Chapter 2

A FLEXIBLE AND EXTERNAL DATA DICTIONARY SYSTEM FOR PROGRAM GENERATION

T. Risch

(Datalogilaboratoriet, Sweden)

A data dictionary is implemented as data structures in a symbol manipulation language, separate from the underlying data base system. A number of programs are built around the data dictionary. The most important program module is a query compiler, which translates a non-procedural query language into a lower level language (COBOL). A feature of the system is compile-time optimization of the access algorithm. Design issues discussed include the decision to separate the data dictionary from the underlying data base system; the future possibility both to interpret and to compile the query language; and the problem of achieving an architecture convertible to any underlying data base system. Experience with one such conversion is reported.

1. INTRODUCTION

This paper describes a data dictionary system with a query language handler. (We will use the term 'data dictionary' and 'data directory' as synonyms). It has the following important properties:

- The system works with an existing data base system but is viewed by the operating system as a separate program. The system includes a data dictionary which describes data bases in the underlying data base system. At present the system works with a data base system called MIMER.[2]
- The user may specify data base access programs in a high level query language. The system then automatically generates COBOL programs which efficiently perform the specified searches.
- The system embodies general knowledge of how to compile the query language. The special method which is used for optimization and code generation, the FOCUS method, is described elsewhere.[19]

- The system is designed to make it possible to use different underlying data base systems. An earlier version of the system[12] worked with IMS.[11,9]

The work has resulted in a working system called LIDAM (LISP Data Manager), which has been used since September 1977 as a tool for data base applications at Uppsala Computer Centre. In this paper we will describe the architecture of the system and discuss different aspects of the design chosen.

2. DATA DICTIONARY IMPLEMENTATION

All data base systems contain some kind of a data dictionary. The data dictionary describes the structure of the data bases using the data base system, the physical data bases. Here 'structure' means a general description of the appearance of physical data bases. In its simplest form the data dictionary only contains names and sizes of fields and files. In more sophisticated systems much additional information is stored, like how different files are connected to each other, statistical information, and so on.

Data dictionaries are used for several purposes. An important use of data dictionaries in LIDAM is to optimize data base searches in a high level query language.

The data dictionary can be connected to the data base system in three main ways:

- The data dictionary can be internally stored in the data base system. The user is given some kind of a language to describe the appearance of data base structures. This Data Definition Language (DDL) is given as input to a compiler, which translates the DDL source code into internal tables in the data base system. The user may not access these tables. This kind of data dictionary is the one normally used by conventional data base systems, such as CODASYL based systems[7] and IMS.
- The tables are stored internally as in case one, but the user is supplied with a number of subroutines to get and store data in these tables. Sometimes, as in System R,[1] the data dictionary is stored in the data base system as conventional files.
- A third method, used by LIDAM, is that the data dictionary system is totally disconnected from the data base system. It can, however, generate data and programs to communicate with the data base system.

3. MOTIVATIONS FOR THE SYSTEM ARCHITECTURE

An important property of the LIDAM system is the separation of LIDAM from the underlying data base system, and therefore of the

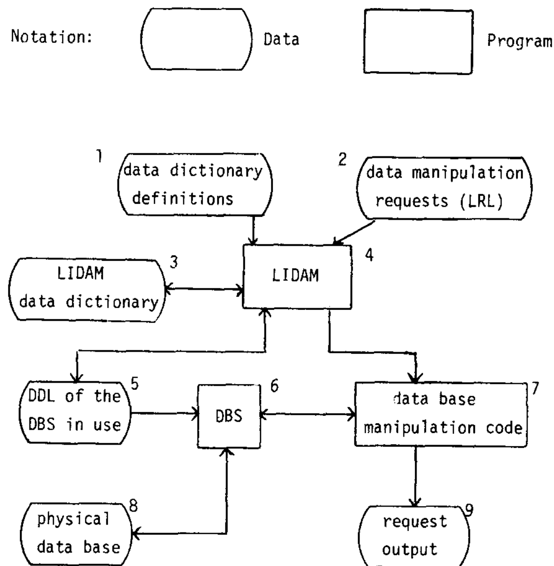

Figure 1. The relationship between LIDAM and the DBS data base system.

