# A federated architecture for database systems*

*by* DENNIS McLEOD and DENNIS HEIMBIGNER
*University of Southern California*
Los Angeles, California

## INTRODUCTION

The contemporary approach to database system architecture requires the complete integration of data into a single, centralized database; while multiple logical databases can be supported by current database management software, techniques for relating these databases are strictly ad hoc. This problem is aggravated by the trend toward networks of small to medium size computer systems, as opposed to large, stand-alone main-frames. Moreover, while current research on *distributed databases*[1,2,3,4,5] aims to provide techniques that support the physical distribution of data items in a computer network environment, current approaches require a distributed database to be logically centralized.

### *Decentralized databases*

A *decentralized database* is a collection of (structured) information, which may be logically distributed, physically distributed, or both. Specifically, it is possible to identify two distinguishable though related aspects of database decentralization:

1. *Logical decentralization* concerns the division of database into components, for purposes of allowing separate control over each component; the control that may be exercised for each component includes specifying the meaning and logical structure of data, describing the accessibility of data items to users, and specifying the form in which users will see the data.
2. *Physical distribution* concerns the allocation of data for storage to the nodes of a network or other assembly of interconnected computer system components.

A comprehensive approach to decentralized database management must address both the issue of logical decentralization as well as the issue of physical distribution.

* This research was supported, in part, by the Joint Services Electronics Program through the Air Force Office of Scientific Research under contract F44620-76-C-0061.

### *The need for logical decentralization*

Traditionally, a database is viewed as a complete and total integration of the data associated with a family of related, though distinct, applications. A database has associated with it a single structural specification: its conceptual/logical *schema*. Users and application programs manipulate the data by performing operations phrased in terms of the schema. The database is then physically realized by a particular *physical design*, which is a collection of storage structures and access methods that actually implement the schema.

In contrast to the approach in which data files are closely associated with application systems and isolated from one another, the "integrated database" approach is founded on the principle of logical centralization. The complete centralization of data at the logical level has many benefits associated with it; in fact, these benefits are largely responsible for the great success of the "database approach" during the past decade:

- Complete integration provides a global view of the data resources of an organization, and provides a basis for the resolution of conflicts; the importance of the database as an organizational resource is recognized.
- By constructing a single integrated database, the amount of data redundancy in the overall information system is significantly reduced; this reduction in redundancy diminishes the opportunities for data inconsistencies and related problems.
- Centralization enables the more ready implementation of database applications that require data from several sources.
- Logical centralization of a database allows uniform modes of access and usage to be established for all the data.

While logical database centralization has important associated benefits, it does impose certain limitations on database-intensive information systems. Specifically, it is often extremely difficult to completely integrate applications that are related, yet separate; integration may go too far in tightly coupling together aggregates of data that ought to retain some individual autonomy.

In the conventional view of database design, based on the concept of complete logical centralization/integration, a central authority is responsible for designing and maintaining "the" database; this authority, be it a single person or a group of individuals acting together, is usually called the *database administrator* (DBA). The DBA maintains control over all the data, and is responsible for determining and adjudicating the disparate needs of the various database applications and users. In such an approach, the ultimate providers and users of the data must relinquish their authority over it to the DBA; this raises a number of concerns:

— Users are often hesitant to entrust their data to an external authority, despite any assurances they receive; experience has shown that these reservations may indeed be valid.
— The DBA is charged with developing a unified specification of the content and meaning of a database. In practice, this is a difficult task, since various database users will have different perceptions of the data.
— In the process of selecting a physical design for a database, the DBA must ascertain the global usage patterns and response requirements, and then select the alternative physical design that provides the best overall performance. While this approach may indeed best serve the overall organization, it may not be well suited to the needs of the principal users of a given portion of the database. A compromise physical design that attempts to satisfy all database users may in fact satisfy none of them.
— Charged with the problem of serving as a liaison between the database and all of its users, the DBA can easily become a bottleneck. All requests for database extensions and revisions are funnelled through the DBA; this indirection can and often does introduce serious delays and inconsistencies, particularly as the complexity of a database grows.
— New databases are often created as combinations of old databases; that is, it is not always true that a database is designed in strictly top-down fashion. In consequence, it is often very difficult to try to totally integrate two related, but separate, databases into a unified whole.

