# Consistent Query Plan Generation in Secure Cooperative Data Access

Meixing Le, Krishna Kant, Sushil Jajodia
Center for Secure Information Systems
George Mason University, Fairfax, VA
{*mlep, kkant, jajodia*}*@gmu.edu*

***Abstract*—In this paper, we consider an environment where a group of parties have their own relational databases, and provide restricted access to other parties. In order to implement desired bushiness services, each party defines a set of authorization rules over the join of basic relations, and the accessible information is constrained by these rules. However, authorization rules are given based on the business requirements and the enforcement issues of the rules may not have been taken into consideration. In this paper, we propose an algorithm to check the rule enforceability for each given authorization rule, and also present mechanisms to generate a query execution plan for an authorized query which is consistent with the authorization rules. As finding the optimal query plan can be very difficult, we propose an algorithm to generate efficient query plans. In addition, we compare the generated query plans with the optimal query plans through case studies. The results show the effectiveness of our approach.**

***Index Terms*—Rule enforcement, Consistent query planning, Cooperative data access**

## I. Introduction

Providing rich services to clients with minimal manual intervention or paper documents requires the enterprises involved in the service path to collaborate and share data in an orderly manner. For instance, to arrange automated shipping of merchandise and to enable automated status checking, the e-commerce vendor and shipping company should be able to exchange relevant information, perhaps in the form of database queries. In such environments, data must be released only in a controlled way among cooperative parties, subject to the authorization policies established by them. In this paper, we expose and study various facets of problem of such a collaboration problem.

In general, enterprise data may appear in a variety of forms. However, we assume that all data is stored with relational tables in a standard form. In such a model, data access privileges are given by a set of authorization rules, each of which is defined either on original tables belonging to an enterprise or over the lossless join of two or more of these. The join operations, coupled with appropriate projection operations define the access restrictions. For each query, the system must ensure that it is allowed only if it satisfies authorization rules. Although the problem is rather straightforward, there are many hurdles in properly specifying and implementing the authorization rules. Since the enterprises are allowed to specify an arbitrary set of authorization rules, it is possible that there is no way to derive a safe execution plan for certain rules. The simplest way to illustrate this problem is by considering the following situation: a rule specifies access to $R \bowtie S$ (where $R$ and $S$ are relations owned by two different parties); however, no party has access to both $R$ and $S$ individually, and it is impossible to do the join operation! Thus, a basic problem is to determine the implementability of the rules and provide good (perhaps optimal) query plans. If no implementation is feasible, we may need trusted third parties to enforce the rules. However, we plan to address this issue in future works.

We address the authorization problem in two steps. First, we examine each authorization rule and check its possible enforcement. To achieve that, we propose a constructive mechanism that works from bottom-up, and it builds a graph structure that captures the relationships among the enforceable rules. This can be done once all the rules are given, and we can do it as a pre-computing step. To actually answer an authorized query, a query execution plan is needed, and such a plan should be consistent with the given set of rules so that no access restriction is violated. Once we know the enforceability of the rules and the built graph structure, we can do efficient query planning based on them. Queries that are authorized by enforceable rules are guaranteed to have safe execution plans. However, due to the large search space for possible optimal query plans, finding the best plan for an authorized query are not always possible. We show the difficulty of finding optimal answer in our scenario, and how it differs from classical query processing. To that end, we propose an efficient algorithm that gives query plans based on greedy heuristic. In addition, we demonstrate through simple examples to show the effectiveness of our approach. At last, we prove that our algorithms are both correct and complete.

The rest of the paper is organized as follows. Section II briefly discusses the related work. Section III defines the problem of cooperative access formally, introduces a number of definitions and concepts, which are illustrated via a running example. Section IV discusses the mechanism to check rule enforceability. Section V analyzes the complexity of query planning. Section VI describes the algorithm for generating query plans. At last, Section VII is about conclusions and future works.

## II. Related Work

In previous works, researchers proposed models for controlling the cooperative data release. The authorization rule model

in our paper is inspired by the study [9]. The main contribution of that work is an algorithm to check if a query with a given query plan tree can be safely executed. However, it does not address the problem of how the given rules are implemented and query plan trees are generated. We note in this regard that regular query optimizers do not comprehend access restrictions and may fail to generate some possible query plans. Thus, we address these problems in this work.

In another work [8], the same authors evaluate whether a query can be authorized if the given authorization rules can be further composed with one another. Their solution uses a graph model to find all the possible compositions of the given rules, and checks the query against all the generated rules. In contrast, we assume authorizations are explicitly given. Data release is prohibited if there is no explicit authorization.

There are also existing works on distributed query processing under protection requirements [4], [10], [14] which considers a limited access pattern called binding pattern so that the portion of data that can be accessed depends on the input data. There are also classical works on query processing in centralized and distributed systems [3], [13], [5], but they do not deal with constraints from the data owners, and our problem is different from these works.

Answering queries using views [12] is closer to our work, since each authorization rule can be thought of as a view over basic relations. These works can be used for query optimization [6], [11], maintaining physical data independence [17] and data integration [15]. Queries can also be rewritten using given views with query rewriting techniques [18], and conjunctive queries are used to evaluate the query equivalence and information containment. The scenario we consider here is different in that we cannot rewrite the queries in terms of rules without knowing how the rules can be enforced. Once we know all the enforcement of the rules and the query does have a consistent query plan, then some of these works can be used to further optimize the query plan. There are services such as Sovereign joins [2] to provide third party join services, we can think this as one possible third party model in our scenario. In addition, there are some research [1], [7], [16] about how to secure the data for out-sourced database services. These methods are also useful for enforcing the authorization rules, but we consider the scenario without any involvement of third parties.

## III. Problem and Definitions

We consider a group of cooperating parties, and we assume simple select-project-join queries (e.g., no cyclic join schemas or queries). The query may be answered by any party that has the required authorizations. We assume that the join schema is given – i.e., all the possible join attributes between relations are known. Each join in the schema is lossless so a join attribute is always a key attribute of some relations. We assume the rules to be upwards closed. That is, if two rules expressly grant permission to access two different relations, say $R$ and $S$, then there also exists a rule providing access to their join result $R \bowtie S$ including all their attributes. It is reasonable since a party could do any computation over the information given to it. The cooperative parties are honestly follow the rules, and we study the problems only involving existing cooperative parties, without any third parties. The basic problems considered here are as follows: Given a set of authorization rules $R$ on $N$ cooperating parties, (a) identifies the subset of $R$ that can be enforced along with consistent plans, and determines the maximal enforceable portions of these rules (b) derives a query execution plan that is consistent with the rules $R$ for an incoming authorized query $q$.

