



# Query Processing and Optimization on the Web

MOURAD OUZZANI  
ATHMAN BOUGUETTAYA  
*Department of Computer Science, Virginia Tech*

mourad@vt.edu  
athman@vt.edu

**Recommended by:** Ahmed Elmagarmid

**Abstract.** The advent of the Internet and the Web and their subsequent ubiquity have brought forth opportunities to connect information sources across all types of boundaries (local, regional, organizational, etc.). Examples of such information sources include databases, XML documents, and other unstructured sources. Uniformly querying those information sources has been extensively investigated. A major challenge relates to query optimization. Indeed, querying multiple information sources scattered on the Web raises several barriers for achieving efficiency. This is due to the characteristics of Web information sources that include volatility, heterogeneity, and autonomy. Those characteristics impede a straightforward application of classical query optimization techniques. They add new dimensions to the optimization problem such as the choice of objective function, selection of relevant information sources, limited query capabilities, and unpredictable events. In this paper, we survey the current research on fundamental problems to efficiently process queries over Web data integration systems. We also outline a classification for optimization techniques and a framework for evaluating them.

**Keywords:** query optimization, Web, data integration, mediators, databases

## 1. Introduction

The turn of the last century has witnessed the largest expansion of the Internet ever [45]. The widespread deployment and use of the Web [10] was the key factor behind this exponential growth. It is now the primary medium shaping all present and future human activities. This has been particularly enabled by the technical advances achieved in different areas including high speed networking, communication protocols, mark-up languages, graphical interfaces, Java technologies, and communication middleware. The Web is continuously attracting waves of new users and service providers. It is now the *de facto* medium for exchanging information and services. This *globalization* has also spurred the development of tools and aids to navigate and share information in corporate intranets that were previously accessible online only in a restrictive way and at prohibitive costs. The information age revolution has highlighted the role of the *database management system (DBMS)* as a key enabling technology. DBMSs are currently the technology of choice for modeling, storing, managing, and efficiently querying large amounts of information. They also provide functions for protecting data integrity, facilitating reliable and flexible data access and manipulation, synchronizing concurrent accesses from applications, and securing data.

The early Web (the period from 1992 to 1996) provided users access to text-based pages through hypertext links. Nowadays, the Web provides access to a variety of data that can

be multimedia-rich. Readily available information retrieval techniques such as inverted indices, which allow efficient keyword-based access to text, largely enabled access to the exponentially growing Web. As pressure from users mounted to allow access to richer types of information and provide services beyond simple keyword-based search, the database research community responded with a two-pronged solution. First, by using databases to model Web pages, information could be extracted to dynamically build a schema against which users could submit SQL-like queries. By adopting XML for data representation, the second proposed solution centered on adding database constructs to HTML to provide richer, queryable data types.

Today's DBMS technology faces yet another challenge as researchers attempt to make sense of the immense amount of heterogeneous, fast-evolving data available on the Web. The large number of cooperating databases greatly complicates *autonomy* and *heterogeneity* issues. This requires better models and tools for describing data semantics and specifying metadata. Techniques for automatic data and metadata extraction and classification (ontologies, for example) are crucial for building tomorrow's Semantic Web [11]. Query languages and query processing and optimization techniques need to be extended to exploit semantic information. Users also need adaptive systems to help them explore the Web and discover interesting data sources and interfaces that support different query and search paradigms. Data dissemination techniques and notification services must be developed to enable effective data delivery services. Web-centric applications such as e-commerce and digital government applications pose stringent organizational, security, and performance requirements that far exceed what is now possible with traditional database techniques.

As part of a wide and successful deployment of emerging Web applications (e.g., electronic commerce, digital government), there is a need to provide support to access information on the Web *uniformly* and *efficiently*. During the last decade, a large body of database research has been devoted to issues related to building data integration infrastructures. This includes research in *distributed database systems* [44] and *multidatabase systems* [14]. In most cases, the traditional focus has been on enabling data sharing amongst a *small* number of databases. Seminal papers on large scale networks of databases provided an early framework [15, 16, 56]. With the deployment of the Web, the database technology has adapted and provided early solution to the glut of information that was suddenly accessible [12].

Databases and applications are now evolving in an environment characterized by a larger number of resources, stricter autonomy, wider spectrum of heterogeneity, high dynamics, easier interconnection, higher distribution, (quasi) lack of organizational and control structures, larger number of clients, etc. In light of these characteristics, goals and means to achieve *coordinated* access to databases on the Web need to be reevaluated. Query optimization takes a central stage in data integration systems on the Web. Web's characteristics make the task of enabling *efficient* querying even more difficult.

A major difficulty in optimizing queries on the Web is that once a query is submitted to a specific information source, control over its execution is no longer possible. Further compounding this problem, that information source may exhibit a different behavior from what has been initially assumed, thus impairing predictions. As a result, traditional optimization techniques that rely heavily on statistical information may be hardly applicable. Query optimization on the Web may also span a larger spectrum of criteria than those in

the classical cost model. Such an example is the *information quality* criterion that codifies reliability and availability of sources, fees, etc. Furthermore, the Web's volatility and highly dynamic nature are a challenge when the expectation is that queries always return results. Further, not all information sources are expected to provide the same *query capabilities*. The query processor needs to make sure that the generated query execution plan is *feasible* with respect to these limitations.

In this paper, we survey queries processing and optimization in *Web data integration systems*. We also present a classification of the different presented techniques and a comprehensive framework to evaluate them. Most of the described systems adopt the mediator approach [54] for data integration. Those are systems that match information requests from consumers, individual users or applications, to information providers. This survey classifies the different systems according to the focus of their query optimization approaches. We overview cost-based optimization techniques, adaptive query optimization, quality-based optimization, and query optimization in presence of sources with limited capabilities. In general, cost-based optimization techniques extend the classical cost model through various means such as estimation, caching, etc. Adaptive schemes addresses efficiency in presence of unpredictable events. Quality-based optimization takes a different direction regarding what matters more in terms of optimization on the Web. Finally, the last group of techniques focuses on optimizing queries while making sure that their execution is possible.

The remainder of this paper is organized as follows. Section 2 summarizes early work on query optimization in the pre-Web era. It also briefly outlines some basic query optimization principles. Section 3 introduces data integration on the Web and a thorough presentation of research issues for query optimization in this context. Section 4 represents the core of the survey. It describes different systems and techniques for query optimization on Web data integration systems. These are covered in four parts according to their focus. Section 5 provides a summary and discussion of the different approaches. Section 6 discusses some trends and open issues for data and application integration on the Web through two major paradigm shifts *Semantic Web* and *Web Services*.

## 2. Pre-Web data integration

We briefly outline, in this section, major work conducted in database integration and query optimization in the pre-Web era. We first summarize some basic concepts on query optimization. We then overview the main data integration approaches and how query optimization is addressed.

Given a declarative query, e.g., in SQL, there are several execution plans that can be used to produce its results. These different plans answer the same query and are equivalent in terms of their final output. However, they may differ according to different performance parameters like response time and resource use. Researchers have recognized the importance of designing a module that would select the "best" query execution plan. Optimization has received a particular attention in the database arena and several techniques have been proposed.

The role of the optimizer is to determine a *query execution plan* that minimizes an *objective cost function*. The optimization problem can be described abstractly as follows [19]:

Given a query  $Q$ , an execution space  $E$  that computes  $Q$ , and a cost function  $c$  defined over  $E$ , find an execution  $e$  in  $E_Q$  (the subset of  $E$  that computes  $Q$ ) of minimum cost:  $[\min_{e \in E_Q} c(e)]$ . An optimizer can be characterized by three dimensions [44]: (i) *Execution space* that captures the underlying execution model and defines the alternative executions. (ii) *Cost model* to predict the cost of an execution, and (iii) *Search strategy* that enumerates the execution plans and select the best one. Traditional query optimization strategies have been classified in three main categories [44]:

- *Heuristic-based*. Heuristic rules are used to *re-arrange* the different operations in a query execution plan. For example, to minimize the size of intermediate results.
- *Cost-based*. The costs of different strategies are estimated and the best one is selected in order to minimize the objective cost function. For example, the number of I/Os.
- *Hybrid*. Heuristic rules and cost estimates are combined together.

In most cases, the focus is on the *join* operation which is the most costly operation.

### 2.1. Data integration approaches

Information systems that provide interoperation and varying degrees of integration among multiple databases have been termed multidatabase systems [31], federated databases [29], and more generally *heterogeneous distributed database systems* (HDDBS). *Data integration* generally implies *uniform* and *transparent* access to data managed by multiple databases. A mechanism to achieve that goal is through an integrated schema that involves all or parts of the component schemas. The taxonomy presented by Sheth and Larson [51] classifies the existing solutions in three categories: *global schema integration*, *federated databases*, and *multidatabase language approach*. These categories are presented according to how tightly integrated component systems are.

*Global schema integration* was one of the first attempts at data sharing across HDDBS. It is based on the complete integration of multiple databases to provide a single view (global schema) [52]. In [8], an exhaustive survey on schema integration is provided along with a comparison of several methodologies. In federated databases [29], a certain amount of autonomy for individual database systems is maintained. Information sharing occurs through *import* and *export* schemas. A particular database exports part of or the whole schema depending on which database it is exporting to. The importing database has to perform any needed integration locally. All databases are registered in a federal dictionary.

The *multidatabase language* approach is intended for users who do not use a predefined global or partial schema. Preexisting heterogeneous local databases are usually integrated without modifications. Information stored in different databases may be redundant, heterogeneous, and inconsistent. The aim of a multidatabase language is to provide constructs that perform queries involving several databases at the same time. One major criticism of this approach is the lack of distribution and location transparency, as users have to *a-priori* find the right information in a potentially large network of databases. Users are responsible for understanding schemas, and detecting and resolving semantic conflicts. In this approach, users are faced with the issues that consist of *finding* the relevant information in

multiple databases, *understanding* each individual database schema, *detecting* and *resolving* semantic conflicts, and *performing* view integration.

### 2.2. Query optimization

Query optimization has received a particular attention in heterogeneous distributed databases systems. It was noted early on that the lack of statistical information from participating databases prevented a direct application of techniques developed for homogeneous systems. Different techniques have been proposed to overcome the lack of statistical information. For example, Pegasus [50] uses a cost model based on logical parameters that include database cardinality and selectivity. These parameters are obtained using calibrating queries. If they cannot be calculated, a number of heuristics are used to optimize queries. In CORDS [57], sampling and probing are applied to local databases to estimate costs by gathering statistical information. Some special queries are executed on top of the participating databases and their behavior and results are recorded. This process is repeated as frequently as needed. Query processing at the global level is then conducted using the gathered information along with a classical cost-based model. Another example is MIND [42]. It uses a dynamic optimization strategy. A statistical scheme is used at runtime to determine and select the less costly and more selective inter-site operation between currently available partial results. Each inter-site operation is assigned with a weight that includes its cost, selectivity, and transfer cost. Operations are scheduled if their weights do not exceed a threshold value computed each time a partial result is available.

