

SRC TR 89-12  
UMIACS TR 89-10  
CS TR #2188

**Interoperability of Multiple  
Autonomous Databases**

by

**W. Litwin, L. Mark**

and

**N. Roussopoulos**

# **Interoperability of Multiple Autonomous Databases**

**Witold LITWIN**

**INRIA 78150 Le Chesnay, France<sup>1</sup>**

**Leo MARK, Nick ROUSSOPOULOS**

**Univ. of Maryland, College Park, MD 20742**

## **ABSTRACT**

Database systems were a solution to the problem of shared access to heterogeneous files created by multiple autonomous applications. To make the data usage easier, one proposed to replace the autonomous files by a globally integrated collection of data called a database. The idea was successful to a large extent and there are now many databases distributed over local and long-haul networks and frequently even on the same larger computer. Unavoidably, users now need shared access to multiple autonomous databases. The question arose as to what the corresponding principles should be. Should one reapply the database approach principles one level up or should new methodologies be introduced ?

We show that new methodologies have appeared, defined specifically for the management of multiple autonomous databases. They lead to a new type of systems, called multidatabase systems or federated systems. These systems make databases interoperable, ie manipulable together in a non-procedural way, without global integration. They also preserve the autonomy of each database to satisfy first its own needs.

Systems of that type will be of basic importance, especially for distributed databases and we analyze the corresponding reasons. We also present the methodologies proposed for their design and discuss their relationship. We further show that the evolution towards multidatabase systems is already on the way, as major commercial relational database systems are becoming of this type. We discuss their capabilities and limitations with respect to advanced prototypes. We also show industrial prototypes and the standardization issues. Finally, we present some research issues.

---

<sup>1</sup> Currently with System Research Center, Univ. of Maryland, College Park, MD 20742

1. INTRODUCTION .....	2
2. DATABASE MARKET .....	3
2.1. Classical corporate databases.....	3
2.2. Relational databases on mainframes or minicomputers.....	4
2.3. Relational databases on personal computers.....	4
2.4. Information retrieval databases.....	5
2.5. Videotex databases .....	6
2.6. Synthesis .....	7
3. METHODOLOGIES FOR MANAGEMENT OF MULTIPLE DATABASES.....	8
3.1. Database approach.....	8
3.1.1. Principles.....	8
3.1.2. Motivations .....	9
3.1.3. Limitations.....	10
3.1.4. Synthesis.....	11
3.2. Multidatabase approach.....	11
3.2.1. The idea.....	11
3.2.2. The concept of a multidatabase system .....	12
3.2.3. Reference architecture.....	13
3.2.4. Functions of a multidatabase language.....	14
3.3. Federated databases .....	19
3.4. Related methodologies .....	20
3.4.1. Distributed databases and systems .....	20
3.4.2. Other methodologies.....	21
4. COMMERCIAL SYSTEMS AND INDUSTRIAL PROTOTYPES .....	22
4.1. Commercial systems .....	22
4.1.1. Relational databases.....	22
4.1.2. Information retrieval databases .....	26
4.2. Industrial prototypes .....	27
4.3. Interoperable database system.....	28
4.4. Impact on standardization.....	28
5. SOME RESEARCH ISSUES .....	29
5.1. Transaction management and concurrency control .....	29
5.2. Data definition.....	31
5.2.1. Self-describing database system .....	31
5.2.2. Self-documenting database systems .....	32
5.2.3. Standard format data units .....	33
5.3. Update dependencies.....	33
5.4. Updates of transformed values.....	35
5.5. Query processing .....	36
5.6. Incremental access of multiple database servers .....	37
5.7. Logic multidatabase systems .....	38
5.8. Dynamic Derivation of Personalized Views .....	39
6. CONCLUSION .....	40
REFERENCES.....	40

## 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 heterogeneity, such as differences in field naming, value types and file structures for a similar purpose. In particular it was difficult 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 a *database administrator*. His task was to make the database *integrated* which meant free of duplications and heterogeneity. The database should then be managed under a centralized control by a system called a *database system* (DBS). A DBS 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 methodology should be. Should one reapply the database approach one level up or should new principles and types of systems be introduced ? In the former case, a distributed database system should be designed to manage the distributed databases through a global conceptual schema, making them a logically single integrated database (to physically replace the existing databases by a single one is obviously a utopia). In the latter case, it should provide new functions for management of multiple autonomous databases without a global schema.

Since the problem of the distributed database management has emerged research tried both approaches. Earlier distributed database system prototypes, like SDD-1, POREL or SIRIUS-DELTA, followed the former approach. However, it appeared that the market demand for such system is low, if any. This triggered the effort towards the latter approach. The autonomous databases that can be managed together without being integrated by a global schema were called multidatabases, or federated databases or interoperable databases. The systems able to manage them are generally called *multidatabase* systems or *federated* systems. However, as commercial systems are now systematically evolving in this directions and multiple databases are frequently on different computers, terms like distributed database system or distributed heterogeneous database systems now also designate this type of systems.

Research on multidatabase (federated) systems showed that the database principles are inapplicable to collections of autonomous databases, unlike they were to files. New principles emerged and will modify the design of database systems at all levels. One reason is that autonomous databases introduce heterogeneity. The amount of accessible data is also much larger and the classical requirements on consistency, concurrency control and transaction management have to be relaxed. Systems will need new components for cooperation with other systems. Finally, new perspectives appear for the query processing, as cases inherently hard for a stand-alone system may be efficiently solved when spread out on cooperating systems.

We show that systems specifically designed for the management of multiple databases, especially distributed, will be of basic importance. We analyze at first the current evolution of the database market. We then show the methodologies proposed for the design of systems for multiple databases. We discuss their rationales with respect the motivations and drawbacks of the database approach. We further show that major operational relational database systems evolved towards multidatabase systems. We discuss their corresponding capabilities and also the limitations with respect to advanced prototypes. We also discuss industrial prototypes and the standardization issues. We finally show selected research issues, particularly these related to the self description and the data interchange.

## **2. DATABASE MARKET**

The market provides more and more types of database systems. While the demand for access to data in multiple databases is by now clear and will be general, it will not concern all types of databases evenly.

### **2.1. Classical corporate databases**

One type of databases are these of well known non-relational systems like IMS, IDMS, ADABAS etc. They are popular with larger corporations, but also limited to them. They are usually a few or one per company, though big corporations may have several hundreds. The need for joint access is limited, usually to a few users and inside the corporation. The data manipulation languages are usually navigational ie record-at-the time manipulation statements. The demand for join access through such classical languages seems almost inexistent, despite some research effort in the past. It does not seem technically feasible neither, because of the low level of these languages and difficulty to implement constructs such as a Codasyl set spanning over several databases. The problem looks obsolete by now also because these systems are nowadays or soon be provided in general with a relational interface. This moves the problem of manipulation of multiple classical corporate databases to the class of relational databases.

## **2.2. Relational databases on mainframes or minicomputers**

This type of database systems is becoming increasingly popular. Known examples are DB2 for IBM mainframes, MRDS for Multics, Ingres, Oracle, Informix or Unify for Unix environment and minicomputers, as well as newcomers like Sybase or Tandem Non-Stop SQL. There are also database machines like Britton-Lee or Teradata. The access to this type of databases is also typically limited to a corporation. However, as smaller computers are becoming popular, there is also increasingly more relational databases than the classical ones. They are also more autonomous, being now usually spread out at the department level. As computers in a corporation are by now in general interconnected, there is a growing need for joint access to these databases, stronger than for the classical databases. An additional reason for this need is that these databases use now generally the common language that is SQL. In fact, they share only the kernel of this language. It is well known indeed that dialects differ with respect to syntax, semantics and availability of statements, error codes, aggregate functions etc.

## **2.3. Relational databases on personal computers**

Until last year, most of so-called database systems on personal computers (PCs) have had the capabilities too limited to be a true database systems. In particular, these advertised as relational, in general were not. This includes even such well known systems as DBASE or RBASE. The term "relational" was used essentially for the capability to define links between files, abusively called relations.

This situation is changing sharply. One finds now several relational systems, like INGRES-PC, ORACLE-PC or XDB, to cite a few, whose prices are around some hundreds of dollars. A very important system is also the recently announced IBM OS2/DB version. It provides SQL under the operating system, together with a number of the popular communication protocols. There are also more powerful systems for the servers, especially the SQL Server announced jointly by Sybase, Microsoft and Ashton Tate.

The common characteristic of all these systems is the usage of SQL. They are in process of generating relational databases at any level of a corporation of any size. They also become popular with non-corporate professionals and private users. There are already thousands of relational databases on PCs and this trend will continue towards millions. These databases will constitute the most important segment of the relational market. As PCs will be naturally interconnected, there is the strong need for easy access to multiple databases of this kind as well. The manufacturers got the message and announce all enhancements towards some kind of future distributed database management.

#### 2.4. Information retrieval databases

This type of databases is popular with the database servers, while it barely exist in corporations. Their usage is usually payable, unlike for corporate databases. In the USA, well known examples are Dialog, CompuServe or The Source, to name only a few. In Europe, best known are Inspec and Questel, but each country has many others. Initially, all these services were designed for bibliographic purpose. Now, they provide information on virtually any domain of the life. The server usually carries several databases ; a large one like CompuServe may carry hundreds (450). However, it should be stressed that some information retrieval databases may be considered rather as large files by database people.

