IS AN EXTENDED RELATIONAL DATABASE MANAGEMENT SYSTEM POWERFUL ENOUGH TO DEAL WITH NETWORK APPLICATIONS ?

MAINGUENAUD Michel
Institut National des Télécommunications
9 rue Charles FOURIER
91011 EVRY - FRANCE
email : MAINGUENAUD@FRINT51.bitnet
33 1 60 76 40 40

Abstract :

Part of Geographical applications is network management. Travel agencies, gas, electricity, telephone applications require a computer assisted tool. Network applications are "easier" to manage than thematic applications from an analytical point of view because networks can be modelized using graph theory concepts (nodes, edges, paths). In this paper we present a taxonomy of queries for a Network Management System, a Data Manipulation Language to define a query and describe the solutions to be used with an Extended Relational Database Management System.
The queries addressed to a Network oriented GIS can be divided in four classes : path from one point to an other point under constraints, intersection of paths, inclusion of paths and node manipulations. A user-friendly interface is required to define a query therefore we briefly describe GROG a geographical query language using graphs and present how to resolve the four classes of queries with a Relational Database Management System extended with a Transitive Closure operator.

1. Introduction :

In current research toward the design of more powerful Data Base Management System (DBMS), different research group are simultaneously concentrating their work on Geographical Information System (GIS). Now GIS needs are well known (SMSE87). The main problems in GIS design are to define data modelization, Data Manipulation Language (DML), and efficient implementation.

Part of GIS applications is network management such as roads, telecommunications, railways etc. These data can be "easily" represented using graph theory formalism (node, edge, path). Manipulations of graphs are very often defined with recursive (or logic-based) formalism. Extended Relational Data Base Management System (ERDBMS) and Object Oriented philosophy (O.O.) can be used to design a GIS architecture.

In this paper we olny focus on ERDBMS. In section 2 we briefly present some typical queries for network management system and GROG a DML to defined such queries. In section 3, we define the organization of an ERDBMS able to deal with such queries. Section 4 has the conclusions and discussions of further works.

2. GROG : a DML for Network oriented Queries :

This section presents various typical queries addressed to a Network oriented Information Management System and a Data Manipulation Language to express such queries.

EGIS 90
European Geographical Inf Syst C
Amsterdam
9.13. Avril 90

2.1 Classes of queries :

Four main classes can be observed : 1) going (path) from one point (node) to an other under constraints 2) intersection of paths 3) inclusion of path 4) node manipulations.

Example of queries :

Query 1 : What are the paths from Nice to Paris ? This query can be evaluated with a graph traversal operator. (Class 1)

Query 2 : What are the paths from Nice to Paris with olny large-city stages ? This query can be evaluated with a graph traversal operator and constraints on the nodes of the graph. (Class 4)

Query 3 : What are the paths from Nice to Paris using AIR FRANCE (AF) company ? This query can be evaluated with a graph traversal operator and constraints over the edges of the graph. (Class 1)

Query 4 : What is the shortest path from Nice to Paris using motorway ? This query can be evaluated with a graph traversal operator, aggregates and constraints over the edges of the graph.(Class 1)

These queries can be written with Horn clauses (CGT88). But Horn clauses are not well adapted as a user-interface : 1) Horn clauses are not a very user-friendly language for the one who is not used to recursive concepts, 2) Horn clauses cannot modelise some typical queries such as :

Query 5 : What are the common parts between a path from Nice to Geneva (which costs less than 10 units) and a path from Paris to Vienna ? (Class 2).

Query 6 : What are the paths between Paris and Nice using motorways between Lyon and Marseille - without cycles- ? (Class 3).

Network Information Management Systems need a user-friendly Data Manipulation Language which allows recursive queries to be expressed.

2.2 Grog : Data Manipulation Language :

The starting point of Grog can be found in (CMW87). An adaptation to GIS applications can be found in (M89). We give here very briefly the main orientations of Grog.

Data are modelised by a directed graph. A directed graph G is represented by G(N, E). N is a set of nodes, E is a set of edges between two elements of N. Nodes and edges are labeled. All the graphs which are used here will be multi-graphs : two edges with different labels can occur between two given nodes (to simply multi-graphs will be called graphs).

