TR-83-006

Query Optimization in a Database Machine

by

Lin, C. C. & Hong, Y. C.
Institute of Information Science
Academia Sinica
Taipei, Taiwan
Republic of China

I. Introduction

There are three well-known database models today, namely the relational model, the hierarchical model and the network model. The relational model had earned much attentions these years for its simplicity and clarity.

Nevertheless, how to implement a relational database efficiently is still a challenging problem up to date. In programming languages, the compiler has to perform the optimization procedure before code generation. Similarly, in database query languages, optimization of query is still necessary. Because the user is deliberately made unaware of the actual data storage mechanism, he may write queries which, though consistent with his relational view and accurate, have a very low efficiency factor. The DBMS must optimize user's query before execute it, in order to achieve a better performance.

Most researches of query optimization concern with DBMS on conventional computers. When applied on a special database machine, some changes have to be made. The principles of optimization rules may be different depending on the hardware structure and algorithms used by the database machine.

This paper describes some concepts of query optimization and how to implement it on a database machine proposed by our earlier works.

II. General Schemes

The basic organization of a query optimization system is shown in Fig. 1, which is proposed by Smith and Chang [1].

The query is first processed to produce a "tree" representation, which is then passed to the tree transformer. The tree transformeer has access to a set of correctness-preserving algebraic transformations, and to a set of rules determine when the application of these transformations will increase efficiency. The transformer optimizes the tree and passes it along to a mechanism which constructs an implementations of each operator as a task. The operator constructor has access to a set of basic implementation procedures. The constructor creates tasks from these procedures, in such a way that the performance of the whole tree of cooperating tasks is optimized. Finally, the set of cooperating tasks is executed on a database machine.

III. Prosposed Architectures and Algorithms

The description of the proposed hardware architecture is omitted here.

The relational operators which had been investigated in our system is as follows :

(1) SELECT

(2) implicit JOIN

(3) explicit JOIN 1 -- which satisfies the referential integrity constraint

(4) explicit JOIN 2 -- other explicit joins

(5) PROJECT 1 -- the PROJECT columns contain the candidate key

(6) PROJECT 2 -- the PROJECT columns do not contain the candidate key, but the length of it is shorter than the address lines of RAM

(7) PROJECT 3 -- other PROJECTs

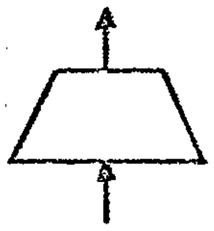
These algorithms are the basic procedures accessed by the coordinating operator constructor in Fig. 1.

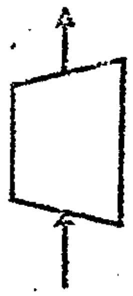
IV. The Optimizer in Our System

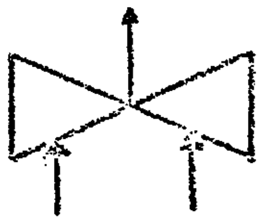
In our system, the query language is supposed to be Relational Algebra. The optimizer model will follow the general scheme described in section II.

In this section, we will discuss the tree transformer and the coordinating operator constructors in more detail. The difference of them on our system with that on conventional computers will be described.

To make the examples below easier to understand, we follow the notations used by [1], as follows :


----- PROJECT


----- SELECT


----- JOIN

<1> Tree Transformer

There are two types of tree transformation proposed by [1]. The frist one moves unary operators down trees. The other type involves replacing a subtree of set operations on the relations and/or restrictions of the same relation by a compound boolean operation.

The second type may be represented by an example shown in Fig. 2. The purpose of this transformation is to speed up the SELECT operator, as that the common relation is never read more than once.

The SELECT operation in our system is performed by the Cellular Logic. The application of this transformation is then determined by the characteristics of the Cellular Logic. For example, if the Cellular Logic used is CASSM, the transformation is no good because CASSM can process only one boolean operation at a time, thus the performance of Fig. 2(a) and Fig. 2(b) will be the same effectively. As another example, if RAP is used as the Cellular Logic, the performance of Fig. 2(b) will be much better than Fig. 2(a) then. The transformation should be applied whenever is possible.

Transformation of the other type, which move unary operators down operator trees, is shown in Fig. 3 by some examples. Similar rules may be found in [2].

The objectives of these tree transformations are (1) to decrease, as low on the tree as possible, the net area of relations passed to higher operators; (2) to group all RESTRICT and SELECT operators on a common relation together; and (3) to allow maximum advantages to be taken of existing directories for stored relations.

