International Journal of Computer Science & Applications
Vol. I, No. II, pp. 76 - 95

A*: Algebra for an Extended Object/Relational Model

S. Nait Bahloul[(1)], Y. Amghar[(2)], M. Sayah[(1)]
[(1)] *Département Informatique,- Faculté des Sciences - Université Es-Sénia*
Bp 1524 El Ménouar Oran 31000 - Algerie
[(2)] *INSA de Lyon – LIRIS FRE 2672 CNRS,*
7 avenue Jean Capelle. 69621, Villeurbanne – France.

Abstract

The object relational data model presents both the advantage of Codd's relational calculus power and the characteristics of the object orientation. Two major approaches have been adopted to satisfy the requirements of new databases applications. A first approach integrates the object characteristics into the new data models with the specification of data constraints and the definition of interrogation language. The second one, called evolutionary approach, keeps Codd's data model enriching with adequate concepts for the coverage of current database applications. In this approach and comparatively with studies presented by Melton, Date and Darwen have proposed the foundations of the object relational model. So, *A*-algebra consisting of first order logic operators has been defined to express various classes of queries in object relational database. To contribute to the improvement of relational/object models and agebra this paper presents an extension of object relational model to new types generated by operators and the related *A**-algebra. These operators, called *Op*, offer a means to specify domains as functions and permit consequently to increase the data model expressiveness. To support this extension, we propose a new data query language, or more precisely a logical data calculation A* as an adapatation of the *A*-algebra. Our *A**-algebra contains algebraic operators which are able to support this new extension.

Keywords: Model, Object relational, Database, Models, Algebra, Extension, Operator, *A**.

1. Introduction

The object relational data model [1,2,3] presents both the advantage of Codd's relational calculus power [5] and the characteristics of the object orientation [6,10]. Two major approaches have been adopted to satisfy the requirements of new databases applications. A first approach widely illustrated in [3,6,13] integrates the object characteristics into the new data models with the specification of data constraints and the definition of interrogation language. This approach is still considered as important research area. The second one [2,16], called evolutionary approach, keeps Codd's data model enriching with adequate concepts for the coverage of current database applications. In this approach and comparatively with studies presented by Melton [14], Date and Darwen have proposed in [1,2] the foundations of the object relational model. So, *A*-algebra consisting of first order logic operators has been defined to express various classes of queries in object relational database.

Database management systems [7] had to deal with complex, heterogeneous, temporal and available data on many sites. Two major tendencies have born to meet the new requirements of database applications: the object oriented database approach [8,9,10,11] and the object /relational database approach [1,2,3,4]. The object/relational database approach has to preserve inherent advantages of the relational model, which offers two language categories: an algebraic language

based on relational algebra and a relational calculation language based on the logic of first order. The logical aspect of the relational calculation allows the user to specify a query in a declarative way. In particular, the user does not need to know how data are physically stored in the database. Besides, relational algebra shows more easily some equivalencies between the algebraic language expressions. The power of language expression is an important question. The limits of relational algebra expressiveness are well known. One of examples is the operator of transitive closure of a relation that cannot be expressed with relational algebra operators. Relational algebra is also known by its limitation to express some simple and useful queries, which are easily expressed in the language SQL: for example, the queries like those asking for the calculation of annual salary from the monthly salary of employees. Indeed, it is admitted [15,16] that any query requiring an arithmetical calculation or functions of SQL aggregates is not possible with the relational algebra. Motivated by the need of a large coverage, researches have been undertaken in order either to increase existing algebras or to define other families of algebras such as the algebras for nested relation models [17,18,19,20,21,22,23], the algebras for structured value models [24,25,26,27] and the algebras for object models [12,28,29]. One the interests of these researches is the formalization of the object/relational model and the proposition of a object/relational algebra [1, 2] called *A*. for the formal object/relational model the *A*-algebra has allowed the identification of adequate operation types. However, the query language design related to these data models consists of defining algebraic operators. These operators are based on the *A*-algebra to which the extension operators must be integrated. They facilitate both the description and interrogation of complex data rich in symbolic representation, but requiring a prohibitive combinatorial calculation during their manipulations. The constructors, used in query languages based on the *A*-algebra, such as *D* language [1], adopt the set theory for the data specification and the first order logic calculation to manipulate such data.

In this paper, we propose to enhance object/relational model through the notion of domain generated by function or operator and to develop an **algebra noted *A**** as an extension of *A*-algebra. The query language based on *A** should be still considered with regard to the power of expression and calculation which is offered by a standard query language like SQL3. ***A*-algebra***, is composed of logical operators (i.e. *not *, or*, and *, compose **) and extension algebraic operators (i.e. : *ext.. add ..by*). Both are specific to the requirements of the new object/relational model. The main idea in this paper consists of integrating within a object/relational scheme, types defined by operators. These types built likewise, are not generated by constructors but defined by operator *Op* as being couples <*Op, TOp*> and where *TOp* is the type returned by the function or constructed by the operator *Op*. Then *A** allows the manipulation of complex entities requiring symbolic representations in their definitions such as in the geometrical and spatial databases. These latter, processing the shape and position of objects in space and time constitute an application domain, which evokes new query classes as, topological queries and geographical queries, aggregate queries where a value is associated to a point set. The treatment of such query classes leads to different problems, such as the choice of data representation model, the data interrogation and transformation language as well as algorithm complexity of such transformations. In [7], extension operators have been developed to take into account sophisticated queries which deal with geometrical figures in relational or extended relational context. In [30], geometrical queries are resolved in polynomial time in a constraint database context. In A*, geometrical queries are tackled in an extended object/relational database context to the concept of domain generated by operator or function. Thus, *A** containing extension operators is carried out by the query language *ERA** for the querying of object / relational databases.

This paper is oraganized as follows. In section 2, a typical example on a road network connecting geographic zones of polyedrical form is presented. This example will illustrate our concepts. Section 3, concerns Date and Darwen's approach related to the formalization of the object/relational model as well as *A*-algebra for the manipulation of data in such models. In sections 4 and 5 which are the core of the paper, we detail the extension of the object/relational model and the specification of A*. In addition elements about ERA* (Extended Relational A* language) are given. Section 6 is a discussion which compares *A** with the relational algebra on the one hand and with specification and querying languages of object database schemas (i.e., ODL, OQL) on the other hand. This paper ends with section 7, which concludes our work.

2. Illustrative example

Let us consider a network of roads in a geographical space and let us interest to topological queries. The matter is to treat the nature of the road network in the neighbourhood of one or several cities. We evaluate, through the metrical query, the characteristics of road or geographical networks such as the different road accesses of a given city. Such query classes comprise particular constraints like the intersection point of different road segments, evolutions of the road network in a region or a given zone, the management and transformation of a such road network and the exploitation of new geographical zones expressed under the form of geometrical figures (Figure 1). However, major difficulty for the resolution and management of such queries implies the necessity to have a database language rich in data modelling and querying. Also, this language has to be effective in the query resolution regarding to both the quantity of available numerical data and the nature of the calculation carried out. We consider, in the rest of this paper, a case of road network where the convex regions *A, B, C* are simple geometrical figures such as rectangles, square, triangles and polygons (Figure 1). So the road network is represented and carried out, in language *D* of Date and Darwen [1, 2], through the object/relational database named *ROAD_BASE*:

ROAD_BASE

Points = RELATION{Id_Point char(10),Abscissa real,Ordinate real } KEY { Id_Point } ;
Road_ Segments= RELATION{ Id_seg int, first_seg char(10), last_seg char(10)} KEY { Id_seg };
Roads = RELATION{ Id_path int, Road SET(int) } KEY { Id_path};
Polygons =RELATION{ Id_poly int, Polygon SET(int)} KEY { Id_poly } ; *where SET is a predefined type within the DBMS*

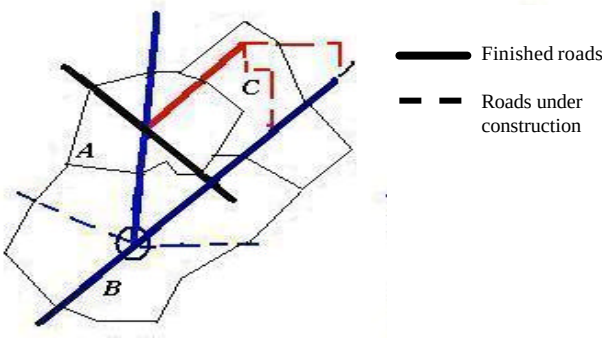

Figure 1: Road network

Notes:

- The road network is defined partially, but sufficiently regarding to our proposition.
- Relations are expressed according to the Date and Darwen's formalism [1, 2].
- The position of a city in the road network is represented with a point belonging to the relation *Points*.
- *Id_path* identifies the road, the set *{Id1, Id2, Idn}* defines the respective identifiers of segments of this road.
- *Id_poly* identifies the polygon, the set *{Id1,Id2,…,Idm}* defines the respective identifiers of segments constituting a given polygon.
- Relation *Road_ Segments* represents the segments that board the cities defined in the relation *Points*. Attributes *first_seg* and *last_seg* are the respective identifiers of departure and arrival points in a segment identified by *id_seg*.
- The relation *Roads* specifies all the possible roads between the different cities in the relation *Points*; the road being a set of road segments.
- The relation *Polygons* defines the set of the convex figures in the road network.

