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 Menouar Oran 310000 - Algerie
e-mail : nait1@yahoo.fr
(2) INSA de Lyon – LIRIS 2672 CNRS,
7 avenue Jean Capelle.
69621, Villeurbanne – France.
e-mail: youssef.amghar@insa-lyon.fr

Abstract: The object relational data model [1,2,3] takes advantage of Codd's relational calculus power [5] and the object concept characteristics [6,7,11]. In fact, two major approaches have been adopted to satisfy the designers and users of the advanced databases. A revolutionary approach [3,7,13,15] which integrates the object characteristics into the new data models and where the specification of data constraints and the definition of interrogation language are considered as important research problems. The second one [2,22,24,48], called evolutionary approach, consists in preserving Codd's data model when being enriched by adequate concepts for the coverage of current applications in database. In this approach and compared with studies presented by Melton [18] and quoted by Gardarin [4], Date and Darwen have proposed in [1,2] the foundations of the object relational model. So, an algebra $A$ consisted of first order logic operators has been defined to express various classes of queries in object relational database. This paper presents an extension of object relational model [1,2] to new types generated by operators. These operators, called $Op$, offer a means to specify domains as functions and allow consequently the increasing of data model expressiveness. Concerning data query language, or more precisely the logical data calculation, the algebra $A$ [1] is adapted and extended, giving $A^*$, to support this extension of object relational model. Algebra $A^*$ contains algebraic operators which are able to support this new extension. Furthermore, to enhance the object relational model, the query language $ERA^*$ that implements $A^*$, consists essentially in both logical calculation operators and new algebraic operators.

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

1. Introduction: Object relational model [1, 2, 3, 4] is the result of several research studies in database field having as main orientations the preservation of facilities and performances offered by the relational model [5] and the enforcement of data modeling capabilities with the integration of object characteristics [3,6,7]. Relational model defined initially by Codd has always been a reference model to the design of databases during the seventies. The data of a relational database has been described and organized around very simple concepts of n-uplet and relation. It is widely recognized that this simplicity has favored the design and definition of formal query languages, which access easily (calculation) and effectively (algebra) to the data. However, this advantage does not correspond exactly to the real world semantic entities, this simplicity lead to an insufficiency of modelling at conceptual level.

About the eighties, the use of object approach for the first time in programming languages [47] and the advance of technology networks have opened many database research axes. In fact, database management systems [8] had to deal with complex, heterogeneous, temporal and available data on many sites. Two major tendencies are presented to meet the database new requirements: the object oriented database approach [9,10,11,12,13] and the object /relational database approach [1, 2, 3, 4]. In Object Database Management System, databases are described by rich and varied concepts that increase the representation of the real world entities in database. Nevertheless, it seems that the surrounded cost to support this richness grows during the designing and definition of object languages [14,15,16,17]. Indeed, the query language must be conceived to consider a multitude of concepts such as, object
identity, complex type, inheritance, method call, overload and late binding. The object database approach [3,6] tends to enforce the object oriented programming languages by the database characteristics such as persistence and interrogation of data via object query languages as OQL (Object Query Language) [3,6,19].

However, the relational object database approach has to preserve inherent advantages to 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 the 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 allows revealing more easily some equivalence between the algebraic language expressions. So the language expression power is an important question. Relational algebra expressiveness limits are well known. The most quoted example is the operator of the transitive closure of a relation that cannot be expressed with relational algebra operators. Relational algebra is also known by its incapacity to express some simple and useful queries, which are easily expressed in the language SQL. For example, queries like those asking for the calculation of annual salary from the monthly salary set of employees. Indeed, it is widely admitted [20,21,22,23] that any query requiring an arithmetical calculation or functions of SQL aggregates is not possible to express with the relational algebra. Motivated by the relational algebra expression coverage, researches have been undertaken in order to increase the existing algebras or to define the other families of algebras such as algebras for nested relation models [25,26,27,28,29,30,31,32], algebras for structured value models [33,34,35,36,37] and algebras for object models [38,39,40,41,42,43,14,44]. One of the interests of these researches is the formalization of the object relational model and the proposition of a relational object algebra [1,2] called $A$. The algebra $A$, for the formal relational object model has allowed the identification of adequate operation types to relational object databases. However, the query language design of these data models consists in defining algebraic operators based on the algebra $A$ to which the extension operators must be integrated for the description and interrogation of complex data, which are rich in symbolic representation, require a prohibitive combinatorial calculation in their manipulations. The constructors, used in query languages based on the algebra $A$, such as $D$ language [1], adopt the set theory for the data specification and the order 1 logic calculation to manipulate such data (e.g. not, now, and, compose, other).

In this paper, we propose an Algebra $A^*$ as an extension of algebra $A$ to enhance object/relational model with the notion of domain generated by function or operator. Still, the query language based on algebra $A^*$ should be considered with regard to the power of expression and calculation which is offered by a standard query language like SQL3 [46]. The algebra $A^*$, for extended object relational models to the notion of domain generated by function or operator $Op$, is composed of logical operators (i.e. not *, now *, 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 in integrating within a relational object scheme, types defined by operator. These types built likewise, are not generated by constructors but defined by function or operator $Op$ as being couples <$Op, TOp>$ and where $TOp$ is the type returned by the function or operator $Op$. Algebra $A^*$, extension of algebra $A$, allows to manipulate complex entities requiring symbolic representations in their definitions as in the geometrical and spatial database. These latter, processing the shape and position of objects in space and time constitutes 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 [8], extension operators have been developed to undertake sophisticated queries dealing with geometrical figures in relational or extended relational context. In [45], geometrical queries are resolved at polynomial time in a constraint database context.

