Data integration through database federation

by L. M. Haas
E. T. Lin
M. A. Roth

In a large modern enterprise, it is almost inevitable that different parts of the organization will use different systems to produce, store, and search their critical data. Yet, it is only by combining the information from these various systems that the enterprise can realize the full value of the data they contain. Database federation is one approach to data integration in which middleware, consisting of a relational database management system, provides uniform access to a number of heterogeneous data sources. In this paper, we describe the basics of database federation, introduce several styles of database federation, and outline the conditions under which each style of federation should be used. We discuss the benefits of an information integration solution based on database technology, and we demonstrate the utility of the database federation approach through a number of usage scenarios involving IBM's DB2 product.

In a large modern enterprise, it is inevitable that different parts of the organization will use different systems to produce, store, and search their critical data. This diversity of data sources is caused by many factors including lack of coordination among company units, different rates of adopting new technology, mergers and acquisitions, and geographic separation of collaborating groups. Yet, it is only by combining the information from these various systems that the enterprise can realize the full value of the data they contain.

In the finance industry, mergers are an almost commonplace occurrence. The company resulting from a merger inherits the data stores of the original institutions. Many of those stores will be relational database management systems, but often from different manufacturers. One company may have used primarily Sybase, Inc. products, whereas another may have used products from Informix Software, Inc. They may both have had one or more document management systems, such as Documentum[1] or IBM Content Manager,[2] for storing text documents. Each may have had applications that compute important information (e.g., the risk of granting a loan to a given customer) or that mine for information about customers' buying patterns.

After the merger, the new company needs to be able to access the customer information from both sets of data stores, to analyze its new portfolio using existing and new applications, and, in general, to use the combined resources of both institutions through a common interface. The company needs to be able to identify common customers and consolidate their accounts, even though the customer data may be stored in different databases and in different formats. In addition, the company must be able to combine the legacy data with new data available on the Web or from its business partners. These are all aspects of data integration,[3] and all pose hefty challenges.

0018-8670/02/$5.00 © 2002 IBM

Sometimes the need for data integration may be as simple as getting a reading from a sensor (say, the temperature at some stage in a manufacturing process), and comparing it to a baseline value derived from historical data stored in a database. Or perhaps data from a spreadsheet are to be compared with data from several databases containing experimental results, or perhaps one has to find customers with large credit card balances whose income is less than some threshold. The need for data integration is ubiquitous in today's business world.

There are many mechanisms for integrating data. These include application-specific solutions, application-integration frameworks, workflow (or business process integration) frameworks, digital libraries with portal-style or meta-search-engine integration, data warehousing, and database federation. We discuss each briefly.

Perhaps the most common means of data integration is via special-purpose applications that access sources of interest directly and combine the data retrieved from those sources with the application itself. This approach always works, but it is expensive (in terms of both time and skills), fragile (changes to the underlying sources may all too easily break the application), and hard to extend (a new data source requires new code to be written).

Application-integration frameworks[4,5] and component-based frameworks[6] provide a more promising approach. Such systems typically employ a standard data or programming model, such as CORBA** (Common Object Request Broker Architecture), J2EE** (Java 2 Platform, Enterprise Edition), and so on. They provide well-defined interfaces to the application for accessing data and other applications and for adding new data sources, typically by writing an adaptor for the data source that meets the framework's adaptor interface. These frameworks protect the application somewhat from changes in the data sources (if the source changes, the adaptor may have to change, but the application may never see it). Adding a new source may be easier (although a new adaptor may need to be written, the change is more isolated, and the adaptor may already exist and be available for purchase). The application programmer is not required to have detailed systems knowledge, so applications will typically be easier to write. However, such systems do not necessarily address data integration issues; if combination, analysis, or comparison of the data received from the various sources is needed, the application developer must provide that code.

Similarly, workflow systems[7] provide application developers with a higher-level abstraction against which to program, but using a more process-oriented model. Again, these systems provide some protection to the application against changes in the environment, and they additionally provide support for routine results from one source to other sources. However, they still provide only limited help for comparing and manipulating data.

Digital libraries, which collect results from multiple different data sources in response to a user's request, represent another style of data integration. For example, Stanford's InfoBus architecture[8] provides a bibliographic search service that looks through several reference repositories and returns a single list of results. Usually the sources are similar in function (e.g., all store text documents, bibliographies, URLs [uniform resource locators], images, etc.). For most of these "meta search engines," no combination of results is done, except perhaps for normalization of relevance scores or formats—all are returned as part of one list, possibly sorted by estimated relevance. Similarly, *portals* provide a means of collecting information—perhaps from quite different data sources—and putting the results together for the user to see in conjunction. Results from the different searches are typically displayed in different areas of the interface, although portal frameworks may allow application code to be written to combine the results in some other way. Portals are often built on top of some sort of application-integration framework, providing a simple and amazingly powerful way to quickly get access to a variety of information. IBM's Enterprise Information Portal, for example, can retrieve various types of data from a range of repositories, including text search engines, document management systems, image stores, and relational databases.[9] Yet again, however, the application developer must write code to do any more sophisticated analysis, comparison, or integration that is required.

Data warehouses and database federation, by contrast, provide users with a powerful, high-level query language that can be used to combine, contrast, analyze, and otherwise manipulate their data. Technology for optimizing queries ensures that they are answered efficiently, even though the queries are posed nonprocedurally, greatly easing application development. A data warehouse is built by loading data from one or more data sources into a newly de-

fined schema in a relational database. The data are often cleansed and transformed in the load process. Changes in the underlying sources may cause changes to the load process, but the part of the application that deals with data analysis is protected. New data sources may introduce changes to the schema, requiring that a new load process for the new data be defined. SQL (Structured Query Language) views can further protect the application from such evolutions. However, any functions of the data source that are not a standard part of a relational database management system must be re-implemented in the warehouse or as part of the application.

A solution based on warehousing alone may not be possible or cost effective for various reasons. For example, it is not always feasible to move data from their original location to a data warehouse, and as described above, warehousing comes with its own maintenance and implementation costs. Database federation (or *mediation* as it is sometimes referred to in the literature) provides users with a virtual data warehouse, without necessarily moving any of the data. Thus, in addition to the benefits of a warehouse, it can provide access to "live" data and functions. A single arbitrarily complex query can efficiently combine data from multiple sources of different types, even if those sources themselves do not possess all the functionality needed to answer such a query. In other words, a federated database system can optimize queries and compensate for SQL function that may be lacking in a data source. Additionally, queries can exploit the specialized functions of a data source, so no functionality is lost in accessing the source through the federated system.

Of course, database federation adds another component between the client application and the data, and this extra layer introduces performance tradeoffs. Indeed, one should not expect that introducing a new layer would reduce access time for any *single* federated data source. The key performance advantage offered by database federation is the ability to efficiently combine data from *multiple* sources in a single SQL statement. Two components of the federated server contribute to this: query rewrite and cost-based optimization.[10–12] The query rewrite component of a federated server can aggressively rewrite a user's query into a semantically equivalent form that can be more efficiently executed. A cost-based optimizer can search a large space of access strategies and choose a global execution plan that is optimal across both local tables and the federated data sources involved in the query.

Note that in some cases, a query issued to a federated server with a sophisticated query processing engine may actually outperform the same query issued directly to the data source itself. During query rewrite phase, a user's query can be rewritten in such a way that the federated data source can choose a more efficient execution plan than it would if it were given the user's original query.

The goal of this paper is to demonstrate that database federation is a fundamental tool for data integration. The rest of the paper is organized as follows. In the next section, we discuss database federation in greater detail and describe the architecture for DB2* (Database 2*) database federation. There are several styles of database federation, and these are introduced in the following section, in which we also discuss when each style of federation should be used. In the next section, we discuss the advantages of building an integration solution on database technology, and in the section that follows we demonstrate the utility of this approach through a number of usage scenarios. We conclude with a summary and some thoughts on future work.

The basics of database federation

The term "database federation" refers to an architecture in which middleware, consisting of a relational database management system, provides uniform access to a number of heterogeneous data sources. The data sources are *federated*, that is, they are linked together into a unified system by the database management system. The system shields its users from the need to know what the sources are, what hardware and software they run on, how they are accessed (via what programming interface or language), and even how the data stored in these sources are modeled and managed. Instead, a database federation looks to the application developer like a single database management system. The user can search for information and manipulate data using the full power of the SQL language. A single query may access data from multiple sources, joining and restricting, aggregating and analyzing the data at will. Yet the sources may not be database systems at all, but in fact could be anything from sensors to flat files to application programs to XML (Extensible Markup Language), and so on.

Many research projects and a few commercial systems have implemented and evolved the concept of database federation. Pioneering research projects included TSIMMIS[13] and HERMES,[14] which used database concepts to implement "mediators," special-purpose query engines that use nonprocedural specifications to integrate specific data sources. DISCO[15] and Pegasus[16] were closer in feel to true database federation. DISCO focused on a design scaling up to many heterogeneous data sources. Pegasus had its own data model and query language, and it had a functional object model that allowed great flexibility for modeling data sources. Each created its own database management system from scratch. Garlic[17] was the first research project to exploit the full power of a standard relational database (DB2[18]), although it extended the language and data model to support some object-oriented features. The wrapper architecture[19] and cross-source query optimization[20] of Garlic are now fundamental components of IBM's federated database offerings.