LIDAM data dictionary from the data base system's data dictionary. Another important decision was to implement the system and the data dictionary in the programming language INTERLISP.[24,25] These design decisions are motivated below.

The relationship between LIDAM and the underlying data base system (DBS) is illustrated in Fig. 1.

Why INTERLISP? For the implementation we have used the INTERLISP system.[24,25] A number of properties of this system have simplified the implementation.

The logical data structures we needed, lists and trees, are well supported in LISP. In LISP many procedures and programs are available to manipulate and debug list structures (a large procedure library, a structure oriented editor, structure oriented I/O facilities etc.).

LISP supports symbolic manipulation allowing the user to associate data structures and procedures with each symbol, by using property lists.

The LISP data structures are very useful for internal representation of program code. Since it is possible to associate procedures and data with each symbol, the manipulation of the internal representation is simplified. To some extent, LISP itself therefore can be regarded as a compiler-compiler.

The data structures of LISP are also very useful for representing the data dictionary. Since LISP is equipped with predefined I/O for its data structures in contrast with other languages, it is easy to make programs to save the internal list structure representation of the data dictionary on an external file and to load it later.

The program development is simplified by the very interactive architecture of INTERLISP and by the favourable debug facilities. An overview of these and other properties of LISP are given by Sandewall.[21]

The disadvantage with the programming system chosen is that such a flexible system as INTERLISP of course will not have as good performance as conventional programming languages. However, we have not regarded the efficiency considerations as critical since:

- The system generates production programs (which of course are as efficient as possible) and this generation is only made a few times.
- The programming effort has been kept within reasonable bounds. This has made it possible for one person to carry out the system development in less than two years of programming time.
- The system has become more flexible and easy to modify, and thus less dependent of underlying data base system and built in strategies.

Why Separate the Data Dictionary? Since the LIDAM data dictionary is independent of the underlying data base system, programs can be built to work with different data base systems. LIDAM is therefore less dependent on the data base system used.

The LIDAM data dictionary is represented using the LISP data structures. This provides full control over the content and the manipulation of the data dictionary, which has greatly simplified the development of the LIDAM program modules. All the LIDAM program modules make extensive use of the content of the data dictionary.

A detailed data dictionary will have a relatively complicated data structure, which is very extensively manipulated. At the same time the data dictionary is of limited size. Conventional data base technology is more oriented towards handling of large data volumes with little structure, than towards small data bases with complicated structure. For these reasons it is preferable not to store the data dictionary using conventional techniques, but as data structures held in the primary memory (i.e. the data structures of LISP in our case).

LIDAM can communicate with the data dictionary of the underlying data base system by a transformation program between the LIDAM data dictionary and the Data Definition Language (DDL) of the data base system. (Illustrated by the arrows between box four and five in Fig. 1).

One disadvantage with having the data dictionary separated from the data base system is that the information stored in the data dictionary may get out of date as the physical data base is modified. For example, some of the statistical information may be changed. If the data dictionary is connected directly to the physical data base, statistical information may be updated continuously. With the LIDAM approach, programs unfortunately have to be run periodically to go through the physical data base in order to generate statistical information for the data dictionary. Some of the statistical information is, however, not simple to

update continuously, so that statistical programs have to be run also by an integrated data dictionary.

In System R, for example, statistical information is collected when loading the data base.[14] Also, statistical programs have to be run when the statistical information becomes out of date. A similar method is used also by Hammer and Chan,[10] where statistical information is collected for each query processed. This statistical information is added to the data dictionary during data base reorganization.

Furthermore, with the LIDAM technique it is possible to generate the statistical programs needed.

4. SYSTEM OVERVIEW

The LIDAM architecture, with a separately stored data dictionary in LISP, has entailed the development of a number of program modules around the data dictionary. First we will give a short description of these modules, and then an overview of the query language and the program generator.

4.1 Program Modules. Figure 2 below shows the program modules in LIDAM and their control structure. The figure is a blow up of box 4 in Fig. 1.