Information retrieval databases basically use key words to describe records and for the queries. Unlike for corporate databases, the user frequently does not know exactly the data names (key words) to be used in the query to make the search precise enough. Even more, in presence of a large number of databases, one may even have trouble with the choice of the database to start with. It is therefore common to the servers to use collective, multidatabase names, classifying the databases into categories like finance, travel, education etc. The categories may nest. The user is guided through the categories, until the most appropriate database is reached. The search goes then in general through a sequence of queries, frequently called *search strategy*, progressively narrowing the scope until it fits the user wish.

There are several languages for the information retrieval, usually different from server to server. They usually lead to different forms of a query that however always consists of key words linked through boolean operators. They are also not very user friendly and typically require professionals. These characteristics, restricted the usage of such databases. There were also other reasons such as fixed subscription and monthly fees, different login procedures,...

The information retrieval database administrators observed that users need the access to multiple databases since already several years. To find references of a subject, one usually needs indeed to search a database storing journal papers, as well as this with technical reports,... This gave rise to networks for easier access to multiple servers from the single terminal. The largest effort is the EEC sponsored network Euronet, now called Diane that provides the access to over two thousands of databases (some are however replicated on several hosts). There is also the central service called ECHO that may provide user with the overall information about DIANE and its particular databases. Hosts started also to make internal connections among them to simplify the access procedures. The user connects to one host and acts as if all the databases of other hosts belonged to this one. The host generates remote sessions, login procedures, passwords etc. and takes charge of the whole billing. These usually multiple and cumbersome manipulations become then transparent to the users of these hosts.

On the other hand, the information retrieval community needed a standard language. For this purpose, EEC has sponsored and then adopted a standard language for Diane, called Common Command Set Language (CCS). Currently, all servers have CCS interface, developed usually with EEC help, in addition to their local languages. The CCS is now available also in the USA. Especially, on the important server EasyNet, providing access to over 850 databases and discussed more in Section 4.1.2.

CCS consists of a few standard commands. The basic one is the **FIND** command. Its parameters are key words linked with boolean operators. For instance **FIND KW = Information Retrieval \* Natural Languages** asks for records indexed with both keywords. Another command **COMBINE** allows the user to combine the **FIND** commands previously issued using boolean operators. The command **COMBINE 1/2** for instance, would ask to eliminate from the set of documents found by **FIND** in step 1 of the search strategy, these indexed with key words in **FIND** command used in step 2. The search usually ends up with the **DISPLAY** command with some formatting specifications that presents the selected documents to the user.

## 2.5. Videotex databases

This type of databases is more popular in Europe. It generally results from nationwide effort of Telecommunication administrations (PTTs) of each country. There are now several videotex standards and EEC starts sponsoring the development of gateways. Most used is the French system TELETEL, the first known system is the British PRESTEL. We will now discuss the TELETEL features, these of other systems are similar.

The idea in videotex systems is to massively provide users with a cheap terminal. A typical videotex terminal has 40 characters per line and limited graphic capabilities. The TELETEL terminal is basically distributed for free, as the replacement of the paper directory. It is then used for access to various services, usually connected to the PTT videotex network, but sometimes on stand-alone servers. To access the services on the network, there are a few nationwide numbers : 11 for the nationwide Electronic Phone Directory service, 3614, 3615,... for others. Each number has its own connection cost per minute. It ranges from free for the first two minutes for the Directory, to several dollars per minute for the number connecting to highly specialized databases. The total bill is charged by the PTT with the monthly telephone bill. On the other end, PTT dispatches it to each server. One avoids in this way the problem plaguing the information retrieval databases.

The physical location of the server site and its name is transparent to the user who knows the services and their databases only by logical names, like AIR FRANCE, AMERICAN EXPRESS etc. The data manipulation languages are usually customized and menu driven. The commands bring to the screen *information pages*, in general organized in hierarchies. Advanced systems use

key words, like information retrieval databases that, by the way, start providing the videotex interface as well. No videotex database with the relational interface is known. However, to provide it does not seem a major problem.

In 1988 the number of videotex terminals in France approached 3 M. This mass, the absence of subscription fees to access databases and PTT's help to develop the videotex, largely boosted the offer. The current figure is about 4 K services, new databases appear everyday, as well as some disappear. They address virtually any aspect of the everyday life. Usually there several services addressing the same subject, in general in strong competition. For instance, there are over thirty databases owned by unions or political parties to express their political views (needless to say that they are highly autonomous and usually not consistent mutually). All major banks, newspapers, insurance companies, travel agencies, airlines, etc. also provide access to their databases and compete for customers. There are also multiple databases for buying or selling cars, choosing a good restaurant or movie etc.

In this environment, the need for manipulations spanning multiple databases became obviously strong, as searching databases one by one is time consuming and expensive. However, there is currently no such a possibility and this slows down the usage of TELETEL. As it became progressively hard to know even what are the databases pertinent to a given need, PTTs have created a metadatabase that is the directory to others (MGS). Databases are qualified in the directory by collective names, such as AIRLINES which serve as the entry point for the search and lead to the list of the pertinent databases. However, there is no automatic switching to databases. The user must leave the directory and connect to the databases by himself.

## 2.6. Synthesis

There are presently many systems providing access to collections of databases, particularly information retrieval databases and videotex databases. Many more databases will be created soon, especially the relational ones on shared minicomputers and personal workstations. There is already a need for manipulations spanning multiple databases and this need will strongly grow. These databases may be on the same site, as they may be on different ones. In the latter case, the physical location is usually transparent to the user. The databases are and will be usually autonomous. They often compete for customers, some die other appear almost on daily basis. A centralized control as well as global integration are unlikely.

The information retrieval databases have the general trend towards CCS as the standard languages. Videotex databases use rather dedicated manipulation languages, as they are younger and only start to move to more general ones, CCS especially. The corporate databases in general evolve towards a dialect of SQL. As both languages are based on very different paradigms, the trend for some years to deal from this point of view separately with two worlds.

The servers of information retrieval databases perceived the need for multiple database access earlier than corporations. The reason is the absence of corporate barriers or the demand and support of powerful institutions like EEC. These systems provide already on operational basis some capabilities making easier manipulations of multiple database, that have to be dealt with by any further work. It is clear that the relational databases will face similar problem and may benefit from the information retrieval world experience. On the other hand, the information retrieval world will benefit from the recent and coming work on relational databases. Many technical solutions will be also common to both worlds. This also true for new areas for database applications and the corresponding new types of databases (object oriented, logical,...). There is therefore the need for a general methodology for the management of multiple databases, taking to the account all the needs and constituting the frame for both general and specific solutions.

### 3. METHODOLOGIES FOR MANAGEMENT OF MULTIPLE DATABASES

Earlier research tried to apply the (classical) database approach as the methodology for the management of multiple databases, especially the distributed ones. The idea therefore was to build a conceptual schema, usually called *global (conceptual) schema*, making all the data behaving as if they were a classical database. More recent proposals consider this approach unrealistic for larger collections and not always wanted by users. Two proposal are presently best elaborated which are the *multidatabase approach* [Lit82] and the *federated databases* approach [Hei85]. As they start from the similar criticism of limitation of the database approach, we will first overview this common aspect.

#### 3.1. Database approach

##### 3.1.1. 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 data manipulations. The systems were mostly batch and business oriented. The manipulations were rather short and could be rerun.

The idea in the database design was that the administrator and the system should insulate the user and even the application programmer from the complexity of data sharing in these conditions. 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 at first 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 means 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.

After that, the administrator should define on one hand the *internal schema* of the database. At this level the administrator was supposed to choose data structures providing optimal performance of the database. Furthermore, he should also define the *external schemas* of the database. These schemas should adapt the conceptual schema to a user's needs, if the conceptual data definition was not that required by the user. The adaptation could be a simple restriction, but it should be also possible to define any translation of names or value types or of data structures required by the user.

**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 an emulation of the administrator. It should provide the database administrator with the tools for the definition of the conceptual schema and of external schemas, including the consistency and privacy constraints. It should also provide him with the tools for the internal schema definition. Then, it should give any user the possibility of manipulating his data as simply as possible. It should also 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 support the above objectives, like the concurrency control, the crash recovery,... . For the underlying technical solutions, it was assumed that if a manipulation is not executed as it should be, then the system reruns it.

### 3.1.2. 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 a 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 is probably only to formulate a 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 him and should remain transparent.

Note that this ideal is not the case of the current database systems. Users have 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 of SQL).

### **3.1.3. Limitations**

The database approach is a generous idea, but largely utopian with respect to its basic goals. The concept of data sharing leads to intrinsic problems that do not have a solution today or even cannot have one :

- the administrator is in charge of optimizing the usage of the database for all users. The optimization for a user may be in contradiction with this global goal.
- the user has to explain his needs to the administrator . This may be a difficult process and the needs may change.
- one may imagine very many data structures 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 may be seen as a proof of the failure of this idea. The complexity of the relational 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 may already access through various nets seem too large a collection to be ever manageable as a database.

#### **3.1.4. Synthesis**

The database approach frees the users from many annoying aspects of data manipulation. The way it is done, implies however the loss of the user control of over his own data, eg of the user autonomy. Some of the corresponding drawbacks may be corrected through the sophistication of a DBS. Others are inherent to the complexity of sharing a large collection of data.

Multiple autonomous databases present heterogeneousness similar to those that were in files and triggered the whole story. The limitations of the database approach are however more pronounced, as the number of users and the size of data to manage are larger. The problem of application of the database approach in these conditions was debatable for years and will remain so. One may however reasonably consider the limitations important enough to propose a new approach.

