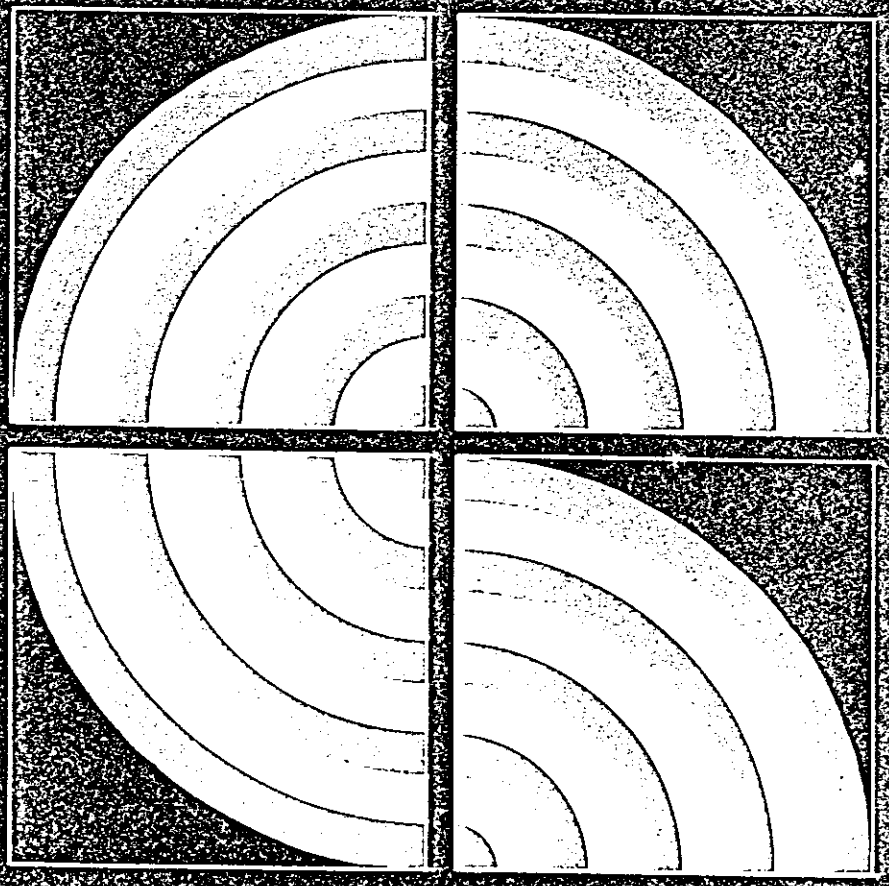
British Computer Society Workshop Series

Proceedings of the Sixth British National Conference on Databases (BNCOD 6)

University College, Cardiff, 11–13 July 1988

Edited by W. A. GRAY



From Database Systems to Multidatabase Systems: Why and How

W Litwin
INRIA, 78150 Le Chesnay, France

The concept of a database (ie an integrated, centrally administered, non-duplicated collection of company wide data) has been applied so widely now that many companies have many databases on one, or possibly many, computers. Users, therefore, now need shared access to multiple databases. In determining the principles for accessing such databases should one reapply the principles behind the database approach, but at one level up, or should new principles and types of system be introduced.

In the following paper it is claimed that database principles are inapplicable to collections of autonomous databases, unlike they were to files. New principles offer the only possible solution and they will deeply modify the design of database systems at all levels. A new type of system is appearing, referred to as a multidatabase system. The various capabilities which such a system should provide are analysed and the basic capabilities for multidatabase management which are beginning to appear in major relational systems and industrial prototypes are presented.

Finally, some research issues are discussed.

1. Introduction

Database systems were proposed as a solution to the problem of shared access to heterogeneous files created by multiple autonomous applications. These files were hard to manage by a single application. They presented duplications and various types of heterogeneities, such as differences in field naming, value types and file structures for a similar purpose. In particular it was difficult under these conditions to provide interfile consistency and overall privacy and efficiency.

To remove these difficulties, it was proposed to replace the autonomous files by a centrally defined collection of data called a database. The authority responsible for the centralized control was called the database administrator. His task was to make the database integrated which meant that it should be defined without duplications and heterogeneities. The database should then be managed under a centralized control by a system called a database system (DBS). This system should in particular give each application the illusion of being alone to use data, while providing overall consistency, privacy, efficiency etc.

The idea was successful to a large extent. There are many databases in any larger company and frequently even on the same computer. There will be even more on workstations and servers on local nets. Unavoidably, users now need shared access to multiple databases. The developments in distributed computing and in networking gave the technical basis at least for the physical access. The question arose as to what the corresponding principles should be. Should one reapply the database approach principles one level up or should new principles and types of systems be introduced ?

In what follows we will claim that database principles are inapplicable to collections of autonomous databases, unlike they were to files. New principles offer the only possible solution and they will deeply modify the design of database systems at all levels. A new type of system is appearing that we have called multidatabase systems. We will analyze various capabilities a multidatabase system should provide. We will show that the basic capabilities for the multidatabase management have started to appear in major relational systems and industrial prototypes. Finally, we will discuss some research issues.

2. Database Design

2.1 Principles

Database principles were elaborated when users had generally no direct access to data and systems. The computer universe was so complex that they had to employ application programmers for their data manipulations. The systems were mostly batch and business oriented. Therefore manipulations were rather short and could be rerun.

The main idea in the database design in these conditions was that the database administrator and system should insulate the users and even application programmers from the complexity of shared data management . The administrator was supposed to analyze the needs of all applications sharing the files and/or the database to be created. From this analysis, he should define:

- the conceptual schema of the database. This schema should define all the data in the database. The whole collection was supposed to be:
  - exhaustive. It should contain all data needed by the applications or at least data sufficient for deriving the application data through views.
  - integrated. This meant that data of the same type should constitute a single data type. Also, there should not be replicated data types or replicated data in a type.
  - consistent. This means that data should respect some predicates defined by the administrators. The predicate may concern a single data type or may interrelate different types.
  - confidential. This means that the interuser privacy should be respected.
- the internal schema of the database. At this level the administrator was supposed to choose data structures providing optimal performance of the database.
- the external schemas of the database. These schemas should adapt the conceptual schema to needs of a particular user if the conceptual data definition was not that required by the user. The adaptation could be a simple restriction or a translation of names or value types or of data structures.

Ex. 1: Consider a company having suppliers supplying parts. The relational database design principles would lead to the definition of a database with three well known tables S, SP and P. The choice of table names, columns, column names and of value types would be made by the administrator for all users. The administrator would also define the internal structures for the maximal efficiency of the database. Finally, views would be used to accommodate the needs of users requiring names or structures or value types different from the common ones.

The database system was intended as a computer emulation of the database administrator. Its objectives were:

- to provide the database administrator with the tools for the definition of the conceptual schema and of external schemas, including the consistency and privacy constraints.
- to provide him with the tools for the internal schema definition.
- to give any user the possibility of manipulating his data as simply as possible.
- to give any user the illusion of being the only user of the database.

