tanulmányok

198/1987

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest

Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutató Intézete
Computer and Automation Institute Hungarian Academy of Sciences

A DATA BASE MANAGEMENT SYSTEM DEVELOPED FOR THE CUBAN MINICOMPUTER CID 300/10

MIGUEL FONFRIA ATAN

Tanulmányok 198/1987
Studies 198/1987

Candidate Dissertation

Advisor

DR. DEMETROVICS JÁNOS

A kiadásért felelős:

DR. KEVICZKY LÁSZLÓ

ISBN 963 311 228 1

ISSN 0324-2951

I want to express my gratefulness to the staff of SZTAKI's Computer Science Division, to Dr. Benczúr András, which colaborated with me in the performance of this work, and specially to Dr. Demetrovics János, my advisor, which observations and recommendations were very useful in the approach and design of the research.

I also want to testimony my acknowledgement to Lic. M. E. Bragado for the support that she gave to me during these years.

Index

Introduction.

The power of hardware has been incremented revolutionary in the latest years: the availability of faster microprocessors, cheaper RAM, and more hard disk storage propiciate the explosive development of microcomputers.

This development besides the technical advances issued from Codd's works about relational data base /CODD70/ promote that Data Base Management Systems had been introduced in all actual computing means with their acknowledged profits.

It is stated that about a 50 % of the uptime of microcomputers in the world is applied in some DBMS and almost all minicomputers and mainframe use a Data Base System.

The relational model was developed to solve the problem of data base rigidity. The relational system allows the user to be unaware of physical links and he can manipulates relations between data, answering in an easy way the data change necessity.

In the microcomputer industry it is applied the relational term to a different kind of programs, making this term almost without sense. Around 1980 these programs invaded the market and in 1981 there are references about more than 15 of these systems /BARLE81/. In a certain way they are used in the most of data processing works, allowing to use a more evolved and flexible technology to develop applications.

Analizing this data processing development it was decided to

implement a DBMS in the cuban minicomputer CID 300/10.

From this decision it was developed an evaluation of the available systems in order to its possible implementation.

In this work it is shown how the data processing in Cuba has been evolved toward the application of data base techniques (chapter 1).

In chapter 2 it is shown the evaluative study developed about the two DBMS available in the moment of doing this work. It is presented a particular methodology to develop the evaluation from the criteria that the goal of the evaluation is not only to select a DBMS for its application but also taking in mind its implementation problems.

Besides it is shown a chapter with the main characteristics of the implementation of the selected DBMS. It was studied the different actual methods of implementations and it is justified the selected ones.

At the end it is presented one of the application that actually executes over our DBMS. This application was originally developed for microcomputers and later it was moved to our system. Besides, it is mentioned the main differences between our DBMS and the selected pattern.

1. Background

The elements needed for an objective assessment of computing media, and their strategic importance, are evident from the significance of information in our time --it having become an industrial product. It is precisely in this sense, and especially to the underdeveloped countries, that achievement of full and final economic, scientific-technical and technological independence assumes a cultural connotation of a new kind: absolute sovereignty over their national sources of information is essential, and also their autonomous capability to obtain information. From this it is concluded that development of an infrastructure for automated processing, teletransmission, storing and retrieving of information is a necessary condition for full economic and social development in these countries. The strategies and policies followed by the Cuban Party and Government in the field of information show the way to development of computing techniques, having as a goal the achievement of high economic and social ends. Specially significant are those related to development of applied research, production and promotion of scientific cadres with a high political and ideologic level. An adequate institutional organization of computing activities, the necessary investment processes in the industrial, research and development spheres, the most advantageous alternatives of international

specialization with a view to the production and establishment of exportable funds, as well as the principal sectors and economic branches for priority development of automated systems of management have been set up in advance. This all constitutes a structured process corresponding to medium and long term plans of economic and social development of the country. This is a logical consequence of the fact that the plans are the product of an undivided dedication of the national government to ensuring prompt and harmonious growth of the country's economy.

The following is a panoramic view of the historical development of computing in Cuba. It is explained the characteristics of the Basic Software evolved in our Institute, which it is regarded as the logical background for our present thesis. Lastly, it is described the characteristics and levels of our users who are particularly influential on the conditions of our software.

1.1 Historical development of computing techniques in Cuba.

1.1.1 Before the Revolution.

The first data processing equipment was received in Cuba around the year 1930, when the data of the population census were processed in equipments based on punched cards. Since that time the U.S. firm IBM established a branch in Cuba to promote the use of electromechanical equipment for accounting, calculating and adding processes.

Development of data processing techniques at that time was entirely dependent on specialized entities belonging to foreign companies.

There was a small number of high level specialists on systematization of the work with data processing equipment using punched cards and accounting.

1.1.2 Since the Revolution.

The first computers introduced in Cuba were one UNIVAC and one RAMAC, both of first generation. They were installed during the first few years following the 1st. January, 1959. The work with these equipments acquainted the few punched card technicians remaining in the country with this change in technology --both with the hardware and with its application.

In 1963 a second generation British computer, the ELLIOT 803-B was acquired, and with this the formation of new computing specialists was started /PEDRO82/. Some university faculties began teaching programming and the first groups were formed in several institutions for the purpose of practical application.

During the period 1963-1968 various attempts were made to structurally organize the development of computing techniques.

The economic blockade of Cuba enforced by the U.S. Government, which included the scientific-technical field, was in full force.

However, two second generation computers known as SEA-4000 were acquired from the Compagnie Internationale pour l'Informatique

(C.I.I.) to make the population census in 1970.

An entity was formed in 1969 by the name of "Plan Calculo", as a part of JUCEPLAN, the Central Planning Board. The "Plan Calculo" was charged with the responsibility to manage, regulate and control the introduction of technical computation media and their applications in the national economy. A large program of investments was set in motion and the formation of computing specialists was given new impetus.

Also in 1969 our leader, Commander-in-Chief Fidel Castro launched the project of a small computer that could be used in certain branches of the economy /LOPEZ1/. To materialize this initiative a research group was formed whose core was made up by investigators who were already working on the design of Digital Systems. These investigators were professors in the Department of Electronics of Havana University's Faculty of Technology. The first Cuban computer CID 201, was the crowning success of this work, one year later.

This result was not regarded as a mere scientific-technical achievement. It was only a first step towards giving answers to urgent needs for application of electronic computing in Cuba. Besides, it was essential to train in the shortest possible time specialists who should be able to apply computing techniques in the country.

CID 201 had a word of 12 bit, and the main store contained 4096

words. The processing rate was of about 20000 additions per second.

With the initial group of designers and the addition of university graduates and undergraduates the institution called Centro de Investigacion Digital (CID) was formed, which was the inception of today's Instituto Central de Investigacion Digital (ICID).

The success achieved by the construction of CID 201A brought about the necessity to develop basic software for its exploitation in the various fields of the economy and in the centers of scientific-technical research. This undertaking was something unheard of in Cuba since imported hardware had only been used so far having been furnished with adequate basic software.

That time the software was written in machine code without the possibility of using an assembler language or an operating system. A consequence of this was low programming productivity. On the other hand, however, the personnel that went through this stage learned all that was indispensable to obtain profound knowledge of the equipment's architecture and this was helpful in the further development of their professional activity.

In 1973, as a result of a research for increasing efficiency and computing potential, model CID 201B was introduced. With a set of instructions similar to that of CID 201, CID 201B enabled a five-

fold increase of the rate of operations, and was extensible up to 32K words of operating memory by 4K modules.

From that time on a period of development commenced in which the basic aims were the following:

1. To develop the production basis of CID 201B.
2. To develop Basic Software for CID 201B.
3. To procure the input/output equipments for CID 201B.

Various Operation Systems (OS) were implemented, depending upon the available configuration. An OS was made on paper tape for configurations not provided with magnetic storage to control the loading the various components of the program packages; and, in addition, to afford a basic environment for control of the system's resources.

In the same manner Operation Systems placed on magnetic tape, data cassettes and minidisks were made. For these OS compilers for LEAL, FORTRAN, ALGOL, COBOL, BASIC and FOCAL languages were implemented.

All these software tools were implemented without having any auxiliary software support such as assembler, macroassembler, linker, debugger, etcetera.

This period was extremely helpful for the software specialists inasmuch as all the work had to be done from scratch. This situation forced them to explore deeply the more advanced techniques and theories so far developed on the subject.

From the beginning, the hardware systems were applied mainly in economic data processing. Correspondingly, the basic software oriented to this kind of application has been mostly developed in our Institute.

It was decided to develop a COBOL compiler on CID 201B in order to facilitate business applications. As an intermediate development a package of subroutines was implemented which, on being called from assembler language, could make up for the lack of something more highly developed. This package included decimal arithmetic coded in binary, comparison of and operations with alpha-numerical chains, etc., but its use was restricted due to the fact that it is difficult to implement any business application on the level of an assembler language.

The COBOL compiler for CID 201B included a file management system for sequential and direct access on magnetic tapes and magnetic disks.

1.1.3 From 1976 to 1982

On 30th. November, 1976 a decision was made to establish the INSAC (Instituto Nacional de Sistemas Automatizados y Tecnicas de Computacion). This is an entity of the State central Administration and is charged with directing, executing and controlling the application of State policy in this activity.

INSAC has a system of enterprises and specialized units that perform the following functions:

- Installation, repair and technical maintenance service.
- Project making and assembling of computer centers.
- To deliver mechanized and automated data processing services.
- Production of computer hardware.
- Design and implementation of automated systems and tasks.
- Research and development of computation techniques and automated systems.
- Formation and re-qualification of specialists.

Since its inception, INSAC gave support to the work of CID, having strengthened the basis of research, development and production of hardware and its corresponding software.

In this period, a new minicomputer system, CID 300/10, was designed, This was a much more powerful system than CID 201B, and was based on the architecture of the family PDP-11 of DEC. Development of the CID 300/10 system took place observing all the requirements of Intergovernmental Commission for Computation (ICC) and now it corresponds to the standard of System SM-3. It was an achievement of multilateral collaboration among the Socialistic countries and successfully passed the international tests late in 1978. There was code number SM-3 given to it in the unified nomenclature of the Socialistic countries.

System CID 300/10 went into production in 1980 and it is the basic hardware for the development of applications of most users of computation techniques in the country.

For nationwide distribution the FOBOS system was adopted --a Socialistic version of DEC's RT-11, as the more powerful systems could not work properly on the 56k bytes of CID 300/10.

Although an OS, efficient an fairly complete, was available, it was pointed out that its utilization was not contemplated for data processing (business application) --an application which since the earliest days has been fundamental in the spectrum of applications of computers in Cuba. It was therefore decided to write a compiler of the COBOL language for the FOBOS Operation System. This decision was based on the experience gained by the specialists who developed the COBOL compiler for CID 201B and on the good results obtained.

For this compiler the standard used was that corresponding to Norm ANSI X3.23 - 1974 and it was submitted within the frame of ICC for approval in joint tests conducted late in 1979, the USSR and Czechoslovakia having figured as co-makers. This compiler has been widely used in our country with satisfactory results and has been the basis for further basic program packages oriented towards data processing.

Based on this COBOL compiler for FOBOS a Base System for Business Applications was evolved which included a monitor that enabled the execution of several COBOL tasks simultaneously. This system was complimented with utility programs for data sorting, creation, validation, condensation and conversion, as well as

automation of report issuing, file listings, file updating, etc., affording a very comfortable environment for data processing work.

1.1.4 From 1982 to date.

Since the early years of the current decade a certain number of microcomputers of 8 and 16 bits have been introduced in the country, forcing us to give considerable thought to the matter of the road to be taken concerning the national production of hardware, inasmuch as before the introduction of microcomputers the tendency was to continue to increase the power of our computing systems using minicomputers and increasing their speed, their memory and the number of displays fitted.

In 1983 the ICID designed and produced a microcomputer of 8 bits based on the microprocessor INTEL 8080, which supports the CP/M 80 with all its software.

Now production of a microcomputer of 16 bits has been started, from a design based on the microprocessor INTEL 8086, which is IBM-compatible, so being able to use all the software yet developed.

The conception of microcomputers has made it imperative to work base towards exploitation and assimilation of the available software, as well as towards completition according to our needs.

Systems have been worked out for combining the microcomputers

with our minicomputers, ultimate outcome of which should be a network system.

Today, the tendency of our efforts are directed towards distributing our processors. Thus the hardware is put as close as possible to the source of data generation.

1.2 Characteristics of the applications.

We now give the reader a view of the main applications made by the users of minicomputers so that he will be thoroughly informed on the present status of computation in Cuba /PEDRO82/.

In the industrial field:

- Planning and control of sugar production, with its various indicators.
- Control of the outlining of production and financial plans in sugar factories.
- Control of accounting, statistics and labor force in factories.
- Operative control of the economic-technical plan in enterprises and factories.
- Calculation and issuance of power consumer receipts.
- Invoicing of sugar cane produced by farmers.

In the field of agriculture:

- Calculation and evaluation of soils and fertilizers.
- Programming of cane cutting.
- Programming of sugar transport routing.
- Production estimates.

- Business applications in farming enterprises.

In the field of transportation:

- Invoicing of maritimes services.
- Control of loading and unloading of vessels.
- Control of goods in harbors.
- Operative control of railroad traffic.
- Business applications tasks in economic units.

In the field of commerce:

- Control of the plan of sales of commodities and repair parts.
- Processing of surveys related to the population.
- Market analysis and demand forecasting.
- Business application tasks in economic units.

In the field of health:

- Hospital disbursements.
- Biological control of drugs.
- Selection of optimal receiver in kidney transplantations.
- Census of hospital beds.
- Statistic control of hospitals.

In the field of education:

- Programming of activities.
- Statistic control of students.
- Control and planning of teaching materials.

In the field of sports:

- Calculation and control of training in various sports.

- Analysis of tactics and techniques in sport activities.
- Recording and control of performance of athletes.
- Business application of sport units.

From the analysis of the main applications the following conclusions may be drawn:

- There is a high rate of application redundancy.
- Most applications are those known as data processing and economic calculations.
- Applications show low efficiency of utilization of all resources furnished by the basic software.
- Integrated program packages which would increase the degree of automation of application are not yet developed.

The fundamental reasons for these conclusions are:

- Lack of maturity in know-how of applications of the computers.
- Low professional level of applications programmers.
- Low reliability of the hardware.

From the above conclusions it must be pointed out that although a number of institutions have made progress in the field of applications and the performance of their specialists, they are rather exceptions in the national picture.

With the recent introduction and production of 8 and 16 bit microcomputers, the situation has changed. First, computation techniques have been spread throughout the country, and now exist in places where minicomputers had never existed. Besides, the

software available for these equipments has imposed an evolution in application conceptions. In the field of management data processing work has commenced on Data Bases Management Systems (DBMS) implemented for microcomputers, which, even though not offering all the possibilities afforded by the theory of Data Bases, represent a fair approximation and their application in our country indicates a qualitatively higher development. This has obliged us to analyse the possibility of implementing a DBMS on our minicomputer CID 300/10 with characteristics similar to those of the microcomputers, thus enabling our users to raise the quality of applications.

With the implementation of a DBMS for CID 300/10, a higher step is reached in the logical line of dialectic development in computation, which in turn allows higher development of users of our computers and their applications with enhanced utilization of available resources.

2. Selecting the DBMS.

An early stage in the implementation of our DBMS for the cuban minicomputer CID 300/10 was the assessment of the various DBMS available in order to determine if it was necessary to design a new DBMS and learn the general characteristics of present systems in order to incorporate them to our system. From this study one important conclusion was denoted: a wide range of Relational DBMS exists in the market, each one with a different man-machine interface; because of this it didn't think to make a new design of one different DBMS with the consequence that our users will not suffer for the problems of incompatible systems.

It was very clear that the new DBMS will be a complement to the software devised for CID 300/10 and it will have to fulfill the following purposes:

- easy man-machine interface.
- efficient management of data structures according to the hardware's memory restrictions.
- to ensure work with the rest of the software, and specially with Operating System for Commercial Applications GES 300 /FONFR03/.

At the time this investigation was done there were only two DBMS for microcomputer available in Cuba: dBASE II and SENSIBLE SOLUTION. For this reason, it will be shown the general evaluation of these systems. It is proposed a new methodology for

general evaluation which is divided into: feature analysis, human factor aspects, performance analysis (benchmarking) and an original phase not included in any paper consulted: an evaluation of the characteristics of the implementation of both systems. There are many papers that make evaluation of different DBMS /BARLE81/, /BITT083/, /BOAR84/, /BOGDA83/, /KEENA81/, /TEMPL/, /BOND84/, etc. but only from the user point of view. That is the reason because any author show the evaluation between different features of implementation of the DBMS.

In the benchmarking section it is compared the following: dBASE II, SENSIBLE SOLUTION, dBASE-300 and COBOL CID 300/10, in order to observe the performance between the DBMS available with our DBMS and the previous software: COBOL compiler /FONFRO2/.

Also it is included, in interest of the readers, the result of the same benchmarks for dBASE III PLUS running on IBM PC and dBASE II running on 32 bit microcomputer under UNIX.

The second section of this chapter shows the conclusions about the general evaluation and justify the DBMS selected.

2.1. Behavior of the Systems.

The rising popularity of database systems for the data management has resulted in an increasing number of new systems entering the marketplace. Database systems have been implemented on many different computer architectures: mainframes, minicomputers,

microcomputers and as standalone database machines. The selection of a database system among these varied alternatives requires a structured, comprehensive evaluation approach.

A complete evaluation methodology for database systems must integrate a feature analysis phase, human factors aspects /CNORT83/, and a performance analysis phase. Also, in our case, it must have in mind that it needs to select a DBMS not only for the user, because it could be implemented in our minicomputer with hard requirements of configuration. Therefore, it complements the methodology with a phase about the characteristics of implementation of each system.

The figure 2.1 shows a summary of our methodology.

2.1.1. Feature analysis phase.

The range of features and capabilities that a database system may support is very large. A feature analysis performs two functions; it first serves as a winnowing process to eliminate those systems that are completely unsuitable for answering the needs of a particular application and second, it provides a ranking of the surviving candidate systems.

Feature analysis has a number of significant advantages over other methods of system evaluation.