In this paper, geometrical queries are tackled in an extended relational object database context to the concept of domain generated by operator or function. Thus, algebra $A^*$ containing extension operators is realized by the language of query \( ERA^* \) for the querying of object / relational databases. In section 1, a typical example on a road network connecting geographic zones of polyedrical form is presented. This example will illustrate our concepts. Section 2, concerns Date and Darwen’s approach related to the formalization of the object relational model as well as the algebra $A$ for the manipulation of data in such models. In sections 3 and 4, the extension of the object relational model and the specification of the algebra $A^*$ are detailed. Section 5 compares the algebra $A^*$ with the relational algebra on the one hand and with specification and querying languages of object database schemas (ie, ODL, OQL) on the other hand. A fragment of query language \( ERA^* \) for the support of types generated by operators $Op$, is presented in section 6. Section 7 discusses the algebra $A^*$ for the querying of extended relational object databases. This paper ends with section 8, which concludes our work.

**Illustrative example:** Let us consider a road network in a geographical space in which we will be interested by 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 as 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 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 imply 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.

![Figure 1: Road network](image)

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 polygones generally
(Figure 1.a). So the road network is represented and realized, in language D of Date and Darwen [1, 2], by the object relational database named $ROAD\_BASE$ and defined as follows:

$$\begin{align*} ROAD\_BASE & \quad Points = RELATION \{ \text{ld\_Point char(10),} \\
& \quad \text{Abscissa real,} \\
& \quad \text{Ordinate real} \} \\
& \quad KEY \{ \text{ld\_Point} \} ; \\
\end{align*}$$

$$\begin{align*} Road\_Segment = RELATION \{ \text{ldseg integer,} \\
& \quad \text{firstseg char(10),} \\
& \quad \text{finalseg char(10)} \} \\
& \quad KEY \{ \text{ldseg} \} ; \\
\end{align*}$$

$$\begin{align*} Roads = RELATION \{ \text{idch integer,} \\
& \quad \text{Road SET(integer)} \} \\
& \quad KEY \{ \text{idch} \} ; \\
\end{align*}$$

$$\begin{align*} Polygones = RELATION \{ \text{idpoly integer,} \\
& \quad \text{Polygon SET(integer)} \} \\
& \quad KEY \{ \text{idpoly} \} ; \\
\end{align*}$$

where $SET$ is a predefined type within DBMS

**Remarks:**

- 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 in the relation $Points$.
- $ldch$ identifies the road, the set $\{ld1, ld2, ldn\}$ defines the respective identifiers of segments forming this road
- $ldpoly$ identifies the polygon, the set $\{ld1,ld2,...,ldm\}$ defines the respective identifiers of segments constituting a given polygon.
- Relation $Road\_Segment$ represents the segments that bind the cities defined in the relation $Points$. Attributes $firstseg$ and $finalseg$ are the respective identifiers of departure and arrival points in a segment identified by $ldseg$.

$$\begin{align*} H\_segroad & = \{<ldseg, integer>, <firstseg, char(10)>, <finalseg, char(10)>\} \ ; \\
B\_segroad & = \{t \mid t = \{<ldseg, integer, n>, <firstseg, char, V1>, <finalseg, char, V2>\} \\
& \quad n \in \mathbb{N} \text{ and } V1 \text{ and } V2 \in \{A,...,L\}; \text{City name set} \} \ ; \\
/* H\_segroad, B\_segroad are the heading and the body of the relation Segment\_road */ \\
\end{align*}$$

- The relation $Road$ specifies all the possible roads between the different cities in the relation $Points$; the road being a set of road segments.

$$\begin{align*} H\_Roads & = \{<idch, integer>, <Road, SET(integer)>\}; \\
B\_Roads & = \{t \mid t = \{<idch, integer, C>, <Road, SET(integer), \{ld1,ld2,...,ldn\}>\} \} ; \\
/* H\_Roads, B\_Roads are the heading and the body of the relation Roads */ \\
\end{align*}$$

- The relation $Polygones$ defines the set of the convex figures in the road network.

$$\begin{align*} H\_Polygones & = \{<idpoly, integer, <Polygon, SET(integer)>\} ; \\
B\_Polygones & = \{t \mid t = \{<idpoly, integer, P>, <Polygon, SET(integer), \{ld1,ld2,...,ldm\}>\} \} ; \\
\end{align*}$$

Let us consider now a state of the database $ROAD\_BASE$ containing 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 \( H_r \) specifies the scheme of the relation whereas the body \( B_r \) contains tuples corresponding to \( H_r \). So, the relations Points, Segmen_Road, Roads and Polygones are defined as follows:

\[
\begin{align*}
H_{\text{Points}} &= \{<\text{Id\_Point}, \text{char}(10)>, <\text{Abscissa}, \text{real}>, <\text{Ordinate}, \text{real}>\}; \\
B_{\text{Points}} &= \{\{<\text{Id\_Point}, \text{char}(10), A>, <\text{Abscissa}, \text{real}, 6>, <\text{Ordinate}, \text{real}, 12>\}, \\
&\quad \{<\text{Id\_Point}, \text{char}(10), B>, <\text{Abscissa}, \text{real}, 12>, <\text{Ordinate}, \text{real}, 10>\}, \\
&\quad \{<\text{Id\_Point}, \text{char}(10), C>, <\text{Abscissa}, \text{real}, 3>, <\text{Ordinate}, \text{real}, 6>\}, \\
&\quad \{<\text{Id\_Point}, \text{char}(10), D>, <\text{Abscissa}, \text{real}, 20>, <\text{Ordinate}, \text{real}, 15>\}, \\
&\quad \{<\text{Id\_Point}, \text{char}(10), E>, <\text{Abscissa}, \text{real}, 18>, <\text{Ordinate}, \text{real}, 3>\};
\end{align*}
\]

