TOWARDS DBMSs FOR SUPPORTING NEW APPLICATIONS[1]

S. Abiteboul, M. Scholl G. Gardarin, E. Simon
Verso Project Sabre Project
I.N.R.I.A.
78153, Le Chesnay
France

1. INTRODUCTION

Due to the success of relational systems, there is a growing demand for such database technology, in fields like computer aided design (CAD), image processing or office automation. It seems well accepted that future database systems should support complex objects and inference rules. The purpose of this paper is to present two approaches currently investigated at Inria in that direction, based on the existing Sabre and Verso projects.

Database research at Inria started in the early seventies. The Sirius project (1976-1981) [L+] was focussing on distributed database systems. This project gave birth to two advanced activities in database research: (1) the study on distributed databases was continued within the Sesame project [Li], (2) the Sabre project was initiated in 1980 with the initial objective of developing a performant Database Management System (DBMS) [H]. Concurrently, the Verso project was started with the objective of designing a DBMS based on efficient back-end filtering [BS].

Today, prototypes of the Sabre and Verso DBMS are running under the Unix system. This paper is devoted (i) to the presentation of these two prototypes and (ii) to the outline of the advanced developements based on these systems. The distributed aspects will not be addressed in this paper.

The paper is organized as follows. The Sabre and Verso projects are presented respectively in Sections 2 and 3. Within each section, the project history and objectives are first discussed; the DBMS architecture is then presented, with focus on the novel features; finally the current directions of research are outlined.

[1] This research was partially supported by the french agency ADI, the CNET, and was part of the french joint research program on databases (PRC-BD3).

2. THE SABRE PROJECT

2.1 History and objectives

The Sabre project was developed in cooperation with the University of PARIS.VI. At that time, the objectives of the SABRE project were set as follows :

(1) To develop an extensible and portable relational database manager able to run on classical machines and/or on specific database computers.

(2) To improve response time in comparison with classical relational database systems running on similar configurations.

(3) To allow different groups of users to define and query multiple databases containing real or virtual relations (i.e. views) .

(4) To guarantee the physical integrity against concurrent transactions interferences or system failures, and the semantic integrity against erroneous updates.

Today, objectives (1),(3) and (4) have been achieved [G+, SV2] while we are still working on objective (2). This performance objective, which is probably the most difficult to be realized, led us to develop a new access path organization based on predicate trees [VV] which also supports secondary indexes [CFM]. Measurements show good performances for query processing. Howewer, many other factors affect the database system efficiency, such as the style of programming and the language compiler that we used to compile the Sabre system. Also, to achieve this second objective, we studied a parallel version of Sabre which will encompass several disks performing I/O in parallel attached to one or more processors performing data operations (i.e. selection, join and sort...) in parallel. We still envision a parallel version of Sabre for the future [CFMT] ...

Although all the objectives of Sabre have not yet been achieved, we are already working on two

new objectives :

(5) To enhance Sabre with deductive functionalities in such a way that it could become a useful tool for solving decision oriented problems.

(6) To extend the data types offered by the system towards user defined abstract domains.

In our view, these two supplementary objectives with the previous four objectives will lead us towards a fifth generation database management system, that is a DBMS supporting rules and complex data types in an efficient way.

In the sequel, we first present the functional architecture of the Sabre system . Then, we focus on the most original features of the system which intend to make it an efficient system; these are:

- a multi-attribute clustering based access path model [CFM],
- optimized join algorithms [VG],
- an integrated query optimization strategy [V] ,

We then present the fully assertional and optimized integrity sub-system [Si]. Finally, we present the new developments of the system to support rules and extended data types.

2.2 System architecture

The current architecture of the system is composed of three layers of abstract machines, going from the end-users to the disk units:

(1) The interface machine composes the external layer. It is responsible for the dialogue with the end-users and the parsing of the user requests into internal messages constituting an application protocol called the Data Manipulation Protocol (DMP). Several types of user interfaces can be offered.

(2) The assertional machine which constitutes the intermediate layer of the system performs the evaluation of relational tuple calculus assertions in terms of an extended relational algebra. This machine also includes integrity, view and security controls . It also manages the meta-database of the system as a relational database.

(3) The algebraic machine which is the most internal layer carries out the relational algebra operations as fast as possible. To supply this function, it manages the access path model based on predicate trees [GFMVV], uses a cache memory and implements efficient join [VG] and filtering algorithms . In addition, the algebraic machine performs the physical controls, that is concurrency and reliability controls.

Each machine is divided into functional processors which are implemented as software modules. In the sequel, we introduce the processors of each machine.

The interface machine is composed of one processor for each type of interface. For the time being, it includes a flexible non-procedural language (called FABRE) which is a super-set of both QUEL and SQL, a query by form and example language (called UQBE) and a pre-compiled PASCAL/R like interface (called SINPA). A Prolog interface [J] and an expert system for database design [BGM] are currently under developpement at this layer. The interface machine also includes a set of utilitaries (system initialization, database save and restore commands ...).

The assertional machine includes four main processors :

- The integrity processor performs the integrity control using a specific algorithm [SV].
- The view processor carries out the view to database mapping.
- The request evaluation processor performs an optimized decomposition of each request in a tree of relational algebra operations.
- The meta-database management processor supplies a set of functions to access and update the relations containing the database schemas.

