An algebraic approach to recursive queries in relational databases

N. Spyratos, C. Lecluse

INRIA
CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P 105
78153 Le Chesnay Cedex
France
Tel. (1) 39 63 55 11

Rapports de Recherche

N° 499

AN ALGEBRAIC APPROACH TO RECURSIVE QUERIES IN RELATIONAL DATABASES

Nicolas SPYRATOS
Christophe LECLUSE

Mars 1986

AN ALGEBRAIC APPROACH TO RECURSIVE QUERIES IN RELATIONAL DATABASES

Nicolas SPYRATOS
INRIA
Rocquencourt - BP 105
Domaine de Voluceau
78153 Le Chesnay Cédex
France

Nicolas SPYRATOS
Université Paris-Sud
LRI - Bât. 490
91405 Orsay Cédex
France

Christophe LECLUSE
Université Paris-Sud
LRI - Bât. 490
91405 Orsay Cédex
France

ABSTRACT: We propose an algebraic approach to recursive queries in relational databases through an algebra based on the product space of all attribute domains. The basic operations of the algebra are set-theoretic difference, intersection and union, projections and their inverses, and function composition. Using our approach, we discuss the specification of two important types of recursive queries: ancestors and cousins. We also discuss the evaluation of recursive queries and we give an optimal algorithm for computing ancestors. Furthermore, we show that functional dependencies can be profitably used to improve performance, during the evaluation of recursive queries.

Résumé: Nous présentons une approche algébrique du problème des requètes récursives dans les bases de données relationelles. Nous utilisons une algèbre basée sur le produit cartésien de tous les domaines des attributs. Les opérations de base de cette algèbre sont les opérations ensemblistes (union, intersection, difference), les projections et leurs inverses ainsi que la composition des fonctions. Nous utilisons cette approche pour spécifier deux exemples importants de requètes récursives: les "ancètres" et les "cousins". Nous étudions également l'évaluation des requètes récursives et nous donnons un algorithme optimal pour le calcul des ancètres. Finalement, nous montrons que les dépendances fonctionelles peuvent etre utilisées avec profit pour rendre l'évaluation plus performante.

1. INTRODUCTION

In the past few years major attempts have been made to improve the power of database systems, in particular those based on the relational model (see [C70]). A significant part of this effort has been in development of deductive databases. As defined in [G84], "a deductive database is a database in which new facts may be derived from facts that were explicitly introduced". A very important difference between a deductive and a conventional relational database is that in the former, new facts may be derived recursively. This very characteristic of deductive databases is what makes query processing a difficult task in such an environment.

The problem of evaluating recursively defined queries has gained considerable attention in the recent litterature [HN84,U85,B85 ...]. The specification of a recursive query is usually done in a logic query language, such as the one described in [U85], whereas the evaluation is usually done by transforming the logic query into a relational algebra program [B85]. The reason for this dichotomy is that logic provides for non-procedural specification, whereas relational algebra provides for procedural evaluation. However, there is a price to pay: translation from logic to algebra can be difficult [U85,B85b].

We propose a purely algebraic approach whereby both the specification and the evaluation of recursive queries is done within the same language. The basic operations of the language are set-theoretic difference, intersection and union, projections and their inverses, and function composition. Recursive queries are specified by well-formed expressions of the language, and they are evaluated by relational algebra programs. As relational join and selection are easily expressed in the proposed language, the translation of recursive queries into relational algebra programs is straightforward.

The paper is organized as follows. Section 2 outlines the definition of the language and illustrates the basic concepts by way of examples. It is also shown how the relational algebra can be embedded in the proposed language. In Section 3 we discuss the specification of two important

examples of recursive queries that have received extensive attention in the literature: the ancestors and the cousins. In Section 4 we discuss query evaluation. More specifically, we discuss query modification and we give an optimal algorithm for computing ancestors. Furthermore, we show that functional dependencies can be profitably used to improve performance when evaluating recursive queries. Finally, Section 5 contains some concluding remarks and suggestions for further research.

2. THE PROJECTION LANGUAGE

We assume familiarity with the basic relational terminology (as in [U83] or [M83]). Let U be a universe of n attributes $A_1, A_2, \ldots, A_n$, and let D_i=Dom(A_i) be the domain associated with the attribute A_i, i=1,2,...,n. We call relation scheme any subset R of the universe U. Frequently, a relation scheme is denoted by the juxtaposition of its attributes. For example, one writes $U=A_1A_2A_3$ or $R=A_1A_2$ instead of U={A_1,A_2,A_3} and R={A_1,A_2}, respectively.

Let T be the set of all universal tuples, that is, let $T = D_1 \times D_2 \times \cdots \times D_n$. We denote the i-th projection function on T by p_i. Recall that p_i goes from T onto D_i and that it associates each tuple t in T with the A_i-value of t. A projection on two or more attributes, say A_i and A_j, denoted by p_{ij}, is defined similarly, and it goes from T onto $D_i \times D_j$. Relational projections and their inverses constitute the basic building blocks of our specification language.

For any set E, let P(E)={x / x $\subset$ E} be the set of all subset of E. Thus, for i=1,2,...,n, an element of P(D_i) is a set of A_i-values. Similarly, an element of P(T) is a universal relation.For technical reasons, we need to extend the projections p_i and their inverses as follows (see also Figure 1) :

Definition 1 For all i=1,2,...,n, define the following functions :

$\pi_i : P(T) \to P(D_i)$ such that $\forall u \in P(T),\ \pi_i(u) = \{p_i(t) / t \in u\}$

$\pi_i^{-1} : P(D_i) \to P(T)$ such that $\forall x \in P(D_i),\ \pi_i^{-1}(x) = \{t \in T / p_i(t) \in x\}$ □

Functions of the form π_{ij} and π_{ij}^{-1}, involving two (or more) attributes can be defined in a similar manner. We recall here some simple facts about cartesian products and projections ,whose proofs are omitted as they are straightforward (see for example the introductory chapter in [D66]).