\[
\begin{align*}
H_{\text{Segment\_Road}} &= \{<\text{Id\_seg}, \text{integer}>, <\text{first\_seg}, \text{char}(10)>, <\text{finalseg}, \text{char}(10)>\}; \\
B_{\text{Segment\_Road}} &= \{\{<\text{Id\_seg}, \text{integer}, 1>, <\text{first\_seg}, \text{char}(10), A>, <\text{finalseg}, \text{char}(10), B>\}, \\
&\quad \{<\text{Id\_seg}, \text{integer}, 2>, <\text{first\_seg}, \text{char}(10), A>, <\text{finalseg}, \text{char}(10), D>\}, \\
&\quad \{<\text{Id\_seg}, \text{integer}, 3>, <\text{first\_seg}, \text{char}(10), E>, <\text{finalseg}, \text{char}(10), A>\}, \\
&\quad \{<\text{Id\_seg}, \text{integer}, 4>, <\text{first\_seg}, \text{char}(10), B>, <\text{finalseg}, \text{char}(10), D>\}, \\
&\quad \{<\text{Id\_seg}, \text{integer}, 5>, <\text{first\_seg}, \text{char}(10), C>, <\text{finalseg}, \text{char}(10), E>\}, \\
&\quad \{<\text{Id\_seg}, \text{integer}, 6>, <\text{first\_seg}, \text{char}(10), E>, <\text{finalseg}, \text{char}(10), D>\}, \\
&\quad \{<\text{Id\_seg}, \text{integer}, 7>, <\text{first\_seg}, \text{char}(10), D>, <\text{finalseg}, \text{char}(10), A>\}, \\
&\quad \ldots\};
\end{align*}
\]

/* The segment AD is different from the segment DA 
Because the departure and arrival segments are different */

\[
\begin{align*}
H_{\text{Roads}} &= \{<\text{Id\_ch}, \text{integer}>, <\text{Road}, \text{SET}(\text{integer})>\}; \\
B_{\text{Roads}} &= \{\{<\text{Id\_ch}, \text{integer}, 1>, <\text{Road}, \text{SET}(\text{integer})>, \{1, 4\}>\}, \\
&\quad \{<\text{Id\_ch}, \text{integer}, 20>, <\text{Road}, \text{SET}(\text{integer})>, \{3, 2\}>\}, \\
&\quad \{<\text{Id\_ch}, \text{integer}, 30>, <\text{Road}, \text{SET}(\text{integer})>, \{5, 3\}>\}, \\
&\quad \{<\text{Id\_ch}, \text{integer}, 40>, <\text{Road}, \text{SET}(\text{integer})>, \{5, 3, 1, 4\}>\};
\end{align*}
\]

\[
\begin{align*}
H_{\text{Polygones}} &= \{<\text{Id\_poly}, \text{integer}>, <\text{Polygon}, \text{SET}(\text{integer})>\}; \\
B_{\text{Polygones}} &= \{\{<\text{Id\_poly}, \text{integer}, 100>, <\text{Polygon}, \text{SET}(\text{integer})>, \{1, 4\}>\}, \\
&\quad \{<\text{Id\_poly}, \text{integer}, 200>, <\text{Polygon}, \text{SET}(\text{integer})>, \{5, 6, 7\}>\}, \\
&\quad \ldots\};
\end{align*}
\]

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 road network. 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 not passing via points in 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 pass via 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 that passes via all the cities?
(11) Etc...

The resolution of such queries requires on the one hand the choice of a data model representing different 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 passing via 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. However, and before starting the resolution with \( A^* \), it is important to begin with the concept definitions. Indeed, in section 2 we describe Date and Darwen’s formalism as well as algebra \( A \) proposed in [1, 2].

2. Date and Darwen’s Approach: Date and Darwen [1, 2] consider that SQL3 [46] does not realize 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 the relation to class. So, unlike the data model developed by the group ODMG in [19] 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 \( \text{Class} = \text{Domain} \) 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 in the algebra \( A \). Indeed, a relation \( r \) is defined with its heading \( H_r \) and body \( B_r \). The heading represents the schema of the relation \( r \) and is defined as a set of couple \( <a,T> \) (with \( a \) an attribute and \( T \) the attribute type of \( a \)), while the body \( B_r \) is a set of tuple. A tuple \( t \) being defined as a set of triplet \( <a,Ti,v> \) with \( <a,Ti> \) is the \( H_r \) element and \( v \) the value of the attribute \( a \).

2.1. Formal definitions: Let 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 \( H_r \) 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 \( a1 \neq a2 \) (i.e.: The names of attributes are different).

Example: The heading of the relation Points quoted above is:
\[ H_{\text{Points}} = \{<\text{Id_Point}, \text{char}(10)>, <\text{Abscissa}, \text{real}>, <\text{Ordinate}, \text{real}>\}; \]
where \( \text{Id_Point}, \text{Abscissa}, \text{Ordinate} : \) different attributes; \( \text{char}(10), \text{real} : \) types;

(b) Tuple: Let \( tr \) be a tuple and \( H_r \) a heading; The tuple \( tr \) is a set of ordered triplets \( <a,T,v> \) where each attribute \( ai \) of \( H_r \) is associated with a triplet \( <ai,Ti,v> \)
Example:
\[t1=\{\text{Id	extunderscore Point}, \text{char(10)}, \text{C}, \text{Abscissa, real, 3}, \text{Ordinate, real, 17}\}\;\]
\[t1\] is a tuple of the relation Points realizing the city C that 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 \(tj \in Br\).

**Example:** Let \(tj=\{\text{Id	extunderscore Point}, \text{char (10)}, X, \text{Abscissa, real, 40}, \text{Ordinate, real, 28}\}\).

We notice that the tuple \(tj\) corresponds to the heading \(H\text{\_Points}\) in the relation \(Points\) defined in the example \(ROAD\_BASE\) above; on the other hand \(tj\) does not belong to the body \(B\text{\_Points}\) of the relation.

**Remarks:**

(1) Every heading \(Hr\) and body \(Br\) is viewed as a set.

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

The object relational algebra \(A\), 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, every algebraic operator in \(A\), applied to a relation \(r\), considers semantic actions on \(Hr\) different from those applied to \(Br\).

2.2 **Operators of the algebra** \(A\): Algebra \(A\) 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 consider 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 \(r = (Hr, Br)\) and \(s = (Hs, Bs)\) with \(Hr, Hs\) and, \(Br, Bs\) respectively headings and bodies of relations \(r\) and \(s\).