Finally, the algebraic machine is divided into six functional processors :

- The relation access processor manages the access paths to a relation, that is the predicate tree associated with each non sequential relation and also clustered indexes.
- The join, sort and aggregate processor performs join, sort and compute aggregate functions using specific algorithms [VG].
- The localization and storage processor allocates physical space on disk to store logical subsets of a relation and retrieves these logical subsets on disks from logical addresses.
- The concurrency control and recovery processor carries out concurrency control using two phase locking and performs reliability control using shadow page and two phase commit methods.

- The filtering processor performs selection, insertion and deletion of tuples in a disk partition which is a fixed size bucket containing tuples stored sequentially.
- The cache memory processor is responsible for managing the random-access memory which contains temporary and intermediate results of user operations; this memory is extended to disk if not enough RAM is available.

In summary, the functional architecture of Sabre is portrayed figure 1. Let us point out that the machines and processors herein defined are virtual in the sense that they correspond to functions; thus, in non parallel versions of Sabre, they are all implemented on a unique real processor. They may be implemented on parallel processors. We shall not focus here on possible parallel implementations of Sabre. The parallel version of Sabre we are currently working on is described in [CFMT].

2.3 System efficiency

Several features allow the system to process queries efficiently, among them :

- a multiple attribute clustering method,
- an efficient join algorithm,
- an optimized query processing strategy.

We briefly describe these three devices below.

2.3.1 Multiple attribute clustering

One of the major tools developped in the system to improve its performance is a multi-attribute clustering based access model. The method is built upon the concept of predicate tree [VV]. A predicate tree is a balanced tree of predicates; each level of the tree corresponds to a list of contradictory predicates of the form :

function (attribute) = value.

A predicate tree is defined by the data base administrator when creating a relation. It specifies the clustering to be performed on that relation. When tuples are inserted, the relation grows according to the predicate tree : whenever a page is full, it is split into two pages according to the current digit of the hierarchical address (called a signature) of the tuples in the predicate tree.

The predicate tree is used to accelerate retrievals : the predicates in the tree are compared

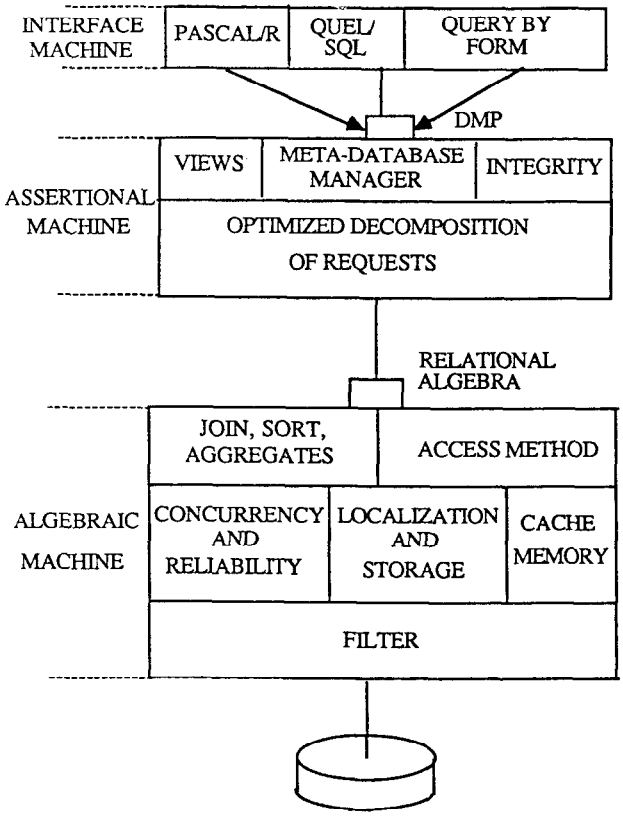

Figure 1 : Sabre architecture

with each selection qualification; when a match occurs, the branch number of the predicate in the tree is generated as the corresponding part of the hierarchical address (i.e. a signature profile); if no match is possible, an unknown address part is generated. Finally, only pages corresponding to the generated signature profiles have to be searched. The search is then performed efficiently by the filter in main memory.

In addition to a predicate tree which directs a relation clustering, the database administrator may specify indexes. Indexes are managed as a clustered relation giving for each indexed attribute value the signatures of the pages containing corresponding tuples. Thus, when an indexed attribute is specified in a query, an index access first gives a list of

signatures to be searched. This list is intersected with the signature profiles obtained from the tree in order to reduce the number of pages to be scanned. A more thorough description of this rather complex but efficient method is presented in [CFM].

2.3.2 Optimized join algorithms

Join algorithms are implemented in the join processor. They work on pages whose content is brought in cache memory. At first, after a paper study and evaluation of three join algorithms (nested loops, sort-merge and hash) in a single and multiple processor environnement, we concluded that each algorithm is the best depending on relation sizes [VG] and has a specific domain of application. Therefore, we implemented the three algorithms in the system. This solution rapidly appeared to be complex and not really optimal.

A new and unique algorithm was then designed and implemented. The algorithm performs joins in two steps. The first step is only activated for relations having a size superior to a ratio of the cache memory size. Such relations are divided into buckets, by using a hash function on the join domain. In the second step, each bucket of one relation is joined with the corresponding bucket of the other relation. For this purpose, a bit array is built by hashing the first smallest relation bucket. For each bit set to one in the bit array, the address of the corresponding tuples are kept in a list. The other relation corresponding bucket is scanned and matched to the bit array. When a match ocurrs, the tuples are compared and if the comparison is successful, the result tuple is written in cache memory. The algorithm proceeds in a similar way for each couple of buckets of the two relations. Measurements have shown the algorithm efficiency [V]. Also, the algorithm fits well with the multi-attribute clustering : when the join attributes are used for clustering, the first step is not neccessary.