Proposition 1 : For all i=1,2,...,n, and for all j=1,2,...,n, we have

$(a)\ \forall r \in P(D_i),\ \pi_i^{-1}(r) = D_1 \times \cdots \times D_{i-1} \times r \times D_{i+1} \times \cdots \times D_n$

$(b)\ \forall r \in P(D_i \times D_j),\ \forall s \in P(D_j \times D_k),$

$\pi_{ij}^{-1}(r) = r \times D_1 \times \cdots \times D_{i-1} \times D_{i+1} \times \cdots \times D_{j-1} \times D_{j+1} \times \cdots \times D_n$

and $r \times s = \pi_{ij}^{-1}(r) \cap \pi_{jk}^{-1}(s)$

$(c)\ \forall r \in P(T),\ r \subset \pi_i^{-1}\pi_i(r),$ and $\pi_i\pi_i^{-1}(r) = r.$

$(d)\ \forall r \in P(T), \forall s \in P(T),$ if $r \subset s$ then $\pi_i(r) \subset \pi_i(s)$

$(e)\ \forall r \in P(D_i),\ \forall s \in P(D_i),$ if $r \subset s$ then $\pi_i^{-1}(r) \subset \pi_i^{-1}(s)$

$(f)\ \forall r_i \in P(D_i),\ r_1 \times r_2 \times \cdots \times r_n = \pi_1^{-1}(r_1) \cap \cdots \cap \pi_n^{-1}(r_n)$

$(g)\ \forall r \in P(D_i),\ \forall s \in P(D_i),\ \pi_i^{-1}(r \cup s) = \pi_i^{-1}(r) \cup \pi_i^{-1}(s)$ and

$\pi_i^{-1}(r - s) = \pi_i^{-1}(r) - \pi_i^{-1}(s)$ and

$\pi_i(r \cup s) = \pi_i(r) \cup \pi_i(s)$ and

$\pi_i(r - s) \supset \pi_i(r) - \pi_i(s)$ □

Let us now see an example that we shall use as a running example.

Example 1.

Let U be a universe of three attributes $A_1, A_2, and\ A_3$, with the following domains

$D_1 = Dom(A_1) = \{a, a'\}$

$D_2 = Dom(A_2) = \{b,b',b''\}$

$D_3 = Dom(A_3) = \{c,c'\}$

Consider the following cartesian products :

$D_1 \times D_2 = \{ab, ab', ab'', a'b, a'b', a'b''\}$

$D_2 \times D_3 = \{bc, bc', b'c, b'c', b''c, b''c'\}$

$T = D_1 \times D_2 \times D_3 = \{abc, abc', ab'c, ab'c', ab''c, ab''c',$
$a'bc, a'bc', a'b'c, a'b'c', a'b''c, a'b''c'\}$

Let us apply Definition 1 to some examples :

for r={a} in $P(D_1)$, we find (see Proposition 1a):

$\pi_1^{-1}(r) = r \times D_2 \times D_3 = \{abc, abc', ab'c, ab'c', ab''c, ab''c'\}$

for r={b,b'} in $P(D_2)$ we find (see Proposition 1a):

$\pi_2^{-1}(r) = D_1 \times r \times D_3 = \{abc, abc', a'bc, a'bc', ab'c, ab'c', a'b'c, a'b'c'\}$

for r={ab,a'b} in $P(D_1 \times D_2)$ we find (see proposition 1b):

$\pi_{12}^{-1}(r) = r \times D_3 = \{abc, abc', a'bc, a'bc'\}$

for r={bc,bc'} in $P(D_2 \times D_3)$ we find :

$\pi_{23}^{-1}(r) = D_1 \times r = \{abc, abc', a'bc, a'bc'\}$ ▫

The specification language that we propose, called "projection language", or PL for short, consists of all expressions that can be (well) formed using the following operations : set-theoretic difference, intersection, union, projections and their inverses, and function composition. Function composition is denoted here by juxtaposition of function symbols. Thus in Figure 1, if x is in $P(D_i)$, then $\pi_i\pi_j^{-1}(x)$ means $\pi_i(\pi_j^{-1}(x))$. Here are some examples of PL-expressions:

$\pi_i\pi_j^{-1}(x)$, where $x \in P(D_i)$

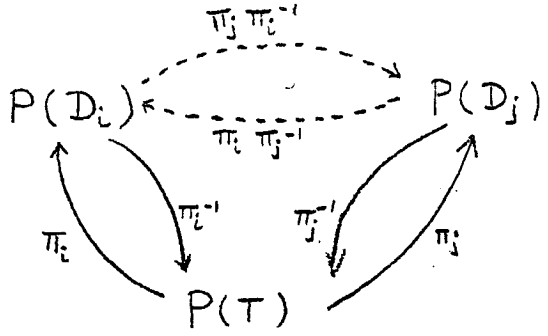
$\pi_i\pi_j^{-1}\pi_j\pi_i^{-1}(x)$, where $x \in P(D_i)$

$\pi_i^{-1}(x) \cap \pi_j^{-1}(y)$, where $x \in P(D_i)$ and $y \in P(D_j)$

The following expressions are not well formed and, therefore, they are not PL-expressions :

$\pi_i\pi_j(x)$, $\pi_i(x) \cap \pi_j^{-1}(y)$, $\pi_i(\pi_i(x) \cup \pi_j(y))$

Figure 1



It is tedious but not difficult to show that join and selection can be expressed in the language PL. We sketch the proof below, where we assume, for simplicity, a universe U of three attributes A_1, A_2, and A_3 (as in Example 1).

JOIN : Let r be a relation over A_1A_2 and let s be a relation over A_2A_3. That is, let $r \in P(D_1 \times D_2)$ and $s \in P(D_2 \times D_3)$. Then we have :

$$r \bowtie s = \{t \in T / \pi_{12}(t) \subset r \wedge \pi_{23}(t) \subset s\}$$
$$= \{t \in T / \pi_{12}(t) \subset r\} \cap \{t \in T / \pi_{23}(t) \subset s\}$$
$$= \pi_{12}^{-1}(r) \cap \pi_{23}^{-1}(s) \quad \square$$

Example 1 (continued)

Consider the following relations :

$r = \{ab, ab'\} \in P(D_1 \times D_2)$

$s = \{bc, b''c\} \in P(D_2 \times D_3)$

Using Proposition 1b, we have:

$\pi_{12}^{-1}(r) = r \times D_3 = \{abc, abc', ab'c, ab'c'\}$

$\pi_{23}^{-1}(s) = D_1 \times s = \{abc, ab''c, a'bc, a'b''c\}$

Therefore, we find :

$\pi_{12}^{-1}(r) \cap \pi_{23}^{-1}(s) = \{abc\} = r \bowtie s \quad \square$

Clearly, if r and s are defined over disjoint sets of attributes then the join becomes a cartesian product.