Let us consider now a given state of the database *ROAD_BASE* containing the five different cities *A, B, C, D, E* in a space *S* (Figure 2). A relation *r*, in the Date and Darwen's formalism is defined by two sets representing respectively the heading and the body of *r*. The heading *Hr*

specifies the scheme of the relation whereas the body *Br* contains the tuples corresponding to *Hr*. So, the relations *Points, Road_Segments, Roads and Polygons are* defined as follows:

H_Points = {<Id_Point, char(10)>, <Abs, real>, <Ord, real>};
B_Points = {{<Id_Point, char(10), A>, <Abs, real, 6>, <Ord, real, 12>},
{<Id_Point,char(10),B>,<Abs, real,12>,<Ord, real,10>},
{<Id_Point, char(10), C>, <Abs, real, 3>, <Ord, real, 6>},
{<Id_Point,char(10),D>,<Abs, real,20>,<Ord, real,15>},
{<Id_Point, char(10), E>, <Abs, real, 18>, <Ord, real, 3>}};

H_ Road _Segments = {<Id_seg, int>, <first_seg, char(10)>, <last_seg, char(10)>};
B_ Road_Segments= {{<Id_seg, int, 1>, <first_seg, char(10), A>, <last_seg, char(10), B>},
{<Id_seg, int, 2>, <first_seg, char(10), A>,<last_seg, char(10), D>},
{<Id_seg, int,3>, <first_seg, char(10), E>,<last_seg, char(10), A>},
{<Id_seg, int, 4>,<first_seg, char(10), B>,<last_seg, char(10), D>},
{<Id_seg, int, 5>,<first_seg, char(10), C>,<last_seg,char(10), E>},
{<Id_seg, int, 6>,<first_seg, char(10), E>,<last_seg, char(10), D>},
{<Id_seg, int, 7>,<first_seg, char(10), D>,<last_seg, char(10), A>},…}; /* *The segment AD is different from the segment DA Because the departure and arrival segments are different* */

H_Roads= {<Id_path,int>,<Road,SET(int)>} ;
B_Roads= {{<Id_path,int,10>,<Road,SET(int),{1,4}>}, /* Road {A,B,D} */
{<Id_path,int,20>,<Road,SET(int),{3,2}>}, /* Road {E,A,D} */
{<Id_path,int,30>,<Road,SET(int),{5,3}>}, /* Road {C,E,A} */
{<Id_path,int,40>,<Road,SET(int),{5,3,1,4}>}, /*Road {C,E,A,B,D} */ …}

H_Polygons= {<Id_poly,int>,<Polygon,SET(int)>} ;
B_Polygons={{<Id_poly,int,100>,<Polygon,SET(int),{1,4}>} , /* Triangle ABD */
{<Id_poly,int,200>,<Polygon,SET(int),{5,6,7}>}} /*Polygon CEDA*/ }

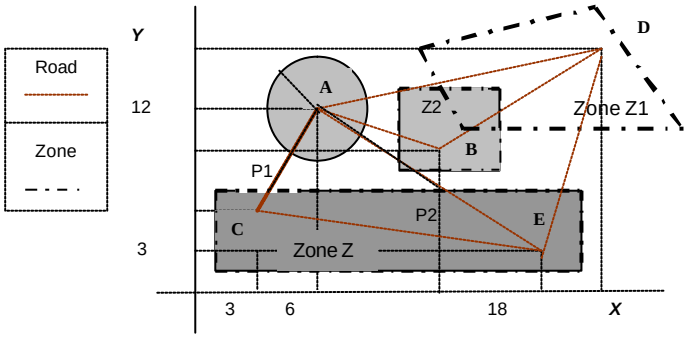

Figure 2 Road network

Let us note in this example (Figure 2) that we have considered data types related to the management and processing of geometrical data in a network of roads. Indeed, and in order to take completely into account the quantity of available data in the database *ROAD_BASE*; it is necessary to express relative queries to many situations in the road network such as:

(1) *What are the respective coordinates of the cities A and C?*
(2) *What is the distance between the cities A and E?*
(3) *What are the cities belonging to zone z?*
(4) *What are the cities not belonging to zone z?*
(5) *What is the transitive closure of the road set in zone z do not cross points of the line Y=3? Deduce the nature of the possible polygons?*
(6) *What are the separate roads connecting the cities A and B?*
(7) *What are all the roads that cross the cities A, B and C?*
(8) *What is the shortest road connecting zones z and z1?*
(9) *What are all the roads connecting zone z and z1 and not passing via zone z2?*
(10) *What is the shortest circuit accross all the cities? Etc…*

The resolution of such queries requires:

- on the one hand the choice of a data model capable to represent complex data types (e.g.: roads, convex regions, circuits) and several situations in a road network (e.g.: separate roads, the shortest road, roads in a region, intersection of regions, the circuit accross all the cities, the cities not belonging to a road etc.) and
- on the other hand the definition of a query database language which can deal with the quantity of available numerical data in such applications.

3. Date and Darwen's Approach

Before starting the resolution with A*, we present in this section Date and Darwen's formalism as well as *A*-algebra proposed in [1, 2]. Date and Darwen consider that SQL3 does not correspond to an object/relational language in terms where the notions of objects and nested tables consist of a logical confusion between the **class type**, defined in Object Oriented languages, and the **relation concept** in relational databases (i.e. *relation = class*, the equation is confused). Indeed, Date and Darwen note that the class is semantically equivalent to the domain or type and criticize research studies on the evolution of the Codd's relational model approaching a relation to a class. So, unlike the data model developed by the group ODMG in [6] and where the class concept is inherited from object-oriented languages, Date and Darwen propose a convivial object/relational extension of the Codd's relational model via domain or type generators (i.e. extensions to object characteristics are realized on the basis of the type theory and where equation *Class =Domain* is adopted).

Date and Darwen propose a relational algebra named, ***A***, slightly different from Codd's studies [5] in terms where it is based on the first order logic calculation. Some operations of the Codd's relational algebra have been revisited for the coverage of the new orientation of *A*-algebra. Indeed, a relation *r* is defined with its heading *Hr* and body *Br*. The heading represents the schema of the relation *r* and is defined as a set of couples *<a,T>* (with *a* an attribute and *T* the attribute type), while the body *Br* is the set of tuples. A tuple *t* being defined as a set of triplets *<a,Ti,v>* with *<a,Ti>* is the *Hr* element and *v* the value of the attribute *a*.

3.1. Formal definitions

Let be a relation *r*, an attribute *a*, a type *T* of the attribute *a* and *v* a value of the type *T*;

(a) Heading: A heading *Hr* is a set of ordered pairs *<a, T>* for each attribute *a* of *r*. So, two pairs *<a1,T1>* and *<a2,T2>* of *r* are such as *a1? a2* (i.e.: the names of attributes are different).

Example: The *heading of the relation Points quoted above is:*

H_Points *= {<Id_Point, char(10)>, <Abscissa, real>, <Ordinate, real>};*

(b) Tuple : Let *tr* be a tuple and *Hr* a heading; The tuple *tr* is a set of ordered triplets *<a,T,v>* where each attribute *ai* of *Hr* is associated with a triplet *<ai,Ti,v>* .

Example: *t1={<Id_Point,char(10),C>,<Abscissa, real,3>,<Ordinate, real,17>} ;*

t1 is a tuple of the relation Points realizing the city C which is located at coordinates (x=3,y=17).

(c) Body: A body *Br* of a relation *r* is a set of tuples *t*. However, there may be tuples *tj* corresponding to the heading *Hr* without *tj Î Br*.

Example: *Let tj*= {<Id_Point, char (10), X>, <Abscissa, real, 40>, <Ordinate, real, 28>},

We notice that the tuple *tj* corresponds to the heading *H_Points* in the relation *Points* defined in the example *ROAD_BASE* above; but *tj* does not belong to the body *B_Points* of the relation.

Remarks:

(1) Each heading *Hr* and body *Br* is viewed as a set.

(2) A subset of heading *Hr* (respectively of body *Br*) is heading *Hr'* (respectively of body *Br'*).

The object/relational *A*-algebra, defined in [1,2], allows a logic calculation of first order on the heading *Hr* independently from the one carried out on the body *Br*. Indeed, each algebraic operator in *A*, applied to a relation *r*, considers semantic actions on *Hr* different from those applied to *Br*.

3.2. Operators of the *A*-algebra

A-Algebra is essentially based on the set theory and consists of five basic operators *AND, OR, NOT, RENAME, REMOVE* and of two derived operators *COMPOSE* and *TCLOSE*. The macro operator COMPOSE considers the composition of relations as a generalization of the composition of functions. The operator *TCLOSE*, based on the Codd's algebra defines explicitly the operator of transitive closure. Let us consider two relations *r* and *s* such as $r \equiv (Hr, Br)$ *and* $s \equiv (Hs, Bs)$ with *Hr*, *Hs* and, *Br*, *Bs* respectively headings and bodies of relation's *r* and *s*.

AND Operator. The *AND* operator, is a conjunction of two relations *r1* and *r2*. The heading *Hs* of the resulting relation *s* is the union of the respective headings *Hr1* and *Hr2* of the relations *r1* and *r2* while the body *Bs* of *s* is a set built by conjunction of some tuples in the respective bodies *Br1* and *Br2* of the relations *r1* and *r2*. We note that this operator corresponds, in Codd's algebra, to a natural join of the relations' *r1* and *r2*.

$s \leftarrow r1\ AND\ r2$; $Hs = Hr1 \cup Hr2$; $Bs = \{ts / \exists tr1, \exists tr2((tr1 \in Br1) \wedge (tr2 \in Br2) \wedge (ts = tr1 \cup tr2))\}$

OR Operator. The *OR* operator, is a relational disjunction, generalization of the operation *Union* in the Codd's relational algebra. So, *Hs*, the heading of the relation result *s*, is the union of the headings *Hr1* and *Hr*2 of the input relations *r1* and *r2*. The body *Bs* of *s* is a set corresponding to the disjunction of tuples in the respective bodies *Br1* and *Br2* of the relation's *r1* and *r2*.