While in our system, directories is not used, so the rules of applying transformations should be modified. In addition, moving unary operators down trees is not always desired in our system. As stated in section III, there are three PROJECT algorithms in our system, the last one is the slowest. It's clear that if the PROJECT in Fig. 3(c) is type 3, then the transformation should not be applied.

<2> Coordinating Operator Constructor

The basic procedures in our system are listed in section III. The coordinating operator constructor takes a two-pass method. The first pass is a down-up pass, each node in the operator tree reports to its upper node all the alternatives it can support. And in the second pass, the up-down pass, the upper node chooses one of the alternative procedures supported by the lower node on a way that the overall performance will be optimized.

To achieve the optimized overall performance, we could apply some kind of performance-measurement formulas. For example, we can specify the time needed and space occupied by temporary results for each basic procedures. To choose best assignment of coordinating operators constructor, we can simply calculate the perofrmance measurement of each assignment, and then choose the best one.

V. Other Approaches

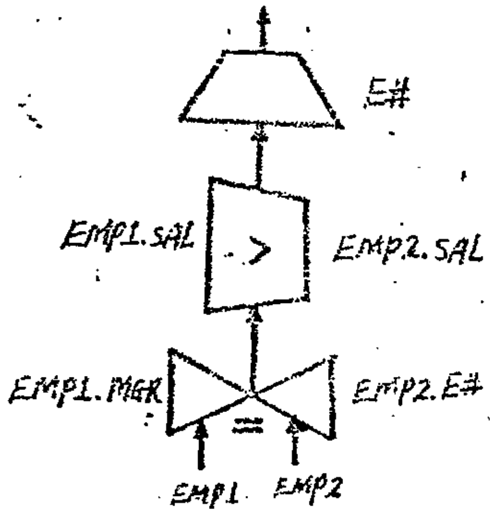
In addition to the basic organizations of optimizer described above, some possible approaches are worth mentioning here.

<1> Currently, our optimizer concerns only three Relational Algebra operators, i.e. SELECT, PROJECT and JOIN. It can not support the full Relational Algebra operators. In [3], it is shown that projection, join, union, difference is a functionally complete subset of the Relational Algebra operators. If we can support these four operators in our system, and optimize the query which contains these operators only, we can then optimize a full Relational Algebra query language.

<2> In [4], a relational expression which contains only SELECT, PROJECT and JOIN (called SPJ expression) is transformed into a talbe representation. Some transformation rules are derived upon the table, and the corresponding expression of the result table can be proved to be optimized. This approach seems attractive since it is supported by mathematical theories.

<3> Semantics model

Suppose we have a relation EMP(E#,SAL,MGR). We want to find out all the employees who earn more than his manager. A possible solution will be



In our system, this query can be solved by a much better method by using of the pointer/counter field in RAM. The optimzed tree will be different with the original tree in a way that no simple transformation can be applied as described on previous sections. The optimizer have to understand the semantic of the query and apply some complicated transformations. This approach is not easy for the semantics model is difficult to be made. But this approach is surely important in order to achieve a promising query optimizer.

VI. References

(1) J. M. Smith and P. Y. Chang, "Optimizing the Performance of a Relational Algebra Database Interface", CACM, Oct. 1975.

(2) J. D. Ullman, "Principles of Database Systems", Chapter 6.

(3) L. L. Beck, "A Generalized Implementation Method for Relational Data Sublanguages", IEEE Trans. on Software Engineering, Mar. 1980.

(4) A. V. Aho, Y. Sagiv and J. D. Ullman, "Efficient Optimization of a Class of Relational Expressions".