### *Distributed databases*

One particular approach to database decentralization is commonly called *distributed database systems*. In this approach, a single logical database schema is defined, which describes all the data in the database system; the physical realization of the database is then distributed among the computers of a network. The physical data of a distributed database can be divided in three ways:

1. *partitioned*, where no data is duplicated,
2. *fully duplicated*, where all data is duplicated in every computer,
3. *partially duplicated*, where some data is duplicated at some computers.

There are two main advantages to distributed databases:

1. A distributed database system is potentially more efficient than a physically centralized database system, because the data can be placed close to where it is needed. If it is needed at two or more places, then it can be duplicated.
2. If data is duplicated, then a distributed database is potentially more reliable than a physically centralized database, because even if one computer fails other computers in the network may continue to operate.

Although there are advantages to distributed database systems, there are also a number of difficult design issues associated with them:

— How can the data be optimally allocated to the computers to minimize some cost, such as access time or physical storage space?
— How can a distributed database continue to operate after the failure of a computer?
— How can duplicated data be kept consistent?

In response to the observation that decentralized computing systems are of increasing general importance, and the realization that logical centralization of a database (with or without physical decentralization) has many problems in practice, it is clear that a fresh approach is required. The goal of this new approach must be to serve as a compromise between total integration/centralization and the disorganization of completely diffused and decentralized databases. The key to successfully realizing this goal is to balance the need for decentralization and the largely conflicting need for effective sharing of information.

## A FEDERATED APPROACH TO DATABASES

The approach to database decentralization advocated here is termed *federated databases*; the basic idea of federation was introduced by Hammer and McLeod.[6] A federated database consists of a number of logical *components*, each having its own logical/conceptual schema (*component schema*). These components are related, but independent, and they may or may not be disjoint. Typically, a component of a federation corresponds to a collection of information needed by a particular application or a set of closely related applications.

All of the components in a federation are tied together by one or more *federal schemas* that express the commonality of data throughout the federation; these federal schemas are used to specify the information that can be shared by the federation components, and to provide a common basis for communication among them.

Database system users and application programs manipulate a database by issuing *transactions*, viz., operations that retrieve information from or modify information in a database. As a database user or application program is most commonly affiliated with a single component of a federation, that

user (or application program) normally issues transactions that can be performed within the local component. This property may be termed locality of reference and is fundamental to federated database systems.

On occasion, a user of component C1 may need to issue a transaction that involves data that belongs to another component, C2 (or several other components). In this case, the user consults a federal schema to find the necessary data; this reference can be explicit or implicit (i.e., the user may either refer to the data in the context of the federal schema, or may refer to it as local derived data (in which case the derivation specification must have already been provided)[7]. A transaction involving nonlocal data is processed by issuing a request to the *federal controller*, which issues the necessary instruction to C2 to actually provide the necessary data. While transactions involving local data execute with all possible speed, transactions that require non-local data are in general substantially less efficient, because the federal controller must intervene to perform data movement and translation. The federal controller is thus an important part of a federated database system, playing the role of coordinator and translator.

In the federated approach, the (conceptual/logical) schema of each component is defined by a *component* DBA. A component schema is designed to suit the users and applications of the component; and, the physical design used to implement a component schema is developed (and will evolve) so as to best satsify the performance requirements of these local users. In this way, the principal goal of each component is to satisfy its most frequent and important users (viz., the local ones).

All federal schemas are defined and controlled by the *federal* DBA. Each federal schema is a virtual one, in the sense that there does not exist a physical database that corresponds to it; rather, a specification is provided that describes how the federal schema constructs are materialized from data maintained by the individual components. In particular, each component defines a subset of its component schema as available to the federal schema(s).

The duties of the federal DBA supplement, rather than conflict with, the activities of the component DBAs. The principal responsibility of the federal DBA is to define the federal schema(s), relate them to the component schemas, and define the interface that each component must provide. The federal DBA is also responsible for determining how logical redundancy in the federation ought to be handled: in some cases, it is appropriate for a single component to take responsibility for it; in other cases, it is better for each component to maintain its own version (with a variety of possible consistency restrictions established to ensure that the various versions remain appropriately related, e.g., the same). The choice may be determined for reasons of efficiency, reliability, or requirements of components that need to access the data.

