# Query Consolidation: Interpreting a Set of Independent Queries Using a Multidatabase Architecture in the Reverse Direction

Aybar C. Acar and Amihai Motro

Department of Computer Science
George Mason University
Fairfax, VA
{aacar, ami}@gmu.edu

**Abstract.** We introduce the problem of *query consolidation*, which seeks to interpret a set of disparate queries submitted to independent databases with a single "global" query. This problem has multiple applications, from improving database design to protecting information from a seemingly innocuous set of apparently unrelated queries. The problem exhibits attractive duality with the much-researched problem of *query decomposition*, which has been addressed intensively in the context of multidatabase environments: How to decompose a query submitted to a virtual database into a set of local queries that are evaluated in individual databases. We set the new problem in the architecture of a canonical multidatabase system, using it in the "reverse direction". The process incorporates two steps where multiplicity of solutions must be considered: At one point the system must infer the most likely set of *equi-joins* for a set of relations; at another point it must discover the most likely *selection constraints* that would be applied to a relation. In each case we develop a procedure that ranks solutions according to their perceived likelihood. The final result is therefore a ranked list of suggested consolidations.

## 1 Introduction

Consider an individual who submits a set of queries to different databases, and then, off-line, consolidates the information obtained in a "big answer" of some sort. Because the information this user requires is dispersed over multiple databases, the user is forced into a laborious process of submitting individual queries to different databases and then correlating and assembling the information off-line. Discovering a single interpretation for his entire query set may help suggest how information could be *reorganized* to facilitate similar tasks in the future. Indeed, the main argument for constructing virtual databases has always been to provide in a single source all the information necessary for a particular task [16]. Thus, discovering interpretations for distributed query sets may suggest useful reorganizations and consolidations, either physical or virtual.

As an analogy, consider a shopping center with multiple stores, and assume that an analysis of sale records shows that within a small time interval, the same customer purchased a box of candy in one store, gift-wrapping paper in another, and a greeting card in a third. A global interpretation of this local information may suggest that service could be improved if these three items where to be sold in the same store. Similarly, query consolidation may suggest redesign of available information sources to correspond more efficiently to popular information needs.

A different, though not entirely unrelated, reason for interpreting query sets, in either the distributed or centralized cases, is *user inexperience or ignorance.* In the distributed case, the user might be submitting a set of queries to different databases and correlating them off-line, when the same goal could be achieved by accessing a single database. In the centralized case, the user might be submitting a set of small queries and assembling them off-line, when the same goal could be achieved with a single query, perhaps using a feature of which the user is not aware. A query consolidation analysis may suggest flaws in the way the system is advertised or in the training of its users. This application is reminiscent of other systems that track user behavior and suggest improvements, such as office software or on-line stores.

Returning to the analogy of the shopping center, the reason for the individual purchases could be that the customer may be trying to *hide* his overall purpose. Accordingly, a possible application of query consolidation is surveillance and security: A consolidated query discloses the intentions of the user posing the queries. While the elucidation of these intentions from consolidated queries is a task for human experts, a query consolidation system can do the preparatory work. Since there could be a large number of users each with multiple queries, the function of the query consolidator will be to sift through the logs, compile likely consolidations, and present them to the expert for judgement. A variety of options are available: The expert can focus on a single user and get a listing of interests during a time period. Alternatively, trends can be analyzed across many sources looking for intentions shared by a group of users. Query consolidation can also be useful as a detection mechanism when the possible intentions and the global queries that imply them are known in advance. Then, an operator can set up the system so that certain information is on a watch-list and any consolidation of queries that significantly overlaps that information is flagged automatically by the system, along with the users who posed these queries. An earlier attempt at security-inspired query consolidation, albeit using a different approach, can be found in [2].

We propose to address the problem of interpreting distributed sets of queries, by using the well-researched architecture of *virtual databases* [17]. Briefly, a virtual database architecture integrates a set of *local* databases by means of a *global* database scheme which is *mapped* into the local databases. A query submitted to the virtual database (based on the global scheme) is *decomposed* into queries against the local databases, and the corresponding answers are assembled in an answer to the original query. The entire process is transparent to the user.

Query decomposition is summarized thusly: Given a global query $Q$, find local queries $Q_1, \ldots, Q_n$ and an assembly expression $E$ such that $Q = E(Q_1, \ldots, Q_n)$.

For our purpose here of interpreting a set of local queries, we adopt the same architecture, but consider a process that is the *reverse* of query decomposition, and which we name *query consolidation*: Given local queries $Q_1, \ldots, Q_n$, find a global query $Q$ and expression $E$, such that the query decomposition procedure will decompose $Q$ into $Q_1, \ldots, Q_n$ using $E$, so that $Q = E(Q_1, \ldots, Q_n)$.

