Database Integration and Federation

Erwin M. Bakker

DBDM 26-09 2005

Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases.

A.P. Seth, J.A. Larson


Federated Database Systems

- The term *Federated Database Systems* was first used by Hammer and McLeod in 1979 and by Heimbigner and McLeod in 1985
- Used for different architectures

Federated Database Systems

- Database System (DBS)
  - Database Management System (DBMS)
  - One or more Databases
- Federated Database System (FDBS)
  - A collection of cooperating but autonomous component database systems (integrated to various degrees)
  - The federated database management system (FDBS) provides a controlled and coordinated manipulation of the component DBSs
- Key characteristic: cooperation among independent systems
Federated Database Systems

- Component Database System (DBS)
  - Can be centralized or distributed
  - Can participate in one or more federations
  - Can be a FDBMS itself
  - Can continue its local operations while participating in federation
  - Managed by users of the federation or by the administrations of the FDBS and DBSs

Federated Database Systems

- Application of the federation concept for managing existing heterogeneous and autonomous DBSs
- Various alternative architectures and components
- Development and operation of such systems

Characteristics of Database Systems

- Characterization based on
  - Distribution
  - Heterogeneity
    - Data models
    - Transaction management
  - Autonomy
- Based on networking
  - Single, many DBSs in LAN, WAN
- Update related
  - No updates, nonatomic, atomic
FDBS Data Distribution

- Much of the data distribution is due to the existence of multiple participating DBSs
- Stored on single or multiple computers
- Co-located or geographically distributed
- Vertical and horizontal database partitions
  - Availability
  - Reliability
  - Improved access times
  - (In distributed databases the distribution of data is induced for these benefits)
- Etc.

FDBS Heterogeneity

- Many Types of Heterogeneity
  - DBSs
    - Due to differences in DBMSs
    - Due to semantics of the data
  - Operating Systems
  - Hardware/Systems
  - Communication

---

**Figure 2.** Types of heterogeneities.
**FDBS Semantic Heterogeneity**

- Meaning, interpretation, intended use of the same or related data (Difficult)

**Example 1:**
Attribute `MEAL_COST` of relation `RESTAURANT` in DB1
- Average cost of a meal per person in a restaurant without service charge and tax

Attribute `MEAL_COST` of relation `BOARDING` in DB2
- Average cost of a meal per person inclusive charge and tax

How do you compare?
`DB1.RESTAURANTS.MEAL_COST` with `DB2.BOARDING.MEAL_COST`

**Example 2:**
Attribute `GRADE` of relation `COURSE` in DB1
- Is grade of a student from the set of values \([A,B,C,D,F]\) \((C \sim [61,75])\)

Attribute `SCORE` of relation `CLASS` in DB2
- A normalized score on the scale from 0 to 10 rounded to 0.5
  \((7.5 \sim [73,77])\)

How do values correspond?
`DB1.COURSE.GRADE` with `DB2.CLASS.GRADE`

---

**FDBS Autonomy**

- DBSs in a federation are in principle autonomous
- Different aspects (Popescu-Zeletin 1988)
  - Design autonomy
  - Communication autonomy
    - Component DBMS can decide with who and when to communicate
    - When and how to respond to requests
  - Execution autonomy

- Design autonomy refers to the ability of a component DBS to choose its own design with respect to any matter, including:
  - The data being managed (i.e., the Universe of Discourse),
  - The representation (data model, query language) and the naming of the data elements,
  - The conceptualization or semantic interpretation of the data (which greatly contributes to the problem of semantic heterogeneity),
  - Constraints (e.g., semantic integrity constraints and the serializability criteria) used to manage the data,
  - The functionality of the system (i.e., the operations supported by system),
  - The association and sharing with other systems (see association autonomy below), and
  - The implementation (e.g., record and file structures, concurrency control algorithms).
### FDBS Execution Autonomy
- DBMS decides on priority and inference of local operations vs external operations.
- When there is execution autonomy a FDMBS cannot enforce an order of execution of the command on a component DBMS.
- DBMS can abort commands that do not meet local constraints.
- DBMS does not need to inform FDBMS about order.
- Operationally it means treating external operations the same way as local operations.