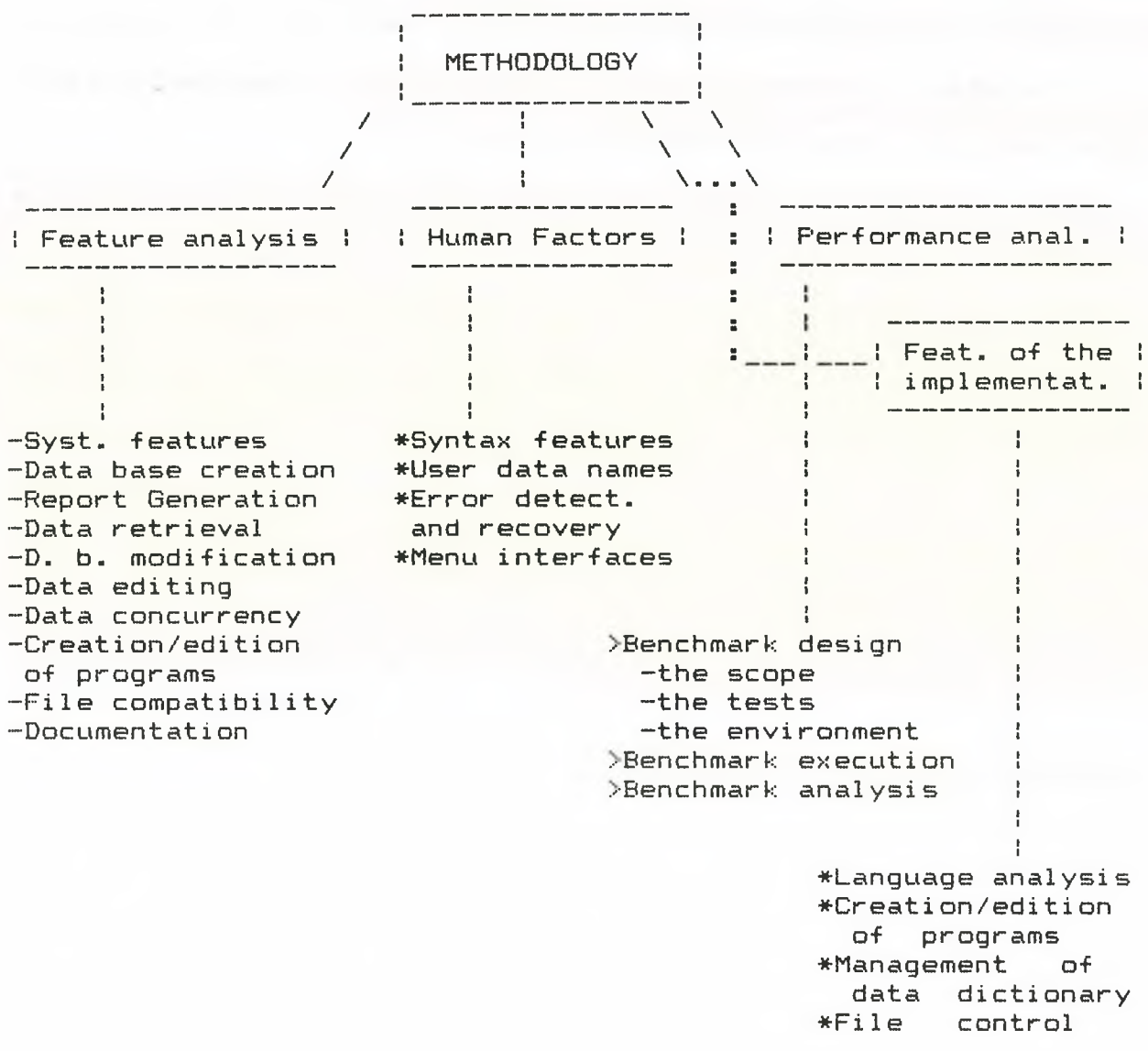


Figure 2.1. A summary of our methodology.

i) Feature analysis provides a structured first cut. The final result of a feature analysis should be a small number of candidate systems. Performance analysis, which is much more costly, can then be performed with only this small number of systems.

ii)There are qualitative aspects of a database system that cannot be quantified in terms of system performance; for example: vendor support, documentation quality, security, user friendliness, etc. Since benchmark analysis cannot directly test the performance of these features, feature analysis remains the best method for their analysis.

iii) Little or no system costs are involved in performing a feature analysis because a database implementation is not required.

In spite of these advantages features analysis should not be used in isolation to evaluate and select database systems. There are several reasons for this.

i) The feature importance coefficients and the system support ratings are given values by a knowledgeable design expert. However, no two experts may come up with same values given the same application environment, because the feature analysis is a subjective exercise.

ii) Feature analysis is a paper exercise that cannot truly evaluate how a system will perform in an organization's

application environment.

The objective of this evaluation is not to choose a system for an application, but to take a DBMS as a pattern to be implemented in our minicomputer. Because of this, several features of the systems do not have great importance; for example, the OS on which the system executes, arithmetic precision, error recovery, etc.; these are implementation characteristics that can be adapted to our necessities. These characteristics are mentioned in this evaluation and its study is not too deep.

SENSIBLE SOLUTION and dBASE II are relational DBMS. Their capabilities exceed those of comparatively simple one-file data managers like pfs-File, and they provide the means to handle complex, multiple-file applications. Separate files may be set up for costumers, vendors, inventory, accounts receivable, accounts payable, and other activities of a typical business-control system, with information intermixed among them.

A "relation" is used to bridge the gap between where information is stored and where it is needed. A common field is used as a unique identifier to provide the necessary bridge.

While a relational feature is essential for complex database, it is only one of a variety of capabilities that a good DBMS should have.

The following are the points included in the feature analysis for

each system.

1) System features.

2) Data base creation.

3) Report generation.

4) Data retrieval.

5) Data base modification.

6) Data editing.

7) Data concurrency

8) Creation and edition of programs.

9) File compatibility.

10) Documentation.

dBASE II.

This system was developed by Ashton-Tate and it is designed for 8-bit microcomputers and it is used at present in 16-bit microcomputers /DBASE83/.

dBASE II is a system guided by a built-in programming language with a very large set of special commands, operators and functions. It uses a very flexible syntax command which increases the easy of use.

dBASE II uses a data dictionary for every file or relation. It does not make difference between relations of different data bases.

To create a new file, enter the command CREATE, the name of your new file, and the characteristics of each field in the file. Fields may be text string, numeric or logical. Only 32 fields per relation may be defined. After the last field is defined, you may inmediately begin entering data. Password protection is not available. Data-entry masks and similar features are possible, but they must be programmed by the user into a dBASE II command file.

dBASE II provide a simple command to index a file (INDEX).

The system has an special command (REPORT) to generate reports from the file in use. REPORT is easy to use but it has limited capabilities. It has only one level of subtotals and existing reports cannot be modified. Complex dBASE reports must be written with the built-in programming language.

dBASE has good tools for data retrieval. In order to know the status of a data base it is only necessary to use the LIST command. Besides, this command allows to know the structure or dictionary of a data base. Also, it is possible to use commands and operators to retrieve the information, in a selective way, from a part of a data base or from the whole data base. These features can be used in interactive form or imbedded in a program. Besides, it has means to retrieve data from several relations, although it is limited by two relations opened simultaneously. If it is used more than two relations simultaneously the work

becomes slower, and more difficult.

To modificate a data base, which is necessary in any application, there are supplied several means which allow, in an easy way, to asimilate all modifications introducing changes in a data base.

To data edition (edition and modification fields of a file) there are supplied powerful tools. To editing data interactively it is available a very good command (BROWSE) that allows full screen viewing and edition of a file. Also, EDIT and CHANGE commands allow the user to selectively change the contents of the data fields in a data base. The REPLACE command is used to replace the contents of specified data fields of the file in ad hoc form or by program. The index file in use is automatically updated when records are modified.

dBASE II has not a general dictionary. It has a data dictionary per file, therefore, it is not possible to make a good data concurrency control.

The creation and modification of programs is done by a command that allows minor full-screen editing of command files.

The compatibility of files between different systems is achieved by the COPY command, which allows to transfer files between different systems, in partial or complete form.

The documentation of the dBASE II is good, clear and easy to use. It contains a User Manual, a Reference Manual and a Tutorial. It has several examples that show the main use of the system.

The Reference Manual is necessary to be completed with a deeper explanation about some commands and error recovery.

SENSIBLE SOLUTION.

This system was developed by O'Hanlon Computer Systems Inc., and was designed for 8-bit microcomputers /SENSEB/.

It is system driven by menu. The system contains a general menu that provides to the user the possibility to choose the desired task: execute a program, create/update the dictionary, create a screen, edit a source program, initialize a data file, compile a program, re-index a data file, change the structure of a data file, etc.

This DBMS uses a general data dictionary distributed into two files: one for the data files which conform the base and the other one for the data contained in every field of every file.

The data fields may be: alphanumeric type, numeric type, date type, overlay type (two fields can be defined as one), and record number type (contains the record number for each data record saved). Simple data-entry masks are possible to be defined. More complex masks must be programmed.

SENSIBLE is a system which compiles and executes separately (not an interpreter) and because of this every program must be created (only by the editing option of the program and not by any other text editor) compiled (by the source program compilation option),

and executed (by the program execution option).

Any change in the data dictionary implies to compile again all programs which use those changed files, and because of this the debbuging of complex systems is too slow. The programming language has powerful instructions but does not supply the flexibility and generality of dBASE II. Besides, the programming is made based on GO TO and GO SUB instructions, because of that it is not a structured language, and also it is a tedious language and not very clear.

The definition of indexed files is made during the dictionary creation

SENSIBLE has two possibilities for the report generation: one for simple and quick reports through the INQUIRE option and another one through programs using instructions oriented to the report handling. The second one is not easy for complex reports. Both possibilities need to use programs.

The data retrieval, for example, to know the status of a file is simple through the option INQUIRE, but not as simple as in dBASE II.

For selective retrieval there are supplied language instructions. For data retrieval from more than one relation there are supplied powerful searching instructions in files with related fields and it is possible to have up to 10 files simultaneously opened.

One of the options supplied in the main menu is to modificate a

This is the feature analysis phase. In figure 2.2 and 2.3 are tabulated the characteristics evaluated.

System Specifications	dBASE II	SENSIBLE
Records per data base file	65535	16,777,216
Characters per records	1000	not limited
Fields per record	32	1000
Characters per field	254	255
Largest number	1.8 * 10(63)	1 * 10(12)
Smallest number	1 * 10(-63)	1 * 10(-11)
Numeric accuracy	10 digits	5 digits
Index key length	100 charac.	72 charac.
Opened data base files	2	10
Type of index organization	B+tree	B+tree
Menu driven	no	yes
Built-in programming language	yes	no

Figure 2.2

Features evaluated	dBASE II	SENSIBLE
Simple data base creation	Very good	Very good
Complex data base creation	Poor	Poor
Data entry	Good	Good
Simple report creation	Very good	Good
Complex report creation	Poor	Good
Ad hoc queries	Good	Poor
File layout modification	Good	Poor
Data concurrency control	Not applicable	Good
Creation/edition of programs	Good	Very good
File compatibility	Good	Poor
Documentation	Good	Poor

Figure 2.3

2.1.2 Human factors aspects.

Relational technology was provided to a new class of users through simplified terminology and a relational algebraic command language. These new users knew their application areas well, but their main tasks were nonprogramming tasks. To the correct evaluation of a DBMS it is necessary to make a heavy analysis of the human factors or psychologic aspects in order to accept a system. A few authors include these factors in the general

evaluation of the systems.

The following phase highlights the human factors aspects, the benefits, and the limitations of each system evaluated.

i) Syntax features of the commands.

In this point it is evaluated the syntax features of the names of the commands: friendly language, relation between the name of the command and the corresponding data base operation, and the full command names and keywords without abbreviations.

The use of a language close to the natural (English) is very important to the assimilation and learning of a system. The user does not feel the difference between the way usually he thinks and the way he works with the computer. This is important to decrease the debbuging time of applications. The use of "noise" words helps to improve the readability of a command. Also, these characteristics improve the self documentation of programs.

dBASE II provides a language formed by English-like commands. This language is close to the natural language but it does not include the "noise" words. The close relation between the name of the command and the data base operation avoid to confuse the user in his application. dBASE II does not use command mnemonics but it employs a few symbolic command names that may have some confusion in the users ("?", "@", etc). In general, it has a good evaluation in this topic.

SENSIBLE has a rigid language and it is not close to the natural language. It uses abbreviations to name the commands, therefore it is not useful to self documentation and legibility of programs. The command names are in correspondence to data base operations. The evaluation is that SENSIBLE has poor conditions in this aspect.

ii) User data names.

Here, it is analyzed the possibilities that the system provides in order to express the names of the user data in legible form. Also, this aspect has influence in the keyboard errors. From some observations, users desire conciseness, but this is overshadowed by the need to express and document ideas in meaningful phrases. Users frequently try to condense abbreviations or use meaningless names such as X or ABC that make errors typing them or cannot remember the precise names that were used. A good system must allow that syntax of data names be legible.

Both systems permit that data names are formed by letters, numbers and several symbols. SENSIBLE permits up to 15 characters per data name and dBASE II permits up to 10 characters, but it is shown in a study presented in /CNORT83/ and confirmed by the author that almost all user can express its data names with 10 characters length in legible form. SENSIBLE provides in the definition of the field (dictionary) an associated field that

contents the description or reminder of what purpose your field serves. Therefore, SENSIBLE and dBASE II have similar conditions in this aspect.

iii) Error detection and recovery.

The time lost when errors are not handled properly for the user indicate the importance of good error handling. Here, it is evaluated if the systems have a good error detection, recovery, and informative messages.

Both systems have similar error detection and recovery. In the compilation phase, dBASE II provides the possibility to edit the line that contents some error, but it is not much used in the practical work because it has few possibilities to make modifications.

Both systems have some problems in the recovery error in the execution phase and fail some diagnostic error or they are very poor.

In general, it is concluded that both systems have good error detection and recovery.

iv) Menu interfaces.

In the same form that increases the interactive way of work with the computer, it increases the use of menu interfaces between the man and the computer. The systems driven by menu are very

easy to use. With the combination of menus defined with meaningful English phrases and availability of "help" messages, users have not much trouble, becoming effective users. Users do not have to learn or remember commands; they simply make choices from a menu. This is an important human aspect for the easy assimilation of a system, specially for the non-specialized user.

SENSIBLE guides all operations by menus. It provides a general menu and each option has sub-menus.

dBASE II fails in this aspect.

v) Learning.

It means how long it takes the user to learn how to work with a system. This is a very important human factor to accept a system by the user. A very efficient system but with difficulty when it shows the form of use, will be difficult to be accepted for the common user.

It is observed that SENSIBLE is not easy to learn. The author makes a monitoring work in two Institutes and he concluded that SENSIBLE is difficult to learn for the non-specialized user because the way of work is different to the common way of traditional data processing. On the contrary, dBASE II provides similar work of data processing environment.

Finally, it is concluded that in the human factors aspects dBASE II has better behavior than SENSIBLE because menu interface is the only point in which dBASE II has bad evaluation, but the easy to use language and the friendly interface of dBASE II compensate this aspect. The figure 2.4 tabulates the aspects evaluated.

Aspects	dBASE II	SENSIBLE
Syntax features	Good	Poor
User data names	Good	good
Error detection/recovery	Good	Good
Menu interfaces	not applicable	Good
Learning	Very good	Poor

Figure 2.4. Human factors aspects evaluated.

2.1.3. Performance analysis phase.

The major methods of performance evaluation are Analytic modelling, Simulation modelling, and Benchmarking.

Analytic modelling represents a system by defining equations that relate performance quantities to known system parameters. The use of these equations allows a fast and accurate means to evaluate system performance. The principal disadvantages are that the equations are inadequate to model the complete range of

functionality found in a data base system and also they fail to account for the dynamic behavior of the data base system. For these reasons analytic modelling has failed to receive wide acceptance as a tool for modelling data base systems.

Simulation is the process of developing a computer program to approximate the behavior of a system over a period of time. Simulation modelling has been applied to data base systems /HULTE77/, /NAKAM75/. The major concern with using simulation is the time and expense that are often necessary to develop a simulation model. Stochastic simulation models also produce only estimates of a model's true performance and the large volume of results returned by a simulation often creates a tendency to place more confidence in the results than may actually be warranted.

Benchmarking is used when a few data base systems are to be evaluated and compared. Benchmarking requires that the systems be implemented so that experiments can be run under similar system environments. Benchmarks are costly and time-consuming but provide the most valid performance results upon which data base systems can be evaluated. While both simulation and analytic modelling are limited in the scope of their system testing, benchmarking offers the chance to evaluate the actual data base system /GOFF73/.

The benchmark experiments publicated concentrate on the

comparison of candidate commercial systems for a particular application /GLESE81/, /ASTRA80/, /KEENA81/, /TEMPL/, etc.

While benchmarking can be a useful and important technique for data base system evaluation; designing, setting up, and running a benchmark is a difficult and time-consuming task. Benchmarking is problematic and at worst, a gross distortion of reality but it is possible to obtain good conclusions if these aspects are known and if specific features are analyzed.

In order to aid in the development and analysis of benchmarks it is essential to show the methodology used. No one methodology has provided the necessary robustness demanded from a generalized methodology. No benchmark methodology can expect to incorporate every aspect of every benchmark.

Our methodology has been divided into 3 principal parts: benchmark design, benchmark execution and benchmark analysis.

2.1.3.1. Benchmark design.

The design of a benchmark involves: a) the scope of the tests, b) the tests to be performed, and c) the environment of the data base system to be tested.

a) The scope of the tests. As it was shown above, the success of benchmarks depends on the objectives be exactly detailed. It has been proved that general benchmarks distort the results and mask the deficiencies /HOUST84/. In our case, it uses the benchmark to

complement the other phases that are included in our evaluation. Besides, the systems evaluated allow to perform several classic processes of DBMS in interactive way as: creation and modification of data bases, edition of programs and data bases, report generation, etc., which are not possible to apply to any classic benchmark test. These features are included into another phase of our general evaluation.

The points to evaluate are the following: arithmetic instructions, management of character strings and file management.

b) The tests to be performed. Owing to the objectives shown above were designed the following tests.

i) Arithmetic instructions.

The test of arithmetic instructions is done by a program called TEST1. It has a cycle of 500 additions, multiplications and divisions working with integer and floating point numbers.

ii) Processing of character strings.

The test for the processing of character strings is contained in the program TEST2. This test contains the more common processing of character strings: concatenation of fields with variable length, to move a string to the field and to convert numeric string types to alphanumeric string types.

iii) Processing of files.

This point is contained in several programs. In this form the

different transactions can be isolated and is possible to know in which points the systems have or do not have a good performance.

TEST3. Sequential creation of files. The data structure appear in the test data point.

TEST4. To index a file.

TEST5. Random access to an indexed file.

TEST6. Random access to two related files. It is simulated the following typical case: "which is the salary of Luis? " in the figure 2.5.

NAME	PROFESSION
Peter	engineer
Luis	physician
Jose	mechanic
Carlos	physician

PROFESSION	SALARY
engineer	$300.00
physician	$350.00
mechanic	$200.00

SALARY	CATHEGORY
$200.00	first
$300.00	second
$350.00	third

Figure 2.5

TEST7. Random access to three related files. It is simulated the following case: "Which is the cathegory of Luis? "

TEST	APPLICATION
1	Arithmetic
2	String processing
3	Sequential file creation
4	To index a file
5	Random access in an index file
6	Two related files
7	Three related files

Figure 2.6. Summary of the tests.

c) Environment of data base systems to be tested.

This point contains: i) System configuration and ii) test data.

i) System configuration.

The hardware and software parameters are included here. There are two basic system configuration owing to the following considerations:

1- performance analysis of dBASE II, SENSIBLE SOLUTION and dBASE III.

2- performance analysis of dBASE 300, COBOL compiler, dBASE II for 8-bit microcomputer, and dBASE II for 32-bit microcomputer.

The two system configurations corresponding with considerations expressed above are the following:

1- SYSTEM1. The parameters are the following:

*IBM PC/AT with 512 Kb RAM clocked at 8 Mhz including a 30-megabyte hard-disk and a high density 5 1/4-inch disk drive.

The operating system was MS-DOS version 3.20.

The DBMS are the following:

dBASE II version 2.4

SENSIBLE SOLUTION version 2.0A.

dBASE III PLUS version 1.1.

2- SYSTEM2. The parameters are the following:

*Cuban minicomputer CID 300/10 with 28 K word of main memory, including a 5-Mbyte Bulgarian mini-disk CM 5400.

The operating system was the FOBOS version 3.0.

The software evaluated are the following:

dBASE-300 version 1.00

COBOL-FOBOS version 2.00

*Cuban microcomputer CID 1408 with 48 Kbytes RAM, and two 8 inch disk drives.

The operating system was CP/M version 2.2.

The software evaluated was dBASE II version 2.4

* 32 bit AT&T microcomputer model 3B2/300 with 1 Mbyte RAM, 30 Mbyte hard-disk, and two WYSE displays attached to serial

ports.

The operating system was the UNIX version V.

The DBMS evaluated was dBASE II version 2.4E.

ii) Test data.

