

Design and Implementation of a Relational DBMS for Microcomputers

F. CESARINI* AND G. SODA

Dipartimento di Sistemi e Informatica v. S. Marta 3, Florence, Italy

The aim of this paper is to present the design and implementation of Mimirel, a relational Data Base Management System for microcomputers especially directed to non-specialist end users. The system overcomes architectural limits of microcomputers by means of a physical data organisation which allows space saving and fast inquiring. Furthermore, the user interface has been developed so as to take the needs of a typical microcomputer end user into account. The critical operations are system driven in a conversational way and suitable facilities are provided for the inquiring. The Mimirel system has been fully implemented in Pascal, under the CP/M operating system, and so it is highly portable.

1. INTRODUCTION

During the last few years, the personal computer has met with wider and wider applications in different environments; in particular, it has been used extensively for storing a large quantity of data. The personal computers have standard configurations (such as 8–16-bit CPU, 64–128 kb and more for main memory, floppy disks or hard disks) and are equipped with various software tools. They range from file systems to real Data Base Management Systems (DBMS). The systems of the former are obviously more numerous than those of the latter because real DBMSs have been defined only recently.

The file system is a useful tool and it is known that it represents an organisation lower than the DBMS, if we consider user utilisation. The DBMSs are usually applied to very large systems because of their characteristics, while only recently DBMS implementation on microcomputers has been faced.

A brief survey of existing DBMS for small machines¹ examines their only partial fulfilment of portability, interactivity and simplicity requirements. In our opinion other commercially available systems, such as MDBS² and DBASE II,³ do not present high flexibility and simplicity for non-expert end users even if they possess many or all the features of a DBMS. Two major points should be considered. The design of the DBMS architecture must be such as to exploit all possibilities offered by the machine. This is an important goal, since in a microcomputer environment resources are limited with respect to larger systems. On the other hand, it is our opinion that a DBMS used on microcomputers must present a simple and flexible user interface, since in this case the user is typically a non-specialist one. Furthermore, the user combines the two major functions traditionally present in a large system, the Data Administrator/Data Base Administrator (DA/DBA) and the end user himself.

In this paper we are presenting a portable DBMS, called Mimirel (MIni and Micro RELational system), especially designed for microcomputers and directly manageable by a non-expert end user. Therefore particular care has been devoted to the database definition and to inquiring, which are the most important functions in a microcomputer environment, and very simple interfaces are provided for them.

* Address for correspondence: F. Cesarini, Dipartimento di Sistemi e Informatica, v. S. Marta 3, 50139 Florence, Italy.

The Mimirel is based on the relational model of data,^{4,5} which is more appropriate for non-specialists than other models are (such as hierarchical or network ones). This is because the tabular representation of data is conceptually simple and natural. Furthermore it is possible to specify logical functions, even if highly complex, only by referring to this tabular view of the data, without knowing their physical organisation.

The unique aspects of Mimirel are the following.

(a) Data structures, called Data Pools,⁶ which optimise the use of storage devices and allow a good performance in query processing.

(b) The interactivity of the schema definition.

(c) The driven input of the data.

(d) A conversational Sequel – like query language.⁷

Mimirel is completely implemented in Pascal and runs under the CP/M operating system, and so it is highly portable.

In this paper we shall describe the data structures used for storing relations. Their definition is of particular importance because they can implement a fast selective inquiry. We shall then describe the user interface, referring to the main moments of a database application: scheme definition, data insertion and updating, and inquiring. The description points out the design and implementation choices made in order to meet the needs of non-expert users, and our whole data management is supported by driven and conversational tools. At the same time, users with some experience are given the chance to control and personalise the Mimirel system. The last section deals with both the limits imposed by the machine environment and the solutions adopted in order to obtain flexibility and portability.

2. PHYSICAL DATA ORGANISATION

2.1 Internal schema

The internal schema of the data is based on the domains on which the relations are defined. Each domain is associated with a specific physical structure, called Data Pool. A Data Pool collects all the distinct values of the corresponding domain appearing in tuples of the data base, whichever relation they may belong to.

