

## DEVELOPING A METHOD FOR QUERYING EXTERNAL DATABASE IN PROLOG

Dr. Hazim A. Farhan, Dr. Hussein H. Owaied

Department of Computer Science, Meddle  
East University of graduate Studies,  
Amman, JORDAN

[hfarhan@meu.edu.jo](mailto:hfarhan@meu.edu.jo), [howaied@meu.edu.jo](mailto:howaied@meu.edu.jo)

### ABSTRACT

The paper presents a method used for querying external Database in PROLOG programming language environment. The method is based on coupling of two methods used in deferent application environments. This method is complementary indexing techniques, Bit-Mapped and B+Tree. The proposed method is very compact and fast to resolve a query, since they use bitwise operations, which are executed quickly by most computers. The method provides the capability of allowing powerful and fast data retrieval from external database in the PROLOG language, even though there is no direct query command is available for the user; therefore, this method can be used to utilize PROLOG language for external database for powerful querying. Since PROLOG language usually dealing with internal database, which is composed of facts and rules, which can be added directly into and remove from PROLOG program at runtime. PROLOG provided with many built-in predicates to add new facts and to remove existing facts to/from the database. Therefore, this method will be augment PROLOG programming language to deal with the external database in powerful and efficient manner. The method has been in experiment database for large amount of records, which are more than 100000 records, and shows the effectiveness in the retrieval of records in term of speed, which is faster than in the ordinary manner. The development of this method is based on the methodologies of both relational database and logic programming aspects. In this paper is the description of the steps which are necessarily for adaptation in real-world domain.

*Keywords: Database, PROLOG Language, Artificial Intelligence and Search Techniques.*

## 1 INTRODUCTION

Knowledge-representation formalism can be viewed as Formal language that can be used to represent knowledge. A typical example of knowledge-representation formalism is predicate logic. Also programming languages can be seen as knowledge-representation languages, although may of those languages have not been designed to represent every-day human knowledge well, but rather as language which allow for representing specialized knowledge of how to control a machine. In the context of expert systems, an object is usually understood to be a structured attribute description of a thing. Often special-purpose languages are used to represent objects, but from a semantic point of view, objects in many different languages for objects can be understood in terms of Horn-clause logic. This meaning is different from that in object-oriented

programming, where an object is an active process that is normally able to communicate by exchanging messages with other objects [9]. The Logical Inference rule can be used for deriving knowledge from given knowledge. If this knowledge is represented in the form of standard logic, then this term is synonymous with deduction. PROLOG program can be defined as a set of Horn Clause, Clause that contains at most one positive literal. So the programmer provides facts and rules and then asking queries. PROLOG System will provide the answer through the process called Resolution process which is a special logical inference rule to drive new clauses from two given clauses. This simple inference rule is both sound and complete for deducing inconsistency (called refutation completeness), except when it is used with clauses in predicate logic, where it has to handle the situation where it can derive clause containing similar literals, only

differing with respect their the universally quantified variables. Thus, resolution plus factoring is sound and refutation complete. Usually PROLOG used the stack for transferring arguments and return addresses for predicate calls. The stack also holds the information for backtrack points. But the heap holds all objects that are more or less permanent, such as database facts, window buffers, file buffers etc. And also uses the global stack, normally called gstack, is the place where lists, compound structures and strings are placed. The Gstack is only released during backtracking. Finally the trail is only used when the program uses reference variables. It holds information about which reference variables must be unbound during backtracking. The trail is allocated in the heap. PROLOG system support different strategies for databases, these are:

1. Internal PROLOG fact databases,
2. External PROLOG databases (with extensive support for b+ trees) and
3. External third party databases. (Bindings is supplied)

Method (1) is very easy and work fast for dataset up to a few thousand records, the reason for that all the processes of records will be done in the main memory. Thus, if there are many thousand of records, in the database required to be processed, in this case it is necessary to implement a strategy for processing huge database to deal with the external database in powerful and efficient manner. Therefore, in this paper is a design and implementation of a method of complementary used of both method (2) and method (3) through two indexing techniques, Bit-Mapped and B+Tree. In the following sections are the descriptions of Bit-Mapped, B+Tree and the proposed method.