At the language level, two main concepts were created to fulfill these goals. They were the data definition language and the data manipulation language.

To enhance the simplicity of the user manipulations further, but also to allow the administrator to choose the physical structures independently of an application coding, the principle of the independence between the logical and the physical level was introduced. Finally a number of capabilities was proposed to ensure the above objectives and in particular:

- the concurrency control. The system should be able to manage concurrent accesses in a transparent way.
- the database reliability. Users should not be concerned with the crashes of the database, operational errors of other applications, etc.

In both cases, it was assumed that if a manipulation is not executed as it should be, then the system simply reruns it.

2.2 *Motivations*

The administrator was supposed to act as a mediator whose action should remove data from independently created files. The reason for the goal of exhaustivity at the level of the conceptual schema was to let an application program carry out operations in as simple way as possible, like get record or get next, etc. The main reason for integration was to avoid multiple searches and replicated updates that created difficulties in multiple and heterogeneous files. The idea behind consistency was to maintain known relationship between values in different structures. Finally, confidentiality was needed to prevent the leak of user information as the sharing of user data should be transparent.

It is worth recalling the ultimate purpose of all this methodology. It was the easiness of data manipulation by a user or rather an application program. All other concepts were tools to achieve this goal. In particular, the notion of conceptual schema and of various complex data structures proposed for such schemas were tools:

- to resolve the problem of binding of names and of corresponding value types in the query. The name in the query should be one of the names in the schema.
- to leave to the user only the problem of traversing a data structure through simple statements like those above, the structures themselves being already identified and provided.
- to have for DBS the explicit definition for the corresponding consistency and confidentiality requirements.

Ex. 2: Consider the database dealing with the suppliers and parts. The goal of the user wishing data about suppliers of some parts, is probably only to for-

mulate the manipulation using the most convenient form for himself, his vocabulary in particular, and to obtain data in the form wished explicitly or implicitly. The user basically does not care what, if any, the schema of the database is. Also, every other capability of the DBS is not of interest to the user and should remain transparent.

Note that ideal is basically not yet the case of the current database systems. Users are supposed to know the schema, at least to know how they should name suppliers and parts to be understood by the system. Then, they obtain value types and data names as they are in the database, unless they specify conversions (value expressions and labels of SQL).

2.3 Drawbacks

The database approach is a generous idea, but unfortunately largely utopian with respect to its goals. There are indeed fundamental problems with the concept of data sharing that cannot have a solution:

- the administrator is in charge of optimizing the usage of the database in priority for all users. The local optimization for a user may be in contradiction with the global goal.
- the user has to explain his needs to the administrator. This may be a difficult process and the needs may change.
- there may be very many data structures one may imagine over a collection of data. It is unrealistic to assume their existence in a conceptual schema for simple expression of queries, name binding etc. The popularity of the relational model with its very simple data structures, but with the manipulation language undoubtedly more complex than CODASYL statements is a proof of the failure of this idea. The complexity of the language is in fact the tool for dynamic definition and creation of data structures that ideally should be all in the conceptual schema and the database.
- a user is supposed to use the common (globally optimized) data names, values and structures, unless an external schema provides him with a more subjective picture. However, this view must be subject to the following constraints:
- there is no way to introduce attributes that do not exist in the conceptual schema.
- the precision of derived values cannot be greater than that in the database.
- there may be no way to update the view data, as the corresponding update to the database may be undecidable or it may violate an integrity constraint the user should not be aware of.
- sometimes the user data may be badly affected by side-effects of other users' manipulations or of an arbitrary decision of the administrator. For

instance, if the administrator changes a table definition, in most current DBSs, the view definition would no longer be valid at the execution time.
- manipulating a large collection of data is fundamentally more complex than manipulating a small one. If most user needs are directed to a subset of data, a large database management must be less efficient than the management of the user data only.
- in particular, data that some users manipulate today through various nets seem already too large a collection to be ever manageable as a database.

Summing up, the database approach is an attractive idea, as it proposes to free the users from many annoying aspects of data manipulation. The required price is however the loss of the user control over his own data e.g. of the user autonomy. Some of the corresponding drawbacks may be corrected through the sophistication of DBSs. Some are however inherent to the foundations of the database approach and more specifically to the complexity of sharing a large collection of data.

These drawbacks become therefore even more pronounced in the current situation where a user has the capabilities to access even very large collections of data. The data may especially be in many autonomous databases, like thousands of public databases, on the same large computer or on several interconnected computers. These databases present heterogeneities similar to those that were in files and triggered the whole database methodology. One may feel therefore that the database design principles should be reapplied, e.g. all data should be made again a (distributed) database. This would mean centralized control and at least the presence of the common conceptual schema (the global schema). We felt that the increase in the size of the collection makes the drawbacks important enough to prohibit the reapplication of this approach. This reason has motivated us to propose the multidatabase approach in [19] and its earlier less known references.

3. Multidatabase Design

The multidatabase design principles were intended as a generalization of the database approach, keeping its advantages, while avoiding or lessening the drawbacks. For this purpose, it was proposed to consider that the data the user may need to manage together may be in several databases without a global schema. These databases may not therefore be mutually integrated and data in different databases may present duplications and discrepancies in naming, data structures etc. as well as inconsistencies the database design was supposed to remove. These phenomena are considered as the user's right (autonomy) to satisfy firstly his own needs. The user should nevertheless be able to manipulate data not only in his database, but also to combine data from different databases. These multidatabase manipulations should further

be simple e.g. non-procedural, despite the heterogeneity. To attain this goal the following methodology was proposed. Details are in [19] - [22] and the references of these articles.

3.1 The concept of a multidatabase system

A multidatabase system (MBS) is a system for the management of several databases without a global schema. The system is therefore supposed to provide two new functions for data management:

- a multidatabase manipulation language for queries (and updates) to more than one database,
- possibly, a language for the definition of interdatabase dependencies.

The concept of a multidatabase system is widely discussed in [19] though it was proposed in earlier references of [19]. By analogy to the concept of a database language, the overall language of an MBS for both data definition and manipulation was called a multidatabase language. A set of (multi)databases for which a multidatabase language exists was called a multidatabase. Thus a set of databases without a multidatabase language is just a set, whereas provided with a multidatabase language it becomes a multidatabase. This principle has its roots in that of making a set of tables (flat files) a relational database.

One motivation for the concept of a multidatabase system was easy access to thousands of databases available today. Another intention was to allow applications to be designed in a way where different data could reside in dedicated databases, especially on users' workstations. It was felt that the user and the administrator should more frequently be the same person so the user autonomy would be better preserved (the programmers representing the user become eliminated anyhow by the general progress). Furthermore, the databases should usually be much smaller and the computational complexity should decrease. The need for systematic sharing should be replaced by that of occasional cooperation. The user should in particular be more able to optimize his own needs with respect to names, value types data structures and physical optimization. If a conflict with other users occurred, it would affect only the global level, the user remaining able to preserve his own needs in priority. All together, it was expected that the drawbacks that would affect many applications if designed according to the database approach, should disappear or should be attenuated using the new approach.