## 3. Web-based data integration

The advent of the Web has brought to the fore the seamless interconnection of diverse and large numbers of information sources. Allowing *uniform* querying of those sources has been a major goal of several research efforts. Most proposed systems and techniques focused on making such uniform querying feasible despite all types of hurdles (e.g., heterogeneity, autonomy, unpredictability of the Web, etc.) However, achieving the full potential of uniformly querying disparate Web information sources is fundamentally dependent on devising adequate query optimization techniques.

Different approaches have been used for Web-based data integration. The most widely used approach is based on the concept of *mediator* initially introduced in a seminal paper [54] by Gio Wiederhold. Most of the systems and techniques in this survey fall into that category. There are also other approaches based on the use of agents [41], ontology [2, 37, 43], and information retrieval techniques [5, 33, 38]. This section gives some details about the mediator approach. Then, it highlights major research issues related to query optimization for data integration on the Web.

### 3.1. Mediator based approaches

A simplified architecture of mediator systems is shown in figure 1. Mediators provide an *integrated view* or *mediated schema* over multiple heterogeneous and autonomous information

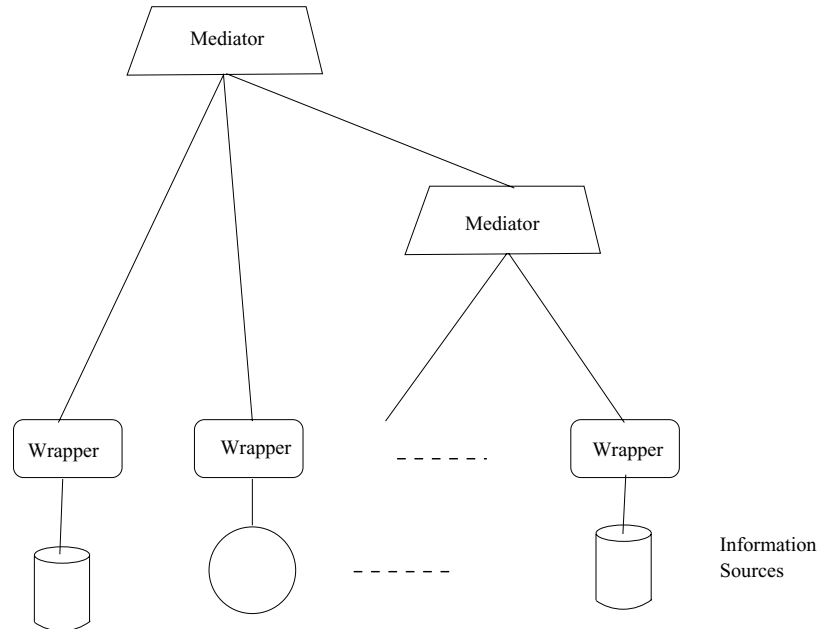


Figure 1. Mediator.

sources. This schema represents generally a *synthesized view* over a specific application domain. Users access the integrated view through a uniform interface that offers *location*, *model*, and *interface transparency*. In general, each source is connected to a *wrapper* that enables its participation in the system. It translates between the source's local language, model, and concepts and those at the mediator level.

To resolve a query, a mediator typically performs three main tasks [27]:

- *Database selection.* Locate and select the databases that are relevant to the query.
- *Query translation.* Decompose the query into sub-queries with respect to the previously selected databases. Each sub-query is transformed into a form that is executable by the corresponding database. The sub-query is then sent to the database (through a wrapper) and results are retrieved.
- *Result merging.* Combine the different results into a global answer to the user.

An important characterization of mediator systems relates to the nature of the relationship between the *mediated schema* and the schemas of participating databases. Two main approaches have been adopted [20]: In the *source-centric* (aka *local-as-view*) approach, relations stored by the information sources are described in terms of the global schema. In the *query-centric* (aka *global-as-view*) approach, the global schema is defined in terms of the relations stored by the local information sources. The source-centric approach scales better than the query-centric since modifications at the information sources do not affect

the rest of the system. On the other hand, translating global queries into local ones is easier in the query-centric approach.

### 3.2. *Research issues*

Query optimization has received a particular attention in different types of database systems (e.g., central, distributed, and multidatabase). Indeed, the ultimate goal of any database system is to allow efficient querying. Unsurprisingly, data integration systems over the Web does not escape to that objective. Query optimization is also central to the deployment of data integration systems over the Web. It has been deemed as more challenging due to the very nature of the Web (large heterogeneity spectrum, strict autonomy, large user base, dynamic behavior, etc.) Queries over Web information sources may be answered in various ways. Each alternative outputs usually the same results. However, alternatives may differ widely in terms of efficiency. This may relate to response time, network resources, number of information sources involved, quality of the information being accessed, quality of returned results, users' satisfaction, and so on. Consequently, query optimization techniques for the Web need to be carefully crafted. Devising the right techniques would necessitate to address a large spectrum of issues. In the following, we outline issues that are directly related to query optimization over data integration systems on the Web.

*Optimization paradigm.* Optimizing queries amounts usually to minimizing the response time. This is the objective function driving most optimizers. Although, this is still desirable on the Web, some applications may require the use of different parameters in the objective function. These include fees to access informations sources, quality of the data (e.g., freshness), number of sources to access, etc. Devising an optimizer requires to first set up an adequate objective function that is relevant to Web applications.

*Optimizing over a large number of heterogeneous and autonomous information sources.* Data integration faces a far more incongruent environment than in the pre-Web era. Heterogeneity can happen at different levels of the data integration system. The time and resources required to bridge that heterogeneity may have an important impact on the optimization process. Autonomy, has a more serious impact since several optimization techniques require specific information from information sources. This information is not always easily available. Furthermore, once a (sub-)query is submitted to a specific information source, the optimizer of the data integration system does not have any control over it. Finally, the Web is witnessing an exponential growth in terms of information sources and potential users. Query optimization should take into account scalability issues to avoid performance degradation. This degradation could lead to very inefficient query execution plans.

*Evolving in a dynamic environment.* A major characteristic of the Web lies in its high dynamism and volatility. Information sources availability and behavior can change without warning. In addition, unpredictable events could happen anytime during query processing and execution. The query optimizer would need adaptive mechanisms to avoid missing optimal query execution in the occurrence of any event. Adaptive techniques could be also used to gather optimization information *on the fly* and use them to modify the execution plan.

*Dealing with limited query capabilities.* Web information sources do not generally exhibit the same capabilities in terms of the queries that they can handle. These limitations are due to several reasons including performance and security. They may range from fully featured database systems allowing full access to their content to information sources with restricted access forms. The query optimizer must generate efficient query execution plans that are effectively executable with respect to any constraint imposed by the accessed sources. This may especially mean that some optimal plans are skipped since they are not feasible.

*Locating sources of interest.* Queries do not generally specify the specific information sources that need to be accessed. In some cases, finding those sources is a straightforward process, e.g., *query unfolding* in global-as-view based mediators. While in others it is more challenging. More precisely, the query optimizer should be able to limit access to only: (1) relevant sources to avoid wasting resources, and (2) sources with superior qualities if alternative sources compete in offering the same information. Addressing this issue is dependent on how much meta-information is available about information sources. This relates to the issue of source description and advertisement.

### 3.3. Dimensions for query optimization on the Web

In surveying research conducted on query optimization in Web data integration systems, we noticed that there is a lack of a comparison of the different approaches. That situation is due to the lack of an analytical model to compare those techniques. In classical systems, well established models and benchmark tools have been developed for such purpose. In the following, we propose dimensions along which query optimization for Web-based data integration systems are compared. Some of these dimensions are subjective and may not be precisely quantified.

- *Scalability.* This dimension measures the *degradation*, if any, of the query optimization technique, when the number of information sources grows. Growth could be in terms of information sources and users. This dimension may be difficult to measure in a real setting. Fine tuning may be necessary.
- *Autonomy preservation.* This relates to how much information is required from the information source for deploying the optimization technique.
- *Optimization performance.* This dimension should provide performance evaluation of the objective function under different scenarios. Scenarios may be varied based on the types of queries being submitted, the number and types of sources being accessed, and other parameters. Both analytical and experimental studies need to be conducted. Note that only very few approaches have done such studies.
- *Adaptiveness.* This reflects the ability of the optimization technique to take into account unexpected changes.
- *Capabilities.* This dimension is for checking whether the optimization technique takes into account sources with different and limited query capabilities. There is a need also to check that the technique does not miss optimal plans.

#### 4. Research prototypes and techniques

We divide this section into four parts. In the first part, the focus is on systems that propose query optimization techniques based on a cost model. The second part describes techniques that use quality information of accessed sources and data to optimize queries. In the third part, adaptive techniques for mediator systems are presented. Finally, we describe several techniques that focus on finding feasible and optimal query execution plans over sources with limited capabilities. We have selected in each part some representative (mostly) mediator-based prototypes and techniques for query optimization. Most of these prototypes provide access to structured and semistructured information sources. They have been selected based on the fact that they have addressed or attempted to address several issues related to query optimization.

##### 4.1. Cost based optimization

Several Web-based data integration systems have based query optimization on extending the classical cost model. A major challenge for such approach relates to optimization information from autonomous information sources. All approaches either assume the availability of such statistics, or that they can be estimated or provided by wrappers.