2.3.3 Query processing strategy

The query decomposition algorithm of Sabre transforms a query into a sequence of relational algebra operations. The query is expressed in conjunctive normal form. The decomposition method is recursive. At each recursion step, we select the couple of relation instances in the query whose join cost is minimal, if any. Then, based on the simple heuristic of performing selections before join, we apply all possible selections to each relation. The join is then performed. When no join is possible, aggregates and unions are considered and performed if possible. Finally, when recursion is not possible anymore, the query is a mono-relation query or an invalid one.

The method is completely interpreted. Thus, the real size of intermediate results is always known. The sum of sizes of relations to join is in fact our cost estimate; thus, we order join starting with the smallest relations. Of course, each selection is optimized using predicate trees and indexes as described above. Join are optimized by the previously described algorithm. The algebraic machine performs the computation of aggregate functions by using optimized algorithms which give intermediate relations.

2.3.4 Performance measurements

A performance evaluation of the system was conducted on a French SM90 mono-processor machine, in cooperation with the French CNET. The benchmark was performed with the University of Wisconsin benchmark database [Bit]. The results show that the performances are similar to that of the INGRES University system. However, Sabre is not very efficient for queries on key attributes; this is due to the filtering approach combined with the multi-attribute clustering : at least, the system accesses to an index and filters a disk partition to get a unique record. That step requires a minimum of 0.4 second in the current implementation. On the contrary, Sabre is efficient for queries which specify a clustering attribute.

An interesting aspect of the benchmark was an estimation of the time ratio spent in each functional processor. A summary of the results for typical queries of the Wisconsin benchmark is given in figure 2. The ratios show the importance of the filtering process overhead and possibly justifies the amount of work spent to build a hardware filter such as the VERSO filter [BS, ERT].

2.4 Reliability and integrity control

The integrity subsystem provides rich functionalities and high performances. A high level language supports the definition of a large subset of multi-variable multi-relation assertions with aggregate functions. These assertions are expressed into an extended relational tuple calculus. Also, predefined key-words are used to express the most usual structural integrity

TIME PERCENT PER PROCESSOR	REQUEST 1	REQUEST 2	REQUEST 3
INTERFACE	1.0	0.1	1.0
VIEWS	0.1	0.1	0.1
INTEGRITY	0.0	0.0	0.0
OPTIMIZER	8.6	0.6	1.2
ACCESS PATHS	7.3	0.1	0.1
JOIN, SORT	0.0	0.0	85.0
LOC. & STOR.	8.8	2.9	0.5
CACHE MEM.	0.5	3.9	0.1
FILTER	59.2	88.1	11.6
DISKS	14.5	4.2	1.4

REQUEST 1: RETRIEVE MIL1.* WHERE
UNIQUE_1 = 1 OR UNIQUE_1 = 254;

REQUEST 2: RETRIEVE MIL1.* ;

REQUEST 3: RETRIEVE MIL1.*, MIL2.* WHERE
MIL1.THOUSAND = MIL2.THOUSAND
AND MIL1.TEN = 5 OR MIL1.TEN = 6;

MIL1 and MIL2 are the benchmark relations with 1000 tuples each.

Figure 2 : time ratio spent in each processor

constraints (key, entity, referential, functional and inclusion dependencies).

High performance is attained through the use of an assertion simplification method activated at assertion definition time. This method can be illustrated as follows. Let A be an assertion containing tuple variables ranging over several database relations. A is transformed, at compile time, into a set of compiled constraints of the form (R, T, E) where R is a relation involved in A, T is a type of update (among insert, delete, modify) and E is a differential pre-test for A, R and T. A differential pre-test for A, R and T is an optimized pre-condition that the T updated tuples of R must satisfy in order to guaranty the preservation of A. By optimized pre-condition, we mean that the amount of data needed to enforce E is much smaller than the one needed to enforce A, whenever R is updated by T. It is assumed that A is always satisfied by the current database state. This transformation of assertions into compiled constraints is performed by a specific module of the integrity subsytem called "assertion compiler". The design of the algorithms used by this module and their implementation are described in [Si, SV2]. The benefits of the method are the following:

(i) it allows to prevent the introduction of inconsistencies in the database,

(ii) it reduces the number of assertions to be enforced at each update,

(iii) it supports a large class of assertions and multiple tuple updates of a single relation,

(iv) it optimizes the assertion enforcement cost.

The algorithm which enforces integrity assertions is specialized for three classes of assertions [SV]. The main properties of the algorithm are: (i) To generalize the integrity control to general database transactions composed of several updates. (ii) To manage automatic or manual reliable integrity checkpoints. An analysis exhibits the value of the algorithm [Si, SV2]. In particular, the proposed method is shown to be better than the query modification method. Finally, all algorithms are integrated in a modular and extensible subsytem architecture. The other functionalities supported are a constraint manipulation language and the precise management of errors.

The mechanisms described above take great benefit of the update validation technique described in [VM]. This technique is based on a Private Workspace Model for transactions. In this model, a private transaction's workspace is associated with each transaction. This workspace reflects the changes made by the transaction independently of the other concurrent transactions. Furthermore, this technique guarantees that updates executed by a request i are visible by all requests i + j within a same transaction. This last point is very useful for the integrity control algorithms. Concurency control and recovery mechanisms are integrated in a unified way. Reliability is implemented by using an improved variation of the shadow page technique [L]. Finally, at transaction end, all updates are commited by using a two-phase commit protocol.