### **3.2. Multidatabase approach**

#### **3.2.1. The idea**

This methodology is discussed in [Lit82] - [Lit87b] and the references of these articles. It considers that the user will in general face multiple databases without a global schema and not a

single database or the corresponding logical image. As autonomous databases may not be mutually integrated, data in different databases will present duplications and discrepancies. The discrepancies will concern the naming, data structuring and value types. One may face also some of inconsistencies the database design was supposed to or would remove. The autonomy may also lead to more frequent changes to data definitions.

Unlike the database approach, the multidatabase methodology does not consider these phenomena as drawbacks to be removed. These are all various facets of the user autonomy to satisfy firstly his own needs. The user should have especially the *data definition autonomy*, including specifically the *naming autonomy*, the *data materialization (duplication) autonomy*, the *data structuring autonomy* at the logical and physical level, the *value type autonomy* etc. In particular, it was postulated that the user may appreciate the possibility to structure his application in the way where different data are in dedicated databases.

To reach these goals, the methodology proposes several concepts, especially this of a *multidatabase language*. Such a language should on one hand provide capabilities of a database language. It should further allow the corresponding databases to be *interoperable* which means usable in common through non-procedural operations, despite the constraints of the autonomy and the absence of the global schema. The formulation of the operations should also remain invariant to changes to data definition, ie these changes should be transparent, as long as the data meaning is preserved. Data definition autonomy should not lead to the burden on the user, making the running formulations of operations obsolete.

### 3.2.2. The concept of a multidatabase system

The system providing a multidatabase language was called a *multidatabase system* (MBS). 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. Multidatabases could bear a collective multidatabase name as they could be unnamed. The multidatabase names could be nested.

A *closed* MBS manages only (multi)databases created through its own interface or through some selected database systems used as servers. An *open* MBS also allows operations on (multi)databases of other MBSs or DBSs. Known systems are all closed MBSs. The ISO-OSI Open System Architecture and especially the Remote Database Access (RDA) protocol is a natural platform for designing an open MBS [Dai88], [Wol89].

One motivation claimed for the concept of a multidatabase system was to respond to developments in the information retrieval, to the usage of collective multidatabase names by many database servers in particular. Another intention was to allow applications to be designed in a way where different data could reside in dedicated databases. It was felt that the user and the

administrator should then 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.

It was in particular felt that multidatabase systems inherently allow the user to better enforce his needs. If the data definition autonomy leads indeed to a conflict with other users, it will affect only the common level operations. All together, it was expected that the drawbacks that would affect many applications if designed according to the database approach, should be at least attenuated with the new approach.

**Ex. 3** Consider users working in a largely autonomous departments. It is likely that most operations in each department would concern this department data, eg suppliers and parts. The multidatabase approach would then be to constitute one database per department. A multidatabase system should then provide some multidatabase language, making implicitly a multidatabase from the databases. The language should allow the definition of each database, as well as the dependencies among them if needed. It should further allow the user to manipulate a database and to easily combine data from different databases. Each department could then have priority in the control of its data, as it has its own database. The corresponding drawbacks of the database approach could be avoided.

In particular, 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 (common) 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.

### **3.2.3. Reference architecture**

The multidatabase systems should obey some reference architecture, extending that of database systems. The architecture at Fig. 1 comes from [Lit82] and extends the well known ANSI-SPARC proposal. The layers of the multidatabase architecture 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 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 to tie the databases together more or less for interdatabase consistency, privacy, etc. In the absence of the global 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. All together, it was postulated in [Lit82] 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.

#### **3.2.4. Functions of a multidatabase language**

To operationally provide the data definition autonomy, a multidatabase language has to offer a number of new functions. At first, one requires functions for a non-procedural definition of data to enter several schemas, including the export and the import of data definitions. One also requires functions for a non-procedural manipulation (queries and updates) of data located in visibly distinct

databases, that may in particular be replicated and should usually differ with respect to names, structures or values despite a similar meaning. The duplication may need to be dealt with logically, as it may have a semantic meaning eg two different recommendations of a restaurant are usually more meaningful than a single one. Finally, one needs functions which let the user provide the formulation which will remain unaffected by changes to autonomous schemas.

Database languages lacked such functions, as they were designed for a single integrated database [Dec87]. 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 [Lit87a]). 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 first characteristic feature 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 should be all in the same database and so there is no need for a common manipulation of different databases. Examples in [Lit82] - [Lit87a] show it may be false for even immediate real-life needs.

Several functions to be provided by multidatabase languages were found through the experimental design of the MRDSM multidatabase system [Lit86], [Lit87]. While some of these functions were intended as general notions, others were specific to the relational data. Their analysis and implementation, was aimed at demonstrating the feasibility and the high utility of relational multidatabase systems. Further work reported in [Lit87a], defines them specifically for the SQL environment, through the proposal of the MSQL multidatabase language. The functions are basically as follows:

- the definition and alteration of multidatabases, to compose (multi)databases into explicitly named multidatabases.
- multidatabase data definition : single statement creation (alteration, drop,...) of a relation in several databases, import of data definition, etc.
- definition of units and of precision of data values, for value type autonomy.
- 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 materialization autonomy (the multiple identifiers in [Lit82]).
- possibility of dynamic unification of heterogeneous names of data objects to deal with naming autonomy (the semantic variables in [Lit86] and column labels in [Lit87a]).
- implicit joins for queries to databases with similar data, but different decomposition into relations, to deal with the data structuring autonomy, [Lit87a]).
- dynamic attributes, for ad-hoc transforms of heterogeneous data values to a user defined basis, for value type autonomy.
- in particular, the capability to update the dynamic attributes [Lit87b].
- 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 [Lit87a]).
- interdatabase queries for import and export of data between databases.
- auxiliary objects like manipulation dependencies, equivalence dependencies and procedures (transactions, stored queries,...) [Lit87a].

Some of these functions required extending the expressive power of database languages. Others concerned only the implementation level (ex. implicit joins; dynamic attribute updates, value unit and precision conversion). MRDSM showed that these functions are feasible. Some of the corresponding implementation techniques and research problems are discussed in Sections 4 and 5. For a while, they were exclusive to MRDSM. Starting from 1987, several now characterize commercial systems and industrial prototypes, which we will discuss in Section 4.

The following example illustrate the capabilities the above functions offer. Although one could express the query below using the actual language of MRDSM, called MDSL, [Lit87], we use MSQ as its SQL compatibility makes it better suitable.

**Ex. 4.** Consider a bank **bnp** with branch offices in Paris called **etoile nation**, **opera** [Lit87a]. Each branch and the main office has a database called upon the corresponding name. The branch databases constitute a multidatabase collectively called **branches**. It was declared by the command : **CREATE MULTIDATABASE branches (etoile nation opera)**. The following samples illustrates how a multidatabase language fits such a typical bank needs.

- The usual practice is that the main database in a bank keeps track of balances and of accounts in branches, being refreshed daily. Consider therefore the query : refresh in **bnp** balances of all branches and add all new accounts, assuming that the following tables are involved :

**bnp** :    **account** (**acc#**, **cl#**, **balance**, **br#**)  
         **br** (**br#**, **brname**, **street**, **tel**)  
**etoile** : **account** (**acc#**, **cl#**, **balance**, **open\_date**)  
**nation** : **acc** (**acc#**, **cl#**, **balance**, **open\_date**, **type**)  
**opera** : **accounts** (**acc#**, **open\_date**, **balance**, **cl#**, **category**)

One formulation of the corresponding MSQL query is as follows :

```
USE bnp branches
LET x BE branches.*
STORE bnp.account  (* *)
SELECT *
FROM x.acc%
```

The **USE** clause defines the scope of the query. The **LET BE** clause defines the explicit semantic variable **x** whose values are names of databases constituting **branches**. The **STORE** command means that one should replace all and only tuples in **bnp.account** whose key matches the incoming tuples (the key here is the column **acc#**). The **(\* \*)** clause means that the source columns are mapped on the target columns with the same names, regardless their order in the table schemas. The values of other target columns are preserved. The designator **x.acc%** in **FROM** clause is a compound semantic variable where **x** is defined above and **acc%** is any table name that starts with the prefix **acc**.

It is highly instructive to try to write this query using SQL which does not have most of the functions used (including the **STORE** command). Note that unless SQL is extended with the concept of database name, there is no way to formulate the query at all. The formulation above is also designed to enhance a relational query *reusability* [Gas87], for a larger data definition autonomy. The administrators have the following possibilities for definition or modification of their schemas that would remain transparent to the query. None of other relational languages provides similar openings.

- the order of matching columns may be changed in their tables.
- new columns may be added at any position in any table schema. As long as the target table has no the column with the same name, no transfer is performed. Otherwise the column is included automatically.
- The administrators have the autonomy to choose and to change the names of the table with accounts as they like, provided the names start with the prefix **acc**. Similar possibility could be applied to column names, through the usage of column labels.

If the system supports the unit conversion at the implementation level, then the users have in addition the autonomy to choose different currencies.

- A new branch **trocadero** is created. To add it to **branches** multidatabase one uses the command :

**ALTER MULTIDATABASE branches INCLUDE trocadero**

From now, any operation with **USE branches** clause, especially the above **STORE**, will have **trocadero** in its scope and will be evaluated also for this database.

- The administrator of **bnp** has created a table **loan** in his database to deal with loans. After a trial period, he decided with the branch administrators to decentralize the management of loans to branch databases. To export the schema of **loan** to all branch databases it suffices to call the **MSQL** command :

```
USE branches
CREATE loan FROM bnp.loan
```

The table name **loan** is here a multiple identifier of all **loan** tables in databases in **branches**. The administrators of these databases may now further customize each table.