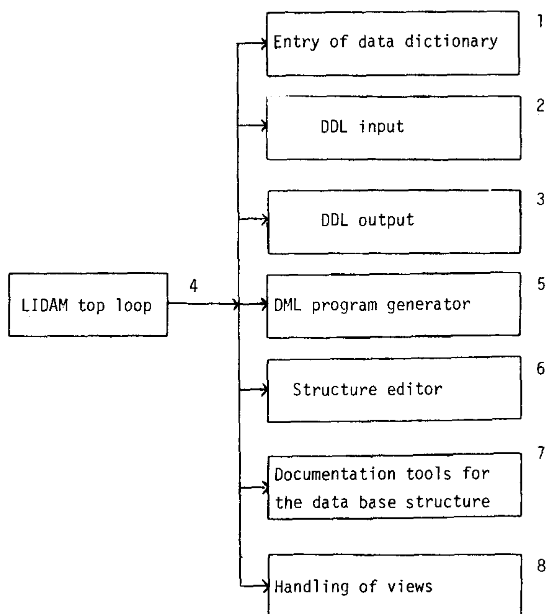

Figure 2. The program modules in LIDAM and their control structure.

The LIDAM top loop. The centre of LIDAM is the LIDAM top loop. The different program modules in LIDAM are activated from this top loop. The user can give commands to call the modules. The top loop is table driven to make is easy to extend it with new commands.

Entry of the data dictionary. The data dictionary is normally created by a special data dictionary entry program. It is an interactive program that prompts the user for name, size, type etc. of data bases, fields and files. From the answers to these questions new parts of the data dictionary are created. During data dictionary entry all the data read is checked to protect the data dictionary from becoming erroneous or inconsistent. The data dictionary entry program is table driven to make it easy to change what to prompt the user for, and how the input will be controlled and stored in the LIDAM data dictionary.

The information content of the data dictionary overlaps with operational information in the underlying data base system. Programs are available to transform data both ways (this is illustrated by the arrows between box 4 and 5 in Fig. 1).

The DML program generator is described later.

The structure editor. In many cases the user wants to make changes in the data dictionary, for example to optimize the data dictionary for common searches. For this reason a specially designed editor, the Structure Editor, is included in the system. The Structure Editor checks that all changes are correct and admissible, in order to keep the data dictionary consistent.

LIDAM as documentation tool. An important use of LIDAM is as a documentation tool. The system can be used to answer queries about data base items, e.g. which file descriptions are stored in the data dictionary, sizes of files and fields, relations between files, statistical values etc. The data dictionary thus documents the structure of the underlying data base.

Views. LIDAM contains a module to define views (or external schemes). A view in the view of LIDAM contains of a number of fields from some of the files in the data base. From that view, the user may regard all of the data base as a flat file with these fields. The program generator automatically maps the view onto the current data base by using the data dictionary.

The views may be linked to different LIDAM data dictionaries. This makes the views less dependent on the data base structure.

4.2 The Query Language. The query language we use, LIDAM Request Language (LRL), is of a similar type as the relational data base languages. In some respects, however, LRL is more powerful than many of the presently available relational data base languages. For example, LRL allows multi-relational queries, i.e. queries where the logical access paths are not specified by the user, but are determined automatically by

the system. Similar techniques are used by Carlsson,[4] Osborn,[16] and Sagalowicz.[20] In some other respects it is less powerful, for example, our intention has not been to construct a relationally complete query language,[8] but to make a language which is user-oriented and solves practical problems.

In LIDAM the data base administrator regards the data base as a network data base, but the user still has the relational view of the data, and furthermore may regard the data base as one big flat file (or relation).

A typical simple LRL statement has the form:

;RETRIEVE <output fields> WHERE <predicate>;
For example:
;RETRIEVE DEPARTMENT WHERE EMPLOYEE="SMITH";

In <output fields> the user states the fields whose values are to be retrieved. The output fields may refer to arbitrary files in the data base, with some restrictions. The <predicate> is the selection rule, i.e. an arbitrary predicate with conditions on some fields in arbitrary files in the data base. The predicate can be composed with AND, OR and parentheses when needed.

The query language mainly contains four types of constructs:

- Data Manipulation Language (DML) constructs, to specify the data base search.
- Features to display the result, i.e. a report generator.
- Features to specify the form of the input to the generated programs, i.e. a dialog generator.
- The usual programming language constructs such as loop statements and procedures. At present LRL only contains a few such constructs, but this set of statements will later be extended.