SELECTION : Let r be a relation over A_1A_2 and assume a selection condition of the form $A_1 = a$. It follows from Definition 1 that:

(i) $\pi_i^{-1}(\{a\})$ is the set of all tuples over $A_1A_2A_3$ that satisfy the given condition and, therefore, that

(ii) $\pi_{12}\pi_1^{-1}(\{a\})$ is the set of all tuples of $D_1 \times D_2$ that satisfy the given

condition.

It follows from (ii) that

(iii) $r \cap \pi_{12}\pi_1^{-1}(\{a\})$ is the set of all tuples of r that satisfy the given condition, and we conclude that

$$r \cap \pi_{12}\pi_1^{-1}(\{a\}) = \{t \in r \;/\; \pi_1(t) = \{a\}\} = \sigma_{A=a}(r) \qquad \square$$

Example 1 (continued)

Consider the relation r = {ab, ab', a'b} and the selection condition A_1=a. It follows from Proposition 1a that:

$$\pi_1^{-1}(\{a\}) = \{a\} \times D_2 \times D_3 = \{abc, abc', ab'c, ab''c, ab''c'\}$$

Applying the projection π_{12} on $\pi_1^{-1}(\{a\})$, we obtain:

$$\pi_{12}\pi_1^{-1}(\{a\}) = \{ab, ab', ab''\}$$

Therefore, we have:

$$\begin{aligned} r \cap \pi_{12}\pi_1^{-1}(\{a\}) &= \{ab, ab', a'b\} \cap \{ab, ab', ab''\} \\ &= \{ab, ab'\} \\ &= \sigma_{A=a}(r) \end{aligned}$$

It is not difficult to see that selection of r following the condition: A_1=a or A_1=a', is expressed by the intersection: $r \cap \pi_{12}\pi_1^{-1}(\{a,a'\})$. More complicated selection conditions may involve the difference operator. For example, the set of tuples of r whose A_2-value is different from b' is expressed by the intersection: $r \cap \pi_{12}\pi_2^{-1}(\pi_2(r) - \{b'\})$, that is

$$\sigma_{A_2 \neq b'}(r) = r \cap \pi_{12}\pi_2^{-1}(\pi_2(r) - \{b'\}) \qquad \square$$

In conclusion, we have seen in this section the language PL and, also, how relational join and selection can be defined as PL-expressions. It follows that the language PL embeds the relational algebra. In the following section we show that the language PL is also suitable for the specification of recursive queries.

3. THE SPECIFICATION OF RECURSIVE QUERIES

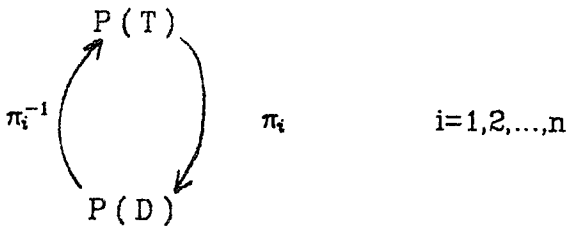
When specifying recursive queries, the underlying assumption is that all attributes (or at least those involved in the query) have the same domain. Thus, instead of the diagram of Figure 1, we have now that of Figure 2 below, where

$D = Dom(A_1) = Dom(A_2) = \cdots = Dom(A_n)$ and

$T = Dom(A_1) \times Dom(A_2) \times \cdots \times Dom(A_n) = D^n$

In order to simplify the presentation, we assume a single universal relation r. Furthermore, we assume that the projections π_i are restricted on the set P(r) (the set of all subset of r). As a consequence, for all s in $P(D_i)$, the relation $\pi_i^{-1}(s)$ is the restriction of r with respect to s (see Section 2).

Figure 2



The basic building blocks for the specification of recursive queries are functions of the form $\pi_i\pi_j^{-1}$, i=1,2,...,n and j=1,2,...,n. It should be clear from Figure 1 that a function of the form $\pi_i\pi_j^{-1}$ goes from $P(D_j)$ into $P(D_i)$. We shall refer to the function $\pi_i\pi_j^{-1}$ as the *ij –mapping*. Under our assumptions, $P(D_j) = P(D_i) = P(D)$ and,thus, every ij-mapping goes from $P(D)$ into $P(D)$. Therefore, every composition of ij-mappings is a (well-formed) expression of the language PL. It is precisely this property of ij-mappings that allows for the specification of recursive queries.

When specifying recursive queries we make use of ij-mappings of order n, that is, mappings of the form $(\pi_i\pi_j^{-1})^n$, defined as follows:

$(\pi_i\pi_j^{-1})^1 = \pi_i\pi_j^{-1}$

$(\pi_i\pi_j^{-1})^n = (\pi_i\pi_j^{-1})(\pi_i\pi_j^{-1})^{n-1}$, n=2,3,...

Loosely speaking, the mapping $(\pi_i\pi_j^{-1})^n$ is obtained by composing $\pi_i\pi_j^{-1}$ with itself n times.

Example 2 :

Let U be a universe of three attributes $A_1A_2A_3$ having as (common) domain the set of all latin characters. Consider a relation r over $A_1A_2A_3$, as shown in Figure 3, and the 21-mapping $\pi_2\pi_1^{-1}$. Let us consider a value of the A_1-column, say the value a, and let us compute $\pi_2\pi_1^{-1}(\{a\})$. We find that:

$$\pi_2\pi_1^{-1}(\{a\}) = \pi_2(\pi_1^{-1}(\{a\})) = \pi_2(\{abc,acb\}) = \{b,c\}$$

What this says is that the value a (of the A_1-column) is "related" to the values b and c (of the A_2-column), or that b and c are the "A_2-relatives" of a. Now, as the values b and c appear also in the A_1-column, we can apply again the 21-mapping $\pi_2\pi_1^{-1}$ on the result of the first computation. We find that:

$$\pi_2\pi_1^{-1}(\pi_2\pi_1^{-1}(\{a\})) = \pi_2\pi_1^{-1}(\{b,c\}) = \pi_2(\{bda, bdc, cea\}) = \{d,e\}$$