2.2.1 **Operator AND:** The operator AND, 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 \(s\), containing every corresponding tuple of heading \(Hs\), 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 \(r2\).

\[
s \leftarrow r1 \text{AND} r2
\]
\[Hs=Hr1 \cup Hr2
\]
\[Bs=\{ts/\exists tr1, \exists tr2((tr1 \in Br1) \land (tr2 \in Br2)) \land (ts = tr1 \cup tr2)\}\]

2.2.2 **Operator OR:** The operator OR, is a relational disjunction, generalization of the operation \(\text{Union}\) in the Codd's relational algebra. So, \(Hs\), the heading of the relation result \(s\), is the union of the headings \(Hr1\) and \(Hr2\) of the input relations \(r1\) and \(r2\). The body \(Bs\) of \(s\) is a reflecting set of the disjunction of some tuples in the respective bodies \(Br1\) and \(Br2\) of the relations \(r1\) and \(r2\).

\[
s \leftarrow r1 \text{ OR } r2
\]
\[Hs=Hr1 \cup Hr2
\]
\[Bs=\{ts/\exists tr1, \exists tr2(((tr1 \in Br1) \lor (tr2 \in Br2)) \land (ts = tr1 \cup tr2))\};
\]
The operator \( OR \), in Date and Darwen’s algebra \( A \), allows to treat the headings and bodies separately and has no equivalent operator in the Codd’s relational algebra.

### 2.2.3 Operator NOT:
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 \( r \).

\[
\begin{align*}
  s & \leftarrow NOT(r); \\
  Hs & = Hr; \\
  Bs & = \{ ts/\forall tr((tr \notin Br) \land (ts = tr)) \} \text{ \( tr \) being a tuple belonging to \( Br \)}
\end{align*}
\]

### 2.2.4 Operator RENAME:
The operator \( RENAME \) allows to rename an 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 the same as that heading \( Hr \) of \( r \) with the exception of the pair \( <a,T> \) that will be 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 \( <a,T,v> \).

\[
\begin{align*}
  s & \leftarrow r \ RENAME \ (a,b); \\
  Hs & = \{ Hr-\{<a,T>\}\} \cup \{<b,T>\}; \\
  Bs & = \{ ts/\exists tr, \exists v((tr \in Br) \land (v \in T) \land (\langle a,T,v \rangle \in tr) \land (ts=\langle a,T,v \rangle)) \} \cup \{<b,T,v>\}) \}
\end{align*}
\]

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.

### 2.2.5 Operator REMOVE:
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 \( Hs \) of \( s \) 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 algebra \( A \), 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.

\[
\begin{align*}
  s & \leftarrow r \ REMOVE \ a; \text{ where } <a,T> \in Hr \\
  Hs & = Hr-\{<a,T>\} \\
  Bs & = \{ ts/\exists tr, \exists v((tr \in Br) \land (v \in T) \land (\langle a,T,v \rangle \in tr) \land (ts=\langle a,T,v \rangle)) \} \cup \{<b,T,v>\}) \}
\end{align*}
\]

### 2.2.6 Operator COMPOSE:
The \( COMPOSE \) operator, is defined by the combination of the operators \( AND \) and \( REMOVE \) such as:

\[
\begin{align*}
  s & \leftarrow r1 \ COMPOSE \ r2; \\
  s & \equiv (r1 \ AND \ r2) \ REMOVE \ an...REMOVE \ a2 \ REMOVE \ a1; \\
  Hs & = \{ (Hr1 \cup Hr2) - \{<a1,t1>,<a2,t2>,...,<an,tn>\} \} \\
  Bs & = \{ ts/\exists tr1, \exists tr2, \forall v \in t1((t1 \in Br1) \lor (t2 \in Br2)) \land (ts=(tr1 \cup tr2) - \{<ai,t,v>/ai \in \{a1,...,an\} \}) \}
\end{align*}
\]

**Remarks:**
- \( a1, a2, ... an \) are the common attributes to the relations \( r1 \) and \( r2 \) (\( n \geq 0 \)).
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 $(r_1 \text{COMPOSE} r_2) \equiv (r_1 \text{AND} r_2)$ and the relation $(r_1 \text{X} r_2)$ is included in the $(r_1 \text{AND} r_2)$ body.

where $(r_1 \text{X} r_2)$: Cartesian product of $r_1$ by $r_2$ 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 $r_1$ or $r_2$. On the other hand, the heading $Hr$ is defined by $Hr = Hr_1 \cup Hr_2$.

### 2.2.7 Operator **TCLOSE**

Compared to the Codd's relational algebra, algebra $A$ 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:

$$r^+ \leftarrow \text{TCLOSE } r;$$

$Hr^+ = Hr$;

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

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

$$\Rightarrow ((t=\{<X,T,x>,<Y,T,y>\}) \land (t \in Br^+))$$

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 $z1, z2, \ldots, zn$ having the same type $T$
   Such as the tuples: $\{<X,T,x>,<Y,T,z1>\};\{<X,T,z1>,<Y,T,z2>\};\ldots;\{<X,T,zn>,<Y,T,y>\} \in Br$
3. There is a way between tuples $<X,T,x>$ and $<Y,T,y>$ in the body $Br$

**Remarks:**

- For the operations $(r_1 \text{ OR } r_2)$ and $(r_1 \text{ AND } r_2)$,
  $((<a,T1> \in Hr_1) \land (<a,T2> \in Hr_2)) \Rightarrow (T1=T2)$;

- For the operation $(r \text{ RENAME } (A,B))$,
  $((<a,T> \not\in Hr) \lor (<b,T> \in Hr)) \Rightarrow (r=r \text{ RENAME } (a,b))$.

- In the case of **TCLOSE**, $(Br^+ = Br \cup E) \Rightarrow (Br \subset Br^+)$
  with $E$: set of the added tuples.