Ex. 3: Consider users working in a largely autonomous departments. It is likely that most manipulations in each department would concern this department data e.g suppliers and parts. The multidatabase approach would then be

to constitute one database per department. A multidatabase definition language should allow the definition of the databases as well as the dependencies if needed. The multidatabase manipulation language should let the user manipulate a database and combine if needed data from different databases. Each department could have priority in the control of its data, as it has its own database and the corresponding drawbacks of the database approach could be avoided.

Indeed, a database could be designed in cooperation with others for the choice of some column names and value types, as well as for the choice of table names and structures. However, other columns and tables could be designed purely for local needs differing from one database to another. Furthermore, the user could have the possibility of adding columns or renaming some even if it could create a conflict at the multidatabase (global) level with users in other departments. The limitations of a view concept on definition and manipulation of a department data would thus no longer apply. Also, the user could refuse an update that would be inconvenient for his department data, even if an inconsistency would appear at the multidatabase level, etc. If however, some interdatabase consistency should be enforced, the appropriate dependencies would allow the administrators to preserve them.

3.2 Reference architecture

The multidatabase language is assumed to be backed by specific capabilities at the implementation level. These capabilities may be more or less complex and numerous depending on the generality of the system. They should be embedded in the reference architecture for an MBS as in Fig. 1. This architecture comes from [19] and extends the classical database architecture of ANSI-SPARC. The multidatabase architecture layers are as follows:

- at the bottom, there are existing DBSs.
- a DBS presents to the next layer, called the multidatabase layer, the conceptual schema of the database of the owner willing to cooperate. This schema may be the actual conceptual schema or a local external schema. In the latter case the actual conceptual schema is called an internal logical schema. The conceptual schema at the multidatabase layer may in particular support a different data model and may hide some (private) data. If at this layer some common model is required, it is the responsibility of each DBS to stick to it.
- the multidatabase layer includes in particular schemas for the definition of dependencies between subcollections of databases. The dependencies may be transitive and uni or bidirectional. They are intended for database administrators and allow them to tie the databases together more or less for interdatabase consistency, privacy, etc. In the absence of the global

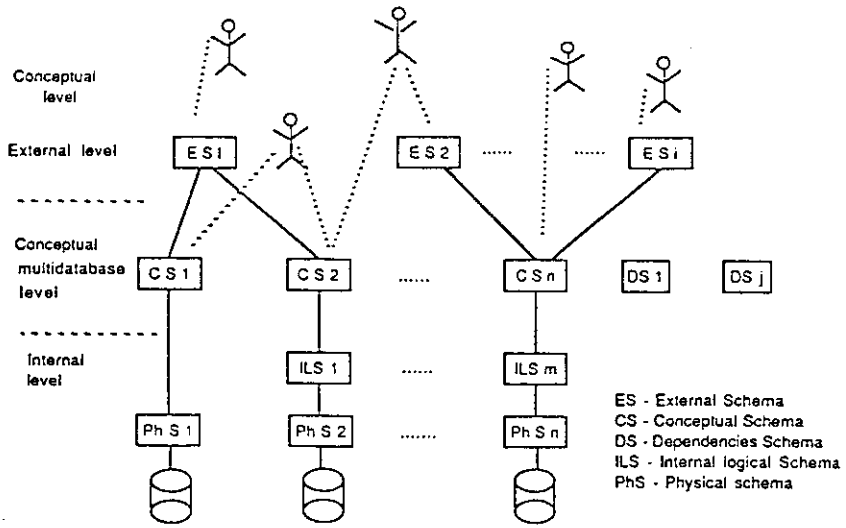


Fig. 1 General Architecture of a multidatabase system

schema they may be the only tool to preserve the consistency of data in different databases.

- above this layer, one may construct external schemas. These mono or multidatabase schemas may in particular present subcollections of databases as single integrated databases. An actual database may however enter different external schemas. It may also be manipulated locally. Thus, unlike with the global schema, even if data from different databases are presented as a single database, the consistency cannot be guaranteed if no appropriate interdatabase dependencies are declared.

As the figure illustrates, it was considered that the user may access multiple databases in two ways:

- directly at the multidatabase level, using the functions of the multidatabase language,
- through an external view, using either a multidatabase language or a database language, if the schema defines a single database.

In particular, it was proposed that at the current stage of database technology, there is a common data model at the multidatabase layer. Furthermore, it was proposed to use the relational model for this purpose, in the sense that each database appears at the multidatabase layer with local relational capabilities. For multidatabase manipulations, it appeared useful to consider additional capabilities at both data definition and data manipulation levels. Other models, DAPLEX in particular, studied at that time by the MULTIBASE project, seemed less attractive for the common model, as no database system supported them and they were not yet sufficiently understood (view manipulation in particular).

The analysis showed that the implementation of basic multidatabase capabilities should be simple. It was therefore postulated in [19] to evolve the design of database system towards multidatabase systems, whether the systems were intended to be monosite or distributed. The postulate was especially directed towards the relational systems, as they seemed the dominating technology for the 80s.

3.3 Related methodologies

3.3.1 Distributed databases and systems. A distributed database (DDB) is a database transparently implemented on several sites, instead of a single one. This widely known concept differs thus from that of a multidatabase and keeps by its nature the drawbacks inherent to the database approach. The multidatabase approach carefully distinguishes further the notion of a multidatabase system and of a distributed system. The concept of MBS is intended as a new general type of database system applicable to both cases: of all databases at the same site and of databases at different sites. It requires functions for the distributed management only in the latter case. The notion of site and of database were carefully distinguished, unlike in particular, in System R and in System R*. A site is a distinct network node supporting a DBS, that belongs to the physical level. A database is a logical model of a universe, bearing in particular a semantically meaningful name. For instance it could be a database AIR-FRANCE at site GCAM. A site may support several databases, like for instance MRDS or Ingres systems. If an MBS is distributed, it is assumed to provide location transparency. This means that the user manipulates the databases as if they were all at a single site.

It should be noted however that while the distinction between a distributed database and a multidatabase used to be strict in research prototypes and the theory, it is now largely disappearing in practice. Most of the commercial systems claiming to be distributed database systems are in fact the distributed

multidatabase systems as we will show soon. The only exception is the Tandem NonStop system.

3.3.2 Federated databases The report [13] proposed the notion of a federated database that was a loosely coupled set of its components. This principle was then extended to the notion of federated databases which were a federation of loosely coupled databases without a global schema [14]. The main principles for a federation constitution were as follows:

- for cooperation, each database presents a schema called an export schema. This schema is either the actual conceptual schema or a derived schema hiding the private data. These data, whose schema is called a private schema, are all those in the local database.
- data to be manipulated by a user are defined by an import schema. This schema may in particular group data from several export schemas.
- there are mechanisms called derivation operators to produce the import schema. There is also a mechanism for negotiation between databases along a dedicated protocol like that in [15] when they wish to cooperate.
- each federation has a single federal dictionary, which is a distinguished component whose information province is the federation itself.