Meanwhile, in the commercial world, the first steps toward federation involved "gateways." A gateway allows a database management system to route a query to another database system. Initially, gateway products only allowed access to homogeneous sources, often from the same manufacturer, and queries could reference data in only one data source. Over time, more elaborate gateway products appeared, such as iWay Software's EDA/SQL Gateways.[21] These products allowed access to heterogeneous relational systems and to nonrelational systems that provided an Open Database Connectivity (ODBC) driver.[22] DB2 DataJoiner*[23] was the first commercial system to really use database federation. It provided a full database engine with the ability to combine data from multiple heterogeneous relational sources in a single query, and to optimize that query for good performance. Microsoft's Access[24] allowed access to a larger set of data sources, but was geared toward smaller applications, as opposed to the mission-critical large enterprise applications that DataJoiner supported.

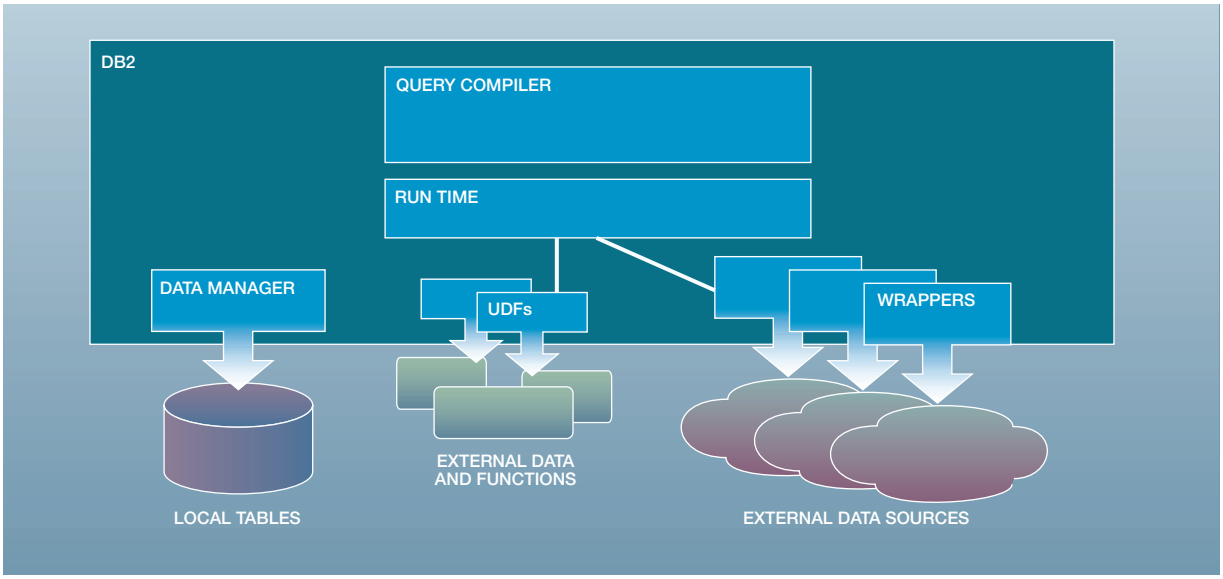
DataJoiner and Garlic "grew up" together, with DataJoiner focused on robust and efficient queries over a limited range of mostly relational sources, whereas Garlic focused on extensibility to a much more heterogeneous set of sources. Today, the best ideas from both projects have been incorporated into DB2,[25] which also enabled the use of user-defined functions to "federate" simple data sources.[26] DB2 thus supports a very rich notion of database federation that can be exploited for data integration. DiscoveryLink*,[27] for example, is an IBM solution that applies DB2 technology to integrate data in the life sciences.

IBM's database federation offerings have pursued several goals. The most basic goal is *transparency*, which requires masking from the user the differences, idiosyncrasies, and implementations of the underlying data sources. This allows applications to be written as if all the data were in a single database, although, in fact, the data may be stored in a heterogeneous collection of data sources. A second key goal is to *support heterogeneity*, or the ability to accommodate a broad range of data sources, without restriction of hardware, software, data model, interface, or protocols. A *high degree of function* provides users with the best of both worlds: not only rich, standard-compliant DB2 SQL capabilities against all the data in the federation, but also the ability to exploit functions of the data sources that the federated engine may lack.

The fourth goal is *extensibility*, the ability to add new data sources dynamically in order to meet the changing needs of the business. *Openness* is an important corollary; DB2 products support the appropriate standards[28,29] in order to ensure the federation can be extended with standard-compliant components. Further, joining a federation should not compromise the *autonomy* of individual data sources. Existing applications run unchanged; data are neither moved nor modified; interfaces remain the same. Last but not least, DB2 database federation technology *optimizes queries for performance*. Relational queries are nonprocedural. When a relational query spans multiple data sources, making the wrong decisions about how to execute it can be costly in terms of resources. Hence, the DB2 federated engine extends the capabilities of a traditional optimizer[12] to handle queries involving federated data sources.

DB2 architecture for database federation. In the remainder of the paper, we focus on the database federation capabilities of DB2. Figure 1 depicts the overall architecture of a DB2 database federation. Users access the federation via any DB2 interface (CLI [Call Level Interface], ODBC, JDBC** [Java** Database Connectivity], etc.). The key component in this architecture is the DB2 federated engine itself. This engine consists of a Starburst-style query processor[30] that compiles and optimizes SQL queries, a run-time interpreter for driving the query execution, a data manager that controls a local store, and several ex-

Figure 1 DB2 architecture for database federation



tension mechanisms that allow access to external data, including *user-defined functions* and *wrappers*.

A user-defined function (UDF) is a routine written by an application developer and registered with the database. User-defined functions may take input parameters and return either a scalar result, or a table of data. They can be used to combine functions already supported by the database, or to provide access to a specific function provided by an external source.

A wrapper mediates between the DB2 engine and one or more data sources, mapping the data model of the sources to the DB2 data model and transforming DB2 data operations into requests that the sources can handle. DB2 provides wrappers for many popular data sources, such as Informix, Oracle, and MS SQL server,[25] as well as wrappers for specialized data sources such as Documentum and BLAST.[31] In addition, there is a toolkit for customers and third-party vendors to build wrappers for their own sources.

Clearly, there are different styles of federation available with this architecture. In the next section, we explore each of these styles in greater detail and give guidelines for when each style is appropriate.

DB2 styles of federation

As shown in Figure 2, the federated architecture of DB2 offers a range of alternatives for federation, from the simplicity of a scalar user-defined function to the flexibility of the DB2 wrapper architecture. The figure illustrates that a natural trade-off exists between the level of federation and the effort involved to achieve federation. In this section we describe points along this range and provide a decision tree that helps to illustrate which type of federation is most appropriate for a given integration scenario.

Scalar UDFs: Federating function. The simplest form of a UDF is a scalar function, which can take data from the surrounding SQL statement as input and produce a single scalar result. Scalar UDFs provide a way to federate *function* with data in DB2, combining data from one source with a function provided by another in a single statement. UDFs can provide two major benefits to client applications: (1) a simpler programming model and (2) greater performance, achieved by reducing network traffic and path length between the calling application, data, and function.

For example, the user-defined function db2mq.mqsend() ships with DB2 and allows a user to send a message to an MQSeries[32] queue. By invoking this function

within a SELECT statement, data can be moved from a database table to an MQ queue without the client application accessing the queue directly:

```
SELECT db2mq.mqsend(a.headline)
FROM Articles a
WHERE a.article_timestamp >= CURRENT TIMESTAMP
```

Scalar functions can also be used as a simple way to bring data into the engine. For example, the function db2mq.mqreceive( ) can be used to pull the next message from a queue. Because it is implemented as a UDF, this function can be used in a complex query to combine the message with other data from the federation.

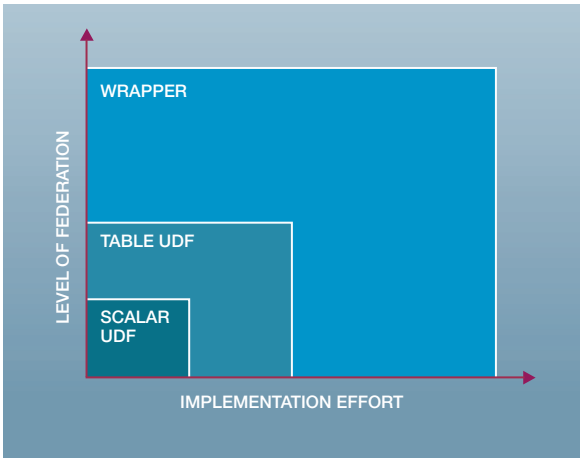
Table UDFs: Federating data. A table function is a more sophisticated UDF that produces a table as output and can appear wherever a table can be referenced in an SQL statement. A row function is a special case of a table function that returns 1 row. A table function can retrieve a set of data and reformat the data into rows and columns, providing a simple way to federate external data. Because a table function can appear anywhere a table can, the full power of SQL can be applied to the resulting data. Furthermore, views can be created on top of the UDF in order to make the external data appear even more like a local table to client applications.