**4.1.1. Disco.** Disco [53] is a mediator system based on the global-as-view approach (figure 2). The mediator generates multiple access plans involving local operations at the information sources and global operations at the mediator level. The data model is based on the ODMG standard. The object definition language is extended to allow multiple *extents*

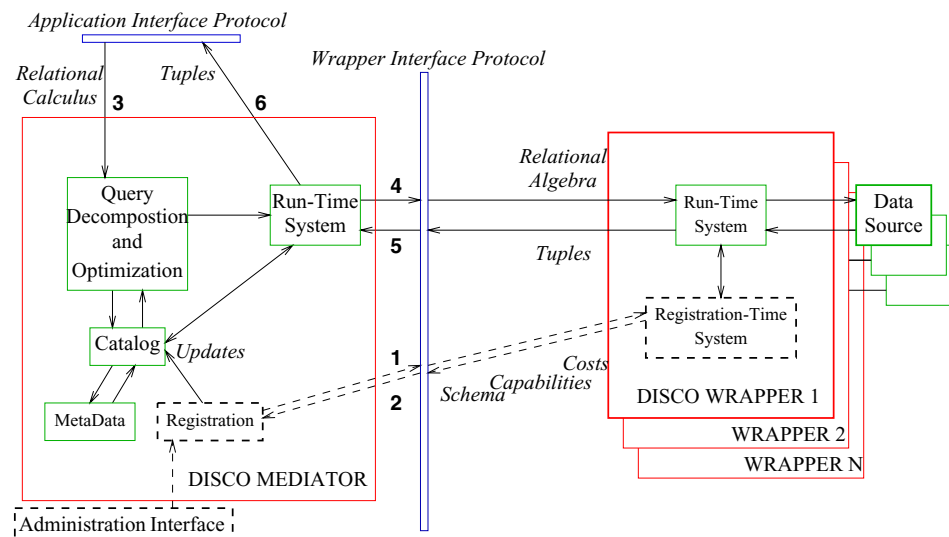


Figure 2. Disco architecture.

to be associated with an interface type of the mediator. It also provides type mapping information between a mediator and a data source.

Disco uses a cost based optimization approach [39]. It combines a *generic cost model* with specific cost information exported by wrappers. The generic cost model uses *cost formulas* established by the *calibrating approach* developed in the IRO-DB federated system [25]. The data source interface is specified using a subset of CORBA IDL extended with a *cardinality section* for data source statistics and a *cost formula section* for specific formulas. The wrapper writer exports statistics (e.g., cardinality of a collection), size rules (reflect the change in result sizes due to an operation), and cost computation rules (compute cost estimates). A query is optimized by estimating the cost of single operations and entire plans. The mediator selects the most specific information available from wrappers and the generic cost model. The cost of the execution of a query plan is determined in a two step bottom-up algorithm: *cost formula integration* and *cost estimation*. In the first step, wrapper rules are integrated into the mediator cost model. In the second step, cost estimates for a plan are generated. The plan is represented as a tree of operator nodes. It is traversed from the root to the leaves and then from the leaves to the root. In the first traversal, cost formulas are associated with nodes. In the second traversal, the cost of each operator is computed.

**4.1.2. Garlic.** Garlic [28, 47] provides an integrated view over heterogeneous information sources using the global-as-view approach (figure 3). Query processing is based on dynamically determining the middleware and wrapper roles in answering a query. A wrapper provides several tasks including: (1) modeling information sources as Garlic objects, (2) allowing Garlic to retrieve references to these object, (3) allowing Garlic to invoke methods on objects and retrieve attributes, (4) participating in query processing and execution. Garlic objects have an interface that abstractly describes the object's behavior and a corresponding implementation that provides a concrete realization. A query is translated into a tree of

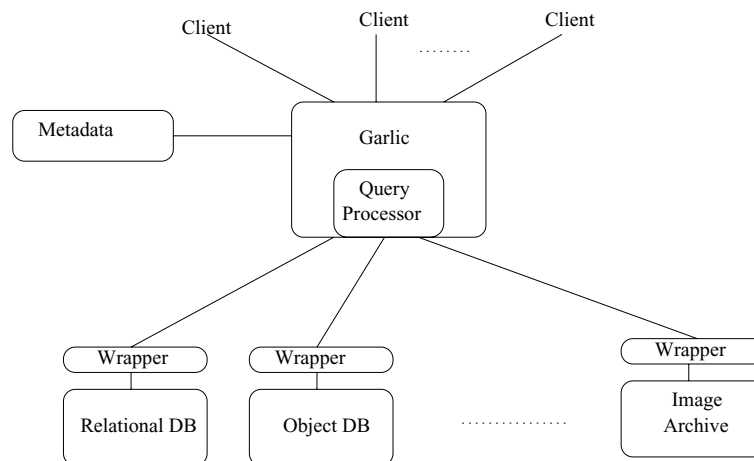


Figure 3. Garlic architecture.

operators or POPs (Plan OPerators). Each operator corresponds to a runtime operator, e.g., join, sort, fetch, and scan. Garlic provides also a generic POP, called PushDown POP, which encapsulates work to be conducted at a information source. Each plan is characterized by properties such as tables used in the plan, output columns, and estimated cost.

Garlic extends the traditional cost-based optimization approach by involving wrappers as important components in the optimization process [46]. Wrappers cooperate in the estimation of the total cost of a given query plan. The proposed framework aims to provide the necessary means to extend the traditional cost-based optimization to a heterogeneous environment. The framework includes a *cost model*, *cost formulas*, and *statistics*. Wrappers may use a default cost model or a more specific one to model their execution strategies. The default cost model considers the total cost of a POP operator as a combination of the cost of two basic tasks: *reset* for initializing the operator and *advance* for retrieving the next result. Default cost formulas are used to compute these two costs based on statistics. Based on the cost of the *reset* and *advance* tasks, the wrapper should be able to provide cost estimates for its own plans that are included in the global query plan. The *total cost*, *re-execution cost*, and *result cardinality* are considered in the cost of a PushDown POP. They are required to estimate the cost of the global query plan. Once the optimizer selects a winning plan, that plan is translated into an executable form. Garlic POPs are translated into operators that can be directly executed by the Garlic execution engine.

**4.1.3. Ariadne.** *Ariadne* [3] is an integration system that uses the local-as-view mediation approach. The *LOOM* knowledge representation system [36] is used to construct the domain model that represents an integrated view of the sources. User queries are formulated against that model. Query processing has two phases: a *preprocessing phase* and a *query planning phase*. A *source selection* algorithm pre-processes the domain model allowing the selection of sources based on classes and attributes. Query planning is a combination of source selection strategy and traditional distributed query optimization. It first generates an initial plan (eventually suboptimal) and then applies *rewriting rules* in order to improve the quality of the plan. These rewriting rules are iteratively applied until either an acceptable solution is found or a resource limit is reached. The quality of a query plan is based on a classical cost model depending on the size of intermediate results, cost of relational operations, and communication costs. The authors assume the availability of some statistics from the underlying sources to compute these costs. Three classes of rewriting rules are then considered. The first class of rules is derived from the properties of the distributed environment and contains four rules. One rule is based on the use of alternative information sources that can answer the same query but with different costs. The three other rules allow the execution of a group of operations in remote sources instead of transferring the data and executing the operations at a local source. The second class of rules is derived from some properties of the relational algebra (e.g., commutativity, associativity, etc.) The last class of rewriting rules relates to the heterogeneity of the environment. They are needed to reconcile the semantic differences between sources. The idea is to use axioms that describe how attributes and classes can be obtained by combining information from a set of sources.

**4.1.4. Hermes.** *Hermes* [1] considers the construction of mediators based on two major distinct tasks: *domain integration*—physical linking of the information sources, and the

*semantic integration*—coherent extraction and combination of the information provided by these sources. Query optimization in Hermes is based on a cost-based model using caching [1]. Statistics of calls to the sources are locally cached in order to estimate the cost of possible execution plans. In addition, *invariants* are used to derive equivalent execution query plans. Invariants represent expressions that show possible substitutions for external domain calls. They specify which call can be substituted. First, the query processor checks the cache to see if the answer for a domain call is already stored in that cache. It then uses the invariants to substitute domain calls and check whether they are in the cache. If the invariants indicate that there is a domain call in the cache that only provides a partial list of answers, then the actual domain call may need to be performed.

Hermes optimizer has four components. The *rule rewriter* finds different possible rewritings of the original query expressed in terms of external calls. The *cache and invariant manager* maintains caches and avoids actual calls to external domain when the answer to that call is physically present in the cache. The manager uses also the invariants to find other acceptable entries in the cache. The *domain cost and statistics module* (DCSM) provides estimates of calls to external information sources. This module keeps execution time and cardinality statistics in a database and provides cost estimates to the *rule cost estimator*. Finally, the *rule cost estimator* takes the rewritten query from the rule rewriter and computes the cost of each plan by obtaining the cost estimates of individual calls to the sources from DCSM. The rule rewriter derives more than one plan for a query. Thus, the DCSM has to estimate the costs of each plan and select the best plan. Cost estimation relies on a cost vector database maintained by the DCSM. This database records cost information about domain calls as they are executed by the mediator.

#### 4.2. *Quality-based optimization techniques*

Some systems have considered different parameters to be included in the cost of a query. Those parameters relate to *information quality* of sources and data. The argument is that such quality parameters are more or as much important, in the context of the Web, than classical parameters like response time.

**4.2.1. *ObjectGlobe.*** ObjectGlobe's [17] data integration is centered around three types of suppliers: *data suppliers*, *function providers* for query processing operators, and *cycle providers* for operators execution. The execution of a query may involve query operators supplied by different function providers that are executed at different cycle providers and that retrieve data from different data providers. Query processing follows a multi-step strategy. First, a *lookup* service locates instances from each of the above types of suppliers that are relevant to the resolution of the query. Some cost information are also gathered during this step. In the second step, an optimal plan is built based on a cost model using the previously gathered information. In the last step, the query execution plan is distributed to relevant cycle providers and executed using an iterator model [26].

Queries for the lookup service are extracted by the parser based on *themes* and *attributes* specified in the different clauses of a query. The lookup service uses the generated queries to locate relevant resources by consulting a meta-data repository. The lookup service is also responsible to gather statistical information for the optimizer and authorization information

from the different providers. Authorization information is recorded in a *compatibility matrix* that will annotate the query execution plan. The optimizer enumerates alternative query execution plans using a *System-R* [49] like dynamic algorithm. Costs of plans are estimated using information from the lookup service. If an information is missing it is set to a default value. The optimal plan is then executed by distributing it to the different cycle providers as specified in the host annotations of the plan. In addition, users can specify *quality constraints* on the execution of their query. Constraints are defined on results (e.g., size of the result), cost (i.e., how much the user is ready to pay), and time (e.g., time to first results). *QoS* management is introduced as part of the query processor. In fact, the quality constraints are treated in all the phases of querying processing. If they cannot be fulfilled, the query plan is dynamically adapted or the query is aborted. Based on that QoS concept, the optimizer's goal is to maximize the percentage of successful queries and abort any query that cannot fulfill its QoS constraints as soon as possible.

**4.2.2. HiQIQ.** HiQIQ (High Quality Information Querying) uses information quality criteria to support query optimization in mediator systems [40]. The focus is on incorporating information quality into query planning. This could be useful in environments such as biological information systems where users are more sensitive to quality criteria than the classical database criterion, i.e., response time. The approach uses a mediator based architecture where the mediator uses *query correspondence assertions (QCAs)* to resolve queries. *QCAs* are *set oriented equations* involving queries over mediator and wrapper schemas. This is a sort of combination of the *source-centric* and *query-centric* approaches used in mediators. The query resolution process tries to find all correct plans (combinations of *QCAs*) and then does a union over their results to obtain the complete answer.