What this says is that d and e are the A_2-relatives of a of "order 2". Summarizing our discussion so far, we have that:

$(\pi_2\pi_1^{-1})^1(\{a\}) = \{b,c\}$ are the A_2-relatives of a of order 1

$(\pi_2\pi_1^{-1})^2(\{a\}) = \{d,e\}$ are the A_2-relatives of a of order 2

In general, the function $(\pi_2\pi_1^{-1})^n$ produces A_2-relatives of order n. Let us note that this process can be explained through an undirected graph defined as follows

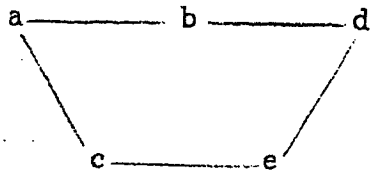
- $\pi_1(r) \cup \pi_2(r)$ is the set of nodes and
- there is an arc between node x and node y iff xy is in $\pi_{12}(r)$.

In Figure 3, we can see the graphical representation of the (binary) relation $\pi_{12}(r)$. It is clear from the graph that b and c are relatives of a of order 1, because there are paths of length 1 going from a to b, and from a to c. Similarly, d and e are relatives of a of order 2, because there are paths of length 2 going from a to d, and from a to e. Let us note that graphical representations are helpful when dealing with only two columns. However, when we deal with three or more columns, graphical representations tend to be cumbersome. □

Figure 3

A relation r over $A_1A_2A_3$ and the graphical representation of binary relation $\pi_{12}(r)$.

A_1	A_2	A_3
a	b	c
a	c	b
b	d	a
b	d	c
c	e	a
d	e	c



Two examples of recursive queries that have received extensive attention in the litterature are "the ancestors" and "the cousins" (see for example in [B85b]). In what follows, we discuss the specification of these two examples in the language PL.

ANCESTORS: Let us begin by stating formally the definition of "relatives", as suggested by our discussion in Example 2.

Definition 2: Let r be a (universal) relation. For all s in P(D) the set of ij-relatives of s of order n (with repect to r), denoted by $R_{ij}^n(s)$, is defined as follows: $R_{ij}^n(s) = (\pi_i\pi_j^{-1})^n(s)$ □

Intuitively, we can think of $R_{ij}^n(s)$ as the set of all "ancestors" of s, n generations backwards. It is important to note that R_{ij}^n is a function that goes from P(D) into P(D).

Example 3.

Referring to Figure 3, let us compute the 21-relatives of s = {a,b} :

Order 1: $R_{21}^1(s) = (\pi_2\pi_1^{-1})^1(s) = \pi_2\pi_1^{-1}(\{a,b\}) = \{b,c,d\}$.

Order 2: $R_{21}^2(s) = (\pi_2\pi_1^{-1})^2(s) = \pi_2\pi_1^{-1}(R_{21}^1(s)) = \pi_2\pi_1^{-1}(b,c,d\}) = \{d,e\}$.

Order 3: $R_{21}^3(s) = (\pi_2\pi_1^{-1})^3(s) = \pi_2\pi_1^{-1}(R_{21}^2(s)) = \pi_2\pi_1^{-1}(\{d,e\}) = \{e\}$.

Order 4: $R_{21}^4(s) = (\pi_2\pi_1^{-1})^4(s) = \pi_2\pi_1^{-1}(R_{21}^2(s)) = \pi_2\pi_1^{-1}(\{e\}) = \phi$.

It follows that : $R_{21}^n = \phi$, for $n \geq 4$ □

The first example of recursive query that we would like to define as a PL-expression is the following:

Q_{ij} : Given a set s in P(D), find all ij-relatives of s (independently of the order).

Let $Q_{ij}(s)$ denote the answer to this query. From what we have said so far, in order to find Q_{ij}(s), it is enough to "accumulate" the ij-relatives of s of order 1,2,..., that is,

$$Q_{ij} = R_{ij}^1 \cup R_{ij}^2 \cup \cdots$$

Let us define

$$Q_{ij}^n(s) = R_{ij}^1(s) \cup R_{ij}^2(s) \cup \cdots \cup R_{ij}^n(s), \qquad n=1,2,... \qquad (1a)$$

Using Definition 2 and proposition 1(g) we can write:

$$\begin{aligned} R_{ij}^2(s) \cup \cdots \cup R_{ij}^n(s) &= R_{ij}^1(R_{ij}^1(s)) \cup \cdots \cup R_{ij}^1(R_{ij}^{n-1}(s)) && \text{[from Definition 3]} \\ &= R_{ij}^1(R_{ij}^1(s) \cup \cdots \cup R_{ij}^{n-1}(s)) && \text{[Proposition 1g]} \\ &= R_{ij}^1(Q_{ij}^{n-1}(s)) && (1b) \end{aligned}$$

Thus we can re-write (1a) as follows:

$$Q_{ij}^n(s) = Q_{ij}^1(s) \cup R_{ij}^1(Q_{ij}^{n-1}(s)), \qquad n=1,2,... \qquad (1c)$$

or, using equations (1a) to (1c),

$$\begin{aligned} Q_{ij}^1(s) &= R_{ij}^1(s) \\ Q_{ij}^n(s) &= R_{ij}^1(s) \cup R_{ij}^1(Q_{ij}^{n-1}(s)), \qquad n=2,3,... && (2a) \\ &= Q_{ij}^{n-1}(s) \cup R_{ij}^1(Q_{ij}^{n-1}(s)), \qquad n=2,3,... && (2b) \end{aligned}$$