The main obstacle here is that whereas query decomposition is usually a *function* (it is a deterministic process in which each query generates a unique decomposition), it is not *injective*. That is, there could be multiple global queries $Q^1, \ldots, Q^m$ and corresponding expressions $E^1, \ldots, E^m$, such that the query decomposition procedure will decompose $Q^i$ into $Q_1, \ldots, Q_n$ using $E^i$ (for $1 \leq i \leq m$).

Our approach to this new problem can be sketched as follows. We assume that the independent databases to which queries are submitted have been incorporated into a virtual database system. Under assumptions of sufficiency (the given query set includes all the information necessary to achieve the goal) and necessity (it includes only information necessary to achieve the goal) we "reverse" the query decomposition process. The process incorporates two steps where multiplicity of solutions must be considered: At one point the system must infer the most likely set of *equi-joins* for a set of relations; at another point it must discover the most likely *selection constraints* that would be applied to a relation. In each case we develop a procedure that ranks solutions according to their perceived likelihood. The final result is therefore a ranked list of suggested consolidations.

The focus of this paper is on the definition of the new problem and its applications, its setting in a virtual database architecture, and the methodology of its solution; detailed discussions of the algorithms, the software prototype, and results of experimentation are largely omitted, for reasons of space. The paper is organized as follows. Section 3 provides the formal framework for this work, Section 4 details the solution methodology, and Section 5 concludes with a brief summary and discussion of work in progress. We begin with a brief review of related work.

## 2 Background

The work presented in this paper draws from a diverse range of subjects, including information integrating systems (multidatabase systems) and query decomposition, join inference, and association analysis.

### 2.1 Information Integration Systems

An information integration system combines information from a heterogeneous collection of autonomous information sources. The integrating site is often referred to as *global*, and the individual sources are termed *local*. There have been many different models and architectures for information integration systems. Of interest to us here are systems that follow the architecture of *virtual databases*.

A virtual database has a database scheme (a *global* scheme), but no database instance. Instead, it has information that maps the global scheme into schemes of local databases. The materialization of a global query is done by translating the query into multiple subqueries to be materialized at the local sources and shipped back to the integrator for assembly. Virtual databases can be classified by the type of their global-local associations [8]. This classification distinguishes between architectures in which the local database schemes are defined as views of the global scheme (termed Local-as-View or LAV), and architectures in which the global scheme is defined as views of the local schemes (termed Global-as-View or GAV). An example of the former type are The Information Manifold [11]. Examples of the latter type are SIMS [4], TSIMMIS [7] and HERMES [22]. The architecture of Multiplex [17] is more powerful in that it associates views of the global schema with views of the local schema's. This hybrid approach earned the term GLAV.

A primary concern in virtual database systems is the process of query decomposition: The translation of a global query to a set of local queries. The main problem here is the need to rewrite queries defined over relations to queries over views of these relations (this is especially difficult for LAV systems) [8]. Optimization is also challenging because statistical information on local data is often unavailable. Finally, the decomposition procedure may have to account for temporary unavailability of some data, or multiple, inconsistent copies of other data [19, 18].

One of the main obstacles to the usability of relational databases among naive users is the difficulty of performing joins. Much effort has been invested over the years to simplify this operation, often by inferring joins "automatically". An early endeavour in this respect was the universal relation model [13]. The universal relation model attempts to make the joins among relations in a database transparent by automatically traversing the scheme through join dependencies. Another approach to the problem of identifying the join path intended by the user assumes the path with the lowest cost tends to be the correct answer [23, 15]. Here, the cost is computed by reducing the problem to a minimum directed cost Steiner tree problem and edge costs are defined in terms of the cardinality of the relationship. The Discover system, described in [9], uses keyword-based queries. Once the keywords are located in the various relations of the database, these relations are connected through their primary-foreign key relationships. Another query interface, INFER [14], generates and ranks the top-k join possibilities and allows the user to select the one intended before materializing the query. The results are ranked by prioritizing shorter join sequences over longer ones and lossless joins over lossy joins.

Association analysis or association rule mining has been an active field for more than a decade. Association analysis mines a set of transactions in a database to find rules that generalize the associations among the items in the transactions. The major problem in association analysis has been the complexity of finding frequent item sets in a set of transactions. While finding individual items occurring frequently in the orders is rather trivial, when all possible sets that can be built from these items are considered, the problem is time consuming indeed. Hence, association analysis algorithms all aim to prune the search space to manageable proportions. Most of these algorithms are based on the fact that the frequency of occurrence of a set of items is anti-monotone with respect to its subsets. Apriori [3], is an example of a breadth-first counting algorithm. It is the first algorithm to utilize the anti-monotone property of support. Apriori works in a breadth-first manner, counting each level in one pass of the transaction database. It is therefore possible to prune any k-item sets without counting them if any of their subsets are infrequent. If one requires only the maximal frequent item sets, depth-first analysis tends to be faster in finding the pruning boundary. Also, with maximal frequent item sets look-aheads and neighbor branch pruning is also possible. A good example of an algorithm that exploits these advantages is MAFIA [6].