$s \leftarrow r1\ OR\ r2$; $Hs = Hr1 \cup Hr2$; $Bs = \{ts / \exists tr1, \exists tr2(((tr1 \in Br1) \vee (tr2 \in Br2)) \wedge (ts = tr1 \cup tr2))\}$.

The operator *OR*, in Date and Darwen's *A*-algebra, allows to treat the headings and bodies separately and has not an equivalent operator in the Codd's relational algebra.

NOT Operator. The operator *NOT*, expresses the complement of a relation *r* noted (*Hr*, *Br*). The heading *Hs* of *s* is equal to the heading *Hr* of *r*; while the body *Bs* of *s*, contains all the tuples *ts*, which do not belong to *Br*, body of *r*.

$s \leftarrow NOT(r)$; $Hs = Hr$; $Bs = \{ts / \forall tr((tr \notin Br) \wedge (ts = tr))\}$ *tr being a tuple belonging to Br.*

Operator RENAME. The operator *RENAME* allows the renaming of attribute named *a* in *r* by another attribute called *b* in the resulting relation *s* without changing its type *T*. So, the heading *Hs* of *s* is identical to the heading *Hr* of *r* except the pair <*a*,*T*> that is replaced by <*b*,*T*>. The body *Bs* is formed by the set of tuples *tr* in *Br* where <*b*,*T*,*v*> replace all the triplets <*a*,*T*,*v*>.

$s \leftarrow r\ RENAME(a,b)$; $Hs = \{Hr - \{<a,T>\}\} \cup \{<b,T>\}$;

$Bs = \{ts / \exists tr, \exists v((tr \in Br) \wedge (v \in T) \wedge (<a,T,v> \in tr) \wedge (ts = \{tr - \{<a,T,v>\}\} \cup \{<b,T,v>\}))\}$;

We notice that the operator *RENAME* is not necessary in Codd's algebra because it does not act on the semantics of the concrete database.

REMOVE Operator. The operator *REMOVE* generates a relation by eliminating a given attribute *a*, of a relation *r*. This operation is equivalent, in Codd's algebra, to the projection of *r* on all the attributes of *r* except for a given attribute *a*. So, the heading *Hs* is equal to *Hr*, the heading of *r*, minus the pair < *a*, *T* >. In this case, the body *Bs* of *s*, is a subset of tuples *tr* of *r* corresponding to the heading *Hs*. The first order logical calculation, well adapted to *A*-algebra, the operator *REMOVE* acts separately on heading *Hr* and the body *Br* of the relation *r*. This possibility of calculation allows enhancement of the possibilities of transformation of database scheme and supplies a better exploitation of concrete relations out of the headings, which they support.

$s \leftarrow r\ REMOVE\ a$; *where* $<a,T> \hat{I}\ Hr$, $Hs = Hr - \{<a,T>\}$;

$Bs = \{ts / \exists tr, \exists v((tr \in Br) \wedge (v \in T) \wedge (<a,T,v> \in tr) \wedge (ts = tr - \{<a,T,v>\}))\}$;

COMPOSE Operator. The ***COMPOSE*** operator, is defined by the combination of the operators ***AND*** and ***REMOVE*** such as:

$s \leftarrow r1 COMPOSE\ r2\ ;\ s \equiv (r1\ AND\ r2)\ REMOVE\ an...REMOVE\ a2\ REMOVE\ a1\ ;$

$Hs=\{(Hr1 \cup Hr2)-\{<a1,t1>,<a2,t2>,...,<an,tn>\}\}\ ;$

$Bs=\{ts/\exists tr1,\exists tr2,\forall v \in ti(((tr1 \in Br1) \vee (tr2 \in Br2)) \wedge (ts=(tr1 \cup tr2)-\{<ai,ti,v>/ai \in \{a1,...,an\}\}))\}.$

Notes:

- *a1, a2, … an* are the common attributes to the relations *r1* and *r2* (n > =0).
- The ***COMPOSE*** operator does not exist in Codd's algebra.
- The order of the ***Remove*** in the definition of ***COMPOSE*** operator is not important because it is concerned with the same structure of the set type.
- If n=0 then (*r1* ***COMPOSE*** *r2*) ≡ (*r1* ***AND*** *r2*) and the relation (*r1* C *r2*) is included in the (*r1* ***AND*** *r2*) body; where (*r1* C *r2*) is the cartesian product of *r1* by *r2* in the Codd's relational algebra.

The body *Bs* of the result relation *s* in the case of the operators *AND* and *OR* cannot contain tuples *tr* already existing in one of the operand relations *r1* or *r2*. The heading *Hr* is defined by *Hr = Hr1 È Hr2*.

TCLOSE Operator. Compared to the Codd's relational algebra, *A*-algebra includes an explicit operator for the transitive closure named ***TCLOSE***. Indeed, given that a relation *r* containing two attributes *X* and *Y* of type *T*, the transitive closure ***TCLOSE (r)***, following attributes *X* and *Y*, is a relation *r+* whose the heading *Hr+* is the same of the heading *Hr*. On the other hand the body *Br+* is such as:

$r+ \leftarrow TCLOSE\ \ r\ ;\ Hr+=Hr\ ;$

$Br+=Br \cup \{tuple\ t/\forall\ sequence\ S,$

$(S=\{<X,T,x>,<Y,T,z1>\};\{<X,T,z1>,<Y,T,z2>\};.;\{<X,T,zn>,<Y,T,y>\}) \in Br$

$\Rightarrow((t=\{<X,T,x>,<Y,T,y>\}) \wedge (t \in Br+))\}$ with three propositions (1), (2) and (3) as being equivalent.

(1) The tuples $<X,T,x>,<Y,T,y> \in Br+$

(2) There is a sequence of values *z1,z2,…,zn* having the same type *T* such as:
$\{<X,T,x>,<Y,T,z1>\};\{<X,T,z1>,<Y,T,z2>\};.;\{<X,T,zn>,<Y,T,y>\} \in Br$

(3) There is a way between tuples $<X,T,x>$ and $<Y,T,y>$ in the body *Br*.

Notes:

- for the operations (*r1 OR r2*) and (*r1 AND r2*), $((<a,T1> \in Hr1) \wedge (<a,T2> \in Hr2)) \Rightarrow (T1=T2)$;
- the operation (r *RENAME(A,B)*), $((<a,T> \notin Hr) \vee (<b,T> \in Hr)) \Rightarrow (r=r\ \ RENAME\ (a,b))$.
- in the case of *TCLOSE*, $(Br+=Br \cup E) \Rightarrow (Br \subset Br+)$ *with E: set of the added tuples.*

4. A*: New algebra for Object/Relational Model

In this paper, we consider that a domain or type is specified by an operator *Op* which would be equivalent, in our case, to a function that returns a value belonging to a domain or type. In the set of the triplets *{<Op,TOp,Op(p1:t1,p2:t2,...,pn:tn)>}* *Op* represents a function, *TOp* the type returned by *Op* and *Op(p1,…)* the function signature. More generally, $Hr=\{<a1,t1>,<a2,t2>, \ldots, <an,tn>, <Op1,Top1>, \ldots,<Opn,TOpn>\}$ is a header where *a1,a2,...,an* are attributes associated with the relation *r; t1, t2, …, tn* types assigned respectively to attributes ai; *Op1,Op2,…,Opm* are the operators domains definitions. *TOp1, TOp2,…, TOpm* are the respective types returned by the operators *Opi, i = 1,…m.*

4.1. Definitions

Comparing with notions presented in Section 3, the concepts of heading, of tuple and of body are redefined according to our approach.

Definition 1. A heading *Hr* is a set of ordered pairs <*X,Tx*> where *X* is an attribute *a* or an operator *Op*. The type *Tx* is either a predefined type or a type of operator result.

Definition 2. Let *Hr* be a heading and t a tuple; The tuple *t* is a set of ordered triplets <*X, Tx, v*> corresponds to each attribute or operator *X* in *Hr*. In this case where *X* is an operator *Op*, the value v is defined by $v = Op([p1],...,[pn])$.

Definition 3. A body *Br* of a relation *r* is a set of tuples t containing tuples tj such as $tj = <Op, TOp, Op(p1,\ldots,pn)>$ where *p1,…pn* are parameters assigned by value or by variable.

Example of some types:

a. < /*operator *Op:*/ Define_Road, /* type *TOp* :*/Set (int), /* *Signature of Op:*/
   Define_Road (Id-departure: char (10), Id-arrival: char (10)) >
b. < Define_Road, set (int), Define_Road(A,C)> /* *A, C are cities, see Figure 2.* */
c. <inverse,real,inverse(x:real)> ≡ <inverse,real,(1/x)> /* *Parameters by variable* */
d. <inverse,real,inverse(4)> ≡<inverse,real,0.25> /* *Parameters by value* */

We note that the operator *Op* can return an abstract data type (a), an instance of type (b), a function (c) or a value (d) according to the input parameters.

Definition 4. An operator $Op(p_1 : t_1,..., p_n : t_n)$ is a scalar function or domain constructor.

Definition 5. An abstract data type, ***ADT,*** is a heading of a relation *r* named *Hr,* which contains at least a couple <*Op, TOp*> and where the operator *Op* is a generator of the domain *TOp*. The tuples *tr* corresponding to heading *Hr* are called ***instances.***

4.2 Corollaries

Following the definitions defined above we deduce the notions of instances for an abstract data type and equivalence between two tuples *T1* and *T2*.

Corollary 1. an instance of an abstract data type defined by the heading $H_{adt}=\{<a_1,T_1>,<Op,TOp>,...,<a_n,T_n>\}$ is a tuple *t* defined by:

$t=\{<a1,T1,v1>,<Op,TOp,Op(p1:t1,p2:t2,...p:t):t_{n+1}>,...,<an,Tn,vn>\}$.