For example, referring to Figure 3, we find:

$$Q_{21}^1(\{a\}) = R_{21}^1\{a\}) = \pi_2\pi_1^{-1}(\{a\}) = \{b,c\}$$

$$Q_{21}^2(\{a\}) = Q_{21}^1\{a\}) \cup R_{21}^1(Q_{21}^1(\{a\}))$$

$$= \{b,c\} \cup \pi_2\pi_1^{-1}(\{b,c\}) = \{b,c\} \cup \{d,e\} = \{b,c,d,e\}$$

$$Q_{21}^3(\{a\}) = Q_{21}^2(\{a\}) \cup R_{21}^1(Q_{21}^2(\{a\}))$$

$$= \{b,c\} \cup \pi_2\pi_1^{-1}(\{b,c,d,e\}) = \{b,c\} \cup \{d,e\} = \{b,c,d,e\}$$

$$= Q_{21}^2(\{a\})$$

It follows that: $Q_{21}^n(\{a\}) = Q_{21}^2(\{a\})$, for n=3,4,.... and therefore, the answer to the query is:

$$Q_{21}(\{a\}) = Q_{21}^2(\{a\}) = \{b,c,d,e\}$$

In general, it follows from (1a), that the sequence of sets: $Q_{21}^1(s)$, $Q_{21}^2(s)$,..., is an increasing sequence (with respect to set inclusion). On the other hand, assuming that the given relation r is finite (as it is the case, if r is a database relation), there is a (least) finite integer n_0 such that: $Q_{ij}^{n_0}(s) = Q_{ij}^{n_0+m}(s)$, for all m=1,2,... Therefore, the answer $Q_{ij}(s)$ can be defined as follows:

$$Q_{ij}(s) = Q_{ij}^{n_0}(s),$$
$$\text{where } n_0 = \min(\{n / Q_{ij}^n(s) = Q_{ij}^{n+m},\ m=1,2...\}) \qquad (2c)$$

Another way to look at the definition of $Q_{ij}(s)$ is as the least fixed point of the following equation:

$$Q_{ij}(s) = R_{ij}^1(s) \cup R_{ij}(Q_{ij}(s)) \qquad (3)$$

Equation (3) is an immediate consequence of (2a) and (2c).

COUSINS: Let us begin by stating formally the definition of a "cousin", as suggested by our discussion of ancestors.

Definition 3. Let r be a given (universal) relation. For all s in P(D) the set of ij-cousins of s of order n (with respect to r), denoted by $C_{ij}^n(s)$, is defined as follows:

$$C_{ij}^n(s) = R_{ij}^n R_{ji}^n(s) = (\pi_i\pi_j^{-1})^n(\pi_j\pi_i^{-1})^n(s) \qquad \square$$

Intuitively, we can think of $R^n_{ji}(s)$ as the set of all "ancestors" of s, n generations backwards. Following the same reasoning, we can think of $R^n_{ij}(R^n_{ji}(s))$ as the "descendants" of $R^n_{ji}(s)$, n generations forward, that is, we can think of the "individuals" in the set $R^n_{ij}(R^n_{ji}(s))$ as "cousins" of s. It is important to note that C^n_{ij} is a function that goes from P(D) into P(D).

Example 4:

Referring to Figure 4, let us compute the 12-cousins of s={f} :

Order 1:

21-ancestors: $R^1_{21}(s) = \pi_2\pi_1^{-1}(\{f\}) = \{c,d\}$

12-cousins: $C^1_{12}(s) = R^1_{12}R^1_{21}(s) = \pi_1\pi_2^{-1}(\{c,d\}) = \{f,g\}$

Order 2:

21-ancestors: $R^2_{21}(s) = R^1_{21}R^1_{21}(s) = \pi_2\pi_1^{-1}(\{c,d\}) = \{a\}$

12-cousins: $C^2_{12}(s) = R^2_{12}R^2_{21}(s) = (\pi_1\pi_2^{-1})^2(\{a\}) = \{e,f\}$

Order 3:

21-ancestors: $R^3_{21}(s) = R^1_{21}R^2_{21}(s) = \pi_2\pi_1^{-1}(\{a\}) = \phi$

12-cousins: $C^3_{12}(s) = R^3_{12}R^3_{21}(s) = (\pi_1\pi_2^{-1})^3(\phi) = \phi$ ▫

Let us note that our notion of "cousins" generalizes that encountered in the literature. Indeed, using our definition we can ask for the cousins of a set of individuals. For example, referring to Figure 4, we can compute:

$C^1_{12}(\{e,f\}) = \{e,g,f\}$ and $C^2_{12}(\{e,f\}) = \{e,f\}$

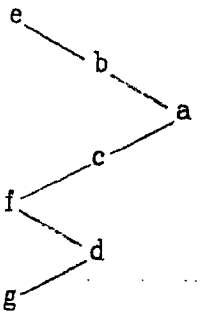
Similarly, we find:

$C^1_{12}(\{f,g\}) = \{f,g\}$ and $C^2_{12}(\{f,g\}) = \{e,f\}$

Clearly, when the argument of $C^n_{ij}(s)$ is a singleton, then Definition 3 coincides with the usual definition of cousins encountered in the literature.

Figure 4 A relation r and its graph representation

A_1	A_2
b	a
c	a
e	b
f	c
f	d
g	d



The second example of recursive query, that we would like to define as a PL-expression is the following:

Q_{ij}: Given a set s in P(D) find all ij-cousins of s (independently of the order)

Let $Q_{ij}(s)$ denote the answer to this query. From what we have said so far, in order to find $Q_{ij}(s)$, it is enough to "accumulate" the ij-cousins of s of order 1,2,... until no more ij-cousins are found. This will happen when no more "ancestors" are found (see Example 4). So, observing that

$$C_{ij}^n(s) = R_{ij}^n R_{ji}^n(s) = R_{ij} R_{ij}^{n-1} R_{ji}^{n-1} R_{ji}(s) = R_{ij} C_{ij}^{n-1} R_{ji}(s)$$

let us define

$$Q_{ij}^n(s) = C_{ij}^1(s) \cup C_{ij}^2(s) \cup \cdots \cup C_{ij}^n(s) \quad n=1,2,... \qquad (3a)$$

Then we can write

$$\begin{aligned} C_{ij}^2(s) \cup \cdots \cup C_{ij}^n(s) &= R_{ij} C_{ij}^1 R_{ji}(s) \cup R_{ij} C_{ij}^2 R_{ji}(s) \cup \cdots \cup R_{ij} C_{ij}^{n-1} R_{ji}(s) \\ &= R_{ij}(C_{ij}^1 \cup \cdots \cup C_{ij}^{n-1}) R_{ji}(s) \quad \text{[from Proposition 1g]} \\ &= R_{ij} Q_{ij}^{n-1} R_{ji}(s) \quad \text{[from definition above]} \end{aligned}$$