The data base used in the tests must be implemented on each of the candidate systems to be tested and after implementation must remain constant over all systems. There are basically, two methods for obtaining a test data base: using an already existing application data base or developing a synthetic data base.

The traditional method has been the use of real data from an application data base but, in our case, this method may produce unexpected problems, because the data must be formatted into the appropiate form for each system to be tested.

In our tests was used the synthetic data. Synthetic data is generated to make up a data base which easily lends itself to benchmark testing. The attributes are assigned unique values. For example, for a relation with 500 tuples the key attribute may take the values 1,2,3,...500.

The main purpose of these attributes is to provide a systematic way of modelling a wide range of selectivity factors.

The data base record size was also selected in order to show the major application data base features.

The test data base structures are the following:

- FILE 1

  Field 1: 6 positions numeric integer (key)

  Field 2: 30 characters string

  Field 3: 30 characters string

  Field 4: numeric with 3 integer and 2 decimal positions

  Field 5: 10 characters string

- FILE 2

  Field 1: 6 positions numeric integer (key)

  Field 2: 40 characters string

  Field 3: 30 characters string

  Field 4: 20 characters string

- FILE 3

  Field 1: 6 positions numeric integer (key)

  Field 2: 40 characters string

  Field 3: 30 characters string

  Field 4: 9 positions numeric integer

  Field 5: 40 characters string

The different types of data were selected in form that they were expressed in all the DBMS tested, because there are special data types used in some DBMS and in another cannot be used, for example date field.

It was made several test to obtain the data base size adecuated. It is concluded that the test time increase or decrease proportionately. The data base size of 500 records was selected

and it permitted to perform the tests with an appropriated machine time.

2.1.3.2. Benchmarks execution.

When the experiment has been formally defined, the next step is to implement the design for each of the candidate systems.

The figures 2.7, 2.8, show the results of the tests by system configurations.

TEST	dBASE II	SENSIBLE	dBASE III PLUS
1	00:13.00	00:14.50	00:15.00
2	00:11.00	00:06.50	00:18.00
3	00:24.00	00:30.30	01:03.50
4	00:17.50	00:55.50 *	00:04.50
5	00:29.00	00:26.50	00:34.00
6	00:56.50	01:08.00	01:19.50
7	05:35.00	01:39.00	04:57.00 ** 01:34.00***

* Indexing file is included in the creation file operation.

** Without apply the option that relate two files (SET RELATION)

*** Using SET RELATION.

Figure 2.7. A summary of test results in SYSTEM1 (IBM PC/AT).

The times are expresed in minutes:seconds.cent.of second.

TEST	dBASE II 8 bit	dBASE 300	COBOL	dBASE II 32 bit
1	01:35.00	00:50.00	00:00:25	01:08.00
2	01:18.00	01:10.00	01:39.57	01:36.00
3	02:55.00	01:50.00	00:22.00	02:40.00
4	03:45.00	01:00.00	00:20.00	02:29.00
5	03:45.00	02:40.00	01:52:00	02:38.00
6	07:45.00	05:35.00	03:40.00	05:00.00
7	44:10.00	08:20.00	04:30.00	07:40.00

Figure 2.8. A summary of test results in 8-bit microcomputer, cuban minicomputer, and 32-bit microcomputer.

2.1.3.3. Benchmark analysis.

The final phase of benchmarking is the analysis of results. Evaluation of the data generated during benchmarking must begin before the tests have been completed. It provides feedback during the testing by suggesting which types of experiments need to be repeated in more detail, or should be extended in some way. Summarizing the meaningful information from these results and discussing them in a report form is a key step in the benchmark testing.

The figure 2.7 shows a summary of results executed in SYSTEM1

(IBM PC/AT). It may observe that in arithmetic, character string and file creation both systems have similar behavior. In the index file operation, dBASE II is faster than SENSIBLE but the last one has the create operation included.

The file management has few differences in time execution, except in test with 3 files involved because dBASE II can not operates with 3 files simultaneously. Therefore, it concluded that in this phase dBASE II and SENSIBLE have similar behavior except in case of more than 2 files opened.

Besides, the reader may observe, although dBASE III PLUS provides more confortable and complex features its behavior is worse compared with dBASE II and SENSIBLE.

The figure 2.8 shows a summary of results executed in SYSTEM2: 8-bit microcomputer, cuban minicomputer, and 32-bit microcomputer. The goal of this test is to show the behavior of dBASE-300. It may observed that our implementation has a intermediate behavior between 8-bit microcomputer dBASE II and COBOL compiler. Also, our implementation is not very slow compared with dBASE II for 32-bit microcomputer under UNIX.

COBOL compiler is faster than dBASE-300 because it has more memory available in the execution phase, and it does not have to compile during execution time. This confirm our premise of design that contemplate a compiler for dBASE-300 which it is a recommendation of our paper. This is explained in the next

chapter.

Another interesting result is the comparison between dBASE II in IBM PC/AT and dBASE II in AT&T 32-bits microcomputer.

2.1.4 Characteristic of the implementations.

As it has shown above, the objective of our DBMS evaluation is to select a system to implement it in our cuban minicomputer.

Therefore, it is necessary that our methodology contains a phase about the difficulty to implement one or another DBMS.

In this phase the following modules will be analyzed: language, creation and edition of programs, management of data dictionary, and file control system.

i) Language.

The characteristics of implementation of both languages are explained below.

dBASE II. This DBMS provides a heavy built-in language. It has a structure formed by the following elements: commands (with their options), expressions (functions), and operands (file names, fields and memory variables). These elements are possible to find mixed between them. For each element it is necessary to make a different syntax and semantic analysis. For example, the syntax

structure of a function does not have definition and it is different from command structure. The name of a function can be a letter string followed by the parameters enclosed in parenthesis (INT(x)). Also, a function can be a single character ("*" deleted record function).

In the same form, the command structure is not simple because the name of a command may be a letter string (DISPLAY) or a single character (@).

These features complicate the parser processing. It is necessary to build independent syntax processors for each element.

A group of simple items and operators is an expression that can be evaluated to form a new simple value. The analysis of an expression is complex because it is formed from the following components: data base field variables, memory variables, constants (literals), functions and operators.

Another difficulty for the grammatical processing of language is that several commands have some options that specify the scope of a command: "scope" means how much does the command cover.

Besides, there are some ambiguity of language that make very difficult the implementation. For example: the symbol "#" can be the record number function or the relation operator of "not equal to", therefore the expression x # y can mean " x not equal to y " or " x multiply by the record number and multiply by y ".

There is another problem from the implementation point of view:

dBASE II do not have compiler separated, it works with the compilation and execution phase at the same time. This feature obligue to conform one module for compiler and executer.
The language has more than 60 commands each one with options.
All these above show that dBASE II has a very difficult language from point of view of implementation.

SENSIBLE. It provides a low level language. The basic elements of the language are: labels, commands and operands. These elements have to appear in a rigid form in the program.
The expressions are simple. It is formed by variables, literals and operands. The four operators are +, -, *, and /.
The cycles are based in "IF-GOTO" commands type. There are other commands that perform jumps when some condition is setted.
In SENSIBLE the auxiliary variables are storaged in a common file and its have the same structure of a real file. Therefore, the processing is similar for auxiliary variables and field files.
The name of fields, of auxiliary variables and labels have the same structure; they are formed by up to 15 characters and may contain almost any character except for ",", "(", ")", "<", and ">".
SENSIBLE has separated the compiler and the run time system. The user has to compile the program before the execution. This approach is more easy to implement because each phase manipulates

all the available memory.

SENSIBLE has no more than 20 different commands.

This system included a separated module for the syntax edition of programs. This module performs the 90% of syntax checks.

For all the aspects shown above it is concluded that implementation of SENSIBLE compiler is not complex.

Finally, it has shown that implementation of dBASE II compiler is more complex and requires more main memory than SENSIBLE compiler. In figure 2.9 it shows a summary of the features analyzed.

Language features implementation	dBASE II	SENSIBLE
Language	evolved	primitive
Type	interpreter	compiler
Structure	complex	simple
Syntax structure	no uniform	uniform
Expression	complex	simple
Ambiguous	yes	not
Branch	no simple (nested)	simple (no nested)

Figure 2.9.

ii) Creation and edition of programs.

The creation and edition of programs are the means included in the DBMS to develop programs.

Both systems evaluated provide options to fulfill these functions. From implementation point of view both editors have similar characteristics: minor full-screen editing of command files. So far, the editors implementation have similar difficulties, but there is some special characteristic for SENSIBLE editor: it has double function. It has the edition normal function and also has a parser function. This is an essential and important difference between SENSIBLE and dBASE II. Because the editor promts you for the command and all necessary parameters, syntax errors can never happen. The editor will reject a reference to a field that has not been defined. Therefore the SENSIBLE editor is more complex to implement than dBASE II editor because SENSIBLE has to contain procedures for the syntax analysis and has included the management of symbol table.

This difficulty is compensated by the way that editor works. The edition is guided by menus and the syntax analysis is very well delimited in each step.

Because of these factors it is concluded that SENSIBLE editor is more complex to implement than dBASE II editor.

iii) Management of data dictionary.

This module includes all the software means necessary to control the operations with data dictionary.

The systems evaluated have differences in data dictionary structure. dBASE II provides a data dictionary for each relation. It does not have general data dictionary and the auxiliary variables are not included in the data dictionary. This characteristic makes the implementation difficult because it is necessary to use two different process to manipulate fields and auxiliary variables.

To build a data dictionary for each file makes more easy the implementation because it is not necessary to control the other files. Besides, the data structure in dBASE II is more simple than SENSIBLE and has not included masking for data.

dBASE II provides means for the manipulation of the data dictionary with minor full-screen facilities. Owing to simple structure of data dictionary the software associated to these screens is not very complex.

SENSIBLE provides a general data dictionary. It is formed by two data files: data file master and data field master. The auxiliary data are included also in one fixed file named MEMORY which contains only one record with the fields used by variables. The software to handle the dictionary is more complex than dBASE II because it has to manipulate both files: data file master and

data field master. Besides, its screen is more complex. Then, it is concluded that implementation of management of data dictionary in SENSIBLE is more complex and uses more memory than dBASE II.

iv) File control system.

The difficulty of programming the file organization and its commands are shown in the following paragraphs.

The type of index organization is the same: B+tree /BOND84/, therefore, the complication of implementation are similar. Also, both systems have structured in similar form: the data and the index file. But, dBASE II has a difference. In dBASE II files may be organized in sequential and random access; it depends if the file is indexed or not. In case of the file is not indexed, the file is accessed in the same order that was created, but in this case is possible to make function of direct access; for example GOTO. Therefore, the organization has to contain two different treatments: indexed and not indexed.

Although in SENSIBLE it is not used the index, the organization creates an implicit index (record number) and makes only one treatement. This approach simplifies the implementation.

The commands that make access to the data files have the following features:

a) Both systems have a common set of commands to manage files: open-close file, read-write-delete record, goto first-

last-next record, etc.

b) SENSIBLE provides commands that make linked access file on the basis of a common field in automatic form. dBASE II does not have these possibilities, then implementation of dBASE II is more easy than SENSIBLE because the user has to make the relation between files by programming, and in any time there are only two files opened in memory. On the contrary, SENSIBLE manipulates all the used files and its relations simultaneously.

c) SENSIBLE provides commands for data protection by programming. It is possible to lock records, files or all the used files. dBASE II does not have these features. The programming of these commands requires more software and main memory.

d) dBASE II includes the selective retrieve of data (scope), then, it needs more software processing to each command.

e) dBASE II provides a command set not included in SENSIBLE like: count the number of records (COUNT), to join two data base together to form a third data base whenever some criterion is met (JOIN), to rename data base (RENAME), to sum fields of a file (SUM), etc.

About the organization of file control system and its commands, it is concluded that both systems have similar requirements of software.

The figure 2.10 shows a summary of points evaluated.

Modules evaluated	dBASE II	SENSIBLE
Language	complex. The compiler is complex and needs many main memory.	simple
Creation/edition prog.	not complex	complex. It has included a syntax analyzer.
Management data dict.	simple	complex
File control system	complex	complex

Figure 2.10.

Then, some partial conclusions are the following:

- dBASE II provides a more complex language than SENSIBLE. SENSIBLE splits the syntax analysis into compiler and edition process.
- The other modules have similar requirements (more or less).
- Owing to the way of work, in SENSIBLE all the modules are independent processes, therefore it has all available resources to make its jobs. On the contrary, dBASE II has all modules in built-in form.

Therefore, although SENSIBLE in several modules requires more software than dBASE II, it has more memory available, then it is concluded that dBASE II implementation is more complex than SENSIBLE implementation.

2.2 Conclusions about general evaluation.

It is presented a summary of results shown in the above phases in order to remark this parameters.

Phase	Results
Feature analysis	dBASE II has better results
Human factors	dBASE II has better results
Performance	Both systems have similar behavior, except in case of more than 2 files opened
Characteristic of implementation	dBASE II is more complex to implement

Fig. 2.11. Final results of evaluation.

Before to show the conclution it will explained some important factors:

1) An important aspect was that in our country dBASE II looks like a standard system and is well-known, which avoids difficulties in the acceptance of our system. The fact to implement a DBMS compatible with dBASE II offers several advantages:

-The user has not to learn a different interface.

-It is possible to transfer to the minicomputer the applications developed in microcomputers.

-The user may design the application independently on which computer it will be executed.

-Connecting the microcomputer with the minicomputer, it is possible to design applications allowing distributed functions. For example, to transfer a data base to the minicomputer and to execute a REPORT in order to use the minicomputer line-printer.

-In an environment with mini and microcomputer, the user always has a backup system in case of hardware fail.

2) An aspect that has great importance in the conclusion was that it was necessary to implement the system with the possibility to link it with the rest of software developed up to date. The means selected were data files. According with this, the structure and organisation of file control of SENSIBLE is very different to the COBOL file control, because SENSIBLE has a general dictionary and splits the data file into two files. On the contrary, dBASE II file organisation is very close to COBOL file organisation because the data dictionary of each dBASE II file is possible to include in COBOL file label.

Finally, it is decided to take dBASE II as a pattern DBMS because it has very good features and our implementation can improve the performance allowing more than 2 files opened.

3. Characteristics of implementation.

In this chapter, it is described the characteristics of the major parts of the DBMS called dBASE-300 and for each of them the consulted methods and theories will be discussed, as well as justification given for the one selected.

At this point the premises for the development of our work should be recalled, which evidently influence the characteristics of the implemented DBMS. These are the following:

1) Characteristics of the hardware: a very scanty memory /CID300/, low memory capacity in magnetic disks and single user configuration (no terminals, only the main console) /FOBOS/.
2) Characteristics of the basic software used.
3) Professional experience of the designer and the executers.
4) The short time assigned for the completion of this project.
5) Compatibility with previously developed software.

During the development of the compiler the fact was bore in mind that the selected system should be highly interactive and therefore processing of its language had to be processed in interpreter fashion, that is, the language processor is made up by an object code generator (compiler) and an object code interpreter (run-time system) /GRIES71/.

A monitor was designed in order to construct a command line, depending on whether input is given through the console or it is

a command file, leaving the corresponding characters in a buffer (command buffer). This monitor transfers control to the corresponding phase, depending on whether the process is compiling or execution.

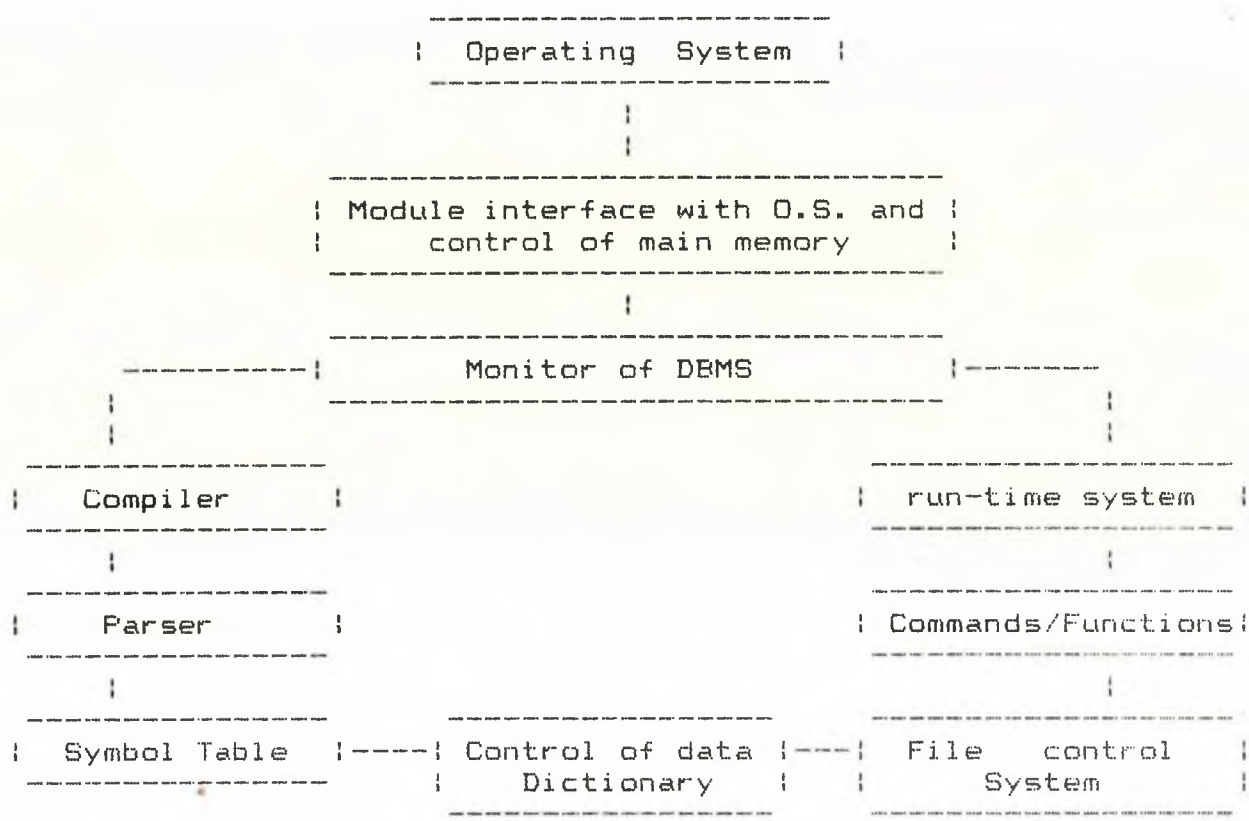
Also, this monitor takes care of errors in the compiling phase and of their recovery.

The result of the compiling phase is, if the line is syntactically correct, an object code corresponding to the transaction. This code is then transferred to the executer module, which interprets and carries out the transaction according to the object code found.

Even though it has, in general, been implemented so that the command lines have to be processed one after another, in the compiling phase and then passed on to the execution phase, the design contemplates the option that complete compilation of a source program may take place so that an object program is obtained and it may be executed some other time. However, this translation might be incomplete, since there are options in the commands such as macrodefinition and command TEST which require compilation at time of execution /DBASE83/; but, taking into, account the frequency in the use of these commands in a program, the use of a mixed code (part object code and part source code), the system's efficiency would be increased by 90% of a program's transaction since in most cases the translation would produce

object code only.

The basic structure of our DBMS is the following:



All these modules were designed according to our hardware characteristics. The following sections are in correspondence with the modules shown above.

In the first section it is shown the characteristics of the current Operating System and its possible adaptation to our conditions and a study to determine whether it provides the

necessary requirements to develop a DBMS, and it is also shown an original design for small configurations of a multitask monitor with time-sharing characteristics.

The second section shows the different variants to make the distribution of the system in overlay regions based in an study about the use frequency of the commands and an algorithm created to make the distribution. Also it is explained a suggestion in order to increase the available main memory.