### A. A Running Example

Our running example for illustration models an e-commerce scenario with four parties: (a) *E-commerce*, denoted as $E$, is a company that sells products online, (b) *Customer_Service*, denoted $C$, that provides customer service functions (potentially for more than one Company), (c) *Shipping*, denoted $S$, provides shipping services (again, potentially to multiple companies), and finally (d) *Warehouse*, denoted $W$, is the party that provides storage services. To keep the example simple, we assume that each party has but one relation described as follows:

1) E-commerce (order_id, product_id, total) as E
2) Customer_Service (order_id, issue, assistant) as C
3) Shipping (order_id, address, delivery_type) as S
4) Warehouse (product_id, location) as W

In the following, we use $oid$ to denote $order\_id$ for short, $pid$ stands for $product\_id$, and $delivery$ stands for $delivery\_type$. The possible join schema is also given in figure 1. Relations $E$, $C$, $S$ can join over their common attribute $oid$; relation $E$ can join with $W$ over the attribute $pid$. The relations are in BCNF, and the only FD (Functional Dependency) in each relation is the underlined key attribute determines the non-key attributes.

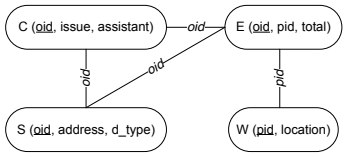


Fig. 1. The given join schema for the example

### B. Authorization model and definitions

An authorization rule $r_t$ is a triple$[A_t, J_t, P_t]$, where $J_t$ is called the join path of the rule, $A_t$ is the authorized attribute set, and $P_t$ is the party authorized to access the data.

*Definition 1:* A **join path** is the result of a series of join operations over a set of relations $R_1, R_2...R_n$ with the specified equi-join predicates $(A_{l1}, A_{r1}), (A_{l2}, A_{r2})...(A_{ln}, A_{rn})$ among them, where $(A_{li}, A_{ri})$ are the join attributes from two relations. We use the notation $J_t$ to indicate the join path of rule $r_t$. We use $JR_t$ to indicate the set of relations in a join path $J_t$. The **length** of a join path is the cardinality of $JR_t$.

We can consider a join path as the result of join operations without limitations on the attributes. Thus, $A_t$ is the set of attributes projection on the join path that is authorized to be

| Rule No. | Authorized attribute set | Join Path | Party |
|---|---|---|---|
| 1 | {pid, location} | $W$ | $P_W$ |
| 2 | {oid, pid} | $E$ | $P_W$ |
| 3 | {oid, pid, location} | $E \bowtie_{pid} W$ | $P_W$ |
| 4 | {oid, pid, total} | $E$ | $P_E$ |
| 5 | {oid, pid, total, issue} | $E \bowtie_{oid} C$ | $P_E$ |
| 6 | {oid, pid, total, issue, address} | $S \bowtie_{oid} E \bowtie_{oid} C$ | $P_E$ |
| 7 | {oid, pid, location, total, address} | $S \bowtie_{oid} E \bowtie_{pid} W$ | $P_E$ |
| 8 | {oid, pid, issue, assistant, total, address, delivery} | $S \bowtie_{oid} E \bowtie_{oid} C \bowtie_{pid} W$ | $P_E$ |
| 9 | {oid, address, delivery} | $S$ | $P_S$ |
| 10 | {oid, pid, total} | $E$ | $P_S$ |
| 11 | {oid, pid, total, address, delivery} | $E \bowtie_{oid} S$ | $P_S$ |
| 12 | {oid, pid, total, location} | $E \bowtie_{pid} W$ | $P_S$ |
| 13 | {oid, location, pid, total, address, delivery} | $S \bowtie_{oid} E \bowtie_{pid} W$ | $P_S$ |
| 14 | {oid, pid} | $E$ | $P_C$ |
| 15 | {oid, issue, assistant} | $C$ | $P_C$ |
| 16 | {oid, pid, issue, assistant} | $E \bowtie_{oid} C$ | $P_C$ |
| 17 | {oid, pid, issue, assistant, total, address, location} | $S \bowtie_{oid} C \bowtie_{oid} E \bowtie_{pid} W$ | $P_C$ |

TABLE I
AUTHORIZATION RULES FOR E-COMMERCE COOPERATIVE DATA ACCESS

accessed by party $P_t$. Table I shows the set of rules given to these parties. The first column is the rule number, the second column gives the attribute set of the rules, join paths of the rules are shown in the third column, and the last column shows the authorized parties of the rules. We assume that each given authorization rule always includes all of the key attributes of the relations that appear in the rule join path. In other words, a rule has all the join attributes on its join path. This is a reasonable assumption as usually when the information is released, it is always released along with the key attributes.

When a query is given, it should be answered by one of the parties that have the authorization. Since our authorization model is based on attributes, any attribute appearing in the Selection predicate in an SQL query is treated as a Projection attribute. In other words, the authorization of a PSJ query is transformed into an equivalent Projection-Join query authorization. Thus, a query $q$ can be represented by a pair $[A_q, J_q]$, where $A_q$ is the set of attributes appearing in the Selection and Projection predicates, and the query join path $J_q$ is the FROM clause of an SQL query. For instance, there is an SQL query:

"Select $oid, total, address$ From $E$ Join $S$ On $E.oid = S.oid$ Where $delivery$ = 'ground'"

The query can be represented as the pair $[A_q, J_q]$, where $A_q$ is the set $\{oid, total, address, delivery\}$; $J_q$ is the join path $E \bowtie_{oid} S$. In fact, each join path defines a new relation/view, and we say two join paths $J_i$ and $J_j$ are **equivalent**, noted as $J_i \cong J_j$, if any tuple in $J_i$ appears in $J_j$ and vice versa. As information release is explicitly defined by the rules, an authorized query must has a matching rule to allow the access.