## 3 Formal Framework

The formal framework for this research consists of three parts: (1) A statement of the problem, (2) a description of a "generic" virtual database architecture and query decomposition procedure, and (3) assumptions on the sufficiency and necessity of the given queries for the overall goal.

### 3.1 The Problem

A virtual database architecture consists of a set of *local* databases $D_1, \ldots, D_n$, a global database scheme $D$, and a mapping of the global scheme into the local databases. The main service of this architecture is *query decomposition*:

| |
|---|
| Given a global query $Q$, find local queries $Q_1, \ldots, Q_n$ and expression $E$ such that $Q = E(Q_1, \ldots, Q_n)$. |

Query decomposition can be viewed as a *function* that assigns each query $Q$ a *unique* set of queries $Q_1, \ldots, Q_n$ and suitable assembly expression $E$.

The problem of *query consolidation*, which is the subject of this paper, is defined as the *reverse* of the query decomposition problem:

| |
|---|
| Given local queries $Q_1, \ldots, Q_n$, find global query $Q$ and expression $E$ such that the query decomposition procedure will decompose $Q$ into $Q_1, \ldots, Q_n$ using $E$, so that $Q = E(Q_1, \ldots, Q_n)$. |

The solution to the problem as stated is not unique. That is, there could be multiple global queries $Q^1, \ldots, Q^m$ and corresponding expressions $E^1, \ldots, E^m$, such that the query decomposition procedure will decompose $Q^i$ into $Q_1, \ldots, Q_n$ using $E^i$ (for $1 \leq i \leq m$). We address this issue in Section 3.3.

### 3.2 The Multiplex Model for Virtual Databases

To solve the query consolidation problem we must adopt a virtual database model. Many different architectures have been proposed for virtual databases, and we adopt the Multiplex architecture [17]. The advantages of Multiplex that are attractive include its simplicity and generality. Simplicity is due to the fact that Multiplex assumes that all databases are in the well-known relational model, without introducing any new concepts or structures. Generality is achieved by the method in which the global and local databases are associated, namely by arbitrary view pairs.

We begin by defining the language for all queries and views. We assume the subset of the relational algebra defined by the operators selection, projection and Cartesian product (SPC)[1], with selections that are purely conjunctive. Although this family of queries is a restricted subset of the algebra (i.e., it excludes union, difference, non-conjunctive selections), it is often considered adequately expressive for the large portion of queries used in the real world [12]. In has been shown that any expression in this language can be written in the form[2]:

$$Q = \pi_A \sigma_C (R_1 \times R_2 \times \ldots \times R_n) \tag{1}$$

Assuming expressions in this form often simplifies discussions and proofs.

A Multiplex database is:

1. A global database scheme $D$,
2. A set $D_1, \ldots, D_n$ of local database schemes, and their associated database instances $d_1, \ldots, d_n$, and
3. A set $(V_1, U_1), \ldots, (V_m, U_m)$ of view pairs, where each $V_i$ is a view of the global scheme $D$, and each $U_i$ is a view of one of the local schemes.

Thus, the global database scheme $D$ has no associated database instance. Instead, there is a collection of views of this scheme that are materialized using the corresponding local views, i.e., the instance of the global view $V_j$ is materialized by the instance of the view $U_j$ (in the appropriate local database).

Assume a virtual database as previously defined, and let $Q$ be a query submitted to its scheme $D$. The decomposition of $Q$ can be outlined in this 7-step procedure:

1. Create a global relation scheme $R$ for the Cartesian product operations in $Q$.
2. Determine the views $V_j$ that are relevant to $Q$ (i.e., overlap with $Q$).
3. Construct queries $Q_i$ to retrieve from the corresponding local views $U_j$ the parts that are relevant to $Q$.
4. Evaluate $Q_i$ in the local databases, obtaining answers $A_i$.
5. Extend $A_i$ with nulls, creating instances $\bar{A}_i$ of scheme $R$.
6. Coalesce the instances $\bar{A}_i$ to a single instance $\bar{A}$.

[1] Or, equivalently, selection projection, join, rename (SPJR).

[2] See [1] for proof.

7. Apply $Q$'s selection and projection operators, yielding an answer $A$ to the query $Q$.

As described in step 3, the local query $Q_i$ retrieves only part of the view $U_j$. If this cannot be accomplished due to local limitations, then $Q_i$ would have to retrieve all of $U_i$, and the answer $A_i$ would have to be processed to extract the part relevant to $Q$.