To include *quality information (IQ)* in the query planning, three classes of quality criteria have been used. *Source-specific* criteria determine the overall quality of a source. They include ease of understanding, reputation, reliability, and timeliness. *QCA-specific* criteria determine quality aspects of specific queries that are computable by a source. These include availability, price, response time, accuracy, relevancy and representational consistency. *Attribute-specific* criteria relate to the ability of a source to provide the attributes of a specific query. Query processing is conducted in a three phase strategy. Based on the source-specific *IQ* criteria, the first phase prunes low-quality sources. The second phase finds all plans, i.e., combinations of *QCAs*, that produce semantically correct answers. The query planning strategy of Information Manifold [34] (see Section 4.4.2) is used. This phase does not use any quality related information. Finally, plans obtained from the previous phase are qualitatively ranked. This phase starts by determining *IQ scores* for the different *QCAs* in the different plans. That score is a vector with eight dimensions; each dimension corresponding to a non-source-specific criteria (six *QCA-specific* and two attribute-specific). Only attribute-specific criteria are recalculated for each new query, the others are more stable. In a second step, an *IQ* vector is computed for each plan by merging *IQ* vectors in join nodes appearing in the plan. In the third and last step, *IQ* scores for the different plans are scaled, weighted, and compared using a simple decision making method (i.e., the simple additive weighting). This method scales the scores to make them comparable, apply the user weighting, and sum up the scores for each plan.

### 4.3. Adaptive query optimization

An important issue in Web-based data integration is the use of *adaptive* or *dynamic query optimization* techniques to face the high volatility of the Web. These techniques address mainly the lack of statistical information and occurrence of unpredictable events during query execution. They generally try to change the execution plan at run-time, re-optimize the query, or use specific operators that deal more flexibly with unpredictable events (e.g., data delivery rates). An interesting characterization of adaptive query processors is given in [30]. It states that a query processing is adaptive if:

- it receives information from its environment,
- it uses that information to determine its behavior, and
- this process iterates over time, generating a feedback loop between environment and behavior.

In this section, we present some major projects that use dynamic query optimization.

**4.3.1. Telegraph.** The Telegraph project [30] aims to build a query engine over Web information sources based on an adaptive *data-flow paradigm*. The objective is to adaptively route unpredictable and bursty data-flows through computing resources. The query processor continuously reorders applications of pipelined operators in a query plan at run-time on a tuple-by-tuple basis [7]. It uses the concept of *eddy*, defined as a  $n$ -ary tuple router interposed between  $n$  data sources and a set of query processing operators.

An *eddy* encapsulates the ordering of operators by dynamically routing tuples through them. The idea is that there are times during the processing of a binary operator (e.g., join, union) when it is possible to modify the order of the inputs without modifying any state in the operator. Such times are called *moments of symmetry*. They occur at the so called *synchronization barriers*. For the case of *merge join*, this corresponds to one table-scan waiting until the other table-scan produces values larger than any one seen before. Most of the reported work for *eddies* is on the join operator due to its impact on query performance. The focus in Telegraph is on join algorithms with frequent times of symmetry, adaptive or non-existent barriers, and minimal ordering constraints. Efficiency of the system depends tightly on the routing policy used in the *eddies*. Different routing policies need to be used under different circumstances. They depend on operator selectivity, operator consumption and production rate, join implementation, and initial delays of input relations. Eddies have been implemented in the context of a shared-nothing parallel query processing framework called River [6].

**4.3.2. Tukwila.** Tukwila is another system addressing adaptiveness in data integration environment [32]. Adaptiveness is introduced at two levels: (1) between the optimizer and the execution engine, and (2) within the execution engine. In the first level, adaptiveness is deployed by annotating initial query plans by (event-condition-action) *ECA rules*. These rules check some conditions when certain events occur and subsequently trigger the execution of some actions. Examples of events include operator failure, time-out, and out of memory exceptions. Conditions include the comparison of actual cardinalities known at

run-time and those estimated. Finally, actions include rescheduling of the query operator tree, re-optimization of the plan, and alteration of memory allocation. A plan is organized into a partially ordered set of fragments and a set of corresponding rules. Fragments are pipelined units of operators. When a fragment terminates, results are materialized and the rest of the plan can be re-optimized or rescheduled. In addition to data manipulation, operators perform two actions: statistics gathering for the optimizer, and event handler invocation in case a significant event occurs. The operator tree execution follows the top-down iterator model described in [26].

For the second level of adaptiveness, two operators are used: *dynamic collectors* and the *double pipelined hash join* operator. The collector operator dynamically chooses relevant sources when a union involves data from possibly overlapping or redundant sources. The optimizer specifies the order to access sources and alternative sources in case of unavailability or slow delivery. A collector will then include a set of children (wrapper calls or table-scans) and a policy for contacting them. The policy is expressed as a set of event-condition-action (ECA) rules. The double pipelined hash join is a symmetric and incremental join. It aims at producing tuples quickly and masking slow data sources transfer rates. This requires maintaining hash tables for in memory relations. The original double pipelined join has been implemented after a few adaptations. The first adaptation was to retrofit the data-driven bottom-up execution model of that operator with the Tukwila's query processing top-down iterator-based scheme. The second relates to the problem of memory overflow. Two strategies based on swapping are used.

**4.3.3. Interleaving scheduling and optimization.** Another dynamic scheduling strategy that deals also with memory limitation has been proposed in [13]. It is based on monitoring arrival rates at the information sources and memory availability. In the case of significant changes, the execution plan is revised. This means that planning and execution phases are interleaved. The query execution plan is represented by an operator tree with two particular edges: *blocking* and *pipelinable*. In a *blocking edge*, the consumption of data cannot start before it is entirely produced. In a *pipelinable edge*, data can be consumed one tuple at a time meaning that consumption can start as soon as one tuple is available. It is then possible to characterize the query execution plan by pipelinable chains which represent the maximal set of physical operators linked by pipelinable edges. The query engine will have to concurrently select, schedule, and execute several query fragments (pipelinable chains and partial materializations) while minimizing the response time.

The query engine's main components are the *dynamic query optimizer*, *dynamic query scheduler*, *dynamic query processor*, and *communication manager*. The dynamic query optimizer uses dynamic re-optimization techniques to generate an annotated query execution plan. Those annotations relate to blocking and pipelinable edges, memory requirements, and estimates of results' sizes. The dynamic query scheduler builds a scheduling plan at each scheduling phase triggered by events from the query processor. Scheduling is based on some heuristics, the current execution status, and information about the benefits of materialization of pipelinable chains. The dynamic query processor concurrently processes query fragments while maximizing the processor use based on priorities defined in the scheduled plan. The execution may be interrupted in case there is no data arriving from

sources, a query fragment has ended, or delivery rates have significantly changed. This is reported to the query scheduler and eventually to the query optimizer for scheduling or optimization changes. The communication manager receives data from the different wrappers for the rest of the system. It also estimates delivery rates at the sources and reports significant changes to the query processor.

**4.3.4. Query scrambling.** A *query scrambling* approach is proposed in [4]. The goal is to react to delays by modifying the query execution plan *on-the-fly*. Unexpected delays are “hidden” by performing other useful works. Scrambling has a two-pronged action: *rescheduling* (scheduling other operators for execution) and *operator synthesis* (new operators are created when there is no other operator to execute). These two techniques are repeated as necessary to modify the query execution plan. Scrambling policies differ by the degree of parallelism they introduce or the aggressiveness with which scrambling changes the existing query plan. Three important trade-offs must be considered in *rescheduling*: First, the number of operators to reschedule concurrently. This concerns the benefits of overlapping multiple delays and the cost of materializations used to achieve this overlapping. Second, rescheduling individual operators or entire subtrees. Finally, choice of specific operator(s) to reschedule. For *operator synthesis*, a significant amount of additional work may be added since the operations were not originally chosen by the optimizer. To avoid this problem a simple heuristic of avoiding Cartesian products to prevent the creation of overly expensive joins is used. However, the performance of this heuristic is highly sensitive to the cardinality of the new operators created.

#### 4.4. Optimizing queries over sources with limited capabilities

Building mediators involves dealing with sources presenting different capabilities in terms of the queries they can handle. This scenario is likely to occur on the Web where sources adhering to specific requirements and constraints limit access through some patterns. For example, users of an online bookstore get information on books via forms. These forms allow several types of keyword based queries including search by title, subject, author, keyword, ISBN, etc. However, a user cannot submit database-like queries using complex conditions on prices or requesting all available books. There are several reasons for limiting the query capabilities of an information source. These include:

- *Performance.* Only some types of queries are allowed to maintain the performance of the system. For example, allowing conditions on only attributes that have indexes.
- *Security.* Supported queries cannot access privileged information.
- *Legacy.* The underlying data may actually be stored in systems with limited query capabilities.
- *Business.* Queries that can retrieve useful business information, e.g., using aggregate queries, are not allowed.

In this section, we present some techniques that have been developed to deal with this problem, i.e., how to efficiently answer queries over sources with limited capabilities.

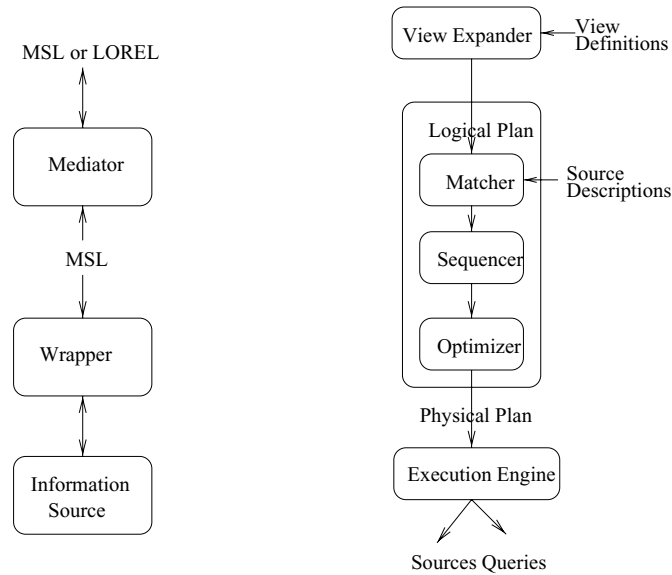


Figure 4. Tsimmis architecture and query processor.

