International Journal of Advanced Technology in Engineering and Science
Vol. No.5, Issue No. 01, January 2017
www.ijates.com
ijates
ISSN 2348 - 7550

JOIN INDICES FOR RELATIONAL DATABASE MANAGEMENT SYSTEM

P.R.Thirumagal[1] , P.Rizwan Ahmed[2]

[1]Assistant Professor of Computer Science, Mazharul Uloom College, Ambur (India)

[2]Assistant Professor & HOD of BCA & M.Sc. (IT), Mazharul Uloom College, Ambur (India)

ABSTRACT

In new application areas of relational database systems, the join operator is used more than in conventional applications. We propose a simple data structure called join index, for improving the performance of joins in the context of complex queries. For most of the joins, updates to join indices incur very little overhead.

Some properties of join index are

i) Its efficient use of memory and adaptiveness to parallel execution,

ii) Its compatibility with other operations (including select and union),

ii) Its support for abstract data type predicates,

iv) Its support for multirelation clustering, and

v) Its use in representing directed graph and in evaluating recursive queries

Relational database technology is widely accepted as a basic support technology for evolving application areas like CAD/CAM. For this relational technology must be extended to fulfill the requirements of new applications. Compared to business applications, they consist of large number of more complex queries. Efficient query processing becomes a more difficult problem since complex queries is the optimization of all primitive operations and the compatibility of the optimization.

Keywords: Joins, Index, B-Trees, Hashing

I INTRODUCTION

This problem proposes a simple solution for optimizing joins in the context of complex queries. Optimizing the joins is to prejoin all relations by storing each domain separately where each domain value associates the list of identifiers of matching tuples. This storage model favors but it is expensive.

This solution is based on two design principles.

1. An algorithm's performance is proportional to the amount of useful information. So, we strive to make the size of useful information for query processing as small as possible.

2. Future computers will have a parallel processing capability and large amounts of random-access memory (RAM).

The application of these design principles to the problem of complex query evaluation leads us to the notion of join indices. A join index is a particular implementation of the concept of link. It is a prejoined relation smaller than the joined relation, and is stored separately from the operand relations.
This problem proposes an efficient implementation of join indices, gives algorithms for joins and updates, shows the compatibility if join indices with inverted indices and their value is answering complex queries. And finally shows the superior performance of the proposed implementation through a detailed analysis.

II BACKGROUND OF THE PROBLEM

Relational database technology is widely accepted as a basic support technology for evolving application areas like artificial intelligence, CAD/CAM and so forth. Relational technology must be extended to fulfill the requirements of these new applications. Compared to conventional applications, they tend to consists of large numbers of more complex operations are used extensively. A first step toward the efficient processing of the complex queries is the optimization of all primitive operations and the compatibility of these optimizations.
We consider the join operation as a paradigm of basic complex operations. Although many join algorithm have been proposed they are generally designed independently of the effect on other operations in the global query optimization. With new database applications, they effect can be important. For example, a transitive closure operator, which we expects to be a primitive operator for knowledge base support, ill use joins and unions repeatedly. Thus the join operation must be optimized for repetitive use. Also the combination of join and union be efficient.

Indices on join attributes can be employed. Clustered indices on the join attribute yield excellent performance, but at the most one clustered index can be defined on a relation. Another recent and important results show that the availability of large main memories in database systems makes hash-based algorithms much more efficient than the sort merge join algorithm. Consequently, e deduces that hash-based algorithms should be more efficient than algorithms using inverted indices.

A different concept useful for joins is the link. The concept is the same as the Conference on Data Systems Languages (CODAYL) set notion. Tsichritzia uses links in a model that allows the coexistence of hierarchical, network, and relation models.

The thesis proposes an efficient implementation of join indices, gives algorithm for joins and updates, shows the compatibility of join indices with inverted indices and their value in answering complex queries, and, finally, shows the superior performance of the proposed implementation through a detailed analysis. We believe that join indices represent a carefully designed accelerator that can effectively use large RAM to increase performance and constitute

International Journal of Advanced Technology in Engineering and Science
Vol. No.5, Issue No. 01, January 2017
www.ijates.com
ijates
ISSN 2348 - 7550

an interesting alternative to hashing. Also, and perhaps more important, join indices optimize recursive as well as traditional complex queries.

Join Indices

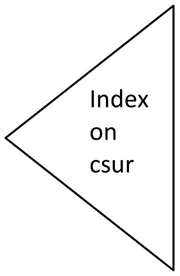
A join index is a binary relation. It only contains pair of surrogates which makes it small. A join index must be clustered. Since we may need fast access to JI tuples via either r values or s values depending on whether there are selects on relations R or S, a JI should be clustered on (r , s).

Concept

We need fast access to JI tuples via either r values or s values depending on whether there are selects on relations R or S, a JI should be clustered on (r, s).A simple and uniform solution is to maintain two copies of the JI, one clustered on r and the other clustered on s. Simplicity and uniformity will lead to increased performance.