## 3.3 Assumptions on Sufficiency and Necessity

We interpret the consolidating query $Q$ as the *goal* of the user in submitting the queries $Q_1, \ldots, Q_n$. This assumes that the user is not using information obtained elsewhere to achieve his goal. In other words, we adopt a principle of *sufficiency*: The information in the local queries $Q_1, \ldots, Q_n$ is sufficient to achieve the goal, and hence can be approximated by an appropriate consolidation.

Recall that we characterized query decomposition as a procedure with a unique outcome. Consider a simple global query that retrieves a single value such as a person's age. Obviously, there could be multiple correct decompositions. For example, the local query $Q_i$ could retrieve just the person's age; or it could retrieve that person's entire tuple, and the expression $E$ would project the age; or it could retrieve the tuples of multiple persons and $E$ would select that person's tuple and project the age. The guiding principle of the query decomposition procedure is to retrieve from the local databases as little as possible, taking advantage of the local system's query processing capabilities. This reduces possible costs charged by the local database, as well as the costs and time of transmitting the data. Hence, the decomposition adopted is one that *optimizes* the process.

A similar principle will guide our query consolidation procedure. In the previous example, assume a given local query $Q_i$ that retrieves tuples of multiple persons. From this, one could conclude a global query $Q$ that needs all this information; or one that selects a single tuple from the set; or one that extracts the age of a particular person. A principle that guides the query consolidation procedure is that of *necessity*: All the information given in the queries is assumed to be necessary for the global query. The consolidation necessity principle is similar to the decomposition optimality principle: both assume that *all* information extracted from local databases is necessary, either to answer $Q$ (decomposition) or to conclude $Q$ (consolidation).

We note that both assumptions are at times unjustified. The user may have some additional information that may be instrumental in achieving his goal. Or he may submit non-optimal queries that retrieve unnecessary information (or he may be dishonest, attempting to to hide his true goals). We discuss such situations in Section 5 where we outline on-going and future work.

Note that while the necessity principle limits the problem space considerably, it does not generate unique consolidations. This issue is addressed next.

## 4 Methodology

In rough strokes, our overall approach may be sketched as follows. We assume a virtual database is available that integrates local databases $D_1, \dots, D_n$ in a global scheme $D$. Given local queries $Q_1, \dots, Q_n$, we follow a procedure that roughly reverses query decomposition:

1. For each local query $Q_i$, determine the views $U_j$ that are relevant (that overlap with $Q_i$).
2. Process the answers $A_i$ to obtain the part $\bar{A}_i$ that is within $U_j$.
3. In the virtual database, materialize the corresponding views $V_j$ with the answers $A_i$.
4. Populate the relations $R_k$ with materialized views $V_j$.

As described in steps 1 and 2, it is possible that a local query $Q_i$ would not be contained in a local view $U_j$, causing some data to be discarded when global structures are populated. As this will decrease the effectiveness of the consolidation, we assume that all local queries are contained in mapped views.

Let $R_k$ be the global relations populated by at least one view $V_j$. These relations must now be joined meaningfully. If a view $V_j$ joins two (or more) of these relations, then a join is implied. Hence, the relations $R_i$ are *clustered* with implied joins, but the clusters still need to be joined.

Assume now that a decision has been made on the remaining joins. A single relation scheme is thus obtained. If it includes attributes that are not in any of the views $V_j$, they are removed. Denote the resulting scheme $R$. The global query $Q$ is assumed to be embedded in $R$.

We now consider processing $R$ by selection and projection, as the canonical representation of $Q$ suggests (Equation 1). The necessity assumption implies that this relation should not be subjected to any selections based on constants, as these should have been done in the local queries. Similarly, the necessity assumption implies that this relation should not be subjected to projections, as these could have been done in the local queries as well.[3]

The multiplicity of possible consolidations is therefore due to two sources:

1. The given relations may be joined in different ways.
2. The resulting relation could be subjected to different *global selections* (selections that compare attributes retrieved from different local queries).

We handle these issues consecutively. First, we generate all the possible join plans and *rank* them according to plausibility. Then, for each join plan, we suggest possible global selections, ranking them as well.

[3] Possibly, some join attributes may not be required in the ultimate query $Q$, but we shall ignore this possibility for now.

## 4.1 Inferring Joins

Upon materializing the views $V_j$ from the received answers $A_i$, and then populating the relations $R_k$ with these views, we find that a view may be contained in a relation, or several views may be contained in the same relation, or a view may span several relations. Consider the example in Figure 1: $V_1$ spans relations $R_1$ and $R_2$, both $V_2$ and $V_3$ are contained in $R_3$, and $V_4$ is contained in $R_4$. The task now is to join the relations that received data; all other relations are ignored.