For example, a table function called addressbook( ) can be used to access an address book stored in a Lotus Notes* database that contains sales contact information. This function can be invoked within a query that joins the result of the function call with a local table that contains company profile information to return sales contacts at financial companies whose gross revenues are greater than $50 million:

```
SELECT a.first, a.last, a.phone, a.email
FROM TABLE (addressbook( )) AS a, Company_Profiles c
WHERE c.industry = 'FINANCIAL' AND c.revenue >
                    50,000,000 AND c.name = a.company_name
```

Although table functions can be used just like tables, they look more like functions. Table functions can take arguments that can be used to restrict the data to be returned. For example, a function to get information about files on the local disk might take one argument that determines the directory to search, and a second that determines what file types are of interest. But the table function can only filter using those predicates that it was designed to handle. For example, in the query below, the dir( ) function could not be passed the predicate on the column last_modified_date:

Figure 2 Different styles of federation




```
SELECT f.filename, f.author, f.last_modified_date
FROM TABLE (dir('\laura\papers', '.pdf')) AS f
WHERE f.last_modified_date > '07/04/2002'
```

Fortunately, the DB2 engine can apply any other predicates to the result returned by the table function. In this example, the engine would filter the results of the table function and return only those modified more recently than July 2002.

Wrappers: Federating function and data. The wrapper architecture provides the most powerful and flexible infrastructure for federation.[33] It provides the means to integrate both function and data by raising the level of federation from a single function or set of data to that of an entire external data source. Client applications can transparently access and use the full power of the query language on data managed by these sources as though DB2 itself manages the data.

For example, consider a set of scientists at a university working in a drug discovery laboratory. The scientists store chemical compound data and experimental results in an Oracle database. The university also has access to Lotus Extended Search[34] (LES), a Web search engine that can perform searches across multiple Web search sites. The scientists use this search engine to retrieve research articles from other scientists. Both the Oracle database and the

search engine can be transparently accessed from DB2 via wrappers, allowing the scientists to ask a single query that combines data and functions from both sources.

The Oracle wrapper maps the compound data and experimental results tables stored in Oracle to nicknames, which can appear wherever a table can appear in an SQL query (for example, the FROM clause). The LES wrapper provides a nickname for a list of articles that can be searched. Each article has a title, subject, and URL. The search functionality itself is mapped as an SQL function that takes two arguments, the section of the article to search (title, subject, or both), and a set of key words, and returns a score that measures the relevance of the article. For example, a common task the scientists might undertake would be to find other research reports on chemical compounds that achieved a certain result in their experiments:

```
SELECT c.name, a.URL
FROM Compounds c, Experiments e, Articles a
WHERE e.result < 1.1e-9 and e.id = c.id and
search(a.subject, c.name) > 0
```

Because Oracle is itself a relational database, the DB2 optimizer can choose a plan in which both the predicate on the Experiments table (e.result < 1.1e-9) and the join between the Compounds and Experiments tables (e.id = c.id) are pushed down to the Oracle database to execute, and this results in only those compounds with the right test results being returned to DB2. DB2 then routes the names of the matching compounds to LES, which will retrieve relevant articles and return their URLs back to DB2.

DB2 supplies wrappers for a variety of relational and nonrelational sources,[18] and also provides a toolkit for third-party vendors and customers to develop wrappers for their own data sources.[17,19] The wrapper toolkit provides an external interface to the wrapper architecture, allowing a developer to federate a new type of data source. The wrapper architecture enables the following federated features:

- *Multiserver integration*. Unlike a UDF, a wrapper can easily provide access to multiple external sources of a particular type. The DB2 engine acts as a traffic cop, maintaining the state information for each server, managing connections, decomposing queries into fragments that each server can handle, and managing transactions across servers.
- *Multidata-set integration and multioperation integration*. Table UDFs are well suited to integrate a single external operation and set of data into the federation. The wrapper architecture expands on this capability by providing the infrastructure to model multiple data sets and multiple operations. Data sets are modeled as nicknames, and nicknames can appear wherever a table can appear in an SQL statement. Operations include query, insert, update, and delete. The wrapper participates in the planning and execution of these operations, and translates them into the corresponding operations supported by the server.

The multidata-set and multioperation features introduce the notion of *common query framework* for external sources. The common query framework is the way in which the wrapper architecture realizes the goal of high function. That is, the framework enables (1) the full power of operations expressible in SQL over the external source's data, and (2) specialized functions supported by the external source to be invoked in a query even though DB2 does not natively support the function.

Part 1 of the goal is addressed through *function compensation*. If the external source does not support the full power of DB2's SQL, client applications will not suffer any loss of query power over those data, because DB2 will automatically compensate for any differences between DB2's capabilities and those of the external source. So, for example, if a data source does not support ORDER BY, DB2 will retrieve the data from the source and perform the sort.

Part 2 of the goal is addressed through *function mappings*. Function mappings can be used to declaratively expose functions supported by the external source through the DB2 query language. For example, a chemical structure store may have the ability to find structurally similar chemical compounds. Although that function is not present in DB2, it can still be exploited in DB2 queries, as long as there is a mapping that tells DB2 which source implements the function.

If the external source implements new functions, the wrapper itself may not require modification, depending on the data source and what the wrapper needs to do to map the function. For sources that invoke new functions in a predictable way, the database administrator (DBA) need only register

the new function mappings. They can then be immediately used in DB2 queries.

- *Optimization*. Operations to be performed outside of the DB2 engine may have significant cost implications for the queries that contain them, and it is critical for DB2 to be aware of these implications when choosing a strategy for executing the query. The wrapper architecture includes a flexible framework for wrapper writers to provide input to the DB2 query optimizer about the cost of their operations and the size of the data that will be returned. This framework includes a dialog between the optimizer and wrapper at query compilation time, allowing the wrapper writer to provide information based on the immediate context of the query.

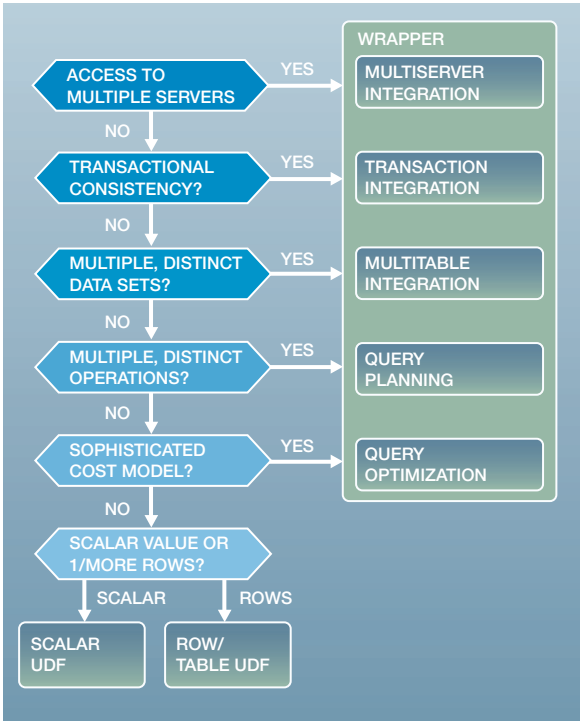
  This information is particularly crucial when the optimizer must consider alternatives for cross-source operations, as these types of operations explode the set of possible execution strategies. Reference 11 provides an example of an image server that can support two kinds of image search, one of which is substantially more expensive than the other. Because the wrapper can provide this information to the DB2 optimizer, the optimizer is able to choose a plan that applies the more expensive operation after another predicate has filtered out much of the data.

- *Transactional integration*. DB2 acts as a transaction manager for operations performed on external sources, and the wrapper architecture provides the infrastructure for wrappers to participate as resource managers in a DB2 transaction. DB2 maintains a list of wrappers that have participated in the transaction, and it ensures that they are notified at transaction commit or rollback time, giving a wrapper an opportunity to invoke the appropriate routine on the external source.

Determining the style of database federation to use. To determine which level of federation is appropriate for a particular integration problem, it is important to characterize the purpose and desired properties of the integration effort. UDFs are easy to implement, but have limited support for data modeling and no support for transactional integration. Wrappers are powerful, but rely on more advanced capabilities of the external source and require a more advanced skill set to implement.

Every application is different, and each can be solved via a "small matter of programming" using the different federation alternatives. However, it is usually true that one alternative is more suitable than the others for a given problem. Figure 3 presents a decision tree that helps characterize the style of federation most appropriate for a particular integration problem.

Figure 3 Determining the style of federation to use



The decision tree is based on a series of YES/NO questions. Consistently answering NO to the questions indicates that the integration problem is sufficiently contained that the best solution is likely to be a UDF. A YES answer indicates that the integration problem exhibits a characteristic that is best handled by the wrapper architecture. The decision tree points to the feature of the wrapper architecture that addresses that characteristic. Next we consider the nodes in the decision tree in more detail.

1. *Does the integration problem reach out to multiple data sources of the same type? If so, is there a benefit to modeling such servers?*

If the server concept is key to the integration problem, then the wrapper architecture is likely to be an

appropriate solution. For example, a function that retrieves the temperature from an on-line thermometer does not require the notion of a server, and the wrapper solution may be overkill. On the other hand, suppose the goal is to integrate a set of Lotus Notes databases. It is possible to write a UDF to access multiple databases; however, the burden is on the UDF developer to take the appropriate information to identify the databases as arguments, manage connections to the databases, and use the scratchpad to store any state information. Furthermore, the lifetime of the scratchpad is within a single statement, so UDF invocations from separate statements will require their own connections. For this integration problem, the multiserver support offered by the wrapper architecture, including connection management, will help to nicely manage access to those databases.

2. *Is transactional consistency important for the integration problem?*

The UDF architecture does not support transactional consistency, whereas the wrapper architecture offers both one-phase and (in a future release) two-phase commit support. If transactional consistency is crucial for the integration problem, and the source that has the data or functions needed is able to participate as a resource in a transaction, the wrapper architecture is the only choice that can provide that level of support.