The users of the system have influenced the design of LRL, and particularly motivated the need for the report generator; the dialog generator; multi-relational queries; and views.

4.3 The Program Generator. LRL is a compiled language. The user gives a number of LRL statements to the LRL compiler at one time. They are transformed into a COBOL program containing calls to the data base system. The COBOL program is compiled by the COBOL compiler and executed the normal way.

A comparison is given in Risch[18] of compilation versus interpretation of high level query languages. The reasons why we have chosen to compile the query language are in brief: a more extensive optimization can be done, the generated programs will be of limited size, and the system will be less dependent on underlying data base system. In Section 6 we will describe how our design can be used to combine compilation and interpretation. Another example of a system having a compiled query language

is System R.[15] Katz[13] has measured the considerable efficiency improvement for different levels of compilation of the query language in the INGRES system.[23]

The method used encourages the specification of simple specialized programs working over the data base. These specialized programs answer simple, specialized queries by prompting the user for desired values of specific fields. The programs are very simple to use, and those who use them do not have to master any query language. It is our intention that the programs shall be used by casual users. LIDAM can generate specialized programs for each application of interest (e.g. account search).

To compile LRL-statements into efficient searches in the physical data base, the program generator has knowledge about search strategies in the data base system. Information from the data dictionary is used to optimize the generated programs.

Steps of the Program Generation. The original LRL statements are successively transformed by the program blocks in the program generator

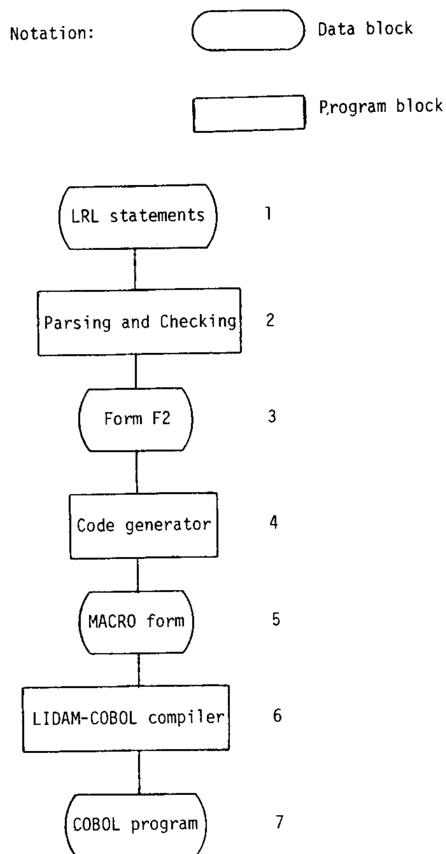

Figure 3. The transformation of the original LRL statement to data blocks.

into new representations or data blocks. The program generator has three steps. They are illustrated by Fig. 3, which is a blow up of Box 4 in Fig. 2. It also illustrates the data flow from box 1 via box 4 to Box 7 in Fig. 1.

Parsing and Checking. The LRL statements are parsed in this program block and their syntactic and semantic correctness is checked. Incorrect statements must be rewritten. There is also a capability to correct some errors interactively when LIDAM finds them, and to do simple editing of the LRL statements. The output data block from this step represents the LRL statements parsed into an internal list structure form, form F2, where references to data base items (files and fields) are replaced by pointers to the corresponding LIDAM descriptors. The substitution of these pointers is done in parallel with the check that the items referenced exist in the data dictionary. This step also checks that the user has the authority to access the referenced data base items, and the names of fields in the view are replaced by the corresponding field descriptors.

Form F2 is saved together with the original LRL statement. No further errors than those detected already by the checker can occur. Form F2 is thus guaranteed to be correct.

The Code Generator. The form F2 data block is given as input to the code generator. A special user command (GO) collects all form F2 structures and gives them to the code generator. The form F2 structures are translated by the code generator into another data block, the MACRO form. This data block is a LISP oriented control structure describing data base manipulations in the data base system, and also describing other normal program operations (arithmetic etc.). The structure of the MACRO form is thus independent of target language (COBOL at present) but contains special handles for the data base system in use (MIMER at present).