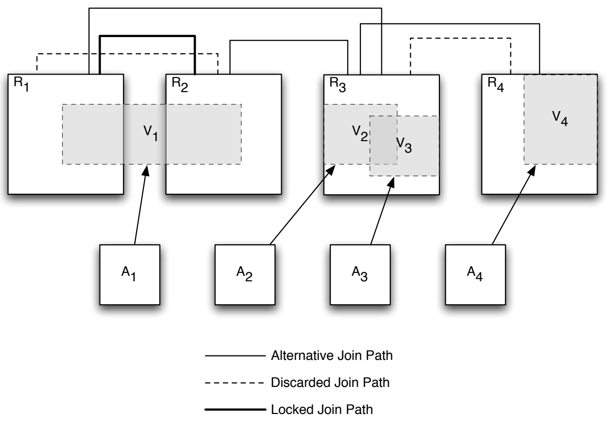

**Fig. 1.** View Mapping and Join Paths

The global scheme contains information (essentially, foreign key constraints) that establishes possible relationships among its relations. Figure 1 also shows the relationships among the four relations. Initially, we ignore the relationships that cannot be used because none of their attributes were materialized. These are shown as dashed lines. Furthermore, any relations that are spanned by a single view are considered to be joined unambiguously. Therefore, the join implied by the spanning view is "locked" and all its alternatives (i.e., other join paths between the two relations) are ignored. Thus, we obtain a graph in which vertices are relations and edges are possible joins. The join graph for the example is given in Figure 2. Locked joins are shown in bold lines.

A join plan is a tree that spans this graph.[4] We can therefore obtain all the possible join plans by enumerating the *spanning trees* in the join graph. To rank these plans with respect to plausibility, we assign a score to each tree: We assign a weight to each edge, and, as all spanning trees have the same number of edges, the score of a tree is the sum of its edge weights. Mandatory edges indicating

[4] Although join plans that include cycles are possible, we consider them to have low plausibility.

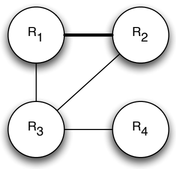

**Fig. 2.** Join Graph

locked joins (such as the one between $R_1$ and $R_2$) are handled by merging their two end vertices.

We now describe a method for assigning weights to edges, to indicate the plausibility of the corresponding joins. Our fundamental assumption is that joins over foreign keys are to be preferred, and when foreign key information is not available, or when the data retrieved does not obey foreign key constraints, extension-based relationships that most resemble foreign keys are to be preferred. Hence our method *quantifies* the levels to which attribute relationships obey referential constraints. The method is based on the concept of *entropy* in information theory.

Consider a relation $R$ with attribute $A$, and let $Dom(A)$ be the set of distinct values in attribute $A$. The entropy of $A$ is defined as

$$H(A) = \sum_{i \in Dom(A)} -p(i) \log_2 p(i)$$

$p(i)$ is simply the proportion of tuples in which it occurs. Intuitively, $H(A)$ measures the uniformity of the distribution of the values of $A$. Assuming $n$ distinct values, entropy ranges between 0 and $\log_2(n)$. The maximal value corresponds to perfect uniformity of distribution (lowest information content); for example, when $dom(A)$ includes 4 distinct values, and each occurs 5 times, then $H(A) = \log_2(4) = 2$. In this case, it is the number of bits required to represent the values of $dom(A)$. Hence, entropy is measured in *bits*. We define the entropy of a relation $R$ as the *sum* of the entropies of its attributes.

Assume now that attribute $A$ is used to partition the tuples of $R$ into several non-overlapping sets (the tuples in each set have the same value of $A$), calculate the entropy of each *slice* of the partition, and then average these slice entropies. The value obtained is the average entropy of this partition by $A$:

$$H_A(R) = \frac{\sum_{i \in Dom(B)} H(\sigma_{A=i}(R))}{|Dom(B)|}$$

In our case, assume attribute $A$ participates in a join with an attribute from another relation, say $S.B$. This join induces a partition of $R$ by attribute $A$, and the average partition entropy is therefore associated with that join. We refer to this as *posterior* entropy. Finally, we combine the *apriori* entropy of $R$ and its posterior entropy in an expression that measures the relative entropy reduction, or *information gain*, that can be attributed to the join:

$$I_B(R) = \frac{H(R) - H_B(R)}{H(R)}$$

Note that a join between $R$ and $S$ on attributes $A$ and $B$, respectively, modifies the entropy of both $R$ and $S$. That is, the join is associated with two different information gain values: $I_B(R)$ and $I_A(S)$. We assign the higher of these as the weight of the join edge.[5]