### 3. Extended Object Relational Model

In this paper, the idea considers that a domain or type is specified by an operator $Op$ which would be equivalent, in our case, to a function, specifies a domain or type. The set of the triplets $\{<Op,TOp,Op(p1,t1,p2,t2,...,pn:tn)>\}$ represents i) $Op$ a function, ii) $TOp$: type returned by $Op$, iii) $Op(p1,...)$ : the function signature:

$Hr=\{<a1,t1>,<a2,t2>,...,<an,tn>,<Op1,TOp1>,...,<Opn,TOpn>\}$

Where $a1,a2,\ldots,an$ are attributes associated with the relation $r$
$t1,t2,\ldots,tn$ types allocated respectively to attributes $ai$.

$Op1,Op2,\ldots,Opm$ are the operator domain definitions.

$TOp1,TOp2,\ldots,TOpm$ are the respective types returned
3.1 Propositions: Compared to concepts quoted above (see Section 2), the concepts of the heading, of tuple and of the body are redefined according to our approach.

3.1.1 Proposition 1: A heading $H_r$ is a set of ordered pair $<X,T_x>$ where $X$ is an attribute $a$ or an operator $Op$. The type $T_x$ is either a predefined type $T$ or a type $TOp$ of the operator $Op$ result.

3.1.2 Proposition 2: Let $H_r$ be a heading and $t$ a tuple; The tuple $t$ is a set of ordered triplet $<X,T_x,v>$ corresponds to each attribute or operator $X$ in $H_r$. In this case where $X$ is an operator $Op$, the value $v$ is defined by $v=Op([p1,...,[pn]])$. The nature of parameters $p_i$ ($i=1..n$) allows expressing different semantic entities in the modeled reality.

3.1.3 Proposition 3: A body $Br$ of a relation $r$ is a set of tuples $t$ containing tuples $t_j$ such as $t_j=<Op,TOp,Op(p1,...,pn)>$ where $p1,...,pn$ are parameters assigned by value or by variable.

**Example:**

a. $<\text{Define\_Road}, \text{set}\text{\,(integer)}, \text{Define\_Road}\,(\text{Id-departure:}\text{\,(char\,(10)}, \text{Id-arrival:}\text{\,(char\,(10))}> /* Signature of $Op$

b. $<\text{Define\_Road, set\,(integer)}, \text{Define\_Road}(\text{A,C})> /* A, C are cities, see Figure 2.

c. $<\text{inverse,real,inverse}(\text{x:}\text{real})> \equiv <\text{inverse,real,}(1/\text{x})> /* Parameters by variable */

d. $<\text{inverse,real,}\text{inverse}(4)> \equiv <\text{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 parameters of the input.

3.1.4 Proposition 4: An operator $Op(p1,...,pn)$ is a scalar function or domain constructor.

3.1.5 Proposition 5: An abstract data type, $ADT$, is a heading of a relation $r$ named $H_r$ which contains at least a couple $<Op, TOp>$ and where the operator $Op$ is a generator of the domain $TOp$. The tuples $t_j$ corresponding to heading $H_r$ are called instances.

3.2 Corollaries: Following the propositions defined above we deduce the notions of instances for an abstract data type and equivalence between two tuples $T1$ and $T2$.

3.2.1 Corollary 1: an instance of an abstract data type defined by the heading $H_{adt}={<a1,T1>,<Op,TOp>,...,<an,Tn>}$ is a tuple $t$ defined by:

$t={<a1,T1,v1>,<Op,TOp,Op(p1,t1,p2,t2,...,pn,t_n)>,...,<an,Tn,vn}>$

**Example** : $T={<\text{id\_ch,}\text{integer,10>,<Define\_Road, set\text{\,(integer), Define\_Road}\,(\text{A,D})}>}$

The road identified by $N=10$ between the city A and the city D is $\{A,B,D\}$ ; /* See Figure 2. */

3.2.2 Corollary 2: Let $H$ be a heading defined by $H={<Xi,Ti>}$ where $Xi \in \{\text{Attribute,Op}\}$.

Two tuples $T1$ and $T2$ corresponding to the heading $H$ are equivalent if and only if:

$$\forall <a,t,v> \in T_1, \exists <b,t,u> \in T_2: ((a=b) \land (v=u))$$

where
∀<Op1,TOp1,Op1(p1:t1,p2:t2,...,p:t)>∈T1,
∃<Op2,TOp2,Op2(q1:t1,q2:t2,...,qm:tm)>∈T2,
⇒((Op1=Op2)∧(TOp1=TOp2)∧(Op1(p1:t1,p2:t2,...,p:t)=Op2(q1:t1,q2:t2,...,qm:tm)))

3.3 Example of an extended data model: Let us consider the illustrative example of the section 1, 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_Segment = RELATION( Idseg integer, firstseg char(10), finalseg char(10) )KEY { Idseg } ;
Roads = RELATION( Idch integer,
{ (' OP Road((Id-departure char(10),Id-arrival char(10)) : SET(integer)
KEY { Idch } ;
Polygones1 = RELATION( Idpoly integer,
{ (' OP Polygon(Idseg-departure integer,Idseg-arrival integer) : SET(integer)
KEY { Idpoly } ;
Remarks:
- The operator Road allows to find 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 (integer) ← 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 Idseg-departure and Idseg-arrival. This operator returns a set of segments that form the convex space or the polygon in question.

SET (integer) ← Polygones (Idseg-departure: integer, Idseg-arrival: integer)

- 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 Polygones defined by the operator Polygon, Type TOp is equivalent to the type SET (integer) and the values v are such as v = Polygon (Idseg-departure: integer, Idseg-arrival: integer). For example, the polygon CEDA in the figure 2 is defined by the tuple t as follows:

\[ t=\{<Idpoly, integer, 200>, <Polygon, SET(integer), \{5,6,7\}>\} /* see Figure 2. */ \]

4. Operators of the algebra A*: The extended object relational model requires a 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 algebra A [1,2] and the work of [8] on the operator bound by the procedures ”EXT” to specify and define algebra A*. This latter consists of the same operators as algebra A except the operator RENAME, which requires a boring calculation in the case of an extended object relational model. The change of the name of an operator Op implies the calculation of all the tuples of which the operator Op references. In

\[ ^1 \text{The keyword OP in the definition of ROAD_BASE* means that the scheme of relations Roads and Polygons contains domains defined by operators.} \]
spite of the resemblance between the operators of the algebra $A^*$ and those of the algebra $A$, there are nevertheless some differences as we highlight below.