### FDBS Association Autonomy
- DBS decides how much to share:
  - Functionality
  - Resources (data)
  - Services
- Ability to associate or disassociate with the FDBMS
- Control over local operation.

These are conflicting requirements with the need to share => relaxing autonomy:
1. Entry or departure of DBS in FDBS must be based on an agreement.
2. Informing the FDBS about the order of execution.

### Federated Databases Systems
- Nonfederated Database Systems (e.g., UNBASE, Brezinski et al. 1984)
- Federated Database Systems
  - Loosely Coupled (e.g., MRDSM, Lwin 1985)
  - Tightly Coupled
    - Single Federation (e.g., DDTS, Dwyer and Larson 1987)
    - Multiple Federations (e.g., Memad, Templeton et al. 1987a)

*Figure 3. Taxonomy of multidatabase systems.*
Reference Architecture: Components

- **Data**: Data are the basic facts and information managed by a DBS.
- **Database**: A database is a repository of data structured according to a data model.
- **Commands**: Commands are requests for specific actions that are either entered by a user or generated by a processor.
- **Processors**: Processors are software modules that manipulate commands and data.
- **Schemas**: Schemas are descriptions of data managed by one or more DBMSs. A schema consists of schema objects and their interrelationships. Schema objects are typically class definitions (or data structure descriptions) (e.g., table definitions in a relational model), and entity types and relationship types in the entity-relationship model.
- **Mappings**: Mappings are functions that correlate the schema objects in one schema to the schema objects in another schema.

Processor Types

Processors perform different functions on the data manipulation commands and accessed data:
- Transforming processor (for data model transparency)
- Filtering processor
- Constructing processor
- Accessing processor

Transforming processor

- for data model transparency
  - Translates commands from one language to another language
  - Transforms data from one format to another format to enable Filtering processor
  - Schema translation
  - Needs mappings between objects of the schema's

Most centralized, distributed and federated database systems can be expressed using these basic components.
Filtering processor

- for data model transparency
  - Constrain the commands and associated data that can be passed to another processor.
  - Syntactic constraint checkers (commands syntactically correct)
  - Semantic integrity constraint checker
    - Check that the commands will not violate semantic integrity constraints (e.g. negative age, postal code, etc.)
    - Modifies commands so that they will be semantically correct.
    - Verifies produced data on semantically constraints
  - Access control

Constructing processor

- Partition and/or replicate an operation submitted by a single processor into operations that are accepted by two or more processors
- Also merge resulting data from several processors into a single data set for consumption by another processor
- Transparency on
  - Location
  - Distribution
  - replication
Accessing Processors

Centralized DBMS three levels (ANSI/X3/SPARC):
- **Conceptual schema**
  - Conceptual or logical data structure description and their relationships
- **Internal schema**
  - Physical characteristics of the logical data structures (placement of records, indexes, physical representations of relations, etc.)
  - Changes can be done without implications on the conceptual schema
- **External schema**
  - Subset/part of the database that may be accessed by a user or class of users is described by an external schema

Tasks that can be handled by constructing processors include the following:
- **Schema integration**: Integrating multiple schemes into a single schema
- **Negotiation**: Determining what protocol should be used among the owners of various schemas to be integrated in determining the contents of an integrated schema
- **Query (command) decomposition and optimization**: Decomposing and optimizing a query (command) expressed on an integrated schema
- **Global transaction management**: Performing the concurrency and atomicity control
A centralized DBMS in terms of our architecture.