3. *Are there multiple, distinct data sets to be accessed? Are there multiple, distinct operations to be federated?*

An operation is a specific external action that can be performed on the data to be integrated. Examples of operations include data retrieval, search, insert, update, and delete. Table UDFs are well suited to integrate a single external operation and set of data into DB2. When there are multiple data sets and/or multiple operations involved, the wrapper architecture may more easily provide the infrastructure for modeling those data sets and operations. This is particularly true when there are restrictions on how multiple operations can be combined, or cost implications that depend on how they are combined. When using the wrapper architecture, data sets become separate nicknames, and flexible query planning and query execution components allow the wrapper writer to control the set of operations the wrapper will support, and how those operations will be executed. In addition, the wrapper architecture includes APIs (application programming interfaces) for the wrapper writer to provide optimization information on an operation-by-operation basis. The wrapper writer is allowed to examine the operation to be performed to determine which portions of the operation the data source can execute. The wrapper reports this information and provides cost and cardinality estimates based on the operations, parameters, persistent cost information stored in the system catalogs, and state information stored in the wrapper. With this cost information the DB2 optimizer can determine an optimal execution plan.

4. *Is there a sophisticated cost model associated with the federated operations?*

The UDF architecture provides some support for providing static cost information. If the operation is typically executed as part of a simple, single-table query, a table function may be the appropriate vehicle for federation. However, if the operation is likely to be invoked as part of a more complex query, and its position in the query plan may greatly impact the performance of the query, it may be worthwhile to use the wrapper architecture to federate this operation, just to exploit the flexible costing infrastructure.

If the answer to all of the above questions is no, then a UDF is likely to be an appropriate choice for federation. In this case, the final question to ask is whether the integrated operation should return a scalar value, or a multirow set. If the operation returns a scalar value, the right choice is a scalar UDF, and if the operation can return a result set, a table UDF is the appropriate choice.

If the answer to any of the questions is yes, the wrapper architecture provides at least one valuable feature that addresses some aspect of the integration problem, and it is worthwhile to consider writing a wrapper to exploit that feature. Note that because the architecture is designed to be flexible, it is possible to implement important pieces of a wrapper without writing a complete wrapper. For example, if the function being integrated is really a simple scalar UDF but transactional integration is key, it is possible to implement that function within a simple wrapper that exploits the transactional architecture, and only minimally implements the data modeling and query optimization aspects. On the other hand, if the operation on the data source is a highly sophisticated read-only function with significant cost implications, it is possible to write a wrapper that

focuses on the optimization component and does not address transaction issues at all.

Why use a database engine for data integration?

Database federation centers on a relational database engine. That engine is an essential ingredient for integrating data, because it significantly raises the level of abstraction for the application developer. Rather than deal with the details of how to access individual sources, and in what order, the application developer gets the benefits of a powerful nonprocedural language, support for data integrity, a local store to use for persisting data and meta-data as needed, and the use (or reuse) of the many tools and applications already written for "normal" relational systems. In this section, we look at each of these benefits in more detail.

The virtues of SQL and the relational data model have often been extolled since their invention in the late 1960s and early 1970s.[35–37] Foremost among these are the simplicity of the model and the nonprocedural nature of the language, which together allow the application developer to specify *what* data are needed, and not *how* to find the data. The system automatically finds a path to the individual relations, and decides how to combine them in order to return the desired result. With the invention of query optimization in the late 1970s,[12] systems were able to choose wisely among alternative execution strategies, greatly enhancing the performance and expanding the range of queries the systems could effectively handle. Over the years, the language has been continuously enriched with new constructs. It is now possible to write recursive queries, perform statistical analyses, define virtual tables on the fly, set defaults, take different actions based on the data retrieved, and so on.[38] User-defined functions and stored procedures make it possible to execute much of the application logic in the database, as close as possible to the data for greatest efficiency. Database federation extends these advantages to data that are not stored in the local store. Now applications requiring distributed data can be written as easily as single source applications, using all the power of SQL.

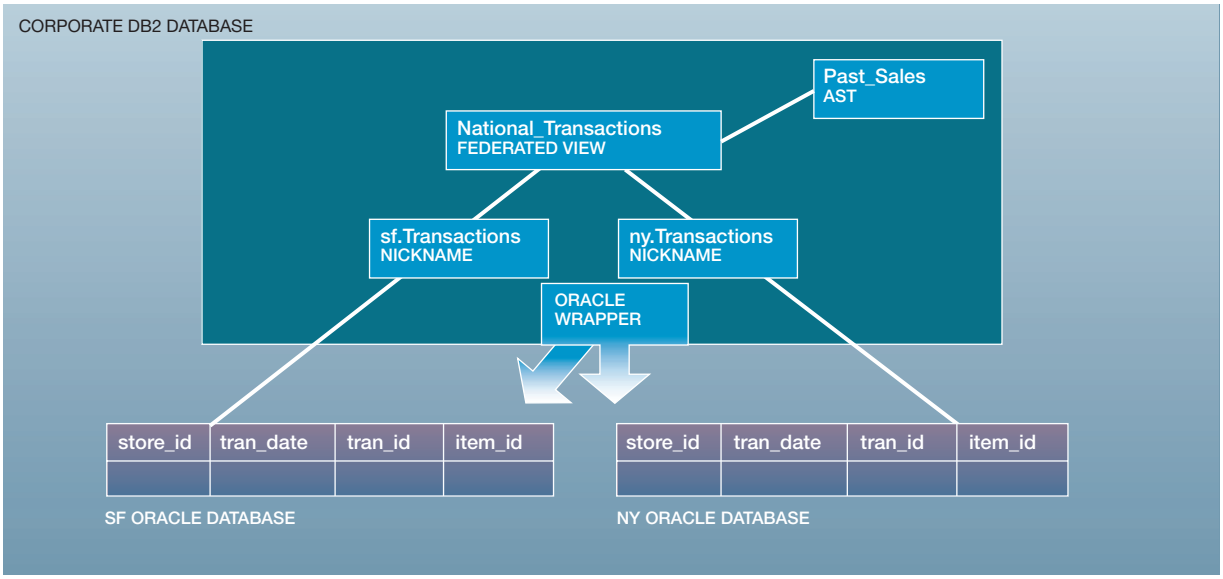
Another fundamental aspect of relational systems is that they protect data integrity. Transaction support[39] guarantees an all-or-nothing semantics for updates. The extension of commit protocols to distributed environments[40] is well understood and many database systems implement one or two-phase commit protocols. These add great value, of course, for integrating data from several sources, allowing integrity guarantees to be offered and building users' trust in the system. Sophisticated integrity constraints[41] and active mechanisms for enforcing constraints of various kinds[42] may also be extended to protect relationships across federated data. Were we to build a data integration engine on some other platform, all of this would need to be built from scratch.

The local store is a further important aid to applications over federated data. Any application dealing with data, and especially distributed data from a variety of sources, needs some place to keep information about the data that it uses. The local store is a convenient place for these meta-data. Without that store, and the automatic cataloging of basic facts about the data, the application developer would be forced to manually track and ensure the persistence of such information. A second use for the local store is to materialize data. Data may need to be materialized temporarily, as part of query processing, for example, or for longer—from a short-term cache to a persistent copy, or even a long-term store of data unique to the federated applications.

Materialized views[43] are an advanced feature of SQL that exploits the local store. Originally conceived of as a way to amortize the cost of aggregating large volumes of data over multiple queries, a materialized view (called an automatic summary table, or AST, in DB2) creates a stored version of the query results. When new queries arrive that match the AST's definition, they are automatically "routed" to use the stored results rather than recomputing them.[44] If the view definition can reference foreign data, then a stored summary of data from federated sources can be created. When summary tables are considered during query planning, DB2 will compare the cost of fetching data directly from the local summary table to the cost of re-evaluating the remote query and pick the cheaper strategy. This is easy and natural in a database federation, but would take an enormous effort to replicate with other integration approaches, as it depends so heavily on multiple features of the database engine approach.

Last but not least, using the database engine to integrate data means that many if not all of the tools and applications that have been developed over the years for relational databases can be used without modification on distributed data. These include query builders such as Cognos[45] and Brio,[46] application development environments such as IBM Web-

Figure 4 A DB2 database federation that includes relational databases



Sphere* Studio,[47] IBM VisualAge*,[48] and Microsoft Visual C++**[49] or WebGain VisualCafé**,[50] report generators, visualization tools, and so on. Likewise, the customer may already have applications developed for a single database that can be reused easily in the federated environment by substituting names of remote tables for local ones, or modifying view definitions to make use of remote data.

In summary, many of the advantages offered by standard relational database systems are equally applicable to database federation. Thus, the database federation approach provides a richly supportive environment for the task of data integration. In the following section, we illustrate some of these benefits through some typical usage scenarios.

DB2 federation usage scenarios

In this section, we describe a set of scenarios that illustrate how database federation can be used to integrate local and remote data from a variety of sources, including relational data, Microsoft Excel spreadsheets, XML documents, Web services, and message queues. Furthermore, these scenarios demonstrate that by using the database as the integration engine, business applications are able to exploit the full power of SQL over federated data, including complex queries and views, automatic summary tables, OLAP (on-line analytical processing) functions, DB2 Spatial Extender, replication, and caching.