The optimization algorithm is applied in this step.

The LIDAM-COBOL compiler. The MACRO form is translated by a LIDAM-COBOL-compiler into COBOL source code. If other languages than COBOL are preferred (e.g. FORTRAN or assembler) this program module must be rewritten. The MACRO form is designed in such a way that it is simple to compile it into source code in any general purpose language.

The COBOL programs contain both calls to the data base system and calls to a number of subroutines to conduct dialogs with the user and to report generation. Thus, it is assumed that there is a small runtime system for the generated programs.

Processing of the Generated Programs. The generated COBOL code is written to a temporary file and the LIDAM system is exited. Then the generated program is completed with control commands. LIDAM generates control cards containing references to the OS-data sets where the physical data base is stored, and to the runtime system for LIDAM generated programs. At this point the generated program can be compiled and executed.

5. EXPERIENCES WITH CHANGING THE UNDERLYING DATA BASE SYSTEM

We will give an overview of the experiences with the change-over from IMS (the previous data base system used) to MIMER (the data base system presently used). This is described in detail in Risch.[18]

Three types of system changes were made:

First, old program modules were adapted to the new data base system.

Second, the system was generalized in order to facilitate adaption to new types of data base systems in the future. It should at least be adaptable on both IMS and MIMER. Since the system has been changed during the conversion, some work remains to extend LIDAM also to work with IMS.

This involved the third, that the system be extended with some wholly new facilities.

File Coupling. The logical linking between files may be achieved in two ways.

Either the access paths could be specified in the query language, or the process paths could be implicitly stored in the data dictionary. The query language compiler would then select the access paths.

In LIDAM we selected the second method in order to minimize the dependence on the underlying data base system and on the data base structure. Our ambition has been to make it theoretically possible to use exactly the same LRL statement to specify a retrieval both for IMS and MIMER (and eventually also another data base system). The use of views (Sect. 4.1) makes it possible for external data bases that have the same content to have the same logical view in both the data base systems.

The Program Generator. The program generator is the module which is the most difficult one to transform to work with different data base systems. We will describe how the different parts of it have been affected.

The change-over from IMS to MIMER had nearly no effect at all on the parser, since the query language used for IMS and LRL on the whole have the same syntax.

The checker is also the same in most parts. One difference is that the algorithm to determine the access paths is different.

The MACRO form may be divided into data base system dependent and data base system independent primitives (as shown in Sect. 4.3). Note that the goal is that LRL shall be independent of the form of the data base system. This will not hold for the MACRO form, which contains direct handles on the data base system. The primitives of the MACRO form may therefore be divided in the following way in a system allowing several types of data base systems (DBS):

When changing data base system, the data base system independent primitives remain. The MACRO form must, however, be extended with new data base system dependent primitives for each new data base system.

The code generation for data base independent parts of LRL may remain the same when changing data base system. However, other code

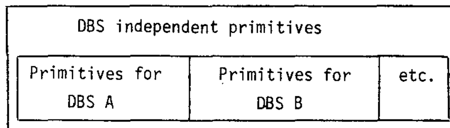

Figure 4. The division of primitives in a system allowing several types of data base systems.

generation will differ considerably. The most difficult problem in the conversion of the program generator is the optimization algorithm. It is not only dependent on the overall structure of the data base system but also on the detailed internal working of the data base system. Information about the latter may be difficult to obtain.

Some data base systems (such as ADABAS[22]) have their own data base search optimizations. To be able to perform detailed optimization as performed by LIDAM, the system must have access to non-optimizing data base system primitives. The more advanced and user-oriented the data base system primitives are, the more difficult it will be to go around the data base system optimizations in order to apply the LIDAM optimizations. This is an example of how intelligence shared by two processes will become difficult to combine. Another example is given by Palme.[17]

If the ambitition level is not too high, both the optimization algorithm and the conversion work is simplified. When changing data base system it is therefore recommended to begin with a simple code generation algorithm, and then successively improve the algorithm. This is the way we worked in the change-over from IMS to MIMER.