4.1 Operator NOT*: The operator NOT* allows to calculate the complement to the relation $r$. However, in an extended object relational model two tuples corresponding to the same heading can be different.

$$s \leftarrow \text{NOT}^*(r);$$
$$Hs=Hr;$$
$$Bs=\{ts/\exists tr ((tr \notin Br) \land (ts \equiv tr))\}/\ast ts \equiv tr \text{ means that ts is equivalent to tr (see corollary 2)}$$

4.2 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 algebra $A$.

$$s \leftarrow r \ \text{REMOVE}^* X; \text{ where } (<X,T> \in Hr) \land (X \in \{A,Op\})$$
$$Hs=Hr-\{<a,T>\};$$
If $X=a$ then
    begin
        $Hs=Hr-\{<X,T>\};$
        $Bs=\{ts/\exists tr,\exists v ((v \in Tr) \land (<X,T,v> \in tr) \land (ts=tr-\{<X,T,v>\}))\};$
    endif
Else($X=Op$)
    Begin
        $Hs=Hr;$
        $Bs=\{ts/\exists tr ((tr \in Br) \land (<X,T,X(p1,t1,p2,t2,\ldots,p:t)\.\ldots.t)) \in tr) \\
        \land (ts=tr-\{<X,T,X(p1,t1,p2,t2,\ldots,p:t)\.\ldots.t})\}\};$
    endelse

4.3 Operator TCLOSE*: This operator allows 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.

$$r+ \leftarrow \text{TCLOSE}^* r;$$
$$Hr+=Hr;$$
$$Br+=Br \cup \{\text{tuple} \ t/\forall \ \text{séquence} \ S, ((S \in Br) \Rightarrow (t=\{<X,T,x>,<Y,T,y>\}) \land (t \in Br)))\}$$
With
$$S=\{<X,T,x>,<Y,T,z_1>,<Y,T,z_2>,\ldots;<X,T,z_n>,<Y,T,y>\}$$
Or
$$S=\{<Op1,T,v>,<Y,T,v_1>,<Y,T,v_2>,\ldots;<Op2,T,v_1>,<Y,T,z_2>,\ldots;<Op2,T,z_n>,<Y,T,v>\}$$

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

Remarks:
- $(\text{AND}*\equiv \text{AND}), (\text{OR}*\equiv \text{OR})$ et $(\text{COMPOSE}*\equiv \text{COMPOSE})$.
- Every operator of the algebra $A^*$ is followed by the character star (*) to mean that the operator belongs to the extended algebra.
4.4 Operators of extension: The definition of an object relational database contains relations having headings of the type \( H = \{ <a_1,t_1>, <a_2,t_2>, ..., <a_n,t_n>, <O \to p_1, T O \to p_1>, ..., <O \to p_m, T O \to p_m> \} \). 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 proposed in [8] have been adapted and integrated into the specifications of the algebra \( A^* \).

4.4.1 Calculated type extension: The calculated type extension operator allows on the one hand to modify the heading of the relation \( Hr \) by inserting the new attribute \( a_{n+1} \) to \( Hr \), and on the other hand to define the tuples results of \( Op \) application in \( r \); the type of the attribute \( a_{n+1} \) being defined by \( T O \to p_\to p \) of the expression of the operator \( O p(p_1,t_1, p_2, t_2, ..., p_t) \) for a given parameters. The value \( v \) in the triplet \( <a_{n+1}, TO \to p, v> \) is expressed by

\[
v = O p([p_1], ..., [p_n]) \text{ where } [p_i] \text{ is the value of } p_i.
\]

4.4.2 New type extension: The operator of new type extension allows modifying the heading of the relation \( Hr \) by inserting a new attribute \( a_{n+1} \) and its type \( t_{n+1} \) in the heading \( Hr \) of \( r \). The body \( B s \) of \( s \) is such that \( B s \) contains for every tuple \( tr \) of \( r \), the triplet \( <a_{n+1}, t_{n+1}, \text{null}> \) where the constant ' null ' belongs to any system or user type. The operator in this case is equivalent to the operator \( \text{ALTER} \) in SQL3.

We note that the sets \( \{p_1, p_2, ..., p\} \) and \( \{a_1, a_2, an\} \) are not necessarily separated.

**Extension in the calculated type**

**Example:** \( \text{Points} \leftarrow \text{EXT} \ \text{Points} \ ADD \ \text{Redius} \)

\[
bys cacul \_redius(Abscissa: \text{real}, \ Ordinate: \text{real}): \text{real};
\]

Relation \( \text{Points} \) becomes

\[
\text{Points} = \text{RELATION} \{ \ \text{id} \_\text{Point} \ \text{char(10)}, \ Abscissa \ \text{real}, \ Ordinate \ \text{real}, \ \text{Redius} \ \text{real}, \ \text{KEY} \ \{ \ \text{id} \_\text{Point} \} \};
\]

With the operator \( \text{cacul} \_\text{redius} \) defined by :

\[
cacul \_\text{redius}(\text{Abscissa :real}, \ \text{Ordinate:real}) : \text{real} \begin{array}{l}
\text{begin} \\
\text{Redius} = \sqrt{\text{Sqr}(\text{Abscissa}) + \text{Sqr}(\text{Ordinate})}; \\
\text{Return} (\text{Redius}); \\
\text{end} \\
\end{array}
\]

**New type extension**

**Example:** \( \text{Points} \leftarrow \text{EXT} \ \text{Points} \ ADD \ \text{Color} : \text{integer} \)

Relation \( \text{Points} \) becomes

\[
\text{Points} = \text{RELATION} \{ \ \text{id} \_\text{Point} \ \text{char(10)}, \ Abscissa \ \text{real}, \ Ordinate \ \text{real}, \ \text{Color} \ \text{integer}, \ \text{KEY} \ \{ \ \text{id} \_\text{Point} \} \};
\]