2.5 New directions of research

2.5.1 Rule support

To allow the system to support complex derived relations, including recursive relations, we are enriching it with a rule definition language. The rules are expressed as a set of production rules of the form :
IF <qualification> THEN <action>[,<action>]...
Qualifications are typical query qualifications while actions are either tuple insertions or deletions. Rules

are grouped into modules which define derived predicates (i.e. inferred relations). A compiled form of rules is stored in a rule base which is a set of relations.

A query of any interface module can refer a derived predicate. In that case, the deduction processor (which is added at the level of the assertional machine) retrieves the relevant rules for the query and performs, in an optimized way, the necessary inference process. The optimization is based on an internal model allowing the system to represent rules and queries in a uniform net [GMS].

Several problems remain to be solved, although a first implementation of the rule support is already operational. Among them are the following :

- rule module consistency,
- rule module commutativity and convergence,
- rule modifications and exception handling,
- improving performances of the inference mechanism using specific algebraic operators.

Another important problem which has been studied is the optimization of recursive rules. The next paragraph introduces our solution.

2.5.2 Recursive Horn clause evaluation

A particular class of rules which are hard to optimize is the class of recursive Horn clauses. We introduce a new method to compile queries referencing recursively defined predicates. The method works for general rules such as the non linear definition of ancestors:

ancestor(x,y) ← parent(x,y)
ancestor(x,y) ← ancestor(x,z) & ancestor(z,y)

The method is based on an interpretation of the query and the relations as functions which map one column of a relation to another column. For example, the query ancestor(c,x?) is considered as a request to evaluate the function a({c}) for a given set of constants c; this function maps the first column of the ancestor relation to the second one. Translating the rules into functional equations, we get :

a({x}) = p({x}) + a(a({x}))

Using Tarski theorem, leads to evaluate the series :

a0({c}) = O
a1({c}) = p({c})
.....
an({c}) = p({c}) + an-1(an-1({c})

whose limit is obviously :

a({c}) = p({c}) + p(p({c})) + ... + p(p(p...p({c}))) + ...

The method lends itself to the specification of polynomial operators to solve each class of rules. For example, linear rules can be solved using an extended transitive closure, called external closure [GM]. Such operators are currently being implemented in the Sabre system. Most of them are special cases of graph traversals.

2.5.3 Extending data types

The current implementation of sabre suppports restricted data types : integer, real and variable length character strings. Our current approach for extending these data types is based on a LISP interpretor : we enrich the system with a new type "list" implemented as a LISP binary tree. Domains of this type may be manipulated through functions, including the basic LISP functions CONS, CAR, CDR, a library of supplied functions as APPEND, REVERSE, MEMBER ... and also database administrator defined functions. The functions may be used at the level of the non procedural languages as a modifier applied to an attribute name, using the dot notation. This approach which seems very powerful and which does not imply strong modifications in the system is currently under investigation.

2.6 Conclusion

The development of a complete DBMS as Sabre, with a sound kernel based on software filtering of disk partitions , an original multi-attribute clustering and a tuple relational calculus including functions, is a huge task. One strenght of the system is its division into abstract machines and processors. This architecture was elaborated to develop a multi-processor system. Surprisingly, this allows us to develop new functionalities without changing the whole system.

The experience shows us many difficulties to achieve the multi-processor goal, among them the lack of multiprocessor operating systems able to support the DBMS in an efficient way.However, multiprocessor operating systems are rapidly evolving. That is why we still think about developing a parallel version of Sabre, as presented in [CFMT].

3. THE VERSO PROJECT

3.1 History and Objectives

The VERSO project was started at Inria, in the early eighties [BS], with the following objectives in mind:

- Justify the approach consisting in relegating some tasks to a processor close to the mass storage device under the conventional assumption that Database Management Systems (DBMS) are I/O bound.
- Check that an automaton-like mechanism for this on-the-fly filtering capability is well adapted to query processing.

The major motivation behind such an architectural approach is to increase the performance of a relational DBMS. Although the usefulness of on-the-fly filtering has been widely accepted (see, for example [H]), no filter has been included in a complete DBMS design to our knowledge. Our intention was therefore to develop a fully relational system that would use the above filtering concept. To take full advantage of the filter, it was decided to store data in hierarchical structures. This physical organization strongly suggests a logical data organization into non first normal form (non 1NF) relations called V-relations. An algebraic language for non 1NF relations is then used to manipulate data [AB].

3.1.1 Non 1NF relations

Existing relational database systems already provide high level query languages like SQL or Ingres which are relatively easy to use. They answer at least partially the data accessing needs of many applications in areas like business administration. Due to the success of relational systems, there is a growing demand for such database technology in other fields like computer aided design(CAD), image processing, VLSI, or office automation. The relational model proposes to represent the logical structure of data in tables, providing data dependencies as the only means of defining more precise semantics. Unfortunately, this approach is too simplistic for the new applications. For instance CAD-systems often manipulate hierarchical data structures with many levels. The flattening of these data structures is not satisfactory neither from a performance point of view, nor for conceptual reasons.