A Five Level Schema Architecture for Federated Databases

- **Local Schema**
  - Is the conceptual schema of a component DBS

- **Component Schema**
  - Derived by translating local schemas into a data model called the canonical or common data model of the FDBS
  - They describe divergent or local schema by using a single representation
  - They add semantics that is missing in local schema
  - They facilitate
    - negotiation and integration for tightly coupled FDBS; and
    - negotiation and specification of views and multiple database queries in loosely coupled FDBS)

- **Export Schema**
  - A subset of a component schema that is available for a FDBS
  - Includes access control info (a filtering processor can be used for this)
  - To facilitate control and management of association autonomy

- **Federated Schema**
  - Integration of multiple export schemas
  - Info on data distribution is produced when integrating export schemas (sometimes kept in a separate distribution or allocation schema)
  - Multiple federated schemas; one for each class of federation users, e.g., managers, employees, applications

- **External Schema**
  - Define a schema for a user and/or application, because of
    - Customization
      - Federated schema can be large and difficult to change. External schema can be defined on a subset, using a different data model to meet user needs.
    - Additional Integrity
      - Can be specified, for example for data mining applications
    - Access control
      - Export schemas provide access control to the data managed by the component databases
      - External schemas provide access control to the data managed by the federated database
The five level schema and the processor concept allows us to give the following system architecture for a FDBS.

Redundancies between external and federated schemas
- A federated schema could in principle be produced for every user, making the external schema redundant

Redundancies between external schema and schema of a component DBS and an export schema
- If access control is correctly handled by the DBS and translation of a local schema into a component schema is not necessary

When component DBS use the CDM (Common Data Model) used by the FDBS, it is not necessary to define a component schema.
**Additional Basic Components in Federated Database System**

- **Auxiliary schema**
  - data needed by the federation but not available in the existing databases
  - Info to resolve incompatibilities (e.g. unit translations, format conversions, etc.)
  - Statistical info for query optimization and scheduling
- **Filtering Processor** in addition to a constructing processor to enforce constraints that span multiple component databases

---

**Auxiliary Schema in Federated Database System**

- Using an auxiliary schema to store translation information needed by a constructing processor.

---

**Integrity Constraints in External/Federated Schemas**

- Filtering Processor
- Constructing Processor

- Export Schema 1 expressed using CDM
- Export Schema 2 expressed using CDM
- Export Schema 2 expressed using CDM

**Figure 14.** Using a filtering processor to enforce constraints across export schemas.
Federated Databases Systems

- Some Examples

Example Databases:

Distributed Database Testbed System
(Devor et al. 1982)

- Single external schema
- Single federated schema
- Users formulate requests
  Against the conceptual Schema using GORDAS

A query language.

Multibase

Figure 18 illustrates the architecture of Multibase, again using the terminology of the reference architecture and the terminology used by Landers and Rosenberg (1982). The federated schema is expressed in a functional data model called DAPLEX. The Transforming modifies global queries by inserting references to Local and Auxiliary schemas. The modified global query is then processed by a series of processors that generates sequences of DAPLEX single-site queries. These processors include:

- A *Global Query* Optimizer that produces a global plan;
- A *Decomposer* that decomposes the global plan into single-site DAPLEX queries;
- A *Filter* that reduces the decomposed queries by removing operations from them that are not supported by the corresponding component DBMSs, and
- A *Monitor* that controls the distributed execution of the subqueries.

The Optimizer, Decomposer, and Filter may be cyclically invoked for nested global queries. Two types of transformations are performed on each DAPLEX single-site query:

- A *Local Optimizer* determines the optimal query-processing strategy for the single-site DAPLEX queries.
MERMAID (Templeton et al. 1987)

- Queries in SQL or ARIEL are Translated to a Data Intermediate Language (DIL)
- The component databases can be different relational databases.
Federated Databases

Introduction from IBM Federated Database Technology
By Laura Haas, and Eileen Lin

Federated Databases

IBM’s federated database systems are built on technology from an earlier product, DB2 DataJoiner [3], and enhanced with features for extensibility and performance from the Garlic research project [2].