In addition to directly accommodating logical database decentralization, the federated architecture also enhances the evolvability of a database. A federation evolves either by changes to components or changes to a federal schema. As long as a component continues to support its interface to the federation, it is free to change either its physical structure or its logical structure without affecting other components (except possibly with regard to performance). The federal schema can change for one of four reasons:

1. a deliberate policy decision to change the federal schema,
2. a radical change in a component that requires a change in its interface to the federation,
3. adding a new component to the federation,
4. deleting a component from the federation.

Changes that enlarge or restructure the federal schema, such as by the addition of components, will impact components to the extent that they must accommodate the new information in the federal schema. Changes that actually remove information, such as the deletion of a component, in general require other components both to accept an altered federal schema and to redesign transactions that access the deleted information.

In sum, in the federated approach, primary control over a database component resides with its principal maintainers and users, but adequate centralized authority is exercised in order to ensure appropriate levels of sharing, data compatibility and data consistency. Each federation component can determine how to optimize its part of the database according to its own needs, and can decide what information should be made available to other components. Sharing of information is accommodated by the federal schema, and conflicts are resolved by the federal DBA. Finally, the federated database architecture is based on the observation that many contemporary integrated databases are actually better suited to partial decentralization than complete centralization; for example, despite the availability of an integrated database, it is often the case in practice that the functional units of an organization make use of only a subset of the total schema and a limited portion of the data; in such cases, the remainder of the database can actually be a burden to a user.

## DESIGN ALTERNATIVES FOR FEDERATED DATABASE SYSTEMS

Any design for a federated database system must deal specifically with the following issues:

- the precise structure of the federation (viz., the number and organization of the federal schemas, and their relationship with the component schemas),
- the handling of physical data storage and access in the federation,
- the specific approach to the operation of the federal controller,
- the component facilities to support interaction with the federation.

These four important design issues are specifically examined immediately below.

*Logical distribution*

The logical distribution of a federation determines the ease with which changes to the schemas can be made and ease of maintaining the federal schemas. There are four principal logical distribution alternatives, with differing ability to handle change and maintenance:

1. The first logical distribution strategy involves a single, global, federal schema derived from all the components. This structure is simple for the federal DBA to maintain, because there is only one federal schema. But such a comprehensive federal schema is difficult for the DBA to design, because it must reflect all the desired interactions between components. In addition, components are restricted from making radical changes in their component schemas because it may require changes to the federal schema. Components may be prohibited from seceding from the federation, because that may also require changes to the federal schema.
2. An alternative distribution uses a separate federal schema for each pair of components. In the worst case of n components totally interconnected, there will be $n(n-1)/2$ federal schemas. Defining and maintaining this number of federal schemas may well place an intolerable burden upon the federal DBA, particularly for a large n. However, it may be that for a given federation only some small portion of the possible interconnections is needed. Each pairwise federal schema is simpler than a global schema, since fewer components are interacting. In this pairwise federal schema approach, adding or removing components is simple: the component and its federal schemas are removed.
3. A third logical distribution alternative is to associate a federal schema with each component, for use by all other components. Each component maintains two interfaces, a local one for its users and another one for use by all other components. In this approach, it is easy to add or remove components, and the number of federal schemas is equal to the number of components.
4. A final logical distribution strategy is a variation of the global distribution strategy: instead of a single global federal schema, there are several federal schemas arranged in a hierarchy. In this organization, the components are separated into disjoint sets with a federal schema for each such set. The federal schemas at the first level are partitioned into groups, and a second level set of federal schemas is defined upon the sets of first level federal schemas. This continues until a single federal schema (designated the root) is constructed. The result is a tree of schemas; the leaves are the component schemas and all interior nodes are federal schemas. In this approach the effects of adding or removing a component may be limited to some subtree of the hierarchy. Clearly, in this strategy, the simplicity of the federal schema hierarchy is determined by the criteria used to structure the tree.