Definitions :

A graph G is defined by G (N_G, E_G, ψ_{1G}, ψ_{2G}, ψ_{3G}, ν_G, ε_G) :

. N_G is a set of nodes $\quad N_G = \{n_1, \dots, n_p\}$

. E_G is a set of edges $\quad E_G = \{e_1, \dots, e_q\}$

. ψ_{1G} is an incident function between nodes

$$\psi_{1G} : \quad E_G \quad \text{->} \quad N_G \times N_G$$

. ψ_{2G} is an incident function between a node and an edge

$$\psi_{2G} : \quad E_G \quad \text{->} \quad N_G \times E_G$$

. ψ_{3G} is an incident function between edges

$$\psi_{3G} : \quad E_G \quad \text{->} \quad E_G \times E_G$$

. ν_G is a node labeling function :

$$\nu_G : N_G \rightarrow I_n \qquad \text{with } I_n = D_{n0} \times \dots \times D_{nn}$$

i.e. : Name x Population x Museum.

. ε_G is an edge labeling function :

$$\varepsilon_G : E_G \rightarrow I_e \qquad \text{with } I_e = D_{e0} \times \dots \times D_{ee}$$

i.e. : Company x Departure_time x Arrival_time

Queries :

A graphical query Q defined on a graph G, is a set of labeled and oriented graphs. Let Q be defined as $\{Q_1, \dots, Q_p\}$. The labels of the nodes can be variables, or constants.

Edges :

Three types of edges are available :

Link Edge 
direct /transitive

Inclusion Edge 

Intersection Edge 

General properties :

These edges follow the same rules : 1) these edges are oriented, 2) these edges represent the results of sub-queries, 3) these edges are a binary operator (one initial point, one end point), 4) these edges represent paths without any cycle.

Definitions of the various manipulations of data :

<u>Manipulation of edges</u> :

Link : 1)- Direct Link	(i.e. query 3)	π_D
2)- Transitive Link	(i.e. query 1, 3, 4)	π_T
3)- Intersection of paths (edges)	(i.e. query 5)	π_{Is_e}
4)- Inclusion of paths (edges)	(i.e. query 6)	π_{Ic_e}

<u>Manipulation of nodes and edges</u> :

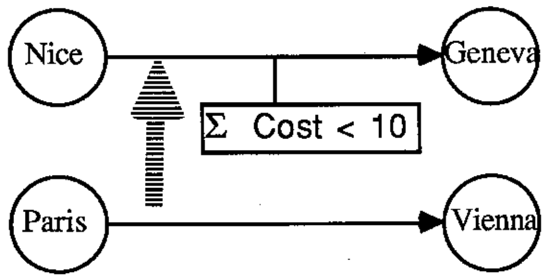
5)- Inclusion (i.e. query 2) π_{Ic_ne}

<u>Manipulation of nodes</u> :

6)- Intersection of nodes (i.e. common nodes between two paths) π_{Is_n}

7)- Inclusion of nodes (i.e. paths using the same nodes even if the edges are different) π_{Ic_n}

With such definitions query 5 is expressed by :



3. Implementation :

Nowadays two main philosophies can be used to design a GIS : Extended Relational Data Base Management System (ERDBMS) and Object Oriented (OO). Various prototypes can be found such as (RS88),(D*86, S*88),(G*89, S*86), (Exodus, Genesis, ...) for ERDBMS and (B*87),(B*88), (Gemstone, ...) for Object Oriented Systems. The persistance of data is still a very difficult problem if we want to have good performances with O.O system. In this section we study how network applications can be managed by an ERDBMS.

3.1 DB organization :

To simply, without lost of generality, we define our toy application with a DB relation Network. To help comprehension let the schema of Network relation be (even if it is not the best implementation) :

Network (#Edge, Origin, Destination, Edge_Information)
where Edge_information can be seen as a set of attributes.

The system is supposed to have a Transitive Closure operator able to evaluate a path without cycles, to define a numerotation of paths and inside a path, to deal with aggregates. This is not a restrictive condition because numerous work have been done on Transitive Closure operator in a DB context (for further information see (BR86)).
This operator manipulates relations (sets) : Sub_Network, Set_Origin, Set_Destination, and a list of constraints defined over nodes or edges. The general form of the TC operator will be :

TC [σ_{C1} (Network), Set_Origin, Set_Destination, Constraints]
where σC_1, Set_origin, Set_destination will be defined for each operation.

