# Query Planning with Information Quality Bounds

Ulf Leser[1] and Felix Naumann[2]

[1] University of Technology, Straße des 17. Juni 135, 10623 Berlin, Germany
[2] Humboldt University, Unter den Linden 6, 10099 Berlin, Germany

**Abstract** Query planning for information integration using a local-as-view approach is exponential in the size of the user query. Furthermore, it may generate an exponential number of plans, many of which will produce results of very poor quality. We propose to use information quality reasoning to speed up query planning. We construct tight upper quality bounds for a branch & bound algorithm. The algorithm uses these quality scores to filter out non-promising plans early on. Experiments show that this approach dramatically improves planning time without compromising the quality of the result.

## 1 Introduction

Information integration in the Internet age must deal with two especially difficult problems: (a) the high degree of heterogeneity between sources and (b) the enormous amount of potential information sources that must be considered. Projects such as Information Manifold [5] and Infomaster [2] use the Local-as-View (LaV) approach described by Ullman in [10] to overcome heterogeneity. However, systems using LaV require exponential query rewriting algorithms. Furthermore, they potentially generate an exponential number of plans for a given query.

On the other hand, it can be observed that many plans will produce results of low *information quality* (IQ). For instance, plans may produce only few tuples, data of low accuracy, or tuples with many missing values. In [7], we proposed to use IQ scores to filter out such plans and suggested a three step procedure: In the first step, we identified data sources that are qualitatively worse than all other sources. The remaining data sources are used for query planning in the second step. Once the set of correct query plans is obtained, the third step selects the most promising plans, based on the expected quality of their results. This *quality-based plan selection* greatly reduces the number of queries that must be submitted via the Internet, but does not improve the performance of query planning itself. At the same time, it delivers query responses that are of high quality according to the IQ scores assigned to the sources.

[1,2] This research was supported by the German Research Society, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316)

In this work, we considerably improve this approach by describing a branch & bound algorithm that merges the second and third step, i.e., we seamlessly integrate the consideration of IQ scores into query planning. We present experiments showing a drastic reduction in query planning time. The algorithm we developed is also very efficient for LaV query planning without IQ scores.

*Structure of the paper.* We review query planning using LaV in Sect. 2, and introduce information quality aspects of query plans in Sect. 3. Section 4 describes the HiQA algorithm which integrates quality reasoning and query planning. In Sect. 5 we present experiments and show the increase in performance gained through this integration. Section 6 discusses related work and we conclude in Sect. 7.

## 2 Local-as-View Query Planning

We assume a global schema $S$ consisting of a set of relations $R$. $S$ exists only virtually; the data physically resides in a set of known data sources. A data source is *logically* modeled as a conjunctive view on $S$, i.e., it is described as:

$$v(\boldsymbol{E}) \leftarrow r_1(\boldsymbol{A}_1), \ldots, r_n(\boldsymbol{A}_n), C_1, \ldots, C_k;$$

where $r_i \in R$. The $C_i$ are conditions of the form '$v$ *op* $c$', where $v$ is a variable, $c$ is a constant, and $op \in \{=, \leq, \geq, <, >\}$. The variables in $\boldsymbol{E}$ are called *exported variables*. The view $v$ must be *safe*. Let $V$ be the set of views describing all available data sources.

The purpose of query planning is to find answers to a user query $u$ against the global schema $S$. Such answers are computed by *plans*. A plan $p$ is a combination of views from $V$ together with a tuple $(\alpha, C)$, where $\alpha$ is a unificator introducing joins between the views, and $C$ is a set of conditions on variables exported in the plan.[1] A plan exports all variables exported in any of the views it contains. The *expansion* $p'$ of $p$ is the conjunction of the bodies of all its views with $C$ appended and $\alpha$ applied. $p$ is *correct* for user query $u$ if and only if $p'$ is *contained* in $u$. The answer to $u$ is defined as the union over the results of all correct plans.

Recall the definition of *query containment* by Chandra and Merlin [1]. The expansion $p'$ of a plan $p$ is contained in $u$, written $p' \subseteq u$, if and only if there exists a *containment mapping* from $u$ into $p'$. A containment mapping is a mapping from the symbols of $u$ into the symbols of $p'$ where (a) every exported variable from $u$ is mapped to an exported variable of $p'$, (b) every literal of $u$ is mapped to at least one literal of $p'$, and (c) the conditions of $p'$ imply the conditions of $u$. Levy et al. prove that we can find all correct plans by testing all plans with no more views than $u$ has literals [4]. The authors also show that finding all correct plans is an NP-complete problem.