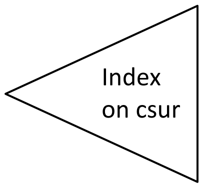
The JI clustered on r makes join accesses from R to S and from S to R efficient. For limited access patterns, a single copy is sufficient. Figure shows the implementation of the join index and of the relations for the join index where the copy clustered on surrogate csur, cpsur is named JI$_{csur}$ and JI$_{cpsur.}$

CUSTOMER



Csur	Cname	City	Age	Job
1	Sony	Boston	21	Clerk
2	cathrine	Austin	26	Secretary
3	Rosy	Austin	36	Manager

JI$_{csur}$

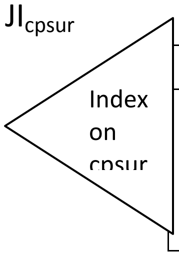


csur	cpsur
1	2
1	3
3	1

CP

Csur	Cname	Pname	Qty	Date
2	Sony	jeans	2	05/05/85
3	Sony	shirt	4	05/25/85
1	Rosy	jacket	3	07/23/96

JI_{cpsur}



csur	cpsur
1	3
2	1
3	1

III JOIN INDEX CLUSTERED TABLE

The above figure shows the use of the join index on relations R(r, A, B) and S(s, C, D) with join predicate (R.A = S.D) to process a query for which the qualification is (R.B = “b” and R.A = S.D). For each tuple of R satisfying the selection predicate (R.B = “b”), its surrogate r_i permits accessing the join index clustered on r. The surrogate s_j associated ith r_i is in turn used to access the matching tuples of S through the index on s.

Surrogates contained in a join index are used for retrieving attribute values in physical relations. It may be implemented by either a clustered index or an inverted index called secondary index.

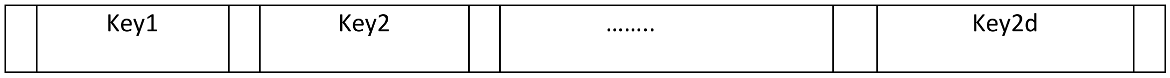
In first case, tuples having adjacent surrogate values ill be the same pages. Since the join index is clustered on surrogates, clustered access to the physical relations makes joins efficient. In the second case, tuples having surrogate values that are close ill seldom be physically close. Thus, random access to tuples is necessary; which makes joins of many tuples less efficient.

B-tree

B-trees have become, de facto, a standard for file organization. File indexes of users, dedicated database systems, and general-purpose access methods have all been proposed and implemented using B- trees. At Sperry Univac Corporation H. Chiat, M. Schartz, and others developed and implemented a system which carried out insert and find operations in a manner related to the B-tree method.

A node in a B-tree of order d with 2d keys and 2d+1 pointers

ijates
ISSN 2348 - 7550

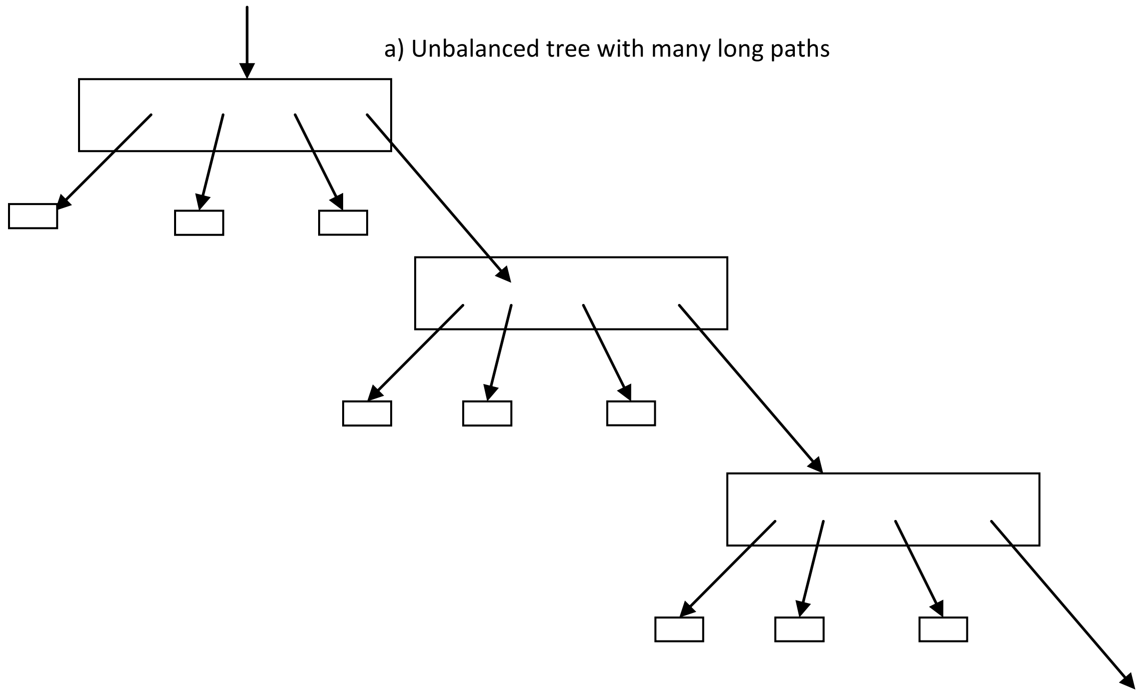


As show in the above figure each node in a B-tree of order d contains at most 2d keys and 2d+1 pointers. Each node is ½ full. Usually, large, multikey modes cannot be kept in memory and require an access to secondary storage each time they inspected.