## 2 BIT-MAPPED

Bit Map Indexing is a technique commonly used in relational databases where the application uses binary coding in representing data. This technique was originally used for low cardinality data but recent applications like the "Sybase" have used this technique efficiently [4]. The word cardinality is an SQL term to refer to the uniqueness of a data value in a particular column in a database table. Low cardinality means that the values in the data column of the data table are pretty common.

These data may include gender, race, age, hair color or status flags and Boolean data. On the other hand, high cardinality data may refer to data in a column which are unique. These data may identification numbers, user names; email addresses, social security number. A bit mapped index is a special type of index that executes queries by performing logical operations on the bit map. Generally, a bit mapped index is generated for one column of the database's table. As an example, consider the gender field in a student's database, this column contains only tow values – male or female. If the bit mapped index used, so the 0 and 1 may be used to denote the values of male and female. If the query is for all male students in a particular department within a large university with several campuses, the search will only try to access the index instead of search the whole table. So, this can greatly speed up the process. In fact, the three main reasons why professionals use bit mapped indexing technique when dealing with low cardinality data are performance, storage and maintainability. In a performance test condition, it was found out that bit mapped index did a very impressive performance in execution times of some given queries by orders of magnitude. Most queries that benefited from bit mapped indexes have WHERE clauses containing multiply predicates on columns with low cardinality. It was also found out that bit mapped indexes are very useful in doing very complex ad hoc queries which contain long WHERE clauses that involve low cardinality data. In terms of storage, bit mapped index can incur only a small storage cost compared to the needs of the B-indexes and this means a big savings on companies. This can be especially true to many companies maintaining large data warehouses spread across several geographic locations But of course there is also a negative to using bit mapped indexing. When doing inserts and deletes on a table, using this indexing technique will automatically results in a wave of updates to all associated indexes. If the number of rows is very large and covered only by a single bitmap index entry, concurrent delete and insert activities can result in massive contention. There are many other alternatives for bit mapped indexes such as dealing with large amount of records in certain types of table, such as students table, employee table, and other types of tables with low cardinality data. A bit mapped index may be the best

alternative to the equally efficient B-Tree index in certain given conditions. The advantage of using bit mapped indexes is typically very quick to build and tend to be small size although the size can dramatically vary with the distribution of data. Bit mapped index are generally beneficial when using queries that use several indexes at once but updating bit mapped columns can dramatically degrade the quality of indexes. Finally, the other advantage of using bitmapped indexes is the performance for certain queries and their relatively small storage requirements.

### 3 B+TREE

The B+ tree is analogous to a binary tree, with the exception that, in a B+ tree there is more than one string as a key is stored at each node. B+ trees are also balanced; this means that the search paths to each key in the leaves of the tree have the same length. Because of this feature, a search for a given key among more than a million keys can be guaranteed, even in the worst case, to require accessing the disk only a few times--depending on how many keys are stored at each node. The B+ tree is a data structure can be use to implement efficient method for very large amounts of data enable a correspondingly efficient searching algorithm. B+ tree provides an index to a database, so this is why B+ trees are sometimes referred to as indices [5]. In PROLOG, a B+ tree resides in an external database. Each entry in a B+ tree is a pair of values: a key string and an associated database reference number. When building the external database, it must insert a record to the database and establish a key for that record. The PROLOG "btree" predicates may then be used to insert this key and the database reference number corresponding to this record into a B+ tree [10]. When searching a database for a record, it must obtain the key for that record, and the B+ tree will give the corresponding reference number for that record. Using this reference number it can be easily retrieve that record from the database. As a B+ tree evolves, its entries are kept in key order. This means easily obtain a sorted listing of the records. Although B+ trees are placed in an external database, they don't need to point to terms in the same database. It is possible to have a database containing a number of chains, and another database with a B+ tree pointing to terms in those chains. In a B+ tree, keys are

