REQUEST: A testbed relational database management system for instructional and research purposes

by BOGDAN CZEJDO and MAREK RUSINKIEWICZ
University of Houston
Houston, Texas

ABSTRACT

A database management system designed for instructional use should offer facilities usually not required in a commercial environment. In particular, it should support a wide range of user interfaces, access methods, and internal organizations in a modular and flexible way, so that the effect on the system performance of choosing one of them may be illustrated.

REQUEST is a relational database management system that, in addition to the usual data definition and data manipulation functions, offers facilities for use in an instructional environment. Various nonprocedural query languages are supported within a single system, using unified database dictionaries. Cross-translation between various query languages is allowed. The results of every important phase of the query transformation during its execution are available to the user.

Preliminary experience with the system has shown that it can significantly facilitate teaching important concepts of the database system organization. At the same time the system has been used as a testbed in many research and development projects.

INTRODUCTION

With the changing emphasis in data processing from algorithms to data, courses in database management are assuming a central position in undergraduate and graduate computer science curricula. When teaching a database-related course, the instructor usually faces the following alternatives[1]: either to use a commercial type DBMS (if available) or to let the students design and implement procedures functionally equivalent to some parts of the DBMS. Both approaches have significant drawbacks.

Commercial DBMSs are (very expensive) software products for the business or scientific, production-type environment. They are, naturally, concerned with problems of reliability, high performance, backup and recovery, data integrity, etc. Such systems are not suitable for use on usually limited and overloaded campus computer installations. The more serious disadvantage of their use for teaching purposes is that, however sophisticated they may be, they are usually used as "black boxes." Not only are the users not allowed to modify the source programs but they cannot even read and analyze them (even if the source code is available the details of performance and security obscure and distract from the basic concepts that support the instructional standpoint). As a result, students get limited experience in writing simple application programs in a database environment and are never exposed to the internal organization of the DBMS. This situation is, of course, highly undesirable.

Letting the students design and implement their own routines to perform some DBMS-flavored data definition and data-manipulation functions seems to be preferred. The great danger of this solution is that the necessary scope limitations and simplifications as well as the small size of such "databases" tend to underemphasize the fundamental differences (at least within current technology) between accessing objects in main memory and secondary storage. As a result, students accustomed to Pascal programming and algorithm complexity analysis tend to develop intuitions that are pathetically inappropriate in a database environment, particularly as far as the suitability of data structures and search algorithms are concerned.

A Relational Query System (REQUEST) was designed at the University of Houston to alleviate the above problems. To facilitate its use in an instructional environment the following general design objectives were adopted.

1. The system should support a wide variety of user interfaces, access methods, internal data organizations, query optimization, and concurrency control techniques in a modular and flexible way, so that the effect of choosing one of them on the user's interactions and system performance can be illustrated.
2. To facilitate the learning of nonprocedural query languages it should allow the student to analyze expressions based on the relational algebra or the relational calculus (queries, integrity constraints, and predicate locks), translate them, and investigate their equivalence or intersections.
3. As a learning tool the system should support an interactive mode in which a user may trace the execution of a query.
4. The reliability and peformance aspects should be assigned secondary importance. Rather, assuming the large number of relatively small databases, we should concentrate on keeping the size of the system manageable so it may be used in an instructional environment with minimal effect on the computer installation.

REQUEST SYSTEM STRUCTURE

The general structure of the system with its main modules and the interactions between them is illustrated in Figure 1. As can be seen from the schema the system supports the usual range of functions expected in a relational DBMS, including parsing, optimization, and interpretation of query language expressions. However, in addition to the above the system includes a number of facilities for instructional use that are not available in commercial DBMSs.

1. Various nonprocedural query languages including user-defined languages are supported within a single system. They are decomposed into a standardized parse tree based on the unified database dictionary system.
2. Cross-translation between various query languages is allowed.
3. A facility to convert query trees back into query expressions in supported languages is provided.
4. The results of every important phase of the query transformation during its execution are available to the user. A facility is provided to examine a query, its equivalent algebraic structure, corresponding parse tree before and after optimization, the access paths selected by a low-level optimizer, and the intermediate pseudocode used by the interpreter.
5. As a query in interpreted, not only the final resulting relation but also the created temporary relations are available to the user; that is, single step tracing is supported.

REQUEST was designed as a relational DBMS running under VAX/VMS. It is intended to support many users concur-