- DB2 DataJoiner introduced the concept of a virtual database, created by federating together multiple heterogeneous relational data sources.
  - Users of DB2 DataJoiner can pose arbitrary queries over data stored anywhere in the federated system.
  - This without worrying about the data's location, the SQL dialect of the actual data stores, or the capabilities of those stores.
  - Users had the full capabilities of DB2 against any data in the federation.
- The Garlic project extends this idea to federated database systems that effectively exploits the query capabilities of diverse, possibly non-relational data sources.
- In both of these systems, and in current DB2, a middleware query processor develops optimized execution plans and compensates for any functionality that the data sources may lack.

Federated Databases

Characteristics

- Transparency
- Heterogeneity
- Functionality
- Autonomy of underlying federated sources,
- Extensibility & Openness
- Optimized performance
Federated Databases

Transparency

The system masks from the user the differences, idiosyncrasies, and implementations of the underlying data sources. Ideally, it makes the set of federated sources look to the user like a single system.

- The user should not need to be aware of
  - where the data is stored (location transparency)
  - what language or programming interface is supported by the data source (invocation transparency)
  - if SQL is used, what dialect of SQL the source supports (dialect transparency)
  - how the data is physically stored, or whether it is partitioned and/or replicated (physical data independence, fragmentation and replication transparency)
  - or what networking protocols are used (network transparency)

The user should see a single uniform interface, complete with a single set of error codes (error code transparency).

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.

Federated Databases

Heterogeneity

Data sources may run on different hardware,
- Usage of different network protocols
- Different software to manage the data stores.
- Different query languages, capabilities, and data models.
- Different error handling
- Different transaction semantics (2-phase commitment, etc.)

Data sources can be for example:
- two Oracle instances, one running Oracle 8i, and the other Oracle 9i, with the same or different schemas.
- high-powered relational database
- simple, structured flat file
- a web site that takes queries in the form of URLs and spits back semi-structured XML according to some DTD
- a Web service,
- an application that responds to a particular set of function calls.

In principle a federated database should be able to accommodate all of these differences, encompassing systems such as these in a seamless, transparent federation.

Federated Databases

Functionality

- standard-compliant DB2 SQL (or other SQL) capability for all data in the federation (Function compensation)
- the existing functionality of the underlying data sources

Function compensation means that if a data source cannot do a particular DB2 SQL query function, the federated database retrieves the necessary data and applies the function itself. For example:
- a file system typically cannot do arbitrary sorts. The federated database will retrieve the relevant data, and do the sort itself.

On the other hand specialized functionality that the federated database lacks, allows the user to identify functions of interest from the federated sources, and then to use them in queries. For example:
- document management systems often have relevance scoring functions
- In the financial industry, time-series data is important, and systems exist that can compare, plot, analyze, and subset time-series data in specialized ways.
- In the pharmaceutical industry, new drugs are based on existing compounds with particular properties. Special-purpose systems can compare chemical structures, or simulate the binding of two molecules.

Federated Databases

Extensible & Open Federation

- It should be easy to add new functionality and data sources.
- The IBM federated database engine accesses sources via a wrapper:
  - Accessing a new type of data source is done by acquiring or creating a new wrapper for that source.
  - Once a wrapper exists, data definition (DDL) statements allow sources to be dynamically added to the federation without stopping ongoing queries or transactions.
  - Any data source can be wrapped. IBM supports the ANSI SQL/MED standard [1](MED stands for Management of External Data). This standard documents the protocols used by a federated server to communicate with external data sources. Any wrapper written to the SQL/MED interface can be used with IBM's federated database.
Federated Databases
Autonomous Data Sources

The existing applications and users of the data sources should not be affected when it becomes part of a federation, i.e., the complete local operation of an existing data source should not be disturbed.

- Existing applications will run unchanged, data is neither moved nor modified, interfaces remain the same.
- The way the data source processes requests for data is not affected by the execution of global queries against the federated system, though those global queries may touch many different data sources.
- No impact on the consistency of the local system when a data source enters or leaves a federation.
- Except for federated two phase commit processing for sources that participate. Data sources involved in the same unit of work will need to participate in commit processing and can be requested to roll back the associated changes if necessary.