The reference architectures proposed by both approaches are very close. This is not a coincidence, as [14] relies in particular on the multidatabase approach (see its references) and the multidatabase approach is inspired by ideas in [13]. An import schema is an external schema. A private schema is either the internal logical schema or the conceptual schema at the multidatabase level. An export schema may be considered equivalent to a conceptual schema at the multidatabase layer. However it does not seem to be specified in the federated architecture whether the user may manipulate the export schemas directly, separately or jointly.

In contrast, the federated architecture as defined in [14] does not seem to have the concept of interdatabase dependencies between the export schemas. There is nevertheless the concept of object equality functions that seems largely equivalent to that of equivalence dependencies in the multidatabase architecture, except the functions are in import schemas. Anyhow, one may easily add interdatabase dependencies to the architecture, as in [15] for instance.

Conversely, the multidatabase architecture does not assume as a basic feature of a multidatabase, a single dictionary that would be an equivalent of the federal dictionary. By the same token, it does not consider as a general feature of an MBS, the capabilities for inter-DBS negotiation. Apart from these aspects, the differences between the methodologies are only in the terms used

for similar concepts, and in the aspects of multiple databases management put forward as key ramifications of the principle of the absence of a global schema. If these differences are neglected, then both methodologies are equivalent. This is in fact the case for the popular usage of these methodologies.

The key words for the federated approach are indeed autonomy plus cooperation in interdatabase sharing. The multidatabase approach also has these goals, though it specifically stresses the concept of multidatabase manipulations. A multidatabase language is assumed to be the minimal tool for the existence of a non trivial federation. A multidatabase is a federation of databases, coupled most loosely through the sole existence of the multidatabase language and more and more strongly with the increase of declarations of the interdatabase dependencies. A conceptual schema at the multidatabase layer (federative layer) may be termed an export schema, as in particular it has no dedicated name in [19]. The federal dictionary and the negotiation may be among the functions supposed secondary for an MBS in the multidatabase approach, depending on the system type or the implementation issues. The single dictionary is probably best choice for a centralized MBS, while negotiation protocols are probably necessary in the open MBSs.

3.3.3 Other methodologies While the above approaches are relatively clearly defined, one may also find in the literature other terms and concepts which are usually rather loosely defined. They frequently overlap with the above terminology or even use the same terms with different meanings. See [23] for deeper discussion of this problem.

3.4 Functions of a multidatabase language

Unlike data in a single database, those at the multidatabase layer will be visibly in different databases. Data from autonomous databases will further be usually mutually not integrated, as the same universe will be modelled differently (ex. different restaurant guides or different political parties, see TELETEL). A multidatabase language should allow such data to be managed in a non-procedural way. In particular, it should be possible for formulations of multidatabase queries to remain unaffected by changes to the local schemas. Such changes will occur under the authority of local administrators and will be frequent in the presence of many databases. It would be cumbersome if they usually made a formulation of a multidatabase query obsolete. One may speak in this context about the overall goals of openness of the multidatabase language or about the degree of query reusability.

The analysis showed that all these goals require on the one hand new functions for a cooperative definition of data at the data definition level. On the

other hand, they require new functions at the manipulation level, for non-procedural manipulation of data located in different databases. These data may in particular be replicated and should usually differ with respect to names, structures or values despite a similar purpose. The duplication may have the semantic meaning. For instance, two different recommendations of a restaurant are usually more meaningful than a single one.

Database languages lacked such functions, as they were designed for a single integrated database [8]. As Ex. 1 points out, the relational model assumes all the suppliers in the same S table, and not in several tables, especially in distinct databases, as in Ex. 3 (see also the example in [21]). If suppliers are split into many tables, even in the same database, the model loses its non-procedurality. For instance, the query "select all suppliers" would require as many SQL SELECT statements as there are tables.

The basic new characteristic of multidatabase languages appeared to be the possibility of using the logical database names in the queries, especially to qualify relations in different databases to resolve name conflicts. The reason for the absence of this feature in classical systems does not seem technical, but rather philosophical, as the corresponding implementation is easy. It is probably a blind application of the classical methodology considering that interrelated data are all in the same database and so there is no need for a common manipulation of different databases. Examples in [19] - [21] show it may be false for even immediate real-life needs.

To deal with further needs, several functions were found through the experimental design of the MRDSM system [20]. While some of these functions were intended as general notions, others were specific to the relational data. Their detailed analysis and implementation, was carried out at INRIA within the SESAME project, and aimed at demonstrating the feasibility and the high utility of relational multidatabase systems. These functions are basically as follows:

- the definition and alteration of multidatabases,
- cooperative data definition: single statement creation (alteration, drop,..) of a relation in several databases, import of data definition, etc.
- classical retrievals and updates of relations, being however in different databases, called elementary multidatabase queries in the MRDSM terminology,
- so-called multiple queries, performing relational operations on sets of possibly heterogeneous tables. For instance a single statement selection from a set of Supplier tables in different departmental databases, each table being to some extent particularized for the department needs.

- possibility of multiple identification of data objects bearing the same name, to deal with data duplication and fragmentation (the multiple identifiers in [19]).
- possibility of dynamic unification of heterogeneous names of data objects to deal with name heterogeneity (the semantic variables in [20] and column labels in [21]).
- implicit joins for queries to databases with similar data, but different decomposition into relations, [21]).
- dynamic attributes, for ad-hoc transforms of heterogeneous data values to a user defined basis.
- in particular, the capability to update the dynamic attributes.
- various new built-in functions. For instance, for transformation of data names into data values subject to relational operations (names in one database may correspond to a data value in another).
- view definition, using the (multidatabase) query modification technique
- multidatabase external schema definition (called virtual database in [21]).
- interdatabase queries for data flow between databases.
- auxiliary objects like manipulation dependencies, equivalence dependencies and procedures (transactions, stored queries,...) [21].

Some of these functions required extending the expressive power of database languages. Others concerned only the implementation level (ex. implicit joins; dynamic attribute updates). MRDSM showed that these functions are feasible. For a while, they were exclusive to MRDSM. Starting from 1987, several now characterize commercial systems and industrial prototypes, which we will discuss now.

4. Commercial Systems and Industrial Prototypes

4.1 Commercial systems

Up to 1987, the work on multidatabase systems had been theoretical and on a few research prototypes. It was therefore still a matter of discussion whether operational multidatabase systems would ever be constructed. The year 1987 is important in this respect, as the first commercial systems appeared. They are Sybase, Empress V2 and Ingres/Star. There is now also an operational multidatabase version of Oracle. These systems appear to be major achievements, destined for widespread and durable use. We will now present their main features, using the MRDSM functions as the framework.

Sybase

This system is designed by Sybase Inc. in Berkeley, California. It is a high performance relational system, currently available on SUN workstations, Vax computers and Pyramid. The implementation on the SUN may be entirely on one machine or it may consist of the front-end software on one machine and of the server software on another. Several front-ends may share a server and a front end may access several servers. This does not mean however that Sybase is a distributed system. The distributed version should be released in mid-88.