- A **bnp** user wishes to have an integrated (single table) multidatabase view of the accounts in branches, in addition to his own table. He also wishes to see the balance in \$ instead of FF. He may get it through the following **MSQL** command :

```
CREATE VIEW bnp.acc_br (acc#, cl#, balance$, brname) AS
USE branches
LET x BE branches.*
D-COLUMN (balance) balance$ = balance / 6
SELECT x.acc%.acc# x.acc%.cl# x.acc%.balance$ NAME(x)
FROM x.acc%
UNION *
```

The **D-COLUMN** clause declares the dynamic column (attribute) named **balance\$** and the value expression for the derivation of its values from the actual values of **balance**. For the updates of the dynamic values, this expression has to be inverted. A technique used by **MRDSM** for updating views with value expressions and dynamic columns is described in Section 5.4 and in [Lit87b], and as far as we know, it is the sole (multi)database system with this capability. The **NAME** is an **MSQL** built-in function converting data name into a value. Note that **NAME** acts here as the implicit *attribute-locality mapping* in the sense of [Sam88]. The queries to **acc\_br** may be processed through the well known principle of the query modification technique.

- User Dupont of **etoile** would like to have the view **acc\_br** also in his database, including for himself the same access rights as defined for user Durand of **bnp.acc\_br**. He may import the view using the following statements :

```
USE etoile
CREATE VIEW etoile.acc_br FROM bnp.acc_br
GRANT ALL ON acc_br TO Dupont FROM Durand ON bnp.acc_br
```

### 3.3. Federated databases

The report [Ham79] 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 [Hei85]. The main principles for a federation constitution were as follows, see other articles in this issue for more details :

- 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 [Hei87], 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 [Hei85] relies in particular on the multidatabase approach (see its references) and the multidatabase approach is inspired by ideas in [Ham79]. 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 [Hei85] 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. Also, one may easily add interdatabase dependencies to the architecture [Hei87].

Conversely, the multidatabase architecture does not assume as a basic feature of a multidatabase, a 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 their popular usage.

The key words for the federated approach are indeed autonomy plus cooperation in interdatabase sharing. The multidatabase approach shares these goals, though it 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 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 a dedicated name in [Lit82]. 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 closed MBS, while negotiation protocols are probably necessary in the open MBSs.

### **3.4. Related methodologies**

#### **3.4.1. Distributed databases and systems**

The concept of a distributed database (DDB) was generally defined as a database transparently implemented on several sites, instead of a single one. This concept differs from that of a multidatabase or of federated databases 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 former 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 are also distinguished, unlike in literature on distributed DBSs until recently [Abb88]. 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 nonetheless that while the distinction between a distributed database and a multidatabase used to be strict in research prototypes and the theory some years ago, it is now disappearing in practice. Especially, most of the commercial systems claiming to be distributed database systems are in fact distributed multidatabase systems. We will show it more in detail later on.

### 3.4.2. Other methodologies

While the above approaches are relatively carefully elaborated, one may also find in the literature other terms and concepts which are rather loosely defined. They frequently overlap with the above terminology or even use the same terms with different meanings. Some have precise meanings, but are frequently used in a popular way with broader, less specific meanings. Some terms seem to gain a new meaning, different from the initial one. The state-of-the-art with respect to the main terms appears to be as follows.

The terms : multidatabase system, federated database(s) system, a system without a global schema and interoperable database system, are in practice synonyms. The term virtual database is the synonym of the terms federated database and distributed database, at least for Mermaid [Tem87a], [Tem87a] and Ingres/Star. The term distributed database system is now frequently used in a loose way to designate all types of systems for the management of distributed databases, including those above. This is especially the case of commercial or popular literature. This may become the new general meaning, as there is no operational system based on the classical definition on the market.

The term external multidatabase schema is a synonym of the terms import schema and superview [Mot87]. These terms are now frequently synonyms of the term global schema though the external multidatabase schema may mean a multidatabase, while the latter is supposed to mean a single integrated database. Also, this meaning of the global schema is new with respect to the classical one, as in particular several such schemas may coexist and data consistency cannot be enforced. One reason is that the work on the global schema design did not consider updates and consistency issues until recently, [Sam88], applying therefore fully also to multidatabase views [Day85], [Cze87], [Man88], [Fan88]. The term gateway designates the system component providing the conceptual schema at the multidatabase layer from an internal logical schema.

The concept of a multidatabase and of a database federation are synonyms, as no one seems to envisage a federation without the possibility of join manipulations. A superdatabase is a particular kind of a multidatabase that is a hierarchical composition of (multi)databases with particular assumptions on concurrency control mechanisms. The supertransactions are also called (atomic) global transactions, while global procedures correspond to non-atomic global transactions. The meaning of the concept of a transaction in multidatabase system is in general evolving toward this latter interpretation.

The concept of a distributed heterogeneous system was some years ago specific to the system for the management of databases with the heterogeneous data models. This is no longer the case, it also refers to data model homogeneous systems with different underlying DBSs or even to homogeneous DBSs with preexisting databases. One then sometimes speaks about the semantic

heterogeneity, in contrast to data model heterogeneity or system level heterogeneity or syntactic heterogeneity [Wol89]. The concept of a multidatabase system is sometimes identified with that of a distributed heterogeneous system, despite the more precise definition of the former and of the broader one of the latter. Sometimes, in contrast, the latter concept seems to designate any type of system for the management of heterogeneous databases, leaving the room for considering systems with a global schema as well.

#### **4. COMMERCIAL SYSTEMS AND INDUSTRIAL PROTOTYPES**

##### **4.1. Commercial systems**

###### **4.1.1. Relational databases**

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 was important in this respect, as the first commercial systems appeared. They were 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 now present their current features related to the scope of this survey, using the MRDSM functions as the framework. Other features of these systems are extensively discussed elsewhere in this issue. Table 1 sums up the comparison, also with respect to functions provided by prototypes other than MRDSM and discussed in Sections 4.2 and 5. The 'Research' column indicates the system that was the first to provide the function or provides the most extensive implementation, unless no system provides the function. The 'Y' in other columns means 'Yes', though the system may support the function less extensively than a research prototype (which is normal). This is particularly true for implicit joins and the multidatabase aggregate functions, where for instance no commercial system provides NAME function, as far as we know. Then, 'YY' means more extensive capabilities than those of the other systems, while 'P' means planned. Note, that while no commercial system provides all the discussed functions, all together they provide most of them.

## Sybase

This system designed by Sybase Inc. is a high performance relational system, available in particular 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 later on.

Sybase language called Transac-SQL is an extension of 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 under the form of triggers or of procedures embedding triggers into a programming language. A manipulation of one database, may thus trigger that of another database. The dependencies may be transitive ie fire one another, provided they are declared as procedures. 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, available 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. 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 soon will be widely used. It 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 under the name of SQL Server. If these plans finalize, database management will become in general multidatabase systems, realizing the postulate in [Lit82]. This will be the case not only for the distributed databases, but also the monosite (physically centralized) environment, as was also conjectured there.

### **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, as it manages the distribution through the NFS file management 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 interoperability of multidatabase and multimedia systems.

### **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 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 seems 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 are required for ad-hoc multidatabase queries than for Sybase and Empress. In addition there is a danger of system pollution with DDBs and the underlying hidden actual databases, created for an ad-hoc 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.

### 4.1.2. Information retrieval databases

The Common Command Set Language (CCS) appears to be the best candidate for the standard language, at least in Europe, for this type of databases. However, as SQL it needs to be extended with functions for multidatabase management, like these provided by the Messidor multidatabase system [Lit82]. Some of these functions also appear now in the commercial systems.

### EasyNet

EasyNet is the gateway switch and the common single site image interface to over 850 databases, distributed on several servers. The connection procedures, passwords etc. are transparent. It offers an extension to CCS as the standard language, but also allow to use local languages. Databases are grouped under multidatabase names which are key words like history, finance, law, etc. A database may belong to any number of multidatabases. The user may search multiple databases under the same multidatabase name using a multiple query. In the terminology of EasyNet such queries are called *scans*. The scan shows the number of records retrieved in each database. The user may then reformulate the query to narrow the search.

### ii

The bibliographic multidatabase system strangely termed ii, was put to commercial service recently. It allows the user to query a given database or a number of databases simultaneously, in an extension to CCS language. It thus supports the concept of multiple queries. As EasyNet it

further provides multidatabases names and a database may belong to any number of multidatabases. The system got the award of the product of the year at the 1987 On-Line conference.

#### **4.2. Industrial prototypes**

We designate in this way prototypes that are likely to give rise to operational systems in the near future. One is the Mermaid system initially intended as a classical distributed DBS, it now evolved towards the federated architecture. It is presented elsewhere in this issue and in [Tem87], [Tem87a], where one describes problems and original solutions to deal with the privacy issues in the multidatabase environment. The following two prototypes are also promising.

##### **Calida**

This system is under development in GTE Research Laboratories. The operational version is destined for the management of numerous databases of GTE, mostly the relational ones. The main features of the system, in the scope of this survey are as follows [Jac88] :

- Calida makes it possible to access relational and Codasyl-like databases or hierarchically structured files. The data model at the multidatabase level is relational. 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 U/DELPHI (Universal DELPHI). It is an SQL like, multidatabase, joinless version of earlier DELPHI language. U/DELPHI is used as a final language for the sophisticated user and as an intermediate language for a natural language for interface. U/DELPHI allows the formulation of elementary multidatabase queries, including the updates, where database names may be used as prefixes to solve name conflicts. 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 equivalences 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.