Let us consider, for example, the following relations:

PERSON (NAME, PER_ADDR, JOB, SALARY)
HOUSE (HOUS_ADDR, DISTRICT, OWNER,
RENT, SIZE)

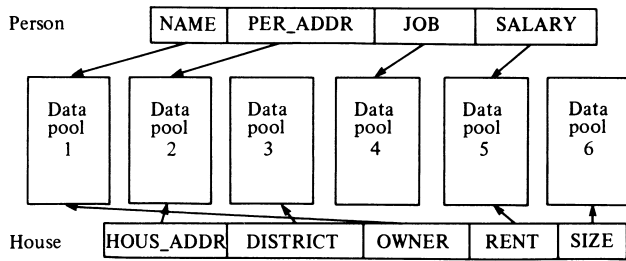


Figure 1. Data Pool schema.

The PERSON relation gives the name, address, job and salary of a certain set of people. The HOUSE relation describes a certain set of houses by means of their address and district, the owner's name, its monthly rent and its size.

The attributes NAME and OWNER are defined on the same domain; the corresponding Data Pool contains all the names appearing in the two relations, and each name is stored only once. The same goes for all the other attributes. The relationships between relations, attributes and Data Pools are represented by Fig. 1.

2.2 Physical data structure

Since Data Pool structures involve the disaggregation of the relations, some control information must be maintained in order to reconstruct the relational view of the data, i.e. to re-aggregate the tuples belonging to each relation.

The values allocated to Data Pools and control information are represented in four (DICTIONARY, INDEX, DATA, RELAT) or two (DATA, RELAT) files, according to whether the domain is inverted or not. Their relationships are illustrated in Fig. 2.

DATA is the file containing the values of the attributes related to the Data Pool, whichever tuple of whichever relation they may belong to. One DATA file for each domain is defined.

RELAT is the control information file to be used to

reconstruct the tuples of a certain relation. Each record in a RELAT file corresponds to a tuple of the relation and is a set of pointers. Each of them points to a value to be used to reconstruct the tuple. The Tuple Identifier (TID) of a tuple is a reference to the corresponding record in the appropriate RELAT file. One RELAT file for each relation is defined.

In case a domain is inverted, the DICTIONARY and INDEX files are defined further on.

The DICTIONARY file contains the access keys structured as a B*-tree. Each access key is constituted by a value (replicated in the DATA file) followed by a set of pointers, one for each relation sharing that domain. Each pointer references a list of TIDs in the INDEX file.

The INDEX file contains lists of TIDs. Each list is constituted by the TIDs of the tuples of the same relation containing the same value in the same column. One DICTIONARY file and one INDEX file are defined for each inverted domain.

3. SCHEMA DEFINITION

The scheme entering follows the logical structure of the data base which can be represented in pictorial form like Fig. 3.

The information is given in a bottom-up way starting from every domain, and completing it with the attributes which constitute the domains and the relations' structure.

Particular care has been given to user interfacing in order to give the user a flexible tool for his applications. The user performs the database schema definition in a conversational way. It consists of two phases: (1) schema declaration, (2) schema initialisation. These are summarised in Fig. 4.

In phase 1, the user supplies suitable information about his own database organisation and data dimensioning; the system then builds the directories containing information about relations, domains and attributes.

Each piece of information which has to be entered in phase 1 is explicitly asked for. This implies a kind of