These limitations of the relational model come essentially from the so-called first normal form assumption which states that values in a relation should be exclusively atomic. The model used in the Verso system does not make that assumption, and is therefore better suited for the new database applications. Indeed, several researchers have studied this concept of non 1NF relations [e.g., AB,FT,SS]. It should also be noted that this notion of hierarchical data arises naturally in the context of semantic database modelling [AH1]. However, the Verso system was, to our knowledge, the first implemented system based on non 1NF relations.

The query language is algebraic. All algebraic operations (except for one, namely restructuring) are performed by the filter. This filter can be viewed as a finite state automaton (FSA) which scans sequentially one or two input buffers, and writes the result of the operation on an output buffer. The restructuring operation involves some sorting, and cannot be realized uniquely by the filter. The performance of the system thereby depends heavily on the performance of the filter, and on its connection to the rest of the system.

3.1.2 Filtering and Performance

The version of the system presented here, runs under the Unix operating system. Prototypes have already been experimented on a 68000 based multiprocessor machine, the SM90. Most of the code is written in Pascal. A specialized hardware processor was first designed to realize the FSA filter [B+] and developed by the Inria SCD team . This hardware processor, connected to the mass storage as well as to the central bus was in charge of data transfer and data filtering. Later on, the hardware filter was abandoned and replaced by a standard disk exchange module including an Intel 8086 processor on which filtering is implemented by software.

A performance evaluation work was conducted which focussed on the problem of choosing between these two competitive approaches for implementing a performant relational DBMS. At the time of this performance study, no real life measures were available, and modelling was used for evaluating the filter's response time to a query in both architectures [G,GS,S]. The result of the comparison was that software filtering should provide an acceptable performance, although very inferior to that of hardware filtering. Later on, when the system was operational with a software filter, the system was tested against a benchmark provided by the french ADI agency, and a benchmark designed at the University of Wisconsin

[Bit]. Two conclusions on software filtering were drawn from these experiments.

- a comparison between the (measured) software filter's response time and the (predicted) hardware filter's response time reported in [GS] is attempted in [S,JV]. To summarize, the hardware filter should be extremely faster (more than 20 times faster) than the current filter implemented by means of an "off-the-shelf" Intel 8086 processor.
- despite the mediocrity of the Intel 8086 processor , the Verso system's performance is acceptable, compared to that reported in [Bit] of other DBMS such as Oracle or Ingres implemented on VAX 11/750 computers.

In the following sections, we describe the Verso data model and system. A thorough presentation may be found in [JV]. Except for the use of a filter, and for the model of V-relations, the Verso system is a quite standard system: the data is stored in relations contained in databases; secondary indexes are not implemented; concurrency is offered via the concept of transaction, and managed using two phase locking; mechanisms for handling crash recovery are provided.

3.2 The System

3.2.1 The Model

In this section, we briefly describe the Verso data model. We first describe the data structure called V-relation. We then present the Verso algebra. A formal presentation of the model, together with some basic results on V-relations can be found in [AB,Bi].

In the Verso data model, the data is organized in non-1NF relations called V-relations. In a V-relation, the values of some attributes are atomic whereas the values of other attributes are V-relations of simpler structure. An example of V-relation is given in Figure 3.1. The first line of the figure represents the structure or *format* of the V-relation.

This database describes information about movies, theaters, times, and actors. Note that:

(1) for the movie "Karate Night", there is no known schedule, and no known actor. Thus V-relations handle null values in a simple manner. As a consequence of this, some queries which are typically complicated to be expressed in the relational model are simple selections in this model. An example of such a query is: "Give all movies with no known schedule".

(2) the data is naturally organized in a hierarchical manner. (It is possible to speak of the schedule of a movie in a theater.) Furthermore this hierarchical data organization induces some implicit connection between attributes. For instance, in this example, there is a connection between theater and actors through movie.

MOVIE	(THEATER	(TIME)*)*	(ACTOR)*
Straw Dogs	Rex	18	D. Hoffman
		20	S. George
		22	P. Vaughan
	Chinese	18.30	
		20.30	
		22.30	
Metropolis	Studio3	16	B. Helm
		18	R. Abel
			G. Frolich
			R. Kleinregge
Pierrot le Fou	Studio3	20	J.P. Belmondo
		22	A. Karina
Karate Night	Studio3		

Figure 3.1: example of V-relation

A simple algebra can be defined for V-relations. As mentioned above, all algebraic operations but one can be computed by the filter. The unique "expensive" operation is restructuring. This operation involves some sorting. Thus, the complexity of main memory computation is restricted to a unique module, namely the sorter.

The algebra consists of unary and binary operations. The unary operations are projection, selection, renaming, and restructuring. The binary ones are join, union, and difference. Examples of unary operations are now given. These queries are expressed here in natural language.

Example 2.1: The following projection/selection can be performed on the MOVIE database: "who are the actors playing in a movie shown at the Rex between 7:30 and 8:30 featuring J.P. Belmondo and A. Karina, or at the Chinese theater after 10:30?"
(Note that the previous operation would involve several joins in the relational model.)

Example 2.2: The following restructuring can be performed on the MOVIE database: "give the list of

theaters, and for each theater, the movies that are shown there". Note that some information may be lost when restructuring data. Even if the loss of information is tolerated (which is typically the case in queries), some restructuring operations have no meaning. For instance, it is not possible to restructure a flat relation (A B C)* into (A(B)*(C)*)*. A thorough study of *lossy,* and *lossless* restructuring of V-relations is presented in [AB].