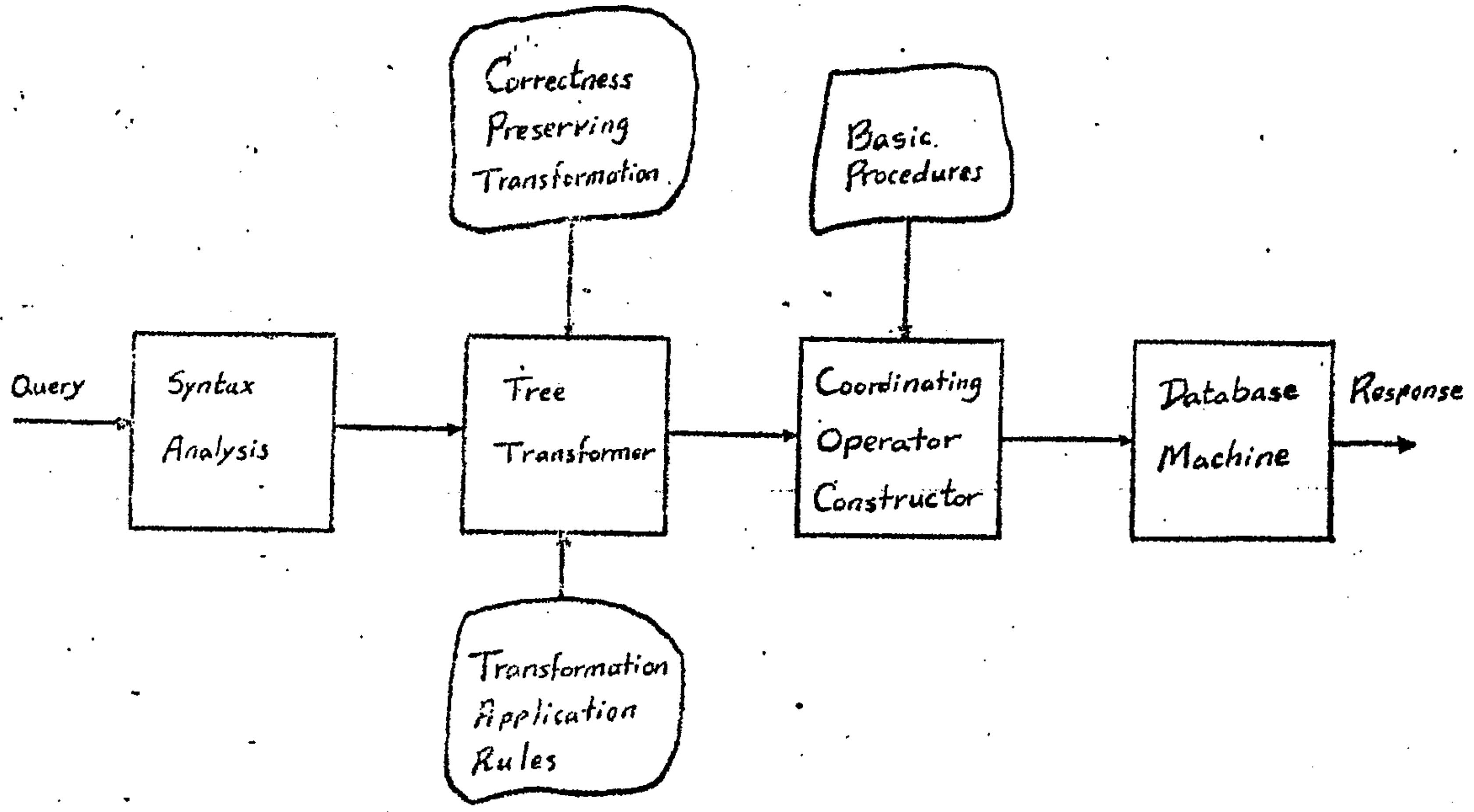
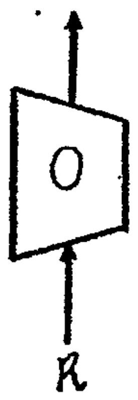
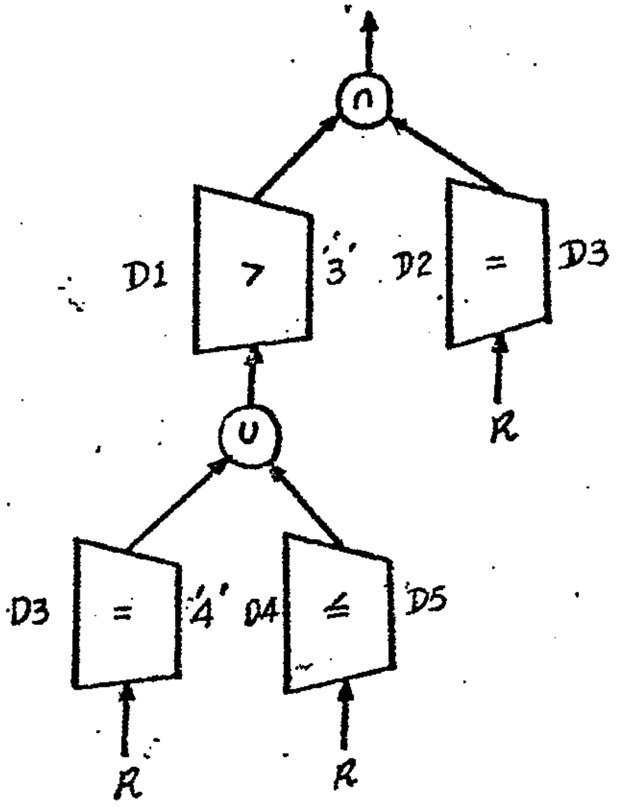

Correctness Preserving Transformation
Basic Procedures
Query
Syntax Analysis
Tree Transformer
Coordinating Operator Constructor
Database Machine
Response
Transformation Application Rules

Figure 1.