Sybase language is an extension of SQL, called Transac-SQL. Also, one may use a more user friendly interface called Visual Query Language (VQL). Transac-SQL and VQL are multidatabase languages, the first on the market, as far as we know. They have several interesting features:

- the user may qualify the relation name, let it be T, with the database name, let it be B using the form B.T . Thus one may formulate elementary multidatabase queries. There is however currently the limitation that the databases have to be at the same server. Theoretically, up to 32 K databases may be used simultaneously.
- the user may define multidatabase views, but not virtual databases.
- the queries may include implicit joins. Unlike in MRDSM, they are however limited to relations with a single connection through primary or foreign keys.
- the user may formulate interdatabase queries using multidatabase INSERT and UPDATE statements. The latter statement then takes values in a table and puts them accordingly into a target table. These statements map column names only by order of their enumeration in the SELECT clause, while MRDSM also allows columns to be mapped by name.
- the user may define interdatabase manipulation dependencies. Thus a manipulation of one database, may trigger that of another. In the current version, unlike in the earlier one, the dependencies may be transitive i.e. fire one another. They may be defined by independent users. The length of the chain is however arbitrarily limited to 8 elements, to avoid cycles.
- in the distributed version, the language will allow multiple queries to be formulated.

It is also interesting to note the differences in the user interface with respect to these functions, compared to MRDSM and MSQL:

- the user opens explicitly only one database at a time, through USE <database name> statement. This database constitutes the default scope for table and column names. All other databases remain however, avail-

able to the user, provided he has the access rights. The access to a database is triggered by the use of its name as the prefix. In contrast, MRDSM allows the user to open explicitly several databases and does not allow other databases to be used. The database name is then required as the prefix only if table names conflict.

- MRDSM had no interdatabase UPDATE. This feature of Sybase inspired the corresponding one of MSQL.
- the multiple queries will most likely be generated through the new statement FOR EACH <table names> <elementary query>. This is somewhat more procedural than the use of multiple identifiers or semantic variables in MRDSM. It also makes the query formulation less open to the local autonomy. For instance, if a new database using the same table name and pertinent to the query intention enters the federation, then the Sybase statement has to be modified, while the MSQL statement may remain valid.

As may be seen, Sybase puts into operational practice many concepts of the multidatabase approach. It is an important system that should be widely used and was indeed selected by Microsoft to become the Microsoft system for IBM-PS2, replying to OS2/DB of IBM. It was also selected by Apple for Mac SE and Mac-2 and by Ashton-Tate to replace the famous Dbase. If these plans finalize, the most frequently used systems for database management will be multidatabase systems, as was postulated in [19]. This will be the case not only for the distributed databases, but also the monosite (physically centralized) environment, as was also conjectured in the same paper.

Empress V2

This system is made by Rhodius Inc, in Toronto, Canada. The version described below is V2, following the "classical" version 1, installed in a number of countries. Unlike Sybase, Empress V2 is a distributed system that runs on a number of computers over the Ethernet network: Sun, Vax, Apollo, IBM-PC/PS,... Currently, however it does not allow different computers to mix under the same system. It uses a multidatabase extension of SQL that is currently as follows:

- table names in a query may be prefixed with database names. The database names may themselves be further prefixed by multidatabase names that are ultimately the site names.
- several databases may be open simultaneously.
- the user may define multidatabase views and virtual databases. Both views and virtual databases may be distributed. A virtual database is

manipulated as a single actual one, with location transparency, except for some updates.
- Empress V2 supports distributed updates using two phase locking and two phase commitment.

Empress V2 has multidatabase features that Sybase has not and vice versa. The possibility of using multidatabase names, in particular allows the resolution of name conflict between database names. Note also that Empress V2 is already a distributed multidatabase system, unlike Sybase.

An auxiliary from our point of view, but interesting feature of Empress V2 is that it supports multimedia data. These data may be declared as a particular "bulk" column of a table. They may then be interpreted as text, image or voice data. This feature is an opening towards future multidatabase and multimedia systems and towards the interoperability with information systems other than DBSs.

Distributed Ingres

Distributed Ingres, also called Ingres/Star, is a software layer to Ingres systems and, in the future, to other types of DBSs, supporting locally an SQL interface. The component providing this interface, for instance to IMS, is called "gateway". It corresponds to the Internal Logical Schema in our multidatabase architecture in Fig. 1.

Documents on Ingres/Star say it differs from traditional distributed DBSs, by its "non monolithic" architecture. The analysis of the system principles shows that this term designates the absence of the global schema. The architecture of Ingres/Star is that of Fig. 1. However, there are currently no interdatabase dependencies. Thus, the consistency of replicated data cannot be guaranteed, unlike in Sybase. However, it has no importance for the current version, as updates through external (import schemas) are not supported, with the exception discussed below.

The main features of Ingres/Star, from the point of view of this survey, are as follows:

- the system allows the definition of any number of the external multidatabase schemas over subcollections of SQL databases, currently only Ingres databases. The virtual database defined by this schema is called a distributed database (DDB) and its elements are called links. Once the DDB is created, it is used as an actual Ingres database, except for update limitations. The DDBs may in particular share an actual table or database.
- In fact, if a DDB creation is requested over n databases, then it is created over n + 1 databases. The latter database is a hidden actual database created at the node of the DDB schema definition and simultaneously

with it. This database is named upon the DDB and allows a DDB user to transparently invoke the CREATE TABLE statement. This statement could not work otherwise, as any table has to be in an actual database, while the user cannot indicate in SQL where it should be. These tables may be updated, altered etc.

- the system does not allow the user to directly formulate multidatabase queries to actual databases or, more precisely, to their export schemas. The reason for this seems mainly implementation dependent, namely the necessity of a dictionary entry, created when a link is declared. The only way to formulate an ad-hoc query is to define a DDB whose links are the addressed tables and formulate the query to the links. The links may be declared temporary in which case the DDB is automatically dropped. Otherwise, the user must drop the DDB himself or keep it for further needs. In both cases, the additional manipulations required clearly make Ingres/Star less flexible for ad-hoc multidatabase queries than Sybase and Empress. In addition there is a danger of system pollution with DDBs and the underlying hidden actual databases, created for a particular query and then forgotten.

Oracle V5

In its new version V5.1.17, Oracle also became an MBS (although the corresponding capabilities were announced for V5 in general, we could see them working only in this release). It allows the creation of several databases at the same site and the formulation of elementary multidatabase queries. The Oracle multidatabase language is termed SQL*PLUS. Unlike in Sybase or Empress, the database name does not prefix the table name, but postfixes it, after the character '@'. The user has also particular statements defining aliases for table names and for database names. The former capability allows the resolution of the name conflict, avoiding the use of the database name. The latter, called database links, should not be confused with the different meaning of this term in Ingres/Star.

The language offers also statements for interdatabase queries unknown to other commercial systems and largely similar to the corresponding ones in MRDSM. All the multidatabase manipulations are moreover available for distributed databases, through the distributed database management component SQL*STAR. It is likely that the latter will be permanently included in Oracle which means that the concept of centralized and monodatabase Oracle will disappear. However, the distributed updates are not yet available, like in Ingres/Star V1.