Remark: The IBM wrapper architecture does not require any software to be installed on the machine that hosts the data source. It communicates with the data source via a client server architecture, using the source's normal client. The IBM's federated data source looks like just another application to the source.

Federated Databases
Optimization

Relational queries have the following properties:
- non-procedural
- several different implementations of each relational operator
- many possible orderings of operators to choose from in executing a query.

The optimizer must decide:
- operations in a query done by the federated server or by the data source?
- determine the order of the operations, and what implementations to use to do local portions of the query.
- Is the data source is a file, or is the source is a relational database system capable of applying predicates and doing joins,
- When is it valid to reduce the amount of data that needs to be brought back to the federated engine,
- Analyze the details of the individual query

The capabilities and the costs of the wrappers are modeled for the different sources involved in a query to evaluate the possibilities.

Federated Databases
Architecture

A federated system is created by installing the federated engine and then configuring it to talk to the data sources.

Steps to add a new data source to a federated system:
1. A wrapper for the source must be installed,
2. The federated database must be told where to find this wrapper. This is done by means of the CREATE WRAPPER statement.

If multiple sources of the same type exist, only one wrapper is needed. For example:
- For 5 Oracle database instances, possibly on different machines, only one Oracle wrapper and only one CREATE WRAPPER statement is needed.
- However, each separate source must also be identified to the system. This is done by five CREATE SERVER statements.

Federated Databases
Configuration

Steps to add a new data source to a federated system:
1. A wrapper for the source must be installed,
2. The federated database must be told where to find this wrapper. This is done by means of the CREATE WRAPPER statement.

If multiple sources of the same type exist, only one wrapper is needed. For example:
- For 5 Oracle database instances, possibly on different machines, only one Oracle wrapper and only one CREATE WRAPPER statement is needed.
- However, each separate source must also be identified to the system. This is done by five CREATE SERVER statements.
The federated database can be informed about a web site data access wrapper via the following statement:

```
CREATE WRAPPER web_wrapper LIBRARY "/u/haas/wrappers/libweb.a"
```

This statement basically tells the federated database where to find the code for the web_wrapper.

- Next, the federated database can be told about the actual web site to be used, by identifying it as a server associated with the web_wrapper.

```
CREATE SERVER weather_server WRAPPER web_wrapper OPTIONS (URL 'http://www.weatherforecast.com')
```

The `OPTIONS` clause allows the basic CREATE SERVER statement to be tailored with information that the wrapper will need to access instances of this data source type.

After the wrapper and server have been defined, the data at the remote source must be described in terms of the data model of the federated middleware. Since the federated database described here supports an object-relational data model, each collection of data from an external source must be described to the federated engine as a table with columns of appropriate types.

- A collection of external data modeled as a table is called a nickname, and its table name and column names are used in the SQL submitted to the federation by applications.
- Nicknames are identified via a CREATE NICKNAME statement. The following statement sets up a nickname for a collection of information about the weather, and identifies the "columns" that can be used in a query.

```
CREATE NICKNAME weather (zone integer, climate varchar(10), yearly_rainfall float) SERVER weather_server OPTIONS (QUERY_METHOD 'GET')
```

The "OPTIONS" clause is again a way to pass information that the wrapper needs, this time in order to process queries against the nickname.

In addition to storing data, many data sources also have the capability to perform specialized searches or other computations, represented in SQL as user-defined functions.

- For example, the user might like to do a temperature forecast based on location and date, to identify customers for an air conditioning sale.

```
SELECT c.Name, c.Address FROM customers c, stores s, weather w
WHERE temp_forecast(w.zone, :DATE) >= 85 AND c.ShopsAt = s.id
AND s.location = w.zone
```

Here, the function `temp_forecast` is used to represent the data source's ability to do a temperature forecast for the nickname `weather`.