With \( \text{init} \) \( \text{Points} \) is defined by :

\[
\text{init}(\text{Points}) \text{ begin for } i = 1 \text{ à card}(\text{Points}) \text{ do } \text{Points}.\text{Color} = \text{null}; \text{ end}
\]
4.4.3 Global operator extension: A global operator extension acts on the heading \( H_r \) or the body \( B_r \) of the relation \( r \) and allows defining new domains, new constraints of integrity defined by the operator \( \text{Op} \) or reorganizations of the relation \( r \) and generates as a result, a relation defined by \( H_s=\{<a_1,t>,<a_2,t>,...,<a_n,t>,<\text{Op},T\text{Op}>,...,<\text{Op}>,T\text{Op}>\} \). The operator \( \text{Ext} \) can be internal (1) or external (2) as the operator \( \text{Op} \) is specified when called or predefined in the data base management system.

\[
s \leftarrow \text{EXT } r \text{ BY } \text{Op}(p_1,t_1,p_2,t_2,...,p_t)[t_{t+1}]
\]

\[
\begin{align*}
&\text{BEGIN} \\
&\quad <\text{Op\_body}> \quad /* \text{Body of the operator Op which can be written} \\
&\quad \text{In a high-level language or in SQL */} \\
&\text{END.}
\end{align*}
\]

\[
\begin{align*}
&H_s=\{<b_1,t>,<b_2,t>,...,<b_m,t>\} \text{ avec } b_i \in \{a_i, \text{Op}\} \\
&B_s=\{ts/\exists T \in \{t_1,t_2,...,t_r\}, \exists X \in \{b_1,b_2,...,b_m\} ((<X,T> \in Hs) \land (ts=\{<X,T,v>\}) \land (v=\text{Op}(p_1,t_1,...,p_t)))\}
\end{align*}
\]

Case (a): in this type of extension, the operator \( \text{Op} \) allows to define a domain by the operator \( \text{Op} \). These latter is integrated into the scheme of a given relation (i.e.: \( H_r=H_r \cup \{<\text{Op},T\text{Op}>\} \), \( T\text{Op} \) is the \( \text{Op} \) result type ). Besides, the application of several operators \( \text{Op} \) to a scheme of a relation \( r \) allows to consider various complex types from the relation \( r \) without changing the dimension of the relation \( r \).

Case (a): Global operator extension (Domain of points in Square C)

Example: Points \( ^2 \) \( \leftarrow \text{EXT } \text{Points} \text{ BY } \text{InSquare}(\text{Id\_Point}:\text{integer},C:\text{inequations}):\text{Boolean} ; \)
Relation Points is :
Points = RELATION{  
  \text{Id\_Point } \text{char}(10) , \text{Abscissa } \text{real} , \text{Ordinate } \text{real} , \text{Redius } \text{real} , \text{Color } \text{integer} ,  
  \text{OP(5) InSquare(\text{Id\_Point}:\text{integer},C:\text{inequations}):Boolean}} \text{ KEY \{ \text{Id\_Point} \} ;}

with \text{InSquare}(\text{Id\_Point},C) \text{ defined by} : 
\text{InSquare(\text{Id\_Point}:\text{integer},C:\text{inequations})}  
\text{Begin} 
  \text{If } \text{Id\_Point} \in (\text{square C}) 
  \text{then return 1 Sinon return 0 ;}
\text{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 \( a_1, a_2, an \) 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 \( r \).

\[
H_r=\{<a_1,t>,<a_2,t>,...,<a_n,t>\} ;
H_s=\{<b_1,t>,<b_2,t>,...,<b_m,t>,<\text{Op},T\text{Op}>\} ;
\]

with \( m<n, bj=a_1...ap \text{ and } j \leq m, \text{ and } p \leq n \)

\( bj \) being an aggregation of several attributes \( a_i \)

Case (b): Global operator extension (Restruction of the relation Points)

Example: Points \( \leftarrow \text{EXT } \text{Points} \text{ BY } \text{AngleTeta}(X:\text{real},Y:\text{real}):\text{real} ; \)
Relation Points is :
Points = RELATION{  
  \text{Id\_Point } \text{char}(10) , \text{Abscissa } \text{real} , \text{Redius } \text{real} , \text{Color } \text{integer} ,  
  \text{OP(5) AngleTeta(Abscissa}:\text{real} , \text{Ordinate}:\text{real} \text{):real}}

\text{init (Points) is a procedure of initialization existing in the dbms}

\text{Inequations is a user type}

\text{There are two possible forms of this extension operator ; that is to say, explicitly or implicitly linked operator.}
KEY { Id_Point } ;

with AngleTeta defined by :

\[
\text{AngleTeta}(\text{Abscissa: real}, \text{Ordinate: real})
\]

begin
 for \( i = 1 \) to card(operande relation) do
 begin
 calculate the angle \( \theta \) from Abscissa and Ordinate,
 remove the coordinates Abscissa and Ordinate ,
 insert tuple of the operand relation into Points;
 end
end
Points \( \leftarrow \text{EXT} \) Points \( \text{BY} \) AngleTeta(X: real, Y: real): real
BEGIN/* Code of the operator AngleTeta */
END

Case (c): \( m > n \) and \( ai \in \{ b1 \ldots bm \} \); that case uses a redefinition of a user data type in its basic types [30,32].