Thus we can re-write (3a) as follows:

$$Q_{ij}^n(s) = R_{ij} R_{ji}(s) \cup R_{ij} Q_{ij}^{n-1} R_{ji}(s) \qquad n=1,2 \ldots \qquad (3b)$$

or,

$$\begin{aligned} Q_{ij}^1(s) &= R_{ij} R_{ji}(s) \\ Q_{ij}^n(s) &= Q_{ij}^1(s) \cup R_{ij} Q_{ij}^{n-1} R_{ji}(s) \end{aligned} \qquad (4a)$$

So we can conclude (as for the "ancestor" query) that:

$$Q_{ij}(s) = Q_{ij}^{n_0}(s) \qquad (4b)$$

where $n_0 = \min(\{ n \;/\; Q_{ij}^{n+m} = Q_{ij}^{n}, m=1,2,\ldots \})$

And we can also see the answer $Q_{ij}(s)$ as the least-fixpoint solution of the following equation:

$$Q_{ij}(s) = C_{ij}^{1}(s) \cup R_{ij}\, Q_{ij}\, R_{ji}(s) \qquad (4c)$$

This equation (4c)is an immediate consequence of the equations (4a) and (4b).

4. THE EVALUATION OF RECURSIVE QUERIES

We have seen that the language PL allows specification of recursive queries as PL-expressions. We have also seen two important examples of such specification, namely, the ancestors and the cousins. In this section, we discuss evaluation of recursive queries. More specifically, we discuss:

(a) Modification and optimization of recursive queries, using the example of ancestors, and

(b) The influence of functional dependencies in the evaluation of recursive queries, using the example of cousins.

4.1 Optimal evaluation of ancestors

Given a set s in P(D), let $Q_{ij}(s)$ denote the set of all ij-relatives of s. We have seen that, in order to find $Q_{ij}(s)$, it is enough to accumulate the ij-relatives of s of order 1,2,... until no more relatives are found. Thus, following equations (2a) and (2c), we can write down the following algorithm for computing $Q_{ij}(s)$:

Algorithm: ANCESTORS 1
Input : A set of values s
Output : The set Q of all ij-relatives of s

```
X:=π_i π_j^{-1}(s);
Q:=X;
repeat
        L:=Q;
        Q:=X ∪ π_i π_j^{-1}( Q )
until  L = Q
```

The transformation of this algorithm into a relational algebra program is straightforward. Indeed, the only non-relational operator that appears in the algorithm is the inverse projection $\pi_j^{-1}(Y)$. However, as we have explained in Section 2, this inverse projection can be computed as a union of selections. More specifically, let $Y=\{y_1,y_2,\ldots,y_m\}$, and let $P=\pi_j^{-1}(Y)$. Then P can be computed as follows:

```
P:=φ;
for k:=1 to m do P:= P ∪ σ_{A_j = y_k}(r)
```

In general, as the only non-relational operator of the language PL is the inverse projection, the transformation of a PL-expression into a relational algebra program presents no particular difficulty. The interesting question here is, rather, that of modifying a PL-expression, usually for reasons of computational efficiency. The objective is to obtain an "equivalent" expression which is "easier" to evaluate. This problem, known as query modification, has received extensive attention in the relational literature (see for example in [M83]). Here, we discuss modification and optimization of a particular class of recursive queries, namely ancestors.

We have seen that equations (2a) and (2c), define the answer of the following query:

Q_{ij} : Given a set s in P(D), find all ij-relatives of s.

However, equation (2a) may lead to redundant computations. Let us see

an example.

Example 5

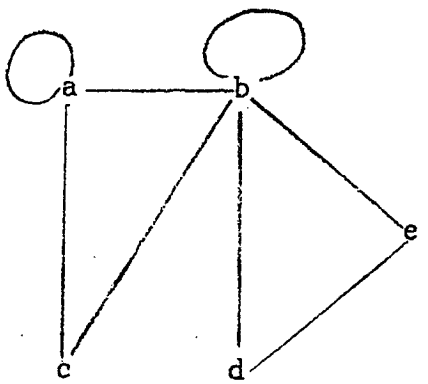
Referring to Figure 5 below, let us compute $Q_{ij}(\{a\})$, that is, the 21-relatives of {a}. Following equations (2a) and (2c), we find:

$$
\begin{aligned}
Q^1_{21}(\{a\}) &= R^1_{21}(\{a\}) = \{a,b,c\} \\
Q^2_{21}(\{a\}) &= Q^1_{21}(\{a\}) \cup \pi_2\pi_1^{-1}(Q^1_{21}(\{a\}) \\
&= \{a,b,c\} \cup \pi_2\pi_1^{-1}(\{a,b,c\}) = \{a,b,c\} \cup \{a,b,c,d\} \\
&= \{a,b,c,d\} \\
Q^3_{21}(\{a\}) &= Q^1_{21}(\{a\}) \cup \pi_2\pi_1^{-1}(Q^2_{21}(\{a\}) \\
&= \{a,b,c\} \cup \pi_2\pi_1^{-1}(\{a,b,c,d\}) = \{a,b,c\} \cup \{a,b,c,d,e\} \\
&= \{a,b,c,d,e\} \\
Q^4_{21}(\{a\}) &= Q^1_{21}(\{a\}) \cup \pi_2\pi_1^{-1}(Q^3_{21}(\{a\})) \\
&= \{a,b,c\} \cup \pi_2\pi_1^{-1}(\{a,b,c,d,e\}) = \{a,b,c\} \cup \{a,b,c,d,e\} \\
&= \{a,b,c,d,e\} = Q^3_{21}(\{a\})
\end{aligned}
$$

It follows that $Q_{21}(\{a\}) = \{a,b,c,d,e\}$. However, we observe that there are many computations that are repeated. This is what is called "naive evaluation" in [B85]. In particular, the relatives of a of order 1 (which are a,b and c) are re-computed in every step. Thus, we may remove a from $Q^1_{21}(\{a\})$ *before* we compute $Q^2_{21}(\{a\})$, without change in the final result. Similarly, we may remove b and c from $Q^2_{21}(\{a\})$ *before* we compute $Q^3_{21}(\{a\})$ and, finally, we may remove d from $Q^3_{21}(\{a\})$ *before* we compute $Q^4_{21}(\{a\})$.□

Figure 5. A relation r and the graph representation of $\pi_{12}(r)$.