Among other interesting features of Calida one may cite the menu driven User Interface Module with the possibility of user defined macros.

### **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 [Bel87]. 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. An appealing feature is also its algorithm for SQL query decomposition. Views are dealt with using the query modification technique. The query is then 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.

### **4.3. Interoperable database system**

This title refers to the name of a large national project in Japan [Int87]. 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 M\$ 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. A longer term goal of the project is also to make databases interoperable 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.

### **4.4. Impact on standardization**

ISO has 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 extensions to SQL that appeared in the commercial

systems, as well as the joint work of most of the corresponding companies on the common version, called Open SQL, it is however likely that ISO will include at least this multidatabase function into coming releases of the standard. One should also note the proposals of C. Date for improvements to SQL [Dat84]. One of the proposals is to provide names for the results of SQL value expressions (p. 35) that 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 used by ISO in its work on Remote Database Access Protocol (RDA) [Iso87]. MBSs are assumed more common. The RDA protocol is intended mainly for these systems though in its current version it lacks many features.

## **5. SOME RESEARCH ISSUES**

The number of investigations of multidatabase (federated) systems has greatly increased recently. Some are reported in the references to this survey, in particular in [Chu87], and [Sar87]. They have led to interesting results and inspiring concepts, opening a number of new issues. Their common rationale is that the classical techniques for database management 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 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 [Alo87], 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 [Alo87]. 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 [Bre87]. 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 [Pu87]. Finally, under similar assumptions, the optimistic control [Elm87] may be used.

- Locking and timestamping have nevertheless limitations which no algorithm using these concepts may overcome, as they results from the paradigms themselves. A more general paradigm is introduced in [Lit88] and termed *value dates*. This concept is well known in the banking, airlines,... and means that a value is certain only after its value date. Otherwise, it is uncertain, though may be used with precautions. The concurrency is controlled through the examination of value dates put by transactions under the control of the scheduler. As value dates define a serial order, one may easily obtain the serializability of the schedules.

Value dates exhibit several interesting properties. The corresponding schedules may be deadlock and livelock free independently of the transaction semantics. The simplicity of the corresponding algorithm compares favorably to that of the two phase locking (which is the most used algorithm, but is not deadlock free). Furthermore, through the dedicated manipulations of value dates or of uncertain values, one may design schedulers enhancing efficiency for particular classes of transactions. Also, the paradigm shows that locking and timestamping are in fact particular cases, where value dates are assumed infinite. This observation explains limitations of those paradigms, especially the impossibility to ever eliminate the deadlock while using locking.

Finally, the value date paradigm provides a new view on the commitment and recovery problems. A value date may be seen as an implicit non-procedural commit point. The current commitment protocols, like two phase commit, appear as particular procedural implementations.

Other choices appear then as well and look at least as useful. All together, these properties make the new paradigm highly promising.

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 [Eli87], [Wie87]. Especially, they bring to consideration as henceforward most important criteria for the transaction processing various facets of the concept of autonomy, like C-autonomy, D-autonomy, and E-autonomy (Communication, Design, Execution) for S-transactions in [Eli87], or Abort Autonomy, Lock Autonomy or timeliness in [Gar88]. 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. Data definition

Databases managed by different systems need conventions for the data exchange. Autonomous systems may in particular use different data formats, encoding or precision of similar data. The brutal approach, such as a list of all possible conversions, is excluded because the number of conversions may be very large in an open system. The usual binding of names by exact and explicit designation is also too limited in the present systems and would require a global schema with a possibly very large number of generalizations, aggregations, etc. The concepts of self-description and of self-documentation are one solution to this problem [Rou82, 83, 85], [Mar87].

A *self-describing database* describes and controls its type objects in terms of meta-objects stored in its catalogs and especially its Data Dictionary [Mar87]. A *self-documenting database* captures into its catalogs and controls the evolution of all derived data objects [Rou83]. Derivation dependencies are used to take appropriate actions during changes on the schema affecting derived objects. The descriptions of the objects, especially the data formats, are exchanged among autonomous databases using *Standard Format Data Units* (SFDUs) [Ccs87].

### 5.2.1. Self-describing database system

Fig 3 illustrates the architecture of a self-describing database system. This architecture is a framework for database management systems, proposed by the Database Architecture Framework Task Group (ANSI/X3/SPARC/DAFTG) [Bur86], and recently adopted by the ANSI/X3/SPARC Database System Study Group. The data dictionary is an active and integrated part of the system, used for information interchange. The core DBMS supports the internal, conceptual, and external schemata. It also supports the intension-extension dimension of data description with four levels. Each level is the extension of the level above it, and the intension for the level below it.

The data are application data. The data dictionary stores the application schemata. The data dictionary schema contains the rules for managing these schemata. Finally, the meta-schema stores the basic set of rules for defining self-describing data models.

The core DBMS supports the *data language* (DL), which is the language used to manipulate data and data descriptions at any level in the intension-extension dimension. The DL provides a set of primitive operations on any data element or data description element at any level in the intension-extension dimension of data description. Any compound operation is implemented as a tool in the data management tool box using the primitive operations of the DL. The tools are plug-compatible with the core DBMS through the DL. Some tools are specific to information interchange.

### 5.2.2. Self-documenting database systems

Conventional database systems have mostly dealt with the management of the database extension. They simply retrieve, insert, modify, delete or restructure (derive views of) data instances. This includes maintenance of the views during updates of the base files, view indexing techniques for storage savings and performance improvement, etc. On the contrary, very little work has been reported in the management of the evolution of the schema because it has been mostly thought of as relatively stable over time. When the intension (schema) changes, several derived objects, views, materialized views, and/or compiled queries may be affected. In a centralized system with tight control and a relatively few data objects, a simple invalidation flag may be adequate. A multidatabase environment leads however to large collections of basic and derived data objects and to the autonomy of schema design. A more systematic approach to management of schema catalogs is necessary. Without it, there is no way for instance to prevent an unexpected failure of a query or of a view definition that ran well, because an autonomous administrator has locally modified the schema of a relations involved. This is what by the way happens on current systems. The failure may be in particular hidden i.e. the execution would take place, but would be fallacious.

A self-documenting database concept is a solution to this problem in that it keeps a record of what has happened to the database objects. For this purpose, all objects (schema and data objects) are derived via some data language whose semantics is controlled by the database. The whole meaning of any derived object is then kept in a *derivation record* in the database's catalogs, instead of being partly in the administrator's mind, as in today systems. A derivation record may in particular contain information about other cooperating databases.

The catalogs may be used in two ways. First, the participating systems can check the affected derived objects and take appropriate correction actions on schema changes, or simply notify the users and invalidate the affected objects. The system may then for instance verify whether a multidatabase view definition is still valid, before processing the query. Also, the users and the

systems can take advantage of derived objects that may be more refined than the ones they used to access or planned to define. In particular, one may avoid the proliferation of equivalent views or of other objects. Self-documentation allows in this way a smoother and more natural evolution of autonomous databases.

Management of the schema catalogs can be a complex process. It is however simpler for relational data, as the relational model provides some mechanisms for specifying or inheriting intensional properties of a derived object. These may be the derived object's name, the names of its attributes, the inherited security and authorization properties or constraints etc. Also, the well-defined semantics of the relational algebra used to derive data views allows a precise interpretation of the schemas of the derived objects. On the other hand, one may also use the Entity Relationship model [Bat88].

In the distributed environment, the autonomy of the cooperating databases requires a distributed control over catalogs for self-documentation. A fully distributed control can be very time consuming especially if instantaneous distributed update of many catalogs is required. The primary copy techniques with incremental updates of the copies may be a more down-to-earth approach [Rou86].

### 5.2.3. Standard format data units

It is proposed that self-describing and self-documenting databases exchange data through Standard Format Data Units (SFDUs) [Ccs87]. DL commands that should be sent are considered simply as data of some type. An SFDU is also self-describing. It contains both data and format represented by a recursive type-length-value (TLV) encoding with the syntax in Fig 4. The fixed length Type Field T identifies the format of data that are in Value Field V. The fixed length Length Field L represents the length of V, basically in bytes. The V field itself can be in any representation supported by the system.

SFDUs are defined using a Format Definition Language, (FDL). Specific standards for an FDL, are not included in [Ccs87]. It is, however, agreed that it should support the description of the data structure, the unit of measure, the range, and the precision of each value. The problem of conversion of value types between autonomous systems, needed especially to make operations like relational join meaningful, has a local solution for each system. Note that RDA [Iso87] is an FDL, but without these possibilities, at least today.

### 5.3. Update dependencies

Frequently, the update of a base relation implies updates of other relations related through some underlying dependencies that are not expressed in the database schema. This situation becomes even more common in multidatabase environment where there is no global schema. To deal with

these update dependencies a solution may be triggers embedded in some programming language, eg Sybase. However, a general solution requires a high level non-procedural formalism capturing update dependencies semantics. Such a logic programming like formalism as well as the corresponding prototype implementation is presented in [Rou84], [Mar85], [Mar87a].

A relation R is *update dependent* on relation S if an update operation on relation R implies update operation(s) on relation S. Such *update dependencies* may be declared in the following form.

```

<op> -> <c,1>, <op,1,1>, <op,1,2>, .. <op,1, n1>.
      -> <c,2>, <op,2,1>, <op,2,2>, .. <op,2, n2>.
      -> .. .