grouped together in pages; each page has the same size, and all pages can contain the same number of keys, which means that all the stored keys for that B+ tree must be the same size. The size of the keys is determined by the KeyLen argument, which generated when creating a B+ tree. If an attempt to insert strings longer than KeyLen into a B+ tree, PROLOG will truncate them. In general, always required the smallest possible value for KeyLen in order to save space and maximize speed. When the B+ tree is created, it must be also give an argument called its Order. This argument determines how many keys should be stored in each tree node; usually, must determine the best choice by trial and error. A good first try for Order is 4, which stores between 4 and 8 keys at each node. Usually the value of Order chosen through try and error method because the speed of searching B+ tree depends on the values KeyLen Order, the number of keys in the B+ tree, and the computer's hardware configuration [10].

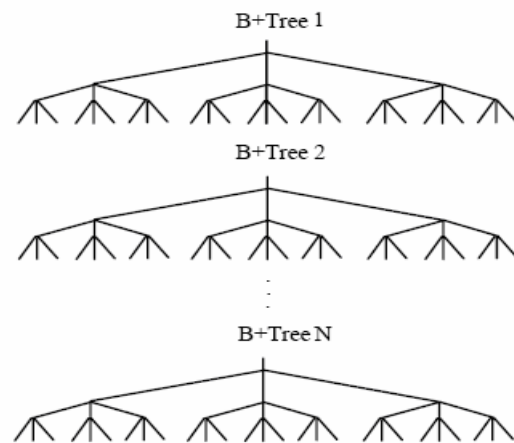


Figure-1: B+Tree

### 4 PROLOG AND RELATIONA DATABASES

PROLOG program consists of facts and rules, and viewed as logical assertions constituting a knowledge base, PROLOG then search the knowledge base for an answer to a query or the goal [5]. The similarity of PROLOG facts to tuples of a relational database should be fairly obvious. A PROLOG predicate is normally interpreted to be nothing more than a relation over some domain. More interestingly, a PROLOG rule may be viewed as a join of multiple database relations. The conclusion of a PROLOG rule would in the relational model is a derived relation, or view, over

base relations. There are good reasons for wishing to marry PROLOG to a relational database. On the one hand, a PROLOG-like language is a concise and intuitive way of specifying instructions to a database (it is arguably more friendly than using SQL). On the other hand, PROLOG systems would greatly benefit from the ability to store large amounts of information in secondary memory and from the optimization techniques built into most database systems. In practice, almost all real world systems linking PROLOG and a relational database system simply tack on a software interface between a pre-existing PROLOG implementation and a pre-existing relational database system. In other words, the two systems are loosely coupled. The interface allows PROLOG to query the database when needed, either via the automatic translation of PROLOG goals into SQL or else by direct embedding of SQL into the PROLOG code. Regarding the former technique, certain PROLOG predicates are designated as database predicates. Instead of attempting to unify them with clauses in an internal PROLOG database, such predicates and their arguments are translated into SQL queries that are then directed to an external database. The tuples returned by the database are treated simply as fully bound PROLOG facts [2].

## 5 PROLOG DATABASE INTERNAL VS EXTERNAL

PROLOG system, usually use the internal database in order to process the fact through the application of the `asserta`, `assertz`, `retract`, and `retractall` commands, which are very simple to and suitable for many applications. However, the RAM requirements of a database can easily exceed the capacity of the computer memory especially when the database is large, so the external database system has been designed for solving the problem associated to large databases [10]. An external database is composed of an external collection of chained terms; these chains give the direct access to data that is not a part of the PROLOG user program. The external database can be stored in any one of three locations: in a file, in memory, or in EMS-type expanded memory under DOS. The external database supports B+ trees, which provide fast data retrieval and the ability to sort quickly, and it supports multi-user access by a mechanism for serializing the file accesses inside transactions [2].