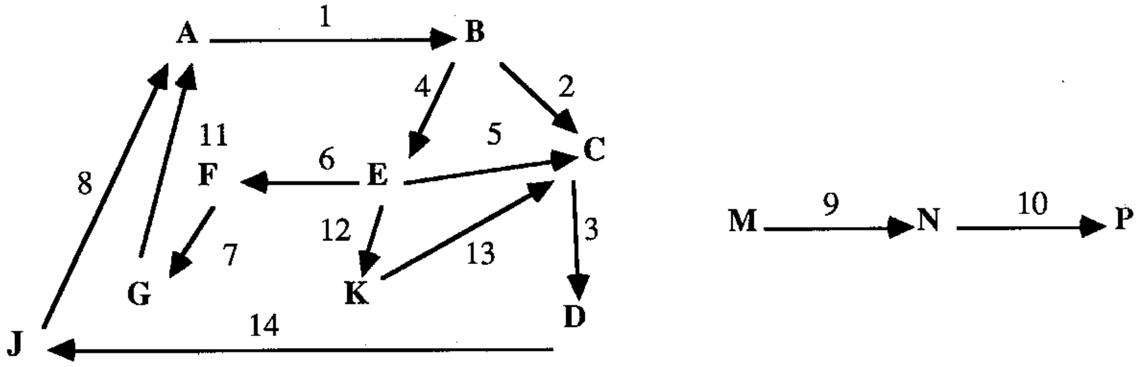
The result of such an operator will be a relation containing paths. To help comprehension let the schema of the relation $Result_i$ be :

$Result_i$ (#Path, #In_the_Path, #Edge, Origin, Destination).

3.2 Operations :

To simply without lost of generality we do not deal with constraints over edges or nodes (because these constraints can be introduced in C_1 selection criteria or with variables). Manipulations will be expressed using Relational Algebra (let Π be the projection, σ the selection and $\bowtie$ the join operators) or with an Extended-SQL-like language when it is not possible to express the query otherwise.

Let G be the following graph :



For our example let Result1 and Result2 be defined as in Figure 1.

Result1 (#Path	#In_the_path	#Edge	Origin	Destination)
1	1	1	A	B
1	2	2	B	C
1	3	3	C	D
2	1	1	A	B
2	2	4	B	E
2	3	5	E	C
2	4	3	C	D
3	1	6	E	F
3	2	7	F	G

Result2 (#Path	#In_the_path	#Edge	Origin	Destination)
1	1	1	A	B
1	2	2	B	C
2	1	8	J	A
2	2	1	A	B
3	1	9	M	N
3	2	10	N	P
4	1	11	G	A
4	2	1	A	B
4	3	4	B	E
4	4	12	E	K
4	5	13	K	C
4	6	3	C	D
4	7	14	D	J

Figure 1

Remarks :

Path 1 of relation Result2 is fully included in a path of Result1. Path 2 has an intersection with a path of Result1. Path 3 has no intersection with any path of Result1. Path 4 has a non-connexe intersection with a path of Result1.

Manipulation 1 : (i.e. query 1) π_D evaluates the direct edges between two nodes

The general form is : $\pi_D = \sigma_{C1}$ (Network)

Value of C_1 depends on the following cases :

Origin : Oi and Destination : Dj are known
C_1 : Origin = Oi $\wedge$ Destination = Dj

Only Origin : Oi (resp Destination : Dj) is known
C_1 : Origin (Destination) = Oi (Dj)

Manipulation 2 : (i.e. query 1, 3, 4) π_T evaluates the paths between two nodes

The general form is π_T = TC [σ_{C1} (Network), Set_Origin, Set_Destination, Constraints]

Values of Set_Origin, Set_Destination depend on the following cases :

Origin : Oi and Destination : Dj are known
Set_Origin = {Oi} and Set_destination = {Dj}

Only Origin : Oi (Destination : Dj) is known
Set_Origin (Set_destination) = {Oi (Dj)}
Set_destination (Set_origin) = $\Pi_{\text{Destination (Origin)}}$ (Network)

Neither Origin or Destination are known
Set_Origin = Π_{Origin} (Network)
Set_Destination = $\Pi_{\text{Destination}}$ (Network)

Let Set_Origin, Set_Destination and Constraints be defined, the ESQL-like will be :

```
Insert into Result
Select TC (Sub_Network, Set_origin, Set_destination, Constraints)
From Sub_Network, Set_origin, Set_destination, Constraints
```

(where Sub_Network = σ_{C1} (Network))

Manipulation 3 : (i.e. query 5) π_{is_e} evaluates the intersections between two sets of paths.

Let Result1, Result2 be the relation obtained by TC operator for sets of paths P1, P2 as in Figure 1 :

π_{is_e} is obtained by : $\pi_{is\_e} = \text{Result1} \underset{\#Edge}{\bowtie} \text{Result2}$

Non-empty intersection constraints is obtained by :

$$T1 = \Pi_{\#Path1} (\pi_{is\_e})$$
$$T2 = \Pi_{\#Path2} (\pi_{is\_e})$$

$$\text{Result1} = \text{Result1} \underset{\#Path1}{\bowtie} T1$$

$$\text{Result2} = \text{Result2} \underset{\#Path2}{\bowtie} \text{T2}$$

π_{is_e} will be defined as in Figure 2.