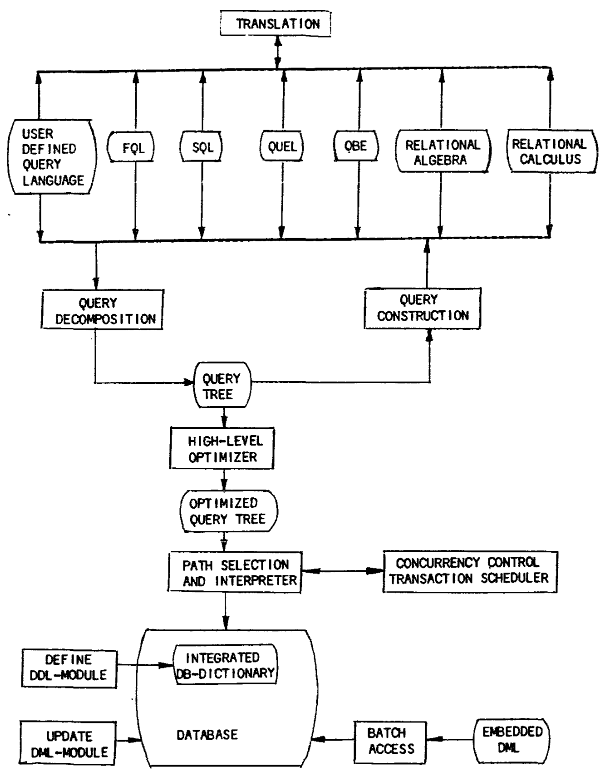

Figure 1—The general structure of the system

rently in both interactive and batch modes. The main modules of the system are discussed briefly below.

Data Definition

The main functions of the DEFINE module are to describe the intension of a databasse and to create and update an integrated data dictionary system. The dictionary is a collection of related files containing the information about database objects stored under the control of the DBMS. The dictionaries are not, however, stored as relations accessible through the system's query facility (as, for example, in SQL/DMS).[2] The reasons for this design decision are pedagogical: it was found that, for beginners, introducing a clear distinction between dictionary relations and data relations is desirable. This enables users to intuitively identify the dictionaries as containing "meta-information" about the data structure. The data dictionaries describe the following:

1. Database relations, both "real" (base tables) and "virtual" (views).
2. Attributes of every relation. For each attribute the corresponding domain together with the "null" value and attribute's location are recorded.
3. Primary and secondary keys for every relation.
4. Integrity constraints.
5. Security constraints such as security clearance required for every type of operation, passwords, etc.
6. Authorized system users with the information about passwords, access grants received, user's security clearances, etc.

The data definition operations can be performed on line, dynamically, in a multiuser environment. Proper synchronization is enforced, if necessary, by the concurrency control module. Relations can be added or dropped; attributes can be added, dropped, modified, or designated as indexes at any time.

Data Update

The update operations (INSERT, DELETE, MODIFY) are performed a record at a time. This was found to be an acceptable solution, because the volume of volatile data manipulated under REQUEST's control is usually small. In addition it allows the concurrency control to be much simplified and a higher degree of concurrency between conflicting transactions to be achieved.

The updates are performed in a user's working space and installed in the database in accordance with the "two-phase-commit" policy.[9] An automatic roll back is performed in case of system malfunction.

Query Decomposition

REQUEST is intended to support a multilanguage environment: a database described by a uniform dictionary system can be queried by any of the query modules corresponding to the different languages. Queries specified directly as sequences of operations of relational algebra and relational calculus or expressed in a user-defined language are also supported. Query decomposition is performed by a parser whose functions include validating relations and attributes names, checking the domains of attributes and constants used in comparisons, and generating an (unoptimized) parse tree.

Query Optimization

While constructing a parse tree the parser does not consider the efficiency of evaluating the tree. The problem of query tree optimization has been substantially researched.[3,10,11] The query optimizer uses several heuristic rules to convert the tree into an equivalent one that could be evaluated faster. Some of these are as follows[12]:

1. selections should be performed as early as possible; that is, select operators should be pushed toward the leaves of the expression tree,
2. selections and projections involving one file should be combined, when possible, so that only one scan of the file is required.
3. joins should be combined with the following projections,
4. if the query involves a common subexpression such as a view it is often beneficial to evaluate it once and then use the resulting relation in subsequent computations.

The optimization rules used here are based on commutative and associative algebraic laws for projections, selections, joins, and Cartesian products and allow one to convert an expression into an equivalent one. These rules can be applied independently of the information about the internal organization of files used to store the relations.

Path Selection and Query Interpretation

Before a join or a selection is performed, the file(s) should be preprocessed; in particular we should take advantage of existing secondary indices and ordering of the files (if applicable to the operation).[4] For every basic operation a decision is made on how it should be implemented, taking into account the cardinality of the relation, the number of distinct values occurring in each attribute's domain, the expected reduction of a table as a result of select operation, the existing secondary indices, etc. If a temporary table has to be created and used as an input argument for a subsequent join or selection operation, the relevant secondary indices should be created while constructing the table.

