellis. Dynamic data warehouse design. In *1st Int. Conf. on DaWak '99*, pages 1–10. Springer-Verlag, 1999.

[17] Q. Wang, T. Yu, N. Li, J. Lobo, E. Bertino, K. Irwin, and J.-W. Byun. On the correctness criteria of fine-grained access control in relational databases. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 555–566. VLDB Endowment, 2007.

[18] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 136–145. Morgan Kaufmann, 1997.

[19] Z. Zhang and A. O. Mendelzon. Authorization views and conditional query containment. In *In Database Theory - ICDT 2005, 10th International Conference*, pages 259–273, 2005.