Union allows to "add" the information of two instances. Join allows to "combine" the information of two instances. Finally, difference is used to withdraw the information of one instance from the information in another one. In that sense, these three operations can be seen as generalizations of the (pure) relational operations of union, join (and intersection), and difference. It is not possible to apply binary operations to relations of arbitrary structures. The two relations involved have to be *compatible.* (See [AB] for a formal definition of comptibility.)

The Verso language is used as *the* communication language between the system, and the rest of the world. Indeed, other interfaces can be viewed as translation modules between more user friendly interfaces and the Verso language. Users can use the Verso system directly in the Verso language, from Pascal programs, or through a screen interface.

The Verso language provides commands for handling transactions, and within a transaction for data definition, and manipulation. The screen interface called Ever (for Editor of Verso Relations) is provided for non sophisticated users. Ever is a multi window screen interface tailored to answer the various needs of a dialogue with the Verso system. Four modes are offered: a mode for command edition, a mode for selection/projection, a mode for data edition, and a mode for format manipulation. The data mode, for instance, is used for browsing through V-relations, and for updating. Except for the particular nature of the data, the editing of V-relations ressembles the editing of text in a conventional editor like Vi or Emacs.

The third interface, V-Pascal, is a Pascal extension which combines the advantages of the Verso system, and that of the Pascal programming language. The major problem raised by this type of interface is that Pascal does not allow the definition of structures like V-relations. Therefore, in the V-Pascal interface, V-relations are viewed through a strict relational view. This is clearly not satisfactory from a logical point of view, but has been developed mainly for being able to realize applications on top of Verso, and to gain experience in the embedding of database features in conventional programming languages.

3.2.2 The Architecture

We first present the hardware architecture, then we give an overall description of the DBMS. The version of the system presented here runs on the Unix operating system and has been experimented on a 68000 based multiprocessor machine, SM90.

As shown in Figure 3.2, the machine includes the following components, which share the central bus, the SM bus:

a) a central processing unit(CPU) including a Motorola 68000 processor, its local memory and a memory management unit;

b) a RAM memory;

c) an exchange module (EM) interfacing with a disk hosting the Unix system and the programs;

d) an user interface (V.24 or Ethernet);

e) another EM interfacing with another Disk where the databases are stored. Filtering is implemented on this EM.

The CPU is in charge of the user interface, the high level DBMS layers to be described below and the filter's control: it sends to the filter data transfer and filtering commands (see [JV] for a description of the filter internal structure) .

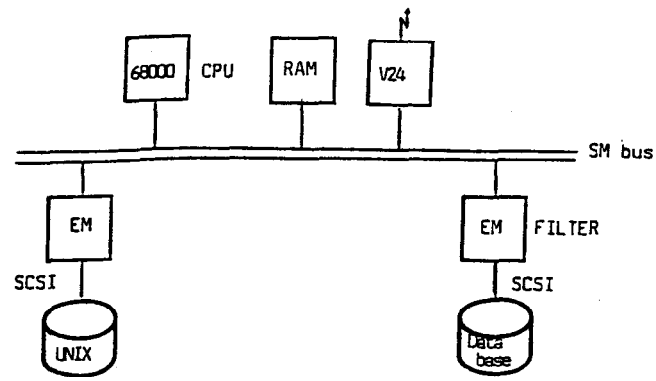

Figure 3.2: Verso hardware architecture

In terms of functionality, Verso is a fairly standard system: it offers data definition, search and manipulation, transaction management, concurrency control and recovery and simultaneous access from separate sites .

The three latter functions will only be roughly sketched, since classical solutions have been chosen for those problems. The interested reader is referred to [B+] for more details.

As usual, a transaction is a sequence of requests. The system accepts interleaving of requests issued from different transactions, but requests are sequentially run. In order to improve the global throughput, pipelining of requests on a single CPU is under study.

A regular two-phase locking protocol is used, together with deadlock prevention. Physical locking has been chosen with granularity of one block (one disk track). However the index is locked only for the duration of the index request (and not until the end of transaction, as for a regular data access).

We will describe in more details the data search and manipulation functions.

The system consists of three layers:

1) The highest level is the V-relational level: the objects seen at that level are the V-relations and the schema.

2) The second level is the file level: the objects defined at that level are Verso files, or physical representations of V-relations and the non-dense Index which permits to locate data. A file is a set of pages or blocks which are not necessarily contiguous. The operations at that level are:
   i) index manipulation in order to locate a V-relation,
   ii) selection/projection, insertion, deletion into/from a file (corresponding to a unary operation on V-relations);
   iii) binary operations on files (corresponding to a binary operation on V-relations);
   iv) file sort (corresponding to restructuring). We use a merge-sorting algorithm: once each block has been sorted, blocks are merged. This merge is a file union performed in linear time by the filter.

3) At the lowest level, we find a block characterized by its address. There are two kinds of operations at the block level: filtering and internal sort of a block. As mentionned earlier, this operation is not performed by the filter.

Let us take the example of the V-selection to illustrate query processing through the three layers as well as the splitting of tasks between the CPU and the filter. A V-selection is submitted to the system. At the first level, given the V-relation name ,the schema is searched to get the V-relation format. Two operations are then performed:

i) compile the query into an FSA to be loaded into the filter memory (LM);

ii) search the index in order to get a subset of the blocks of the V-relation that have to be filtered. The result of this index search is a list of one or more block addresses.