*Definition 2:* A query $q$ is **authorized** if there exists a rule $r_t$ such that $J_t \cong J_q$ and $A_q \subseteq A_t$

The rule and the authorized query must have the equivalent join paths. Otherwise, the relation/view defined by the rule will have fewer or more tuples than the query asks for. Here we don't consider the situation where the projections on two different join paths get the same result (e.g., by joining on foreign keys) since data coming from different parties usually does not have foreign key constrains. For instance, the example query $Q_1$ is authorized by $r_{11}$, but it cannot be authorized by $r_{13}$. Although all the required attributes are authorized by $r_{13}$, their join paths are not equivalent.

On the other hand, "authorized" is only a necessary condition for a query to be answered but not sufficient as some of rules are not enforceable. For it to be sufficient, we need to give at least one query execution plan to answer the query. A **query execution plan** or "query plan" for short, includes several ordered steps of operations over some information and provides the resulting information to a party. As our authorization model does not define selection operation, the result of a query execution plan $pl$ can also be presented with a triple $[A_{pl}, J_{pl}, P_{pl}]$. To answer a query, the query plan must satisfy the following condition: A plan $pl$ answers a query $q$, if $J_{pl} \cong J_q$ and $A_q \subseteq A_{pl}$. In the next subsection, we introduce the notion of consistent query plan, and only consistent plans are considered safe to answer the queries.

### C. Query plan consistency

A query plan recursively contains a series of operations over subplans until the subplans are access plans getting information from basic relations. Each operation in the plan takes the result of subplans as input and generates another plan as output. In our context, the possible operations on plans are projection, join and data transmission. For instance, there is an enforcement plan for $r_3$ in Table I, and such a plan contains a join over two subplans on the data authorized by $r_1$ and $r_2$ respectively. $P_W$ owns the information authorized by $r_1$, and the subplan for it is an access plan reading the table $W$. The sub plan for $r_2$ includes an access plan reading table $S$ at $P_S$, and another operation transmitting the data from $P_S$ to $P_w$. The example plan authorized by $r_3$ has the $J_{pl} = E \bowtie_{pid} W$, and $A_{pl} = \{oid, pid, location\}$. We say a rule $r_t$ **authorizes** ($\succeq$) a plan $pl$, if $J_{pl} \cong J_t$, $P_{pl} = P_t$, and $A_{pl} \subseteq A_t$.

*Definition 3:* An operation in a query plan is **consistent** with the given rules $R$, if for the operation, there exist rules

that authorize access to the input tuples of the operation and to the resulting output tuples.

For the three types of operations in our scenario, we give the corresponding conditions for consistent operation.

1) Projection ($\pi$) is a unary operation. For a projection to be consistent with the rules, there must be a rule $r_p$ authorizes ($\succeq$) the input information.
2) Join ($\bowtie$) is a binary operation, and two input subplans $pl_{i1}$ and $pl_{i2}$ do a join operation and the resulting plan $pl_o = pl_{i1} \bowtie pl_{i2}$. For a join operation to be consistent with $R$, all the three plans need to be authorized by rules. Since join is performed at a single party, and rules are upwards closed, if the input plans are authorized by rules, the join operation is consistent.
3) Data transmission ($\rightarrow$) is an operation involves two parties. The input is a plan $pl_i$ on a party $P_i$, and the output is a plan $pl_o$ for a party $P_o$, where $pl_o = pl_i \rightarrow P_o$. In our scenario, data cannot be freely transmitted between parties. As each join path defines a different relation, the receiving party must have a rule that is defined on the equivalent join path as the information being sent. Otherwise, the transmission is not safe. Therefore, a data transmission operation to be consistent with $R$, if $\exists r_i, r_o \in R, J_i \cong J_o, P_i \neq P_o$ and $r_i \succeq pl_i, r_o \succeq pl_o$. If $P_i$ is sending information with attributes not in $A_o$, $P_i$ should do a projection operation $\pi_{A_o}(pl_i)$ first.

In the example, $r_8$ authorizes $P_E$ to get information on $(S \bowtie E \bowtie C \bowtie W)$. If $P_S$ sends the information of $r_{11}$ to $P_E$, it will not be allowed. Although the attribute set of $r_{11}$ is contained by $r_8$, there is no rule for $P_E$ to get data on the join path of $(E \bowtie S)$, and the data transmission is disallowed.

*Definition 4:* A query execution plan $pl$ is **consistent** with the given rules $R$, if for each step of operation in the plan is consistent with the given rule set $R$.

## IV. Checking rule enforcement

We introduce a few more concepts. A *join path can be enforced* if there exists a consistent plan for it which enforces all the key attributes of the relations in the join path. In some cases, a rule does not have a total enforcement plan, but only some partial plans. A **partial plan** only enforces a rule with an attribute set that is a proper subset of the rule attribute set. We say that an attribute set is a **maximal enforceable attribute set** for a rule, if it is enforced by a plan of the rule, and there is no other plan for the same rule that can enforce a superset of these attributes. If the maximal enforceable attribute set is equal to rule attribute set, the rule is **totally enforceable**.

*Lemma 1:* A rule has only one maximal enforceable attribute set.

*Proof:* Each rule on a basic relation only has one maximal enforceable attribute set since it is totally enforceable. A plan for a rule is composed by subplans from other rules, and we do not apply projections on any authorized attributes in the plans. Thus, the join attributes are always preserved in all plans. All plans for the same rule can be merged by joining over the key attributes. Therefore, the resulting attribute set is the only maximal enforceable attribute set for the rule. ■

### A. Finding relevant information

As discussed, it is desired to have a mechanism checking the rule enforceability for given set of rules so as to tell which queries can be actually answered. To check the enforceability of a rule, we check if the join path of the rule can be enforced, and examine what is the maximal enforceable attribute set of the rule. We propose a constructive mechanism that checks the rules in a bottom-up manner.

When examining a rule $[A_t, J_t, P_t]$, we call such a rule $r_t$ as *Target Rule*, $A_t$ as *Target Set*, $J_t$ as *Target Join Path*, and $P_t$ as *Target Party*. All the other parties are *Remote Parties*. To check the enforceability of $r_t$, we first find relevant information that can be obtained locally at $P_t$. If it is not enough, we check the information on remote parties. On the same party, we call a join path as a **sub-join path** of $J_t$ if it contains a proper subset of relations of $JR_t$. Rules not on the sub-join paths are not relevant to $r_t$ since any composition with these rules results in information more than what $r_t$ authorizes. At party $P_t$, a plan that is on a sub-join path of $J_t$ is a **relevant plan**, and the rule authoring it is a **relevant rule** of the target rule. Parties having rules defined on the equivalent join path of $J_t$ are called **$J_t$-cooperative parties**, and information regulated by $J_t$ is allowed to be exchanged only between these parties.