In another part, it is explained the current technics about dynamic storage allocation and it is shown the adaptation of the selected method in our environment and its connection with the other modules of the system.

The next section explains the main interesting parts of our compiler. An automata Generator System was used in the syntactic analysis. Because of low memory capacity and the complexity of the grammar it was necessary to accomplish several variants which are detailed. It is shown an original method of automata implementation of complex grammars on configurations of low memory capacity.

The actual methods of symbol table handling are exposed and the selected methods are theoretically and practically discussed.

The last section shows the main characteristics of the generated Object Code and the File Control System. It is discussed how was resolved the selective retrieval of the information present in

those commands which manipulate data base.

In the topic of File Control System it is explained the file management, basic function and the indexing treatment based in the B+ tree technique.

3.1 Operating System.

In this section it is maked an analysis about the main elements that an Operating System (OS) must include in order to satisfy the implementation requirements of a DBMS. This analysis is interesting because the implementation of a DBMS for small configurations has not been studied in detail in any consulted papers. It is shown the characteristics of the current OS and it possible adaptation in our conditions. Moreover, it is shown an original design (for small configurations) of a multitask monitor with time-sharing characteristics.

The base OS used is the FOBOS which is compatible with System RT-11 of DEC. This is a very simple system, oriented towards real-time applications /FOBOS/. It has two monitors: one of a single user and another of foreground/background. The single user monitor occupies 4k bytes while the foreground/background one takes 9k.

From an evaluation of both monitors it was concluded that although coupling the system in a foreground/background regime looked attractive as our DBMS is highly interactive (which allows

time enough for another task execution) the memory requirements precluded this option. It was thus decided to work with the single user monitor.

The OS was submitted to a thorough study to determine whether it provided the necessary requirements to develop DBMS. The characteristics suggested by Stonebraker /STONE81/ were analyzed and those which, according to our requirements were to be implemented, were selected. These characteristics are:

3.1.1 Management of buffers.

This topic deals generally with methods of managing buffers for systems with memory cache /MATS070/. The CID 300 does not provide this service. Access to files is direct and very simple and so the system's overhead is minimal. Also, free memory space is small and therefore it cannot be used the solution of including a buffer pool in user area, as is done, for instance, by INGRES /STONE80/ and System R /BLASG79/. If improvements mentioned in this chapter should be carried out for reducing the amount of memory used by the DBMS, a buffer pool could be implemented in which a "toss immediately" strategy of replacement would have to be used since in dBASE-300 transactions are made on a file at random without again being referenced in the near future (generally) or in sequential access to blocks that shall not be referenced again.

Regarding a file's block prefetch, the FOBOS does not contemplate any means for establishing prefetch strategy in automatic form. In dBASE-300, this strategy may be implemented in case of sequential treatment if a buffer poll, as referred to above, is available.

3.1.2 Crash recovery.

A very important service, not included in the selected DBMS is the crash recovery. Absence of this possibility is partially accounted for by the simplicity and easy use of the system, and also by the scanty possibilities of the hardware.

We will now examine the recovery techniques proposed by Verhofstad /VERHO78/ and their possible implementation in our system.

i) Salvation programs: A salvation program in a database system is used after a crash to restore the database to some consistent state. The salvation program tries to restore the state of the database as it was before or at the time of the failure. A salvation program scans through the data structures and tries to reconstruct the database or restore consistency, possibly at the cost of deleting some files or data.

In our DBMS, this might be a utility program that reviews the data files, verifying and if necessary restablishing the lost

links in a base's structure and its implementation can be performed without great difficulties and without introducing changes into the system.

ii) Incremental dumping.

Under this technique, updated files are copied in an auxiliary storage (generally tapes). This is normally done when a job has been completed but may be performed at regular time intervals, as well, thus establishing checkpoints. After a crash has occurred the incremental dump tapes can be used to bring all the files to their previous consistent state, so that jobs completed before the crash will not be lost.

In our system, copy is made on completion of the job and this is a necessity because disk capacity is very small. Unless the user makes an updated copy of his data he may lose them when the next task is executed.

When files are copied at time intervals the more convenient variant for implementation would be if special commands instructed making a copy and restablishing the system from the last checkpoint similarly to clause RERUN of COBOL /COBOL74/.

iii) Audit trail.

An audit trail records the sequence of actions performed on a file. The audit trail file contains information on time and date when each operation has occurred, its effects, and the name of the program that performed it. From an implementation point of

view, the File Control System can make these recordings because at time an operation is carried out, all necessary data are at hand. The shortcoming of this technique in our case is the small disk capacity --it necessitates to keep a file permanently open. Also, the overhead would be excessive.

iv) Differential files.

This scheme keeps in a main file the original one without any changes. In another file (differential file) all desired changes are stored. The two files are merged regularly, thus updating the original file. Data which are not found in the differential file are to be looked up in the main file.

From the implementation point of view, this technique can be developed provided changes are made in the design of the File Control System --for every one file, two files would have to be dealt with. The setback in this connection is an increase of memory used for each task and extra time of access to files. Therefore, this scheme is unfeasible in our system.

v) Backup and current versions.

This scheme involves having several versions of a file so that a file may be retrieved which has sustained a crash, returning it to a previous state. This technique has been established by practical experience of users of our system.

vi) Multiple copies.

In this method, an odd number of updated copies is maintained

simultaneously. All files should be identical and if an inconsistency is found it can be corrected by checking up all copies.

As in the case with the previous method (differential files) this one cannot be practically implemented for lack of central and auxiliary memory. The overhead imposed on the system is also an added burden.

vii) Careful replacement.

The purpose of this technique is to obviate updating of the data structures directly in the file. The altered parts are put down in a copy of the original. Updating is done after verifying its validity and then its copy is eliminated. That is, unlike other methods, the copy exists only at the time of updating.

This scheme can be implemented by making small changes in the design as it is necessary only to place the record and its links in an auxiliary file and when completing recording, to do so by copying from the auxiliary file. In the event of a crash during updating the file can be retrieved, updating being completed from the copy file. The main snag is that an auxiliary file must be available for each file in use, which increases the need for central memory because of additional buffer areas.

Summing up, on the several variants studied it can states that Salvation Program technique is the more feasible to implement in our system under the present conditions. If suggestions are

developed in order to increase the main memory available to the system the "Careful Replacement" scheme may be implemented without making many changes.

3.1.3 The file system.

The file system provided by FOBOS supports files with fixed length unlike, for instance, UNIX, which provides files of dynamically variable length /UNIX/. Also, a unit of treatment consists of a subset of the file's characters fixed to blocks of 512 bytes, unlike systems of the RMS-11 type /RMS-11/ which enable direct access to the record according to its keys in order to be defined, filled up, updated, and maintained on direct access storage devices.

The FOBOS provides elementary, but quite efficient primitives for file management and therefore the overhead is minimal, although this made it necessary to design a file system of its own for dBASE-300, which, considering its importance, it shall explain in detail further on.

3.1.4 Scheduling and process management.

Our design contemplated the organisation of a Multiuser Data Base System. The design includes a process scheduler or dispatcher that controls task concurrence by using the server model. As will be seen below, the server model, unlike the process-per-user

approach, is more adaptable to our conditions.

Process-per-user or shared scheduler: each concurrent database user runs in a separate process; the nucleus and scheduler are then potentially contained in the address space of all processes and execute within any process.

Server model or master scheduler: this second method centralize the scheduler and to allocate one run-time database process which acts as a server. All concurrent users send messages to this server with requests.

In the area of each task's object code a table is added which contains:

IDENTIFICATION: contains the task name and number of the associated terminal.

STATUS: Informs at all times on the task's status. There are two of them:

- ready: task in memory and ready to receive the control for its execution.
- blocked: task awaiting data input through the terminal (commands INPUT, GET, ACCEPT, etc.) or task which because of lack of memory is put in suspense and stored in magnetic disk (sleeping task).

We consider blocking of a task only by these two previous causes for the following reasons:

. generalizing the waiting for other input/output media involves an effort in programming and in memory.

. the only equipment to which this status could be considered would be the magnetic disk since the printer output will always be treated by the spooling techniques /DONOV72/ in our design and waiting time for reading or writing of a block in disk is not practical because of the overhead involved by blocking and unblocking task.

PRIORITY: indicates whether the task has priority or not. For the sake of simplification it has not contemplated several priorities, which would force us to maintain several task lists by priorities.

MEMORY: word which is pointing at the head of a linked list of data blocks associated with the task; meaning that by means of this pointer the whole memory associated to each task may be perused.

FILES: this is a word which is pointing at a table associated to each task, bound by means of a double linked list, which contains the addresses of each open file's parameters, viz: file name, equipment, number of accessed record, etc. This table is described in detail in the description of the File Management System.

QUANTUM: here is place what is classically used as the time slice in which a task is executed without suspension.

In our design this time concept /SHAW74/ was changed to amount of transactions carried out with the following advantages:

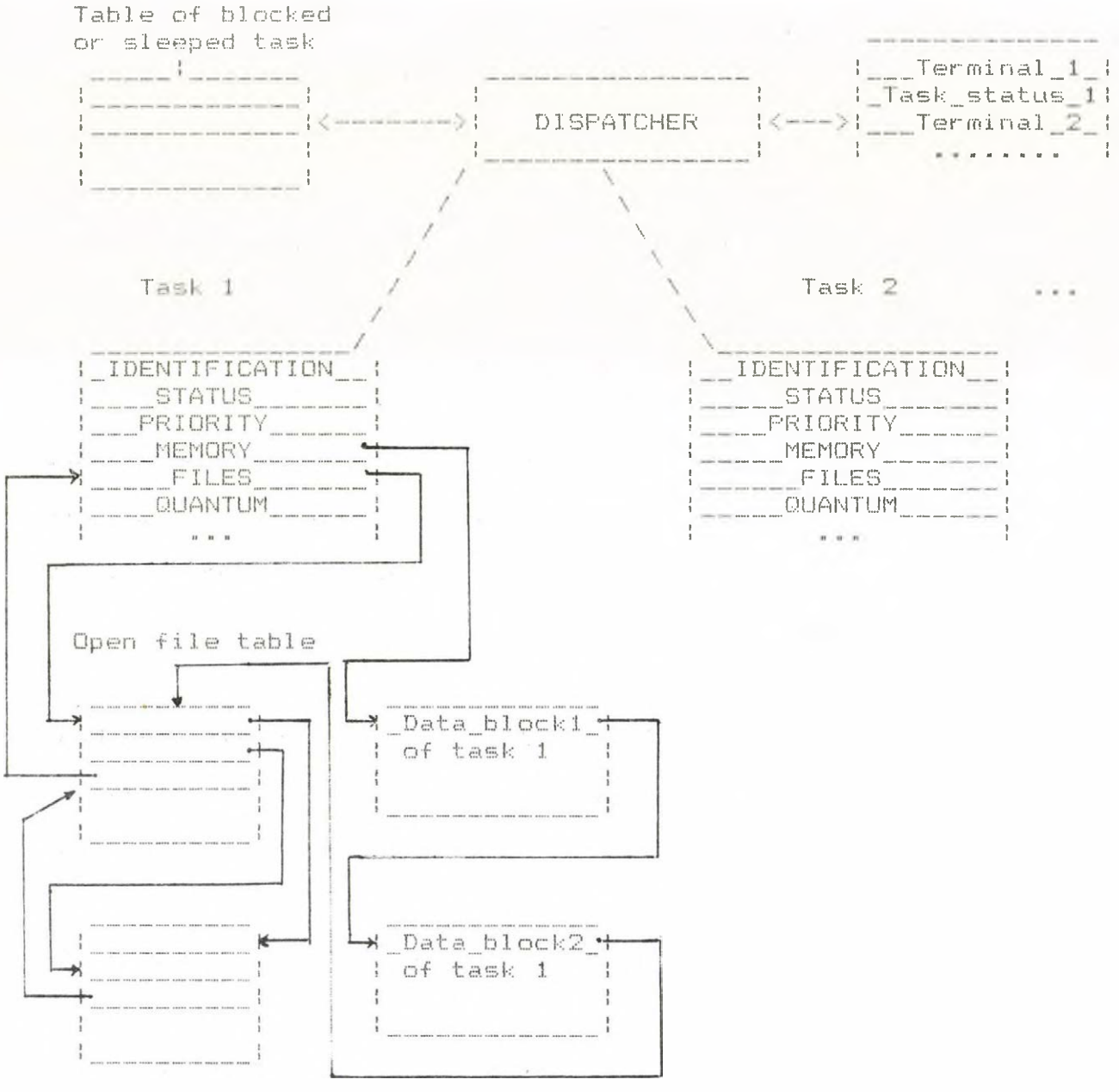
- It is very easy to know the number of transactions because whenever one is ended, the monitor takes over control and decreases the QUANTUM. If it differs from zero, execution of the task keeps going; if it is zero, control is delivered to the Dispatcher and the Dispatcher decides to which task should be given the control.
- The problems of critical section are reduced as transactions are never interrupted, therefore access to file sharing causes no deadlock problem /HANSE73/.
- Tasks are always interrupted in stable-state and this reduces overhead on prevention of system consistency problems /BERNS81/.

The dispatcher reviews the installed terminals which are awaiting commands and on finding a complete line it analyses it, and in case the command is "load task" it goes ahead with the loading, thus creating the above table.

Each process that requests memory updates the list headed by MEMORY.

The Dispatcher has a table of terminals which informs it, for

each installed terminal, whether it is awaiting commands from the Dispatcher (attached) or is being governed by the execution of a task.



The Dispatcher accepts two commands: the command "load task" and a second command that informs the user on the state of the system: tasks being executed, memory available, free terminals, etc.

Whilst tasks are in the process of being installed, tables and lists are set up which enable accomplishing a Round-Robin Scheduling with static priorities /SHAW74/.

If memory comes to an end, the task requesting memory is sleeped, its memory freed, it is written on the list of tasks sleeped because of lack of memory and is stored in magnetic disk to be continued later on. Before to decide that the task requesting memory is sleeped the following algorithm is applied in order to find the enough memory for the task:

a) If there are tasks awaiting data input from terminal, it is decided which one, or ones, release the memory request. If such task are found to exist, they are sleeped and the memory is delivered to the requesting task.

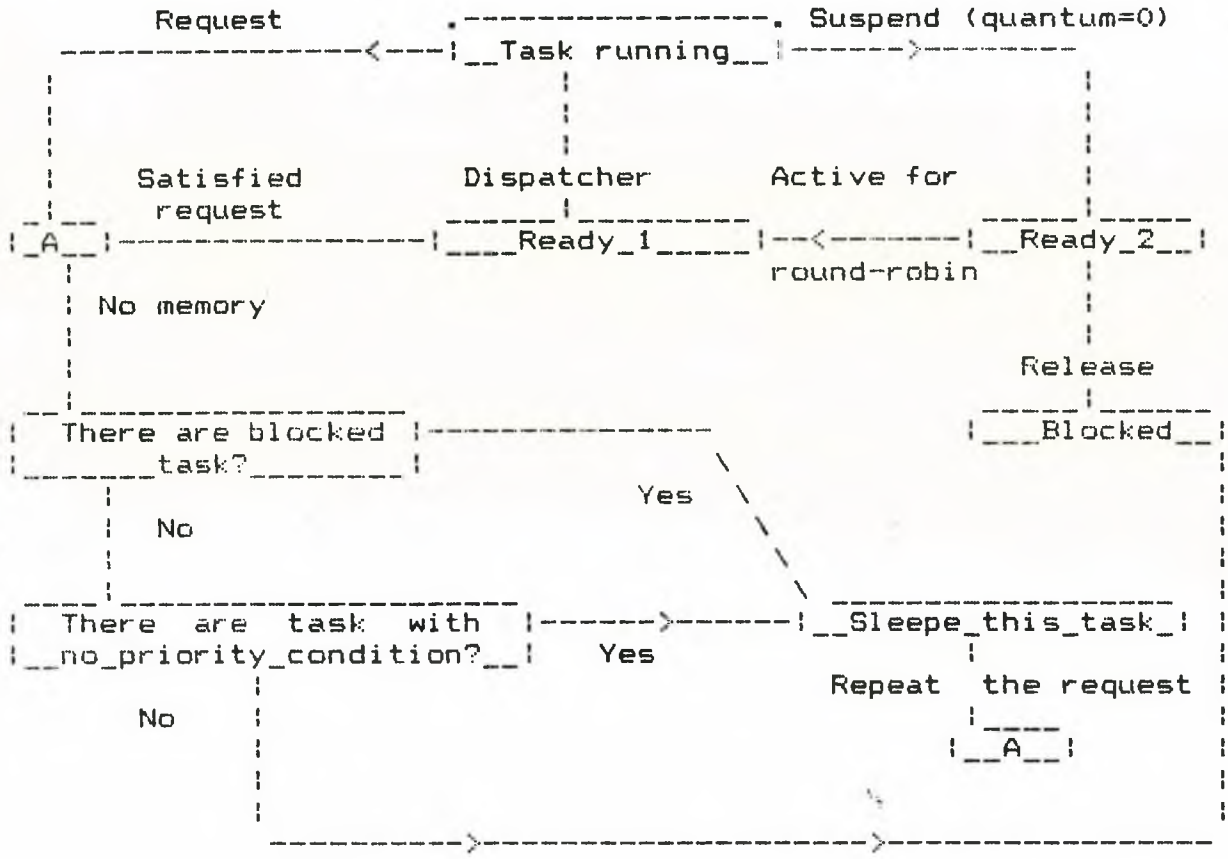
b) If no task is awaiting terminal data input it is determined whether or not the requesting task has priority.

If the requesting task has no priority, it become sleeped.

If the requesting task has priority the non priority task will be blocked in order to satisfy the memory request. In other case, the requesting task is sleeped.

The figure summarizes the possible status changes and the

operations causing these changes.



This approach simplifies organization of the dispatcher by restricting resources to be distributed to: the processor (DBMS) and the system's central memory. This is an apt model for the solution of the multitasking problem of our system CID 300/10. Even though in dBASE-300 the dispatcher has not been implemented as yet, a similar model was tested with COBOL tasks and the

results were satisfactory.

Further to the above features, the primitives for accessing procedures were studied and they were completed for line-printer output; in the case of display management new primitives were designed which enable full-screen operations of the system. The basic procedure of the FOBOS system to control the character output was complemented by checking the SET FORMAT TO PRINT/SCREEN and SET CONSOLE ON/OFF associated with character output and the recovery due to failure of the output equipment having been improved. For managing the display, full-screen procedures were implemented to erase the screen and to place the cursor on any X,Y point of the display.

3.2 Distribution of the main memory.

In this section it is shown two main topics of our DBMS: the distribution of the system in main memory and the Dynamic Storage Allocation Module.

In the first theme, it is explained the different variants in order to make the distribution of the system in overlay regions and the study about the frequency of use of the commands and the algorithm used to make the distribution. Both technics are originals and they haven't been found in the consulted bibliography. Also it is explained a suggestion in order to increase the main memory available to the system.

In the second theme, it is explained the current technics about dynamic storage allocation and it is shown the adaptation of the selected method in ours environment and it connection with the other modules of the system.

3.2.1 Location system in main memory.

Owing to its complexity, the whole system cannot be located in the main memory. Therefore, the overlay technique had to be applied.

For the distribution of functions in the overlay regions, account was taken of the frequency of use of the commands. In this connection a statistical investigation was made on 4 major application systems performed in dBASE II.

i) System to control components of a hardware:

Number of programs: 23.

Total number of commands 1052.

ii) System to control resolutions emanated for governing boards in enterprises:

Number of programs: 40.

Total number of commands: 2572.

iii) System to control documents handled by enterprises:

Number of programs: 47.

Total number of commands: 6320.

iv) System for aid to medical care applications:

Number of programs: 39.