**4.4.1. Tsimmis.** Tsimmis prototype [24] integrates heterogeneous information sources using the global-as-view approach for mediation (figure 4). The proposed mediation is supported through a number of concepts. An object model called *OEM (Object-Exchange Model)* is used to exchange information between components of the system. A *Mediator Specification Language (MSL)* based on OEM serves as the query language, the specification language for mediators, and the query language for wrappers. Capabilities of sources are described using *templates*. OEM is a self-describing object data model, i.e., data can be parsed without reference to an external schema. Templates represent the set of queries that are *answerable* by each source. Generally speaking, a query is *answerable* if it provides all inputs required by the source. The capability descriptions of sources are used to decide on the feasibility of physical query plans.

Query resolution in Tsimmis focuses mainly on finding *feasible query plans* that respect the limited capabilities of available sources. A user query is initially expressed against an integrated view. The *view expander* module (in the mediator) translates the query into a *logical plan* using the *view definitions*. Such logical plan does not specify the order in which source queries are processed. Processing order is specified in the *physical plan* which in turn may have several alternatives. The plan generator processes the logical plan and generates an optimized feasible physical plan in a three step process. First, the *matcher* finds all source queries that can process parts of the logical plan based on the source descriptions. Second, the *sequencer* pieces together the source queries for processing the conditions of the logical plan to build feasible plans. It has to find a feasible sequence of source queries. For example, a query in a feasible sequence has the property that its binding requirements are exported by the source queries that appear earlier in the sequence.

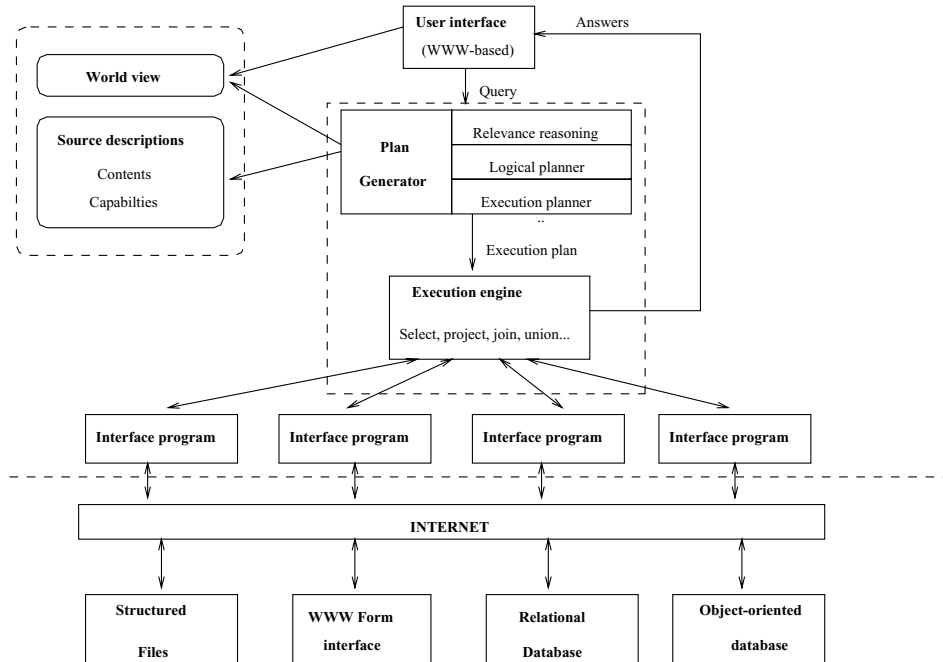


Figure 5. Information manifold architecture.

Finally, the *optimizer* chooses the most efficient feasible plan using a cost based optimization algorithm.

**4.4.2. Information manifold.** Information Manifold [34] provides a uniform interface to structured information sources (figure 5). It proposes a mechanism to describe *declaratively* the content and query capabilities of information sources. The system uses the source descriptions to prune sources for a given query and generate executable query plans. Sources' content is described by queries over a set of relations and classes. Users express their queries against a mediated view consisting of a collection of *virtual relations* and *classes*. Relations are described in the sources as *queries* over the *world view relations*. Thus, Information Manifold follows a *local-as-view* approach for mediation.

The execution plan for a query  $Q$  is generated using a two-phase algorithm. In the *first phase*, a *semantically correct plan* is generated. It is a *conjunctive query*  $Q'$  that uses only source relations and is contained in the user query  $Q$ . First, for each *subgoal* in the query, the algorithm computes a *bucket* containing the information sources from which tuples of that subgoal can be obtained. After that, all possible combinations of information sources, one from each *bucket*, are considered. The obtained plan is checked for semantic correctness. For each subgoal in the query, relevant information sources are computed separately. Conjunctive plans are then built by selecting one relevant source for every subgoal in the query. Each plan is checked for *relevance* and *soundness*. A conjunctive

plan is *sound* if all the answers it produces are guaranteed to be answers to the query. A conjunctive plan is *relevant* if it can produce answers to the query according to the descriptions of the sources and the constraints in the query. In the last step of the first phase, each plan is checked for *minimality*. This ensures that if a subgoal is removed from the plan then the plan is not sound anymore. The *second phase* orders subgoals such that the plan is executable with respect to the capabilities of sources. The algorithm generates the ordering based on possible *bindings*. It maintains a list of available parameters initialized by the set of bound variables in the query. A subgoal is added to the ordering if its input requirements are satisfied with respect to the list of available parameters and source requirements, and it was not already considered.

**4.4.3. Infomaster.** Infomaster [21] provides integrated access to distributed heterogeneous information sources, including databases and semistructured data (figure 6). Infomaster uses the knowledge interchange format (KIF) as an internal content language. KIF is used to represent first order logic expressions. The Infomaster system follows the *source-centric* approach for mediation. Infomaster considers three types of relations. First, *interface relations* are used by users to formulate queries. Second, the *source relations* describe the data stored in the information sources. Both relations are described using a third type, *world relations*, that represents a reference schema for the system.

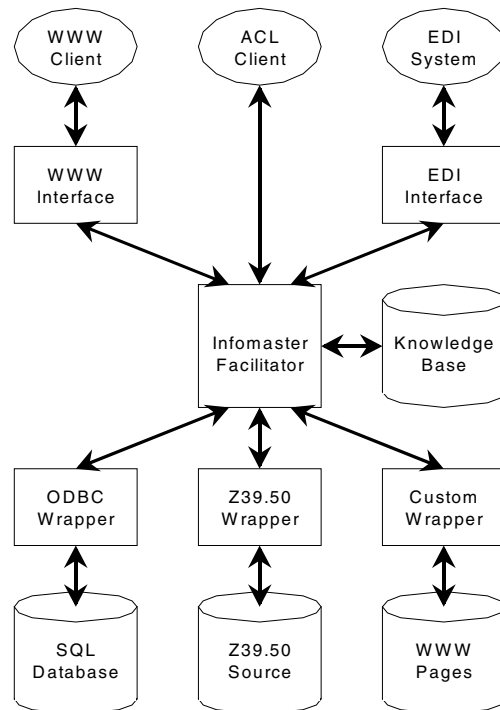


Figure 6. Infomaster architecture.

Query optimization is based on the notion of *local completeness* that limits the number of sources to be queried. Infomaster uses a weak form of completeness that assumes that some subset of the data stored by an information source is complete. This is assumed even if the entire data stored by this information source might not be complete. The claim is that such assumption is reasonable within a specific application domain. In this approach, any source relation is represented by two views over the global relations. The *liberal view* and *conservative view*. These two views have the same schema as the corresponding source relation. The conservative view is a *lower bound* of the source relation and describes the subset of the data that is known to be complete. The liberal view is an *upper bound* of the source relation. Based on conservative and liberal views of the different information sources, the query processor generates a query plan that is: (1) *retrievable*, (2) *semantically correct*, (3) *source complete*, and (4) *view minimal*. The query processor generates such plans in a five step algorithm. (1) *Reduction*: The query is transformed in terms of base relations by rewriting each atom using definition of the associated predicates. (2) *Abduction*: Given the expression  $Q_b$  in terms of base relations and a set of definitions, the set of all consistent and minimal conjunctions and retrievable atoms is produced. This set is contained in  $Q_b$  with respect to the given definition. (3) *Conjunctive minimization*: Any redundant conjunct is eliminated. (4) *Disjunctive minimization*: Any disjunct that can be shown to be contained in the union of the remaining disjuncts is discarded. (5) *Grouping*: The conjuncts within each conjunction are grouped so that the atoms in each group share a common provider.

**4.4.4. Annotated query plans.** The approach proposed in [22] extends the *System-R* optimizer to deal with sources having limited query capabilities. It uses *annotated query plans* in the search space. Annotations describe the *input* that *must* be given to each sub-plan in the query plan to be *answerable*. The proposed algorithm aims to prune invalid and non-viable plans as early as possible and uses a best-first search strategy to produce a complete plan early in the search. Limitations of sources are modeled through binding patterns. A binding pattern specifies which attributes must be *bound*, i.e., have an input value, in order to retrieve tuples from a relation. Each binding pattern is also labeled with the cost of accessing it once and cardinality of the expected output per given input. The cost of a sub-plan is computed based on these costs and the cost of the associated operator. The approach assumes *select-project-join* queries or *conjunctive* queries. The goal is then to find the cheapest complete query execution plan for a conjunctive query  $q$  given a set of binding patterns. A *complete* query execution plan is a plan that covers all the conjuncts in the query and whose *adornment* maps precisely the bound variables in the query to *bound*. Atomic plan costs are deduced from the data access descriptions. Non-atomic plan costs are estimated based on the costs of the operators and the costs of the sub-plans.

The query optimization algorithm starts by an initial set  $S$  of plans containing only atomic plans, i.e., single relations.  $S$  is then iteratively updated by adding new plans obtained by combining plans from  $S$  using selection and join operations. The choice of the partial plans to be combined is based on a *utility* measure. The *utility* measure is based on the number of conjuncts covered by a plan, cost of the plans, number of free variables, or a combination thereof. During the combination of partial plans, pruning occurs by comparing the costs of equivalent plans and the costs of *covering plans*. A plan  $p_1$  covers another plan  $p_2$  if they

are labeled with the same set of conjuncts from the query and  $p_1$  may need less variables to be bound than  $p_2$  ( $p_2$  is equivalent to a *selection* applied to  $p_1$ ). At the end of the iteration, two cases can occur. In the first case,  $S$  contains the *equivalence class* of the query and thus the optimal plan. In the second case, further processing is needed to get the optimal plan. The algorithm applies an exhaustive enumeration of all possible placements of selections in the obtained plans. This process leads to the optimal plan.