As it proceeds bottom-up, when examining a target rule with join path of length $n$, all the rules on join paths with smaller lengths have already been examined. We assume that each inspected enforceable rule is represented by an enforcement plan with its maximal enforceable attribute set, and we consider using these relevant plans to enforce the target rule being inspected. We say "join among rules" below, which refers to these enforcement plans. The iteration begins with the rules defined on the basic relations on various parties. These rules can always be enforced which only require data owners sending their data to the authorized parties. Next, the algorithm checks the rules in the order of join path length. Meanwhile, it also builds a graph structure capturing the enforceable information and the relationships among the rules. Each node in the graph is an enforceable rule with its maximal enforceable attribute set. All non-enforceable rules and attributes are discarded. Two nodes on the same party are connected if one is relevant to the other. Among different parties, nodes can be connected if they have equivalent join paths. Figure 2 shows the built graph for our running example. The different parties are separated vertically. The bold boxes show the basic relations owned by different parties. The algorithm starts the iteration with the rules on basic relations $r_1$, $r_2$, $r_4$, $r_{10}$, $r_{14}$ and so on.

### B. Checking local information for enforcement

When a target rule $r_t$ is inspected, the algorithm first checks its enforceability locally using relevant rules on $P_t$. After that, all the rules with equivalent join paths of $J_t$ on $J_t$-cooperative parties are locally checked respectively. Then the algorithm

checks the possible enforcement by exchanging information among these parties. In figure 2, on the level of join path length 2, the algorithm checks the rules with the order of $r_3$, $r_{12}$, $r_5$, $r_{16}$, $r_{11}$. $J_t$-cooperative parties such as $P_W$ and $P_S$ on $J_3$ will check the possible remote enforcement between $r_3$ and $r_{12}$. To check local enforceability, the algorithm finds its local relevant rules in the currently built graph structure. It only checks with the **top level rules** in the graph, which are the ones not connected to any higher level nodes (rules with longer join paths). For example, in figure 2, when the algorithm examines $r_{13}$ on $P_S$, $r_{11}$, $r_{12}$ are top level rules. We take advantage of the upwards closed property, so that the top level rules cover all possible join results among its lower level rules, and it has the superset of attributes of its relevant rules. If these top level rules cannot be composed to enforce the $J_t$, there is no need to check other rules locally.

The next step is to check whether the join path $J_t$ can be enforced locally by performing joins among these top level relevant rules. The algorithm checks each pair of these rules. We check it pairwise because if a pair of them can join, the result must be able to enforce $J_t$. To see if a pair of rules ($r_s$, $r_r$) can join, the algorithm first tests their relation sets to check if $JR_s \bigcap JR_r = \emptyset$. If they have overlapped relations, they can join over the key attribute of the overlap part, and $J_t$ can be enforced. Otherwise, we need to further check the attributes of two rules to see if they can perform the required join. If $J_t$ can be locally enforced, we mark $r_t$ as **local enforceable** rules. Otherwise, it has to wait and see if $J_t$ can be enforced on remote parties. Meanwhile, the algorithm also computes the union of the attributes from top level relevant rules. The resulting attribute set $A_r$ includes all attributes that can be obtained locally if $J_t$ can be enforced. It is always the case that $A_r \subseteq A_t$ since rules are upwards closed. If $A_r \neq A_t$, we call the set of attributes $A_t \setminus A_r$ as **missing attribute set** $A_m$. The attributes in $A_m$ can only be obtained from it $J_t$-cooperative parties. As in the example, the attribute $assistant$ in $r_8$ cannot be found in its top level rules $r_7$ and $r_5$, so it is a missing attribute.

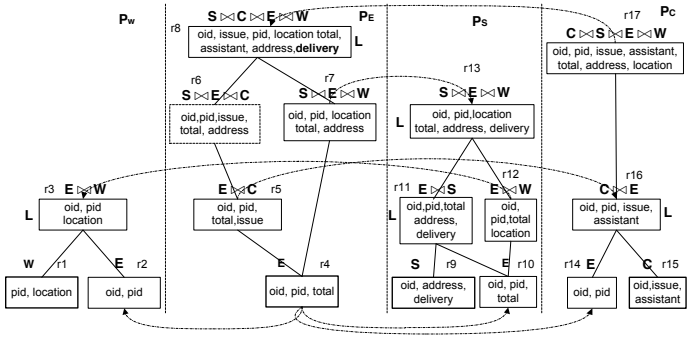


Fig. 2. Graph structure built for the example

### C. Checking remote information for enforcement

The algorithm then checks the remote information that party $P_t$ can use to enforce the rule $r_t$, so only $J_t$-cooperative parties are checked. As the algorithm already checked rules locally on $J_t$-cooperative parties, if there exists any party that can enforce $J_t$, then such a plan can be shared among all the $J_t$-cooperative parties. Thus, party $P_t$ can get attributes from all its $J_t$-cooperative parties to enforce $r_t$. We take the union of the attribute sets from all $J_t$-cooperative parties to check if $r_t$ can be totally enforced. If the missing attribute set $A_m \subset A_{r_1} \bigcup A_{r_2} ... \bigcup A_{r_k}$ (where $A_{r_i}$ is the relevant attribute set on a $J_t$-cooperative party $P_i$), then $r_t$ can be totally enforced. Otherwise, $A_m$ is updated by removing the attributes appear in any $A_{r_i}$. Consequently, $r_t$ has a node in the graph structure with the attribute set $A_t \setminus A_m$ which is its maximal enforceable attribute set, and edges are added among the $J_t$-cooperative parties. For example, attribute $delivery$ of $r_8$ cannot be found in its $J_t$-cooperative party $P_C$ either, so $r_8$ in the graph is represented with the attribute set without $delivery$. We use bold font in figure 2 to indicate it is not enforceable. In addition, since $J_6$ cannot be enforced at any party, $r_6$ is not enforceable, and it will not be included in the graph structure. Figure 2 gives the graph structure of the example built by our algorithm, and the dashed box shows $r_6$ is removed. The local enforceable rules are marked with "L", and the detailed algorithm is described in Algorithm 1.