```

The term <op> is a *compound update operation* on some relation. The <c,i> are alternative conditions on the database state, expressed as predicates. Each <op,i,j> is either a compound update operation, defined in another update dependency, or a *primitive operation, implied* by <c,i>. An evaluation of a compound update operation succeeds (evaluates to true) iff, at least one <c,i> evaluates to true and all the operations implied by <c,i> succeed. The compound update operations are the data manipulation commands of the database language eg MySQL. The evaluation of primitive operations always succeed. They are **add** for adding a single tuple, **remove** for eliminating one, **write** and **read** for the interface with the user and some others. See [Mar85] for details of all these operations.

When an update dependency is invoked its variables are bound to the actual values, selected by the database system or to values supplied by the interacting user. The evaluation of conditions, replacement of implied compound update operations, and execution of implied primitive operations is in order of appearance, left-to-right and depth-first. The evaluation of conditions assumes a closed world interpretation. If an evaluation or an execution fails, the backtracking takes place. The database is physically modified iff the <op> evaluation succeeds (unlike if Prolog expressions were used with assert etc. operations).

An implied compound update operation matches the left-hand side of an update dependency if the operation names and the relation names are the same, and all the domain components which are variables or constants match. They match if they are the same constant or if one or both are variables. If a variable matches a constant it is instantiated to that value. If two variables match they share value.

**Ex. 5** Recall the bank example Ex. 4. We will declare the update dependencies (i) to create an entry in the **bnp** main office for each new account created in the branch **etoile** and (ii) to refresh the balances as accounts get updated in the branches.

```
insert(etoile<x,y,z,w>)    -> insert(bnp<x,y,z,'etoile'>).
                           -> write("unable to insert to bnp").
update(etoile<x,y,z,w>)    -> bnp(x,y1,_,_) & not(y = y1),
                           write("inconsistency detected").
                           -> bnp(x,y,_,_), update(bnp<x,y,z,'etoile'>).
                           -> not(bnp(x,_,_,_)), insert(bnp<x,y,z,'etoile'>).
                           -> true.
```

The first update dependency takes care of the insertions in **etoile**. The insert in **bnp** is tried first. If it succeeds, ie no logical constraint forbids such an insertion in **bnp** independently of when it will be physically stored it, then the insert in **etoile** succeeds and both databases are allowed to accept the new account. Otherwise, the second clause gets executed and sends the message to the **etoile** administrator. Because the write operator always succeeds, the insert in **etoile** also always succeeds. This is consistent with the notion of autonomy of the databases.

The second update dependency checks first whether the accounts to update in **etoile** and in **bnp** are consistent with respect to the sharing of the **cl#** values, **y** and **y1**. The '\_' is like in Prolog an anonymous variable. If an inconsistency appears, the message is sent and the update dependency succeeds. Otherwise, the second clause makes the update of the new balance **z** in **bnp** while the third clause takes care of the case of **bnp** yet unaware of the updated account (for instance, when the insertion to **bnp** has failed when this to **etoile** had occurred). To guarantee the autonomy of **etoile** again, the true clause is terminates the definition of the dependency that always succeeds, even all the specified manipulations of **bnp** have failed. Again here, the update dependency only checks for logical evaluation to true. The actual updates to the database may be at a different time depending on the implementation of the underlying system.

#### 5.4. Updates of transformed values

In the multidatabase environment, heterogeneous data values should frequently be converted or combined to a value type defined by the user. This is one of the facets of the value type autonomy. MRDSM provides for this purpose 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 defined by the user in queries and disappear once the query is processed or, at most, at the end of the session. 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 in general not considered in the literature or one could understand that they are impossible (eg [Dat86], p. 187). MRDSM provides a solution for dynamic attributes that naturally applies to view update as well. Its principle is as follows [Lit87b]:

- 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  $F' : D \rightarrow A$ .

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, including MACSYMA, and may be even cheap on PCs (TK!, Eureka etc.). The capabilities of cheaper versions are more limited, but are sufficient for many applications and are extend rapidly with new releases.

### 5.5. Query processing

Multidatabase languages introduced new capabilities whose optimization is an open research area. The following problems seem particularly important.

- Traditional work on the distributed query decomposition considered that 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 a need for the self-description of a database system 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. A knowledge base may be used to check and disambiguate multidatabase queries in presence of semantic conflicts [Rus88].
- Different types of formulas  $F$  for value conversion lead to different optimal algorithms. Furthermore, the most efficient algorithm for a few tuples, may be not the optimal one for many tuples. For instance, if  $F$  is a polynomial, then the Horner algorithm is the optimal one for a single tuple. If however several tuples are involved, then the Knuth algorithm, preprocessing the coefficients, outperforms it [Kro87].
- The multiple queries are 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 [Ros88], [Sel88].

- Similar problems may occur for interdatabase queries, where data selected from some source database should be dispatched to several databases. Basically, such 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 use as the source, a small temporary relation produced by the selection expression of another subquery, instead of large base relations.

#### **5.6. Incremental access of multiple database servers**

The system termed ADMS+- proposes an architecture for the frequent case of workstations that manage private data and share those in databases on servers [Rou86]. ADMS+- has two subsystems. One termed ADMS+ runs on each server and manages the shared database. The other, called ADMS-, runs on the workstations and manages a local database consisting of a) private non-shared relations, and b) downloaded subsets of the shared server databases. Private relations are created and managed only locally on the workstation. The downloaded shared subsets are acquired as a result of running queries against the server databases. From then on, they are incrementally maintained on the local database as materialized views [Rou87]. Therefore, by running the workstation's application, the data that is pertinent to those applications incrementally migrate to the workstation. The communication of data is only between the workstations and the servers ie there is no direct communication between workstations. However, extensions of the of ADMS+- principles to more distributed topologies is a subject of current research.

ADMS+ is a full-fledged system that keeps track of the downloaded subsets of the database to the workstations through self-description and self-documentation. ADMS- is a trimmed version of ADMS+ that has neither a security nor a concurrency control subsystem because it operates in a single-user mode. ADMS- can be thought of as an intelligent cache database access subsystem that capitalizes on the locality of the downloaded data and the ability to process it independently from ADMS+. The user can even introduce to the local database additional access aids such as local indexing.

The design goal of the ADMS+- architecture is to download to each workstation a tailored subset of the database to be processed locally by special-purpose transactions running on the workstation. The servers maintain the interoperable databases and additional access aids for subsets shared by several workstations. The user can combine private and shared data in relational views derived from shared and private relations. This gives him the autonomy to run SQL as if they were only the local database. Another benefit is that the download of the share database subsets

provide quicker access to data, as most of the transactions run locally. This not only increases the overall throughput, as all workstations do parallel processing, but also reduces the servers' load considerably.

Updates to the shared databases are only allowed on the servers. A deferred, periodic, or immediate broadcasting update strategy can be selected for the downloaded portions. When an update of a downloaded object is decided, incremental update techniques are used to minimize data transmission and message traffic. Downloaded portions can be discarded if they are no longer useful. This has no effect on the performance, other than space saving on the local disk, if the deferred update strategy is specified.

The most attractive feature of ADMS+- architecture is its efficiency with respect to the multidatabase management of private and shared databases. The local database integrates private and downloaded views from the shared server databases. ADMS+- provides, however, at the same time the overall interoperability of the shared parts to enforce their consistency. It also provides the workstation site autonomy, as the local databases may run even when the servers are down. The ADMS+- techniques for dealing with queries involving private and shared data through exported views are in particular a promising approach to efficient processing of distributed multidatabase queries.

### **5.7. Logic multidatabase systems**

Future databases will frequently be knowledge bases using the concept of logic programming, especially Prolog. This will obviously lead to the situation of autonomous knowledge bases to be manipulated together. These bases will differ with respect to names, values and predicate structures for similar models. Functions will be needed for the corresponding non-procedural logic multidatabase manipulations.

Probably the first attempt to solve this problem is described in [Kuh88]. The presented system called VIP-MDBS, extends the actual Prolog database system VIP-DBS. The latter system is also built by the authors, on the top of the Vienna Integrated Prolog, designed by the same team as well. VIP-MDBS provides several functions transposing those of MSQL to the logic multidatabase environment. It is shown that to build multidatabase manipulation functions in this environment may be surprisingly hard. One reason is that Prolog alone lacks semantics and thus the existence of a conceptual schema over the set of predicates constituting a database is necessary. Another reason is that Prolog statements basically use the mapping by position of the data designators which are in addition arbitrary variable names. In contrast, SQL and relational languages in general, use the mapping by data object names.

The language of VIP-MDBS described in [Kuh88] overcomes these structural properties of Prolog databases for most MSQL multidatabase functions. It also provides features that does not

exist in MSQL, since a relational system does provide them in practice. Most significant feature are intentional relations and the transitive closure. VIP-MDBS allows its usage, in multidatabase queries, to define multidatabase views (called recursive semantic relations), and to formulate multidatabase integrity constraints. This feature allows to handle practical problems such as of a bank seeking to find the companies among its customers which are controlled by a holding X, given that X may have shares of a company Y, but also of a company Z in some other bank, that is a shareholder of company Y etc. The corresponding integrity constraint may be that the sum of shares of a company owned by X can never exceed 100 %.

### 5.8. Dynamic Derivation of Personalized Views

Traditionally, user interfaces to databases have either assumed complete knowledge of the conceptual schema or they have relied on the utilization of predefined views, to restrict the universe of discourse for end users or application programmers. This approach is appropriate if the scope of the queries a user may want to formulate is known in advance. In the case a user wants to pose an unexpected query or in a case of a new user, a predefined view is not the answer. One needs rather functions for a dynamic derivation of views corresponding to the hypothetical (universal) view in the user's mind. Such functions for multidatabase environment are proposed in [Neu88].