Access Method

REQUEST uses its own access method implemented on top of the VAX/VMS Record Management Services (RMS). Access method routines that could be invoked from a high-level programming language perform basic file and record manipulation functions. The decision to provide an interface to the RMS rather than implement a totally independent file system was made to achieve an acceptable speed of operation. The file organizations include index sequential, hashing, and extendible hashing.[13]

REQUEST Batch

Both data definition and data manipulation facilities discussed so far are available to the user in an interactive mode from a terminal. In many applications an access to a database from a general-purpose host programming language is required. In REQUEST, access to a database can be achieved in the embedded mode through one of the two available interfaces:

1. a Pascal preprocessor for SQL that produces relatively small executable modules by performing the syntax error checking, name validation, and access path selection at the preprocessing stage so that only selected relevant modules of the DBMS need to be linked with the host-language program
2. a general call facility that allows the DML statements to be executed from any programming language obeying VMS calling and parameter-passing conventions

Concurrency Control

To support a multiuser environment it is necessary to schedule conflicting transactions using some concurrency control mechanism. The well-known concept of serializability is employed to assure that both read-write and write-write conflicts are scheduled according to some serialization order.[9] A wide variety of concurrency control algorithms proposed in the literature can be classified into three main groups:

1. locking-based methods (exclusive and shared locks, predicate locks, intent locks)
2. timestamp-ordering-based methods (basic T/O, conservative T/O, multiversion T/O)
3. circulating-permit-based algorithms

A transaction scheduler is a program module that performs the following functions: it keeps track of the status of each data item in the database; it receives transaction's requests to access a data item; it either allows the transaction to proceed (updating the status indicators) or rejects it if the requested operation is in conflict with other transactions in progress. Depending on the concurrency control algorithm used, the rejected transaction will be queued, or under some algorithms it will be rolled back and restarted. Communication between transactions and the scheduler is implemented through mailboxes and event flags.

It was found that an intent locking scheme capable of supporting different granularities of locks seems to be particularly appropriate in a system that like REQUEST, supports both record-at-a-time and set-at-a-time operations.[8]

Query Translation

Translation of queries is an important facility that enables the user of an instructional system to see the equivalence of expressions specified in different languages.[7] Some transformations such as reduction of relational algebra to tuple-relational calculus are well described in the literature.[12] Others, such as the direct transformation from SQL to QUEL need to be investigated. The approach adopted in REQUEST is based on formulating translation rules and employing them to perform symbol and tree manipulation.[5,6]

Query Construction

Query construction is a unique feature of the instructional DBMS. This module accepts the parse tree based on relational algebra as an input. It produces as an output query expressions in any of the supported languages including a user-defined language. This feature is not necessary in commercial DBMSs but very useful for student training. In addition, this facility provides an alternate way of translating queries, by first decomposing a query into a parse tree and then constructing a query expression in a different language.

CONCLUSIONS

The development of the system started in 1980 as a research project of the authors at the Department of Computer Science of the University of Houston. The first version, which constitutes a functional subset of the system, was completed in 1981. Since then it has been used successfully both as a teaching tool in database courses and as a testbed system for research. Currently available modules include, among others, integrated data dictionary system, parsers, optimizers and an interpreter for SQL, locking and T/O based transaction schedulers, etc. An important implemented part of the system is a friendly query interface that guides an inexperienced user through the database definition process and allows him to formulate queries based on the relational algebra in a menu-driven mode. Other parts of the system including the query translation and query construction modules are currently being designed and developed.

The preliminary experiences with the system have shown that it can significantly facilitate teaching of the important concepts related to the database system organization. At the same time, REQUEST has been used in many research and development projects including the design of an integrated text and graphics database system.[14] Although the initial results are quite satisfactory, a number of important research issues will have to be resolved before the system can achieve its full functional scope.

REFERENCES

1. Bradley, J. *File and Data Base Techniques.* Holt, Rinehart and Winston, 1981.
2. Astrahan, M. M., et al. "System R: Relational Approach to Database Management." *ACM TODS,* 1 (1976).
3. Smith, J. M., and Yang, Y. T. "Optimizing the Performance of a Relational Algebra Database Interface." *Comm. ACM,* 18 (1975), pp.
4. Grifiths, P. P., et al. "Access Path Selection in a Relational Database Management System." R. J. 2479, IBM San Jose, 1970.
5. Czejdo, B. "Transformation of Universal Algebraic Expressions in PASCAL," ACM Computer Science Conference, Kansas City, February 12–14, 1980.
6. Czejdo, B. "ALGEBRA—Language for Automatic Transformation of Universal Algebraic Expressions," ACM Computer Science Conference, St. Louis, February 24–26, 1981.
7. Czejdo, B., and M. Rusinkiewicz. "Query Transformation in an Institutional Database System." *ACM SIGCSE Bulletin,* 1 (1984), pp. 217–223.
8. Gray, J. N., et al. "Granularity of Locks in a Large Shared Database." *Proc. 1st Int. Conf. on VLDB,* September 1975.
9. Bernstein, P. A., and Goodman, N. "Concurrency Control in Distributed Database Systems." *Computing Surveys,* 13 (1981), pp.
10. Yao, S. B. "Optimization of Query Evaluation Algorithms," *ACM TODS,* 4 (1979), pp.
11. Aho, A. V., et al. "Equivalence of Relational Expressions." *SIAM J. Computing,* 8 (1979), pp.
12. Ullman, J. D. *Principles of Database Systems.* Computer Science Press, 1983.
13. Fagin, R., et al. "Extendable Hashing—A Fast Access Method for Dynamic Files", *ACM TODS,* 4 (1979), pp. 315–344.
14. Rusinkiewicz, M., and Li, Y. Y. "Textual and Graphics Database for SAL Geophysical Models." University of Houston, *SAL Review,* 10 (1982), pp. 417–423.