Example : $T=\{<id\_ch,int,10>,<$ *Define_Road,* ***set****(int), Define_Road(A,D)*$>\}$. The road identified by *id_path =#10* between the city A and the city D is {A,B,D} ; /* See Figure 2. */

Corollary 2. Let *H* be a heading defined by $H=\{<Xi,Ti>\}$ where $Xi \in \{Attribute, Op\}$. Two tuples *T1et T2* corresponding to the heading *H* are equivalent if and only if:

$\forall <a,t,v> \in T_1, \exists <b,t,u> \in T_2\ ((a=b) \wedge (v=u))$ *where* $\forall <Op1,TOp1,Op1(p1:t1,p2:t2,...,p:t)> \in T_1,$
$\exists <Op2,TOp2,Op2(q1:t1,q2:t2,..,qm:tm)> \in T2,$
$\Rightarrow ((Op1=Op2) \wedge (TOp1=TOp2) \wedge (Op1(p1:t1,p2:t2,...,p:t) \equiv Op2(q1:t1,q2:t2,..,qm:tm)))$

4.3 Example of an extended data model

Let us consider the illustrative example of the section 2, the database *ROAD_BASE** in the new object/relational model is as follows:

ROAD_BASE*

Points = RELATION{ Id_Point char(10), Abscissa real, Ordinate real } KEY{ Id_Point } ;
Road_Segments = RELATION{Id_seg int, first_seg char(10), last_seg char(10)}
KEY{ Id_seg };
Roads = RELATION{Id_path int, OP[1] Road(Id-departure char(10),Id-arrival char(10)):
SET(int) KEY { Id_path};
Polygons = RELATION{ Id_poly int, OP Polygon(Id_seg-departure int,Id_seg-arrival int):
SET(int) KEY { Id_poly } ;

Notes:

- The operator ***Road*** allows finding all the possible roads between the cities identified respectively by ***Id_departure and id_arrival.*** This operator generates the set of segments, which form the parts of the road in question: ***SET (int)← Road (Id_departure: char (10), Id_arrival: char (10)).***
- The operator **Polygon** allows building the domain of all the possible convex figures between segments, identified respectively by ***Id_seg-departure and id_seg-arrival.*** This operator returns a set of segments that form the convex space or the polygon in question: ***SET (int)← Polygons (Id_seg-departure: int, Id_seg-arrival: int).***
- The name of the operator *Op* represents the domain and *TOp* is the representation of the domain in the object/relational database. Indeed, in the case of the domain *Polygons* defined by the operator *Polygon*, Type *TOp* is equivalent to the type **SET (int)** and the values *v* are such as *v = Polygon (Id_seg-departure: int, Id_seg-arrival: int).* For example, the polygon CEDA in the figure 2 is defined by the tuple t as follows: ***t={<Id_poly, int, 200>, <Polygon, SET(int), {5,6,7}>}*** */* see Figure 2. */*

4.4. Basic operators of extended model

The extended object/relational model requires new algebra for the support of the domains generated by operator *Op*. Indeed, in our approach, we have exploited the Date and Darwen's studies on the *A*-algebra and previous work [7] on the operator bound by the procedures "*EXT*" to specify and define *A**. This latter is based on same operators as *A*-algebra.

Operator NOT*. The operator NOT* allows the elaboration of complement to the relation *s* relatively to another relation *r*.

$s\ NOT^*\ r\ ; Hs=Hr\ ;\ Bs=\{ts/\exists tr((tr\in Br)\wedge(ts\neq tr))\}$

Operator REMOVE*. The operator **REMOVE*** allows to remove, from the body *Br*, all the semantic entities derived from an operator *Op*. In case where the element to be removed is an attribute, **REMOVE*** is equivalent to the operator **REMOVE** in *A*-algebra.

$s \leftarrow r\ \ REMOVE^*\ X\ ;\ where\ (<X,T>\in Hr)\wedge(X=\{A,Op\}))$

$Hs=Hr-\{<a,T>\}\ ;$

If X=a then begin

$Hs=Hr-\{<X,T>\}\ ;$

$Bs=\{ts/\exists tr,\exists v((tr\in Br)\wedge(v\in T)\wedge(<X,T,v>\in tr)\wedge(ts=tr-\{<X,T,v>\}))\}\ ;$

Endif Else(X=Op) begin

$Hs=Hr\ ; Bs=\{ts/\exists tr((tr\in Br)\wedge(<X,T,X(p1:t1,p2:t2,..,p:t):t_{n+1}>\in tr)$

$\wedge(ts=tr-\{<X,T,X(p1:t1,p2:t2,..,p:t):t_{n+1}>\}))\}$

endelse

Operator TCLOSE. This operator allows the semantic improvement existing in the object/relational database. Indeed, the body *Br* of a relation *r* is increased by new tuples expressing all information deduced by transitive closure.

$r+\leftarrow TCLOSE^*\ r\ ;\ Hr+=Hr\ ;$

$Br+=Br\cup\{tuple\ t/\forall\ séquence\ S,((S\in Br)\Rightarrow(t=\{<X,T,x>,<Y,T,y>\})\wedge(t\in Br))\}\ With$

$S=\{<X,T,x>,<Y,T,z1>\};\{<X,T,z1>,<Y,T,z2>\};.;\{<X,T,zn>,<Y,T,y>\}$

Or $S=\{<Op1,T,v>,<Y,T,v1>\};\{<Op2,T,v1>,<Y,T,z2>\};.;\{<X,T,zn>,<Y,T,y>\}$

In this case the heading *Hr* of *r* contains the operators *Ops*, **TCLOSE*** expresses a transitive closure between the operators or functions.

Operator RENAME*. In addition to renaming an atomic attribute this operator is used to change of the name of an operator *Op*. The heading *Hs* of *s* is identical to the heading *Hr* of *r* except the pair $<O1p,TO1p>$ that is replaced by $<O2p,TO2p>$. The body *Bs* is formed by the set of tuples *tr* in *Br* where $<O2p,TO2p,v2>$ replace all the triplets $<O1p,TO1p,v1>$. This operator can be written:

$s \leftarrow r\ RENAME(Op1,Op2);\ Hs = \{Hr - \{<Op1,T1>\}\} \cup \{<Op2,T2>\}.$

Notes:

- $(AND^* \equiv AND), (OR^* \equiv OR)$ and $(COMPOSE^* \equiv COMPOSE)$.
- Each operator of *A** is followed by the character star (*) to mean that the operator belongs to the extended algebra.

4.5 Extension operators

The definition of an object/relational database contains relations having headings of type $H = \{<a_1,t_1>,<a_2,t_2>,...,<a_n,t_n>\!\!<Op_1,TOp_1>,...,<Op_m,TOp_m>\}$. So, the interrogation and exploitation of such database require appropriate algebraic operators besides those used for an object/relational model. Several variants of the extension operator that we have proposed in [7] have been adapted and integrated into the specifications of *A*-algebra*.

4.5.1. New type extension

The operator of new type extension allows the modification of the heading of the relation *r* by inserting a new attribute *an+1* and its type *tn+1* in the heading *Hr* of *r*. The body *Bs* of *s* is such that *Bs* contains for each tuple *tr* of *r*, the triplet $< an+1\ ,\ tn+1,\ null >$ where the constant ' null ' belongs to any system or user type. The operator in this case is equivalent to the operator ALTER in SQL3.

$s \leftarrow EXT\ r\ ADD(a_{n+1} : t_{n+1});\ Hs = Hr \cup \{<a_{n+1} : t_{n+1}>\};$

$Bs = \{ts / \forall tr \in Br(ts = tr \cup <a_{n+1},t_{n+1},null>)\}$

$Hr \quad being \quad Hr = \{<a_1,t_1>,<a_2,t_2>,...,<a_n,t_n>\}\ "\ i\ \hat{I}\ \{1,\ldots,cardinality(r)\},$

$"\ (tr\ \hat{I}\ Br)\ (tr \leftarrow tr\ \grave{E}\ <an+1,tn+1,null>)$.

Example of new type extension

Points ← EXT Points ADD Color :int ; Relation Points becomes:

Points = RELATION Id_Point char(10), Abscissa real, Ordinate real, Redius real,Color int} KEY { Id_Point }; with init(Points) is defined by : init(Points) {for i=1 to card (Points) do Points.Color=*null*;}

4.5.2. Extension of computed type

The extension of computed type operator allows on the one hand to modify the heading of the relation *Hr* by inserting the new attribute *an+1* to *Hr*, and on the other hand to define the tuples results of *Op* application in *r;* the type of the attribute *an+1* being defined by the type of $Op(p_1 : t_1, p_2 : t_2,..., pn : tn)$. The value *v* in the triplet $< a_{n+1},TOp,v >$ is expressed by $v = Op([p_1],...,[p_n])$ where *[pi]:* is the value of *pi*. The expression of type extension is as follows:

$s \leftarrow EXT\ \ r\ \ ADD\ \ a_{n+1}\ \ BY\ \ Op(p1 : t1, p2 : t2,..., pn : tn)$ where:

$Hr = \{ < a1,t1 >,< a2,t2 >,\ldots,< an,tn > \}$ and $< an+1,TOp > \notin Hr$

$Hs = Hr \cup \{ < an+1,tn+1 > \}$

$Bs = \{ ts / \forall tr, \exists v((tr \in Br) \wedge (ts = tr \cup \{ < an+1,tn+1,v > \}) \wedge (v = Op([p1],[p2],\ldots,[pn])\}$.