**4.4.5. Source sequencing algorithms.** In [55], a simple *greedy strategy* and a *partitioning scheme* are proposed to deal with limited capabilities of sources. Both algorithms use a cost model based on the number of source queries in the execution plan. Users submit conjunctive queries over integrated views provided by a mediator. Each integrated view is defined as a set of conjunctive queries over the source relations. Each source relation defines a set of access templates that specify the binding requirements.

The first algorithm starts by finding all answerable subgoals using the initial bindings in the logical plan, and picks the one with the least cost. The selected subgoal will then provide additional bound variables. Thus, the process is repeated by selecting the *cheapest* subgoals and updating the set of bound variables. The process stops when there is no more subgoals or none of the left subgoals is answerable. It is clear that such greedy approach may miss the optimal plan.

The second algorithm generates optimal plans more frequently than the first but has a worse running time. It has two phases: In the first phase, subgoals are organized into a list of clusters based on source capabilities. All the subgoals in the first cluster are answerable by block queries (i.e., queries that use only initially bound variables) and all other are answerable by parameterized queries (i.e., queries that use variable bindings from the subgoals of earlier clusters.) In the second phase, the algorithm finds the best sub-plans for each cluster and then combines them to obtain the best feasible plan for the query.

**4.4.6. Capability sensitive plans for selection queries.** In [23], two algorithms for generating plans for *selection queries* over Web data sources are presented. Sources exhibit restrictions on the size and structure of condition expressions along with limitations on the set of returned attributes. Sources are described using a context free language. Queries are of the form  $\pi_A(\sigma_C(R))$  and are noted by  $SP(C, A, R)$ . The selection condition  $C$  is represented by a condition tree. *Leaf nodes* represent atomic conditions without any disjunctive or conjunctive conditions. *Non-leaf nodes* represent Boolean connectors  $\wedge$  and  $\vee$ . Mediator query plans are represented by query trees. Leaf nodes represent  $SP$  source queries and non leaf nodes represent selection, projection, union, and intersection operations. These operations are performed at the mediator level to combine results from source queries.

The first algorithm is a simple exhaustive modular scheme that has four modules: *rewrite*, *mark*, *generate*, and *cost*. The *rewrite* module produces a set of equivalent rewritings for the target query condition. It employs a number of rules including *commutative*, *associative*, and *distributive* transformations of condition expressions. The *rewrite* module outputs a set of *condition trees* (CTs) that represent the condition expressions of the query. For each condition tree produced by the *rewrite* module, the *mark* module determines the parts that can be evaluated at the source. If the condition is not supported by the source, the *generate* module produces a set of *feasible* plans by repeatedly invoking the *Exhaustive*

*Plan Generator* (EPG) algorithm on each of the *CTs* passed on by the *mark* module. *EPG* is a recursive algorithm that generates a feasible plan for a query  $SP(n, A, R)$ . If *EPG* finds feasible plans, it represents them using a special operator to indicate alternative query plans for  $SP(n, A, R)$ . Finally, the *cost* module selects the best plan using a cost model.

The second scheme improves the first one by reducing the number of *CTs* that need to be processed and the search space using some knowledge about the cost model. As a result, the *Rewrite* module triggers a fewer number of rules. The commutativity rule is included in the source description itself and is not triggered. This requires rewriting the source descriptions and parsing a larger set of rules. This overhead occurs only once and should not have much impact on performance. The cost of querying a source consists of: (1) the overhead of sending messages and starting a query, (2) the cost of computing the answer at the source, and (3) the cost of transferring results over the network. Based on this cost model, three pruning rules are used to reduce the search space: (1) prune *impure* plans when *pure* plan exist, a *pure* plan process the entire query at the source, (2) prune locally sub-optimal plans, when decomposing an *impure* plan into sub-queries and choose only sub-queries with the least cost, and (3) prune *dominated* plans, i.e., plans with a more expensive cost than other plans on the same relation and attributes, and whose condition includes the condition of the expensive plan (this can only be applied for conjunctive conditions).

#### 4.5. Commercial data integration systems

In this section, we overview major database products that provide some support for data integration. Major database vendors (*Oracle*, *IBM*, *Sybase*, and *Microsoft*) claim different levels of support for data integration on the Web. The purpose of this section is not to compare commercial products but to overview their main features. Because there are a large number of products, this section does not attempt to cover all of them. Instead, we focus on the major players in this arena. Our coverage is based on available information from user manuals and white papers since there are few or no published technical papers detailing those commercial products. Additionally, existing products are at various development stages and operate at different levels of disclosure. A major concern shared by most products is to allow access to external databases or other data sources (e.g., XML, multimedia sources, etc.). Details on how query optimization is achieved and how these DBMSs would be used for Web databases are missing in most cases.

*IBM DB2*. Data integration in *IBM DB2* started with *DataJoiner* that allows to submit SQL queries to external non-DB2 databases. *DB2 Relational Connect* has brought the concept of data integration further by allowing federation of DB2 and non DB2 databases to be accessed uniformly. A DB2 federated system consists of a DB2 instance and one or more data sources. The federated database contains catalog entries identifying data sources and their characteristics. Note that the Garlic research project (described in this survey) has been an important asset in enabling data integration within *DB2*.

*Oracle*. *Oracle Data Access Gateways* enable to access and manage data from heterogeneous data sources. Two types of access are possible: (i) transparent gateways are used for

non-Oracle databases and (ii) procedural gateways provide procedural access to non-Oracle transactional systems. Data integration can be also achieved through data warehousing of external non-Oracle databases using the *Oracle Warehouse builder*.

*Sybase.* *Enterprise Connect Data Access* is Sybase's tool to access non-Sybase databases including DB2, Oracle, and MS SQL Server. It is used within *Sybase Adaptive Server* to build a federation of heterogeneous databases. Users have transparent access to heterogeneous databases. This is achieved through *Component Integration Services (CIS)* that enable the definition of "proxy tables" which are mapped to actual tables in distributed heterogeneous databases. Query optimization is based on a cost model that assumes the availability of access costs of remote tables. *CISs* focus on two aspects of distributed query optimization: (i) query optimization determines optimal join strategy and order and (ii) query decomposition determines the work load to be pushed to remote databases for processing.

*MS SQL Server.* *MS SQL Server* supports data integration through distributed query facilities. These allow SQL Server users to access data stored in other instances of SQL Server and heterogeneous data stored in various relational and non-relational data sources. Access is enabled through OLE DB providers. OLE DB providers expose data in tabular objects called row-sets. SQL Server allows row-sets from OLE DB providers to be referenced in SQL statements as if they were a SQL Server table.

## 5. Discussion

Enabling data integration on the Web and the subsequent need for efficient querying have been targeted by a large research effort. In this survey, we presented a representative segment of that research. The following discussion outlines how the various query optimization issues have been addressed. Figure 7 depicts a classification of different approaches for optimizing queries over Web data integration systems. The figure highlights four categories of query optimization techniques in the context of Web-based data integration. These are cost-based, quality-based, capability-based, and adaptive. For each category, the figure lists the main features of different techniques used to implement query optimizers in that category.

Some of the proposed approaches extend the classical cost model. A major difficulty relates to gathering optimization information (e.g., cardinality, selectivity) from autonomous information sources. Some approaches (e.g., Disco) simply make the assumption that such information is (easily) available from the sources or exported by the wrappers. Others (e.g., Garlic) provide wrapper builders with tools to gather such information (e.g., query sampling) without requiring the involvement of the database. For the two above approaches, if the cost information cannot be obtained, a generic model is generally used. Using a generic model may not be always applicable in the Web context with a large spectrum of autonomous and heterogeneous information sources. In addition, making the assumption that statistics are available or can be easily obtained may not be always reasonable. This suggests that most of the techniques may be more appropriate for intranet-like applications.

For some systems, the main objective in terms of optimization is to reduce the number of sources to be accessed. Sources reduction occurs at query planning time (e.g., Infomaster)

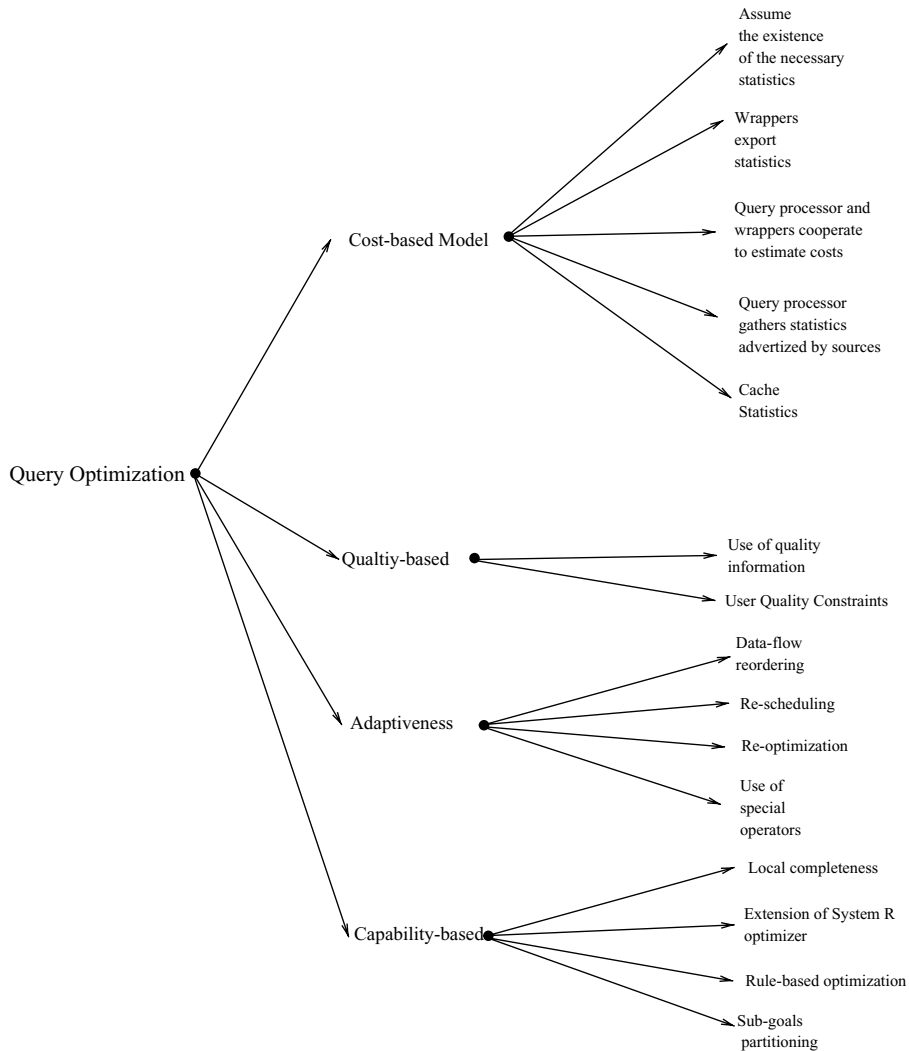


Figure 7. Classification of query optimization techniques.

based on available source descriptions and the user query, and at run time using some special types of queries (e.g., Ariadne). Reducing the number of sources is important in the context of the Web where the number of potential sources for a given query is very large. However, this may not be enough and more optimization would be necessary. One technique (Hermes) bases its optimization on caching statistics about sources calls to estimate the cost of possible execution plans. It also uses a notion of expression invariants to derive equivalent execution query plans. The use of cache may not be sufficient for optimization purpose. It could be used in addition to other optimization techniques.