4.2 Industrial prototypes

We designate in this way prototypes that are likely to give rise to operational systems in the near future. Work on two such prototypes is in progress.

Mermaid

This system is being developed in the System Development Group of UNISYS [29] and [30]. While it was initially intended as a classical distributed DBS, it is now evolving towards the federated architecture. Its overall features now resemble those of Ingres/Star and so will not be discussed here. However, there are numerous differences at the implementation level. In particular, Mermaid uses an original pivot language designed for easy translation towards heterogeneous relational languages.

Calida

This system is currently under development in GTE Research Laboratories. The operational version is destined for the management of numerous databases of GTE, mostly the relational ones. The 1st operational implementation should be in 1988 in California, where, ultimately, there are about a hundred large bases to be connected. If the experimentation is satisfactory, Calida will be extended to other states in the U.S. The main features of the system, largely unpublished as yet, are as follows:

- Calida makes it possible to access relational and Codasyl-like databases. The internal logical schema and the corresponding manipulation are generated through the original rule processing system. This system provides a particularly flexible interface to data model heterogeneous databases.
- the multidatabase manipulation language is not the SQL, but a proprietary relational language called DELPHI. This language is used as a final language for the sophisticated user and as an intermediate language for a natural language for interface. DELPHI allows the formulation of elementary multidatabase queries, including the updates, where database names may be used as prefixes to solve name conflict. The query decomposition is carefully optimized, using field statistics gathered by the system. Calida moreover allows the definition of external schemas and of views through the usual query modification technique.
- the system supports the implicit joins that, in particular, may concern columns in tables in different databases. This feature existed only in MRDSM, as it requires the definition of equivalencies between domains or tables of different databases. In the GTE system, the corresponding equivalence dependencies are stored in a so-called global dictionary. The algorithm for the query completion is similar to that of MRDSM in that it

searches for a minimal spanning tree over the intersection of the non- connected query graph and the connected database graph whose nodes are relations and edges are connections through keys. However, the algorithm is limited to the case of a single connection between two relations (acyclic graphs). If there are multiple connections, the user is asked to make a choice, unlike in MRDSM. One advantage is a fast recursive algorithm for the spanning tree edges computation.

DQS/Multistar

The relational model also seems an appropriate common model for access to non-relational databases. The Distributed Query System (DQS) prototype is a multidatabase system developed for investigation of the corresponding issues [2]. The system should soon lead to a commercial version called Multistar. It allows multidatabase retrievals from IMS/VS, IDMS, ADABAS and RODAN databases, as well as from standard VSAM files. The objects of these databases are presented as relations through dedicated mapping commands. The retrievals are formulated in SQL over so- called global schema in DQS terminology. However the DQS global schema is in fact an import schema, as several different schemas may be defined which may be partial, and they may overlap. These schemas may also include views, in the SQL sense of this term. Views are the principal data abstraction mechanism for aggregations and generalizations in DQS.

DQS has several interesting features, especially its algorithm for SQL query decomposition. Views are dealt with using the query modification technique. The query is represented as a tree subject to the algebraic transformations to reduce intermediate relations. A heuristic algorithm is also used to produce the query tree optimized with respect to data movements between the sites. For execution, this tree is finally transformed to a Petri Condition-Event net.

Interoperable database system

This title refers to the name of a large national project in Japan [17]. As with the 5-th Generation Project, this one is backed by MITI and involves all major Japanese computer manufacturers grouped into an organization named INTAP. The project budget is around 120 MS over five years. The project goal is to build the software and hardware which would permit distributed database systems to exchange data and to be manipulable together. The databases are not in general integrated under a global schema, they are only interoperable. External multidatabase schemas may be created and it is assumed that there may be several over the same collection of databases. The ultimate goal of the project is further to make DBSs interoperable also with other types of information systems. For both multidatabase interoperability

and interface to other systems the vehicle should be the ISO/OSI Open System Architecture and protocols. The description of the current stage of the project is in [17].

4.3 Impact on standardization

ISO has currently issued the SQL standard of the so-called level 1. This standard is the kernel of the classical SQL, without the concept of the database name in the statements, and thus inappropriate for multidatabase manipulations. Given the new extensions to SQL that appeared in the commercial systems, it was proposed to ISO to include them into the so-called level 2 standard. Work on the level 2 standard is currently on-going.

In this context one should note the proposals of Chr. Date for improvements to SQL [6]. One of the proposals is to provide names for the results of SQL value expressions (p. 35). This will give rise to dynamic attributes in SQL.

The notion of a multidatabase system, as distinct from the classical distributed database system concept was on the other hand recognized by ISO in its work on Remote Database Access Protocol (RDA) [28]. MBSs are assumed more common. The RDA protocol is intended mainly for these systems though in its current version it lacks many features.

5. Research Issues

The number of investigations of multidatabase (federated) systems design has greatly increased recently. Some are reported in the references to this survey, in particular in [7], and [10]. They have led to interesting results and new general concepts, some of which we report below. The classical ideas on the transparency of data sharing by other users, on transaction duration, atomicity and isolation, on the possibility of transparent roll- back proved too simplistic for the new needs.

5.1 Transaction management and concurrency control

The autonomy and lack of integration between databases put new requirements on the concurrency control. The classical two phase locking and commitment will be increasingly inappropriate. In particular, a multidatabase operation (a multidatabase manipulation language statement or a sequence of such statements) may not need to hold on to all of its resources until it completes. In this case, it is useful either to extend the classical notion of a transaction, or to create new concepts. The following ones seem particularly promising [1], as they are close to real life procedures in many organizations:

- a global procedure is a procedure initiated at some node that can request other nodes to execute procedures (usually transactions). At each node the global procedures are managed by a global procedure manager (GPM), interfacing for local operations the local transaction manager (LTM). Because of local autonomy, the GPM has no control over the local concurrency control and transaction processing. In particular, once a transaction has been run by an LTM on behalf of some global transaction, it cannot be undone or rolled-back by the GPM. The only recourse of the GPM is to request the execution of another transaction, called compensating transaction. Thus, a global procedure cannot be atomic in the transactional sense.
- For many applications, it is not necessary to serialize global procedures. For instance, consider a multiple query to several "Scheduled Meetings" personal databases proposing a meeting for a given date, provided all persons are available. As the serial consistency for the entire corresponding procedure is not required, this procedure may be broken up into a number of transactions which can be interleaved in any way with other transactions. Such a procedure is a saga and as the example shows, many operations in the multidatabase environments may be run as sagas. It may be shown that by running global procedures as sagas, instead of ensuring their total serializability with other global procedures, substantial performance benefits may be achieved.
- Sagas are not however always appropriate, especially if a multidatabase manipulation uses an aggregate function. In this case, two phase locking may be used. However, there may be a substantial performance problem when many nodes are involved, as no lock should be released, until the last lock is requested. Even worse, the corresponding delay may be greater than the local time-out of an LTM which will then release the lock, believing a dead-lock. To avoid these problems, new locking protocols providing a higher degree of concurrency are needed. Also, it may be necessary to combine locking with other protocols and methods for the recovery.
- It is possible, on the one hand, to then use the altruistic locking [1]. On the other hand, the same order of subtransactions at each site may be maintained, detecting and recovering from global deadlocks in a simple way [3]. Furthermore, providing the knowledge of the serial ordering at each site, one may attempt to group databases into sets called superdatabases inside which different concurrency control and crash recovery methods may be used together [27]. Finally, under similar assumptions, the optimistic control [11] may be used.