We illustrate these definitions with five short examples. Consider relations

$$\begin{aligned} R(A, B) &= \{(1, b_1), (2, b_2), (3, b_3), (4, b_4)\} \\ S(A, C) &= \{(2, c_1), (3, c_2), (4, c_3), (5, c_4)\} \end{aligned}$$

and a join between $R.A$ and $S.A$. It is a one-to-one matching and it results in three tuples. Consider now the effect of the join on the entropy of $S$. Initially, the entropy of the attributes of $S$ are $H(S.A) = 2$ and $H(S.C) = 2$, and the apriori entropy is therefore $H(S) = H(S.A) + H(S.B) = 2 + 2 = 4$. When the join iterates over the four values of $R.A$ it creates in $S$ four slices: $A = 1$ creates an empty slice, $A = 2$ creates $\{(2, c_1)\}$, $A = 3$ creates $\{(3, c_2)\}$, and $A = 4$ creates $\{(4, c_3)\}$. Each slice has entropy $0 + 0 = 0$, and the posterior entropy is therefore $H_A(S) = (0+0+0+0)/4 = 0$. Consequently, the information gain for $S$ from this join is $I_A(S) = (4 - 0)/4 = 1$. In this case, the information gain for $R$ from this join would be identical: $I_A(R) = 1$.

As a second example, consider

$$\begin{aligned} R(A, B) &= \{(1, b_1), (2, b_2), (3, b_3), (4, b_4)\} \\ S(A, C) &= \{(2, c_1), (2, c_2), (3, c_3), (3, c_4), \\ &\qquad (4, c_5), (4, c_6), (5, c_7), (5, c_8)\} \end{aligned}$$

and a join between $R.A$ and $S.A$. It is a one-to-many matching in which every tuple of $R$ matches zero or two tuples of $S$, and it results in six tuples. The apriori entropy of $S$ is $H(S) = H(S.A) + H(S.B) = 2 + 3 = 5$. When the join iterates over the four values of $R.A$ it creates in $S$ one empty slice and three slices with two tuples each: $\{(2, c_1), (2, c_2)\}$, $\{(3, c_3), (3, c_4)\}$ and $\{(4, c_5), (4, c_6)\}$. The empty slice has entropy 0, and each of the other three slices has entropy $0 + 1 = 1$, and the posterior entropy is therefore $H_A(S) = (0+1+1+1)/4 = 0.75$. Consequently, the information gain for $S$ from this join is $I_A(S) = (5 - 0.75)/5 = 0.85$. The information gain for $R$ from this join would be $I_A(R) = 1$.

Next, consider a join between

$$\begin{aligned} R(A, B) &= \{(1, b_1), (2, b_2), (3, b_3), (4, b_4)\} \\ S(A, C) &= \{(2, c_1), (2, c_2), (3, c_3), (4, c_4), (4, c_5), (5, c_6)\} \end{aligned}$$

[5] The spanning tree algorithm uses the lower values to resolve ties.

It is a one-to-many matching in which tuples of $R$ match different numbers of tuples of $S$, and it results in 5 tuples. The apriori entropy is $H(S) = 4.5$ and posterior entropy is $H_A(S) = 0.5$. The information gain for $S$ is $I_A(S) = 0.89$. The information gain for $R$ would be $I_A(R) = 1$.

Next, consider

$$\begin{aligned} R(A, B) &= \{(1, b_1), (1, b_2), (2, b_3), (2, b_4), \\ &\quad (3, b_5), (3, b_6), (4, b_7), (4, b_8)\} \\ S(A, C) &= \{(2, c_1), (2, c_2), (3, c_3), (3, c_4), \\ &\quad (4, c_5), (4, c_6), (5, c_7), (5, c_8)\} \end{aligned}$$

It is a many-to-many matching in which every tuple of $R$ relation matches zero or two tuples of $S$. Both information gains are identical: $I_A(S) = 0.85$ and $I_A(R) = 0.85$.

Finally, consider

$$\begin{aligned} R(A, B) &= \{(1, b_1), (2, b_2), (2, b_3), (2, b_4), \\ &\quad (3, b_5), (4, b_6), (4, b_7), (4, b_8)\} \\ S(A, C) &= \{(2, c_1), (2, c_2), (3, c_3), (3, c_4), \\ &\quad (4, c_5), (4, c_6), (4, c_7), (5, c_8)\} \end{aligned}$$

It is a many-to-many matching in which tuples of $R$ match different numbers of tuples of $S$. The information gains are $I_A(S) = 0.82$ and $I_A(R) = 0.83$.

The examples demonstrate how the method is sensitive to the selectivity of the join (on both participating relations). A join matching a single tuple is scored with perfection, and as the average number of tuples matched increases, the score decreases. Therefore, on average, a one-to-$n$ join will be scored higher than a one-to-$m$ join when $n < m$.

This method is an information theoretic way of *quantifying* referential constraints. If a foreign attribute has an information gain of 1 over a relation, that relation is functionally dependent on the attribute. A one-to-one relationship is one where gains in both directions are 1. A one-to-many relationship has gain of 1 in one direction. Indeed, this approach generalizes the definition of dependency from a binary concept to a gradual one. The more an attribute acts like key, the closer its gain will be to 1. Conversely, an attribute that has no selectivity at all will have a gain of 0.