$O \equiv [D1 > 3 \text{ and } (D3 = 4 \text{ or } D4 \leq D5) \text{ and } D2 = D3]$

(b)



(a)

Figure 2.

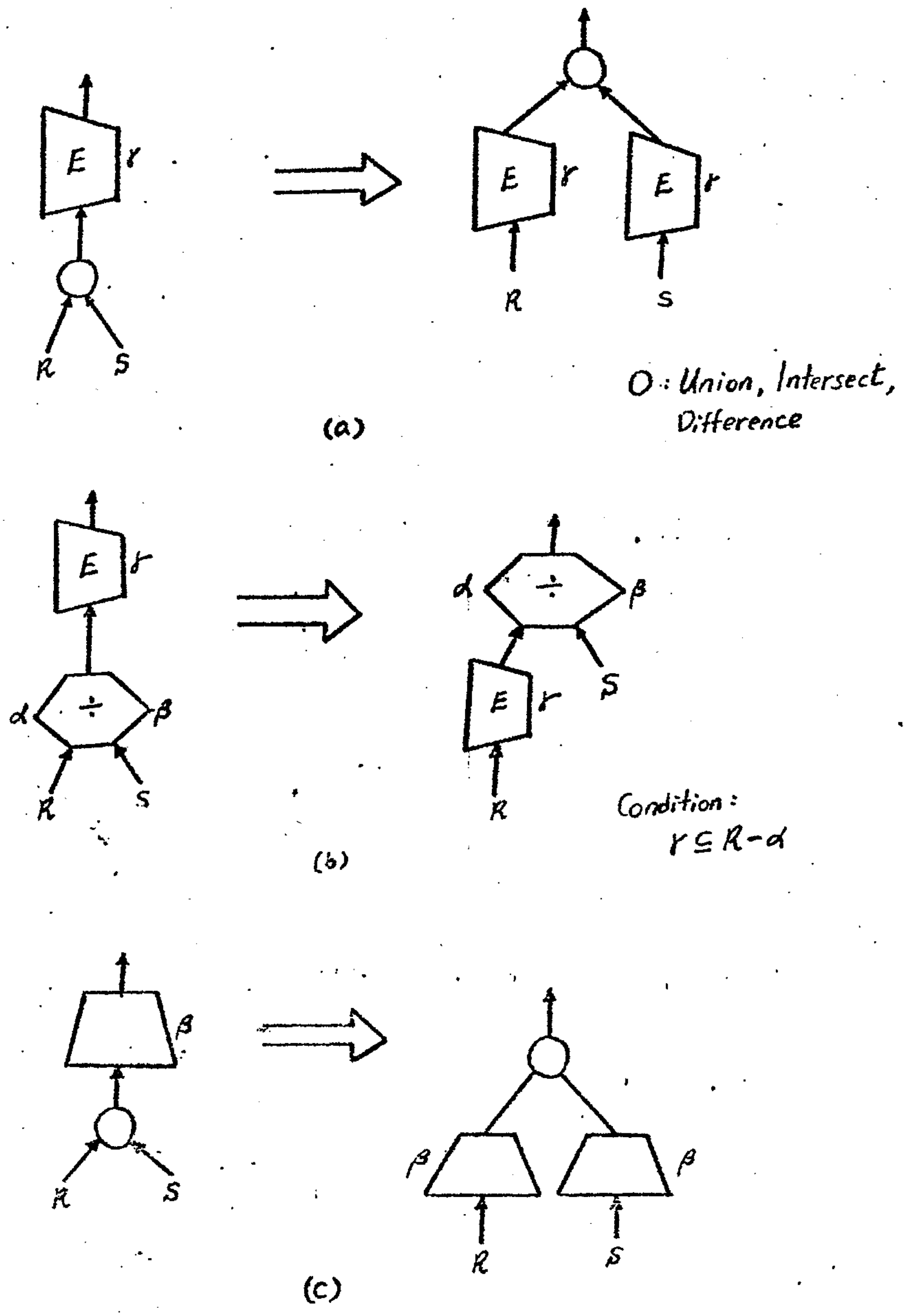

E
γ
R
S
O : Union, Intersect, Difference
(a)
E
γ
α
÷
β
R
S
Condition:
γ ⊆ R − α
(b)
β
R
S
(c)

Figure 3.