Result (#Path1	#Path2	#Edge)
1	1	1
1	2	1
1	4	1
1	1	2
1	4	3
2	1	1
2	2	1
2	4	1
2	4	4
2	4	3

Figure 2

Manipulation 4 : π_{ic_e} evaluates the inclusion of paths

π_{ic_e} is one of the most complex operation. This operation cannot be expressed using Relational Algebra (but it can be expressed using SQL-statements), because this manipulation is a kind of generalized division.

Relational division finds sub-tuples in a unique relation satisfying a sub_relation. The problem here is to find sub_relation of Result1 satisfying a sub_relation belonging to a different relation (Result2).

The main idea is :

$P(Result_j) \supseteq P(Result_i) \Leftrightarrow Card\ (P(Result_i)) = Card\ (P(Result_i) \cap P(Result_j))$ P : set partition

Let generalize this notion to sub_sets (Paths)

$\forall$ P1 $\in$ $Result_i$, P2 $\in$ $Result_j$ P1 $\supseteq$ P2 <=> Card (P2) = Card (P1 $\cap$ P2)

The paths in $Result_i$ and $Result_j$ are numeroted independently P1 $\cap$ P2 is obtained using a join operator over #Edge (and not with the relational algebra operator Intersect).

Let Result1 and Result2 be defined as in Figure 1, the following statements allow to define the inclusion of path (a path of Result2 in a path of Result1):

1/ Let Jointure represents the relation obtained by : $\text{Jointure} = \text{Result1} \underset{\#Edge}{\bowtie} \text{Result2}$

This operation allows to define all the common edges between the sets of two paths (see Figure 2).

2/ Let Number_Edge represents the relation obtained using the Group By and Count SQL-statements over relation Jointure. Let relation Number_Edge defines for each couple of paths the number of common edges (Nb) : Number_Edge (#Path1, #Path2, Nb)

```
Insert into Number_Edge
        Select      #Path1, #Path2, count (*)
        From        Jointure
        Group By    #Path1, #Path2
```

Number_Edge (#Path1	#Path2	Nb)
1	1	2
1	2	1
1	4	2
2	1	1
2	2	1
2	4	3

3/ Let Size_Path represents the relation obtained using the Group By and Count SQL-statements over relation Result2. Let Size_Path defines the length of each paths (Nb) : Size_Path (#Path2, Nb)

```
Insert into Size_Path
        Select      #Path2, count (*)
        From        Result2
        Group By    #Path2
```

Size_Path (#Path2	Nb)
1	2
2	2
3	2
4	7

4/ Let R3 represents the join between the Number_Edge relation and the Size_path relation over #Path2 and Nb

$$R3 = \text{Number\_Edge} \underset{\#Path2,\ Nb}{\bowtie} \text{Size\_Path}$$

R3 (#Path1	#Path2	Nb)
1	1	2

The Group By clause defines the length of the path for the relation Size_Path and the number of the common edges between the two paths for the relation Jointure. A path P2 is included in a path P1 if the number of common edges between P1 and P2 is equal to the number of edges of path P2.

Manipulation 5 : (Query 2) π_{ic_ne} evaluates the set of nodes of a path

Let Result1 be defined as in Figure 1 :

$$\pi_{ic\_ne} = \Pi_{\#Path,\ Origin}(\text{Result1}) \cup \Pi_{\#Path,\ Destination}(\text{Result1})$$

Manipulation 6 : π_{is_n} evaluates the set of common nodes between two paths

Let Result1 and Result2 be defined as in Figure 1. Let N1 (resp N2) be the set of nodes of paths P1 and P2 (see manipulation 5). The schema of N1 (resp. N2) is (#Path, Node).

$$\pi_{is\_n} = N1 \underset{Node}{\bowtie} N2$$

Manipulation 7 :

As π_{ic_e}, π_{ic_n} is a complex operation. It cannot be expressed using Relational Algebra. Let Result1 (resp Result2) be defined as in Figure 1. Let N1 (resp N2) be a relation defined over the schema (#Path, Node) (see manipulation 5). The following statements (similar to manipulation 4) allow to define the inclusion of node

1/ Let Jointure represents the relation obtained by : $Jointure = N1 \underset{Node}{\bowtie} N2$

This operation allows to define all the common nodes between the two sets of paths.