A_1	A_2	A_3
a	a	a
a	a	b
a	b	c
a	c	a
b	a	d
b	b	e
b	d	b
c	b	b
d	e	d
e	b	e



The above observations lead us to replace equations (2a) and (2c) as follows:

$$\bar{Q}^0_{ij}(s) = s$$
$$\bar{Q}^1_{ij}(s) = R^1_{ij}(s) = \pi_i \pi_j^{-1}(s) \qquad (5a)$$
$$\bar{Q}^n_{ij}(s) = \bar{Q}^{n-1}_{ij}(s) \cup R^1_{ij}[\bar{Q}^{n-1}_{ij}(s) - \bar{Q}^{n-2}_{ij}(s)] \quad \text{for } n = 2,3,\ldots$$

$$Q_{ij}(s) = \bar{Q}^{n_0}_{ij}(s), \qquad (5b)$$
$$\text{where } n_0 = \min(\{ n \;/\; \bar{Q}^n_{ij}(s) = \bar{Q}^{n+m}_{ij}(s),\ m=1,2,\ldots\})$$

The following proposition shows that this replacement can be done safely:

Proposition 2. For all s in P(D), and for all n=1,2,... we have $\bar{Q}^n_{ij}(s) = Q^n_{ij}(s)$ ▫

Proof To make this proof more readable, we shall write Q^n instead of Q^n_{ij}, $\bar{Q}^n$ instead of $\bar{Q}^n_{ij}$ and R^n instead of R^n_{ij}

We can deduce from Proposition (1g) that:

$$R^1(Q^{n-1}) - R^1(Q^{n-2}) \subset R^1(Q^{n-1} - Q^{n-2}) \subset R^1(Q^{n-1})$$

and

(i) $$[R^1(Q^{n-1}) - R^1(Q^{n-2})] \cup Q^{n-1} \subset R^1(Q^{n-1} - Q^{n-2}) \cup Q^{n-1} \subset R^1(Q^{n-1}) \cup Q^{n-1}$$

Using the equation (1b), we have

(ii) $R^1(Q^{n-2}) \subset Q^{n-1}$

Combining (i) and (ii), we obtain

$$R^1(Q^{n-1}) \cup Q^{n-1} \subset R^1(Q^{n-1}-Q^{n-2}) \cup Q^{n-1} \subset R^1(Q^{n-1}) \cup Q^{n-1}$$

that is,

$$Q^{n-1} \cup R^1(Q^{n-1}) = Q^{n-1} \cup R^1(Q^{n-1}-Q^{n-2})$$

and then, using equation (2b)

$$Q^n = Q^{n-1} \cup R^1(Q^{n-1}-Q^{n-2})$$

Comparing this equation with the equation (5a), we can finally conclude that

$\bar{Q}^n = Q^n$, for n=1,2,... **Q.E.D**

Equations (2) and (5) can be seen as two different computations of the answer $Q_{ij}(s)$. It follows from Proposition 2 that these two computations produce the same result and, thus, they are equivalent. On the other hand, we have seen earlier that the computation of $Q_{ij}(s)$ following (5), is more economical, as it retreives fewer tuples of r (see Example 5). So let us define *performance* of a computation as the total number of tuples retrieved from r during the computation. Clearly, a computation of $Q_{ij}(s)$ is optimal if the following conditions holds:

(a): no tuple of r is retrieved more than once and,

(b): if a tuple t of r is retrieved, then t contributes to the result of the computation

Referring to equations (5) we observe that:

$T^1 = \pi_j^{-1}[\bar{Q}^0_{ij}(s)]$ is the set of tuples needed in the computation of Q^1_{ij} and

$T^n = \pi_j^{-1}[\bar{Q}^{n-1}_{ij}(s) - \bar{Q}^{n-2}_{ij}(s)]$ is the set of tuples needed in the computation of $\bar{Q}^n_{ij}(s)$, for n=2,3...

The following proposition shows that the computation defined by equations (5) is optimal:

Proposition 3. For all n=2,3,..., and all i=1,2,...,n-1 we have $T^n \cap T^i = \phi$ □

Proof We use the same simplified notations as in the proof of Proposition 2. Let t be a tuple in T^i for some i in {1,...,n-1} such that $\pi_{ij}(t) = ab$. From the definition of T^i and equations (5), we have $\pi_i(t) \in \bar{Q}^i$ and using Proposition 2, we obtain that $a \in Q^i$ and we deduce that $b \in Q^{i-1}$. Referring to equation (1a), we conclude that $b \in Q^{n-2}$ because $n-2 \geq i-1$. So we obtain that $b \notin (Q^{n-1} - Q^{n-2})$ and finally that $t \notin T^n$ **Q.E.D**

In view of Propositions 2 and 3, the following algorithm optimally computes the answer $Q_{ij}(s)$

Algorithm : ANCESTORS 2
Input: A set of values s
Output: The set Q of all ij-relatives of s
L1:=s;
Q:=$\pi_i \pi_j^{-1}$(s);
repeat
L2:=L1;
L1:= Q;
Q:=L1 $\cup$ $\pi_i \pi_j^{-1}$(L1 - L2);
until Q = L1;

It is important to note that the above algorithm computes ij-relatives of a set of values s and not the transitive closure of a relation.

4.2 The influence of functional dependencies

We have seen that the performance of a computation can be improved by query modification, that is, by finding a different PL-expression that produces the same answer. In this Section, we show that further improvement in performance is possible if we use semantic information such as functional dependencies. We explain this, using the example of cousins, that we have seen in section 3. In order to simplify the notation (and without loss of generality) we shall work with a relation r over two attributes.