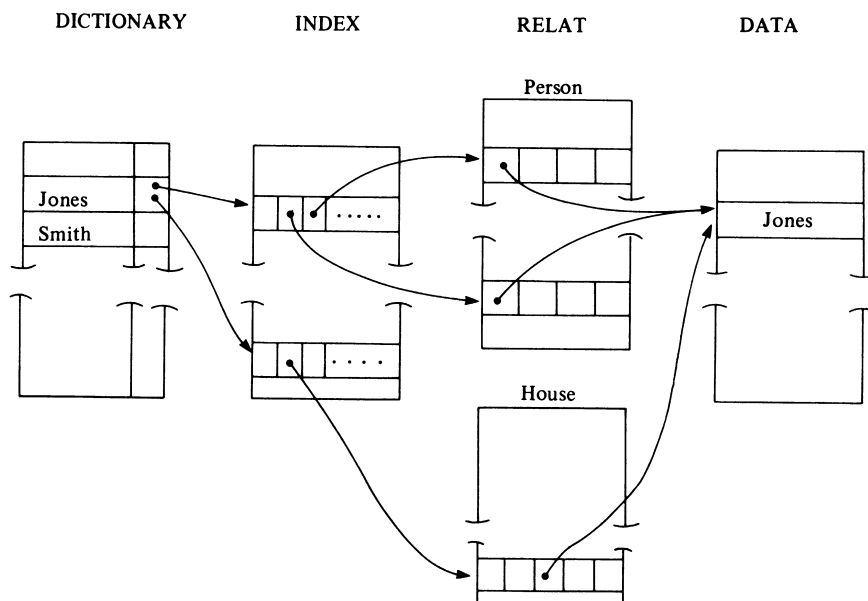


Figure 2. File relationships in case of inverted domain.

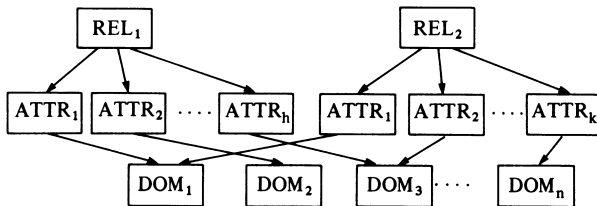


Figure 3. Pictorial form of schema.

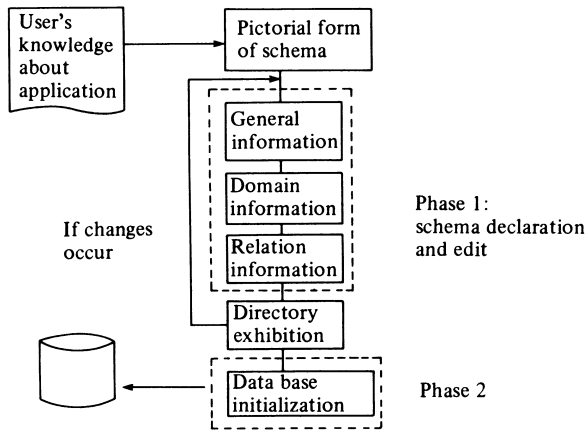


Figure 4. Database schema definition.

rigidity in the database entry flow, but on the other hand this mode of operating offers greater simplicity of use. In particular, the user can detect all mistakes inherent to the consistency of the input with the data previously entered in the schema. For example, at the time of relations definition, all the attributes belonging to that relation must have already been defined.

The system asks for three types of information.

(a) General information about the system dimensioning. This information concerns the space allocation of the four basic files: RELAT, DATA, DICTIONARY, INDEX. The system asks for maximum space occupation for each of them, and the user can answer with suitable values computed by means of considerations on the space occupied by every record of the files with respect to the user's application. This piece of information is optional because the system can also consider default values. If it does not, the user can exploit this feature in order to optimise space allocation to perform typical DBA/DA tasks.

(b) This information is concerned with Data Pool definition. The system asks for the number of domains and, for every domain, it asks for its name and type (inverted or not), the names of the attributes defined on the domain, the maximum number of distinct values of the domain, the type (numeric or alphanumeric) and length of its values.

(c) This information is concerned with proper relational schema. By its means the user makes a logical re-aggregation of the domains. The system asks for the number and name of each relation, the names of its attributes, the primary key and the cardinality.

The above values are used by the system to obtain space allocation. If in phase 1(a) the user gives proper information instead of using default values, the system displays the filling percentage of the disk space estimated

in phase 1(a) with respect to the values entered in phase 1(b) and 1(c). In such a way the user who wants to control the physical data storing by himself has a tool for detecting space occupation of the various files at his disposal.

The system also displays the directories in order to check the entered data schema organisation. If necessary, the user can change some or all parameters.

In the second phase of the schema definition, the system performs the pre-allocation of the storage available for the physical files. The increase of the database will agree with the dimensioning of these files.

4. DATA MANIPULATION

Once the initialisation is completed, the user can enter his data. For every relation, the system prompts the relation name and asks for input modality.

The entering can be made in two modes: a conversational mode and a batch mode.