Also at issue in logical distribution is the nature of the view seen by a user associated with a given federation component. A user associated with a given component must be able to access both local data, through the local schema, and non-local data through a federal schema. The local data is accessed by the normal mechanisms of the system (i.e., a data manipulation facility/language or programming language interface). Access to non-local data depends upon the user's view of the federation. At one extreme, the federal schema is integrated with and extends the local schema in such a way that the user cannot tell if he is accessing local data or non-local data. At the other extreme, the federal schema is separate from the local schema, although not necessarily disjoint; in this situation, the user must specifically address his request to the local schema or the federal schema.

Complete integration makes it simple for a user to express a database transaction, since both local and non-local data look the same; the principal problem with this approach is that the user cannot directly observe that a potentially expensive non-local reference may be required to process the transaction. When there is no integration, the user must perform extra steps to retrieve non-local data, which may then be combined with a manipulation of local data. In this case, the user knows that a potentially expensive non-local reference is needed. There is, of course, a viable middle ground between these extremes, in which the user sees two separate schemas, (component and federal) and the database transaction processor accepts combined references to both local and non-local data. In this way, the user knows a costly non-local reference is being made, but the details of accessing are delegated to the database system.

*Physical distribution*

The federated database architecture does not assume that a database will actually be supported in a distributed environment; that is, it is not assumed that the database is to span a number of nodes in a computer network. A federated database could well be implemented on a single computer. However, there are advantages to physically distributing data:

1. to achieve better performance and allow higher degrees of concurrency by placing data close to its principal sources and users,
2. to provide a higher degree of reliability and survivability by redundantly storing data items.

The federated database architecture directly addresses the first of these two main goals; the concept of locality of reference is key in the federated architecture. Moreover, the federated architecture provides a basis, through the federal schema and federal DBA, for establishing a policy for redundant data storage.

As noted above, one of the main principles of the federated database architecture is that the responsibility for storing and supporting physical access to the data in each compo-

nent of a federation is the responsibility of that component. Thus, the most general approach might be to allow each component to choose its own method for storing data; if a computer network is being used to implement a federated database, each component may distribute its own data throughout the network.

However, intolerable complexity may result from complete flexibility for physical distribution along with complete flexibility for logical decentralization. Moreover, logical decentralization and physical distribution are not orthogonal issues. In consequence, it is appropriate in many cases to directly combine logical decentralization with physical distribution. In this approach, if a computer network is available for database implementation, then each federation component is allocated to a node in the network. The matching of a federation component to a node in a computer network provides a direct and natural way to implement a database that can be both logically decentralized and physically distributed.

Another aspect of physical distribution is the control of duplicate data. When two components contain duplicate information in their schemas, the federal DBA must decide how that data is to be handled in the federation. Duplicate data can be eliminated from the physical level of the federation by selecting one copy as the official copy; all references to a specific data item then refer to the official copy.

If duplicate data is retained, and it is desired that it be kept consistent (i.e., that all copies ultimately reach the same value after database modifications cease), then it is possible to apply the techniques developed for controlling duplicate data in distributed databases.

A number of control algorithms for maintaining consistency in distributed databases have been developed;[5] the algorithms that support partially duplicated data are directly relevant to federated databases. The proposed algorithms for maintaining the consistency of partially duplicated data are complex, since they attempt to keep all duplicate data as current as possible. This is important in a distributed database system so that the users continue to see a logically centralized database. However, complete consistency is not necessarily important for federated databases, because they are logically decentralized. In consequence, it may be possible to apply looser and simpler algorithms for controlling duplicate data in the federated environment.

*Federal controller operation*

A federated database requires a control component not present in conventional (centralized) database system: the federal controller. As described above, the federal controller performs the bulk of the transformations necessary to satisfy a request from a component for information described in a federal schema (and that is contained in another component); the request takes the form of a specified transaction. The federal controller must perform a sequence of seven steps for each such request/transaction:

1. The transaction is checked for legality against the federal schema. The access rights of the requester are also verified at this time.
2. The transaction is decomposed into a collection of simpler *target transactions*, each of which can be ultimately satisfied by a single *target component*. The target component is the component that supports that part of the federal schema referenced by the target transaction.
3. Each target transaction is translated from a reference to the federal schema to a reference to the target component schema.
4. The target transactions are sent to the corresponding target components for processing.
5. The federal controller waits for all the target transactions to be processed, and then the controller collects the results.
6. The results are translated from target schema form back to federal schema form.
7. The translated results are combined and returned to the requester.