Total number of commands: 1546.

The total number of programs in the 4 systems is: 149.

The total number of commands in the 4 systems is: 11490.

It should be noted that these applications have been selected from widely differing fields, which enables us to draw general conclusions.

From the total set of commands it was shown that the per cent of use of the principal commands behaves as follows:

USE/CLOSE	4.03%	DO/ENDDO	7.06%	IF/ENDIF	9.96%
SAY/GET	18.15%	GOTO/SKIP	12.11%	DELETE	0.45%
STORE	18.51%	REPLACE	12.35%	REPORT	0.19%
DEL.FILE	0.30%	SAVE/RESTORE	1.45%	LOCATE	8.15%
ACCEPT	1.03%	INPUT	0.19%	WAIT	0.26%
SET	1.50%	JOIN	0.11%	RENAME	0.11%
Index.Comm.	2.25%	TOTAL	0.0%	UPDATE	0.0%
SORT	0.0%				

There are not included others commands that have only use in maintaining and creation of the bases.

A particular algorithm was designed to perform the distribution of the main memory based in the per cent of use of commands and its requirement of memory. The algorithm depends of one parameter: the number of overlay regions. From this parameter it selects the more used commands and makes main segment in each

overlay region (head segment). In the next step, the memory defined by the head segments it is completed. Therefore, all overlay region has one j such as:

Segment(j)% > Segment(i)% for all i<>j

where

Segment(k)% = $\sum_{i=1}^{n}$ Ci% .and Ci = command or function included in the segment

Also, the algorithm try to meet joined the modules with 0.0% of use (interactive commands or commands with low use) to increase the probability that a command with per cent greater than zero will be in main memory when it will be used. Owing to its relation, some commands or functions need to be joined in the same segment. From the algorithm, these commands or functions are only one with the sum of their particular per cents and requirements of main memory.

The overlay regions selected are the following:

a)	Root segment (13.2 k)
b)	Overlay region 1 (2.5-k)
c)	Overlay region 2 (1.4 k)
d)	Overlay region 3 (5 k)
e)	Overlay region 4 (0.5 k)

a) Root segment: contains the system's internal variables, the general subroutines, the monitor, syntax analyzers, semantic subroutines, the module for generating object code, a module for processing arithmetical-logical expressions, etc.

b) Overlay region 1: this region is formed by the following overlay segments:

b1) Table of syntax analyzers.

b2) Processors for the commands JOIN, DELETE FILE, RENAME, DISPLAY FILES, COPY STRUCTURE, SAVE and RESTORE.

b3) Processors for the commands TOTAL, UPDATE, SORT.

b4) Processors for the commands MODIFY COMMAND, MODIFY STRUCTURE, and DISPLAY STATUS.

b5) Subroutines for the management of file indexes.

b6) A procedure for the management of errors.

In this overlay region it is accomplish the following:

(b1 segment)% = 86.4 based on the frequency of use of the commands that need be compiled with the syntax analyzer.

(b2 segment)% = (JOIN)% + (DELETE-FILE)% + (RENAME)% + (DISPLAY-FILES)% + (COPY-STRU)% + (SAVE)% + (RESTORE)%

but DISPLAY-FILE, COPY STRUCTURE as well as MODIFY COMMAND, EDIT, etc. are commands useful in interactive process; that is the reason because they have 0% in our account for the small overhead introduce in the system.

Then

(b2 segment)% = 0.11 + 0.30 + 0.11 + 1.45

(b2 segment)% = 1.97

In the same form it obtains:

(b3 segment)% = 0.0

(b4 segment)% = 0.0

(b5 segment)% = 2.25

(b6 segment)% = 0.0

c) Overlay region 2: this region contains the following overlay segments:

c1) File Control System routines in charge of base opening and closing procedures as well as procedures of input/output canal allocation.

c2) File Control System routines in charge of record read/write procedures.

In this region, processes not performed simultaneously have been located in separated segments: to open and close a file and read/write on the file. In this region the more frequency of use is determinate by the read/write module since generally a base is opened and close once in the program.

d) Overlay region 3: this region contains the following overlay segments:

d1) Processors for commands USE, DO, IF, LOCATE, SAY and GET.

d2) Processors for commands CREATE, APPEND, EDIT, and DISPLAY.

d3) Processors for commands COUNT, ACCEPT, INPUT, SUM, WAIT,

and SET.

d4) Processors for commands REPORT.

d5) Buffer of command SORT.

This region was designed according to the frequencies of use, so that commands USE, DO, IF, LOCATE and SAY/GET should remain in the main memory as far as possible since they are the more frequently used in this region.

e) Overlay region 4: this region is made up of two overlay segments:

e1) Processors for commands STORE and REPLACE.

e2) Processor for command HELP.

From the region the segment corresponding to commands STORE and REPLACE, which have a very high frequency rate, is practically maintained in memory.

Finally, it's possible to make an evaluation in time about the overhead invested for the use of overlay technics.

In this approach it is supposed that one program execute in 100% of time with all the system in main memory (without overlay technics) i.e. the comparison unit will be the execution time of a program with all the system in main memory.

As regards in the overlay distributions it is obtained the following conclusions:

i) In the overlay region 1 the overhead is about 4.22%

ii) In the overlay region 2 the overhead is about 4.03% (is the

per cent due to the use of subroutines of USE/CLOSE commands)

iii) In the overlay region 3 the overhead is about 3.17%

iv) In the overlay region 4 the overhead is nule.

In brief, it is concluded that overhead for the use of overlay technics, in the worst case (because is not considered the case that one segment of overlay is already in main memory when it's solicited) is calculated by the following formula:

Overhead = 4.22% + 4.03% + 3.17%

Then, the total overhead introduced with this overlay distribution is about 11.42% .

As regards above distribution the following comment can be made: Overlay region 3 has a length of 5 k, determined by the longest segment, this is the set of buffers needed by command SORT for its operation. According to the results of the investigation on frequency of use of the commands and extending this investigation to other general applications, the author suggest that a way to gain memory would be removing command SORT from our system and make it a utility program. By this removal, the overlay region 3 is reduced by nearly 50%, in this case brought about by the processor for command REPORT. With this, about 2.3 k would be gained which could be used to enhance efficiency of the system.

3.2.2. Dynamic Storage Allocation Module.

Because of the characteristics of the work of the dispatcher proposed for dBASE-300, it was necessary to develop a dynamic storage allocation module since the FOBOS operating system does not supply this kind of feature.

The dispatcher of dBASE-300 must handle, at a given moment, memory areas for the loading of tasks to be executed. These areas are not always of the same length and once the execution is finished, they are not needed any more. By the same token, the File Control System requires memory for buffer areas and memory for record areas in order to accomplish its work with the files, and again this memory is necessary only when the file is opened and is no longer necessary after that the file was closed. Also, the length of the record area is not a fixed one, it being dependent on the record length itself.

From the above analysis, the necessity ensues of a module that will manage the memory in a dynamic way.

For this purpose the following methods were studied and evaluated for possible selection: first-fit, best-fit, boundary tags, and buddy system /KNUTH73/.

After considering the best-fit method, it has arrived at the conclusion that it is rather slow since it requires a long search, and our main objection is that it tends to increase the

fragmentation phenomena --meaning that best-fit increases the number of small size blocks, which is not desirable since this decreases the possibility of fulfilling memory requests of larger sizes.

On the other hand, boundary tags requires extra storage for TAG fields on both sides of the block and for double linking and in fact it only improves the process of block liberation.

The buddy system also requires more storage to maintain the structure of separate lists of power 2 size blocks available. Besides, it always allocates power 2 size blocks and then there is not really requested memory allocated, which is not convenient in our case because of the restricted capacity of main memory available.

So, the method that was chosen and implemented was first-fit, with the improvement of going through the list always from the last block allocated in order to avoid fragmentation at the beginning of the list of available blocks. Also to avoid fragmentation and save time, a block is allocated when the difference between the size requested and the size of that block is less than 10 as suggested in /KNUTH73/.

Block liberation is done by checking whether it is possible to merge the liberated block with the adjacent blocks, if they are available. And, finally, a process of memory compacting was added to be used when all available blocks together could fulfill the

memory request.

In practice, it is verified the "fifty per cent rule" proven in /KNUTH73/, which asserts that "the average number of available blocks tends to approximate 1/2(pN) where N is the number of reserved blocks in the system on an equilibrium condition and p is the probability of the difference between the available and requested amounts of memory, this difference being not equal to zero (or greater than a constant and predetermined value)".

The behavior of the algorithm is also excellent as to the number of inspections of available blocks.

3.3 Language compiler.

Our DBMS dBASE-300 includes a common language to describe and to manage the data. The language is compatible with the Assembly Language Relational Database Management System dBASE II.

It is explained the method of processing the language as well as the organisation of the tables of symbols. An Automata Generator System was used in the syntactic analysis. This system allows to perform the syntactic analysis in a easier way.

Because of low memory capacity and the complexity of the grammar it was necessary to accomplish several variants which are detailed. The selected method is formalized. All of these is performed based on different applications of automata theory and the result is an original method of automata implementation of

complex grammars on configurations of low memory capacity.
The actual methods of symbol table handling are exposed and the selected methods are theoretically and practically discussed.
In this paper it does not go into details of the compiler technique: scanner and the semantic analyzer since these follow a classic structure. Object code generation will be discussed in the section dealing with the run-time system.

3.3.1. Syntax analysis.

In programming or writing compilers three problems must be solved: syntax analysis, lexicological analysis, and code generation or semantic management.
At the present level of development of the theory of languages and compilers, it is almost an established principle that from the three problems mentioned above only the third one requires a heavy treatment by the writers of compilers, i.e. the lexicon, and especially the syntax, should be a frame only for semantic management. The writer of compilers should find a syntactical method flexible enough to enable adequate semantic management.
In order to achieve the automation of syntax analyzers, a wide variety of syntax analyzer generators have been developed from the description of a language grammar described in Backus's Normal Form /BACKU59/, or in some other similar manner to provide

the compiler writer with the syntax analyzer (parser).

The methods of syntax analysis can be divided into two larg groups: the ascending ones which build the syntax tree of recognition from the chain been analyzed up to the generation's syntax auxiliary, and the descending ones which, starting from the generation's syntax auxiliary, work down to the text to be recognized.

Within the recognisers of the ascending type, techniques of precedence have been successful, and also that defined by Knuth in 1965 /KNUTH65/, the LR technique, which has been widely accepted for theoretical and practical studies. From this, the techniques SLR and LALR, defined by De Remer /DEREM71/ in the late sixties, have been derived.

With particular reference to LALR(1) techniques, a syntax analyzer generator was implemented in 1971 starting from LALR(1) grammars, that is, grammars to which the LALR(1) technique is applicable, thus demonstrating the feasibility of generating analyzers of this type /LALON71/.

In 1972, a Syntax Analyzer Generator (SAG) similar to the former was built at the Centro de Investigacion Digital /MUNIZ76/, from which tables were generated which enabled syntax analysis of ALGOL 60 and COBOL compilers for the CID 201-B minicomputer, the effectiveness of this method of analysis having been demonstrated in both.

The method of analysis LALR(1) in the form of a program guided by tables has the required flexibility as mentioned above. It possesses qualities which render it effective as a base for a SAG, among which are the following:

- Determinism in the analysis.
- Easy interaction with lexicologic and semantic management.
- Instantaneous detection of errors.
- Generality, in the sense that the class of LALR(1) grammars is wide.
- Easy alteration of language syntax.
- Easy recovery of syntax errors.
- Valuable parameters with respect to memory required and speed of analysis.

The method of analysis with LALR(1) grammars should be regarded as the programming of a deterministic pushdown automata that performs the syntax analysis by making adequated changes in its states.

The types of states present in this automata are three: applications, read, and look-ahead states. For each rule or production in the grammar there is a state of application, these states being the suitable ones to interact with the semantic management of the compilation process. The action that takes place in these may be resumed, from a properly syntactical viewpoint, into two tasks:

- Reducing or increasing the analyzer stack according to the number of symbols present on the right portion of the rule which produced it.
- Comparing the top of the stack with the first component of a set of associated pairs and coming up to the state indicated by the second component of the corresponding pair.

The reading states' function is to read the chain of which analysis is desired as to whether or not it belongs to the language. The syntactical action associated with them is one of reading the symbol and comparing it with one or more that can occur in the state, coming up to the corresponding destination of the matched symbol. If the symbol being read is not among those legal in that state. a syntax error is detected.

The look-ahead states have their origin in the automata's need to look at a symbol in the chain (that is, the head) in order to determine which one is to be the next state that will enable proper continuation of the analysis. The associated syntactical action is to ascertain in what branch the looked symbol is located and come up to the corresponding destination. If the symbol is not found in any of the branches then a syntax error is detected.

These two states are the ones which interact with the lexicologic analyzer in the compilation process. In conceiving the look-ahead states, it should always be remembered that these states do

not read but only look at the symbol.

The method of analysis operates with a stack into which the reading states are pushed as they are consulted. the necessary history being maintained in the stack that enables continuation of syntax analysis.

Our experience with the use of the SAG of CID 201-B began with the COBOL compiler written for the CID 201-B, with which the grammar corresponding to PROCEDURE DIVISION was processed. At that time the tables resulting from SAG was a listing that had to be manually loaded and optimized. Notwithstanding this difficulty, the use of this technology increased the efficiency of the System and the speed of its setting. We will not go further into this application because the current version of SAG is an improvement by which, besides the listing with all the states, terminals, non terminals, etc., a paper tape is obtained containing the tables already coded and optimized.

In the case of the COBOL compiler for the CID 300/10 it is decided to use this method for the syntax analysis of the whole language.

Starting from the syntax of the COBOL sentences, the grammar for the language was designed. However, due to the structure of the compiler, which owing to memory capacity problems is divided into overlay regions, the language's grammar was divided into three parts: one for compiling IDENTIFICATION DIVISION and ENVIRONMENT

DIVISION, one for compiling DATA DIVISION, and another one for PROCEDURE DIVISION. With each of these parts the SAG was used to obtain the tables and thus the corresponding parser.

The procedure followed is simple: the compiler has an initial state to detected the symbol IDENTIFICATION, then it delivers control to the parser of the first two divisions, which finishes upon detecting the terminal symbol DATA. Next the parser of DATA DIVISION is loaded and given control, which ends when symbol PROCEDURE is detected, thus loading and delivering control to the parser of this division, which finishes when end of compilation is detected.

The grammar of PROCEDURE DIVISION has a special characteristic in so far as it was necessary to divide the parser into two parts owing to memory capacity problems with CID 201-B when this grammar was processed (it has the highest number of rules and is the more complex due to recursiveness). So there exists two grammars named "PROCEDURE P" and "PROCEDURE S", with which the syntax analysis of COBOL instructions is performed.

There were grouped in the "PROCEDURE P" grammar, the conditional sentences of COBOL and all grammatic rules presenting recursiveness while in the "PROCEDURE S" grammar the imperative sentences of the language were placed.

The main problem with this approach was the way in which the change from one table to another was to be carried out (in the

same memory area) so that it should be transparent to the compiler. The following is a description of the solution adopted. Parser "PROCEDURE P" was the first to receive control, as has been explained, upon detection of the word PROCEDURE. For this analyzer, imperative sentences of the language such as ACCEPT, DISPLAY, etc., are terminals. This enables one state alone of "multiple reading" generated by the analyzer to detect the type of instruction being managed. If it was a conditional instruction, its analysis can be done within the analyzer "PROCEDURE P".

In case of an imperative instruction, when the word identifying the instruction is detected (it appears as a terminal), control is given to a special procedure (it appears in the compiler as one more semantic subroutine) that substitutes in the compiler the parameters of the tables generated by "PROCEDURE P" grammar by those of "PROCEDURE S", adequately placing certain indicators within the compiler, and continuing the processing by analyzer "PROCEDURE S". When the end of the instruction is detected, control is given to another special procedure (it appears in the compiler as a semantic subroutine) that carries out the process in reverse order, and processing then continues by analyzer "PROCEDURE P".

It is to be noted that the side effect of this solution is a reduction in the speed of compilation, since the grammar tables

have to be exchanged when an imperative instruction appears in the source program.

In this manner it was able to apply the SAG in the COBOL compiler for CID 300/10 and solve the problems.

With the experience obtained in applying the SAG extensive, complex grammars, it is tackled the design of the compiler of the dBASE-300 language, having in mind to process its grammar in automated form. A grammar was designed that included all sentences in the language. This first grammar, however, could not entirely be processed by SAG. At this point it is shown the deficiencies and shortcomings of SAG so that the final decision adopted be understood.

The Syntax Analyser Generator (SAG) written for the minicomputer CID 201-B with 32 K words of central memory (like PDP-8), behaves like an independent program, that is, it operates with no Operating System. Due to memory restrictions, it was not possible to go deep into the diagnosis of errors; both the syntax errors of grammar (due to ambiguity of common errors in punching) and the error of exceeded capacity causes the program to stop execution without issuing any specific error message, so the user must go into an analysis of its whole grammar and find with a pragmatic approach of "trial and error" a solution of the problem. This is a slow, tedious process in which the grammar must be rewritten again and again, and executed by SAG.

This process took, in general, more time than foreseen and it has to make a decision to obtain our parser through the automated method (produced by SAG) and the ad hoc method.

To achieve this end it is gradually reduced the initial grammar and finally obtained a grammar that can successfully be executed by SAG, the generated tables are loaded and the rest of the sentences are processed by the ad hoc method. To accomplish this it was necessary to alter both the scanner and the parser so that it contemplated the work with the two methods. Every command of the dBASE-300 language is processed by SAG except those that are simple (formed by terminal symbols only) and the arithmetic expressions which upon the parser detecting terminal symbols with which an arithmetic expression can be started, gives control to a module called "arithmetic scanner" that processes the expression by ad hoc method. When detecting a symbol not belonging to the expression the "arithmetic scanner" returns control to the parser delivering the read symbol as not read so as not to affect the syntactic analysis.

The use of a Syntax Analyzer Generator allows to increase the performance of the programming techniques in any process that requires a complex syntactic analysis.

Using SAG for CID 201-B it was possible to verify the theoretical advantages that have been described in the introduction of this paper, although due to peculiarities of the implementation these

advantages are reduced. In any case, it has been shown that its application may with satisfactory results be accepted as a "partial" solution for the syntax analysis of complex grammars.

3.3.2 Organisation of the Tables of Symbols.

One fundamental element of a compiler is the set of symbols tables. These tables are used by all other components of the compiler, hence the systems's efficiency depends very much on their structure, organization and handling.

In our system, the symbol tables are those corresponding to fields of a database or a data dictionary, the memory variables, and the language's reserved words.

Here, the actual method of symbol table handling are exposed and the selected methods are theoretically and practically discussed.

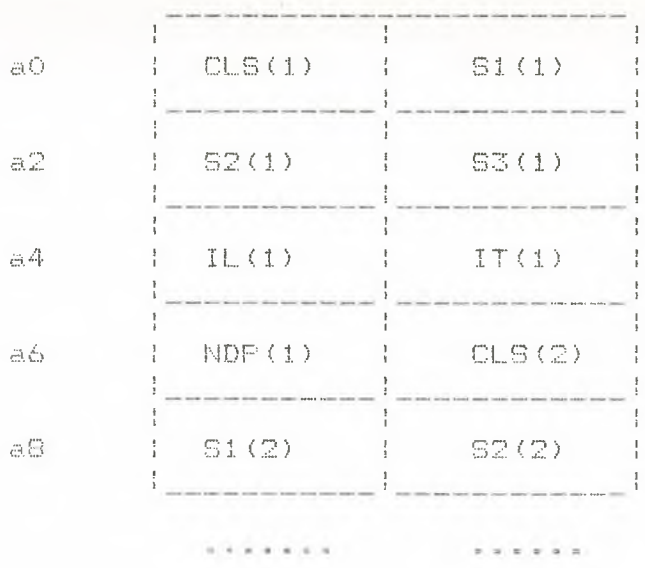
3.3.2.1 Data Dictionary.