Once the weights are assigned to each edge, the enumeration of spanning trees can be done by a variety of algorithms. We use the algorithm reported by Kapoor and Ramesh [10], which allows the enumeration of an arbitrary number of spanning trees in order of total weight (i.e., the top-k trees can be listed without necessarily enumerating every candidate).

### 4.2 Inferring Global Selections

Once a viable join is found among the relations and the irrelevant attributes are removed, a single relation $R$ is obtained that encompasses the information retrieved by the user from the various sources. Our sufficiency principle guarantees that the sought-after goal of this user is embedded in this relation. As previously discussed, to achieve his goal, the user who gathered this information could apply further operations to $R$. Yet, the principle of necessity implies that these operations are global selections: comparisons between attributes that were retrieved from different sources.

Domain information available in the global scheme reduces the number of possible global comparisons, as the pairs of attributes that can be compared are known. Nonetheless, the number of possible comparisons is still prohibitively large.

Our approach to the problem of inferring likely global comparisons is to extract pertinent knowledge from the *query repository*. This repository is a log of queries that have been previously submitted to this virtual database, and thus includes information on likely global comparisons. We shall refer to this repository as our *training set*.

Referring to the tax example, if there are a significant number of examples in our training set that project the attributes ( *TaxOwed*, *TaxWithheld* ), and a significant portion of these also include the selection predicate *TaxWithheld* > *TaxOwed*, then we can infer the rule:

$$\Pi(TaxWithheld) \wedge \Pi(TaxOwed) \rightarrow \\ \sigma(TaxWithheld > TaxOwed)$$

A similar problem has been researched extensively in the area of data mining. The goal of *association analysis* is to analyze sets of transactions to discover frequent item sets. The classical example is market basket analysis, where purchase records of retailers are mined to find out which products are purchased together (e.g., beer and peanuts).

In analogy, we mine our training set of queries to find out which attributes are frequently projected together. Furthermore, we would like to determine whether the fact that a set of attributes is projected also implies a comparison. We therefore mine our training set of queries for sets of projected attributes that have at least a certain degree of support.[6] The threshold for support depends on many factors, including the number of attributes in the domain, the size of the training set, and the extent of generality desired.

In our case, we prefer to set the support level so that the total occurrences of the set of projected attributes is above some absolute number, a number that reflects significant interest in those attributes. Consequently, as the training set gets larger (while the threshold of occurrences is maintained), the level of support decreases. The result is an increase in the number of rules generated and in the time required for training. Hence, the support threshold is a compromise between the desire to discover all the significant attribute sets, and practical considerations of rule-base size and time.

[6] When a set of $n$ cases suggests a rule or association of the type $\alpha \rightarrow \beta$, the ratio $\frac{|\alpha|}{n}$ is the *support* of the rule, and the ratio $\frac{|\alpha|}{|\beta|}$ is its *confidence*.

Once a threshold is set, a standard algorithm is used to find maximal frequent item sets (we use MAFIA). Yet, a difference between our case and that of standard association analysis should be pointed out. In our case, the item sets must have *two distinct* and *non-empty subsets*: a set of projected attributes and a set of selection constraints. This requirement results in substantial reduction in the time needed for the algorithm.

The discovered item sets generate rules much like in standard association rule mining. Each item set is partitioned into a rule such that, given the items in its antecedent (the projected attributes), the queries in the training set have at least a minimum probability of having its consequent (a comparison or conjunction of comparisons).

The rule base thus mined is used to finalize candidate consolidations with the most likely global selections, as follows. Once a single relation has been formed by the appropriate joins, its attributes are compared against the rule base. When the attributes match the antecedent of a rule, the selection constraint of its consequent is retrieved. These possible completions of the query are ranked by the confidence of the rule.

## 5 Conclusion

We described a new problem, which we termed *query consolidation*. Query consolidation seeks to interpret a set of disparate queries that were submitted to independent databases with a single global query: A query that expresses the ultimate goal of the user. Setting the problem in the architecture of a virtual database, it exhibits attractive duality with the much-researched problem of *query decomposition*.

We assumed that the independent databases to which the component queries are submitted are "monitored" by means of a virtual database. Since the same set of queries could be consolidated in different global queries (all of which will decompose back to the same component queries), our solution *ranks* the possible consolidations. The rankings are derived from our own treatment of the problems of join inference and selection constraint discovery.

The assumption that the databases had been integrated previously in a virtual database implied the existence of a global scheme. This scheme provided semantic associations among the individual queries, and thus simplified the task of consolidation. A more challenging situation is when such a virtual database had not been constructed. In this situation the extensions of given queries must be analyzed to infer their semantic associations, a task reminiscent of the well-known scheme-matching problem [20, 5].