The proposals considers the object oriented paradigm and applies the framework defined in [Fan88]. It further transposes to this universe the principles of the universal scheme and the implicit joins method. More specifically, it is observed that the algorithm in [Lit85] computes implicit joins through adding to the incomplete query graph as few edges as possible to obtain at least one connected spanning tree (the edges are selected from the database graph whose nodes are relations and edges so-called natural dependencies). This minimization of the number of joins may not always be the correct interpretation of a query, as the number of joins is merely a syntactic measure.

To refine answers, one proposes in [Neu88] to use an object oriented data model and a knowledge-based approach to complete the incomplete queries. The central idea is the concept of *message forwarding plan* that defines at the class level how messages that cannot be handled directly are to be forwarded to semantically related objects. The corresponding forwarding paths are for the method the equivalent of the natural dependencies. The most promising paths are dynamically detected by an interactive *knowledge navigator*, that is an expert system using domain knowledge base, various thesauri,...

The concept of a *context of an object* is introduced to capture the semantic surroundings of the object classes visited on the message forwarding path, instead of making a selection from a syntactic measure. The paths determined with help of these tools are finally selected and combined to define the final view. The main problem for the combination algorithm is the detection of cyclic paths. A rule based algorithm that is proposed ensures that only "useful" cycles are included in the

plan and that they can be expected to terminate during the actual execution of a query against the plan. The whole technique is under implementation on Sun workstations for the construction of dynamic personal views for multiple and multi-media databases.

## 6. CONCLUSION

The systems for the interoperability of multiple autonomous databases are emerging. It is likely that a database system which would not be a federated (multidatabase) one and distributed, will soon be hard to find. The major commercial systems already evolved in this direction. The perspectives of worldwide usage of systems like SQL Server, Distributed Ingres, Oracle V5, etc. show that systems of this type will be among popular software tools.

The capabilities for the multidatabase manipulations already present in the commercial systems are nonetheless only a limited subset of those proposed 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. They will be of fundamental importance for the economical development of our information based post-industrial societies.

## REFERENCES

- [Abb88] Abbott, K. R., McCarthy, D. R. Administration and Autonomy in A Replication-Transparent Distributed DBMS. 14-th Int. Conf. on Very Large Databases, Los Angeles, USA, (Aug. 1988), 195 - 205.
- [Alo87] Alonso, R., Garcia-Molina, H., Salem, K. Concurrency Control and Recovery for Global Procedures in Federated Database Systems. IEEE Data Engineering, (Sep. 1987), 10, 3, 5-11.
- [Bat88] Batini, C. Di Battista, G. A methodology for conceptual documentation and maintenance. Inf. Syst. 13, 3, 1988, 297-318.
- [Bel87] Belcastro, E & all. DQS -Distributed Query System. (Sept. 1987), CRAI, Italy, 21. Int. Conf. on Extending Database Technology, Springer Verlag, 1988.
- [Bre87] Breitbart, Y., Silberschatz, A., Thompson, G. An Update Mechanism for Multidatabase Systems. IEEE Data Engineering, (Sep. 1987), 10, 3, 12-18.
- [Bur86] Burns, T. Fong, E. Jefferson, D. Knox, R. Reedy, C. Reich, L. Roussopoulos, N. Truszkowski, W. Reference Model for DBMS Standardization. ACM SIGMOD Records, (March 1986).

- [Ccs87] Consultative Committee for Space Data Syst. : Standard Formatted Data Units - Structure and Construction Rules. Red Book, Issue 2, (Feb. 1987). Nat. Aeronautics and Space Adm.
- [Cer87] Ceri, S., Pernici, B., Wiederhold, G. Distributed Database Design Methodologies. Proceedings of the IEEE, (May 1987), 533-546.
- [Chu87] Special Issue on Distributed Database Systems. Chu, W. (ed). Proceedings of the IEEE, (May 1987), 532-735.
- [Czc87] Czejdo, B., Rusinkiewicz, M., Embley, D. A Unified Approach to Schema Integration and Query Formulation in Federated Databases. Res. Rep. University of Houston, 1987, 25.
- [Dai88] Distributed Aspects of Information Systems (DAISY Working Group Rep). Research into Networks and Distributed Applications. R. Speth (ed.). Elsevier Science Publ. 1988, 1029 - 1049.
- [Dat84] Date, C., J. A Critique of the SQL Database Language. SIGMOD Record, 1984, 8-54.
- [Dat86] Date, C., J. An Introduction to Database Systems, 4-th Ed. Vol. 1. Addison-Wesley, 1986, 639.
- [Day85] Dayal, U. Query Processing in a Multidatabase System. Query Processing in Database Systems, 1985, Springer Verlag, 81 - 108.
- [Dec87] Deen, M., S., Amin, R., R. Taylor, M., C. Data Integration in Distributed Databases. IEEE Trans. on Soft. Eng., 13, 7, (July 1987), 860-864.
- [Eli87] Eliassen, F., Veijalainen, J. Language Support of Multi-database Transactions in a Cooperative, Autonomous Environment. IEEE Region 10 Conf., Seoul, (Aug. 1987).
- [Elm87] Elmagarmid, A., Leu, Y. An Optimistic Concurrency Control Algorithm for Heterogeneous Distributed Database Systems. IEEE Data Engineering, (Sep. 1987), 10, 3, 26-32.
- [Fan88] Fankhauser, P., Litwin, W., Neuhold, E., Schrefl, M. Global View Definition and Multidatabase Languages : Two Approaches to Database Integration. Research into Networks and Distributed Applications. R. Speth (ed.). Elsevier Science Publ. 1988, 1069-1082.
- [Gar88] Garcia Molina, H., Kogan, B. Node Autonomy in Distributed Systems. IEEE Int. Symp. on Databases in Parallel and Distr. Systems. 1988, 158-166.
- [Gas87] Gash, B., Kelter, U., Kopfer, H., Weber, H. Reference Model for the Integration of Tools in the "EUREKA Software Factory". ACM-IEEE Fall Joint Comp. Conf. (Oct. 1987), 183-190.
- [Ham79] Hammer, M., McLeod, D. On database management system architecture. MIT Lab. for Comp. Sc. MIT/LCS/TM-141, (Oct 1979), 35.
- [Hei85] Heimbigner, D., McLeod, D. A Federated Architecture for Information Management. ACM Trans. on Office Information Systems. (July 1985), 3, 3, 253-278.
- [Hei87] Heimbigner, D. A Federated System for Software Management. IEEE Data Engineering, (Sep. 1987), 10, 3, 39-45.
- [Hew85] Hewitt, C., De Jong, P. Open Systems. On Conceptual Modeling. Springer Verlag, 1985, 147-164.
- [Int87] Interoperable Database System. 1st International Symposium. INTAP, (May 1987), 167.
- [Iso87] Remote Database Access Protocol. 2-nd Working Draft. ISO/TC 97/SC 21/WG 3, 1987.
- [Jac88] Jakobson, G., Piatetsky-Shapiro, G., Lafond, C., Rajinikanth, M., Hernandez, J. CALIDA : A Knowledge-Based System for Integrating Multiple Heterogeneous Databases. 3-rd Int. Conf. on Data and Knowledge Bases : improving usability and responsiveness. Jerusalem, (June 1988), Morgan Kaufmann Publ., 3-18.
- [Kro87] Kronsjö L. Algorithms : Their Complexity and Efficiency. Jonh Wiley & Sons. 1987, 363.
- [Kuh88] Kuhn, E., Ludwig, Th. VIP-MDBS : A Logic Multidatabase System. IEEE Int. Symp. on Databases in Parallel and Distr. Systems. 1988, 190-207.
- [Lit82] Litwin W. et al. SIRIUS Systems for Distributed Data Management. Ed. H. J. Schneider. North-Holland, 1982, 311-366.
- [Lit84] Litwin, W. MALPHA : A relational multidatabase Manipulation Language. 1-st IEEE Conf. on Data Engineering, Los Angeles (Feb 1984).
- [Lit85] Litwin, W. Implicit Joins in the Multidatabase System MRDSM. IEEE-COMPSAC, 1985, 495-504.
- [Lit86] Litwin W., Abdellatif, A. Multidatabase Interoperability. IEEE Computer, (Dec. 1986), 19, 12, 10-18.