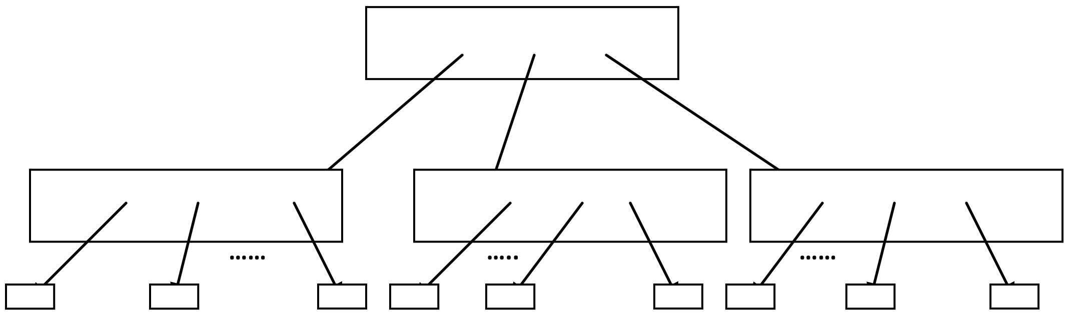
3.1 Balancing

The B-tree lies in the methods for inserting and deleting records that always leave the tree balanced. As in the case of binary search trees, random insertions of records into a file can leave an unbalanced.

a) Unbalanced tree with many long paths



b) A Balanced tree with all paths to leaves exactly same length.



As shown in the above figure an unbalanced tree, has some long paths and some short ones. But for a balanced tree has all leaves at the same depth. The longest path in a B-tree of n keys contains at most about $\log_d$ n nodes, d being the order of the B-tree. A find operation may visit n nodes in an unbalanced tree indexing a file of n records, but it never visits more than $1 + \log_d n$ in a B-tree of order d for such a file.

IV PROPERTIES OF JOIN INDICES

There are several properties of join indices. All these properties result from the fact that a join index is stored in a simple and separate data structure.

Algorithms Exploiting hardware Availability

Future computers have large amounts of RAM and a parallel processing capability. The adaptation of the join algorithm to parallel execution is easy. The join index can be divided into independent subsets, each being carried out by a different processing unit. A page of the external relation is not read by more than one processor. The same page of the internal relation might be read by several processors.

Compatibility with other Operations

The real value of join indices becomes apparent when they are combined with other indices fro processing relational queries. A complex relational query is divided into two steps. The first step applies the query to indices producing an abstract of the result. The second steps, the base data that satisfy the query are accessed.

ADT Join Predicate

By storing separately the relationships existing between tuples, join indices can support ADT join predicates. For example, if a query is to list the information about customers of Austin and parts, such that customer was twenty years old
C.City = “Austin” and C.age – 20 = P.age
The update overhead incurred with ADT join predicates is significantly greater than equi-join predicates.

IV CONCLUSION

This problem proposes a simple solution for optimizing joins in the context of complex queries. Optimizing the joins is to prejoin all relations by storing each domain separately where each domain value associates the list of identifiers of matching tuples. This storage model favors but it is expensive.

ijates
ISSN 2348 - 7550

REFERENCE

[1] R. H. Guting, "GraphDB: modeling and querying graphs in databases," In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), p. 297–308, 1994.

[2] M. GRAVES , E. R. BERGEMAN and C. B. LAWRENCE, "Graph database systems for genomics," IEEE Eng. Medicine Biol. Special issue on Managing Data for the Human Genome Project 11, 6., 1995.

[3] R. Angles and C. Gutierrez, "Survey of graph database models," ACM Comput. Surv., Vol. 40, No. 1, pp. pp. 1-39, 2008.

[4] K. N. Satone, "Modern Graph Databases Models," International Journal of Engineering Research and Applications , 2014. [

[5] 5] P. T. Wood, "Query languages for graph databases," SIGMOD Rec Vol. 41, No. 1, pp. pp. 50-60, 2012.

[6] [6] P. Macko, D. W. Margo and M. I. Seltzer, "Performance introspection of graph databases," in 6th Annual International Systems and Storage Conference, Haifa, Israel , June 30 - July 02, 2013.

[7] P. Jadhav and R. Oberoi, "Comparative Analysis of Graph Database Models using Classification and Clustering by using Weka Tool," International Journal of Advanced Research in Computer Science and Software Engineering, pp. 438-445, Volume 5, Issue 2, February 2015.

[8] H. R. Vyawahare and P. P. Karde, "An Overview on Graph Database Model," International Journal of Innovative Research in Computer and Communication Engineering, Vol. 3, Issue 8, August 2015.

[9] J. Reutter, Graph Patterns: Structure, Query Answering and Applications in Schema Mappings and Formal Language Theory, PhD. Dissertation, University of Edinburgh, 2013.