[1] For simplicity, we ignore the unificator and conditions in the rest of the paper.

*Example.* Consider a system that provides integrated access to stock information. The global schema and a set of data sources are given in Fig. 1. A user query $\mathrm{u}(sc, cn, sp) \leftarrow \mathrm{price}(sc, cn, cc, sp), cc = \text{`}D'$ has two correct plans, each containing only one view, namely $p_1 = \{v_5\}$ and $p_2 = \{v_6\}$. $p = \{v_4\}$ is not a correct plan because the conditions on the company's country do not match.

| |
|---|
| price*(shareCode, companyName, companyCountry, date, sharePrice)* |
| press*(companyName, pressRelease, domain, date)* |
| rating*(shareCode, compName, rating, domain, date)* |

| |
|---|
| 1. Business press releases for the IT market<br>$v_1(cn, pr, d) \leftarrow \mathrm{press}(cn, pr, dom, d), dom = \text{`}IT\text{'}$ |
| 2. Press announcements<br>$v_2(cn, pr, dom, d) \leftarrow \mathrm{press}(cn, pr, dom, d)$ |
| 3. Business ratings of the day<br>$v_3(cS, cn, r, d) \leftarrow \mathrm{rating}(sc, cn, r, dom, d), dom{=}\text{`}cars\text{'}$ |
| 4. Business ratings for US IT companies<br>$v_4(sc, cn, r, d) \leftarrow \mathrm{rat.}(sc, cn, r, dom, d), \mathrm{price}(sc, cn, cc, d, \text{-}), cc = \text{`}US\text{'}, dom{=}\text{`}IT\text{'}$ |
| 5. New York stock exchange<br>$v_5(sc, cn, cc, d, sp) \leftarrow \mathrm{price}(sc, cn, cc, d, sp)$ |
| 6. German stock exchange<br>$v_6(sc, cn, d, sp) \leftarrow \mathrm{price}(sc, cn, cc, d, sp), cc{=}\text{`}D\text{'}$ |

**Figure1.** Global schema and data sources

## 3 Quality of Query Plans

Query planning based only on views cannot distinguish different correct plans. For instance, it cannot tell apart several Web services that all offer the same type of stock information, although these sources certainly differ in their reliability, completeness, timeliness etc. In such cases, human users would make their choice based on *quality estimations* gained through past experiences. We aim at introducing the same ability into query planning. Our final goal is not to compute all correct plans, but only the best $N$ plans. To this end, we suggest to use information quality (IQ) as basic "cost" factor.

We capture information quality as a set of *IQ criteria* $c_1, \ldots, c_q$. The criteria may range from subjective ones like *understandability* and *reputation* to objective ones like *availability* and *completeness.* Our method is not restricted to a certain set of criteria. To describe the quality of the information stored in a data source, we attach to each view $v$ an IQ vector with one dimension for each criterion, denoted as $IQ(v)$. Depending on the nature of a criterion, such scores may be obtained by inspecting typical answers, measuring response times, estimating coverage, etc.

Let $p$ be a plan consisting of the views $\{v_1, v_2, \ldots, v_k\}$. The IQ vector of $p$, written as $IQ(p)$, is obtained by combining $IQ(v_1)$, $IQ(v_2)$, …, $IQ(v_k)$ following the *join structure* of $p$. Imagine the join tree of $p$. The leaves represent views, which provide the data, and the inner nodes represent joins between data sources. We compute the IQ vector of each inner node bottom-up by combining the scores of the IQ vectors of its children. Each of the dimensions of the IQ vector is handled separately by a criterion-specific *merge function.* $IQ(p)$ is the IQ vector of the root of the tree. For a list of criteria and their merge functions see [7].