The input in conversational mode is completely driven by the system. In such a mode, the system prompts the layout of each tuple's attribute, and displays the attributes' name, type and length. The inputs are checked, possible error messages are output and the tuple is accepted only when all the entered values agree with the schema definition.

The batch mode is concerned with the input of existent files. Since the system accepts the tuples of every relation one at a time, it is possible to see an input file as a 'flat' storing of a relation. In this case, the system associates the input file name to the relation name and reads each record, considering it as a tuple. Records not agreeing with the schema are rejected.

Since the Mimirel system is strongly oriented to the inquiring, both deletion and updating are implemented in a simple way. The tuple deletion is performed by specifying its primary key and the updating consists in deleting and inserting a whole tuple.

5. QUERY LANGUAGE

The query language is Sequel-like and allows the end user to put his requests into a form closely resembling a natural formulation.

The implementation on a personal computer of a complete, or nearly complete, Sequel query language is an almost unrealistic goal because of the strict hardware restrictions. On the other hand, the user operating in a personal computer environment usually does not require such implementation.

Because of this, a subset of Sequel has been chosen which can be implemented in a limited hardware environment and offer the user a flexible and powerful enough tool for inquiry. The main characteristics of the chosen subset are a one-level-nested mapping and the possibility of using arithmetical expressions and aggregation functions. The definition of the language is reported in Appendix B.

5.1 Simple queries

This type of query corresponds to the basic mapping operation of the Sequel language. The use of simple arithmetical expressions and aggregation functions

(% AVG, % MAX, % MIN, % SUM, % COUNT) is allowed in the Select clause. For example, the following query asks for the addresses of houses bigger than 120 which are situated in the fourth district, and asks for their minimum and maximum annual rent price.

```
SELECT HOUS_ADDR, 12 × % MIN(RENT),
      12 × % MAX(RENT)
FROM HOUSE
WHERE SIZE > 120 .AND. DISTRICT = 4
```

The absence of arguments in the SELECT or WHERE clause implies all attributes or all tuples respectively.

5.2. Compound queries

This kind of query selects tuples of one relation according to values belonging to another relation. The form of the query is nested mapping. In particular, the inner mapping returns a value, or a set of values, which is used in evaluating the WHERE clause of the outer mapping.

To give an example, let us suppose we want to know the jobs (and their number) of people whose salary is higher than the average rent price of the houses situated in the fourth district.

```
SELECT JOB, % COUNT
FROM PERSON
WHERE SALARY >
      SELECT % AVG(RENT)
      FROM HOUSE
      WHERE DISTRICT = 4
```

5.3. Join queries

This kind of query basically corresponds to the equijoin relational operator. A condition to be verified by the tuples belonging to the second relation can also be specified. The form of the query is nested mapping where the inner mapping may specify the additional condition in the WHERE clause.

For example, the following query asks for the name and address of the owners of houses with a rent price lower or equal to 300,000 and which are larger than or equal to 100; the addresses of the houses are also requested.

```
SELECT NAME, PER_ADDR/HOUSE_ADDR
FROM PERSON, HOUSE
WHERE NAME =
      SELECT OWNER
      FROM HOUSE
      WHERE RENT ≤ 300,000 .AND. SIZE ≥ 100
```

5.4 Query language implementation

The types of query chosen for the interface language can be answered by exploiting the advantages offered by the dictionary files, without any particular optimisation of resolution paths. For this reason, an interactive implementation by means of an interpreter has been preferred to a compilation.

The conversational approach offers the user an interface with two main characteristics: (1) driven syntax, (2) immediate error signals and correction facility without re-entering the whole query.

As far as the first point is concerned, the system itself outputs the query keywords and takes care of the linking of the mappings. For example, if it realises that a query is a join query (from the expression entered in the

SELECT clause), the FROM clause of the inner mapping is completely output by the system because it must repeat the second relation name appearing in the outer mapping. In this way the user only enters indispensable data on a sort of form displayed by the system dynamically, on the basis of the data previously entered. The query input is then made easier and the possibility of error reduced.