In order to achieve data independence in a DBMS it is fundamental to have a data dictionary. The data dictionary must contain the number of each date item, its physical location, type, length, individual specifications and, in some cases, the synonyms and the level of access /MARTI77/.

The data items dictionary that must be univocally related to the data may have its place either in the same physical area of the

data or in another file. In our case the dictionary was included in the already existing label so that the files should be compatible with those of COBOL. This label is the first block of 512 characters of every file. From this block, COBOL uses only 92 characters, the dictionary being located in the rest of the block.

To organise the dictionary, it was considerated that for the system's requirements the fields must be reviewed following the order in which they were created and so we chose the following structure:



Byte a0 contains the Characters Length of the Symbol (CLS), that is, if it is stated an item called SALARY, CLS will have a 6 in binary. Next appear the characters of the item's name (S1, S2, S3, ...). Then byte Item Length (IL) shows the total number of characters contained in the field while Item Type (IT) contains

character N, C, or L depending on whether the stated item is numerical, chain of characters or logical, respectively. Lastly, byte Number of Decimal Places (NDP) tells us if the item is numerical, the amount of characters after of the decimal point.

The address of the item within the record does not have to be stored, since the table is sequentially reviewed and the contains of bytes IL are summed up, so that when the required item is found it has the displacement within the record (address). That is, if Di is the displacement of item i,

$$Di = \sum_{j=1}^{i-1} IL(j) \qquad \text{for } i <= 1$$

where IL(1) = 0

To detect the end of the table, a zero is placed in the entry following that corresponding to the last item.

The table length is fixed but the number of items depends on the number of characters in each symbol.

3.3.2.2 Table of memory variables.

With dBASE-300 exits the concept of memory variables, whose function is store both constants and the variables that are not connection with any file, such as indicators, pointers, etc.

From a general viewpoint, the characteristics of an item in a file and a memory variable are similar, only conceptually differing in the fact that the former resides in a file and the

latter resides in a memory zone. This structural similarity was used to organise the table of memory variables in a same way to the dictionary, which enables to unify the routines and managements of access to both tables. Also, here we wanted to avoid sorting the table on account of its inevitable overhead. Again, this table is dynamically increased or decreased and therefore, if we apply some hash technique we would have to select an area large enough for the table or apply some of the solutions proposed for hash techniques about variable length tables whose programming becomes more complicated. For all the above, we selected for memory variables a structure and organisation same as those explained for the data dictionary, with the difference that following the describer of the data appear the data's contents.

This organisation might be improved to obtain more efficiency regarding time of access to a data. A hash table could be organised having a block of access according to the calculated hash and from this point of departure to have the data sequentially linked (direct chaining) /MORRI68/.

This method would be more time-efficient since it reduces the number of average comparisons from n/2 in the first method to

$E = 1 + a/2$ where $a = k/n$ and k is the number of occupied entries, n is the size of the table, but it has a drawback in the consumption of additional memory and non

compatibility with the structure of the data dictionary, which involves reprogramming different procedures for each one.
In the event of a version being obtained where central memory would not be a critical problem, the method of organisation of the memory variables table should be improved by the above described method with the resulting increase in efficiency.

3.3.2.3 Language reserved words table.

In this table, the words belonging to the language of the system's command are stored. This is the more widely used in the System since its elements are the language's grammar's terminal words and for this reason the scanner uses it at least once for each command of a program. It has been stated that compilers often spend 20 percent of their time in this function /BELL70/. Thus this table's speed of access directly influences the general efficiency of the System.
In designing this table the fact was taken into consideration that the elements (reserved words) are of variable sizes and that there are 171 symbols.
Based on all these facts a hash scheme was selected to organise the table using a logical method to calculate the function applied to all symbols' characters and then selecting the central bits to avoid clusters. To deal with collisions the Random Probing method developed by Morris /MORRI68/ was chosen, which

guarantees an average number of trials given by:

$$E = N/k \int_{c}^{\infty} dx/(1-x)$$

$$E = -(1/\alpha)\log(1-\alpha)$$

where k : number of symbols in the table (171)

N : table size (256)

thus

$$E = 1.65$$

The main disadvantage of this method is that elimination of symbols becomes a troublesome process. However, in our case this does not occur inasmuch as symbols (words of the language) are not removed.

Evaluation was also made of the organisation method based on the Binary search /PRICE71/, which fulfilled the requirements of table classification and of being a fixed number of elements (it grows dynamically, insert has to be made to keep the table classified). This approach is easier to program but it has the following drawbacks:

a) The average of trials is given by:

   $E = \ln(N)$, where N is the table size, thus in our case it would be E = 5.14, which is almost 4 times the average number of trials needed to find a symbol in the table as against the selected method.

b) The scheme with Binary Search demands that symbols should be

of even length, which in our case involves making all our words into 10 characters, thus losing 445 words on this account as against 256 words that are saved in removing the area of entries for the hash method.

Our conclusion is, therefore, that the selected method fulfills the System's requirements.

3.4 Run-time system.

In this section it has included the main characteristics of the generated Object Code and the File Control System.

In the topic of object code it is explained the characteristics of the selective retrieval of the information present in those commands which manipulate data base. This allows to obtain a modular and compact code for every command, independently of the retrieval characteristics. The developed algorithm is original and it does not appear in the specialized literature.

In the topic of File Control System it is explained considering three logical modules: file management, basic functions and indexing treatment based in the B+ tree technique.

The basic file organization is described as well as the basic functions concerning the file management.

3.4.1 Characteristics of the Object Code.

To begin with, it will discuss the parameters, buffers and lists that play a role in the executer.

The part of the compiler which generates object code operates over an area of memory called Generation Buffer. This is a 512-byte block that stores the contents of the object code from a transaction. To us, a transaction is equivalent to language command of the dBASE-300. The Generation Buffer is controlled by two pointers: the buffer's initial pointer (INIPO), and a pointer that signals the end of the object code chain (FINPOB).

This Generation Buffer was designed having in mind the optional possibility of developing a pure procedure of compilation, generation of the object program, and subsequent execution of the object program, that is, not in interpreter form, thus increasing the speed of execution of the system since no compiling of the commands is to be done each time they are to be executed. To do this, object code would have to be generated sequentially through the Generation Buffer and, on reaching the end of the block, record in a file containing the desired object program.

The method of object code representation used is the Polish representation with an arithmetical-logical accumulator for the results.

To address a memory variable the address of the variable

describer is employed since all the necessary data appear in its describer, and then its contents are found.

To address a file item, three parameters are required:

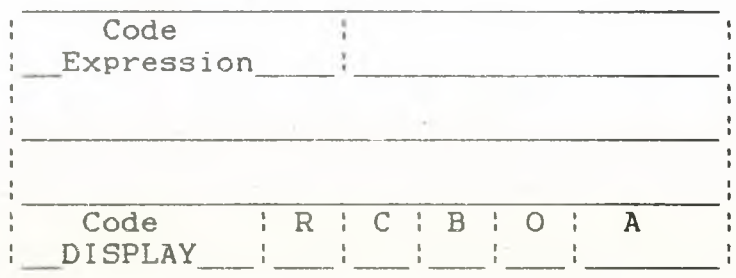
a) There is a list of open files that enables accessing all the data in the file (address of the record in memory, address of the data dictionary, etc.). The first parameter is the position of the file in the list.

b) Address of the item's describer. This describer is found in the files's data dictionary and its contents are explained in the Table of Symbols (3.3.2.1).

c) Location within the record. The item is addressed relative to commencement of the record so that in order to calculate its absolute address this parameter is added to the address of commencement of the record (this is provided by the File Control System).

The object code is made up by a first byte containing the code of the function to be performed, afterwards a byte with individual options of each function and then the word to address the memory variables, item of files or void if the command does not require them. For an example, let us see the object code corresponding to the command:

DISPLAY/LIST <expression>



The code of the expression corresponds to the arithmetical representation in Polish notation, which, on being executed, leaves the result in the arithmetical-logical-accumulator, where it is taken by the execution of the function corresponding to command DISPLAY. The parameters R, C, B, O, and A correspond to the different characteristics of the DISPLAY/LIST command.

Now let us see how language options enabling selective retrieval of information in a database were implemented. These options are:

a) SCOPE: is used to specify the application range of a given command. It has 3 possible values:

ALL: the command affects the whole file, that is, the file must be "rewinded" and the command applied to all records in the file.

NEXT n: means that the command is applied to next n records.

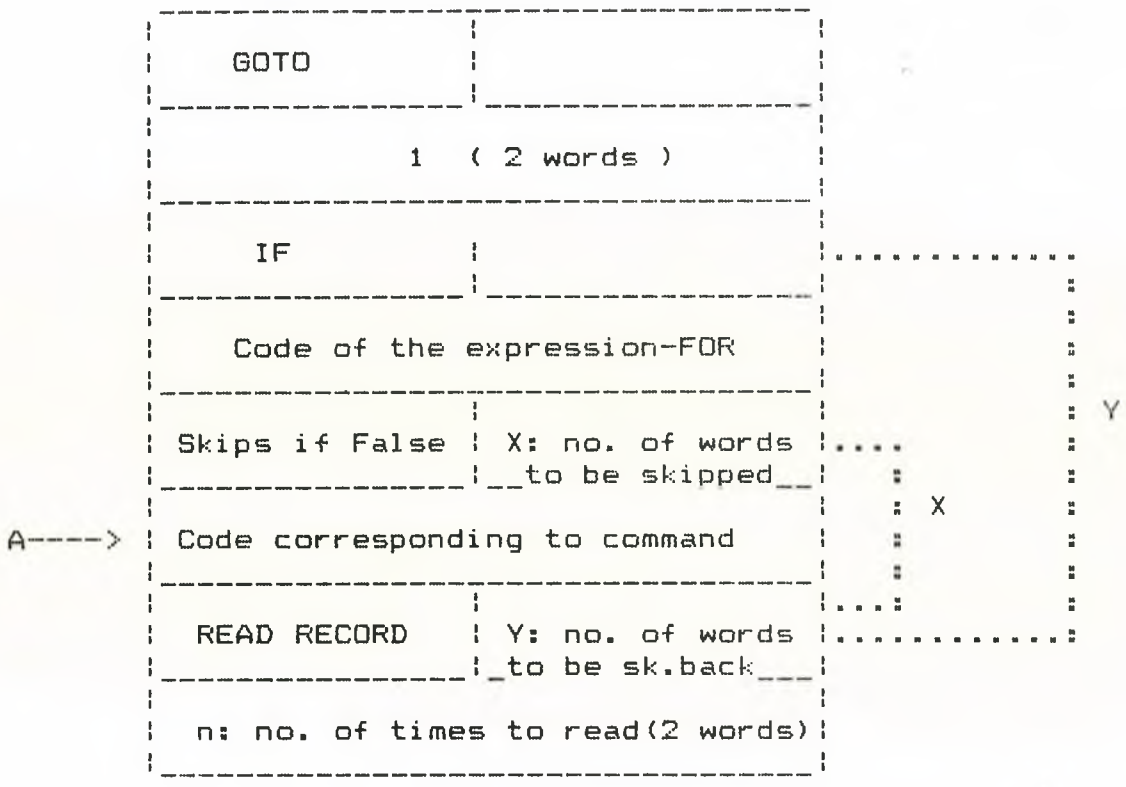
RECORD n: means that the command is applied to record no. n only.

b) FOR <exp>: implies that in all records of the database the logical expression is to be evaluated and the command be applied to all records whose expression has a true value.

c) WHILE <exp>: all records are examined, starting with the cursor record. The command stops its execution from the moment expression is false.

These options may be combined with all commands that make reference to records of a file and therefore their implementation is a complex one if each function of the executer has to contemplate these options. This problem was solved by means of the object code and performing several special functions in the executer.

Let us see how option FOR is solved.



The first code generated is a function GOTO to the first recording the file, which solves the rewind operation of the file implied by option FOR. Then is generated an evaluation of the logical expression used in the FOR, which leaves the result in the arithmetical accumulator. The function "SKIPS IF FALSE" gives, according to the result of the arithmetical accumulator, the control X words ahead if the result of the logical expression was false, or gives control to the following function if it was true. The following function (A) is that corresponding to generation of object code of the command without SCOPE, FOR or WHILE. Lastly, a function "READ RECORD" is generated which

performs the reading of a record, decreases counter n and, if the counter is other than zero, gives control Y words back. If the counter is zero, it ends with the cycle that has been set. In the event option FOR, "Y" has the maximum value to be set on the 2 words of the code, that is, the cycle is run until end-of-file (EOF) is found.

This structure is generated using procedures included in the compiler in a form transparent to the commands, that is, the routines of object code of the commands do not have into account options SCOPE, FOR or WHILE. Also,

a) ALL: is a particular case of FOR assuming the null expression.

b) NEXT c: initial function "GOTO" is not generated. The value of c-1 is placed in parameter n of the function "READ RECORD".

c) RECORD g: generates initial function "GOTO" with value of "g" as the parameter. Does not generate the function "READ ARTICLE".

Also in the same scheme the case of option WHILE is solved. However, initial function "GOTO" is not generated and the parameter of skip "X" generated by function "SKIPS IF FALSE" signals end of the cycle, not the function "READ RECORD".

3.4.2 File Control System.

The File Control System of dBASE-300 may be considered logically divided into three modules:

- file management
- basic functions
- indexing treatment

File management.

The basic file organization in dBASE-300, without considering indexing treatment,is a relative organization; that is, each record is identified by the position it occupies in the file at creation time.

This kind of organization enables fulfilling the requirements of all transactions referred to files in dBASE-300 and also allows easy connection with the structures defined for indexing treatment; this means that it is not necessary to use any randomising formula or overflow areas.

The chosen organization and the structures defined for its implementation enable dBASE-300 to be compatible with COBOL files with relative organization specified with BLOCK CONTAINS clause. This feature satisfies the compatibility requirement with the software developed up to date.

BASIC FUNCTIONS

The primitives concerning the file management which OS FOBOS

supplies are elemental. They are not designed for relative organization, nor to manage data grouped in fixed size records and records grouped in blocks.

Because of this, it was necessary to add some other primitives to the system in order to accomplish the requirements of the File Control System of dBASE-300.

These new primitives are supported, of course, on those elemental primitives of the OS and make a good use of them.

This additional primitives added are the basic tools for file management and also for the indexing treatment, because these primitives carry out the operations in the file.

Besides this information in the label, the File Control System keeps in memory, during the work with a file, some information which helps it perform correctly and efficiently the dBASE-300 file transactions.

This information table is called "open file table" and it is placed previous to the file label in dynamic memory as long as the file is open.

Buffer area, map area and record area are also placed in dynamic area. This means that by using the Dynamic Storage Allocation Module, memory used in the treatment of a file is released when it is closed. In other words, the memory is occupied only while the file is open.

Indexing treatment.

In the following it is shown the method implemented for index treatment in the Data Base Management System dBASE-300.
The technique employed to treat indexes in dBASE-300 is the organisation called B+ tree. With the use of this technique it is possible to accomplish all transactions defined in dBASE-300 in a suitable way as it is efficient for both random and sequential access to records and modification operations.
In order to have a better understanding of the implementation of B+ tree in dBASE-300 File Control System, the analysis is divided into the following sections:

- B-trees and their data structures.
- Find, Insertion and Deletion.
- Operation costs.
- B+ trees.
- Other variants of B-trees.

The first two sections are referred to B-trees because the characteristics explained are also present in B+ trees.

B-trees and their data structures.
The structure implemented for index treatment in dBASE-300 corresponds to the B-tree definition given in /BAYER72/, i.e., it has the following properties:

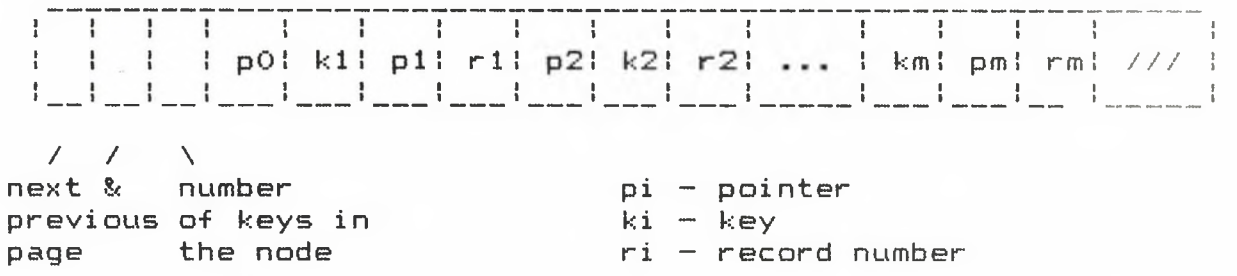
a) Each path from the root to any leaf has the same length.

b) Each node, except the root and leaves, has at least k+1 sons, where k is a natural number. The root is a leaf and has at at least two sons.

c) Each node has at most 2k+1 sons.

A node of a B-tree is the page in which the index is stored.

A page of the B-tree in dBASE-300 looks like this:



This data structure of a page has the following properties /BAYER72/:

a) Each page contains between k and 2k keys except the root page which may contain between 1 and 2k keys.

b) Let m be the number of keys on a page (not a leaf). P has m+1 sons.

c) The keys in a page are sequential in increasing order and each page contains at most 2k keys and 2k+1 pointers.

d) Let P(pi) be the page to which pi points, let K(pi) be the set of keys on the pages of that maximal subtree of which

P(pi) is the root. Then the following conditions always hold:

$\forall$ y $\in$ K(p0) $\Rightarrow$ y<k1

$\forall$ y $\in$ K(pi) $\Rightarrow$ ki<y<ki+1 ; 1$\leq$i<m

$\forall$ y $\in$ K(pm) $\Rightarrow$ km<y

The record number behind each key in the page enables to access directly the requested record in the relative file.

This information in the page is a peculiar characteristic of this implementation, taking advantage of the relative organisation of the data file.

Find, Insertion and Deletion.

Find.

A find operation in a B-tree of order k never visits more than $1+\log_k n$ pages, where n is the number of records in the file.

This is possible because record operations in a B-tree always leave the tree balanced.

The B-tree balancing scheme restricts changes in the tree to a single path from a leaf to the root, so it can not introduce "runaway" overhead /COMER79/.

The find algorithm is simple logically.

Insertion.

The insertion operation requires a previous find operation with which it is possible to get the page where the new key is going to be inserted.

Three cases may be found (assuming the new key is not present):

i) empty tree

ii) page not full

iii) page full

In the first case it is necessary to create a root page with the new key.

In the second case the new key may be inserted in the correct position.

In the third case it is necessary to split the page, that is, the smallest k keys are placed in one page, the largest k keys are placed in another page, and the remaining value is promoted to the parent page where it serves as a separator. In the worst case splitting propagates all the way to the root and the tree increases in height by one level.

Deletion.

Like insertion, the deletion operation needs a previous find operation to locate the key to be deleted.

Assuming the key is present, it is possible to find two cases:

a) the key is on leaf page

b) the key is not on a leaf page.

In the first case the key can be deleted from leaf.

In the second case it is necessary to find the adjacent key and place it in the position of the deleted key. To find the adjacent key in key-sequence order it is necessary to search for the leftmost leaf in the right subtree of the deleted key.

In both cases it is necessary to check whether an underflow condition is present, that is, if the leaf has less than k keys. In this case it is possible to redistribute the remaining keys between the two neighboring pages but only if there are at least 2k keys to distribute. If there are less than 2k keys it is necessary to perform a concatenation process, where keys are combined into one page and the other is discarded. Then the separating key in the ancestor is no longer necessary and is also added to the single remaining leaf.