**Algorithm 1** Rule Enforcement Checking Algorithm

**Require:** All given authorization rule set $R$ on all parties
**Ensure:** Find enforceable rules and build graph
Mark rules with length 1 as total enforceable rules
Get the maximal length of join path length $N$
**for** Join path of length 2 to $N$ **do**
  **for** Each join path $J_t$ length equal to $i$ **do**
    $AJ_t \leftarrow \emptyset$, the set of shared attributes on $J_t$
    **for** Each party $P_t$ has a rule $r_t$ on $J_t$ **do**
      Obtain the set of top level relevant rules $R_v$
      Add connection to these rules in graph
      $A_v \leftarrow$ the set of attributes in $R_v$
      Missing attribute set $A_m \leftarrow A_t$
      **for** Each pair of relevant rule $(r_s, r_r)$ **do**
        **if** $r_t$ is locally enforceable **then**
          $A_m \leftarrow A_m \setminus A_v$
      **if** $A_m \neq \emptyset$ **then**
        Put $r_t$ with $A_m$ into the Queue of $J_t$
      $AJ_t \leftarrow AJ_t \bigcup A_v$
    **for** Each rule $r_t$ in the Queue of $J_t$ **do**
      **if** $J_t$ can be enforced on some party **then**
        Add connection to remote party in graph
        $A_m \leftarrow A_m \setminus AJ_t$
        **if** $A_m \neq \emptyset$ **then**
          Replace $A_t$ with $A_t \setminus A_m$ in graph
      **else**
        $r_t$ cannot be enforced, remove it from graph
  Join path length $i$++

Assuming the total number of rules is $N_t$, and the maximum number of relevant rules of a rule is $N_o$, and checking attribute sets takes constant $C$. Then the worst case complexity for algorithm 1 is $O(N_t * N_o^2 * C)$, where $N_o$ is usually very small. The algorithm can be used as a pre-compute step once rules are given.

## V. Complexity of query planning

The above mechanism tells us which queries can be safely answered. However, we still need to generate the consistent

plans. From performance perspective, we always want optimal plans with minimal costs. Unfortunately, finding the optimal query plan is $NP\text{-}hard$ in our scenario.

*Theorem 1:* Finding the optimal query plan to answer an authorized query is $NP\text{-}hard$.

*Proof:* The optimization of set covering problem is know to be $NP\text{-}hard$. In the set covering problem, there is a set of elements $U = \{A_1, A_2, ..., A_n\}$, and there is also a set of subsets $S = \{S_1, S_2, ...S_m\}$ where $S_i$ is a set of elements from $U$ and is assigned a cost. The task is to find $C$ with minimal total cost that is a subset of $S$ and covers $U$. We can convert it into our cooperative query planning problem. Assuming there are two basic relations $R$ and $S$ which can join together, we map each element in $U$ into an attribute in relation $R$. Attribute $A_0$ is given to $R$ and $S$ as their key attributes. Thereby, $R$ has the schema $\{\underline{A_0}, A_1, A_2, ..., A_n\}$, and relation $S$ is $\{\underline{A_0}, A_{n+1}\}$. We then consider a query with the attribute set $\{A_1, A_2, ..., A_n\}$ on the join path of $R \bowtie S$. Thus, we can construct rules on $m + 1$ parties according to the set covering problem. Party $P_0$ has the rule with join path $R \bowtie S$ that authorizing the query. For each other party $P_i$, it has a rule $r_i$ on $R \bowtie S$ with the attribute set $S_i \bigcup \{A_0\}$. $P_0$ cannot locally do the join $R \bowtie S$, but other parties can enforce their rules $r_i$ locally, and their costs are known. Therefore, for $P_0$ to answer the query, it needs a plan bringing attributes from other parties and merging them at $P_0$ (multi-way join on attribute $A_0$) to answer the query. The optimal plan needs to choose the rules with minimal costs, and the union of their attribute sets must cover the query attribute set. If the optimal query plan can be found in polynomial time, the set covering problem also has a polynomial solution. Hence, finding the optimal query plan in our scenario is $NP\text{-}hard$. ■

### A. Query plan cost model

It is reasonable to assume that the numbers of tuples in the relations are known. In addition, the join selectivity between the relations are also known so that the size of join paths can be estimated as well. The cost of a query plan mainly includes two parts: 1) cost of the join operations, 2) cost of data transmission among the parties. We assume joins are done by nested loop and indices on join attributes are available. The cost of a join operation between $R$ and $S$ can be estimated as: $\alpha(Size(R) * Size(S) * P_{(R,S)}) + (Access(R) + Size(R))$, where $Size()$ is the number of tuples in the relation and $Access()$ is the cost of retrieving the relation. $R$ is the smaller relation, $\alpha$ is the cost of generating each tuple in the results, and $P_{(X,Y)}$ is the known join selectivity. The costs of data transmission are only decided by the size of the data being shipped. The cost of moving $R \bowtie S$ from a party to another is $\beta(Size(R) * Size(S) * P_{(R,S)})$, where $\beta$ is the per tuple cost for data transmission. Under such assumption, we can compare the costs of different query plans.

### B. Upper bound complexity of plan enumeration

Although finding an optimal query plan is $NP\text{-}hard$, we want to see if it is possible to enumerate all possible query plans and compare them to get the optimal one as the join path length in our scenario is usually limited. We assume the longest join path is 5, and we begin the process based on the graph produced by the previous algorithm. When a query comes in, we first filter out rules and attributes that are not relevant. In the classical query processing, the query attributes are always retrievable from the corresponding relations and usually the generated plan does not contain repeated joins (two join operands have overlapped relations). However, additional join operations may be required in our scenario because of the constraints of the rules. To enumerate the plans, we should not only list the different ways to perform the join operations, but also the different paths to retrieve the query attributes.