Steps five through seven can be performed in either *set-at-a-time* or *element-at-a-time* fashion. In set-at-a-time processing, the federal controller collects the results from all of the target components into a single result set, which is then returned to the requester. In element-at-a-time processing the federal controller translates and returns to the requester each element of the result as it is made available by a target component. The choice between set-at-a-time and element-at-a-time processing should be made based on storage cost and communication cost information.

The federal controller can itself be either centralized or distributed. If a computer network is used for implementing a federated database system, then there are three main approaches to federal controller placement:

— The federal controller resides on a special node of the network, i.e., one which does not also contain a component. This approach has the advantage of isolating the federal controller, and the controller node need possess only the computational power necessary to perform the controller's functions. In this approach, it is also possible to easily replace the controller, should it fail. The disadvantages of a special controller node include the need for additional hardware, and the potential problem of a system performance and reliability bottleneck.
— The federal controller can be co-located on a node with one of the components of the federation. This saves the cost of extra hardware, at the cost of possible competition for node resources with the component controller. The controller can also be made to migrate from one component node to another, should a node fail or a performance improvement be possible by shifting control.
— The federal controller can be distributed, in which case a part of the controller is located at every node (or some subset of the nodes). This has advantages for

reliable operation, but the coordination of all the controllers is a difficult problem.

The choice between a centralized or a distributed federal controller must be made in the context of the relative complexity of the algorithms for supporting coordination (analogous to work on distributed database control algorithms[5,8,9,10,11,12]), and the relative storage and communication costs involved.

*Component control*

In most respects, the control aspects of a component of a federation are the same as those for a centralized database system; but since a component is part of a federation, it must support an appropriate interface to the federation. In particular, there are several important issues that a component must address, vis-a-vis its interaction with the federation:

- The component must allow for concurrent access to its data, because while a local user is accessing some part of the component data, some other component may be attempting to simultaneously access the same data (through the federal controller). If the component already has the capability for concurrency control for local users, then requests by the federal controller present no difficulty. Otherwise, the component software must be augmented by software to control the simultaneous access attempts.
- The component software must provide for communicating results back to the federal controller, on either a set-at-a-time or an element-at-a-time bases. Set-at-a-time processing requires bulk transfer of information to the federal controller, and element-at-a-time processing requires the buffering of the results at the component followed by single element transfers to the federal controller.
- The component must recognize locally-issued transactions that require accessing the federal schema, and forward an appropriate request for processing to the federal controller. When the federal controller returns the result of a transaction, the component combines the results from the federal controller with the results of any portion of the transaction that referenced data local to the component.

In sum, it is the combined functioning of the components and the federal controller that allows a federated database system to effectively support information sharing and the decentralization of data.

## SUMMARY

A federated architecture for database systems has been presented, which supports the logical decentralization of databases, and provides a basis for database physical distribution (in a network of computer systems). The federated architecture responds to a number of problems associated with the complete centralization and integration of database systems (as detailed above).

A federated database consists of a number of logical components, each having its own user-level structural specification (component schema). The components of a federation are related, but independent, and they may or may not be disjoint. Typically, a component corresponds to a collection of information needed by a particular user or application. The components in a federation are tied together by one or more federal schemas that describe the data that is to be shared by the various federation components, and provide a common basis for communication among them. A federal controller, which is an essential constituent of a federated database system, supports communication and translation of data among federation components, based on the federal schema(s).

In this paper, a number of design issues and alternatives for federated database systems have been reviewed. Alternative logical distributions and physical distributions were described, and the issues relevant to the operation of the federal controller and federation components discussed. We are presently developing a specific design approach based on the principles described in this paper.[7] A critical aspect of our present approach is the use of a meaning-based (semantic) database description and structuring formalism (database model) (such as those described in [13,14,15,16,17,18,19,20,21]) to specify the component schema interface with the federal schema(s).