There is also a trend to use adaptive or dynamic approaches in dealing with query optimization. This is motivated by the intrinsic dynamics of the Web where unpredictable events may occur during the execution of a query. The types of actions that are considered in these approaches fall into one of the following cases: (1) change the query execution plan, (2) change the scheduling of operations in the same query execution plan or in different concurrent query plans, (3) introduce new operators to cater for the unpredictable events, or (4) modify the order of inputs of binary operators. Adaptive techniques have yet to demonstrate their applicability to real Web-based applications with large numbers of information sources. There is also a need to show how they react under heavy fluctuations.

Another interesting direction in optimizing queries over Web data integration systems is the use of information quality. The main motivation is that those quality parameters reflect the user's needs better in such environments. One of the approaches (HiQIQ) uses different classes of quality criteria to select relevant sources and rank alternative query plans. The other approach (ObjectGlobe) allows users to specify quality constraints on the execution of queries. These constraints may relate to the expected results, cost of the query, or time for the first results or complete execution. These quality of service (QoS) constraints are then enforced during the execution of the query. Quality constraints are of crucial importance for some applications. A major issue is how to get related information from sources scattered on the Web. Furthermore, due to the "subjective" nature of some of these constraints, the query optimizer may not always succeed in finding the best plans.

An important issue in processing queries over Web data integration systems is to deal with the often limited query capabilities of sources. These limitations have a direct impact on the optimization process. Some query execution plans may be optimal but not feasible. The different query planning strategies focus generally on finding feasible and optimal subgoal orderings based on available bindings and supported conditions at the information sources. Proposed techniques assume a full knowledge of the query capabilities of every participating source. They rely heavily on the way that information sources are described (e.g., binding patterns and free grammars) and the objective function of the optimizer (e.g., number of sources, response time). Using the same source description model may not always be possible across a large spectrum of information sources.

Table 1 summarizes the main features of the different prototypes and techniques that have been covered in this survey. Commercial products were not included in the table due to the lack of details about them. The features that are covered in the table are:

- *Underlying information sources.* Types of sources that are supported. The majority of Web data integration systems attempt to include a large spectrum of information sources like databases, HTML files, text files, knowledge bases, etc. In general, mediator systems assign a wrapper or interface that hides the underlying heterogeneity to each source. That wrapper takes care of the interaction between the source and the rest of the system.
- *Sources description.* Sources description and advertisement have a direct impact on the source location process and consequently on query optimization.
- *Data model.* Data model that usually serves as the basis for data exchange amongst different components of the data integration systems.
- *Query language.* Specifies the language being used to formulate queries.
- *Query optimization.* The main concept underlying query optimization is summarized.

Table 1. Summary of systems' features.

System	Underlying information sources	Sources description	Data model	Query language	Query optimization
Disco	Structured data	Relations are represented by information types	ODMG data model	Object or relational SQL	Cost-based optimization through wrappers exporting statistics
Garlic	Databases, Image archives, etc.	Garlic objects are specified in GDL, a variant of ODMG Object Description Language	Object-oriented	Extension of SQL	Strategy alternative rules [35] (STAR) are used to build query trees. Cost based optimization is achieved by means of wrapper cooperation
Ariadne	Semistructured data	Embedded catalogs	Loom knowledge representation system	SQL or Loom query language	Planning by rewriting and cost-based optimization. Rewriting is based on rules derived from the properties of distributed and heterogeneous environments and the relational algebra
Hermes	Structured and semistructured data	Domains	Relational	Domain calls	Cost-based optimization based on caching and use of invariants
ObjectGlobe	Semistructured and semistructured data	Sources are described using RDF	XML based model	Extension of SQL	Distribution of query processing capabilities and QoS based optimization
HiQIQ	Structured data	Query correspondence assertions (QCAs) (set oriented equations)	Relational	Extension of SQL	Query planning based on QCAs and several quality criteria for selecting sources
Telegraph	Mainly relational streaming data sources	Relations	Relational	SQL	Based on the concept of <i>eddy</i> , a tuple router interposed between data sources and query processing operators. Adaptiveness is implemented at the level of tuples

Tukwila	Relational databases and XML sources	Relations	Relational	SQL	Operator reordering based on ECA rules. Introduction of two optimized operators: <i>dynamic collectors</i> and <i>double pipelined hash join</i>
Interleaving scheduling and optimization Tsimmis	Relational databases	Relations	Relational	SQL	Adapt scheduling according to arrival rate changes
Information Manifold	Structured and semistructured data	Templates	Object Exchange Model (OEM)	MSL and LOREL	Capability based processing and standard optimization based on the OEM data model
Infomaster	Structured data	Capability records	Relational data model with some O-O features	Conjunctive queries	Conjunctive plans generation and subgoals ordering using the declarative description of source content and capabilities
Infomaster	Structured and semistructured data	Relations and built-in predicates	Knowledge Interchange Format (KIF)	Conjunctive queries	Query optimization based on local completeness where source relations are represented by two views: The <i>liberal view</i> and <i>conservative view</i> that represent a <i>upper</i> and <i>lower bounds</i> of the source relations respectively
Annotated Query Plans	Structured data	Binding patterns	Relational	Datalog	Extension of System-R optimizer by annotating query plans to take into account capabilities of sources
Source Sequencing	Structured data	Access templates	Relational	Datalog	Greedy optimization and sub-goals partitioning based on source capabilities
Selection Queries	Structured data	Context free grammar	Relational	Datalog	Exhaustive modular optimization algorithm and rule based optimization using knowledge about the cost model

## 6. Future trends

As highlighted by this survey, providing efficient access to interconnected databases and other information sources received a sustained interest. Several approaches and techniques have been proposed to deal with the various issues. These issues are particularly challenging due the characteristics of the Web, including amounts of Web data, large heterogeneity spectrum, strict autonomy, volatility, etc. In this section, we shed the light on the major paradigm shift that is taking place on the Web through the two concepts of *Semantic Web* and *Web services*. We believe that this paradigm shift would have an important impact on the issue of data integration and subsequently on query processing and optimization over the Web. While classical data integration is still relevant in some specific domains, providing query facilities over Web services covers a broader scope.

The Web is geared towards an important paradigm shift through the concept of *Semantic Web* [11]. This basically means a transition from documents for human consumption to documents with machine-processable semantics. Allowing machine-processable content would be mainly based via another new paradigm shift, namely *Web services* [9]. A Web Service is a piece of software that can be defined, described and discovered by XML artifacts (<http://www.w3c.org/2002/ws>). Enabling efficient access over this plethora of services is an emerging research challenge for tomorrow's *Semantic Web*. Interacting with Web resources (e.g., databases, XML data stores, etc.) is taking a new direction with the emergence of *Web services*. Web services are gaining momentum and starting to be widely used for diverse activities on the Web. The use of Web services offer new perspectives in addressing issues related to querying the Web. It opens the door for a broader view of information integration on the Web.

An important future direction is to build a *query infrastructure* where Web services would be treated as *first class objects*. Users and applications would submit queries that are resolved by combining invocations to several Web service operations. The surge of research in Web services is due to several reasons that make them an attractive alternative to data-centric approaches. First, the ongoing standardization efforts to describe, publish, search, and invoke Web services have been instrumental in their emergence. Second, Web Services are becoming ubiquitous; access to Web resources, including Web databases, is mainly achieved through Web services. This makes Web services an ideal candidate for enabling querying on the Web. Hence, the Web is evolving from a passive medium for publishing data into a more active platform for conducting business. As a result, there is a need to build an infrastructure to organize and efficiently query Web services where they would be treated as *first-class* objects.

Building the service-based query infrastructure requires addressing several research issues as outlined in the following:

- *Web Service-based query language*. Devise a declarative Web service-based query language. Such language targets Web services as first class objects. Its declarative nature would allow the expression of complex queries in a precise and simple way. Another advantage is that the same query specification holds whatever underlying information is available.

- *Computation model.* The resolution of any query may involve an iterative process between the different players within the infrastructure. We need to devise a *computation model* for the interaction of the different players. They include Web services, Web service providers, Web service registries, and Web service portals. Web service registries are repositories that offer advertising and discovery facilities for Web service providers and Web services consumers. Web service portals are applications that capitalize on Web services by offering one-stop shops for querying.
- *Optimization model.* Performance has a prime importance in successfully deploying a query infrastructure over Web services. It mainly relates to query optimization. Recent literature [18, 48] shows that *Quality of Service* (QoS) of individual Web services is crucial for their competitiveness. In addition, there is an increasing need to provide acceptable *Quality of Service* (QoS) over Web applications. The concept of *QoS* would capture more accurately users and applications' requirements for efficiency and hence for query optimization. The challenge is to define appropriate metrics to characterize *QoS* and devise techniques to measure that *QoS*.

A first step towards a Web service query infrastructure would be to investigate whether the above issues can be addressed using existing techniques in Web-based data integration systems. *A-priori*, this could be possible. However, a fundamental difference between most existing techniques and a Web service-based approach lies in the manipulated objects. The *first class objects* in the Web service-based query infrastructure are *Web services* while *data* is in traditional Web data integration systems. The work in adaptive techniques is relevant to Web services since they are highly dynamic. On the negative side, these techniques are data-centric. The focus is mostly on modifying how some data operators (e.g., join) work and regulating data fragments transfer. These types of techniques would hardly be applicable to Web services where different manipulation operations (e.g., composition) and objects (i.e., services) are used. Techniques for processing queries over sources with limited capabilities may be also relevant. Indeed, access to Web services is limited to a specific set of operations defined by that Web service. However, proposed techniques were also developed for data-centric environments. Their main focus is on ordering accesses to data sources using relational operations.