This synthesis is by no means complete. The concepts discussed have ramifications either discussed in the corresponding papers or which remain to

be studied. However, it already appears that the multidatabase systems put new constraints on the transaction processing and trigger interesting extensions [9], [31]. These extensions are closer to real life procedures in human organization than the classical rather idealized concepts and algorithms. It is therefore likely that the corresponding studies will largely widen their scope.

5.2 Access control

There are many aspects of access control in a multidatabase system. One problem is that the autonomous system may require different login procedures and may assign different passwords to the same user. One of the first solutions to this problem was investigated in the Mermaid prototype, developed at UNISYS. This prototype provides the SQL interface to federated databases through import schemas. See [29] for more details.

For the management of the passwords and of login procedures, the system uses three approaches. If security is not critical, the system presents itself as the user to the accessed databases with its own unique name and password. Otherwise, various passwords of the actual user are stored in the data dictionary-directory (DD/D) and retrieved transparently when needed. Finally, if security is crucial, the user is asked for different passwords, when opening the corresponding databases.

This pragmatic mechanism requires an efficient coding of the passwords stored in the DD/D. As the DD/D may be replicated on different nodes, the encryption/decryption algorithm has to be portable and independent of various eventual local tools for this purpose. Mermaid uses a simulated Enigma machine. This tool has the advantage of several tunable parameters, so that even though the general algorithm is well known, the implementation particulars keep the passwords tolerably secure.

5.3 Data definition

If the databases are managed by more than one system, some conventions are needed for the exchange of data and of manipulation statements. Autonomous systems may in particular use different data formats, encoding or precisions of similar data. To make the relational operations such as a join feasible, data must nevertheless be dynamically converted to a common representation. Similar needs exist with respect to the query execution, since the query expression may be considered as defining the data type the user wishes, (to be bound therefore transparently to or from the actual schemas, regardless of differences in naming, structures and value types). The brutal approach such as a list of all possible conversions is excluded, as in a large open system, their number may be very large. The usual binding of names by exact and explicit designation is also too limited in the present systems and would require

global schemas with an endless number of generalizations, aggregations etc. A better approach seems to be as follows:

- the participating databases are self-describing [24]. This means that they provide enough information about both data and their formats to allow them to be interpreted or converted with only some elementary knowledge. The cooperating systems have the corresponding components, detailed in [24]. For name binding, the systems share auxiliary knowledge like a general purpose thesaurus and a sophisticated linguistic processing, allowing the name matching to be inferred dynamically and implicitly. The current systems are not usually self-describing and do not use such tools. In particular, as far as we know, no relational DBS provides the units or the precision of column values.
- the manipulations are standardized into some protocols common to all participating DBSs and fitting the Open System Architecture. The LINDA (Loosely INtegrated DAtabases) project at the Technical Research Center (Finland) investigates the corresponding issues [1]. The protocols are implemented over the TCP transport services, including Remote Database Access Protocol. The prototype is intended for heterogeneous databases on heterogeneous DBSs, but only the relational systems Empress V1 and Informix are addressed at the current stage. For data manipulation, one uses SQL extended with some multidatabase manipulation functions. The hardware environment are MicroVAx and Sun 3 stations.

5.4 Update dependencies management

The manipulation dependencies usually forward updates. Limited languages for the corresponding definitions were proposed for MRDSM and the Sybase system, presented above. However, a more general proposal was needed. The work in [25] addresses this problem. Update dependencies are described using a predicative language which is easy to implement using Prolog. Several examples show the application of the proposed concepts, especially to a transaction processing. It is also shown that the concept is useful for classical problems like view update and update anomalies in non normalized databases. Furthermore, it is shown that the system using the proposed approach should be capable of combining independently defined dependencies, though details of this interesting issue remain to be investigated.

5.5 Updates of transformed values

In the multidatabase environment, heterogeneous data values must frequently be converted or combined to a value type defined by the user. In order to do this MRDSM provides the concept of a dynamic attribute, let it be D ; D = F

(A) ; where F is a transform of values of some actual attributes A. Usually F is an arithmetical formula, like a value expression in SQL. Dynamic attributes are dynamically defined by the user in queries. Virtual attributes are intended for the same purpose in view definitions. While the concept of a virtual attribute has been known for sometime, MRDSM is the only system to provide dynamic attributes.

The transform F is defined for retrievals. However, one may also need to update D. As far as we know, updates of virtual attributes were considered impossible and no DBS had the corresponding capability. MRDSM provides a solution for dynamic attributes that naturally applies to view update as well. Its principle is as follows [22]:

- the formula F has to be defined by a computable function that is transformed to an equation whose roots are new values of A. The system passes F to MACSYMA which is a large symbolic calculus system. If MACSYMA finds the symbolic solution, it passes it back to MRDSM. If several solutions exist, MRDSM chooses that matching the values of D and of A before the update.
- Otherwise MACSYMA sends factorized formulas back to MRDSM. MRDSM then applies a numerical method (the Bairstow method).

For F that is not a computable function, MRDSM in general requires the user to provide the transform F': D - A.

Note that the above cooperation between MRDSM and MACSYMA is an example of interoperability between information systems of different types that as far as we know, had not been previously integrated together. Note also that although MACSYMA is a mainframe system, the proposed approach applies to systems entirely on workstations as well. Equation solvers are indeed becoming available on workstations and are now cheap (TK!, Eureka etc.). Their possibilities are more limited, but are sufficient for many applications and are extending rapidly with new releases.

5.6 Query processing efficiency

Multidatabase languages introduced new capabilities whose optimization is an open research area. One may cite for example the following problems:

- Traditional work on the distributed query decomposition considered that the the necessary capabilities are available within each participating system. The multidatabase query processing may however need sophisticated services like a thesaurus, an equation solver, rule processing, outer-joins etc. It is unreasonable to consider that these services are available universally. There is therefore a need for the self-description of a database sys-

tem not only with respect to data schemas, but also with respect to operational capabilities. The object oriented approach seems useful for this purpose. The query decomposer has to take into account the availability of capabilities as additional constraints.

- Different types of formulas F for value conversion lead to different more efficient algorithms. Furthermore, the most efficient algorithm for a few tuples, may be not the optimal one for many tuples. For instance, for F in a polynomial form, the Horner algorithm is the optimal one for a few tuples, but Motzkin and Knuth algorithms are faster for many tuples.
- The multiple queries are at present evaluated in MRDSM as sets of queries resulting from the substitutions of the unique identifiers to the multiple ones and to semantic variables. The resulting queries may have common subexpressions. Factorization of these subexpressions may be useful, as duplication of retrievals or some intersite transfers may be avoided.
- Similar problems may occur for interdatabase queries, where data selected from some source database should be dispatched to several databases. Basically, such an interdatabase query is a set of subqueries each one performing a selection from the source database(s) and some kind of insertion into one target database. Target tables may in particular differ to some extent. The factorization may avoid replicated selections between subqueries and may speed up other selections. It may indeed use as the source, a small temporary relation produced by the selection expression of another subquery, instead of large base relations.