Much of the research described in this paper has been completed, and a prototype system has been implemented. Indeed, the research and implementation addressed also the more difficult problem just described. Work is continuing in several directions, and we mention here briefly four problems under current investigation.

We assumed the given queries $Q_1, \ldots, Q_n$ constitute a single task. The first issue is how to cull from query logs (whether logs of a single database or logs of multiple databases) a set of queries that constitute one task. Another issue is the relaxation of the assumptions on sufficiency and necessity; that is, how to find an interpreting global query when the set $Q_1, \ldots, Q_n$ is neither sound (some queries should be discarded) nor complete (some information has been obtained externally). Choosing consolidating queries often poses an interesting dilemma, as to which consolidation should be preferred: a *complex* query that integrates all the gathered information *precisely*, or a *simpler* query that only *approximates* the total of information [21]. Quite often the latter is more revealing, especially in situations when the query set is imperfect to begin with. Finally, security-oriented applications of this problem often require that the discovery of roguish intentions would be done in real-time. This means that sequences of queries are analyzed as they are formed, and their interpretations are updated continuously as the sequences progress. Obviously, real-time interpretations pose challenging performance issues.

## References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*, chapter 4, pages 55–56. Addison-Wesley, 1995.
2. A. C. Acar and A. Motro. Why is this user asking so many questions? Explaining sequences of queries. In *Proceedings of DBSEC 04, 18th IFIP Annual Conference on Data and Applications Security, Sitges, Catalonia, Spain*, pages 159–176, 2004.
3. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of VLDB 94, 20th International Conference on Very Large Data Bases, Santiago, Chile*, volume 487–499, 1994.
4. Y. Arens, C. A. Knoblock, and W.-M. Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 6(2/3):99–130, 1996.
5. J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *Proceedings of CAiSE 02, 14th Conference on Advanced Information System Engineering, Toronto, Canada*, pages 452–466, 2002.
6. D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of ICDE 01, 17th IEEE International Conference on Data Engineering, Heidelberg, Germany*, pages 443–452, 2001.
7. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.
8. A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.
9. V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Proceedings of VLDB 02, 28th International Conference on Very Large Data Bases, San Fransisco, CA*, pages 670–681, 2002.
10. S. Kapoor and H. Ramesh. Algorithms for enumerating all spanning trees of an undirecting graph. *SIAM Journal on Computing*, 24(2):247–265, 1995.

11. T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *Proceedings of the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments, Palo Alto, CA*, pages 85–91, 1995.
12. A. Y. Levy, A. Rajaraman, and J. D. Ullman. Answering queries using limited external query processors (extended abstract). In *Proceedings of PODS 96, 15th ACM Symposium on Principles of Database Systems, Montreal, Canada*, pages 227–237, 1996.
13. D. Maier, J. D. Ullman, and M. Y. Vardi. On the foundations of the universal relation model. *ACM Transactions on Database Systems*, 9(2):283–308, 1984.
14. T. Mason and R. Lawrence. INFER: A relational query language without the complexity of sql. In *Proceedings of CIKM 05, 14th ACM Conference on Information and Knowledge Management, Bremen, Germany*, pages 241–242, 2005.
15. A. Motro. Constructing queries from tokens. In *Proceedings of ACM SIGMOD 86, International Conference on Management of Data, Washington, DC*, pages 120–131, 1986.
16. A. Motro. Superviews: Virtual integration of multiple databases. *IEEE Transactions on Software Engineering*, SE-13(7):785–798, 1987.
17. A. Motro. Multiplex: A formal model for multidatabases and its implementation. In *Proceedings of NGITS 99, Fourth International Workshop on Next Generation Information Technologies and Systems, Zichron Yaacov, Israel*, Lecture Notes in Computer Science No. 1649, pages 138–158. Springer-Verlag, 1999.
18. A. Motro and P. Anokhin. Fusionplex: Resolution of data inconsistencies in the integration of heterogeneous information sources. *Information Fusion*, 7(2):176–196, 2006.
19. F. Naumann, U. Leser, and J. C. Freytag. Quality-driven integration of heterogenous information systems. In *Proceedings VLDB 99, 25th International Conference on Very Large Data Bases, Edinburgh, Scotland*, pages 447–458, 1999.
20. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.
21. C. D. Shum and R. Muntz. Implicit representation for extensional answers. In *Proceedings of EDS 88, Second International Conference on Expert Database Systems, Tysons Corner, MA*, pages 257–273, 1988.
22. V. S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. J. Lu, A. Rajput, T. Rogers, R. Ross, and C. Ward. HERMES: A heterogeneous reasoning and mediator system. http://www.cs.umd.edu/projects/ hermes/publications/abstracts/hermes.html, 1994.
23. J. A. Wald and P. G. Sorenson. Resolving the query inference problem using Steiner trees. *ACM Transactions on Database Systems*, 9(3):348–368, 1984.