Federation of distributed data. A nationwide department store chain has stores in several regions of the country. Each of the stores relies on a relational database to maintain inventory records and customer transactions. However, because the department store has gone through several technology migrations, not all of the stores use the same database products. Both the San Francisco store and the New York store use Oracle databases to record business transactions, while the corporate headquarters has recently migrated to DB2.

Each store maintains a Transactions table, which contains an entry for each item scanned during a customer transaction. It is easy for individual stores to generate storewide sales reports using the information stored in this table. Figure 4 shows how the corporate office can use DB2 technology to generate a sales report across *all* stores. Because the San Francisco and New York offices both use Oracle, the corporate office can use the Oracle wrapper provided with DB2 to access both stores' databases. Likewise, the corporate office can access other stores' databases using the wrappers appropriate for those databases. Note that the schemas for the individual databases need not be the same, as long as queries can be for-

mulated to extract the same information from each store.

Each store's database is registered as a server in the corporate headquarters database, and the tables that the corporate office needs to access are registered as nicknames. For example, each store's Transactions table is registered as nickname. Once the nicknames are in place, a federated view that shows company-wide transactions can be defined as follows:

```
CREATE VIEW National_Transactions (store_id, tran_date,
                                   tran_id, item_id) AS
SELECT store_id, tran_date, tran_id, item_id
FROM sf.Transactions
UNION ALL
SELECT store_id, tran_date, tran_id, item_id
FROM ny.Transactions
```

Note that if more stores are added to the department chain, the corporate office does not need to modify its business application. Rather, the National_Transactions view definition can be updated to include the information for the new stores. Given this view, the corporate office can run a single query to generate a national sales report that shows the total of the number of items sold per month by all stores in the company:

```
SELECT MONTH(tran_date), item_id, COUNT(*)
FROM National_Transactions
WHERE YEAR(tran_date)=2001
GROUP BY MONTH(tran_date), item_id
```

In addition, the corporate office can create an automatic summary table over the federated view to cache the transaction information for previous years locally, since it is not likely to change. The following statement can be used to create this materialized view:

```
CREATE TABLE Past_Sales AS (
SELECT YEAR(tran_date) AS year, MONTH(tran_date) AS
                        month, item_id, COUNT(*) AS sales
  FROM National_Transactions
  WHERE YEAR(tran_date) <= 2001
  GROUP BY YEAR(tran_date), MONTH(tran_date), item_id)
DATA INITIALLY DEFERRED REFRESH DEFERRED
```

The Past_Sales summary table is locally stored, and as a result, indexes can be created over the table and statistics can be collected to improve query performance. In addition, a special register can be set to indicate that DB2 should automatically check to see whether a given query could be executed over the (local) summary table rather than over the (remote) nicknames. If so, in addition to plans that send remote requests to the regional stores to get the information and compute the result, DB2 will consider plans to extract the information from the Past_Sales summary table directly. This analysis is done transparently and does not require changes to the original query.

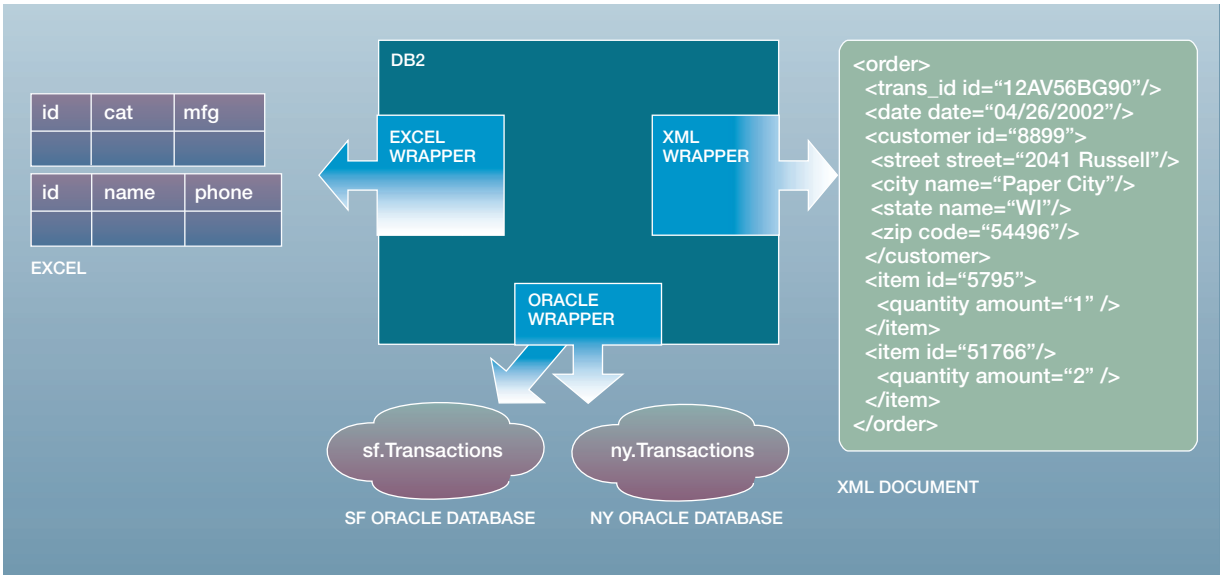
Federation of nonrelational structured data. Figure 5 shows how the corporate office of the department store chain can use a DB2 database federation to access nonrelational data sources as well. For example, the procurement office might like to know the manufacturer and the supplier of the best-selling television in 2001. Item and supplier information are stored in Excel spreadsheets and can be accessed from DB2 using the Excel wrapper. The items spreadsheet is mapped to an Items nickname, and the suppliers spreadsheet is mapped to a Suppliers nickname. Data contained in the spreadsheets can be retrieved using SQL as shown by the following query:

```
SELECT i.mfg, s.id
FROM Items i, Suppliers s
WHERE i.id = s.id AND i.id = (SELECT g.id
    FROM (SELECT g.id, COUNT(*), ROWNUMBER( )
      OVER (ORDER BY COUNT(*) DESC) AS rownum
      FROM National_Transactions g, Items it
      WHERE it.cat='television' AND g.id = it.id AND
                                    YEAR(tran_date)=2001
      GROUP BY g.id) AS tv_total_2001
    WHERE rownum = 1)
```

In the above query, the OLAP function ROWNUMBER is used to order the COUNT(*) in descending order in the nationwide total of sales for every model of television in the year 2001. The first row is then selected to find the item identifier (ID) of the most frequently sold television. This example shows that by exploiting DB2 database federation technology, the corporate office can use a single (complex) SQL statement to correlate information among the stores and the corporate office, although the data may be stored and represented differently at each location.

Federation of semi-structured data. The department store chain also provides on-line shopping for customers. Customer data are stored in a Customers table in the corporate office database, and orders are generated as XML documents by a Web application. These XML documents can also be accessed from the

Figure 5 A DB2 database that includes relational and nonrelational data sources



corporate database using the XML wrapper to be shipped with DB2 Life Sciences Data Connect.[31]

XML documents map to nicknames, and elements of the XML document map to columns of the nickname. A single XML document can be mapped to multiple nicknames. An option associated with the nickname allows the user to specify the location of the XML document in the query, and an XPATH option on the nickname column definition maps the column to its location in the XML document. For example, the order document contains both order and item information, and as a result, the document is mapped to an Orders nickname and an Order_Items nickname, and queries involving items from a particular order can be expressed as joins of these two nicknames.

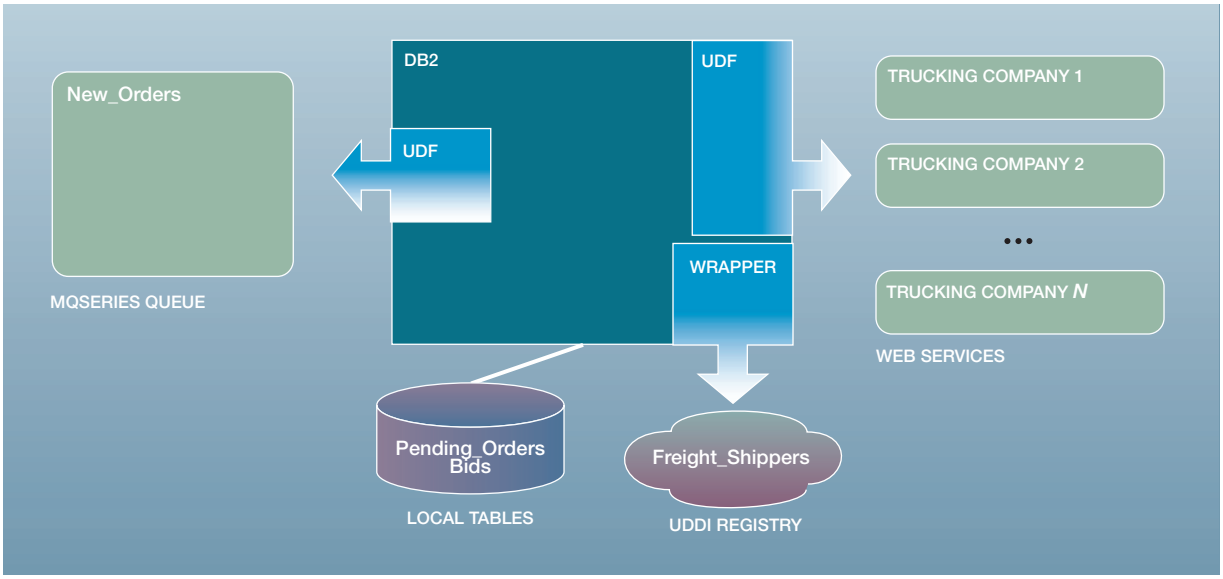
As part of order processing, the Web application must determine and dispatch the order to the distribution center closest to the customer. The corporate office can store the geo-coded location of each distribution center in a Distribution_Centers table, geocode the customer's location, and use the DB2 Spatial Extender function db2gse.st_distance( ) to identify the distribution center closest to the customer:

```
SELECT s.store_id, db2gse.st_distance(GEOCODE(o.street,
                o.city,o.state,o.zip), s.location, 'mile') AS distance
FROM Distribution_Centers s, Orders o
WHERE o.order = '/home/customers/order5795.xml' AND
                                o.transaction_id='12AV56BG90'
ORDER BY distance
```

Federation of Web services. A small furniture company supplies several nationwide retail stores with its products. The retail stores submit orders for furniture through a Web application, the furniture company fulfills the order and ships the furniture to the stores. Since freight shipment represents a significant portion of the production cost, the furniture company contracts with several trucking companies and puts each freight shipment up for bid. Figure 6 shows a system configuration using DB2 for the order processing system. New orders are placed on an MQ Series queue. A back-end order processing system removes orders from the queue and solicits the bids from various trucking companies for shipment.