The above processing is performed by the CPU. Once the FSA corresponding to the query has been loaded into the filter's memory , the filter starts processing a set of blocks. Except for input and output blocks of data, no data are transferred through the central bus or main memory.

To summarize, V-relation operations are performed by the CPU, including transaction management and concurrency control. The CPU is also in charge of Index operations, as well as FSA generation and loading. The filter is in charge of file and block level operations on data (except internal sort of a block). Binary operations can also be performed in linear time by the filter since the files are sorted.

We end up this description with a few words on filtering. Recall that the filter sequentially scans a source buffer and writes into a target buffer the relevant data. In the case of insertion or binary operations, two source buffers are concurrently scanned. Three processes are pipelined: (i) loading the source relation into a source buffer, (ii) filtering another source buffer previously loaded, (iii) unloading a target buffer either onto disk or to the user.
The FSA filtering principle has been thoroughly described in [BS,BRS]. It was shown in [BS,S] that an automaton-like device is sufficient to perform on the fly the V-algebra operations. The reader interested in a detailed description of the filtering mechanism is referred to [BS,JV].

3.3 Conclusion and New Directions of Research

With respect to more classical relational DBMS designs, Verso major novel features are the following:

1) It includes a filter implemented on a separate processor close to the mass storage device. This filter is in charge of all algebraic operations except for restructuring. This automaton-like mechanism is extremely well adapted to processing of both unary, and binary operations. Furthermore, the filter is also used for providing fast updates.
2) Data is organized in non 1NF relations. This allows to combine the advantages of the relational model (e.g., an algebraic language), and the possibility of hierarchical data organization. To our knowledge, the Verso system is the first running system based on non 1NF relations.

The first response time measurements clearly show that the Verso system is not faster than commercial systems such as Oracle or Ingres. The main reason is that the 8086 microprocessor on which filtering was implemented is slow. By using dedicated hardware for filtering, one should gain two orders of magnitude on response time. However, standard microcomputers have a performance that increases rapidly with time. For that reason, following [BD], we believe that the use of "off-the-shelf" components for filtering should be preferred to a time-consuming and costly design of dedicated hardware.

Besides, this first experience with a non 1NF model is quite promising: users seem to adjust quite fast to those more complex structures. For instance, it turned out that although the Verso language was not intended to be user friendly, it didn't require too much practice from the user to be capable of writing even complex queries in that language. Not surprisingly, the screen interface Ever has been quite an improvement for users.

The Verso model is based on non 1NF. In this model, set and tuple constructors are used alternatively to construct higher order relations. It is assumed that at each level at least one attribute is atomic, and furthermore that this attribute forms a key for the relation. Although these restrictions are useful for implementation reasons (they form the basis for the functionning of the filter), they are certainly not logically needed. We are actually looking at some query languages for typed objets where the type is defined using set and tuple constructors in an unrestricted manner [AB].

Various possible extensions should also be considered like:

- union of type (e,g, an object is of type either A or B) which can roughly be seen as a variant record in Pascal [AH1,AH2].
- unknown values (e.g., not applicable nulls) [AH2],
- lists, and
- data strutures with possibly recursive type definitions (e.g., the Unix dictionnaries).

Tomorrow's database systems should provide such logical data structures. The challenge is to incorporate them elegantly in query languages. This motivates again (if necessary) the need to abandon the strict relational model: the simplistic data structure it uses makes that model inappropriate for embedding query constructs in powerful programming languages. In particular, the use of more powerful data structures should facilitate the incorporation of the database paradigm in classical functional programming, or logical programming approaches.

These more elaborate data structures, and languages may be restricted to the external level. We intend to develop some new layers on top of the Verso system which will provide all the new functionalities. As mentioned above, it is clear that a standard relational system would be inappropriate for such development. We believe that the Verso system allowing to directly manipulate hierachical data provides the minimal kernel on which to base future development.

Other fields of interest of the group include form manipulation [RB] and image databases. The group objectives are twofold:

1) objects modelisation: to look for new data structures, and query languages adapted for the manipulation of spacial objects and office forms.
2) experimentation: we intend to develop new applications based on these complex objects on top of the Verso system, and check whether the current system is suited for such applications.

REFERENCES

[AB] Abiteboul S., Bidoit N., "Non First Normal Form Relations to Represent Hierarchically

Organized Data", Proc. of ACM-SIGMOD Conf. on Principles of Database Systems, Atlanta, 1984, pp. 191-200 (To appear in Journal of Computer Science and Systems).

[ABe] Abiteboul s., Beeri C., "On the Power of Languages for the Manipulation of Complex Objects", in preparation.

[AH1] Abiteboul, S., R. Hull, "IFO, a Formal Semantic Database Model", Proc. of ACM-SIGMOD Conf. on Principles of Database Systems, Waterloo, 1984.

[AH2] Abiteboul, S., R. Hull, "Restructuring Complex Objects and Office Forms", Proc. of International Conference on Database Theory, Rome, 1986.

[B+] Bancilhon F. et al, "VERSO: A Relational Back End Data Base Machine", IWDM proceedings, San Diego, Sept. 1982 ; also in [H].

[BD] Boral H.,DeWitt D.J., "Database Machines: An Idea whose Time has passed. A Critique of the future of Database Machines", in Database Machines, H.O. Leilich and M. Missikoff editors,Springer-Verlag,1983 pp 166-187.

[Bi] Bidoit N., "Un Modele de Donnees Relationnel Non Normalise: Algebre et Interpretation", These 3e cycle, Universite Paris-Sud, 1984.