- [Lit87] Litwin W., Abdellatif, A. An Overview of the Multidatabase Manipulation Language MDSL. Inv. paper. Proc. of the IEEE, 75, 5, (May 1987), 621-632.
- [Lit87a] Litwin W., 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.
- [Lit87b] Litwin W., Vigier, Ph. New Functions for Dynamic Attributes in the Multidatabase System MRDSM. HLSUA Forum XLV, New Orleans, (Oct. 1987), 467-475.
- [Lit88] Litwin W., Tirri, H. Flexible Concurrency Control using Value Dates. IEEE Distr. Proc. Techn. Comm. Newsletter. Spec. Issue on Heterogeneous Distributed Database Systems, 10, 2, Nov, 1988, 42-49.
- [Man88] Mannino, M., V., Navathe, S., B. Effelsberg, W. A rule-based approach for merging generalization hierarchies. Inf. Syst. 13,3, 1988, 257-272.
- [Mar85] Mark, L., Roussopoulos, N., Chu, B., Update Dependencies. IFIP TC2 WG 2.6 Working Conference on Database Semantics, (Jan. 1985), Belgium.
- [Mar87] Mark, L. Roussopoulos, N. Information Interchange between Self-Describing Databases. IEEE Data Engineering, (Sep. 1987), 10, 3, 46-52.
- [Mar87a] Mark, L., Roussopoulos, N. Operational Specifications of Update Dependencies. SRC Res. Rep., Univ. of Maryland, (Feb. 1987), 44
- [Mot87] Motro, A. Superviews : Virtual Integration of Multiple Databases. IEEE Trans. on Soft. Eng., 13, 7, (July 1987), 785-798.
- [Neu88] Neuhold, E., Schrefl, M. Dynamic Derivation of Personalized Views. 14-th Int. Conf. on Very Large Databases, Los Angeles, USA, (Aug. 1988), 183-194.
- [Pu87] Pu, C. Superdatabases : Transactions Across Database Boundaries. IEEE Data Engineering, (Sep. 1987), 10, 3, 19-25
- [Ros88] Rosenthal, A., Chakravarthy, U., S. Anatomy of a Modular Multiple Query Optimizer. 14-th Int. Conf. on Very Large Databases, Los Angeles, USA, (Aug. 1988), 230 - 239.
- [Rou82] Roussopoulos, N. Wallace, S. Self-Description vs. Self-Documentation. Workshop on Self- Describing Data Structures, Univ. of Maryland, October 1982.
- [Rou83] Roussopoulos, N. Intensional Semantics of a Self-Documenting Relational Model. Dept. of Computer Science, Techn. Rep. 1264, (April 1983), Univ. of Maryland.
- [Rou84] Roussopoulos, N. Mark, L. Update Dependencies in Relational Databases. 1st International Conference on Expert Database Systems, Kiawah Island, South Carolina, (Oct. 1984).
- [Rou85] Roussopoulos, N. Mark, L. Schema Manipulation in Self-Describing and Self-Documenting Data Models. Int. J. of Comp. and Inf. Sc., 14, 1, 1985, 1-28.
- [Rou86] Roussopoulos, N., Kang, H. Principles and Techniques in the Design of ADMS+. IEEE Computer Magazine, Vol.19, No. 12, (Dec. 1986), pp. 19- 25.
- [Rou87] Roussopoulos, N. Overview of ADMS: A High Performance Database Management System. Inv. Paper, Fall Joint Comp. Conf., Dallas, Texas, October 25-29, 1987.
- [Rus88] Rusinkiewicz, M et al. Query Processing in OMNIBASE : a loosely coupled multi-database System. Tech. Rep. #UH-CS-88-05, Univ. of Houston, (Feb. 1988), 27.
- [Sam88] Samy Gamal-Eldin, M., Thomas, G. Elmasri, R. Integrating Relational Databases with Support for Updates. IEEE Int. Symp. on Databases in Parallel and Distr. Systems. 1988, 202 - 209.
- [Sar87] Special Issue on Federated Database Systems. Sarin, S. (ed.). IEEE Data Engineering, (Sep. 1987), 10, 3, 64.
- [Sel88] Sellis, T. Multiple queries optimization. ACM-TODS, 1988.
- [Tem87] Templeton, M. et al. Mermaid : A Front-End to Distributed Heterogeneous Databases. Proceedings of the IEEE, (May 1987), 695-708.
- [Tem87a] Templeton, M., Lund, E., Ward, P. Pragmatics of Access Control in Mermaid. IEEE Data Engineering, (Sep. 1987), 10, 3, 33-38.
- [Wie87] Wiederhold, G., XiaoLei, Q. Modeling Asynchrony in Distributed Databases. 3rd IEEE Conf. on Data Engineering, Los Angeles, (March 1987), 246-250.

[Wol89] Wolski, A. LINDA : A System for Loosely Integrated Databases. 5-th IEEE Conf. on Data Engineering, Los Angeles (Feb 1989).

Function	Research	Sybase	Empress	D. Ingres	Oracle	EasyNet ii
Multidatabase names	MRDSM		Y			Y
Multidatabase data def.						
Unit & precision def.						
Elementary queries	MRDSM	Y	Y		Y	
Multidatabase updates	MRDSM	Y	Y			
Interdatabase queries	MRDSM	Y	Y		YY	Y
Multiple queries	MRDSM	P				Y
Semantic variables	MRDSM	P				
Implicit joins	MRDSM	Y				
Dynamic attributes	MRDSM					
Updates of dyn. attr.	MRDSM					
Mdb aggr. functions	MRDSM	Y	Y	Y	Y	Y
Mdb transitive closure	VIP-MDBS					
Multidatabase views	VIP-MDBS	Y	Y		Y	
Manipul. dependencies	MRDSM	YY			Y	
Virtual databases			Y	Y		
Heterog. data models	DQS			P	P	Y
Distributed databases	LINDA		Y	Y	Y	Y
RDA draft protocol	LINDA					
Self-descr. & docum.	ADMS+-					
Incremental views	ADMS+-					

Table 1 Functions for multidatabase management in existing systems

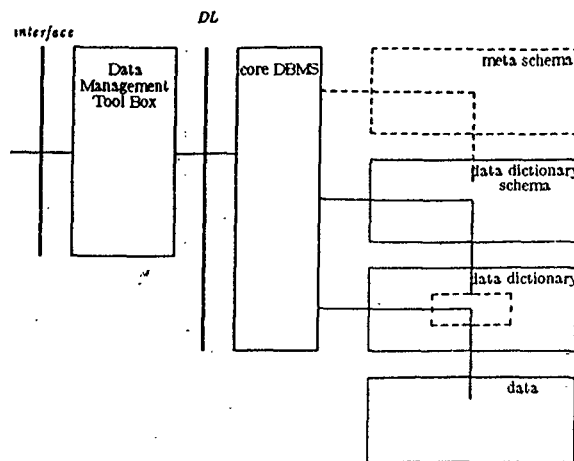


Fig 3 Architecture of a self-describing database system

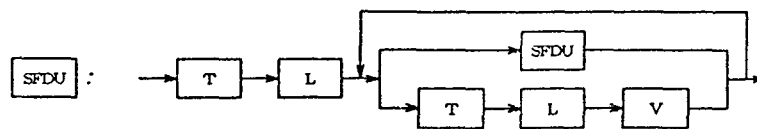


Fig 4 SFDU syntax

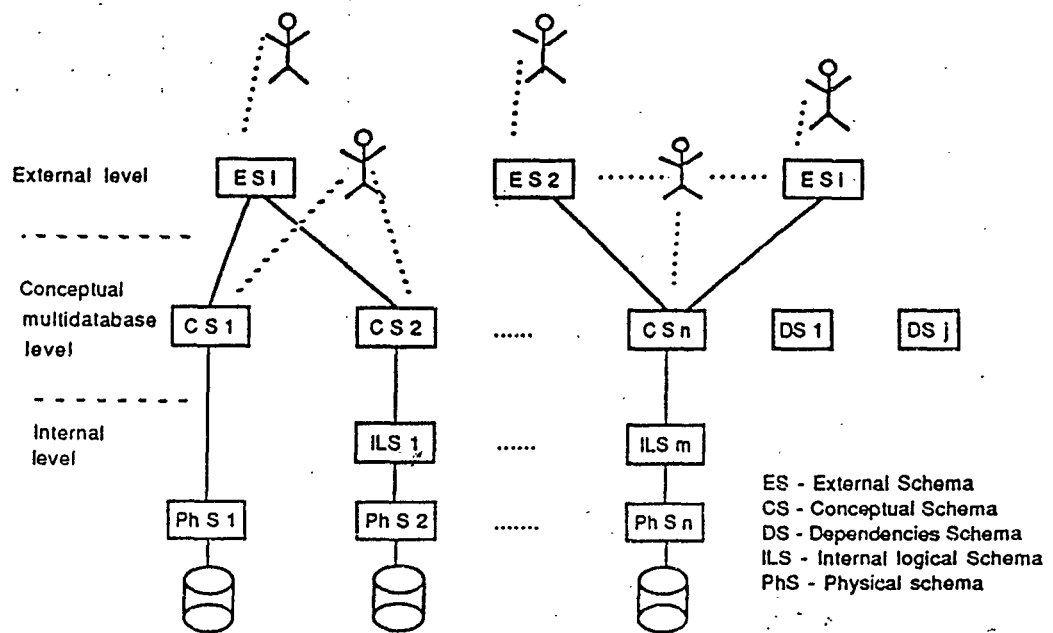


Fig 1 Reference architecture of a multidatabase system

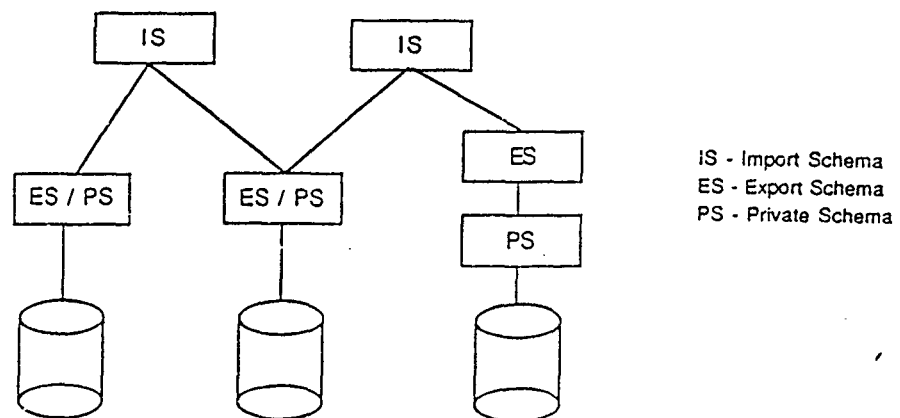


Fig 2 Federated Architecture