View Maintenance for Materialized Transitive-Closure Relations

Ismail H. Toroslu
METU

View maintenance for intentional relations is one of the most active research topics in deductive databases. The simplest, but the most frequently found form of the recursive relations in deductive databases is transitive-closure relations. Materialized transitive closure view definitions and maintenance of these views under updates on the base relations, is one of the techniques for the view maintenance of transitive closure relations. In this paper a special data structure is presented that represents transitive closure relation in compressed form. Update operations on this structure are defined and their algorithms are given. Unlike previous proposals, the technique presented in this paper treats cyclic and acyclic relations in the same way without requiring the determination of cycles after the update operations. The materialized form of the transitive closure relation is also suitable for different forms of transitive closure queries.

Recently, research on database systems has concentrated on extending the power of conventional relational databases to create intelligent database systems that can solve more complex real world problems easily. One of the main extensions to relational databases is on handling recursive queries in database systems. It is not possible to define recursive views and asking queries directly on these views in relational database systems. In traditional relational DBMSs, recursive queries can be solved by developing programs with general purpose programming languages embedded into the DBMSs. Unlike conventional relational databases, in deductive databases users can make recursive view definitions and define queries on them directly.

In deductive database systems in general, any kind of recursive views can be defined. However, most of them are not very practical and do not appear in real life problems. Also there are not very efficient algorithms for every kind of recursive queries (Bancilhon, Ramakrishnan, 1986). Therefore, most of the research on this topic is dedicated to the more practical recursive queries. Transitive closure relations are the simplest, but also the most frequently found forms of recursive definitions. There is a lot of recent research on efficient solutions of queries on transitive closure relations (Agrawal, Dar, Jagadish, 1990; Agrawal, Jagadish, 1990; Dar, Jagadish, 1992; Dar, Ramakrishnan, 1994; Han, Qadah, Chaou, 1988; Ioannidis, Ramakrishnan, 1988; Ioannidis, Ramakrishnan, Winger, 1993; Ioannidis, 1986; Jakobsson, 1992; Jiang, 1990; Qadah, Henschen, Kim, 1991; Toroslu, Qadah, 1992; Toroslu, Qadah, 1993; Toroslu, Qadah, Henschen, 1993; Toroslu, Qadah, 1996; Ullman, Yannakakis, 1990).

Also, even though the view maintenance of the recursive relations is a very active research topic, there are only a very few works dedicated specifically to materializing transitive closure relations (Agrawal, Borgida, Jagadish, 1989; Agrawal, Kiernan, 1993; Guh, Yu 1992; Jagadish, 1990; Larson, Deshpande, 1986).

Simply, the idea in the materialized transitive closure is to precompute the transitive closure relation even though it is defined as recursive IDB relation, and compress it. In this way queries on transitive closure relation will be answered very efficiently without recomputing the transitive closure, or part
of it each time such a query is issued. The materialized transitive closure relation should be suitable for all kinds of transitive closure queries, namely, fully instantiated, partially instantiated or uninstantiated. In addition to that, updates on the base relation, on which the transitive closure relation is defined, should be able to be reflected into the materialized transitive closure relation. Otherwise, whenever an update is performed on the base relation, the materialized transitive closure should be recomputed which will be very inefficient for dynamic databases. Also, because it is not possible to store the whole transitive closure relation due to its enormous size (compared to the size of the base relation), new data structures must be developed to store the transitive closure relation in compressed form.

In this paper a new data structure for materialized transitive closure is proposed, whose main difference from the previous proposals is its uniform treatment to both cyclic and acyclic relations.

The second section summarizes the basic concepts and problems related to the transitive closure relations. The third section describes some of the previous works for materialized transitive closures.

The fourth section describes the new data structure and algorithms on how the updates on base relation and queries on the transitive closure relation are handled. The fifth section discusses the performance of the new technique using simulations and the last section is the conclusion.

**Transitive Closure Relation**

In general, a logic rule is *linearly recursive* if the rule’s head predicate appears only once in its body. A TC rule is a linearly recursive rule of the following form:

\[ R(X,Y) :- A(X,Z), R(Z,Y) \]

where \( A(X,Z) \) is an extensional (base) predicate. Within the context of deductive databases, \( A(X,Z) \) is defined by a two-attribute normalized database relation with very many tuples as shown in Figure 1a.

Another common view for the base relation is the one shown in Figure 1b, where the base relation is represented as a directed (tree, acyclic or cyclic) graph with the possibility of having more than one component.

The nodes in such a graph are the set of distinct values in the two columns of the base relation. For every tuple \( <x,y> \) of the base relation, there exists, in the corresponding digraph, a directed edge from node \( x \) to node \( y \).

To generate solutions using recursive rule (1), another non-recursive rule, the *exit rule*, which defines the predicate \( R(X,Y) \) must exist. For example, the following rule is an exit rule for \( R \):

\[ R(X,Y) :- A(X,Y) \]

A TC query is a headless rule whose predicate is defined by a TC rule and its associated exit rule. For example,

\[ :- R(X,Y) \]

is a TC query.

In general, a 2-place unit query, such as (3), may have different forms depending on the instantiation status of the query’s variables (Qadah, Henschen, Kim, 1991). Most recursive queries are either in these forms, or can be converted to these forms and can, therefore, be evaluated using transitive closure operators (Ullman, 1989).

The first form of the recursive query, when both variables of a query are uninstantiated (\( :- R(X,Y) \)), can be converted to the computation of the *full transitive closure* (FTC) of the relation graph (Qadah, Henschen, Kim, 1991). We can think of FTC as the set of tuples showing all the nodes that are reachable from each node in the graph (i.e., start and end nodes of the all paths in the graph).

In another form of transitive closure queries only one of the variables of the recursive query is instantiated (\( :- R(X,c) \) or \( :- R(c,Y) \)). In this case the problem turns into the computation