The PROLOG' external database system supports these different types of applications, while meeting the requirement that some database systems must not lose data during update operations--even in the event of power failure. PROLOG' external database predicates provide the following facilities [10]:

- efficient handling of very large amounts of data on disk the ability to place the database in a file, in memory, or in EMS-type expanded memory cards under DOS
- multi-user access
- greater data-handling flexibility than provided by the sequential nature of PROLOG
- automatic backtracking mechanism
- the ability to save and load external databases in binary form

## 6 PROLOG EXTERNAL DATABASE

The PROLOG external database consists of two components: the data items--actually PROLOG terms stored in chains, and corresponding B+ trees, which will be used to access the data items very quickly. The external database usually stores data items in chains rather than individually so that related items stay together. For example, one chain might contain part numbers to a stock list, while another might contain customer names. Simple database operations, such as adding new items or replacing and deleting old items, do not require the B+ tree. The B+ tree comes into play when you want to sort data items or search the database for a given item [6, 8].

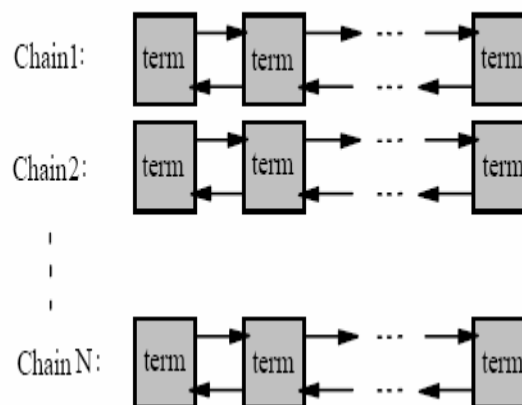


Figure-2: Structure of PROLOG External Database

## 7 QUERY LANGUAGE VS. PROLOG EXTERNAL DATABASE PROBLEM

Query Language is used to express queries to be answered by a database system [8]. For example SQL (Structured Query Language) a standard query language used by many programs that manipulate large database (such as ORACLE, MS-SQL Server ... etc). SQL is a relational complete language, which has become the standard language for relational system [8]. It is perceived as being, easy to use, and portable between different vendors DBMS-SQL.

Now, in the next few lines, we will discuss the problem that exists in PROLOG external database. We can clearly observe this problem by comparing SQL with PROLOG external database, data retrieval and manipulations. As we mentioned in section 6, that PROLOG external database consist of two parts; the chains of terms and the B+Tree. The B+Tree consist of two parts; the key string and the database reference number. When we want to retrieve any record from the database, we must give the key string to the B+Tree index, and the B+Tree will return the corresponding reference number, by using this reference number we can retrieve the record from the database. This means it is only one record can be retrieved from the database at a time (Specifying the key string). But in case of querying the database, about all records that satisfy the desired query conditions (query where part). Now, we can clearly observe the PROLOG external database weakness. This means, that the PROLOG external database can not be used as a powerful query language (like internal database). In the next section we will discuss the proposed solution to eliminate this weakness.

## 8 THE PROPOSED METHOD

In this section, we will explain the proposed method, which enables quick and powerful technique to querying PROLOG external database. As mentioned before, we can retrieve only one record at a time from Prolog's external database, by giving the KEY STRING to B+Tree, and it will return the corresponding reference number, which allows easily retrieve this record from the external database.

By using Bit-mapped indexing technique we can retrieve set of KEY STRINGS that matches

the query conditions. B+Tree will use these KEY STRINGS, in order to retrieve the database reference numbers, and usually these numbers used to retrieve the group of corresponding records from the database, as shown in figure 3 – the proposed method. Bit-mapped indexing technique is very compact because it is composed of long strings of bits; it contains one entry for each row in the table. The bit string for one value in million row table would take up only around 122KB [4, 6]. PROLOG can quickly scan that bit-string for rows that match the criteria specified in the query. Bit-Mapped indexed, really shine when you can use several condition together to solve the query [4].