## ACKNOWLEDGMENTS

The authors are grateful for the very helpful efforts of Michael Hammer of MIT, a co-developer of the federated database concept. Gerald Short of TRW and the CADAM group of the Lockheed California Company (under Don Kawamoto) have provided additional motivation for the federated architecture.

## REFERENCES

1. Champine, G., "Six Approaches to Distributed Databases," *Datamation*, Pages 45-48, May 1977.
2. Lien, Y. E. and Ying, J. H., "Design of a Distributed Entity-Relationship Database System," *Proceeding of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1978.
3. Miller, M., "A Survey of Distributed Data Base Management," *Information and Management*, Pages 243-264, 1978.
4. Ramamoorthy, C. V. and Wah, B. W., "Data Management in Distributed Data Bases," *Proceedings of National Computer Conference*, New York, NY, 4-7 June 1979.
5. Rothnie, J. B. and Goodman, N., "A Survey of Research and Development in Distributed Database Management," *Proceedings of International Conference on Very Large Data Bases*, Tokyo, Japan, 6-8 October 1977.
6. Hammer, M. and McLeod, D., *On the Architecture of Database Management Systems*, Technical Report 79-4, Computer Science Department, University of Southern California, Los Angeles CA, April 1979.
7. McLeod, D., *An Approach to Database Decentralization*, Technical Report, Computer Science Department, University of Southern California, Los Angeles CA, 1980 (to appear).

8. Bernstein, P. A., Rothnie, J. B., Goodman, N., and Papadimitriou, C. D., "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (the Fully Redundant Case)," *IEEE Transactions on Software Engineering,* Volume SE-4, Number 3, May 1978.
9. Bernstein, P. A. and Shipman, D., "A Formal Model of Concurrency Control Mechanisms for Distributed Database Systems," *Proceedings of Third Berkeley Conference on Distributed Data Management and Computer Networks,* Berkeley CA, 29-31 August 1978.
10. Garcia-Molina, H., "Performance Comparison of Two Update Algorithms for Distributed Databases," *Proceedings of Third Berkeley Conference on Distributed Data Management and Computer Neworks,* Berkeley CA, 29-31 August 1978.
11. Stonebraker, M. and Neuhold, E., "A Distributed Database Version of INGRES," *Proceedings of Second Berkeley Conference on Distributed Data Management and Computer Networks,* Berkeley CA, May 1977.
12. Thomas, R. H., "A Majority Consensus Approach to Concurrency Control," *ACM Transactions on Database Systems,* Volume 4, Number 2, June 1979.
13. Buneman, P. and Frankel, R. E., "FQL—A Functional Query Language," *Proceedings of ACM-SIGMOD International Conference on the Management of Data,* Boston MA, 30 May-1 June 1979.
14. Chen, P. P. S., "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems,* Volume 1, Number 1, Pages 9-36, March 1976.
15. Codd, E. F., "Extending the Database Relational Model," *ACM Transactions on Database Systems,* vol. 4, no. 4, December 1979.
16. Hammer, M. and McLeod, D., "The Semantic Data Model: A Modelling Mechanism for Database Applications," *Proceedings of ACM SIGMOD International Conference on the Management of Data,* Austin TX, 31 May-2 June 1978.
17. Hammer, M. and McLeod, D., *SDM: A Semantic Database Model,* Technical Report, Computer Science Department, University of Southern California, Los Angeles CA, 1980 (to appear).
18. McLeod, D. and King, R., "Applying a Semantic Database Model," *Proceedings of International Conference on the Entity-Relationship Approach to Systems Analysis and Design,* Los Angeles CA, 10-12 December 1979.
19. Shipman, D., "The Functional Data Model and the Data Language DAPLEX," *ACM Transactions on Database Systems,* 1980 (to appear).
20. Smith, J. M. and Smith, D. C. P., "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems,* Volume 2, Number 2, Pages 105-133, June 1977.
21. Su, S. and Lo, D., "A Semantic Association Model for Conceptual Database Design," *Proceedings of International Conference on the Enti-Relationship Approach to Systems Analysis and Design,* Los Angeles CA, 10-12 December 1979.