To generate a consistent plan for a query, we first need a plan that enforces the query join path. Once we have such a plan, it can further join with other plans to get all requested attributes. Thereby, we first enumerate different possible ways to enforce the query join path. As we consider the worst case scenario, we assume the query join path is length of 5, and we examine the possible last join operations among a pair of relevant rules to enforce the target join path. In the worst case, the possible pairs of rules with different join path lengths are $(3, 2), (3, 3), (3, 4), (4, 1), (4, 2)$, where two numbers in parenthesis are the lengths of two relevant rules. We cannot discard pairs with overlapped relations such as $(3, 3), (3, 4), (4, 2)$ because enforcing a longer join path may be more efficient than a shorter one in our scenario. Next, we need to recursively search for the possible ways to enforce the join paths in each possible join pair listed. We give the possible combinations for this recursive process below. Moreover, instead of counting the possible ways locally, we need to further consider the possibility that the join path is enforced via a remote party.

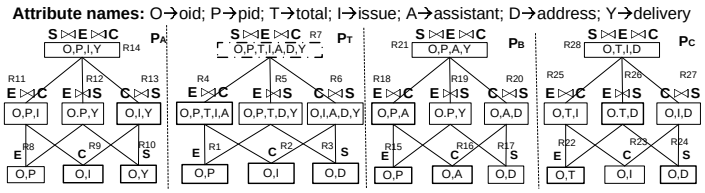


Fig. 3. A simple worst case example

We have to enumerate all possible ways of join path enforcement instead of keeping only the optimal way to do it because the optimal query plan does not necessarily have the optimal enforcement plan for the query join path. Besides the join path enforcement, additional steps are needed to retrieve the missing attributes which are query attributes but not enforced by the chosen join path enforcement plan. Different join path enforcement plans may result in different sets of missing attributes. Therefore, for each missing attribute set, we need to further enumerate the steps that retrieve these attributes via relevant rules on the cooperative parties. As each missing attribute may appear in multiple relevant rules, choosing the optimal set of relevant rules is similar to a set

covering problem. Thus, we need to enumerate all the possible sets of relevant rules that cover the missing attributes to find the optimal answer. Since it is similar to set covering problem, given $N_r$ relevant rules that have the missing attributes, there are $2^{N_r}$-1 combinations to check. In addition, once we select a set of rules, we still need to further find plans enforcing these rules to known their costs.

To illustrate the complexity, we construct a simple example with join path length of 3. In figure 3, there are four parties and they all have rules on different join paths. The attribute names are simplified to save space, and edges connecting the rules with equivalent join paths across the parties are also omitted to keep the graph clear. In the example, the query asks for all the attributes and only $r_7$ (dashed box) can authorize the query. For various plans enforcing the target join path, none of them can enforce all the query attributes. The possible ways to enforce the join path locally on $P_t$ is $3 * (1 + 2) = 9$. Considering other 3 parties, we have $(3 * 4 * (1 + 2 * 4)) * 4 = 432$ different ways of enforcing the join path, and these plans result in 6+4 =10 different missing attribute sets. For each of them, we need to check the ways to get missing attributes. For example, if the missing attribute set is $\{total, assistant, delivery\}$. Then, there are 12 relevant rules having the missing attributes, and the possible combinations to consider are $2^{12}$-1.

In table II, we list the maximum numbers of possible join path enforcement plans for each join path length. Notation $S_i$ indicates the number of plans for a join path of length $i$. We assume there are at most $T_n$ parties having the rules on equivalent join paths.

| **JoinPath Length** | **Maximum number of join path enforcement plans** |
|---|---|
| $S_1$ | 1 |
| $S_2$ | $T_n$ |
| $S_3$ | $(C_3^2 * S_2 * (1 + 2 * S_2)) * T_n$ |
| $S_4$ | $(C_4^3 * S_3 * (1 + C_3^1 * S_2 + C_3^2 * S_3) + C_4^2 * S_2 * S_2) * T_n$ |
| $S_5$ | $(C_5^4 * S_4 * (1 + C_4^1 * S_2 + C_4^2 * S_3) + C_5^3 * S_3 * (S_2 + C_3^1 * S_3)) * T_n$ |

TABLE II
MAXIMUM NUMBER OF PLANS FOR EACH JOIN PATH LENGTH

To sum up, in the worst case, to enumerate all the possible plans for a query, the total number of cases is:

$S_5 + N_e * (2^{C_m} * -1) * C_m * S_4$

$C_m$ is the number of missing attributes that should be the number of all non-key attributes in the worst case. $N_e$ is the different number of missing attribute sets based on all the join path enforcement plans that can be very large.

## VI. CONSISTENT QUERY PLANNING

Due to the difficulties in enumerating all possible ways of answering a query, we consider using a greedy algorithm.

### A. Query planning algorithm

To find an efficient consistent query plan, we always choose the optimal join path enforcement plan first, and then apply the set covering greedy mechanism on the missing attributes to find required relevant rules. The optimal enforcement plan for a join path on a specified party can be pre-determined by extending the rule enforcement checking algorithm in a dynamic programming way. When checking a rule $r_t$, instead of inspecting only top level relevant rules, all the possible join pairs at $P_t$ are inspected. These possible plans are compared and only the one with minimal cost is kept. As $J_t$-cooperative parties find their optimal local enforcement plans respectively, each party finds its optimal way of enforcing $J_t$. As discussed, the selected plan $pl$ usually results in a missing attribute set. To get these attributes, we explore the graph structure to decompose $r_t$ into a set of relevant rules that can provide these attributes. We record the required operations among these rules, and then recursively find ways to enforce these rules to generate a query plan.

Firstly, as the plan enforces $J_t$, it can be extended to get missing attributes that appear in the relevant rules of basic relations on all $J_t$-cooperative parties. This can be done through semi-join operations. In such cases, the party $P_t$ can send only the join attributes to the $J_t$-cooperative party, and the receiving party does a local join to get these attributes and sent it back. $P_t$ then performs another join to add these attributes to the query plan. In this way, we can reduce the missing attribute set by removing these attributes.

The remaining missing attributes can always be found in the relevant rules on $J_t$-cooperative parties. However, these relevant rules are defined on join paths instead of basic relations. Similar to the above case, the missing attributes carried by these relevant rules can be brought to the final plan by performing semi-join operations. Our next effort is to determine these relevant rules. Here, we always pick the relevant rule that covers the most attributes in the missing attribute set until all the missing attributes are covered by the picked rules. This is a greedy approach, and is similar in spirit to the approximate algorithms used for the set covering problem. The relevant rules effectively allow us to decompose the rule (i.e., express in terms of) rules with smaller join paths. The missing attributes are also reduced in the process by considering the rules involving basic relations. During the decomposition, the algorithm associates the set of attributes with the decomposed rule that are the missing attributes expected to be delivered by this rule. We record the operations between the existing plan and these decomposed ones. If they are on the same party, a join operation between them is recorded. Otherwise, a semi-join operation is recorded. Since each decomposed rule can be iterated decomposed, the algorithm uses a queue to process the rules until all the rules are on basic relations. This decomposition process gives the hierarchal relationships among rules that indicate how required attributes can be added to the final plan.

The decomposition process gives a set of rules, but we also need the subplans to enforce the join paths of these rules so as to generate a complete plan. To achieve that, we inspect the join paths of these decomposed rules from bottom-up. We use another priority queue to keep all the join paths from

the decomposed relevant rules, and the shortest join path is always processed first. This allows the use of results from the enforcement plans of sub join paths as much as possible. The algorithm uses the best enforcement plan for each join path as discussed. When an enforcement plan of a join path is retrieved, the algorithm combines previously recorded operations to generate the subplan for the decomposed rule on such join path. Finally, the algorithm finds the plans for each join paths in the queue, and generates the final query plan with a series ordered operations starting from the basic relations, and it is described in Algorithm 2.

**Algorithm 2** Query Planning Algorithm

**Require:** The structure of rule set $R$, Incoming query $q$
**Ensure:** Generate a plan answering $q$.

**if** There is a rule $r_t$, $J_t \cong J_q$ and $A_q \subseteq A_t$ **then**
  Missing attribute set $A_m \leftarrow A_q$
  Initialize queue $Q$, and priority queue $P$
  Enqueue $r_t$ to $Q$ with $A_m$
  **while** Queue $Q$ is not empty **do**
    Dequeue rule $r_t$ and the associated $A_m$
    **for** Each $J_t$-cooperative party **do**
      Finds the attribute set $A_b$ from basic relations
      $A_m \leftarrow A_m \setminus A_b$
      Record connections between $r_b$ and $r_t$
    **while** $A_m \neq \emptyset$ **do**
      **for** Each relevant rule $r_s$ on $P_{co}$ **do**
        Find the rule with max $A_m \bigcap A_s$
      Enqueue the rule $r_s$ with $\pi(A_m)$
      Enqueue the join path $J_s$ to priority queue $P$
      Record connections between $r_s$ and $r_t$
      $A_m = A_m \setminus A_s$
  **while** The priority queue $P$ is not empty **do**
    Dequeue the rule $r_s$ with join path $J_s$
    Add the path to enforce $J_s$ to plan
    **for** Each $J_s$-cooperative party **do**
      **if** The party has recorded $A_b$ on $J_s$ **then**
        Add ($\bowtie$ / $\rightarrow$) operations between $r_b$ and $r_s$
    **for** Each decomposed rule $r_d$ from $r_s$ **do**
      Add ($\bowtie$ / $\rightarrow$) operations between $r_d$ and $r_s$
**else**
  The query $q$ cannot be answered

*Theorem 2:* A query plan generated by *Query Planning Algorithm* is consistent with the set of rules $R$.

*Proof:* First of all, the subplans to enforce join paths are consistent. They are generated during the rule enforcement checking. Each join operation in such a plan is added according to a legitimate local join over the relevant rules, and each data transmission operation happens only between $J_t$-cooperative parties. In the iteration of decomposing rules, there are join and semi-join operations between the decomposed rules and the original rule. A join operation between a rule and its local relevant rule is always consistent. A semi-join between a rule and a relevant rule on its $J_t$-cooperative party is also consistent. It is because the attributes in the relevant rule can be obtained by the rule with $J_t$ on the same party, and the data transmission between two parties is consistent as they are $J_t$-cooperative parties and the original rule is always authorized to access these missing attributes. Since each operation in the plan is consistent, the plan generated by *Query Planning Algorithm* is consistent with the rule set $R$. ■

*Lemma 2:* The *Rule Enforcement Checking Algorithm* finds all consistently enforceable information.

*Proof:* As all the information can be obtained on join results comes from the basic relations, the algorithm works in bottom-up manner to capture all possibilities. If the join path of a rule cannot be enforced, then none of the rules on this join path can be enforced. Thereby, the algorithm first searches for all the possible ways for a join path to be enforced. As a join between a rule and its local relevant rule is always consistent, this step in the algorithm finds all the locally enforceable attributes. The only other information can be used to enforce $r_t$ must come from $J_t$-cooperative parties, and the algorithm considers these possibilities. There is no other way to enforce more information for $r_t$. ■

*Theorem 3:* For a query $q$ and a set of given rules $R$, if the *Query Planning Algorithm* does not give a query plan, then there does not exist a consistent query plan.

*Proof:* According to lemma 2, *Rule Enforcement Checking Algorithm* gives all the enforceable information. Thus, if there is an enforceable rule $r_t$ to authorize $q$, the *Query Planning Algorithm* can always generate a consistent query plan. Otherwise, $q$ cannot be answered safely. ■

### B. Preliminary performance evaluation of the algorithm

Since our query planning algorithm works in a greedy way, we want to evaluate the output results. Since the optimal plan cannot be found in general, we cannot compare our results with the optimal ones. Thus, we use simple examples, where manually finding the optimal plans becomes possible, and we perform preliminary evaluation on these cases. In the following, we assume the selected join path enforcement plan carries the maximal attributes along with it.

*1) Case 1:* Firstly, we can take a look at the example in figure 3. For simplicity, we assume all the relations have the same sizes. Given the same query discussed before which only $r_7$ can authorize, the optimal plan should be as follows: join two relations at $P_t$ first, and then join with the third one at $P_t$ to enforce the join path of $S \bowtie E \bowtie C$. Then $P_t$ sends the $oid$ on the join path of $S \bowtie E \bowtie C$ to other parties, and do semi-joins with each of the party to obtain the missing attributes $\{total, assistant, delivery\}$. Finally, $P_t$ does a local join with this information got from remote parties and such a plan answering the query. In this case study, our greedy algorithm generates the same optimal plan. The optimal way to enforce join path $S \bowtie E \bowtie C$ is the local enforcement at $P_t$, and our plan also gets the missing attributes via semi-join operations.

*2) Case 2:* In the example shown in figure 2, assuming the query has the join path $S \bowtie C \bowtie E \bowtie W$, and the attribute set includes all the attributes in $r_8$ except $delivery$. For such a query, our algorithm first finds the optimal way to enforce the join path, which can be represented as $(((r_1 \bowtie r_2 \rightarrow P_S) \bowtie r_9) \rightarrow P_E) \bowtie (r_{14} \bowtie r_{15} \rightarrow P_E)$. This plan results in a missing attribute set $\{total, assistant\}$. Next, the algorithm adds a local join with $r_4$ to retrieve $total$, and a semi-join with $r_{15}$ to obtain the attribute $assistant$. In fact, there are only two ways to enforce the query join path in this example.

The other way is to perform $r_9 \bowtie r_{10}$ first and then join with $r_{12}$ at party $P_S$. By doing that, the plan can carry the attribute $total$ and only has $assistant$ as missing attribute. However, if we compare the two plans, the difference is that our plan gets the attribute $total$ via a join among relation $E$ and join path $S \bowtie C \bowtie E \bowtie W$, and the latter plan perform the join among $E$ and $S$ on $P_S$. As the longer join path usually has much fewer tuples, and no matter the sizes of relation $S$ and $E$, the former plan is better than the latter one in this example case. For the missing attribute $assistant$, as it can only be retrieved from party $P_S$, getting it from $r_{15}$ is better than $r_{16}$. Therefore, the query plan generated by our algorithm is actually the optimal plan is this example case.

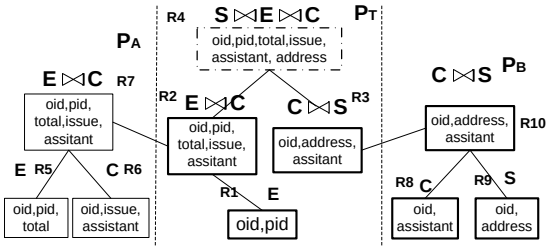


Fig. 4. A simple non-optimal example

*3) Case 3:* However, our algorithm cannot guarantee the generated plan is always optimal. In figure 4, we consider a query which is the same as $r_4$. The way to enforce the query join path $S \bowtie E \bowtie C$ in our generated plan is labeled with bold boxes. The other way to enforce it is to enforce $r_7$ at $P_A$ first, and send the results to $P_T$ to enforce $R_2$ and join with $R_3$. As the latter plan requires one more join and data transmission operation, our plan to enforce the query join path is better. However, the latter plan has no missing attribute, and our plan still need to enforce $r_7$ again to retrieve attributes $\{total, issue\}$. Therefore, our plan is not optimal in this case. Compared to the optimal plan, our generated plan just has one extra step which is $r_1$ join with $r_3$. Only in extreme situations, where the sizes of $E \bowtie C$ and $C \bowtie S$ are very large and $S \bowtie E \bowtie C$ is very small, our plan can be better.

To sum up, these simple example cases show our query planning algorithm is effective to find a good query plan for an authorized query.

*4) Complexity of the algorithm:* Assuming there are $N_q$ rules local relevant to the query $q$, the number of relevant rules on $J_t$-cooperative parties is $N_r$, and $C$ is a constant to record operations. The overall worst case complexity is $O(N_q * N_r^2 * C)$ which is $O(N^3)$($N$ is the total number of rules).

## VII. Conclusions and future work

In previous research work, a flexible data authorization model has been proposed to meet the security requirements for collaborative computing among different data owners in a collaborative environment. As authorization rules are made based on business requirements, it is possible that some rules cannot be enforced among the cooperative parties. In addition, a regular query optimizer cannot give consistent query plans under the constraints of these rules. In this work, we first propose an algorithm to check the enforceability of the given rules, and another algorithm to generate corresponding efficient consistent query plans for answerable queries.

For the future work, we will study the problem of making the unenforceable rules to be enforceable. We can consider using a trusted third party to enforce the rules, and we may also augment the given set of rules. Trusted third parties can be also used to improve the consistent query planning. To evaluate of our approaches comprehensively, we will study the cooperative relationships among enterprises in various real world scenarios, and test our mechanism under these cases.

## References


[1] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep A secret: A distributed architecture for secure database services. In *CIDR*, pages 186–199, 2005.

[2] R. Agrawal, D. Asonov, M. Kantarcioglu, and Y. Li. Sovereign joins. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 26. IEEE Computer Society, 2006.

[3] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. Rothnie, Jr. Query processing in a system for distributed databases (SDD-1). *ACM Transactions on Database Systems*, 6(4):602–625, Dec. 1981.

[4] A. Calì and D. Martinenghi. Querying data under access limitations. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 50–59. IEEE, 2008.

[5] S. Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the 7th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 34–43, 1998.

[6] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *International Conference on Database Engineering*, pages 190–200. ieee, 1995.

[7] V. Ciriani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Keep a few: Outsourcing data while maintaining confidentiality. In *ESORICS*, volume 5789 of *Lecture Notes in Computer Science*, pages 440–455, 2009.

[8] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Assessing query privileges via safe and efficient permission composition. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008*, pages 311–322.

[9] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Controlled information sharing in collaborative distributed query processing. In *ICDCS 2008*, Beijing, China, June 2008.

[10] D. Florescu, A. Y. Levy, I. Manolescu, and D. Suciu. Query optimization in the presence of limited access patterns. In *Proc. 15èmes Journées Bases de Données Avancées, BDA*, pages 41–60, 1999.

[11] J. Goldstein and P. Larson. Optimizing queries using materialized views: a practical, scalable solution. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 331–342.

[12] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[13] D. Kossmann. The state of the art in distributed query processing. *ACM Computer Survey*, 32(4):422–469, 2000.

[14] C. Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB Journal*, 12(3):211–227, 2003.

[15] R. Pottinger and A. Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB J*, 10(2-3):182–198, 2001.

[16] R. Sion. Query execution assurance for outsourced databases. In *VLDB*, pages 601–612. ACM, 2005.

[17] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for physical data independence. *The VLDB Journal*, 5(2):101–118, 1996.

[18] H. Yang and P. A. Larson. Query transformation for PSJ-queries. In *Proc. Int'l. Conf. on Very Large Data Bases*, page 245, Brighton, England, Aug. 1987.