As far as the output printing is concerned, the system asks the user which device has to be used: video console, printer or disk file. Suitable formats are provided for each of them. The files can be processed by applicative programs coded in whatever language is desired. Hence this option allows the user to personalise the system in order to obtain output tailored to his applications or to perform further elaborations on information extracted from database.

6. PERFORMANCE CONSIDERATION

The Data Pool organisation implies data dis-aggregation in the sense that the relations are not stored in any flat form. The tuples rebuilding is then a tedious task for the Mimirel system and constitutes an intrinsic limit to its use. For example, the display of a whole relation is one of the most expensive operations possible.

On the other hand, the Data Pool schema offers remarkable advantages, such as the following.

(a) A good use of space memory, because each value of an attribute (or more attributes which belong to the same domain) is stored in a single copy. In this way, the more the values are repeated in the tuples, the more memory we save.

(b) The implementation of a Data Pool as an inverted file allows for a very fast selection of the tuples which satisfy particular conditions. Moreover the B*-tree implementation for the DICTIONARY files permits fast retrieval not only in case of = conditions, but also for <, ≤, >, ≥, ≠ conditions (the tree structure maintains the keys sorted).

(c) A fast execution of the Join operation, which makes it possible to formulate complex queries but which is usually the most time-consuming relational operator. In our case, the Data Pool schema implements the Join operation directly. The Join is simply realised by scanning the dictionary associated to the domain on which the two attributes of the joined relations are defined. For each value belonging to both relations, the corresponding lists of TIDs are collected in order to obtain the cartesian product resulting from the Join operation.

Let us now consider the cost of query execution in some detail. Since a wide range of different features can be used in formulating queries (various comparison operators, arithmetic expressions, aggregation functions), we only consider the performance of the basic steps involved in query execution and we refer to some sample queries which operate selections and joins followed by projections.

The performance index we use for quantifying the query cost is the number of page accesses necessary for retrieving data. Two basic steps are performed when executing a query. The first one consists in identifying the tuples which satisfy the conditions expressed in the query; the second one consists in retrieving the values to be printed. The first step is accomplished by accessing the DICTIONARY and INDEX files and it has an almost

constant cost. The second step is performed by accessing the RELAT and DATA files and it is strongly dependent on the number of selected tuples (query selectivity) and on the number of projected attributes. In appendix A we report the formulas used in calculating the cost of our sample queries, which are selections which use '=' conditions (example of simple queries), semijoins which use '=' condition (example of compound queries), and equijoins (example of join queries).

In order to assess the query-processing capability of our system, we compare its performance to DBASE II,³ which is very widely used on microcomputers like ours. The sample queries mentioned can be requested in DBASE II by a single command. They are a reasonable basis for a comparison based on our analytic model because operations requested by means of a sequence of commands involve overhead due to files management between the execution of single commands.

DBASE II stores relations as flat files and is able to build B-tree indexes on selected attributes. The index is used to execute selections which define a single '=' condition. In all other cases, selections are executed by scanning the relation. The join command is executed by the nested-loop algorithm. The formulas used in calculating the cost of these operations are given in appendix A.

We use a sample data base for evaluating query costs. The size of the database and selectivity factors of the queries we use can be found in other case studies.^{8,9} We consider relations of 1000 tuples, where each tuple has 8 attributes of 16 bytes. We assume that the values of secondary keys are uniformly distributed and each value is duplicated three times, on the average. As far as the physical data layout is concerned, we assume that there are pages of 512 bytes (according to the CP/M operating system), an average of 2 page accesses for retrieving a specified value in an index, an average of 16 keys in the leaves of a B*-tree in Mimirel, and 2 bytes for TIDs.

Printing the whole relation requires 250 page accesses for DBASE II and about 8000 for Mimirel; therefore, that operation can be considered expensive in our system. The cost of executing real queries is illustrated in Tables 1 and 2. As we can see, DBASE II is fast when the selections have a single '=' condition, while in the other cases scanning the file involves a larger number of page accesses. The cost of join commands increases dramatically and makes the operation very time-consuming. On the contrary, Mimirel is also able to execute complex queries (that is, predicates with several conditions and joins) in a satisfactory way. Moreover, it gives its best results with very selective queries. It may be noted that our cost index only refers to approximate execution time for two main

Table 2. Number of page accesses in equijoins on 1000-tuple relations. The two values attributed to Mimirel correspond to retrieving 1 or 8 attributes for semijoin and 2 or 15 attributes for join

Number of output tuples	Mimirel		DBASE II	
	Semijoin	Join	Semijoin	Join
10	44-114	52-182	62 500	62 500
100	254-954	328-1628	62 500	62 500

reasons. The first is that it does not consider the overhead for file managing, which is especially expensive for DBASE II because it is not able to maintain more than two files open at the same time. The second reason is that special features for reducing I/O times, such as buffer management techniques, which reduce the number of physical I/O operations, are not considered. At any rate, our cost index is meaningful for preliminary comparison purposes.

7. SYSTEM IMPLEMENTATION AND PORTABILITY

In order to give the Mimirel system a high degree of flexibility, both the design and implementation are studied in such a way as to maintain its characteristics of modularity and portability.

Portability is obtained by means of the programming language used for implementation. Mimirel is fully implemented in a Pascal version running under CP/M operating system supporting random access files. Pascal language is chosen both because it offers all the advantages of modern structured and modular programming languages and because it is widely used on microcomputers. Two main characteristics have been maintained in designing our programs:

(a) file organisation partially independent of operating system;

(b) an interface file which makes system reconfiguration easier.

As far as file implementation is concerned, it is useful to have flexible management of the variable dimensioning of the files, while operating systems running on personal computers, such as CP/M, usually do not allow this. In the strategy chosen for the implementation we used self-contained management of the disk space instead of assigning it to the operating system. A single file is defined for storing homogeneous structures. So only four 'physical' files are defined for storing all the 'logical' RELAT, INDEX, DATA and DICTIONARY files. Inside each physical file, the space allocation for the logical files is managed directly by the Mimirel system in a dynamic way. In the main memory we have four I/O windows associated with the four file types and four buffers, managed by a modified LRU technique, used for maintaining pages of B*-trees in order to speed up their access. Therefore 10K bytes altogether are used for the various I/O buffers.

As far as step (b) is concerned, in the present version of the Mimirel system we have introduced an interface file which contains the definition of global constants and variables used in all programs. These are of two types.

Table 1. Number of page accesses in selections on a 1000-tuple relation by predicates with m '=' conditions. The two values attributed to Mimirel correspond to retrieving 1 or 8 attributes per tuple

Number of output tuples	Mimirel		DBASE II	
	m = 1	m = 2	m = 1	m = 2
1	4-11		3	
3	9-30	12-33	5	250
10		26-96		250

The first one concerns the maximum values allowed for the database parameters, such as number of relations, domains, attributes, number of attributes definable on the same domain, number of keys contained in a B*-tree page, and so on. The second one concerns screen parameters which consider cursor addressing, width and height, character set and so on.

The interface file is actually a portion of a Pascal program containing the declarations of the above-mentioned global parameters. It can be edited with the standard editor of the host operating system and appended to the programs to be compiled.

In brief, we can say that the feature of step (a) allows portability with few changes in the programs. Since random access files are not standard in Pascal, the used primitives are strictly related to the used Pascal version (AMC Pascal), and they may need to be changed in case of different versions. The feature of step (b) allows us to adapt the system to a particular environment by recompiling the programs appended to the interface file.

8. CONCLUSIONS

We have presented the design and implementation of a DBMS for microcomputers. The inverted Data Pool schema, used in the physical data organisation, allows for space saving and fast data search. Particular care has been devoted to user interface. A conversational approach has been used to implement the schema definition and driven data input. The Sequel-like query language allows for inquiry of reasonable complexity and also offers the possibility of obtaining calculated data by means of aggregation built-in functions. All these features make the system a simple and flexible tool, oriented to non-specialist end users.

REFERENCES

1. G. Falquet, D. Petitpierre, N. Magnenat-Thalmann and D. Thalmann, A portable relational Data Base Management System for microcomputer, *Microprocessing and Microprogramming*, **9**, 17-25 (1982).
2. Micro Data Base System Inc. *MDBS DDL-DMS-QRS Reference Manual*, Lafayette, Indiana (1981).
3. Ashton-Tate Inc., *DBASE II Reference Manual*, Los Angeles, CA, (1981).
4. J. D. Ullman, *Principles of Database Systems*, Computer Science Press, Maryland (1980).
5. E. F. Codd, 'A relational model of data for large shared data banks', *CACM*, **13**, 6 (1970).
6. F. Cesarini and F. Pippolini, *Analysis of an Inverted Data Pool Organization*, Tech. Rep. DATANET/DBMAC, Rome (1981).
7. D. D. Chamberlin *et al.*, 'SEQUEL 2: a unified approach to data definition, manipulation and control', *IBM J. Res. Develop.*, **20**, 6 (1976).
8. J. L. Abbott, 'A comparison of five database management programs', *Byte* (May 1983).
9. D. J. De Witt and P. B. Hawthorn, 'A performance evaluation of database machine architectures', *Proceedings of the 7th International Conference on Very Large Data Bases, Cannes, 1981*.

APPENDIX A

Cost of query execution

Let r be the number of tuples which satisfy the query;
 n be the number of attributes to be projected;
 d be the number of page accesses to the dictionary for retrieving a selected key;
 s be the number of page accesses necessary for scanning the dictionary leaves;
 f be the number of pages occupied by a relation stored as a flat file;
 l be the average length of a tid list;

An attempt has also been made to obtain some compromise between the needs of totally non-expert end users and users with some knowledge of their system. We have indicated three moments in which the end user could usefully personalise his data management: in disk space dimensioning, restructuring of existent files, and elaborating in a special way data selected from the data base. Suitable features are provided for these activities.

The Mimirel system is strongly oriented towards selective inquiry, and this is also a limit to its usefulness. Better performance is obtained when non-repetitive queries with different degrees of complexity are formulated. On the other hand, our experience has been limited to a microcomputer environment, with space and time constraints. Therefore a flexible, high-level interface involves a limitation on the user profile especially addressed by the system.

Mimirel is presently implemented on the AMD SYS 8/8 microcomputer equipped with two floppy disks and 64K bytes of main memory. It is completely implemented in AMC Pascal supporting random access files and runs under the CP/M operating system. Due to its design and implementation, the system has a high degree of portability.

As far as the main memory layout of the present implementation is concerned, about 9K bytes are occupied by the CP/M operating system, 31K by the Pascal interpreter and only 24K are utilisable by the Mimirel system. Due to these constraints, the programs run in overlay.

Acknowledgements

We thank the referee for his suggestions as to how to improve the final version of our paper.

b be the byte size of a tid;
 p be the byte size of a page.

The formulas we have attributed to the operations are made up of three terms. The first one counts the DICTIONARY file accesses, the second one counts the INDEX file accesses and the third one counts the RELAT and DATA file accesses. As far as the primary keys are concerned the second term is missing because the TID is stored in the dictionary instead of the pointer referring to the TID list.

Mimirel

Selection (Simple query) with '=' conditions on m attributes

Cost = $m \times d + m \times \lceil b \times t/p \rceil + r(l+n)$, where $\lceil b \times t/p \rceil$ is the number of pages occupied by a single TID list.

Semijoin (Compound query) with '=' condition

Cost = $s + \lceil r/t \rceil \times \lceil b \times t/p \rceil + r(l+n)$, where $\lceil r/t \rceil$ is the number of TID lists to be fetched.

Equijoin (join query)

Cost = $s + 2 \times \lceil \sqrt{r/t} \rceil \times \lceil b \times t/p \rceil + r(l+n)$. The second term is based on the following assumptions. Since r is the

number of tuples constituting the result, let each relation contribute with \sqrt{r} tuples to the result. The number of TID lists to be retrieved for each relation is $\lceil \sqrt{r/t} \rceil$.