6. FUTURE WORK

There are many possible future improvements to the LIDAM system, and directions for further research. We will discuss two extensions to the system which are relatively easy to implement.

Query Language Interpretation. We will describe a method to combine compilation with interpretation of the query language. The idea has been used in an implementation of a procedural query (and update) language[4] for our data base system (MIMER).

An important difference between LISP and most other programming languages is that programs and data have the same representation. As a matter of fact, the programs are list structures of a particular form. These programs, represented by list structures, are interpreted by the LISP interpreter.

Because of this property it is easy to write programs in LISP to manipulate other LISP programs. It is also easy to make programs generate other programs, and then immediately execute (interpret) the programs generated. This can be done without leaving the LISP system, unlike normal programming languages which have to be recompiled before execution.

This property of LISP has made it possible to construct advanced programming systems within the LISP system, containing file handling, editors, error handling etc. Normally there is also a compiler included in the LISP system to translate the LISP programs from list structure representation into pure machine language.

There are a few other languages having this property, among them APL, SNOBOL, and pure machine language. In APL and SNOBOL the programs are represented as strings instead of list structures. APL programming systems have also been constructed, even though they are not as advanced as the LISP programming systems. We know of no similar programming system in SNOBOL, although it is probably possible to construct one.

One interesting extension of the system is to make an interpreter in LISP for the MACRO form. When the MACRO form is generated, instead of translating it with the LIDAM-COBOL compiler (see Fig. 3), it may be directly interpreted. It is possible to go even further; the different MACRO expressions may be defined as LISP functions. The normal LISP interpreter may then perform the interpretation.

To make it possible to interpret the Macro form directly, handles must be built into LISP to access the data base (i.e. the data base system). Thus, it must be possible to call the data base access functions directly from LISP. This can be done in either of two ways:

- A general facility can be built into the LISP system to load external subroutines into the LISP system and thus make them directly callable from LISP. In this way the subroutines of the data base system can be made generally callable from LISP functions. The facility to connect external subroutines to the LISP system is available in several LISP systems.
- The LISP interpreter can be extended with the new data base access functions. This implies that a new LISP system is generated with the data base system built in.

Once the data base system is accessible from LISP, the LRL compiler could be used both to generate specialized programs (in COBOL as at present) and to generate and directly execute the MACRO form.

The MACRO form will in this case be used as a conventional LISP program. Therefore the MACRO form can be compiled by using the LISP compiler. In this way specialized and efficient LISP programs may be generated directly callable from LIDAM. To optimize the compilation, 'compiler macros' may be written to inform the LISP compiler how to compile the different MACRO expressions more efficiently.

Distributed Data Bases. Using LIDAM and the query language LRL, it is possible to generate programs for different data base systems. A possible LIDAM extension is to add modules that know in which computer and in which data base system each type of data is stored. Then, if the data bases are connected in a computer net, and LIDAM has access to the net, LIDAM can make connections automatically and send the generated program to the desired computer to be run there.

A similar technique is used by Sagalowitz[20] in the IDA system. The IDA technique, however, differs from our technique in that it transfers several pieces of code for each retrieval, whereas LIDAM generates a complete retrieval program at once.

8. SUMMARY

We have presented a data dictionary system having an architecture with the following properties:

- The data dictionary is stored separately from the underlying data base system, and it is represented as data structures in a high level symbol manipulation language (LISP).
- Efficient COBOL programs may be generated for data base access from specifications in a very high level query language.
- The architecture of the system has made it possible to work with different types of underlying data base systems.

The general principles of the query language, LRL, are discussed. Both the design and implementation are of interest. The query language allows a powerful type of queries, multi-relational queries, which makes it user oriented and little dependent on the structure and type of underlying data base system. Our practical experiences with LRL have shown the LRL-type of query language to be very useful for solving practical retrieval problems, even though LRL at present is not fully relationally complete.[8] Several LIDAM-generated programs are in practical use, and many of the features of LRL are developed from users' demands.

The architecture of the query language compiler as well as properties of the programming language LISP make it possible to use the query language in the future both in compiling and interpreting mode.

The design of the system also makes it possible to use the system in a distributed computer system, where programs are generated in one computer and executed in other computers.