Example of extension of computed type

Points ← EXT Points ADD Redius BY calcul_redius(Abscissa:real, Ordinate:real): real;
Relation Points *becomes*
Points = RELATION{Id_Point char(10), Abscissa real, Ordinate real,Redius real, KEY { Id_Point } ; *with the operator* calcul_redius *defined by*: calcul_redius(Abscissa:real, Ordinate:real): real
begin
    Redius=sqrt (Sqr(Abscissa)+Sqr(Ordinate));
    Return (Redius) ;
end

4.5.3. Global operator extension

A global operator extension acts on the heading *Hr* or the body *Br* of the relation *r* and allows defining new domains, new constraints of integrity defined by the operator *Op* or reorganizations of the relation *r* and generates as a result, a relation defined by $Hs = \{ < a1,t1 >,< a2,t2 >,..,< an,tn >,< Op1,TOp1 >,...,< OpL,TOpL > \}$. The operator $\boldsymbol{Ext}^{G}$ can be i) internal if *Op* is specified when it is called; ii) external if *Op* is DBMS predefined operator.

$s \leftarrow EXT^{G}\ r\ BY\ Op(\ p1 : t1, p2 : t2,\ldots..,pn : tn) : tn+1$

[begin / Body of Op which can be written in a high-level language or in SQL */ end.]*

$Hs=\{<b1,t1>,<b2,t2>\ldots,,<bm,tm>\}$ *with* $bi \hat{I} \{ai,Op\}$,

$Bs = \{ ts / \exists T \in \{ t1,t2,\ldots,tn \}, \exists X \in \{ b1,b2,\ldots,bm \})\}$

$((< X,T > \in Hs) \wedge (ts = \{ < X,T,v > \}) \wedge (v = Op(\ p1 : t1,..,p : t\ )))\}$

Case (a): for this type of extension, the operator *Op* allows to define a domain by the operator *Op* which is integrated into the scheme of a given relation (i.e.: $Hr=Hr \cup \{<Op,TOp>\}$, *TOp* is the *Op* result type). Besides, the application of several operators to a scheme of a relation *r* allows to consider various complex types from *r* without changing its dimension.

Example of domain of points in Square C

Points[1] ← EXT Points BY InSquare(Id_Point :int,C :Polygon) :Boolean ;
Relation Points is:
Points = RELATION{Id_Point char(10), Abscissa real, Ordinate real, Redius real, Color int, OP InSquare(Id_Point :int,C :Polygon) :Boolean} KEY { Id_Point } ;
with InSquare (Id_Point,C) defined by :
InSquare(Id_Point :int,C :Polygon)
begin if id_Point $\hat{I}$ (square C) then return 1 else return 0 ; end

Case (b): this type of extension uses aggregation, which allows restructuring the relation r. Let us; consider the case of a grouping of attributes *a*1, *a*2. *a*n in a new attribute *b*. Indeed, in the example below, the domain «AngleTeta» replaces the attributes «Abscissa» and «Ordinate» in the Relation Points. The dimension of the result relation *s* is inferior to the rank of the operand relation *r:*

$Hr=\{<a_1,t_1>,<a_2,t_2>,..,<a_n,t_n>\}$;

$Hs=\{<b_1,t_1><b_2,t_2>,...,<b_m,t_m>,<Op,TOp>\}$; with $m<n$, $bj=a1\ldots ap$ and $j=m$, and $p=n$ bj being an aggregation of several attributes ai

[1] init (Points) is a procedure of initialization existing in the dbms

Example of restructuration of the relation Points

Points ← EXT^G Points BY AngleTeta(X:real,Y:real):real ;
Points = RELATION{Id_Point char(10), Redius real, Color int, OP
AngleTeta(Abscissa:real,Ordinate:real): real} KEY { Id_Point } ;
AngleTeta(Abscissa:real, Ordinate:real)
Begin for i=1 to card(operande relation) do {
compute the angle Teta fromAbscissa and Ordinate,
remove the coordinates Abscissa and Ordinate ,
insert tuple of the operand relation into Points; }
end

Case (c): *m>n* and *ai Î{b1…bm}*; that case uses a redefinition of a user data type in its basic types [21,23].

Example: Redefinition of a domain D

- Let the relation Points defined by: Points(Id_Point char(10), Redius real,Color int, OP AngleTeta(Abscissa:real, Ordinate:real)); we have: TYPE Segment {begin: char(10) , end: char(10) }
- Points ← EXT^G Points BY S :Segment; Relation Points becomes: Points = RELATION{Redius real,Color int, *Id_Point char(10), S :Segment*, OP AngleTeta(Abscissa:real, Ordinate:real)} KEY { Id_Point } ;
- Points ← EXT^G Points BY Display(s:Segment); Relation Points becomes : Points = RELATION{ Id_Point char(10), Redius real,Color int, S-Begin :char(10), S-End : char(10), OP AngleTeta(Abscissa:real, Ordinate:real), OP *Display(s:Segment)}* KEY { Id_Point }.

4.5.4. Local operator extension

Unlike the global operator extension, this type of extension acts **exclusively on the body *Br*** of a relation *r* and allows consequently expressing queries on tuples *t* or entities *e* having the characteristics *Ci* defined by an operator *Op*.

s ← EXT^L ALL r BY Op(p1:t1,p2:t2,…,pn:tn):tn+1
[begin / Body of Op which can be written in a high-level language or in SQL */ end]*

$Hs = Hr \cup \{ < Op, TOp > \}$; $Bs = \{ts / \forall tr \in Br\ ts = tr \cup \{ < Op, TOp, Exp(Op) > \}$; *where Exp (Op)*: the value, which defines the operator *Op* for a given query. *TOp*: specified by the expression Exp (*Op*). Figure 3 depicts extension operators.

Example of local extension operator

Neighbourhoods ← **EXT^L ALL** Points **BY** Near_of(P :char(10), d :real) :Set (int) ;

Let us consider the relation Points of the data base ROAD_BASE*:

Points = RELATION{ Id_Point char(10), Abscissa real, Ordinate real} KEY { Id_Point } ;

So, neighbourhoods = RELATION{Id_Point char(10), Abscissa real, Ordinate real, OP
Near_of(P: char(10), d : real) : Set (int) } KEY{ Id_Point }
are declared by:
VAR pp tuple(Id_Point char(10), Abscissa real, Ordinate real)

/ Definition of the point A(6,12) */* pp.Id_Point ='A'; pp. Abscissa=6; pp. Ordinate=12 ;
Neighbourhoods ← EXT^L ALL Points BY Near_of('A', 2)
/ Points in a circle of centre* A(6,12) *and of* Radius 2; see Figure 2. */
/ Exp(Op) is defined in that case by Exp(Op) ={(x,y)/ $(x-6)^2+(y-12)^2 =4$} */*
with **Near_of(P :char(10), d:real)** *defined by:*
Near_of(P :char(10), d :real) */* *d: distance* */*
Begin
Var E :set(int) ; E← F ;
for i=1 to count(Points) do Begin

read tuple(i) ;
if (P-tuple(i).id_point)<= d then t← tuple(i).Id_Point ;E← E È {t};
insert tuple(i) of the relation Neighbourhoods ; endif
endfor
return (E) ;
end

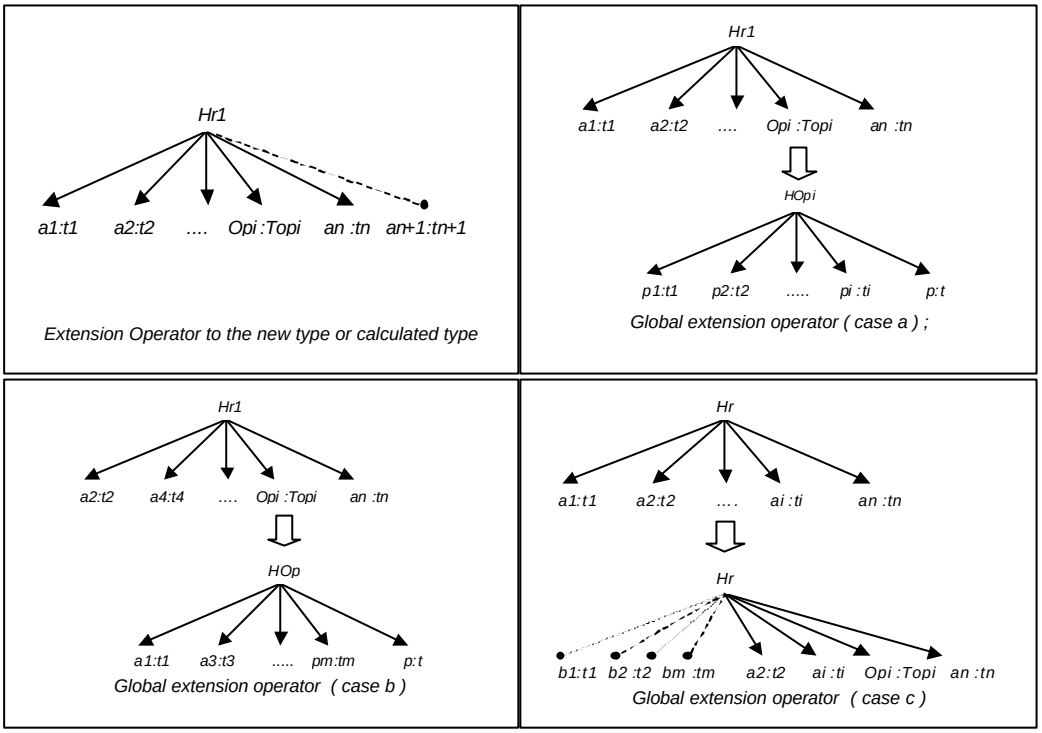

Figure 3. Various types of extension to the operators Op .