As can be seen, the process of concatenation may force concatenating at the next higher level and so on, to the root of the level, and it is possible that B-tree decreases in height by 1.

Operation costs.

The cost of a find operation is increased with the growth of the

file size logarithm. This is:

$$h \leqslant \log_k \frac{n+1}{2} \qquad \text{/BAYER72/ /COMER79/}$$

where : h - height of the page tree

n - number of keys

k - minimum number of keys per page

Insertion and deletion take time proportional to log n in the worst case because these operations need additional access beyond the cost of a find operation as it progresses back up the tree. The costs are at most double, so that the height of the tree still is the main element for these costs.

As can be seen, the number of keys in a page is the parameter on which the performance of all operations depends.

Bayer and McCreight /BAYER72/ show a way to determine the optimal number of keys in a page to achieve minimum time per transaction, in terms of fixed time spent per page, transfer time, key size, and a factor for average page occupancy.

In the case of dBASE-300 there are practical limits to page size. The disks on which the system is supported are divided into fixed blocks of 512 bytes length.

Thus, each page in dBASE-300 is 512 bytes in order to avoid extra overhead in the process.

In practice, considering standard key size the behavior of the algorithm with this size page was quite suitable.

B+ trees.

The organisation chosen was B+ tree, a variant that avoids the problems with sequential processing in B-trees.

These problems are the requirement of extra space for at least $\log_k (n+1)$ pages /KNUTH73/ in main memory to avoid reading them twice in the sequential processing.

Also, the next operation may require to access several pages before finding the desired key.

In a B+ tree all keys reside in the leaves.

The relative organisation of a file in the dBASE-300 enables implementing B+ trees in a very comfortable way, because this file can be accessed directly once it has the number of correspondent record in the sequence order.

With B+ tree the logarithmic cost properties for operations by key are retained and the next operation for a sequential processing requires at most 1 access. Besides, no page will be accessed more than once.

On the other hand, the feature of B+ trees that all keys reside in the leaves allows to simplify the deletion operation because the removal of the key to be deleted is simple, and the tree need not be changed while the leaf remains al least half-full. It means that a copy of a deleted key will be present in the tree and can direct searches to the correct leaf.

Redistribution or concatenation process will be necessary only if

an underflow condition arises in the leaf and this process is similar to that in B-tree.

Other variants of B-trees.

Several variants of B-trees were analyzed in order to choose a technique for the implementation of index treatment in dBASE-300. These variants were: B* tree /KNUTH73/, Virtual B-trees /COMER79/, Compression technique /WAGNE73/, and Binary B-trees /BAYER71/.

B* tree was discarded because its implementation seems to be a little more complicated and then it requires more main memory with the resulting increase of overhead.

In the case of Virtual B-trees there is a practical inconvenience because our system computer does not have facilities of virtual memory.

With the compression technique pointers can be compressed using a displacement from a page address instead of the absolute address value. This technique is particulary useful for virtual B-trees where pointers take on large address values.

Finally, Binary B-trees are appropriated for a one-level store, because a Binary B-tree is a B-tree of order 1. This means that each page has 1 or 2 keys and 2 or 3 pointers.

B+ trees retains the logarithmic cost properties for operations

by key and has no problems with the next operation for a sequential processing.

In practice, B+ tree proved a very suitable organisation for index treatment in dBASE-300.

4. An application in dBASE-300.

In this chapter it is described a system to control resolutions emanated from governing boards in enterprises.

The system is actually used in many enterprises in our country. It was developed in dBASE-II for microcomputers of 8 and 16 bits. The system is composed by 40 programs with 2572 commands.

This system was chosen to run under our DBMS for CID 300/10, dBASE-300, in order to prove the almost compatibility between dBASE-II and dBASE-300 and also to prove that the changes which are necessary to made, are very simple and in an easy way the system to control resolutions was ready to run under dBASE-300.

This changes which were made, are also explained in this chapter.

This system has also a subsystem which controls the access to the system in order to protect the information against non authorized users. This control has three access priorities for the users, in order to allow read only access, read and modify and read, modify and delete, respectively. Thus, the first action which takes any function of the system is to ask the user number, in order to check its validity.

4.1. System to control resolutions emanated from governing boards in enterprises.

This system allows to control resolutions emanated by any document from governing boards in enterprises.

With this system it is possible to find a resolution by different criteria such as document which generates the resolution, date, who proposed the resolution, actual status, compliment date, as well as a listing of all resolutions by responsable of compliment, a listing of resolutions by compliment date and also statistics of resolution compliment and the amount of resolutions generated by a document.

This system also allows to introduce, modify or delete a resolution.

In order to accomplish the mentioned functions, the system works with two files:

RESOLUTION

RESPONSABLE

These files are treated indexed or not according to the characteristics of the specific function.

These two files are treated simultaneously and some actions taken in the RESOLUTION file could imply to take some other actions in the RESPONSABLE file.

The structure of RESOLUTION file is the following:

FIELD	LENGTH	TYPE
- resolution number	5	character
- proposed by	25	"
- number of document which generated the document	5	"
- document date	8	"
- resolution synthesis	240	"
- compliment status	2	"
- observations	160	"
- compliment date	15	"

The structure of RESPONSABLE file is the following:

FIELD	LENGTH	TYPE
- resolution number	5	character
- responsable	25	"

The RESOLUTION file is indexed by resolution number field and the RESPONSABLE file is also indexed by resolution number field, when it is necessary.

For a resolution number it is possible to find more than one record in the RESPONSABLE file, because in fact, a resolution could has more than one responsable of compliment.

In order to accomplish the functions of introduction, modification and deletion of resolutions, the system works with the following screen, easy to manipulate by the user.

RESOLUTION UPDATING

==========

RESOLUTION No.: _ PROPOSED BY:

SOURCE DOCUMENT No.: DATE:

RESOLUTION SYNTHESIS:

C O M P L I M E N T

DATE STATUS RESPONSABLE

O B S E R V A T I O N S

==========

To introduce a new resolution, the system shows the screen with each field empty, and the user proceeds to give the information field by field.

When the screen is complete, the files RESOLUTION and RESPONSABLE are updated.

Note that this screen generates records for both files mentioned above.

To modify a resolution the screen is shown initially empty. The resolution number is taken and its validity is checked. Then, the screen is filled with the information of the resolution and the user can change any field he desired.

It is possible also to go on forward or backward through the resolutions without doing any change, if it is necessary, only to see the resolutions.

Also in modify operation the file RESOLUTION and/or the file RESPONSABLE could be updated, if it is necessary.

In order to delete a resolution it is only necessary to give its number and the validity checking an other necessary operations are performed.

An interesting feature of the system is the resolution selection by different criteria. In order to accomplish this, the following screen is shown.

RESOLUTION SELECTION

SELECTION CRITERIA

1) Proposed by: 2) Source doc. number: 3) Date:

C O M P L I M E N T

4) Date: 5) Status: 6) Responsable:

Choice number: _

With this screen, the user can choose one or more criterium in order to obtain the listing of resolutions which hold these properties.

In the field corresponding to CHOICE NUMBER the user may type a digit between 1 and 6 and then the information in the space corresponding to the typed digit. It is possible to supply more than one choice.

The user can get the information through a screen or in a listing by the printer.

This function does not alter any file, but it accesses them in a

different way to bring a good response time to the user requests.

4.2. Adaptation to dBASE-300.

In order to adapt this system to dBASE-300 it was necessary to make only a few simple changes /DBASE86/ which are going to be detailed in the following.

- To change the symbol # by the symbols <> in comparisons.

If the symbol # could be used for comparisons and for the function " number of actual record" like in dBASE-II, the syntactical analysis would be more complicated and in order to simplify it, the symbols <> were used to specify comparisons.

- In dBASE-300 it is possible to handle more than two files simultaneously. Because of that the sense of the SELECT instruction change and it was necessary to change the SELECT PRIMARY and the SELECT SECONDARY by SELECT <number>.

For example, the following sequence :

```
USE FILEA
LIST
SELECT SECONDARY
USE FILEB
LIST
* editing file A
SELECT PRIMARY
EDIT
```

must be changed by:

```
USE FILEA
LIST
SELECT 2
USE FILEB
LIST
* editing file A
SELECT 1
EDIT
```

- The file specifications must change because of the different operating systems used by the micro and by the minicomputer.

For example, the following specification for CP/M:

```
USE B:FILENAME
```

must be changed by:

```
USE DKn:FILNAM
```

where n is the number of a disk unit.

5. Recommendations and conclusions.

In this chapter it is shown a summary of recommendations already included in another chapter and other recommendations observed during the practical work in dBASE 300.

- Considering the results of benchmark tests shown in chapter 2, it is proposed to implement a compiler for dBASE 300. In the chapter 3 it is shown how this compiler can be implemented. The compiler must increase the performance and its implementation is not very difficult because it was contemplated in the design of the system.

- Owing to the lack of memory and according to the result of commands used survey shown in chapter 3, which expresses a very low use of SORT command, it is proposed to remove the SORT command in order to reform the segmentation scheme to improve the general efficiency of the system. The procedure of data sorting could be included in an utility program.

- For the complementation of the system it is possible to develop some utility programs that dBASE II has in microcomputer environment. From these utilities the author proposes in priority order, to implement ZIP: a screen and printout form writer. With this program it is possible to compare dBASE 300 with SENSIBLE facilities to build screens.

- To make ease the moving of microcomputer applications to our

system it is possible to implement a utility program that converts the data base format of dBASE II into our dBASE 300 format. Besides, it is convenient to implement an utility program that convert the source program of dBASE II into dBASE 300 contemplating the few differences mentioned in chapter 4.

- In order to provide a software procedure in case of crash in accessing file process, it is possible to implement a Salvation Program. This program looks over data files and in an interactive way re-build the data loosed. It was discussed in 3.1.2 section.

Finally, as a conclusion it is shown the following:

The primary aims of this undertaking have been achieved with high quality level according to the results expressed in this paper.

This is the first integrated study in Cuba investigating the complex aspects about evaluation, selection, implementation, and application of DBMS. Also, it is the first time that this kind of study is reported for this type of cuban minicomputer.

It is developed a complete methodology to evaluate a DBMS in order to implement it. This methodology is used in our Institute in other similar works.

Several computing tools were adapted and some of them have been developed and improved according to our conditions.

With the acquisition of a DBMS for cuban minicomputer, the user is provided with a high technology for data processing.

Complemented with the other utilities of software, this system enables fast information retrieval from a data base created, maintained and controlled by different program packages already implemented. It enables the development of applications, with a high degree of data-program independence resulting in a saving of debbuging and maintenance time. It also enables exchanging application programs between microcomputer and CID 300/10, thanks to language compatibility.

dBASE 300 has been sold to many Organisms and Institutes in our country.

This work has been helpful in raising the technical level of those who use system CID 300/10 and at the same time has been instrumental in updating and improving the technique culture of the group under author's direction.

6. Bibliography.

/AHO72/ Aho A. V. and J. D. Ullman. The Theory of Parsing, Translation and Compiling. Volume 1: Parsing. Prentice-Hall, Inc., 1972.

/AISER71/ Aiserman M. A., et al. Logic, Automata, and Algorithms Academic Press, 1971.

/ARATO76/ Arato, M. A Note on Optimal Performance of Page Storage. Acta Cybernetica 3: 25-30, 1976.

/ARATO81/ Arato, M. and A. Benczur Dynamic Placement of Records and the Classical Occupancy Problem. Comp. and Math. with Appls., 7: 173-185, 1981.

/ARATO82/ Arato, M and A. Benczur. A General Treatment of Rearrangement Problems in a Linear Storage. Performance Evaluation, 2: 108-117, 1982.

/ASTRA80/ Astrahan, M. et al. Performance of the System R Access Path Selection Mechanism. Proc. IFIP Conf., 1980.

/BACKU59/ Backus, J.M. The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference. Proc. International Conf. on Information Processing, UNESCO, 1959.

/BAKER82/ Baker, R.A. Internal Data Base Management.

Datamation, May 1982.

/BAKON84/ Bakonyi, P. et al. A Microcomputer-Network Based Decision Support System for Health-Care Organizations.
c IFAC 9th. World Congress, 3: 1984.

/BARLE81/ Barley,K. and J. Driscoll. A Survey of Data-Base Management Systems for Microcomputers.
BYTE, Nov. 1981.

/BAYER71/ Bayer, R. Binary B-trees for Virtual Memory.
New York, Proc. 1971 ACM SIGFIDET Workshop, ACM, 219-235.

/BAYER72/ Bayer, R and C. McCreight, Organization and Maintenance of Large Ordered Indexes.
Acta Informatica 1(3): 173-189 1972.

/BEKES79/ Bekessy, A. and J. Demetrovics. Contribution to the Theory of Data Base Relations.
c North-Holland, Discrete Mathematics, 27: 1-10, 1979.

/BEKES80/ Bekessy, A., J. Demetrovics, L. Hannak. On the Number of Maximal Dependencies in a Data Base Relation of Fixed Order.
c North-Holland, Discrete Mathematics, 30: 83-88, 1980.

/BELL70/ Bell, J.R. The Quadratic Quotient Method: A Hash Code Eliminating Secondary Clustering.
Comm. ACM, Vol.(13)2: Feb. 1970.

/BELL73/ Bell, A. G., et al. A Universal Benchmark?
Software Practice and Experience, 3(4): Oct 1973.

/BELL74/ Bell, D.E., L.J. La Padula Secure Computer Systems: Mathematical Foundations and Models.
Conference of the Society for General Systems Research, Sacramento, 1974, (Mitre Corp. Report M74-244).

/BELL82/ Bell, D. A., S. M. Deen, Key Space Compression and Hashing in PRECI.
The Computer Journal 25(4): 486-492, 1982.

/BELL84/ Bell, D. A., S. M. Deen, Hash Trees Versus B-trees.
The Computer Journal, 27(3): 1984.

/BENCZU79/ Benczur, A and A. Kramli, An Example for an Adaptive Control Method Providing Data Base Integrity.
c IIASA, North-Holland Publishing Company, 1979.

/BENCZU80/ Benczur, A. A note on Dynamic Placement of Records in Linear Storage.
Szamki Tanulmanyok 5: 55-60, 1980. (in Hungarian).

/BENCZU83/ Benczur, A. Problems in Modelling of Data Base Performance.
Moscow, URSS, Perspectives in Developing Computer Technics, 85-98, 1983.

/BENCZU84/ Benczur, A and J. Stahl, On Updating a Large-scale Data-system.
Alkalmazott Matematikai Lapok, 10: 1-13, 1984.

/BENDE84/ Bender, M. Experience with the DELPHI Technique for DBMS Selection.

Proceedings Trends and Applications 1984: Making Data Base Work, 173-182, May 1984.

/BENWE75/ Benwell, N. Benchmarking.
Toronto, J. Wiley and Sons.1975.

/BERNS81/ Bernstein P.A., N. Goodman Concurrency Control in Distributed Database Systems.
Computing Surveys, 13(2): June 1981.

/BERNS83/ Bernstein, P. A. et al. Analysing Concurrency Control Algorithms when User and System Operations Differ.
IEEE Trans. on Soft. Eng. SE-9(3): 233-239, May 1983.

/BITTO83/ Bitton, H. et al. Benchmarking Database Systems: A Systematic Approach.
Computer Sciences Department, University of Wisconsin, Technical Report #526.

/BLACK81/ Black, J.P. et al. A Robust B-tree implementation.
Proceedings of the Fifth International Conference on Software Engineering, pp 63-70 (9-12 March 1981).

/BLASG79/ Blasgen, M., et al. System R: An Architectural Update.
San Jose, Calif. Rep Rj2581, IBM Res. Ctr. July 1979.

/BOAR84/ Boar, B. H. Ten Criteria For Selecting Mature DBMSs.
c Auerbach Publishers Inc, Data Base Management, 1984.

/BOGDA83/ Bogdanowicz, R. et al. Experiments in Benchmarking Relational Database Machines.

Munich, Proc. of the third International Workshop on Database Machines, Sept. 1983.

/BOIES74/ Boies, S.J. User Behavior on an Interactive System. IBM System Journal 13,1-18 1974.

/BOND84/ Bond, G. A Database Catalog. BYTE. Oct. 1984.

/BORAL84/ Boral, H. and D. DeWitt. A Methodology for Database System Performance Evaluation. Univ. of Wisconsin, Computer Sciences Department, Technical Report #532, January 1984.

/BOYLE84/ Boyle, B. Software Performance Evaluation. BYTE, 9(2): Feb. 1984.

/BRAGA01/ Bragado, M. E., Fonfria M., Muniz E., Data Entry, A Very Important Process. Hungary, MTA SZTAKI Kozlemenyek, 30: 53-59, 1984.

/BRAGA02/ Bragado M. E., Fonfria M. Indexing Treatement of a DBMS for Minicomputer. Budapest, MTA SZTAKI Kozlemenyek, 1987.

/CABRE/ Cabrera, L.F. Benchmarking UNIX: A Comparative Study. In Experimental Computer. Amsterdam, Performance Evaluation, eds. D.Ferrari and M. Spadoni.

/CARDE73/ Cardenas, A. Evaluation and Selection of File Organization: A Model and System. Comm. ACM, 16(9): 1973.

/CARRO/ Carrol. J.M., M.B. Rosson Usability Specifications as a Toll in Interactive Development.
Norwood. N., Advances in Human-Computer Interaction. H.R. Hartson.

/CARRO84/ Carrol. J.M., M.B. Rosson Beyond MIPS. Performance Is Not Quality.
BYTE. 9(2): Feb. 1984.

/CERI82/ Ceri S., S. Owicki. On the Use of Optimistic Methods for Concurrency Control in Distributed Databases.
Proc. 6th Berkeley Workshop on Distributed Data Management and Computer Networks, Feb. 1982.

/CHANG80/ Chang, C.C. The Study of an Ordered Minimal Perfect Hashing Scheme.
Comm. ACM. 23(1): 17-19, 1980.

/CHANG86/ Chang, C.C., R.C.T. Lee A Letter-Oriented Minimal Perfect Hashing Scheme.
The Computer Journal, 29(3): 277-281, 1986.

/CHRIS84/ Christian, K. Inside a Compiler: Notes on Optimization and Code Generation.
BYTE, 9(2): Feb. 1984.

/CID300/ Manual CID 300 /10
C. Habana, ICID., 1980.

/CNORT83/ c North-Holland Pub. Company. Human Factors Aspects of a Modern Data Base System.

Information and Management, 6(1): Feb. 1983.

/COBOL74/ American National Standard Programming Language COBOL.
New York, American National Standards Institute, C 1974.

/CODD70/ Codd, E.F. A relational model for Large Shared Data Banks.
Comm. ACM 13:377-387, 1970.

/CODD72/ Codd, E.F. Relational Completeness of Data Base Sublanguages.
N.J., Englewood Cliffs, Prentice-Hall, 1972.
(Courant Computer Science Symposia Series, Vol.6)

/COMER79/ Comer, D. The Ubiquitous B-tree.
Computing Surveys 11(2): 121-137, June 1979.

/CURNO76/ Curnow, H.J., B.A. Wichman A Synthetic Benchmark.
Computer Journal, 19(1): Feb. 1976.

/DAHL72/ Dahl, O.J., E.W. Dijkstra, C.A.R. Hoare Structured Programming.
London, APIC, Studies in Data Processing No.8, Academic Press, 1972.

/DATE/ Date C.J. An Introduction to database System.
La Habana, Dirección de Cálculo Electrónico, JUCEPLAN.

/DAVIE81/ Davies, D.J.M. Benchmarking in Selection of Timesharing Systems.
Proceedings of the 14th. meeting of the CPEUG, Nov. 1981.