[Bit] Bitton D. et Al., "Benchmarking Database Systems: A Systematic Approach", Computer Science Department, Technical Report no 526, University of Wisconsin, December 1983.

[BRS] Bancilhon F., Richard P., Scholl M., "On Line Processing of Compacted Relations", Proc. Inter. Conf. on Very Large Data Bases, Mexico, 1982.

[BS] Bancilhon F., Scholl M., "Design of a Backend Processor for a Database Machine", Proc. ACM-SIGMOD, Santa Monica, May 14-16, 1980, pp. 93-93g.

[BGM] Bouzeghoub M., Gardarin G., Metais E., "Database Design: An Expert System Approach", Int. Conf. on Very Large Data Bases, Stockholm, Aug. 1985.

[CFM] Cheiney J.P., Faudemay P., Michel R., "An extension of access paths to improve Joins and Selections", 2nd. Int. Conf. on Data Engineering, Los Angeles, Feb. 1986.

[CFMT] Cheiney J.P., Faudemay P., Michel R., Thevenin J.M., "A reliable Multiple Backend using Multi-attribute Clustering", Int. Conf. on Very Large Data Bases, Kyoto, Aug. 1986.

[ERT] El Masri A., Rohmer J. et Tusera D., "A Machine for Information Retrieval", Proc. 4th Workshop on Comp. Arch. for Non-numerical Processing, Syracuse, New York, August 1978.

[FT] Fisher, P., S. Thomas, "Operators for Non-First-Normal-Form Relations", Proc. of the 7th International Comp. Soft. Applications Conf., Chicago, 1983.

[G] Gamerman S., "Où l'on decouvre que les performances des Filtres dans les Machines Bases de Donnees ne sont pas celles que l'on croyait", These de 3e cycle, Universite de Paris-Sud, Juin 1984.

[GS] Gamerman S., Scholl M., "Hardware versus Software Data Filtering: The Verso Experience", Proceedings of the Fourth International Workshop on Database Machines, Grand Bahama Island, March 6-8, 1985, pp. 112-136.

[GFMVV] Gardarin G., Faudemay P., Michel R., Valduriez P., Viemont Y., "An integrated approach to Multi-Dimensional Searching Using Predicate Trees and Filtering", INRIA Internal Report, Dec. 1984.

[G+] Gardarin et al., "SABRINA: Un Systeme de Gestion de Bases de Donnees Relationnelles issu de la recherche", to appear in TSI-AFCET-DUNOD, 1986.

[GMS] Gardarin G., De Maindreville C., Simon E., "Extending a relational DBMS towards a Rule Based System", Int. Workshop on AI and Data Bases, Creta, June 1985.

[GM] Gardarin G., De Maindreville C., "Evaluation of Data Base Recursive Logic Programs as Recurrent Function Series", Int. ACM-SIGMOD Conf., Washington, May 1986.

[H] "Advanced Database Machine Architecture", D.K. Hsiao Editor, Prentice-Hall 1983, pp.36-86.

[HY] Hull, R., C. Yap, "The Format Model: A Theory of Database Organization", Journal of the Assoc. for Comp. Machinary, 1984.

[J] Jouve M., "Etude du couplage de Prolog et d'un systeme de gestion de bases de donnees: Application

au SGBD SABRE", Thèse de 3e cycle, Universite de Paris VI, Dec. 1985.

[JV] Jules Verso (pen name for the Verso team), "Verso: a Database Machine Based on non-First-Normal-Form Relations", Inria Research Report, No 523, May 1986.

[L+] Lebihan J. et al., "SIRIUS: A French Nation-wide Project on Distributed Data Bases", Int. Conf. on Very Large Data Bases, Montreal, Oct. 1980.

[Li] Litwin W. "Concepts for multidatabase manipulation languages", COMPDEC, Los Angeles, May 1984.

[RB] Richard P.,G.Barbedette, "An object oriented approach for form management", in preparation.

[SS] Scheck, H.-J., M. Scholl, "The Relational Model with Relation-Valued Attributes", Information Systems, Vol. 11, No 2, 1986

[S] Scholl M., "Architecture pour le Filtrage dans les Bases de Donnees Relationnelles", Thèse d'Etat, INPG, Grenoble, 1985.

[S+] Schweppe H. et al., "RDBM - A dedicated Multiprocessor System for Database Management", Proc. of the 7th International Workshop on Database Machines, San Diego, September 1982, also in [H], pp.36-86.

[SV] Simon E., Valduriez P., "Design and Implementation of An extendible Integrity Subsystem", Int. ACM-SIGMOD Conf., Boston, June 1984.

[SV2] Simon E., Valduriez P., "Design and Analysis of a Relational Integrity Subsystem", submitted to publication (50 p.), 1986.

[Si] Simon E., "Conception, analyse et realisation d'un sous systeme d'integrite relationnel", Thèse de Doctorat, Univ. Paris VI, 1986.

[V] Verlaine L., "Optimisation des requetes dans une machine base de donnees", These de 3e cycle, Univ. of Paris VI, 1986.

[VG] Valduriez P., Gardarin G., "Join and Semi-join algorithms for a Multiprocessor Database Machine", ACM-TODS, Vol.9, N.1, March 1984.

[VV] Valduriez P., Viemont Y., "A multi-key Hashing Scheme using Predicate Trees", Int. ACM-SIGMOD Conf., Boston, June 1984.