DBASE II

Selections with one '=' condition

Cost = $d + r$

Selections of another kind

Cost = f

Join

Cost = $f_1 \times f_2$

APPENDIX B**Query language syntax**

$\langle \text{QUERY} \rangle ::=$
 $\langle \text{SIMPLE_QUERY} \rangle \mid \langle \text{COMP_QUERY} \rangle \mid \langle \text{JOIN_QUERY} \rangle$
 $\langle \text{SIMPLE_QUERY} \rangle ::=$
 $\text{SELECT } \langle A1 \rangle \text{ FROM } \langle \text{REL_NAME} \rangle \text{ WHERE } \langle C1 \rangle$
 $\langle \text{COMP_QUERY} \rangle ::=$
 $\text{SELECT } \langle A1 \rangle \text{ FROM } \langle \text{REL_NAME} \rangle \text{ WHERE } \langle \text{RED_COND} \rangle$
 $\text{SELECT } \langle \text{EXP} \rangle \text{ FROM } \langle \text{REL_NAME} \rangle \text{ WHERE } \langle C1 \rangle$
 $\langle \text{JOIN_QUERY} \rangle ::=$
 $\text{SELECT } \langle A2 \rangle / \langle A2 \rangle \text{ FROM } \langle \text{REL_NAME} \rangle, \langle \text{REL_NAME} \rangle \text{ WHERE } \langle \text{ATTR} \rangle =$
 $\text{SELECT } \langle \text{ATTR} \rangle \text{ FROM } \langle \text{REL_NAME} \rangle \text{ WHERE } \langle C1 \rangle$
 $\langle A1 \rangle ::= \text{empty} \mid \langle A11 \rangle$
 $\langle A11 \rangle ::= \langle \text{EXP} \rangle \mid \langle A11 \rangle, \langle \text{EXP} \rangle$
 $\langle C1 \rangle ::= \text{empty} \mid \langle \text{AND_COND} \rangle \mid \langle \text{OR_COND} \rangle$
 $\langle A2 \rangle ::= \text{empty} \mid \langle \text{RED_EXP} \rangle$

$\langle \text{RED_EXP} \rangle ::= \langle \text{ATTR} \rangle \mid \langle \text{RED_EXP} \rangle, \langle \text{ATTR} \rangle$
 $\langle \text{EXP} \rangle ::=$
 $\langle \text{TERM} \rangle \mid \langle \text{TERM} \rangle \langle \text{OP} \rangle \langle \text{CONST} \rangle \mid \langle \text{CONST} \rangle$
 $\langle \text{OP} \rangle \langle \text{TERM} \rangle \mid \% \text{COUNT}$
 $\langle \text{AND_COND} \rangle ::=$
 $\langle \text{COND} \rangle \mid \langle \text{AND_COND} \rangle . \text{AND} . \langle \text{COND} \rangle$
 $\langle \text{OR_COND} \rangle ::=$
 $\langle \text{COND} \rangle \mid \langle \text{OR_COND} \rangle . \text{OR} . \langle \text{COND} \rangle$
 $\langle \text{RED_COND} \rangle ::= \langle \text{ATTR} \rangle \langle \text{COMP} \rangle$
 $\langle \text{COND} \rangle ::=$
 $\langle \text{ATTR} \rangle \langle \text{COMP} \rangle \langle \text{VAL} \rangle \mid \langle \text{ATTR} \rangle \langle \text{COMP} \rangle (\langle \text{VAL_LIST} \rangle)$
 $\langle \text{VAL_LIST} \rangle ::= \langle \text{VAL} \rangle \mid \langle \text{VAL_LIST} \rangle, \langle \text{VAL} \rangle$
 $\langle \text{TERM} \rangle ::= \langle \text{ATTR} \rangle \mid \langle \text{FUNCT} \rangle (\langle \text{ATTR} \rangle)$
 $\langle \text{FUNCT} \rangle ::= \% \text{AVG} \mid \% \text{MAX} \mid \% \text{MIN} \mid \% \text{SUM}$
 $\langle \text{OP} \rangle ::= + \mid - \mid \times \mid :$
 $\langle \text{COMP} \rangle ::= = \mid < \mid \leq \mid \geq \mid > \mid \neq$
 $\langle \text{ATTR} \rangle ::= \text{attribute name}$
 $\langle \text{REL_NAME} \rangle ::= \text{relation name}$
 $\langle \text{VAL} \rangle ::= \text{attribute value}$
 $\langle \text{CONST} \rangle ::= \text{numeric constant}$