- User-defined functions that are implemented by an external data source are referred to as mapped functions. These are identified to the federated system via DDL statements.

```
CREATE FUNCTION temp_forecast (integer, date) RETURNS float AS TEMPLATE
DETERMINISTIC NO EXTERNAL ACTION
```

The `AS TEMPLATE` clause tells the federated database that there is no local implementation of the function.

- Several function mappings may be created for the same function. For our example, the following statement accomplishes the mapping:

```
CREATE FUNCTION MAPPING tf1 for temp_forecast SERVER weather_server
```

The above DDL statements produce metadata describing the information about nicknames and the signatures of mapped functions. This metadata is used by the federated query processing engine and is stored in the global catalogs of the federated database.
An application submits a query written in SQL to a federated server. The federated server optimizes the query, developing an execution plan in which the query has been decomposed into fragments that can be executed at individual data sources. The optimizer chooses among alternatives on the basis of minimum estimated total resource consumption. Once a plan has been selected, the federated database drives the execution, invoking the wrappers to execute the fragments assigned to them.

To execute a fragment, the wrapper performs whatever data source operations are needed to carry it out, perhaps a series of function calls or a query submitted to the data source in its native query language. The resulting streams of data are returned to the federated server. The federated server combines them, performs any additional processing that could not be accomplished by a data source, and returns the final result to the application. The federated server’s optimizer and the wrappers together arrive at a plan for executing the query. By default, the optimizer uses a greedy dynamic programming method to select from possible query plans. At each level, the optimizer considers various join orders and join methods, and if all the tables are located at a common data source, it considers performing the join either at the data source or at the federated server. (See next figure.)
Federated Databases Examples:
Major Pharmaceutical Company

Research labs in both Europe and the U.S. with multiple sources of data, and a need to combine the information from these various sources.
- Each of the labs houses scientists looking for new drugs to battle particular diseases.
- Databases of chemical compounds, stored in specialized systems (structural similarity searches).
- High throughput screenings of compounds for testing.
- Other data sources include large flat files of genomic and proteomic information, patent databases, spreadsheets of data and analysis, images, and text documents.
- Different experimental setups.
- Access to the data being produced at another site.
- The test result data changes rapidly.
- The warehouse must either be replicated at both sites, for performance reasons.
- The compound data, today stored in specialized repositories.

Federated Solution
- Data is left in the existing data sources, with their native access paths, and current applications run unchanged.
- It is easy to build new applications that can access data from any of the sources, regardless of continent.
- Local data stays local, for rapid access.
- The less frequently used remote data is still accessible, as needed, and queries are optimized by the federated server to ensure that they are retrieved as efficiently as possible.
- Replication can still be used if desired for those portions of the data that are heavily accessed by both laboratories.

Federated Databases Applications

- Heterogeneous Replication
  - Federated technology makes it easy to not only move data, selecting it from the sources and inserting it into the warehouse, but to re-shape it as well, aggregating information from the various outlets before inserting it into the warehouse. Heterogeneous replication is made easy thanks to federated technology.
- Distributed Data Warehouse
  - With IBM's federated technology, data marts and warehouse can be on separate systems, yet users of the data mart can still drill down with ease from their local level of summarization into the warehouse. Federated technology shields the users, who have no need to know that the data warehouse is distributed, by providing a virtual data warehouse.
- Integration of geospatial data with traditional business data
  - It requires advanced query analysis functions for correlating the data, and end-user tools that can visually display the data in a geospatial context. This kind of functionality is supported by federated systems.

Federated Databases Optimization of the existing functionality:
- Wrapping, caching, performance, standards, etc.

Static vs Dynamic Federation
- Development of wrappers is costly.
- New data sources may need new wrappers.
- How can you automatically add such data sources.

Semantics:
- Data integration only partially solved.
- Integration of applications as well as data remains a problem.
- Data quality.
- Annotation, ontology's, semantics of data and applications for a sound integration.
- etc.

References