The order processing system uses a Pending_Orders table to maintain a list of orders currently being processed. Each order has an auto-generated unique order number, as well as the XML document that contains the order information. The furniture company maintains a private UDDI registry for trucking companies that support a common Web services interface to request freight shipment bids. This registry is available to the order processing system via a wrapper, and a Freight_Shippers nickname supported

Figure 6 A DB2 database federation that includes Web services and other sources



by the wrapper supplies the names and URLs of trucking companies that support the bid Web service. A UDF called bid( ) takes the URL of a company and an XML description of an order, sends a Web services request to retrieve a bid for the order from the company specified by the URL, and returns the company's bid. A Bids table contains a list of bids obtained for a given order, including the order number, the name of the trucking company that supplied the bid, and the bid itself.

The furniture company can exploit DB2's federation capabilities to automate a significant portion of the order process. For example, orders can be removed from the MQ Series queue (via the db2mq.mqreceive( ) UDF), assigned a unique order number, and inserted into the Pending_Orders table with the following statement:

```
INSERT INTO Pending_Orders
VALUES(GENERATE_UNIQUE( ), db2mq.mqreceive( ))
```

Furthermore, a trigger defined on the Pending_Orders table can automatically kick off the bid process as new orders are inserted into the table:

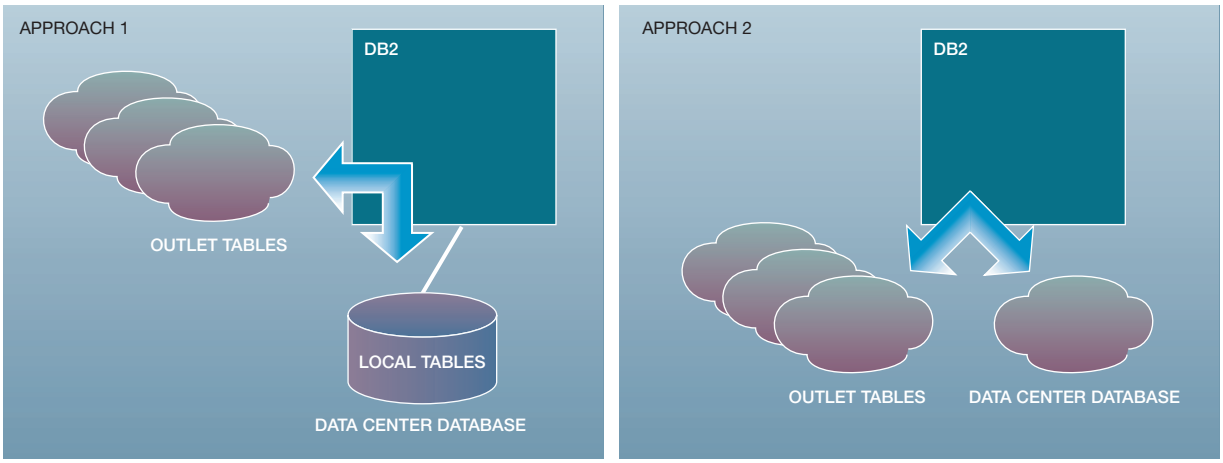
```
CREATE TRIGGER Get_Bids
AFTER INSERT ON Pending_Orders
REFERENCING NEW AS order
FOR EACH ROW MODE DB2SQL
INSERT INTO Bids
SELECT Order.ordernum, s.name, bid(s.url, order.orderxml)
FROM Freight_Shippers S
```

This SQL statement illustrates the power of a database federation solution for integrating data. A simple insert statement causes a sophisticated trigger to execute over federated data that are transparently combined via user defined functions (db2mq.mqreceive( ) and bid( )) and wrapper-based federation (the Freight_Shippers nickname).

Heterogeneous replication using database federation. Many businesses choose to keep multiple copies of their data for various uses, including data warehousing, fault tolerance, and fail-over scenarios. A major retailer with outlets all over the United States backs up data from its various locations to regional data centers. Due to independent purchasing decisions, the retail outlets use one relational database management system, while the data center might use another. The replication process is relatively straightforward, and involves extracting data from the outlets' databases, optionally reshaping the data and/or aggregating the data, and inserting the data into the data center database.

Figure 7 shows two approaches that use DB2 technology as the extract/transform vehicle to transfer

Figure 7 Heterogeneous replication using database federation



data from the outlets to the data center. Approach 1 in Figure 7 shows that if the data center uses a DB2 database, the data transfer can take place with simple statements of the form:

```
INSERT INTO <data center local_table>
            SELECT . . . FROM <outlet nickname>. . . .
```

Approach 2 in Figure 7 shows that even if the data center uses another relational database product, the same INSERT statement can be used to populate the database, with the only difference being that the federated database will insert into a nickname instead:

```
INSERT INTO <data center nickname>
            SELECT . . . FROM <outlet nickname>. . . .
```

Note that in either approach the SELECT statement can be arbitrarily complex and can be used to selectively retrieve, re-shape, and aggregate data according to the semantics of the application.

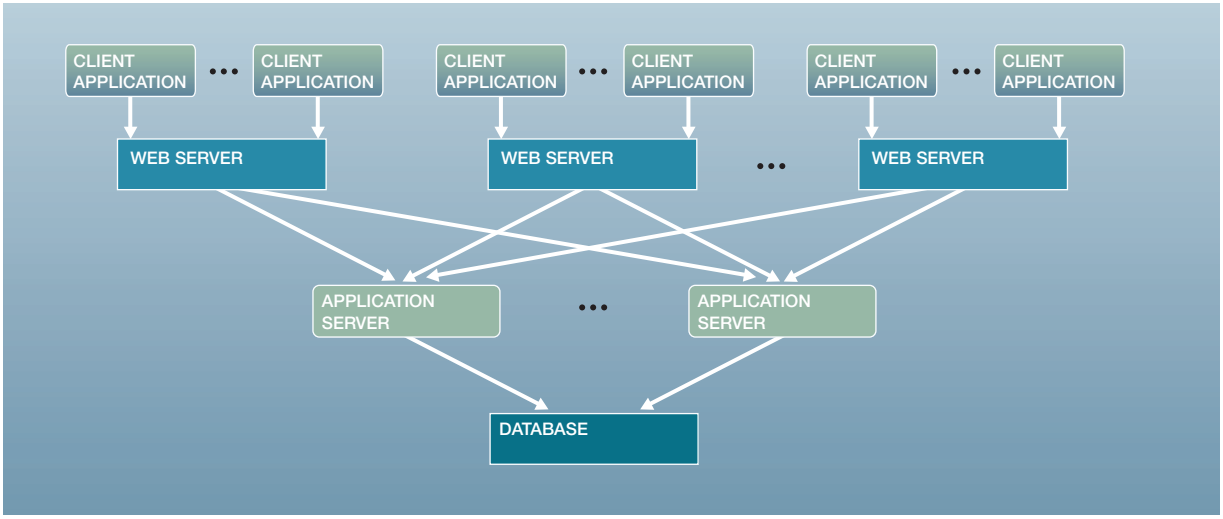
DB2 DataPropagator*[51] is an IBM product that uses DB2 federation technology to replicate data. DataPropagator automates the copying of data between remote systems, providing automated change propagation, flexible scheduling, and other customization features. DataPropagator exploits the remote insert/update/delete capability illustrated above to transparently apply changes to all relational sources including non-DB2 sources.

Dynamic data caching. As shown in Figure 8, a typical e-commerce Web application consists of a three-tiered architecture: the Web server tier, the application server tier, and the data tier. User requests are routed to one of multiple Web servers, which forward the user requests to one of the application servers for processing. The application servers in turn retrieve product data from, and insert order information into, a single back-end database.

It is easy to see from the figure that as traffic increases, the back-end database can quickly become the bottleneck. DBCACHE[52,53] is a research prototype built with federation technology that provides scalability of the data tier. Each application server node may include a front-end database server, the "cache" of DBCACHE. DBCACHE allows database administrators to replicate portions of the back-end database across multiple front-end databases, allowing client requests to be routed to the front-end databases. This topology is often less expensive than a single large parallel system, and also provides a layer of fault tolerance; a single database crash does not cause the entire database to become inoperative.