REFERENCES

1. M.M. Astrahan et al., "System R: Relational Approach to Data Base Management", ACM Transactions on Data Base Systems, pp. 97-137 (June 1976).
1. A. Berghem, A. Haglund, S.G. Johansson, A. Persson, "A Partially Inverted Data Base System with a Relational Approach, MIMER (earlier RAPID)", Uppsala University Data Centre, Uppsala, Sweden (1977).
3. C.R. Carlson and R.S. Kaplan, "A Generalized Access Path Model and its Application to a Relational Data Base System", Proc. of the International Conference on Management of Data, Washington D.C., pp. 143-154 (1976).

4. M. Carlsson, "MIMAN - a query language for DBMS Mimer", DLU 79/5, Datalogilaboratoriet, Sturegatan 2B, Uppsala, Sweden (1979).
5. D.D. Chamberlin, J.N. Gray, I.L. Traiger, "Views, Authorization, and Locking in a Relational Data Base System", Proc. AFIPS National Computer Conference, Anaheim, California (May 1975).
6. D.D. Chamberlin et al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control", IBM Research Journal (November 1976).
7. "CODASYL Data Base Task Group April 71 Report", Ass. for Comp. Machinery, New York (1971).
8. E.F. Codd, "Relational Completeness of Data Base Sublanguages", Data Base Systems, Courant Computer Science Symposium 6, pp. 65-98, Ed. R. Rustin, Prentice Hall, New York (1972).
9. C.J. Date, "An Introduction to Data Base Systems", Addison-Wesley Publishing Company, ISBN 0-201-14452-2 (1975).
10. M. Hammer, A. Chan, "Index Selection in a Self Adaptive Data Base Management System", Proc. of the International Conference on Management of Data, Washington D.C. (1976).
11. IBM, Information Management System Virtual Storage (IMS/VS) General Information Manual, GH20-1260-1.
Information Management System/360, Version 2, Application Programming Reference Manual, SH20-0912-4.
Information Management System/360, Version 2, Utilities Reference Manual, SH20-0915-2.
12. M. Jainz, T. Risch (Eds.), "A Data Manager for the Health Information System Berlin", Computer Programs in Biomedicine 6 (1976).
13. R.H. Katz, "Performance Enhancement for Relational Systems through Query Compilation", National Computer Conference (1979).
14. F. King, Speech at Data Base Symposium at Uppsala Computer Center (September 1977).
15. R.A. Lorie, B.W. Wade, "The Compilation of a Very High Data Language", IBM Research Report RJ008(28098) (May 1977).
16. S.L. Osborn, "Towards a Universal Relation Interface", Conference on Very Large Data Bases, Rio de Janeiro (1979).
17. J. Palme, "How I Fought with Hardware and Software and Succeeded", FOA, Report C10075-M3(E5,H9), Swedish National Defense Institute (FOA), Stockholm (1977).
18. T. Risch, "Compilation of Multiple File Queries in a Meta-Data Base System", (Ph.D. Thesis), Department of Mathematics, University of Linkoping, Linkoping, Sweden (1978).
19. T. Risch, "Optimizing Non-Procedural Multiple File Queries", DLU 79/1, Datalogilaboratoriet, Sturegatan 2B, Uppsala, Sweden (1979).
20. D. Sagalowicz, "IDA: An Intelligent Data Access Program", Conference on Very Large Data Bases, Tokyo (October 1977).
21. E. Sandewall, "Programming in an Interactive Environment: The LISP Experience", ACM Computing Surveys, Vol. 10, No. 1 (1978).
22. R.C. Sprowls, "Management Data Bases". ISBN 0-471-81865-8, John Wiley & Sons Inc. New York (1976).

23. M. Stonebreaker, E. Wong, P. Kreps, G. Held, "The Design and Implementation of Ingres", ACM Transactions on Data Base Systems (TODS), pp. 189-222 (Sept. 1976).
24. W. Teitelman, INTERLISP Reference Manual, XEROS Palo Alto Research Center, Palo Alto, California (1974).
25. "INTERLISP/360 and /370 User Reference Manual", 1975-02-01, Uppsala University Data Center, S-75002 Uppsala, Sweden.