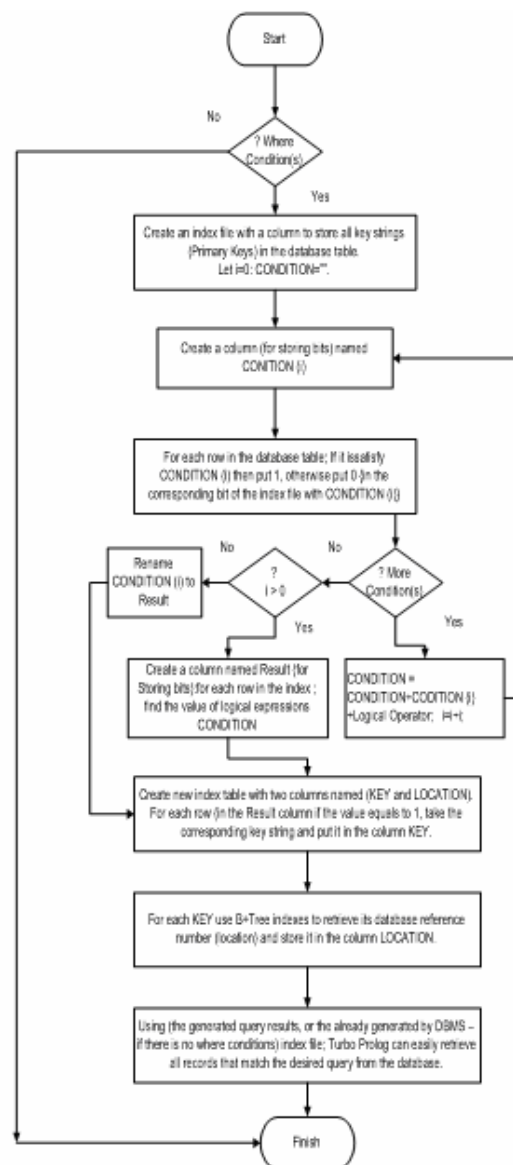


Figure-3: The flowchart of the proposed method

## 9 PRACTICAL EXAMPLE

The following database tables used in Engineering Company. The entities to be modeled are the employees (EMP) and projects (PROJ). for each employee, we would like to keep track of the employee number(ENO), name(ENAME), title in the company(TITLE), salary(SAL), identification number of the project(s) the employee is working on(PNO), responsibility with the project(RES), and duration of the assignment(DUR) in months. Similarly, for each project there are details stored in PROJ table, which are the project number (PNO), the project name (PNAME), and the project budget (BUDGET) [9].

Table-1: EMP table

<i>E NO</i>	<i>ENAME</i>	<i>TITLE</i>	<i>SAL</i>	<i>P NO</i>	<i>RESP</i>	<i>DUR</i>
1	J. Do	Elect. Eng.	40000	P1	Manger	12
2	M. Smith	Analyst	34000	P1	Analyst	24
2	M. Smith	Analyst	34000	P2	Analyst	6
3	A. Lee	Mech. Eng.	27000	P3	Consultant	10
3	A. Lee	Mech. Eng.	27000	P4	Engineer	48
4	J. Miller	Programmer	34000	P2	Programmer	18
5	B. Casey	Sys. Analyst	34000	P2	Manager	24
6	L. Chu	Elect. Eng.	40000	P4	Manager	48
7	R. Davis	Mech. Eng.	27000	P3	Engineer	36
8	J. Jones	Sys. Analyst	34000	P3	Manger	40

Table-2: PROJ table

<i>PNO</i>	<i>PNAME</i>	<i>BUDGET</i>
P1	Instrumentation	150000
P2	Database Developer	135000
P3	CAD / CAM	250000
P4	Maintenance	310000

Consider the following query statements using SQL and PROLOG respectively:

```
SELECT * FROM EMP WHERE
RESP="Manager" and TITLE="Elec. Eng.";
EMP (_,_,Elec. Eng.",_,_, "Manager",_).
```

The following table (3), show the conceptual view of the Bit-mapped indexing, it contains two columns, one for the responsibility within the project (RESP="Manager") and other

for the title in the company (TITLE="Elec. Eng.").