Definition 4. Let r be a relation over A_1A_2. For all n=1,2..., we denote by r^n a relation on the (common) domain D defined as follows: For all x and y in D, $x\ r^n\ y$ if there is a sequence $x=u_0, u_1,...,u_n=y$ of n+1 elements of D, such that: $u_i\ u_{i+1}$ is in r for i=0,1,...,n-1 □

The following simple facts are immediate consequences of definitions 3 and 4:

Fact 1: the tuple xy is in r if and only if $x\ r^1\ y$ □

Fact 2: $y \in C_{12}^n(x)$ if and only if there is z such that $x\ r^n\ z \wedge y\ r^n\ z$ □

There is an interesting relationship between functional dependencies and cousins, stated formally in the following proposition:

Proposition 4. Let r be a relation over A_1A_2, satisfying the functional dependency $A_1 \rightarrow A_2$. Then for all x in D, and for all n=2,3,... we have:

$$C_{12}^n(\{x\}) \neq \phi \implies C_{12}^{n-1}(\{x\}) \subset C_{12}^n(\{x\}) \qquad \square$$

Proof Let $y \in C_{12}^{n-1}(\{x\})$. It follows, from Fact 2, that there is z such that: $x\ r^{n-1}\ z$ and $y\ r^{n-1}\ z$. It follows, from Definition 4 and Fact 1 that there are sequences $x=x_0, x_1, \ldots, x_{n-1}=z$ and $y=y_0, y_1, \ldots, y_{n-1}=z$ such that :

$$x_i\ r^1\ x_{i+1}\ ,\quad i=0,1,...,n-1 \qquad (a)$$

$$y_i\ r^1\ y_{i+1}\ ,\quad i=0,1,...,n-1 \qquad (b)$$

On the other hand, as $C_{12}^n(\{x\}) \neq \phi$, we have: $x \in C_{12}^n(\{x\})$. Thus, Fact 2 implies that there is w such that $x\ r^n\ w$. It follows from Definition 4 that there is a sequence $x=w_0, w_1, \ldots, w_n=w$, such that

$$w_i\ r^1\ w_{i+1}\ ,\quad i=0,1,...,n-1 \qquad (c)$$

Now, the relation r satisfies the functional dependency $A_1 \rightarrow A_2$, and x_0x_1, w_0w_1 are tuples of r. As $x=x_0=w_0$ it follows that $x_1=w_1$. If we repeat this argument n-1 times, we obtain that:

$$x_i = w_i\ ,\quad i=0,1,...,n-1 \qquad (d)$$

As $x_{n-1}=z$, it follows from (d) that $z=w_{n-1}$. Thus, (c) implies that $z\ r^1\ w_{n-1}$ and, as $w_n=w$, it follows that $z\ r^1\ w$. Finally, as $y\ r^1\ z$ and $z\ r^1\ w$, we obtain $y\ r^n\ w$. Thus we have: $x\ r^n\ w$ and $y\ r^n\ w$. It follows, from Fact 2, that

$y \in C_{12}^{n}(\{x\})$, **Q.E.D**

With Proposition 4 at hand, let us look again at the example of cousins. We are interested in the following query:

Q_{12} : Given a set s in P(D) find all 12-cousins of s.

The answer $Q_{12}(s)$ to this query can be computed using the equations (4a) and (4b). It follows from (3a) that

$$Q_{12}^{n}(\{x\}) = C_{12}^{1}(\{x\}) \cup \cdots \cup C_{12}^{n-1}(\{x\}) \cup C_{12}^{n}(\{x\}) , \quad n=1,2,...$$

Now, assume that the relation r satisfies the functional dependency $A_1 \rightarrow A_2$. Then according to proposition 4, we have :

$$C_{12}^{n}(\{x\}) \neq \phi \implies C_{12}^{n-1}(\{x\}) \subset C_{12}^{n}(\{x\}) , \qquad n=2,3,...$$

It follows that

$$Q_{12}^{n}(\{x\}) = C_{12}^{n}(\{x\}) \ , \ \text{if } C_{12}^{n}(\{x\}) \neq \phi$$

$$= C_{12}^{n_0}(\{x\}), \text{if } C_{12}^{n_0}(\{x\}) \neq \phi \text{ and } C_{12}^{n_0+1}(\{x\}) = \phi, \text{otherwise}$$

Therefore,if the relation r satisfies the functional dependency $A_1 \rightarrow A_2$, then there is a significant improvement of performance.

In conclusion, we have seen two ways of improving performance when evaluating recursive queries. The first, modifies the structure of the PL-expression, whereas the second uses the knowledge of semantic information, such as functional dependencies.

5. CONCLUSIONS

We have presented a purely algebraic approach to recursively defined queries in relational databases. Our algebra is based on the product space of all attribute domains, and its basic operations are set-theoretic difference, intersection and union, projections and their inverses, and function composition. The well-formed expressions of the

algebra constitute a language that we called PL.

The basic building block for the specification of recursive queries in the language PL is the ij-mapping: $\pi_i \pi_j^{-1}$. Using ij-mappings, we have presented two important examples of recursively defined queries: the ancestors and the cousins. We have also discussed performance issues, and we have given an optimal algorithm for computing ancestors. Furthermore, we have shown that functional dependencies can be profitably used to improve performance in the evaluation of cousins.

We are currently working on two aspects of our approach. The first is related to implementation. Our ij-mapping is closely related to the main operator of the relational language SQUARE. The SQUARE operator, also called mapping, is a selection followed by a projection. In fact, the mapping concept is carried over into the basic syntax of the language SQL as well. We are currently investigating the possibility of implementing the language PL in SQUARE or SQL. The second aspect of our current work is related to semantic issues. We have seen that functional dependencies can be taken into account when evaluating recursive queries. We are currently investigating the possibility of using other kinds of semantic information in the evaluation of recursive queries.

References

[B85] F. BANCILHON, "Naive Evaluation Of Recursively Defined Relations" Technical Report Number: DB-004-85 M.C.C.

[B85b] F. BANCILHON, D. MAIER, J. ULLMAN, "Magic Sets and Other Strange Ways to Implement Logic Programs" submitted to PODS 86

[C70] E.F. CODD, "A Relational Model of Data for large shared data banks" CACM 13,6 1970 pp 377-387

[D66] J. DUGUNDJI, "Topology" Allyn and Bacon Inc 1966

[G84] H. GALLAIRE, J. MINKER, J.M. NICOLAS, "Logic and Databases: A Deductive Approach", ACM Computing Surveys 16,2 June 1984

[HN84] L. HENSCHEN and S. NAQVI, "On Compiling Queries in Recursive First-Order Databases" JACM, Vol31, January 1984. pp 47-85

[M83] D. MAIER "The Theory of Relational Databases" Computer Science Press 1983

[U83] J. ULLMAN "Principles of Database Systems" Computer Science Press 1983

[U85] J. ULLMAN, "Implementation Of Logical Query Languages For Databases" ACM Transactions on Database Systems, 10:3, pp 289-321, 1985

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique