LBASE: A LOGICAL DATABASE MANAGEMENT SYSTEM

N. Vasiliades and J. Vlahavas
Digital Systems & Computers Lab.
Dept. of Physics, Aristotle University of Thessaloniki
54006, THESSALONIKI, Greece

1. Abstract

Logic offers a uniform environment both for data description and program execution and provides a powerful programming language with the use of "headed" queries. Relational data model proved to be the best model of the "conventional" database theory This paper describes the advantages of connecting a relational database management system (RDBMS) with logic programming, building a logical DBMS (LDBMS) that extends the power of the relational data model. Relations are described both by facts (assertions) and rules (deductions), so data become more meaningful, and complex queries can be answered. Memory organisation also provides space saving and efficient indexing using b-trees instead of sequential searching.

2. Introduction

Even though relational data model has boosted database theory [1], some problems and limitations arise from the fact that relational databases are flat and therefore full exploitation of the relational model capabilities requires programming skills from the user. Query languages for the RDBMSs have extended to full procedural programming languages. A relationally complete query language should include five basic relational algebra operations: selection, projection, cross-product, union and difference, while a minimally relational one, like DBASE III PLUS, should include only three: selection, projection and join.

PROLOG (PROgramming in LOGic) is the most widely known representative of logic programming. Logic programming in general and PROLOG in particular are more humanised versions of declarative programming and are mainly used for symbolic computation and theorem proving [2]. PROLOG programs consist of facts and rules. Rules are general axioms that compute data at query time, while facts are special rules without bodies and act as the actual data.

As logic blurs the distinction between databases and programs, it is increasingly favoured as a query language for a database described in logic [3]. The connection of an RDBMS and PROLOG results in a Logical Database Management System (LDBMS), which inherits the advantages of both its ancestors and eliminates their main disadvantages [4]. In this paper we describe LBASE, an LDBMS which joins DBASE III PLUS to Arity PROLOG.

3. Implementation issues

The advantages of our system can be grouped into two main categories: data organisation and data semantics. PROLOG's structure flexibility allows LBASE to save space both on disk and in memory, because records and fields have variable length and empty fields do not occupy space. Records are stored as lists in memory data chains, so they can be indexed using b-trees, instead of being searched sequentially. PROLOG offers an underlying deduction mechanism that allows complicated logical queries to be answered. Our problem is to build such an meta-interpreter of PROLOG in PROLOG, so as to incorporate the database search mechanism that we proposed above. In order to do so we separate the facts, the actual data, from the rules that compute the data at query time. The former are stored in the Extensional Database (EDB), while the latter in the Intensional Database (IDB). This allows rules and facts to be treated differently. Therefore we have to analyse an incoming query, so as to become a ground one, i.e. to be consisted of queries only to the EDB and not to the IDB. Next we analyse our question answering mechanism for the logical query language (LQL).

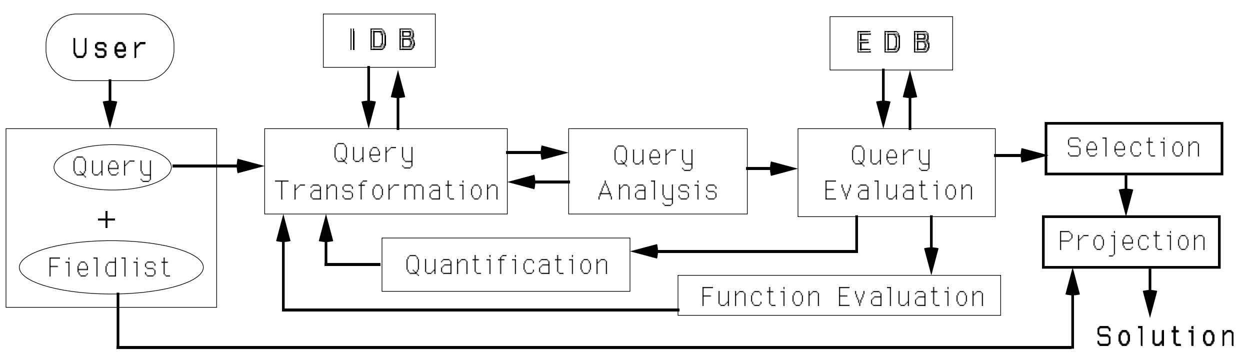

Figure 1. The LQL question-answering mechanism.

The query consists of a condition to be tested in the database and make the selection of the tuples and a fieldlist which defines the projection to the attributes. The condition is being entered in the "transformation cycle" which consists of five modules. After a query is entered the IDB is searched for a rule to be unified with the query, in the query transformation module. Should not the query be consisted of a simple virtual relation, then it is advanced to the next stage of query analysis. There all complex queries break down into many simpler ones and the cycle is repeated for each one of them. Query evaluation stage decides whether simple queries are functions or quantifiers or ground ones. Functions are being proceeded to the function evaluation stage, where the function argument is being re-cycled and the results of the new query are being used to compute the function output. Quantification uses the function "count" which counts how many times its condition-argument is being satisfied in the existing database. Notice that we use Boolean algebra to transform a complicated query to a query that consists only of logical ANDs and ORs. This will help to execute a query in parallel in a future implementation.

Finally, some ground queries are being produced and the query evaluation stage makes use of the EDB in order to produce the results of the query. Search in the EDB is done only with the use of existing b-trees that accompany every relation in memory. Data chains are not being used by LQL but only from CQL, the DBASE III PLUS equivalent query language of LBASE. This technique enable us to benefit from the binary representation of a relation at the physical layer and from the unary one at the virtual layer [3].

4. Conclusions and future work

LQL is relationally complete, in contrast to DBASE III PLUS and is set oriented according to the relational model theory. In LQL data become more meaningful due to the use of facts and rules that model the database according to the user's attitude towards the data. In LQL we can state deduction rules that compute the data in query time [5] rather than explicitly store them in memory. We can also state data integrity rules that check the correctness of the data according to our interpretation or according to functional dependencies among the attributes [5]. LQL also supports use of set and bag functions, quantifiers, and constants. Finally, one can build complicated queries, then add a name in front of them and transform them to logic programs [3]. Thus, the connection of predicate calculus with a relational database results in a more powerful system than the conventional ones, which could become the core of larger "intelligent" systems, such as expert systems, tutoring systems and decision support systems. Future work may include use of hashing tables and hashing buckets in order to reduce the amount of data to be searched.

5. References

[1] E.F. Codd,"A Relational Model of Data for Large Shared Data Banks", CACM 13, 1970, 377-387.

[2] W.F. Clocksin and C.S. Mellish, "Programming in Prolog." 2nd ed., Springer-Verlag, Berlin, 1984.

[3] R. Kowalski, "Logic for Data Description", In "Logic and Databases", editors H. Gallaire and J. Minker, Plenum Press, 1978, pp. 77-103.
[4] H.L. Berghel, "Simplified Integration of Prolog with RDBMS", DATA BASE Spring, 1985, pp. 3-12.
[5] H. Gallaire, J. Minker and J.M. Nicolas, "An Overview and Introduction to Logic and Data Bases", In "Logic and Databases", editors H. Gallaire and J. Minker, Plenum Press, 1978, pp. 3-30.