Figure 9 shows an e-commerce application using DBCACHE technology. Application tables are divided into two categories: cached and noncached. Cached tables are mostly read-only, and accessed by multiple users. For these kinds of tables, maintaining strict consistency is not necessary, and reading stale data is acceptable. From an on-line shopper's perspective, product catalog data are read-only (query Q1 in Figure 9, for example). Updates occur infrequently, usually through a scheduled deployment scenario (never by an on-line shopper), and it is not a

Figure 8 Three-tier architecture for an e-commerce application



critical loss if a shopper sees product information that is a few seconds out of date. Furthermore, since all shoppers browse the product catalog before making a purchase, the product catalog is heavily accessed.

Noncached tables are typically read/write. For these tables, maintaining strict consistency is important, and accessing stale data is intolerable. Order information is an example of data stored in a noncached table. From the perspective of the shopper, order information is mostly write-only (query Q2 in Figure 9, for example). It is captured as part of on-line order processing and read only by back-end order fulfillment applications and by the shopper to check order status. Because orders result in transfer of money, it is critical that they present an accurate view of the data.

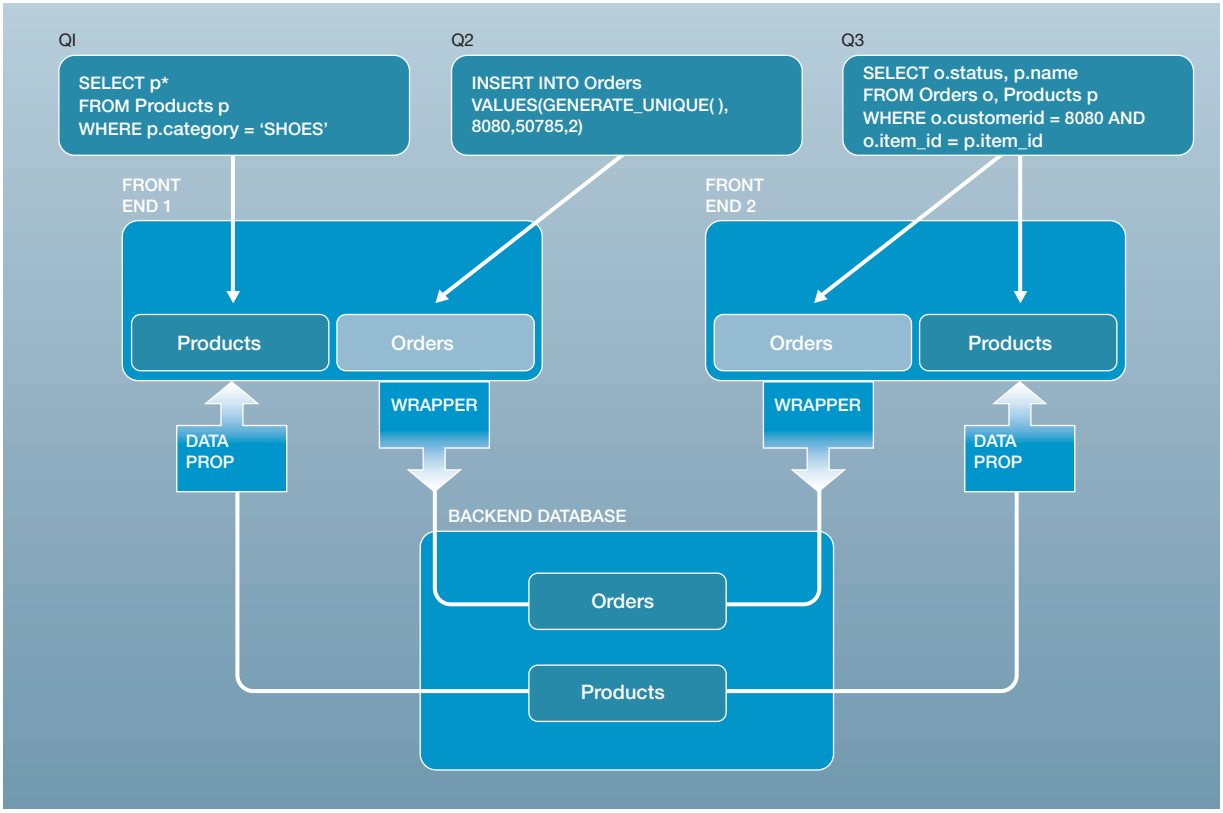
As shown in Figure 9, the Products back-end table is automatically replicated to a cached table across multiple front ends using DataPropagator (Data Prop in Figure 9), according to a schedule defined by the DBA. Reads to the Products table are transparently routed to one of the front-end databases, whereas writes (if any) are routed directly to the back-end database. The Orders back-end table is represented as a nickname in the front-end databases, and both reads and writes are passed to the back-end database using DB2 federated technology.

The client application sees a single view of the data. Queries for product information (query Q1 in Figure 9) are dynamically routed to one of the cached tables on the front-end database, providing a level of load balancing and fault tolerance for these heavily accessed data. Insert statements to the Orders table (query Q2 in Figure 9) are routed directly to the back-end table. Statements can span both cached and noncached tables. For example, a query to check the status of an order (query Q3 in Figure 9) can join information from the Orders nickname with the Products table.

Conclusions

In this paper, we have shown that database federation is a powerful tool for integrating data. Database federation employs a database engine to create a virtual database from several, possibly many, heterogeneous and distributed data stores. We identified three styles of database federation. In all of them, the database engine is the key driver, but the method by which data or functions are included in the federation differs. We presented guidelines on when each style of federation should be used: user-defined functions are most appropriate for fairly simple integration tasks, whereas the wrapper architecture supports a much broader and more complex set of tasks. We discussed why database technology is so crucial to data integration, and how many of the

Figure 9 E-commerce application using DBCASHE



features of relational databases can be applied in a distributed environment to ease the development of new applications that span multiple data sources. Finally, we demonstrated through a number of use cases how various database features—including views, ASTs, OLAP functions, joins, unions, and aggregations—can be used in conjunction with multiple federation styles to integrate data from sources as diverse as MQ queues, XML documents, Excel spreadsheets, Web services, and relational database management systems. We showed how federation could be used as the basis for report gathering, for warehouse loading and replication, and even for caching. The diversity of applications for this technology is reflective of the many ways in which data must be integrated, and the applicability of database federation to the broad range of challenges demonstrates its importance for data integration.

Of course, database federation is not a panacea. There may be some integration scenarios for which it is overkill, or for which another of the many approaches to data integration might be easier. However, we strongly believe that database federation must be a fundamental part of any integration solution. Our future efforts will be directed toward improving the ease of use for this technology. Tools are needed to develop robust and efficient wrappers quickly and with minimal effort. We continue to enhance the performance of the system, improving the optimization of queries and the set of execution strategies available to the optimizer. Further challenges in this modern age include being able to handle asynchrony everywhere, better integration of XML capabilities, and including native support for XML query. As customers exercise the technology, we are finding that richer support for semantic meta-data would be helpful, as well as more automation in the administration of the federated system. With enhancements such as these, we expect that database federation will come to be widely perceived as the cornerstone of data integration.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of the Object Management Group, Sun Microsystems, Inc., Microsoft Corporation, or WebGain, Inc.