/DAVIS84/ Davis, R.H., P. Coumpas A Dynamic File Organisation

Model.

The Computer Journal, 27(2): 143-150, 1984.

/DBASE83/ dBASE II. Assembly Language Relational Database Management System. User Manual.

c Ashton-Tate, 1983.

/DBASE86/ Manual de Usuario de dBASE-300.

C. Habana, ICID, 1986.

/DEARN78/ Dearnley, P. Monitoring Database System Performance.

The Computer Journal, 21(1): 1978.

/DEEN81/ Deen, S.M., et al. The Design of a Canonical Database (PRECI).

The Computer Journal 24(3): 200-209, 1981.

/DEMET/ Demetrovics, J. Mathematical Analysis of the Relational Data Model.

Proc. Third Hung. Computer Sci. Conf.

/DEMET78/ Demetrovics, J. On the Number of Candidate Keys.

Information Processing Letters, 7(6): 266-269, 1978.

/DEMET80/ Demetrovics, J. Candidate Keys and Antichains.

SIAM J. Algebraic Discrete Methods 1: 1980.

/DEMET82a/ Demetrovics, J. and Gy. Gyepesi. Logical Dependencies in Relational Data Base.

Hungary, MTA SZTAKI Tanulmanyok, 133: 59-78, 1982.

/DEMET82b/ Demetrovics, J. Logical and Structural Investigations of the Relational Data Model.

Hungary. MTA SZTAKI Kozlemenyek. 27: 23-35. 1982.

/DEMET82c/ Demetrovics. J.,E. Knuth. P. Rado. Specification Meta Systems.

IEEE. 29-35. 1982.

/DEMET83/ Demetrovics. J. and Gy. Gyepesi. Some Generalized Type Functional Dependencies Formalized as Equality Set on Matrices.

c North-Holland. Discrete Applied Mathematics 6: 35-47. 1983.

/DEMET84/ Demetrovics. J., et al Some Remarks on Statical Data Processing.

Hungary. MTA SZTAKI Kozlemenyek. 30: 37-51 1984.

/DEREM71/ De Remer. F.L. Simple LR(K) Grammars.

Comm. ACM. 14. 453-460. 1971.

/DONOV72/ Donovan. J.J. System Programming.

New York. McGraw-Hill Book. 1972.

/EPSTE80/ Epstein. R. and M. Stonebraker. Analysis of Distributed Data Base Processing Strategies.

Montreal. Proc. of the 6th VLDB. Canada. 1980.

/FEIER79/ Feiertag. R.J., P.G. Neumann The Foundations of a Provably Secure Operating System.

AFIPS Conf. Proceedings. NCC 79. p.329. 1979.

/FERRE78/ Ferreri D. Computers Systems Performance Evaluation.

Prentice-Hall. Inc., 1978.

/FOBOS/ Manual del Usuario del Sistema de Operacion FOBOS.
C. Habana, ICID.

/FONFR01/ Fonfria Atan, M. COBOL para la Computadora CID-201b
La Habana, ICID, 1974.

/FONFR02/ Fonfria Atan M. et al. COBOL 74 for Cuban Minicomputer.
To appear.

/FONFR03/ Fonfria Atan M. et al. Operating System for Commercial Applications (GES-300) for Cuban Minicomputer.
To appear.

/FONFR04/ Fonfria Atan M. et al. Software Base developed in Cuba for Data Processing on Cuban Minicomputer.
To appear.

/FONFR05/ Fonfria Atan M., Bragado M. E. About a methodology to Select a DBMS.
Budapest, MTA SZTAKI Kozlemenyek, 1987

/FONFR06/ Fonfria Atan M. et al. Data Base Management System dBASE-300 for cuban minicomputer CID 300/10.
To appear.

/FONFR07/ Fonfria Atan M. et al. An approach on Automated Syntax Analysis.
Budapest, MTA SZTAKI Kozlenemyek, 1987

/GARCI84/ Garcia, D. y N. Torres. Tres Sistemas de Gestion de Bases de Datos para Microcomputadoras. Estudio

Comparativo y Experiencias Practicas.
Encuentro de Especialistas. JUCEPLAN. 1984.

/GARDA80/ Gardarin, G. Integrity, Consistency, Concurrency, Reliability in Distributed Database Management Systems.
North-Holland Pub. Co., Distributed Data Bases, 1980.

/GHOSH77/ Ghosh, S.P. Data Base Organization for Data Management.
New York, Academic Press, 1977.

/GILBR81/ Gilbreath, J. A High-Level Language Benchmark.
BYTE, 180-198 Sept. 1981.

/GILBR83/ Gilbreath, J., G. Gilbreath Eratosthenes Revisited: Once More Through the Sieve.
BYTE. 283-326 Jan. 1983.

/GLESE81/ Gleser, M. et al. Benchmarking for the Best.
Datamation, May 1981.

/GOFF73/ Goff, N.S. The case for Benchmarking.
Computer and Automation, May 1973.

/GOLSH83/ Golshani, F., T.S. Maibaum, M.R. Sadler A Model System of Algebra for Database Specification and Query/Update Language Support.
Proceedings of the 9th Internatinal Conference on Very Large Data Bases, 1983.

/GRAPP81/ Grappel, R.D., J.E. Hemeway A Tale of Four Microprocessors: Benchmarks Quantify Performance.

EDN, 1981,179-185, April 1981.

/GRIES71/ Gries, D. Compiler Construction for Digital Computers.
New York, John Wiley, 1971.

/HANSE73/ Hansen, Per Brinch Operating System Principles.
N.J., Englewood Cliffs, Prentice-Hall, 1973.

/HAWTH79/ Hawthorn, P. and M. Stonebraker. Performance Analysis of a Relational Data Base Management System.
Boston, Proc. of the ACM SIGMOD Conf. 1979.

/HEHNE83/ Hehner, E.C.R., B.A. Silverberg Programming with Grammars:An Exercise in Methodology-Directed Language Design.
The Computer Journal, 26(3):277-281, 1983.

/HENNI68/ F. C. Hennie. Finite-State Models for Logical Machines
John Wiley and Sons, Inc., 1968.

/HIRSC73/ Hirschberg, Daniel, S. A Class of Dynamic Memory Allocation Algorithms.
Comm. ACM, 16(10): Oct. 1973.

/HOPCR67/ Hopcroft J. E. and J. D. Ullman. Formal Languages and their Relation to Automata.
Mass., Addison-Wesley, Reading, 1967.

/HOUST84/ Houston, J. Don't Bench me in.
BYTE, 9(2): Feb. 1984.

/HULTE77/ Hulten, C. and L. Soderlund. A simulation Model for

Performance Analysis of Large Shared Data Bases.
Proc. Third VLDB Conf., Tokio, 1977.

/JACOB84/ Jacobson, B. Database vs Condor and dBASE II.
BYTE. 289-302, Oct. 1984.

/KEENA81/ Keenan, M. A Comparative Performance Evaluation of Database Management Systems.
Berkeley, EECS Dept., University of California, 1981.

/KEREK84/ Kerekfy, P. et al. Patient Registers on Microcomputers.
c North-Holland, Cybernetics and Systems Research 2,1984.

/KERNI84/ Kernighan, B. W. and R. Pike. The UNIX Programming Environment.
New Jersey, Prentice-Hall, Inc., 1984.

/KNOWL65/ Knowlton, K.C. A Fast Storage Allocator.
Comm. ACM 8(10): 623-625, Oct. 1965.

/KNOWL66/ Knowlton, K.C. A Programmers Description of L6.
Comm ACM 9(8): 616-625, Aug 1966.

/KNUTH65/ Knuth, D. On the Translation of Language from Left to Right.
Information and Control, 8, 1965.

/KNUTH73/ Knuth, D. The Art of Computer Programming
Addison-Wesley, Reading, MA., Vol 1,3 1973.

/KOHAV78/ Kohavi, Zvi. Switching and Finite Automata Theory.
McGraw-Hill, Inc., 1978.

/KUNG79/ Kung, H. T. and C. H. Papadimitriou. An Optimality Theory of Concurrency Control for Databases. Proc. ACM-SIGMOD Int. Conf. Management of Data, 1979.

/LALON71/ Lalonde, W.R. An Efficient LALR Parser Generator. University of Toronto, Computer System Group, 1971.

/LAMPS73/ Lampson, B.W. A Note on the Confinement Problem. Comm. ACM, 16(10): Oct. 1973.

/LARSO70/ Larson, P.A. Dynamic Hashing. BIT 18: 184-201, 1970.

/LEWIS76/ Lewis, P. and G. Shedler. Statistical Analysis of Nonstationary Series of Events in a Data Base Systems. IBM Journal of Research and Development,20(5), 1976.

/LOPEZ1/ Lòpez, Jiménez, T. Resultados Principales del Desarrollo de los Medios Tècnicos de Computaciòn en Cuba. CID Electrònica y Proceso de Datos en Cuba, 1(1): 9-13.

/LUCAS71/ Lucas, H. Performance Evaluation and Monitoring. Computers Surveys, 3(3), Sept. 1971.

/MANNA74/ Manna, Z. Mathematical Theory of Computation. McGrow-Hill, 1974.

/MARTI77/ Martin, J. Organizaciòn de las Bases de Datos. Prentice-Hall International, 1977.

/MARVI84/ Marvit, P., M. Nair. Benchmark Confessions. BYTE, 9(2): Feb. 1984

/MATE85/ Mate, L. and M. Ruda. Microcomputers in Retirement

Service.

Hungary. MTA SZTAKI. Dec 1985.

/MATSO70/ Matson. R. et al. Evaluation Techniques for Storage Hierarchies.

IBM Syst. J. June 1970.

/MCILR63/ McIlroy, M.D. A Variant Method of File Searching.

Comm. ACM. Vol.6, Jan. 1963.

/MCKEE70/ McKeeman, W. M. et al. A Compiler Generator.

N.J., Prentice-Hall, Inc., Englewood Cliffs, 1970.

/MIYAM75/ Miyamoto, I. Hierarchical Performance Analysis Models for Database Systems.

Proceedings First VLDB Conf., Farmingham, 1975.

/MORRI68/ Morris, R. Scatter Storage Techniques.

Comm. ACM. 11(1): Jan. 1968.

/MUNIZ76/ Muñiz. E. et al. Informe Tècnico del Generador de Analizadores Sintàctico LALR(1) CID. 1976.

/MUNIZ82/ Muniz E., Fonfria M. COBOL 74.

C. Habana, Edit. Ciencia y Educacion, 1982.

/NAKAM75/ Nakamura, F., et al. A Simulation Model for Data Base System Performance Evaluation.

Proceedings NCC, 1975.

/PEDRO82/ Pedroso O. Introducciòn y Desarrollo de las Tècnicas de Computaciòn en Cuba.

CID Electrònica y Proceso de Datos en Cuba, 1(4): 2-8,

1982.

/PIROT82/ Pirote A. A Precise Definition of Basic Relational Notions and of the Relational Algebra.
ACM. SIGMOD Record, pp.30-45, 1982.

/PFLUG84/ Pflug, G.Ch. Dynamic Memory Allocation-A Markovian Analysis.
The Computer Journal, 27(4): 1984.

/POTIE80/ Potier, D. and P. Leblanc. Analysis of Locking Policies in Database Management Systems.
Comm. ACM, 23(10): Oct. 1980.

/PRICE71/ Price, C.E. Table Lookup Techniques.
Computing Surveys, 3(2): June 1971.

/PURDO70/ Purdom, P., S. Stigler Statistical Properties of the Buddy System.
J. ACM. 17(4): 683-697, Oct. 1970.

/RAMOS1/ Entrevista a O. Ramos.
CID Electrònica y Proceso de Datos en Cuba, 1(1): 8-9.

/REEVE80/ Reeves, C.M. Free Store Distribution Under Random Fit Allocation, Part 2.
The Computer Journal, 23 298-306, 1980.

/REEVE83/ Reeves, C.M. Free Store Distribution Under Random Fit Allocation, Part 3.
The Computer Journal, 26, 25-35, 1983.

/REISN81/ Reisner, P. Human Factors Studies of Database Query

Languages: a Survey and Assesment.
ACM Computing Surveys, 13(1): 13-32, 1981.

/RMS-11/ RMS-11 User's Guide.
Digital Equipment Corp.

/ROBER84/ Roberts, B. Benchmarks and Performance Evaluation.
BYTE, 9(2): Feb. 1984.

/RODRI75/ Rodriguez-Rosell, J. and D. Hilderbrand. A Framework for Evaluation of Data Base Systems.
Research Report RJ 1587, IBM, San Jose, 1975.

/ROSEN78/ Rosenberg, A.L., L. Snyder Minimal-Comparison 2,3-Trees.
SIAM J. Computer, 7(4): Nov. 1978.

/SCHRO72/ Schroeder, M.D., J.H. Saltzer A Hardware Architecture for Implementing Protection Rings.
Comm. ACM, 15(3): 157-170, Mar. 1972.

/SENSB/ O'Hanlon Computer System. The SENSIBLE SOLUTION Language Reference Manual.

/SHAW74/ Shaw, Alan The Logic Design of Operating System.
Englewood Cliffs, Prentice-Hall, N.J. 1974.

/SHORE77/ Shore, J.E. Anomalous Behavior of the Fifty Percent Rule in Dynamic Memory Allocation.
CACM, 20, pp.812-820, 1977.

/SIBLE84/ Sibley, E. H. DBMS Evaluation and Selection.
c Auerbach Publishers Inc, Data Base Management,22-04-01.

/SMITH80/ Smith, C. and J. Browne. Aspects of Software Design Analysis: Concurrency and Blocking.
Proc. of the Performance 80 Symposium, Toronto, 1980.

/SPOON/ Spooner, C.R. Benchmarking Interactive Systems: Modeling the Application.
Proceedings of the 15th. Meeting of the Computer Performance Evaluation Users Group (CPEUG), pp.53-63.

/STONE80/ Stonebraker, M. Retrospection on a Database System.
ACM Trans. Database System, 5(2): June 1980.

/STONE81/ Stonebraker, M. Operating System Support for Database Management.
Comm. ACM, 24(7): July 1981.

/STONE83/ Stonebraker, M. et al. Performance Enhancements to a Relational Database System.
ACM Transactions on Database Systems, 8(2): June 1983.

/SU81a/ Su, S. et al. A DMS Cost/Benefit Decision Model: Cost and Preference Parameters.
National Bureau of Standards,Report NBS-GCR-82-373,1981.

/SU81b/ Su, S. et al. A DMS Cost/Benefit Decision Model: Analysis, Comparison, and Selection of DBMSs.
National Bureau of Standards,Report NBS-GCR-82-375,1981.

/TAYLO86/ Taylor, D.J., J.P. Black A Locally Correctable B-tree Implementation.
The Computer Journal, 29(3): 1986.

/TEMPL/ Templeton, M. et al. Evaluation of 10 Data Management Systems.
SDC document TM-7817/000/00.

/TEORE76/ Teorey, T. and K. Das. Application of an Analytical Model to Evaluate Storage Structures.
Proc. of ACM SIGMOD Conf., 1976.

/THOMA79/ Thomas, R. H. A Majority Concensus Approach to Concurrency Control for Multiple Copy Databases.
ACM Trans. Database Systems 4(2): 1979.

/ULLMA82/ J. Ullman. Principles of Database Systems.
Computer Science Press, Second Edition, 1982.

/UNIX/ UNIX Programmer's Manual.
New Jersey, Bell Telephone Lab., Inc. Murray-Hill, 1979.

/VERHO78/ Verhofstad, J.S.M. Recovery Techniques for Database Systems.
Computing Surveys, 10(2): June 1978.

/WAGNE73/ Wagner, R. Indexing Design Considerations.
IBM Syst. J. 4: 351-367, 1973.

/WALTE76/ Walters, R.E. Benchmark Techniques: A Constructive Approach.
The Computer Journal, 19(1): Feb. 1976.

/WEISS81a/ Weiss, H. Down-scaling DBMS to the Microworld.
Mini-Micro Systems, April 1981.

/WEISS81b/ Weiss, H. Which DBMS is Right for You.

Mini-Micro Systems, October 1981.

/WILKE84/ Wilkes. M.V. Security Management and Protection-A Personal Approach.

The Computer Journal, 27(1): 1984.

/YAO75/ Yao,S. and A. Merten. Selection of File Organizations through Analytic Modeling.

Proc. First VLDB Conf., Framingham, 1975.

/YAO77a/ Yao, S. An Attribute Based Model for Database Access Cost Analysis.

ACM Transactions on Database Systems, 2(1): 1977.

/YAO77b/ Yao, S. Approximating Block Accesses in Database Organizations.

Comm. ACM, 20(4): 1977.

/YAO78/ Yao S. and D. DeJong. Evaluation of Database Access Paths.

Proc. of ACM SIGMOD Conf., 1978.

/YAO79/ Yao, S. Optimization of Query Evaluation Algorithms.

ACM Transactions on Database Systems, 4(2): 1979.

/YOUSE79/ Youseffi, K., and E. Wong. Query Processing in a Relational Database System.

Proc. Fifth VLDB, 1979.

/YAO84/ Yao, S. B., et al. Architectural Comparisons of Three Database Systems.

Report submitted to the National Bureau of Standards,1984

1986-BAN MEGJELENTEK:

179/1986 Terlaky Tamás: Egy véges criss-cross módszer és alkalmazásai

180/1986 K.N. Čimev: Separable sets of arguments of functions

181/1986 Renner Gábor: Kör approximációja a számitógépes geometriai tervezésben

182/1986 Proceedings of the Joint Bulgarian-Hungarian Workshop on "Mathematical Cybernetics and Data Processing" Scientific Station of Sofia University, Giulecica /Bulgaria/, May 6-10, 1985 /Editors: J. Denev, B. Uhrin/ Vol I

183/1986 Proceedings of the Joint Bulgarian-Hungarian Workshop on "Mathematical Cybernetics and Data Processing" Scientific Station of Sofia University, Giulečica /Bulgaria/, May 6-10, 1985 /Editors: J. Denev, B. Uhrin/ Vol II

184/1986 HO THUAN: Contribution to the theory of relational databases

185/1986 Proceedings of the 4th International Meeting of Young Computer Scientists IMICS'86 /Smolenice, 1986/ /Editors: J. Demetrovics, J. Kelemen/

186/1986 PUBLIKÁCIÓK - PUBLICATIONS 1985 Szerkesztette: Petróczy Judit

187/1986 Proceedings of the Winter School on Conceptual modelling /Visegrád, 27-30 January, 1986/ /Editors: E. Knuth, A. Márkus/

188/1986 Lengyel Tamás: A Cluster analizis néhány kombinatorikai és valószinüségszámitási problémája

189/1986 Bernus Péter: Gyártórendszerek funkcionális analizise és szintézise

190/1986 Hernádi Ágnes: A tipus fogalma, és szerepe a modellezésben

191/1986 VU DUC THI: Funkcionális függőséggel kapcsolatos néhány kombinatorikai jellegü vizsgálat a relációs adatmodellben

192/1986 Márkusz Zsuzsanna: P a p e r s on Many-stored logic as a tool for modelling

193/1986 KNVVT Conference on Automation of Information Processing on Personal Computers
Budapest, May 5-9, 1986 Vol I.
Szerkesztette: Ratkó István

194/1986 KNVVT Conference on Automation of Information Processing on Personal Computers
Budapest, May 5-9, 1986 Vol II.
Szerkesztette: Ratkó István

A TANULMÁNYOK SOROZATBAN 1987-BEN EDDIG MEGJELENTEK:

195/1987 Telegdi László: Bináris változók strukturájának vizsgálata

196/1987 Rónyai Lajos: Algebrai algoritmusok

197/1987 Hernádi Ágnes - Bodó Zoltán - Knuth Előd: A tudásábrázolás technikái és gépi eszközei