To compare the quality of different plans, we first scale the values inside each vector to make them comparable. Then we compute the overall quality of a plan $p$, written $iq(p)$, as their weighted sum, according to a user-defined weighting. $iq(p)$ is the scalar IQ score of $p$. In [7], we proposed to first compute all correct plans, order them according to their IQ scores, and execute only the top $N$ plans. Now, we avoid the inefficiency of computing all correct plans using IQ based *subplan selection.*

We call each inner node of the join tree of a plan $p$ a *subplan* ($sp$). Obviously, we can compute $IQ(sp)$ and $iq(sp)$ for each subplan $sp$ in the same manner as for plans. In the following section, we use these measurements to order subplans according to the *expected* maximal quality a plan including this subplan can reach.

## 4 High Quality Planning

To test whether a plan $p$ is correct for a user query $u$, we need to find a containment mapping $h$ from $u$ into $p'$. $h$ must map each literal of $u$ into a literal of $p'$, i.e., $h$ must map all variables appearing in a literal $l$ of $u$ into variables of a literal $l'$ of $p'$. We use this property to devise an enumeration strategy for subplans such that we can use their IQ score to prune the search space.

Suppose a user query $u$ has literals $l_1, \ldots, l_n$. First, we compute a bucket $B_i$ for each literal $l_i$. Let $v$ be a view from $V$. We put the tuple $(v, h)$ into the bucket $B_i$ if $v$ contains a literal $l'$ such that $h$ is a containment mapping from $l_i$ into $l'$, i.e., if $v$ is 'useful' for answering $u$ wrt. $l_i$.

In a second step, we consider each combination of one element from each bucket: $((v_1, h_1), (v_2, h_2), \ldots, (v_n, h_n)) \in B_1 \times B_2 \times \cdots \times B_n$. Each such tuple represents a *potential plan* consisting of the views $v_1 \ldots v_n$. Unfortunately, not all potential plans are correct. The containment mappings from the literals of $u$ into the views of the plans, i.e., $h_1$, $h_2$, …, $h_n$, may be *incompatible.* For instance, they may map one variable of $u$ into different constants. Also, conditions may conflict, and it may be necessary to merge views connected through non-exported variables. For this reason, a simple greedy approach will not work.

Now, we describe the High Quality Branch & Bound algorithm (HiQA). It intelligently enumerates correct plans in such a way that it finds the best $N$ plans, usually after computing only a fraction of the total number of plans.

The HiQA algorithm enumerates all potential plans by traversing a tree constructed as follows. The root of the tree is empty. We connect all elements of $B_1$ as children to the root. Each of these children gets all elements of $B_2$ as children, and so forth (see the following example and Fig. 2). This tree represents the search space; each path from the root to a leaf is a potential plan.

HiQA traverses this tree and incrementally constructs plans by building subplans of increasing length. In each *branching* step it adds the elements of a new bucket to an existing set of subplans. A subplan is discarded whenever the views it contains are incompatible. This practice already greatly reduces the average time necessary to find correct plans even without considering IQ scores (see Section 5).

HiQA improves query planning further. It uses IQ scores to prune those subplans that will provably not be part of any top $N$ plan. This step is the *bounding* part. Therefore, HiQA efficiently calculates for each subplan the maximum IQ score that any plan containing this subplan can ever reach (see Section 4.1). This score is denoted as $ub(sp)$. HiQA uses it as *upper bound* for other subplans. Suppose that $N$ complete plans have been obtained, and let $iq(p_N)$ be the IQ score of the worst of those. The algorithm may safely prune any subplan $sp$ with $ub(sp) \leq iq(p_N)$. Whenever a new complete plan enters the top $N$ plans, $iq(p_N)$ rises and further subplans can possibly be pruned. Depending on $N$, this additional pruning increases the performance of query planning by magnitudes.

*Example.* We use fictitious IQ scores and omit the associated containment mappings for simplicity. Our goal is to find the 3 best plans ($N = 3$). Consider the following user query against the global schema of Fig. 1.

$$\begin{aligned} u(sc, sp, r, pr) \leftarrow\ & \mathrm{press}(cn, pr, dom, d), \mathrm{rating}(sc, cn, r, dom, d), \\ & \mathrm{price}(sc, cn, -, d, sp),\ d{=}\text{'today'} \end{aligned}$$

First, we compute buckets for the literals of $u$: $B_1 = \{v_1, v_2\}$, $B_2 = \{v_3, v_4\}$, and $B_3 = \{v_4, v_5, v_6\}$. From those buckets we construct the search tree of Fig. 2. We attached to each subplan $sp$ (i.e., inner node) the pair $[iq(sp), ub(sp)]$. For leaves, i.e., complete plans, only one value is given since there $iq(sp) = ub(sp)$. Subplans or plans without such a pair in the figure are never generated because of early pruning.

HiQA starts with one subplan consisting only of $v_1$ and one consisting only of $v_2$. It greedily chooses the next subplan to be expanded. Since $iq(v_1) > iq(v_2)$, it expands $v_1$ next. Combining $v_1$ and $v_3$ fails because the associated containment mappings are incompatible: The variable price.*dom* from $u$ is once mapped to *'IT'* (in $v_1$) and once to *'cars'* (in $v_3$). Next the algorithm generates the subplan $\{v_1, v_4\}$. Since $iq(v_1, v_4) > iq(v_2)$ we expand this subplan next with the three elements from $B_3$. This leads to three complete plans, of which the worst has an IQ score of 0.3. From now on, the algorithm prunes all subplans $sp$ with $ub(sp) \leq 0.3$.

However, it is not yet clear if the three completed plans are the best three plans. HiQA must examine the remaining subplans. Therefore, it expands the subplan $v_2$ with the views from $B_2$. The subplan $\{v_2, v_3\}$ can be discarded because its maximal IQ score is too low. Only three more plans are considered, which leads to the final set of plans $\{v_1, v_4, v_6\}$, $\{v_2, v_4, v_6\}$, and $\{v_1, v_4, v_5\}$.

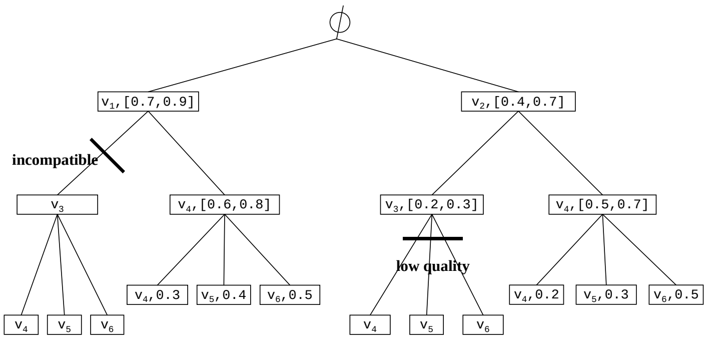

**Figure2.** Branch & bound example for $N = 3$

In total, the algorithm calculated six complete plans. This is considerably better than a brute-force method, which entails the construction and NP-complete testing of $|B_1| * |B_2| * |B_3| = 12$ plans.

## 4.1 Efficient Pruning – the Upper Bound

The efficiency of the branch & bound algorithm HiQA relies heavily on its pruning ability. The more and the earlier branches can be pruned, the faster optimal plans will be reached, and the less time will be spent constructing plans that turn out to be of low quality.

Imagine HiQA has created a new subplan $sp$. We need to compute $ub(sp)$, i.e., an upper bound for the best possible IQ score of any plan containing $sp$. Hence, $ub(sp)$ must obey $ub(sp) \geq \max[iq(p)|sp \subseteq p]$. The tighter this bound is, the more efficient the algorithm will be. Clearly, we could calculate the *exact* bound by simply computing $iq(p)$ for each $p$ with $sp \subseteq p$. This would require a complete enumeration of the search space *in each step*, and would hence destroy the gain of the HiQA approach.

Instead, we estimate $ub(sp)$ efficiently by forecasting plan quality. Let $sp^{(i)}$ be a subplan of length $i$. We compute $ub(sp^{(i)})$ as the IQ score of $sp^{(i)}$ itself combined with an *artificial* IQ vector $\boldsymbol{I}$ for each remaining bucket:

$$ub(sp^{(i)}) := iq\left(IQ(sp^{(i)}) \circ \boldsymbol{I}_{i+1} \circ \ldots \circ \boldsymbol{I}_n\right)$$

A first attempt for $\boldsymbol{I}_j$ $(i < j \leq n)$ would be to simply choose the IQ vector of the view from $B_j$ with the best IQ score. But this will not necessarily result in the best possible plan, because the measure for plan quality is not necessarily monotone; adding a view to a subplan can decrease the merged scores for some criteria, while increasing the merged score for others. Simply choosing the best view from $B_j$ is therefore not safe.

To compensate, we construct $\boldsymbol{I}_j$ by choosing for *each criterion* the best score of any view in the bucket. Since $\boldsymbol{I}_j$ has a higher IQ score than any other view in $B_j$, it is guaranteed that the IQ vector of a plan expanded by *any* $v \in B_j$ is bound by $\boldsymbol{I}_j$.

**Theorem 1.** *$ub(sp^{(i)})$ $(1 \leq i \leq n)$ is an upper quality bound for plans containing sp, i.e., $ub(sp^{(i)}) \geq \max\left[iq(p) | sp^{(i)} \subseteq p\right]$.*

For space constraints we omit a formal proof in which we show that $ub(sp^{(i)})$ decreases monotonically with the length of the subplan. To determine the upper bound for each subplan we precalculate the artificial IQ vector for each bucket. This calculation is in $O\left(n \cdot q \cdot |V|\right)$ where $n$ is the size of the query (= number of buckets), $q$ is the number of IQ criteria, and $|V|$ is the number of views. Then, during the bounding phase we must only include the predetermined artificial IQ vector to the IQ score of the subplan. This is possible in linear time.

## 4.2 Efficient Branching

In the previous section we have shown that we can safely prune subplans using upper quality bounds. Further improvements can be achieved by choosing an intelligent branching strategy. We found the following two heuristics to be particularly effective:

1. We sort buckets by increasing size to produce long subplans, and hopefully complete plans, more quickly. It is desirable to obtain some complete plans as early as possible because we can only start pruning after we have $N$ complete plans.
2. At each branch the algorithm continues with the best subplan found so far. This heuristics is most important in our branch & bound approach. Without it, the algorithm would be reduced to a random, though non-redundant, exploration of the search space. With it, we proceed purposefully towards best plans.

## 5 Experiments and Evaluation

To evaluate the performance of the HiQA algorithm, we carried out several simulation experiments and observed a dramatic performance increase for different parameter settings such as the size of the query, the number of sources, etc. The main cost factor of HiQA is the number of branches, which corresponds to the number of compatibility tests that are performed for each new child node. Therefore, the focus of our experiments lies in counting the number of compatibility tests performed.

We compare HiQA with two other algorithms. The first uses a brute-force enumeration of the search space, i.e., it enumerates the cartesian product of all buckets. No branching or pruning technique is used. The results are labeled as 'no branching' in Figs. 3 and 4. The second algorithm uses branching but no IQ pruning. This approach also enumerates the cartesian product of all buckets, but already requires considerably less compatibility tests. The results are labeled 'branching' in the figures. Our algorithm, labeled HiQA, includes both pruning and branching techniques.

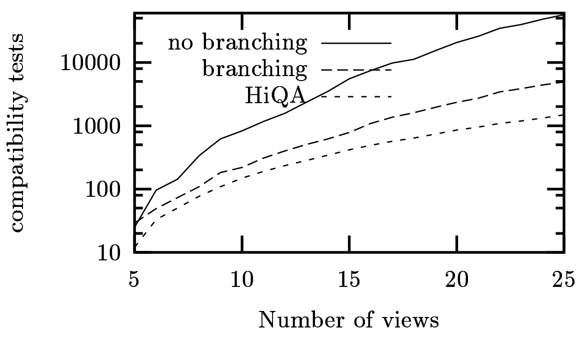

**Figure3.** Scalability in the number of sources

Figure 3 shows the behavior of the algorithms on a logarithmic scale for an increasing number of sources. Since access to many sources is one of the main opportunities of the Internet, these tests are highly relevant for the type of systems we consider.

Figure 4 plots the number of compatibility tests on a logarithmic scale for different query sizes, i.e, for an increasing number of literals in the user query. This experiment shows the limitations of only branching and the impressive efficiency of IQ pruning. While branching still has some advantage compared to a naive algorithm, only IQ pruning overcomes the exponential increase of compatibility tests and thus of the overall performance. Both results show the superiority of the HiQA.

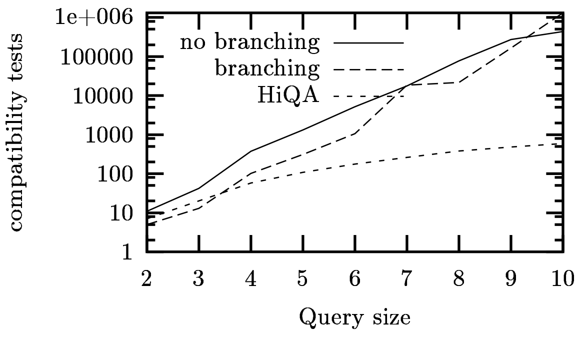

**Figure4.** Scalability in the query size

## 6 Related Work.

The Local-as-View approach to information integration is – for instance – used in the Information Manifold [5] and the Infomaster [2] projects. The Information Manifold introduced the *bucket algorithm* (BA) for query planning. The BA constructs buckets in a similar way as the HiQA. It then enumerates all elements of the cartesian product of all buckets and tests for each whether it is contained in the user query. The HiQA can be considered as an efficient implementation of the BA, enhanced with quality reasoning.

The Infomaster project uses the *inverted facts algorithm* (IFA). Given a user query $u$ and a set of LaV views, the IFA generates a DATALOG program which computes the answers to $u$. Generating the program is polynomial; executing the program essentially amounts to traversing the entire search space of the HiQA. The IFA has the advantage that it can deal with recursive views and functional dependencies, but it cannot cope with built-in comparisons. A similar algorithm is described by Qian [8].

To the best of our knowledge, [7] was the first paper to consider IQ criteria for query planning, although IQ reasoning is known to be very important for large-scale distributed information systems as pointed out by Redman [9]. In [7], we wrapped the BA with IQ considerations. We pruned poor sources *before* planning and identified and executed best plans *after* planning. In contrast, HiQA integrates these phases into a much more efficient one-step algorithm.

Some projects have chosen only one or few IQ criteria for examination. Motro and Rakov, for instance, have studied the completeness and soundness criteria and suggest to use these to improve query results [6]. However, the authors do not go beyond a general definition of the terms and only state possible application areas, whereas we actually use the criteria for planning. The Data Warehouse Quality (DWQ) project also considers IQ criteria, but also, no algorithm is suggested to actually implement their usage [3].

## 7 Conclusions

In this paper, we have pointed out the importance of information quality reasoning when integrating autonomous WWW information sources. Building on a logical query planning method using view-based query rewriting mechanisms, we showed how to seamlessly integrate IQ aspects.

The main contribution of this paper is an efficient method for the problem of finding the best $N$ plans to answer a query in a heterogeneous and distributed environment. Our HiQA algorithm has as distinguishing features (a) a way of enumerating query rewritings that is more efficient than previously published algorithms, (b) an intelligent branching strategy, and (c) IQ scores to prune away non-promising plans. Especially its pruning ability greatly reduces the search space. We have shown the usefulness of IQ pruning in several experiments. Essentially, many user queries simply cannot be answered efficiently without IQ pruning.

## References

1. A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 77–90, 1977.
2. O.M. Duschka and M.R. Genesereth. Answering recursive queries using views. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 109–116, Tucson, Arizona, 1997.
3. M.A. Jeusfeld, C. Quix, and M. Jarke. Design and analysis of quality information for data warehouses. In *Proceedings of the International Conference on Conceptual Modeling (ER)*, pages 349–362, Singapore, November 1998.
4. A.Y. Levy, A.O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 95–104, San Jose, CA, 1995.
5. A.Y. Levy, A. Rajaraman, and J.J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 251–262, Bombay, India, 1996.
6. A. Motro and I. Rakov. Estimating the quality of databases. In *Proceedings of the 3rd International Conference on Flexible Query Answering Systems (FQAS)*, Roskilde, Denmark, May 1998. Springer Verlag.
7. F. Naumann, U. Leser, and J.C. Freytag. Quality-driven integration of heterogenous information systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Edinburgh, 1999.
8. X. Qian. Query folding. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 48–55, New Orleans, LA, 1996.
9. T.C. Redman. The impact of poor data quality in the typical enterprise. *Communications of the ACM*, 41(2):79–82, 1998.
10. J.D. Ullman. Information integration using logical views. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 19–40, Delphi, Greece, 1997.