Cited references

1. Documentum, Inc., Content Management: Documentum Products, http://www.documentum.com/content-management_Products.html.
2. IBM Corporation, Content Manager, http://www.ibm.com/software/data/cm/.
3. M. A. Roth, D. C. Wolfson, J. C. Kleewein, and C. J. Nelin, "Information Integration: A New Generation of Information Technology," *IBM Systems Journal* **41**, No. 4, 563–577 (2002, this issue).
4. IBM Corporation, WebSphere Application Server, http://www.3.ibm.com/software/webservers/appserv/enterprise.html.
5. LION Bioscience AG, LION DiscoveryCenter, http://www.lionbioscience.com/solutions/discoverycenter/.
6. middleAware.com, Component Focus, http://www.middleware.net/components/index.html.
7. F. Leymann and D. Roller, "Using Flows in Information Integration," *IBM Systems Journal* **41**, No. 4, 732–742 (2002, this issue).
8. M. Roscheisen, M. Baldonado, C. Chang, L. Gravano, S. Ketchpel, and A. Paepcke, "The Stanford InfoBus and Its Service Layers: Augmenting the Internet with Higher-Level Information Management Protocols," *Digital Libraries in Computer Science: The MeDoc Approach*, Lecture Notes in Computer Science, No. 1392, Springer, New York (1998), pp. 213–230.
9. IBM Corporation, Enterprise Information Portal, http://www-3.ibm.com/software/data/eip.
10. H. Pirahesh, J. Hellerstein, and W. Hasan, "Extensible Rule-Based Query Rewrite Optimization in Starburst," *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, San Diego, CA, June 2–5, 1992, ACM, New York (1992), pp. 39–48.
11. M. Tork Roth, F. Ozcan, and L. Haas, "Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System," *Proceedings of the Conference on Very Large Data Bases* (VLDB), Edinburgh, Scotland, September 1999, Morgan Kaufmann Publishers, San Mateo, CA (1999), pp. 599–610.
12. P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price, "Access Path Selection in a Relational Database Management System," *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, Boston, MA, 1979, ACM, New York (1979), pp. 23–34.
13. H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, "Integrating and Accessing Heterogeneous Information Sources in TSIMMIS," *Proceedings of the AAAI Symposium on Information Gathering*, Stanford, CA, March 1995, AAAI Press (1995), pp. 61–64.
14. S. Adali, K. Candan, Y. Papakonstantinou, and V. S. Subrahmanian, "Query Caching and Optimization in Distributed Mediator Systems," *Proceedings of the ACM SIGMOD Conference on Management of Data*, Montreal, Canada, June 1996, ACM, New York (1996), pp. 137–148.
15. A. Tomasic, L. Raschid, and P. Valduriez, "Scaling Heterogeneous Databases and the Design of DISCO," *Proceedings of the 16th International Conference on Distributed Computer Systems*, May 1996, Hong Kong, IEEE, New York (1996), pp. 449–457.
16. M.-C. Shan, "Pegasus Architecture and Design Principles," *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington, D.C., May 1993, ACM, New York (1993), pp. 422–425.
17. M. Tork Roth, P. Schwarz, and L. Haas, "An Architecture for Transparent Access to Diverse Data Sources," *Component Database Systems*, K. R. Dittrich, A. Geppert, Editors, Morgan-Kaufmann Publishers, San Mateo, CA (2001), pp. 175–206.
18. IBM Corporation, DB2 Product Family, http://www-3.ibm.com/software/data/db2/.
19. M. Tork Roth and P. Schwarz, "Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources," *Proceedings of the Conference on Very Large Data Bases* (VLDB), Athens, Greece, August 1997, Morgan Kaufmann Publishers, San Mateo, CA (1997), pp. 266–275.
20. L. Haas, D. Kossmann, E. Wimmers, and J. Yang, "Optimizing Queries Across Diverse Data Sources," *Proceedings of the Conference on Very Large Data Bases* (VLDB), Athens, Greece, August 1997, Morgan Kaufmann Publishers, San Mateo, CA (1997), pp. 276–285.
21. iWay Software, iWay Adapters and Connectors, http://www.iwaysoftware.com/products/e2e_integration_products.html.
22. Microsoft Corporation, Microsoft ODBC, http://www.microsoft.com/data/odbc/.
23. IBM Corporation, DataJoiner, http://www.software.ibm.com/data/datajoiner/.
24. Microsoft Corporation, Access 2002 Product Guide, http://www.microsoft.com/office/access/default.asp.
25. IBM Corporation, DB2 Relational Connect, http://www-3.ibm.com/software/data/db2/relconnect/.
26. B. Reinwald, H. Pirahesh, G. Krishnamoorthy, G. Lapis, B. Tran, and S. Vora, "Heterogeneous Query Processing Through SQL Table Functions," *Proceedings of the 15th International Conference on Data Engineering*, March 1999, Sidney, Australia, IEEE, New York (1999), pp. 366–373.
27. L. M. Haas, P. M. Schwarz, P. Kodali, E. Kotlar, J. E. Rice, and W. C. Swope, "DiscoveryLink: A System for Integrated Access to Life Sciences Data Sources," *IBM Systems Journal* **40**, No. 2, 489–511 (2001), http://www.research.ibm.com/journal/sj/402/haas.html.
28. ISO/IEC 9075-2:2000, Information technology—Database languages—SQL—Part 2: Foundation (SQL/Foundation), International Organization for Standardization, Geneva, Switzerland (2000).
29. ISO/IEC 9075-9:2000, Information technology—Database languages—SQL—Part 9: Management of External Data (SQL/MED), International Organization for Standardization, Geneva, Switzerland (2000).
30. L. Haas, J. Freytag, G. Lohman, and H. Pirahesh, "Extensible Query Processing in Starburst," *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, Portland, OR, May 31–June 2, 1989, ACM, New York (1989), pp. 377–388.
31. IBM Corporation, DB2 Life Sciences Data Connect, http://www-3.ibm.com/software/data/db2/lifesciencesdataconnect/.
32. IBM Corporation, IBM MQSeries Integrator V2.0: The Next Generation Message Broker, http://www-3.ibm.com/software/ts/mqseries/library/whitepapers/mqintegrator/msgbrokers.html.
33. V. Josifovski, P. Schwarz, L. Haas, and E. Lin, "Garlic: A New Flavor of Federated Query Processing for DB2," *Pro-*

ceedings of the ACM SIGMOD Conference on Management of Data, Madison, WI, June 2002, ACM, New York (2002).
34. IBM Corporation, Lotus Extended Search, http://www.lotus.com/products/des.nsf/wdocs/home.
35. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM* **13**, No. 6, 377–387 (June 1970).
36. E. F. Codd, *The Relational Model for Database Management*, Version 2, Addison-Wesley Publishing Co., Reading, MA (1990).
37. C. J. Date and H. Darwen, *A Guide to SQL Standard*, *4th Edition*, Addison-Wesley Publishing Co., Reading, MA (1997).
38. D. Chamberlin, *A Complete Guide to DB2 Universal Database*, Morgan Kaufmann Publishers, San Mateo, CA (1998).
39. J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers, San Mateo, CA (1993).
40. I. Traiger, J. Gray, C. Galtieli, and B. Lindsay, *Transactions and Consistency in Distributed Database Systems*, Research Report RJ2555, IBM Almaden Research Center, San Jose, CA 95120 (1979).
41. M. Stonebraker, "Implementation of Integrity Constraints and Views by Query Modification," *Proceedings of the 1975 ACM SIGMOD Conference on Management of Data*, ACM, New York (1975), pp. 65–78.
42. *Rules in Database Systems*, T. K. Sellis, Editor, *Proceedings of the Second International Workshop* (RIDS '95), Glyfada, Athens, Greece, September 25–27, 1995, Lecture Notes in Computer Science, No. 985, Springer, New York (1995).
43. D. Srivastava, S. Dar, H. V. Jagadish, and A. Levy, "Answering Queries with Aggregation Using Views," *Proceedings of the 22nd International Conference on Very Large Data Bases* (VLDB '96), Morgan Kaufmann Publishers, San Mateo, CA (1996), pp. 318–329.
44. M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, and M. Urata, "Answering Complex SQL Queries Using Automatic Summary Tables," *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ACM, New York (2000), pp. 105–116.
45. Cognos Incorporated, Business Intelligence Products, http://www.cognos.com/products/index.html.
46. Brio Software, Inc., Business Intelligence solution for data query and analysis, http://www.brio.com/products/overview.html.
47. IBM Corporation, WebSphere Studio Application Developer, http://www-3.ibm.com/software/ad/studioappdev/.
48. IBM Corporation, VisualAge C++, http://www-3.ibm.com/software/ad/vacpp/.
49. Microsoft Corporation, Visual C++ .NET, http://msdn.microsoft.com/visualc/.
50. WebGain, Inc., Visual Café, http://www.webgain.com/products/visual_cafe/.
51. IBM Corporation, DB2 Universal Database, DB2 Data Propagator, http://www-3.ibm.com/software/data/dpropr/.
52. C. Mohan, "Caching Technologies for Web Applications," *Proceedings of the 27th International Conference on Very Large Data Bases*, September 11–14, 2001, Roma, Italy, Morgan Kaufmann Publishers, San Mateo, CA (2001).
53. Q. Luo, S. Krishnamurthy, C. Mohan, H. Pirahesh, H. Woo, B. Lindsay, and J. Naughton, "Middle-Tier Database Caching for e-Business," *Proceedings of the ACM SIGMOD Conference on Management of Data*, Madison, WI, June 2002, ACM, New York (2002).

Accepted for publication July 22, 2002.

Laura M. Haas *IBM Software Group, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, California 95141 (electronic mail: lmhaas@us.ibm.com).* Dr. Haas is a Distinguished Engineer and senior manager in the IBM Software Group, where she is responsible for the DB2 Query Compiler development, including key technologies for DiscoveryLink™ and Xperanto. Dr. Haas, who joined the IBM Almaden Research Center in 1981, has made significant contributions to database research and has led several research projects, including R*, Starburst, and Garlic. She has received an IBM Outstanding Technical Achievement Award for her work on R* and DiscoveryLink, an IBM Outstanding Contribution Award for Starburst, a YWCA Tribute to Women in Industry (TWIN) Award, and an ACM SIGMOD Outstanding Contribution Award.

Eileen T. Lin *IBM Software Group, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, California 95141 (electronic mail: etlin@us.ibm.com).* Dr. Lin is a Senior Technical Staff Member and lead architect for DB2 database federation. She joined IBM in 1990 after receiving her Ph.D. degree from the Georgia Institute of Technology in Atlanta. She was the lead architect for DataJoiner query processing and led the team that merged DataJoiner and Garlic technology into DB2 for UNIX® and Windows®. She is currently a lead architect responsible for the delivery of database federation technology in DB2 and Xperanto.

Mary A. Roth *IBM Software Group, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, California 95141 (electronic mail: torkroth@us.ibm.com).* Ms. Roth is a senior engineer and manager in the Database Technology Institute for e-Business at IBM's Silicon Valley Lab. She has over 12 years of experience in database research and development. As a researcher and member of the Garlic project at IBM's Almaden Research Center, she contributed key advances in heterogeneous data integration techniques and federated query optimization and led efforts to transfer Garlic support to DB2. Ms. Roth is leading a team of developers to deliver a key set of components for Xperanto, IBM's information integration initiative for distributed data access and integration.