2/ Let Number_Node represents the relation obtained using the Group By and Count SQL-statements. Number_Node defines an aggregate value for each path (number of nodes). Let Number_Node be defined as (#Path1, #Path2, Node)

```
Insert into Number_Node
        Select     #Path1, #Path2, count (*)
        From       Jointure
        Group By   #Path1, #Path2
```

3/ Let Size_Node represents the relation obtained using the Group By and Count SQL-statements. Size_Node defines an aggregate value for each path (number of nodes). Let Size_Node be defined as (#Path2, Node)

```
Insert into Size_Node
        Select #Path2, count (*)
        From       N2
        Group By   #Path2
```

4/ Let R3 represents the join between the Number relation and the Size_Node relation over #Path2 and Node.

$$R3 = Number\_Node \underset{\#Path2,\ Node}{\bowtie} Size\_Node$$

The Group By clause defines the number of node for the relation Size_Node and the number of the common nodes between the two paths for the relation Jointure. The set of nodes of path P2 is included in a path P1 if the number of common nodes is equal to the number of nodes of path P2.

4. Conclusion :

In this paper we present a taxonomy of various queries addressed to a Network oriented Information System and then a graph-oriented query language is briefly described. We show that a Relational Database Management System extended with a Transitive Closure operator (Relational Data

Base Management System using a weak couplage with Prolog for instance) is powerful enough to deal with the main queries addressed to a Network oriented Information System.

Implementation of such system is a real problem but technology is now available in a Data Base context. The aim of Cigales system (MP90) is to unify Network approach and Thematic approach in an unique user-friendly query language. Object Oriented approach has to be evaluated to deal with such applications : What is the query resolution model ?, Is it very realistic ? ... (an open problem to see).

REFERENCES

B*87 J. Banerjee and all, Data Model Issues for Object Oriented Applications, ACM Transaction on Office Information System, 5(1), April 87

B*88 F. Bancilhon and all, The Design and Implementation of O2, an Object Oriented Database System, Proc. of the OODBS II Workshop, Bad-Munster - FRG, September 1988

BR86 F. Bancilhon, R. Ramakrishnan, An amateur's Introduction to Recursive Query Processing Strategies, Proc. of the ACM SIGMOD Conference, Washington - USA, Mai 1986

C*86 M. Carey and all, The architecture of the Exodus Extensible DBMS, Proc of the Int. Workshop on Object-Oriented Database System, Pacific Grove, Ca - USA, Sept. 1986

CGT88 S. Ceri, G. Gottlob, L. Tanca, Logic Programming and Databases, Springer-Verlag 1988

CMW87 I.F.Cruz, A.O. Mendelzon, P.T. Wood, A Graphical Query Language supporting Recursion, Proc. of the ACM SIGMOD Conference, San-Fransisco - USA 27-29 May 1987

D*86 P. Dadam and all, A DBMS Prototype to support Extended NF2 Relations : An integrated View of Flat Tables and Hierarchies, Proc. of the SIGMOD Conference, Washington - USA, May 1986

G*89 G. Gardarin and all, Managing Complex Objects in an Extensible Relational DBMS, Proc of the VLDB Conference, Amsterdam, The Netherlands, 22-25 Aug 89

M*86 D. Maier and all, Development of an Object-Oriented DBMS, Proc. of the ACM OOPSLA Conference, Portland - USA, 1986

M89 M. Mainguenaud, GROG : Geographical queries using graphs, Advanced Database System Symposium - IPSJ, Kyoto - Japan, 7-8 Dec. 1989

MP90 M. Mainguenaud, M.A. Portier, Cigales : A graphical Query Language for Geographical Applications, to appear in 4th Int. Symposium on Spatial Data Handling, 23-27 July 1990, Zürich - Switzerland

RS88 L. Rowe M. Stonebraker, The Postgres Data Model, Proc. of the 13th VLDB Conference, Brighton - GB 1987

S*86 P. Schwarz and all, Extensibility in the Starburst Database System, Proc of the Int. Workshop on Object-Oriented Database System, Pacific Grove, Ca - USA, Sept. 1986

S*88 M. Scholl and all, Verso : A DBMS based on Nested Relations, Nested Relations and Complex Objects Workshop, Darmstadt - FRG, 1988, L.N.C.S. Springer-Verlag

SMSE87 TR Smith, S. Menon, JL Star, JE Ester, Requirements and principles for implementation and construction of large scale GIS, Int. Journal of Geographical Information Systems, vol 1, n°1, 1987, pp 13-31