### Acknowledgment

This research is supported by the National Science Foundation under grant 9983249-EIA. The authors would like to thank the anonymous reviewers and Brahim Medjahed for their valuable comments on earlier drafts of this paper.

### References

1. S. Adali, K.S. Candan, Y. Papakonstantinou, and V.S. Subrahmanian, "Query caching and optimization in distributed mediator systems," in Proceedings of ACM SIGMOD International Conference on Management of Data, Montreal, Canada, June 1996.

2. B. Amann, C. Beeri, I. Fundulaki, and M. Scholl, "Querying XML sources using an ontology-based mediator," in Proceedings of the Tenth International Conference on Cooperative Information Systems, Irvine, CA, USA, Oct. 2002.
3. J.L. Ambite and C.A. Knoblock, "Flexible and scalable query planning in distributed and heterogeneous environments," in Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, Pittsburg, USA, June 1998.
4. L. Amsaleg, P. Bonnet, M.J. Franklin, A. Tomasic, and T. Urhan, "Improving responsiveness for wide-area data access," IEEE Data Engineering Bulletin, vol. 20, no. 3, pp. 3–11, 1997.
5. G.O. Arocena and A.O. Mendelzon, "WebOQL: Restructuring documents, databases and Webs," in Proceedings of the 14th International Conference on Data Engineering, Orlando, Florida, Feb. 1998.
6. R.H. Arpaci-Dusseau, E. Anderson, N. Treuhaf, D.E. Culler, J.M. Hellerstein, D. Patterson, and K. Yelick, "Cluster I/O with River: Making the fast case common," in Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems. ACM Press, May 1999.
7. R. Avnur and J. Hellerstein, "Eddies: Continuously adaptive query processing," in Proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, Texas, USA, May 2000.
8. C. Batini, M. Lenzerini, and S.B. Navathe, "A comparative analysis of methodologies for database schema integration," ACM Computing Surveys, vol. 18, no. 4, 1986.
9. T. Berners-Lee, Services and Semantics: Web Architecture. <http://www.w3.org/2001/04/30-tbl>, 2001.
10. T. Berners-Lee, R. Calliau, A. Luotonen, H.F. Nielsen, and A. Secret, "The world wide Web," CACM, vol. 37, no. 8, 1994.
11. T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic Web," Scientific American, vol. 284, no. 5, 2001.
12. E. Bertino and A. Bouguettaya, "Introduction to the special issue on database technology on the Web," IEEE Internet Computing, vol. 6, no. 4, 2002.
13. L. Bouganim, F. Fabret, C. Mohan, and P. Valduriez, "Dynamic query scheduling in data integration systems," in Proceedings of the 16th International Conference on Data Engineering, San Diego, CA, USA, Feb.–March 2000.
14. A. Bouguettaya, B. Benatallah, and A. Elmagarmid, Interconnecting Heterogeneous Information Systems, Kluwer Academic Publishers (ISBN 0-7923-8216-1), 1998.
15. A. Bouguettaya and R. King, "Large multidatabases: Issues and directions," in IFIP DS-5 Semantics of Interoperable Database Systems, E.K. Hsiao, E.J. Neuhold, and R. Sacks-Davis (Eds.), Elsevier Publishers, 1993.
16. A. Bouguettaya, R. King, and K. Zhao, "FINDIT: A server based approach to finding information in large scale heterogeneous databases," in First International Workshop on Interoperability in Multidatabase Systems, Kyoto, Japan, April 1991.
17. R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, and K. Stocker, "ObjectGlobe: Ubiquitous query processing on the internet," The VLDB Journal, vol. 10, no. 1, 2001.
18. M. Conti, M. Kumar, S.K. Das, and B.A. Shirazi, "Quality of service issues in internet Web services," IEEE Transactions on Computers, vol. 51, no. 6, 2001.
19. W. Du, R. Krishnamurthy, and M.-C. Shan, "Query optimization in a heterogeneous DBMS," in Proceedings of the 18th International Conference on Very Large Data Bases (VLDB), Vancouver, Canada, 1992.
20. O.M. Duschka, Query Planning and Optimization in Information Integration, PhD thesis, Computer Science Department, Stanford University, 1997.
21. O.M. Duschka and M.R. Genesereth, "Query planning in infomaster," in Proceedings of the Twelfth Annual ACM Symposium on Applied Computing, SAC '97, San Jose, CA, USA, Feb. 1997.
22. D. Florescu, A. Levy, I. Manolescu, and D. Suciu, "Query optimization in the presence of limited access patterns," in Proceedings ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, USA, June 1999.
23. H. Garcia-Molina, W. Labio, and R. Yerneni, "Capability sensitive query processing on internet sources," in Proceedings of the 15th International Conference on Data Engineering, Sydney, Australia, March 1999.
24. 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, vol. 8, no. 2, 1997.

25. G. Gardarin, F. Sha, and Z. Tang, "Calibrating the query optimizer cost model of IRO-DB," in Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB), Bombay, India, Sept. 1996.
26. G. Graefe, "Query evaluation techniques for large databases," *ACM Computing Survey*, vol. 25, no. 2, 1993.
27. L. Gravano and Y. Papakonstantinou, "Mediating and metasearching on the Internet," *IEEE Data Engineering Bulletin*, vol. 21, no. 2, 1998.
28. L.M. Haas, D. Kossmann, E.L. Wimmers, and J. Yang, "Optimizing queries across diverse data sources," in Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB), Athens, Greece, Aug. 1997.
29. D. Heimbigner and D. McLeod, "A federated architecture for information systems," *ACM Transactions on Office Information Systems*, vol. 3, no. 3, 1985.
30. J.M. Hellerstein, M.J. Franklin, S. Chnadrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Ramana, and M.A. Shah, "Adaptive query processing: Technology in evolution," *IEEE Data Engineering Bulletin*, vol. 23, no. 2, 2000.
31. A.R. Hurson, M.W. Bright, and H. Pakzad, *Multidatabase Systems: An Advanced Solution for Global Information Sharing*, IEEE Computer Society Press: Los Alamitos, CA, 1994.
32. Z. Ives, D. Florescu, M. Friedman, A. Levy, and D. Weld, "An adaptive query execution system for data integration," in Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, PA, USA, June 1999.
33. D. Konopnicki and O. Shmueli, "WWW information gathering: The W3QL query language and the W3QS system," *ACM Transaction on Database Systems*, vol. 23, no. 4, 1998.
34. A. Levy, A. Rajaraman, and J. Ordille, "Querying heterogeneous information sources using source descriptions," in Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB), Bombay, India, 1996.
35. G. Lohman, "Grammar-like functional rules for representing query optimization alternatives," in Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, 1988.
36. R. MacGregor, "A deductive pattern matcher," in Proceedings of AAAI-88, The National Conference on Artificial Intelligence, St. Paul, MN, USA, 1988.
37. E. Mena, A. Illarramendi, V. Kashyap, and A. Sheth, "OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies," *International Journal Distributed and Parallel Databases*, vol. 8, no. 2, 2000.
38. A.O. Mendelzon, G.A. Mihaila, and T. Milo, "Querying the world wide Web," *International Journal on Digital Libraries*, vol. 1, no. 1, 1997.
39. H. Naacke, G. Gardarin, and A. Tomasic, "Leveraging mediator cost models with heterogeneous data sources," in Proceedings of the 14th International Conference on Data Engineering, Orlando, Florida, Feb. 1998.
40. F. Naumann and U. Lesser, "Quality-driven integration of heterogeneous information systems," in Proceedings of the 25th International Conference on Very Large Data Bases (VLDB), Edinburgh, UK, Sept. 1999.
41. M. Nodine, W. Bohrer, and A.H.H. Ngu, "Semantic brokering over dynamic heterogeneous data sources in InfoSleuth," in Proceedings of the 15th International Conference on Data Engineering, Sydney, Australia, March 1999.
42. F. Ozcan, S. Nural, P. Koskal, C. Evrendilek, and A. Dogac, "Dynamic query optimization in multidatabases," *IEEE Data Engineering Bulletin*, vol. 20, no. 3, 1997.
43. M. Ouzzani, B. Benatallah, and A. Bouguettaya, "Ontological approach for information discovery in internet databases," *Distributed and Parallel Databases, an International Journal*, vol. 8, no. 3, 2000.
44. M.T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, Prentice Hall, 1999.
45. J.S. Quarterman and J.C. Hoskins, "Notable computer networks," *Communications of the ACM*, vol. 29, no. 10, 1986.
46. M.T. Roth, F. Ozcan, and L.M. Haas, "Cost models do matter: Providing cost information for diverse data sources in a federated system," in Proceedings of 25th International Conference on Very Large Data Bases, Edinburgh, Scotland, UK, Sept. 1999.
47. M.T. Roth and P. Schwarz, "Don't scrap it, wrap it! A wrapper architecture for legacy data sources," in Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB), Athens, Greece, Aug. 1997.

48. A. Ruiz, R. Corchuelo, Duran, and M. Toro, "Automated support for quality requirements in web-based systems," in Proceedings of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems, Bologna, Italy, IEEE, Oct.–Nov. 2001.
49. P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price, "Access path selection in a relational database management system," in Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, P.A. Bernstein (Ed), Boston, Massachusetts, ACM, May 30–June 1, 1979.
50. M.-C. Shan, "Pegasus architecture and design principles," in Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, June 1993.
51. A.P. Sheth and J.A. Larson, "Federated database systems and managing distributed, heterogeneous, and autonomous databases," *ACM Computing Surveys*, vol. 22, no. 3, pp. 183–226, 1990.
52. S. Spaccapietra and C. Parent, "A step forward in solving structural conflicts," *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 2, 1994.
53. A. Tomasic, L. Rashid, and P. Valduriez, "Scaling heterogeneous database and design of DISCO," in Proceedings of the 16th International Conference on Distributing Computing Systems (ICDCS), Hong Kong, May 1996.
54. G. Wiederhold, "Mediators in the architecture of future information systems," *IEEE Computer*, vol. 25, no. 3, 1992.
55. Y. Yerneni, C. Li, J. Ullman, and H. Garcia-Molina, "Optimizing large join queries in mediation systems," in Proceedings of the International Conference Database Theory, Al Qods, Jan. 1999.
56. K. Zhao, R. King, and A. Bouguettaya, "Incremental specification of views across databases," in First International Workshop on Interoperability in Multidatabase Systems, Kyoto, Japan, April 1991.
57. Q. Zhu and P. Larson, "Global query processing and optimization in the CORDS multidatabase system," in Proceedings of ACM SIGMOD International Conference on Management of Data, San Jose, CA, USA, 1995.