6. Conclusion

The multidatabase approach was intended as a new methodology for the design of database systems, in particular the distributed ones. The general idea was that such systems should be able to manage collections of autonomous databases without a global schema and control. The methodology proposed new basic concepts, a general architecture and several new capabilities for relational languages. The proposed principles progressively gave rise to developments at research level. They were recently further applied to industrial prototypes, within major new commercial systems and to new domains for database systems [23].

The existence of major commercial systems shows that the proposed principles were an appropriate framework. The perspectives of worldwide usage of these systems show that multidatabase (federated, interoperable,..) systems will be among popular software tools. Most probably, database systems will therefore systematically become multidatabase systems as was postulated. The utopian notion of a database being a single, possibly "very large", data recipient is being abandoned.

The functions for the multidatabase manipulations already present in the commercial systems are nevertheless only a rather limited subset of those investigated at research level. It seems most likely that the new assumptions about the lack of global knowledge of all data and about user autonomy will influence the design of future systems much more deeply. New principles will emerge at all levels, starting from logical data definition and manipulation functions and going down to concurrency and transaction management principles, as well as to the physical implementation and distributed execution. They will apply further to new types of information bases, in particular knowledge bases. Finally, they will provide interoperability with respect to other types of information management systems, broadly named services in the present terminology. They promise interesting perspectives for both users and researchers running after exciting issues.

Acknowledgement
The author is grateful to Mr. Richard JAMES for help with editing.

References

1. R. Alonso, H. Garcia-Molina, K. Salem, Concurrency Control and Recovery for Global Procedures in Federated Database Systems. IEEE Data Engineering, (Sep. 1987), 10, 3, 5-11.
2. Bellcastro, E & all. DQS-Distributed Query System. (Sept. 1987), CRAI, Italy, 21. To appear also in Proc. of Int. Conf. on Extending Database Technology, Springer Verlag, 1988.
3. Y. Breitbart, A. Silberschatz, G. Thompson, An Update Mechanism for Multidatabase Systems. IEEE Data Engineering, (Sep. 1987), 10, 3, 12-18.
4. S. Ceri, B. Pernici, G. Wiederhold, Distributed Database Design Methodologies. Proceedings of the IEEE, (May 1987), 533- 546.
5. B. Czejdo, M. Rusinkiewicz, D. Embley, A Unified Approach to Schema Integration and Query Formulation in Federated Databases. Res. Rep. University of Houston, 1987, 25.
6. C. J. Date, A Critique of the SQL Database Language. SIGMOD Record, 1984 , 8-54.
7. Special Issue on Distributed Database Systems. Proceedings of the IEEE, (May 1987), 532-735.
8. M. S. Deen, R. R. Amin, M. C. Taylor. Data Integration in Distributed Databases. IEEE Trans. on Soft. Eng., 13, 7, (July 1987), 860-864.
9. F. Eliassen, J. Veijalainen, Language Support of Multi- database Transactions in a Cooperative, Autonomous Environment. IEEE Region 10 Conf., Seoul, (Aug. 1987).
10. Special Issue on Federated Database Systems. IEEE Data Engineering, (Sep. 1987), 10, 3, 64.
11. A. Elmagarmid, Y. Leu, An Optimistic Concurrency Control Algorithm for Heterogeneous Distributed Database Systems. IEEE Data Engineering, (Sep. 1987), 10, 3, 26-32.
12. B. Gash, U. Kelter, H. Kopfer, H. Weber, Reference Model for the Integration of Tools in the "EUREKA Software Factory". ACM-IEEE Fall Joint Comp. Conf. (Oct. 1987), 183-190.

13. M. Hammer, D. McLeod, On database management system architecture. MIT Lab. for Comp. Sc. MIT/LCS/TM-141, (Oct 1979), 35.
14. D. Heimbigner, D. McLeod, A Federated Architecture for Information Management. ACM Trans. on Office Information Systems. (July 1985), 3, 3, 253-278.
15. D. Heimbigner, A Federated System for Software Management. IEEE Data Engineering, (Sep.1987), 10, 3, 39-45.
16. C. Hewitt, P. De Jong, Open Systems. in On Conceptual Modeling. Springer Verlag, 1985, 147-164.
17. Interoperable Database System. 1st International Symposium. INTAP, (May 1987), 167.
18. E. Kuhn, Th. Ludwig. VIP-MDBS: A logic multidatabase System. Europ.Teleinf. Conf. EUTECO - 88, Wien, (April 88).
19. W. Litwin et al. SIRIUS Systems for Distributed Data Management. Ed. H. J. Schneider. North-Holland, 1982, 311-366.
20. W. Litwin, A. Abdellatif, Multidatabase Interoperability. IEEE Computer, (Dec. 1986), 19, 12, 10-18.
21. W. Litwin, et al. MSQL: a Multidatabase Language. INRIA Res. Rep. 695, (June 1987), 41. To appear in Inf. Science - An International Journal, Special Issue on Databases.
22. W. Litwin, Ph. Vigier, New Functions for Dynamic Attributes in the Multidatabase System MRDSM. HLSUA Forum XLV, New Orleans, (Oct. 1987), 467-475.
23. W. Litwin, A. Zeroual. Advances in Multidatabase Systems. European Conference on Teleinformatics EUTECO-88, Wien. North Holland (publ.).
24. L. Mark, N. Roussopoulos, Information Interchange between Self-Describing Databases. IEEE Data Engineering, (Sep. 1987), 10, 3, 46-52.
25. L. Mark, N. Roussopoulos, Operational Specifications of Update Dependencies. SRC Res. Rep., Univ. of Maryland, (Feb. 1987), 44
26. A. Motro, Superviews: Virtual Integration of Multiple Databases. IEEE Trans. on Soft. Eng., 13, 7, (July 1987), 785- 798.
27. C. Pu, Superdatabases: Transactions Across Database Boundaries. IEEE Data Engineering, (Sep. 1987), 10, 3, 19-25
28. Remote Database Access Protocol. 2-nd Working Draft. ISO/TC 97/SC 21/WG 3, 1987.
29. M. Templeton, E. Lund, P. Ward, Pragmatics of Access Control in Mermaid. IEEE Data Engineering, (Sep. 1987), 10, 3, 33-38.
30. M. Templeton, et al. Mermaid: A Front-End to Distributed-Heterogeneous Databases. Proceedings of the IEEE, (May 1987), 695-708.
31. G. Wiederhold, Q., XiaoLei, Modeling Asynchrony in Distributed Databases. 3rd IEEE Conf. on Data Engineering, Los Angeles, (March 1987), 246-250.
32. A. Wolski, LINDA: Overview. Techn. Rep. Techn. Res. Centre of Finland. (May 1987), 19.