Table 3: Bitmapped index on RESP and TITLE

<i>ENO</i>	<i>RESP (Manger)</i>	<i>TITLE (Elec. Eng.)</i>
1	1	1
2	0	0
2	0	0
3	0	0
3	0	0
4	0	0
5	1	0
6	1	1

Adding two bits stream as a logic operation that most computers can be extremely quickly (Turbo PROLOG provided with commands for bitwise operations. i.e. it can be easily manipulate with Bitmapped indexes). PROLOG can easily add the two bits then use the resulting Bitmap to identify those rows that match all the criteria.

Table 4: Query result file (Adding two bits together)

<i>ENO</i>	<i>RESP (Manger)</i>	<i>TITLE (Elec. Eng.)</i>	<i>Anded Result</i>
1	1	1	1
2	0	0	0
2	0	0	0
3	0	0	0
3	0	0	0
4	0	0	0
5	1	0	0
6	1	1	1
7	0	0	0
8	1	0	0

Now it can easily retrieve the records from the database using the Prolog's B+Tree index that takes the key strings from the resulting Bitmapped, and return the database reference numbers to retrieve the corresponding records from the database (in our example, records number 1 and 6).

## 10 CONCLUSION

In this paper, the proposed method is for coupling two indexing techniques to make queries in Prolog's external database. PROLOG supports a powerful querying for internal database, but it is not for external database it is poor and very slow. As known, Prolog's external database can retrieve one record (term) at a time by giving a key string to B+Tree to return its database reference number, this

reference will be used to retrieve the desired record from the database table. Moreover, Bitmapped indexes are very compact and fast to resolve a query, because it uses the bitwise operations, which are executed quickly by all computers. Therefore, this method can be used to utilize Prolog's external database for powerful querying. Finally, the suggested method is compact but relatively slow compared with other SQL's that use purely Bitmapped indexing technique.

## REFERENCE

- [1] Belkin N. and Coll C., "Cases Scripts, and Information-Seeking Strategies on the Design of Interactive Information Retrieval Systems", Expert System with Applications, Vol 9, No. 3, PP. 379-395, USA,1995.
- [2] Henning C., "Introduction to PROLOG as a database language", A course note, Roskilde University, Computer Science Dept., 2003.
- [3] Huang Y., et al, "Comparing Case-Based and Backtrack Search in a Database Application", Expert System with Applications, Vol 12, No. 1, PP. 53-63, USA,1997.
- [4] Ian S. and Jim M., "Oracle Rdb and Bit Mapped Indices", Relational Technology Group, Oracle New England Development Center, 2005.
- [5] Maier F. ,et al, "Efficient Integration of PROLOG and Relational Databases in the NED Intelligent Information System", Artificial Intelligence Center, The University of Georgia, Athens, GA, USA,2004.
- [6] Marg N. S., "ORACLE9i Bible", IDG Books WorldWide, Inc., USA, 2002.
- [7] Mitri M., "Combining Semantic Networks with Multi-Attribute Utility Models: An Evaluation Database Indexing Method", Expert System with Applications, Vol 9, No. 3, PP. 283-294, USA,1995.
- [8] Page A. J., "Relational Database – Concepts, Selection and Implementation", SIGMA PRESS – Willmslow, U.K., 1998.
- [9] Peter Lucas, "EXPERT SYSTEMS", Published in the Encyclopedia of Life Support Systems of UNESCO, 2006.
- [10] PROLOG Development Center, "TURBO PROLOG User's Guide, H.J. Holst Vej 3A-5A, Copenhagen, DK - 2605 Broendby, Denmark", 1997.
- [11] Rolland F. D., "The Essence of Databases", Prentice Hall International Inc., U.K., 1998.
- [12] Subrahmanyam P., "The Software Engineering of Expert System: Is PROLOG Appropriate? ", IEEE Trans. Software Tamer M. and Valduriez P., "Principles of Distributed Database Systems", 2nd Ed., Printace Hall International, Inc, USA,1999.
- [13] Tamer M. and Valduriez P., "Principles of Distributed Database Systems", 2nd Ed., Printace Hall International, Inc, USA,1999.