4.6. Advanced operators

4.6.1 The operator of relation extraction

The operator of relation extraction builds a relation from one or several operand relation *r1, rp*. Indeed, the operator *Op* applied to headings *Hr(i)* allows to define the result heading: Hs={<a_1,t_1>, <a_2,t_2>, …,<a_n,t_n>, <Op_1,Top_1>, <Op_L,Top_L>};the operators Op_1, …, Op_L, in the relation result s, provide the possibility of the expression of new situations in the database. Moreover, the operator ***EXTRACT*** allows also the extraction of tuples *tr* by the operator *Op* (*p1,p2...p)* according to functional dependences linking the operand relations *r1,..., rp*: *s← EXTRACT r FROM r1,r2,…, rp BY Op(p1:t1, p2:t2, …,p : t) :tn+1;* $Hs = \{\forall < Xi,Ti >, \exists < a,T > \in Hri, \exists < Op,TOp > \}$ *with* <*Xi,Ti*>=<*a,T*> Ú <*Xi,Ti*>=<*Op,Top*>.

$Bs = \{ ts / \exists tr \in \bigcup_{i=1}^{n} Hr_i \quad ((ts \subset tr) \wedge (ts \, compliant \, with \, Hs)\}$ with *Hr (i)*, the heading of the relation *r(i)* and (*i=1..n*). The operator *Op* allows to extract entities of type *tn+1*, according to the heading *Hr* of the operand relation *r*, from the relations *r(1), r(2),…, r(n)*. The attributes <*Xi, Ti*> of the result relation *s* are either attributes already existing in the values relations' *r(i)* or attributes <*Op,TOp*> generated by the operator *Op*.

Extraction of a relation by operator OP
Let us consider Points and Neighbourhoods two relations (4.5.3 (a)) and (4.5.4):

Points = RELATION{Id_Point char(10), Abscissa real, Ordinate real, Redius real, Color int, OP InSquare(Id_Point :int,C :Polygon) :Boolean } KEY { Id_Point } ;
Neighbourhoods = RELATION{ Id_Point char(10), Abscissa real, Ordinate real, OP Near_of(P: char(10), d :real) :Set (int) } KEY { Id_Point } ;

Now, let the following query: EXTRACT s FROM Points, Neighbourhoods By P_and_N(void)
with: P_and_N(void) Begin Points AND Neighbourhoods End
S = RELATION{Id_Point char(10), Abscissa real, Ordinate real, Redius real,Color int,
OP InSquare(Id_Point :int,C :Polygon) :Boolean OP Near_of(P :char(10), v :real) :Set (int)}
KEY { Id_Point } ; *which are points in the square C defined by the inequalities Cs{0<x<a,0<y<a}*
EXTRACT Q FROM S BY Squaree(Cs) :Set (int) ; with Square (Cs) defined by:
Square(Cs :set(int))
for i=1 to card(relation Points) do begin
Read tuple(i) ; T← tuple(i).Id_Point ; If (InSquare(T,Cs) then Bs ← tuple(i); end

4.6.2. Abstract data type generation

In an object/relational model the definition of the relation notion as a heading *Hr* and a body *Br* in the Date and Darwen's formalism [1, 2] has allowed to approach the relation notion like the domain set *Hr={<a1, T1>, <a2, T2>, <an, Tn>}* where the management of the body *Br* of the relation does not influence the heading calculation. The integration of Operators *Op* in the definition of an extended object/relational model expresses each heading *Hr* of a relation *r* with an abstract data type *Hadt={<A1, T1>, <Om, TOm>, <An, Tn>}*. The generation operator of types ***SHOW*** **(a)** expresses all the abstract data types in the heading *Hr = {< a1, T1 >, < Op, TOp >, < Om, Tom >, < an, Tn >}* following the nature of operator *Op* and its input parameters. Once the abstract data type or ADT has been generated, it is possible to extract ***SHOW*** **(b),** all the instances respecting the signature of such type or domain.

(a):): *s ← SHOW[ALL] ADT [<ADT_name>] ON r WHERE <conditions> with <conditions> an expression on the heading Hs and can be a definition by value or by variable parameters of the operator Op (p1:t1, p: t)*

$$Hs = \{< X_i, T_i > / \exists < Op, TOp > \in Hr, (< X_i, T_i > \in Hr) \Rightarrow ((< Op, TOp > \in Hs) \wedge (< Conditions > \equiv True))\}$$

$$Bs = \{ ts / \exists tr \in Hr \quad ((ts \subset tr) \wedge (ts \text{ compliant with } Hs))\}$$

(b): *s ← SHOW[ALL] [<instance_name>] INSTANCE ON <ADT_name> WHERE <conditions>*

$$Hs = H < ADT\_name > \equiv \{< X, T > / \exists < X, T > \in H < ADT\_name > \} ((X = Op) \wedge (T = TOp))\}$$

$$Bs = \{ts / ((ts \quad compliant with \quad Hs) \wedge (< Conditions > \equiv True)\}$$ *with <Conditions> an expression on the body Bs.*

Let be the relation Neighbourhoods in the section (4.4.4) :
Neighbourhoods = RELATION{ Id_Point char(10), Abscissa real, Ordinate real,
OP Near_of(P :char(10), v :real) :Set (int) } KEY { Id_Point } ;
SHOW ADT N_Proches **ON** Neighbourhoods */* Query SHOW without conditions. The following query expresses all the abstract data types defining, possible Neighborhood. */*

5. Extended relational algebra language (ERA*)

ERA* is an algebraic language based on the operators of *A**-algebra. Indeed, realization of the algebraic extension operators offers new functionalities in a database language. The definition of relation r and data logical calculation proposed in the Dates and Darwen's formalism on the object/relational models [1,2] leads:

- to enrich the object/relational model seen by Melton on the one hand;

- the reinforcement of data interrogation language by sophisticated operators for the resolution of some query classes on the other hand.

Consequently, the integration of possibilities offered by the language ERA* in SQL3 should improve this standard. To illustrate partially ERA*, let us consider the database ROAD_BASE* defined in section 4.3 and the set of queries of the illustrative example of the section 2. Our objective in the following is summarized in the presentation of the solutions of some types of queries by using both SQL3 and *ERA**.

5.1 Geometrical queries

(1) What are the respective coordinates of the cities A and C?
(2) What is the distance between the cities A and E?
(3) What are the cities belonging to the zone z defined with the rectangle {1<x<20;2<y<7}?
(4) And the cities not belonging to the zone z?

Query	SQL3	ERA*
(1)	**Select** Abscissa,Ordinate **From** Points **Where** (Points.Id_Point='A') OR (Points.Id_Point='C')	**Ext ALL** Points **BY** InTown(Id_Point:int,{A,C}):Boolean; Points **REMOVE*** Id_Point; SHOW ALL INSTANCE ON **Points;**
(2)	**Select** Distance(P1,P2) **From** Points P1, Points P2 Where **P1.Id_Point='A'** AND **P2.Id_Point='E'**	**Ext** Points **BY** Eval_dist(Departure:char(10), Arrival:char(10)):real, **SHOW ADT ON** Points Where **Eval_dist(x,y) AND x='A' AND y='E';**
(3)	**Select** P1.Id_Point **From** Points P1 **Where** (1<P1.Abscissa<20) **AND** (2<P1.Ordinate<7)	**(3.1) : Res** = **Extract** Point_in_zone **From** Points **By** inzone(p:char(10),S):boolean **;** **(3.2) : Res** = Res **Remove*** Abscissa, Ordinate **;** **ShowAll ADT On** Res **Where** true**;**
(4)	**Select** P1.Id_Point **From** Points P1 **Where Not inzone(**P1.Abscissa, P1.Ordinate);	**(ShowAll ADT On** Res(*) Where **Not** inzone(**p:char(10), S):boolean**;

Notes.

- In the case of simple questions, answers in *ERA** are less condensed and more complex to be expressed with regard to the language SQL3.
- The operator InTown(Id_Point:**int** ,s:**set**):**Boolean** allows to identify whether the city Id_Point belongs to the set of cities s
- Eval_dist(Departure:**char(10)**, Arrival:**char(10)**) expresses the distance between an arrival city and a departure one.
- Distance (P1 , P2) is a stored procedure or PSM [14] in the language SQL3.
- In addition tgo query resolution, the solution to the query (1) in language *ERA**, allows to enrich the relation Points by the operator InTown below.
  Points=**RELATION**{Id_Point:**int**,Abscissa:**int**,Ordinate**:int**,
  OP InTown(Id_Point:**int**,{A,C}):**Boolean** } **KEY**{ Id_Point }
- The operator InTown is necessary in other queries which make reference to any subset { A, C} like : What are the roads containing the cities A and C ?
- The S system represents the zone Z of the figure 2 where S={1< x <20 and 2<y<7}.
- The signature of operator inzone(p :char(10),S :char(10)) :boolean, S :char(10) :boolean ; express the following two propositions (p∈ S) and (p ∉ S) depending on returned value.

Treatment of the query (3)

***(3.1)** :Res=**RELATION**{Id_Point:int,Abscissa:int, Ordinate:int,**OP** inzone (C:char(10) ,S: char(10)): boolean } **KEY**{ Id_Point }*

***(3.2)** : Res=**RELATION**{ Id_Point:int, **OP** inzone(C:char(10),S:char(10)):boolean } **KEY**{ Id_Point }. The body B_Res of the relation Res above is: B_Res = {{<Id_Point, char(10),C>, <inzone,1>}, {<Id_Point, char(10),E>, <inzone,2>}};*
The result B_Res extension is {("C",1),("E",2)}

5.2. Topological Queries

What is the transitive closure of the road set in zone Z does not intersect with points ofine Y=3? What is the nature of the possible polygons? The steps of resolution of query (5), in the example depicted in Figure 4, are:

a. Searching for roads in zone Z
b. Determination of roads not passing via the point E
c. Determination of the transitive closure
d. Deduction of polygons

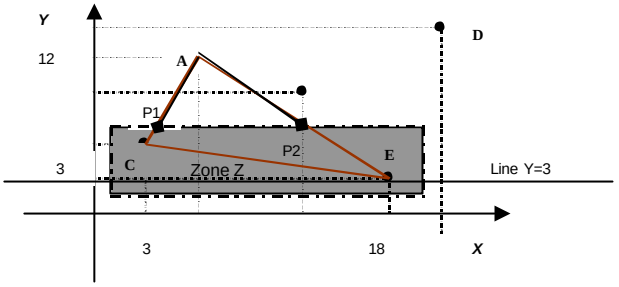

Figure 4. Topological situations.

Indeed, roads in the zone Z (figure belo w) are defined by the set S such that :
S={{P1,C},{P1,P2},{P1,C,E},{P1,P2,E},{C,E},{C,E,P2},{P1,C,E,P2},{E,P2}} ;(a)
So roads do not pass via points of the line Y=3 represent the set SS such that:
SS={{P1,C},{P1,P2}} (see figure above); (b)
so the transitive closure SS* is: SS*=SS È {P2,C}={{P1,C},{P1,P2},{P2,C}} ; (c)
We have deduced that there is only one polygon (P_1P_2C) of type triangle (d)

However, in the general case the resolution of query (5) has some particularities:
(1) The possibility of defining and extracting new domains from the concrete database already defined;
(2) The definition of infinite objects but logically finite
(3) The definition of non-specified property *a priori* in the scheme of the object/relational databas e (eg road length, the nearest road to a zone).

This highlights the choice of a database definition and interrogation language which can:
- generate types of complex data such as the set collections or the sets and their managements;
- express complex algorithms for the detection of the intersection points of roads or zones
- do the data logical computation to deal with the road network evolution in its topological aspect
- define a great number of possible situations for the road network management. etc…

Consequently and following the limits of stored procedures which are provided in the module SQL / PSM [14], it is important on the one hand to reinforce the object/relational model defined in [1,2] by the operators *Op (p1:t1 , p2:t2, pn tn) which* can express complex situations under the form of general algorithms and on the other hand to allow a relational computation in data domain in the case of complex queries.

6. Discussion

The DBMS based on relational model is widely used today. This is essentially due to the integration of a query language based on the relational algebra [5] which has undergone several extensions to improve the relational model. Thus, new algebras are defined to meet the

requirements of advanced databases for designers and users. In addition to the basic operations (i.e.: *Union, Difference, Cartesian product, Projection, Restriction, Join*), these algebras, qualified as extended algebra, group other derived operations (i.e.: *Intersection, Division, Complement, Unflat, External Join, Transitive closure, extension or others*). Consequently, our object/relational *A**-algebra consists of relational operators based on the logic of first order and algebraic extension operators. *A** is composed of both operators called relational expressed in first order logic and algebraic extension operators. In fact, a study of *A** completeness and consequently the elaboration of any language based on *A** (i.e.: *ERA** Language) depends on the object/relational queries space to be expressed. From a purely relational point of view, *A** is complete in the sense where it offers the possibility of expressing any relational query (see comparison in the *Table1*.). For an object or object/relational context, even *A** completeness was one of our main preoccupations the query language based on *A** (ERA*) should be more investigated. In fact, the expression of any object/relational query requires an object/relational query language being able to contain and implement program. The first elements of this language presented in this paper can be considered as a plat-form for the kernel of a strong and powerful query language to manipulate logically (or) symbolically complex objects. In an object/relational database scheme, a relation *r* of the heading *H* and body *B* is defined by: $H = \{< a_1,T_1 >,< Op,TOp >,...,< a_n,T_n >\}$ with *Op(x1,x2,…,xn):y* and *B={t/t is compliant with H;* with *t={a1,T1,v1>,<Op,Top,Op (p1:t1, p2:t2, …, pq:tq): tn+1>,…,<an,Tn,vn>};* the relation *r*, in that case, can be considered as a functional structure implying a part of the heading *H* with the rest of the heading *H*. The subset of attributes in *H* that determines the rest of the heading is called a key of the relation *r*. In the case of a functional dependence $a_1,a_2,\ldots,a_{n-1} \rightarrow a_n$ of the relation *r* expresses a function $f(a_1,a_2,\ldots,a_{n-1}) = a_n$. To show the interest of A* we compare basic operators of *A** with relational algebra while the extension operators of A* are compared on the one hand to the specification language of object scheme, ODL [6] and to the object/relational language SQL3 on the other hand.

Basic operators in A*. Compared to the Codd's relational algebra, *A*-algebra* contains five basic operators and two derived operators. We give in Table 1 the algorithmic structure to translate A* into relational algebra. The set of basic operators of *A** contains operators based on the first order logic (e.g. : ***And *, Or*, Not****) besides an operator of deletion ***remove**** and other one to rename the attributes ***rename****. This last operator requires a boring calculation in the case of an extended object relational model. Indeed, the change of the name of an operator Op implies the calculation of all the tuples of which the operator Op makes reference. The derived operators ***Tclose**** and ***Compose**** are based on the composition and transitive closure operations, defined by Codd. We notice that the A* operators are a more simply way to express queries that necessitate many Codd relational operators (see Table1).

Extensions operators in A*. The extension operators of the object/relational scheme are classified into three categories: i) extensions to new type, ii) extensions to operators (global or local) and iii) extension to extraction operators and/or data type generation. The extension operators defined in the section 4.5 are compared in Table 2 (a & b) with those of the language ODL (e.g., standard object scheme specification) and those of SQL3 (e.g., standard language for object/relational database).

A operators*	*Expression basing on relational algebra*
a) Basic operators	
*REMOVE**	**If** (∀ i ∈ {p+1,..,n}, ai is a basic type or an user type different from the type OP) **Then** $\Pi_{a1,...,ap}(r)$ **Else** (∃ i ∈ {p+1,..,n}, ai is of type Operator); 1. ∀ j ∈ {p+1,..,n} / aj is an operator Op do (*remove* OP*); 2. $\Pi_{a1,...,ap}(r)$; **endElse**
*AND**	**If** (∀ i ∈ {p+1,..,n}, ai is a basic type or an user type different from the type OP) **Then If** ((Hr∩Hs) ≠ ∅) **Then** (r **And*** s)=(r **X** s) **Else** ((r **And*** s) **remove*** (a1,…,ap)=(r **natural join** s) **Else** (∃ i ∈ {p+1,..,n}, ai is of type Operator); ∀ j ∈ {p+1,..,n} / aj is an operator **do** ∪ Op; (r **And*** s)=(r **X** s) giving res; res=res **union** (The operators Op) ; res=res **remove*** a1,…,ap; **endElse**
*OR**	**If** (∀ i ∈ {p+1,..,n}, ai is a basic type or an user type different from the type OP) **Then If** (Hr=Hs) **Then** (r **Or*** s)=(r **union** s) **Else** ((r **Or*** s) ≡((r **union** s) **remove* (**a1,…,ap) **Else** (∃ i ∈ {p+1,..,n}, ai is of type Operator); ∀ j ∈ {p+1,..,n} / aj is an operator Op do ∪ Op; (r **Or*** s)=(r **union** s)=res; res=res **union** (The operators Op) ; res=res **remove*** (a1,…,ap); **endElse**
*NOT**	**If** (Hr=Hs) **Then** (r **-** s)=(r **And *(Not***(s))) ;
*RENAME**	*Null*
(**b) Derived Operators**	
r Compose s*	**If** (∀ i ∈ {p+1,..,n}, ai is a basic type or an user type different from the type OP) **Then If** ((Hr∩Hs)≠∅) **Then** (r **Compose*** s)=(r **x** s) ; **Else** (r **Compose*** s)=((r **And*** s) **remove*** a1,…,ap); **Else**(∃ i ∈ {p+1,..,n}, ai is of operator type); **begin** ∀ j ∈ {p+1,..,n} / aj is of operator Op do ∪ Op; (r **compose*** s)=(r **x** s)=res; res=res **union** (The operators Op) ; res=res **remove*** (a1,…,ap); **endElse**
Tclose (r)*	**If** (∃ai∈ Hr,∃aj∈ Hr / (ti=tj)) **Then** r+=**Tclose***(r) ; r+=r+ **union** { Operators } ;

Table 1. A* versus relational Algebra.

A Extension Operators*	*Object / Relational DBMS (SQL3)*
(1) EXT r ADD(an : tn+1) BY Op(P1,:t1,p2:t2, …) : tn+1	▪ Add attribute an+1 of type tn+1 in the mapping of r ▪ Add values [An+1] such as: [An+1]=Op([p1],…,[pn]). Query SQL3 : ***Alter Table*** Br **add** an+1 **;** ***Update*** Br ***set*** Br.an+1= Op([p1],…,[pn])**;**
(2) EXT r ADD(an : tn+1)	▪ Add attribute an+by assigning values determined as Null to [an+1]. ***Alter Table*** Br **add** an+1 **;** ***Update*** Br ***set*** Br.an+1=null**;** ▪ The above query acts on the relation body Br and not on the heading Hr of the relation r. Then, any necessary treatment for the logical manipulation of the relation r is not possible.
(3) EXT r BY Op(P1,:t1,p2:t2, …) : tn+1 [begin <Op_body> end]	Not possible with SQL. The only possibility is to create trigger associated to a new column, acting during updating of column. ***1. Alter Table*** Br **add** an+1 **;** **2. Create Trigger** Trig_op **on** Br **for** an+1 */*implements statement Ext r */*
(4) EXT ALL r BY Op(P1,:t1,p2:t2, …) : tn+1 [begin <Op_body> end.]	**1. Alter Table** Br ADD (Op tn+1); **2. Create Function** Op(p1 t1,p2 t2,…, p t) **Return** tn+1 **is** **begin** <Op_body> **end**; **3. Update** Br **set** Br.Op**=** Op(p1 IN t1,…, p IN t)
(5) EXTRACT r FROM (r1, r2, …, rp) BY Op(p1:t1,p2:t2,…)	operator not exist in SQL3. So, it should be interesting to integrate this possibility into SQL3 : Create a relation r, from relations r1,r2,…,rp, by using the operator *Op*. A such statement could be: **create relation** r **by** Op(p1 IN t1,p2 IN t2,…, p IN t) **is** (a1:T1,a2:T2,…,an:Tn);
(6.a) SHOW [ALL] ADT [ADT_name> ON r WHERE <conditions>	This generation operator of the data abstract types is not available on SQL3 because it concerns the relation headings in the object/relational model.
(6.b) SHOW[ALL] INSTANCE ON <ADT_name> WHERE <conditions>	▪ This object generation operator is not available on SQL3 because it concerns relations generated from abstract data types or ADTs.

Table 2a. A* versus query language SQL3.

A* Extension Operators	Object DBMS (ODMG ODL)
(1) EXT r ADD(an : tn+1) BY Op(P1,:t1,p2:t2, …) : tn+1	***interface** Hext :Hr* *(**extent** Bext)* *{ **attribute** tn+1 an+1; **void** init (tn+1 an+1);} ;* ***void** init (tn+1 an+1)* *{ **Select** e.an+1 From **e** in **Bext** **set_value**(Op(p1:t1,p2:t2,..,p:t): an+1) };* /* Keywords in bold correspond to ODL and OQL languages of the ODMG system.. The inheritance exploitation (ie: Hext Hr) allows to carry out this algebraic extension. It is important to note that the computed type extension operator is integrated into the ODBMS according to the programming language chosen for object interface specification. Consequently, the realization of such an operator is limited by the weaknesses of object models concerning the data interrogation language */
(2) EXT r ADD(an : tn+1)	Variant of the operator (1), this operator allows to extend a type defined by a heading H. Indeed, *Hext extension* of the heading Hr is defined by : ***interface** Hext :Hr* *(**extent** Bext) { **attribute** tn+1 an+1; **void** init (tn+1 an+1);} ;* ***void** init (tn+1 an+1)* *{ **Select** e.an+1 From **e** in **Bext** **set_value**(nil:an+1)};*
(3) EXT r BY Op(P1,:t1,p2:t2, …) : tn+1 [begin <Op_body> end]	▪ ***interface** Hext :Hr* (**extent** Bext) { **attribute** tn+1 Op; *tn+1 Op(p1:t1,p2:t2,…,p:t) ;* **[begin** <Op_body> **end.]**} The nature of Op determines the nature of the extension EXT.Op applied to the *Hext* domain allows either to specialize the latter (section 4.5.3 a) or to redefine it by grouping types included in *Hext* (section 4.5.3 b) or unflat the type *Hext* into its elements or basic domains (section 4.5.3 (c)). We note in that case that the scheme manipulations are not possible in the object model OMG. Still, the effective manipulation of objects as domains or types requires a conception of objects on the basis of the set theory. Propositions (1, 2 and 3), quoted above, allow to preserve the relational concepts by integrating the operator notion.
(4) EXT ALL r BY Op(P1,:t1,p2:t2, …) : tn+1 [begin <Op_body> end.]	▪ Unlike the operator *Op* in the extension (3) above, the operator *Op* in that case acts exclusively on the body *Bext* of the relation r such as : ***interface** Hext :Hr (**extent** Bext) {* ***attribute** tn+1 Op; tn+1 Op(p1:t1,p2:t2,…,p:t) ;* ***[begin** <Op_body> **end.]**}
(5) EXTRACT r FROM (r1, r2, …, rp) BY Op(p1:t1,p2:t2,…)	▪ ***interface** Hr :Hr1,Hr2,…,Hrp* (**extent** Br) {**[*begin*** <Op_body> ***end.***]} The operator *Op* extracts the object Hr from many objects *Hr1,Hr2,…Hrp.*
(6.a) SHOW [ALL] ADT [ADT_name> ON r WHERE <conditions>	▪ This operator does not exist in the ODL, because it allows a computation on the database scheme and its evolution (see section 4.4.6 (a)).
(6.b) SHOW[ALL] INSTANCE ON <ADT_name> WHERE <conditions>	▪ ***For all** x **in** B_{<name_ADT>}:x.<Conditions>=**True*** The variable x represents instances corresponding to a given heading H < ADT_name >. Structure **For all.. In ;** is adopted according to the language ODL of the ODM (see section 4.4.6 (b)).

Table 2 b. A* versus object description language (SQL3).

7. Conclusion

The object/relational model extension proposed in this paper, with the operators *Op (p1:t1 , p2:t2, pn tn)* is inspired by Darwen and Date's formalism. The definition of any relation *r* according to the form *<Hr,Br>* where *Hr* is a heading and *Br* is the body of *r*, has allowed operating independently on the scheme of object/relational data base from concrete relations. We have shown the interest of such an extension of algebraic operators, comparing to stored procedures or PSM [14], for application domains in which data representation necessitates complex types. The definition of a fragment of language *ERA** to exploit geometrical and topological data in a road network, has shown its importance. This language can be considered as a new possibility of SQL3, a functionality dealing with the complex algorithms of computation, modeling and querying for new applications. Indeed, on the one hand, the operator *OP* in the definition *Hs={<a1,t1>,<an tn>,<OP1,TOP>}* is useful in the enhancement of object/relational database scheme and on the other hand, the extended algebraic operators permits to further improve the data querying language. Consequently, the undertaking of a prototype ORDBMS and its integration within RDBMSs will allow developers to deal with new queries in database.

8. References

[1] H.Darwen and C.J.Date. *Foundation for Object/Relational Database: The third manifesto*. Addison Wesley, 1998.

[2] C.J Date, Hugh Darwen, *Foundation for Future Database Systems: The Third Manifesto* , Hardcover, Pearson Education 2000.

[3] P.Seshadri. *Enhanced abstract data types in object-relational databases*. The VLDB Journal, 7(3): 130-140, 1998.

[4] M. Stonebraker, P.Brown. *Object-relational DBMSs: Tracking the Next Great Wave*. 2e, Morgan Kaufmann, 1999.

[5] E.F.Codd. *A relational data model for large shared databanks*. Communications of the ACM, 13(6): 377-387, 1970.

[6] R.G.G.Cattel. *The Object data standard ODMG 3.0*, Morgan Kaufmann, 2000.

[7] Nguyen bahao, Nait bahloul safia, A.E. Harmanci and E.Gelenbe(Eds), *Sophistical queries to relational and object oriented databases*; Proc. of ISCIS, pp 687-695; Turkey 1990.

[8] M.P. Atkinson, et al. *The OODBS manifesto*. DOOD, page73-94, 1989.

[9] H.Darwen, C.J.Date. *The third manifesto*. SIGMOD Record, 24(1):39-49, 1995.

[10] W.Kim. Modern database systems. *The Object Model, Interoperability*, and Beyond. Addition-Wesley, pages 238-254, 1995.

[11] G. Lausen and G.Vossen. *Models and language of object-relational databases*. Addition-Wesley, 1997.

[12] W.Kim. *Object-oriented database system: Promises, reality and future*. VLDB Conference, pp 676-687, 1993.

[13] W.Kim. *A model of queries for object-oriented database*. VLDB Conf., pp 423-432, 1989

[14] J. Melton. *Understanding SQL's Stored Procedures*. Morgan Kaufmann Publisher 1998.

[15] L.Libkin and L.Wong. *Query language for bags and aggregate function*. Journal of Computer and System Sciences, 55(2):241-272, 1997.

[16] G. Ozsoyoglu, Z.M.Ozsoyoglu, V.Matos. *Extending relational algebra and relational calculus with set-valued attributes and aggregate functions*.ACM TODS,12(4) , 1987.

[17] B.Jaeschke, H.J.Schek. *Remarks on the algebra of non first normal form*. PODS, pp124-138, 1982.

[18] H.J.Schek, M.H.Sholl. *The relational valued-attributes*. IS, 11(2):137-147, 1986.

[19] L.S.Colby. *A recursive algebra for nested relations*. IS, 15(5):567-582, 1990.

[20] M.A.Roth, J.E.Kirkpatrick. *Algebras for nested relations*. Data Engineering, 11(3):39-47, 1988.

[21] H.F.Korth, M.K.Roth. *Query languages for nested relational databases*. In Nested Relations and Complex Objects in Databases. LNCS 361, pp 190-204, 1989.

[22] M.Levene, G.Loizou. *The nested universal relation data model*. Journal of Computer and System Sciences, 49(3):683-717, 1994.

[23] M.Gyssens, D.Van Gucht. *A comparison between algebraic query languages for flat and nested databases*. Theorical Computer Science, 87(2): 263-286, 1991.

[24] S.Abiteboul, C.Berri. *The power of languages for the manipulation of complex values*. The VLDB Journal, 4(4):727-794, 1995.

[25] R.Hull. *A survey of theoretical research on typed complex database objects*. J.Pareadens, Ed, Academic Press, pp 193-256, 1987.

[26] J.Van Den Bussche, J.Paredaens. *The expressive power of complex values in object-based data models*. Information and Computation, 120(2):220-236, 1995.

[27] F.Bancilhon and S.Khoshafian. *A calculus for complex objects*. PODS, pp 53-59, 1986.

[28] T.Leung, et al. *The AQUA data model and algebra*. DBPL, pages 157-175, 1993.

[29] S.Cluet, G.Moerkotte. *Nested queries in object bases*. DBPL, pp 226-242, 1993.

[30] D. Gross-Amblard. *Approximation dans les bases de données contraintes* Ph.D.Thèse 2000.