Case (c): Global operator extension (Redefinition of a domain \( D \))
Example:

Let the relation Points defined by:

Points(Id_Point char(10), Redius real,Color integer, 
OP AngleTeta(Abscissa: real, Ordinate: real));

we have:

\[
\text{TYPE Segment} \{ \text{Begin : char(10), End : char(10)} \}
\]
Points \( \leftarrow \text{EXT} \) Points \( \text{BY} \) S : Segment ;
Relation Points becomes :

Points = RELATION{ Id_Point char(10), Redius real,Color integer, 
S : Segment, 
OP AngleTeta(Abscissa: real, Ordinate: real)}
KEY { Id_Point } ;
Points \( \leftarrow \text{EXT} \) Points \( \text{BY} \) Display(s: Segment); 
Relation Points becomes :

Points = RELATION{ Id_Point char(10), Redius real,Color integer, 
S.Begin : char(10), 
S.End : char(10), 
OP AngleTeta(Abscissa: real, Ordinate: real), 
OP Display(s: Segment)}
KEY { Id_Point } ;

4.4.4 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 characteristics \( Ci \) defined by an operator \( Op \).

\[
s \leftarrow \text{EXT} \ ALl \ r \ BY \ Op[(p_{t1}, p_{t2}, \ldots, p_{td})_{t_{s+1}}]
\]

\[
\begin{array}{c}
\begin{align*}
\text{BEGIN} \\
\langle \text{Op\_body} \rangle \\
\text{END.}\end{align*}
\end{array}
\]

\[H(s) = H(r) \cup \langle \text{Op, TOp} \rangle; B(s) = \{ ts | \forall tr \in Br \ (ts = tr) \cup \langle \text{Op, TOp, Exp(\text{Op})} \rangle \} \];

where \( \text{Exp (\text{Op})} \): the value, which defines the operator \( \text{Op} \) for a given query.

\( \text{TOp} \): Type specified by the expression \( \text{Exp (\text{Op})} \).

Local operator extension
Example:

Neighbourhoods \( \leftarrow \text{EXT} \) ALL Points \( \text{BY} \) Near_of(P : char(10), v : real) : Set (integer) ;

Let us consider the relation Points of the data base \( \text{BASE}^* \) :
Points = RELATION{ Id_Point char(10), Abscissa real, Ordinate real}
KEY { Id_Point } ;

So,

\[5 \] We notice well that the heading \( H \) of the Points has been reduced to support the angular coordinates ; that is to say, Redius, \( \theta \).
4.4.5 The Extraction of relation by operator: The relation extraction operator builds a relation from one or several operand relation $r_1$, $r_p$. Indeed, the operator $Op$ applied to headings $Hr(i)$ allows to define the result heading:

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

The operator $Op_1, Op_L$, in the relation result $s$, provides the possibility of expressing new situations in the database. Moreover, the operator $EXTRACT$ allows also to extract tuples $tr$ following the operator $Op (p_1, p_2... p)$ while respecting functional dependences binding the operand relations $r_1,..., r_p$:

$$s \leftarrow EXTRACT r \text{ FROM } r_1, r_2, ..., r_p \text{ BY } Op(p_1, p_2, ..., p_t)[t_{n+1}]$$

$$Hs = \{ \forall <X_i, T_i> \in Hr_i \exists <Op_i, TOp_i> ((<X_i, T_i> = <a, T>) \lor (<X_i, T_i> = <Op_i, TOp_i>)) \}$$

$$Bs = \{ ts \exists \exists tr \in \bigcup_{i=1}^{n} Hr_i \}$$

with $Hr (i)$, the heading of the relation $r(i)$ and ($i=1..n$). The operator $Op$ allows to extract entities of the type $tn+1$, according to the heading $Hr$ of the relation operand $r$, from the relations $r(1), r(2), ..., r(n)$. The attributes $<X_i, T_i>$ of the result relation $s$ are either attributes already existing in the values relations’ $r(i)$ or attributes $<Op, TOp>$ generated by the operator $Op$. 

Remark: $\bigcup_{i=1}^{n} Hr_i = \{ <X, T>, \exists r_i (\forall X, T \in Hr_i) \}$.

Extraction of a relation by operator OP

Let us consider relations Points and neighbourhoods in the respective sections (4.4.3 (a)) and (4.4.4) ;

Points = RELATION(
    Id_Point char(10), Abscissa real, Ordinate real,
    OP InSquare(Id_Point :integer, C :inequations) Boolean
) KEY { Id_Point }

Neighbourhoods = RELATION(
    Id_Point char(10), Abscissa real, Ordinate real,
    OP Near_of(P :char(10), v :real) Set (integer)
) KEY { Id_Point }

Now, let the following query:
4.4.6 Abstract data type generation: The definition of the relation notion, in an object relational model, as a heading $H_r$ and a body $B_r$ in the Date and Darwen’s formalism [1, 2] has allowed to approach the relation notion to the domain set $H_r=\{<a_1, T_1>, <a_2, T_2>, <a_n, T_n>\}$ where the management of the body $B_r$ of the relation does not have influence on the heading calculation. The integration of Operators $Op$ in the definition of an extended object relational model has allowed to express every heading $H_r$ of a relation $r$ under the form of an abstract data type $Hadt=\{<A_1, T_1>, <Op, TOp>, <Om, Tom>, <a_n, T_n>\}$. The generation operator of types $SHOW$ (a) expresses all the abstract data types in the heading $H_r=\{<a_1, T_1>, <Op, TOp>, <Om, Tom>, <a_n, T_n>\}$ following the nature of operator $Op$ and their 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 a such type or domain.

(a): $s \leftarrow SHOW[ALL] \mbox{ ADT } [<\mbox{ADT\_name}>] \mbox{ ON } r \mbox{ WHERE } <\mbox{Conditions}>
\begin{align*}
&\text{with } <\mbox{Conditions}> \text{ an expression on the heading } H_s \text{ and can be a definition by value or}
&\text{by variable parameters of the operator Op } (p_1:t_1, p: t) \\
&H_s=\{<X,T>/\exists<Op,TOp>\in H_r, (<X,T>\in H_r)\Rightarrow((<Op,TOp>\in H_s)\land(<Conditions>\equiv \mbox{True}))\} \\
&B_s=\{ts/\exists r \in H_r ((ts \in r) \land (ts \mbox{ conforme à } H_s))\}
\end{align*}
(b): $s \leftarrow SHOW[ALL] \mbox{ [<instance\_name>] }\mbox{ INSTANCE ON } <\mbox{ADT\_name}>
\begin{align*}
&\mbox{WHERE } <\mbox{Conditions}>
H_s=H <\mbox{ADT\_name}>\equiv \{<X,T>/\exists<X,T>\in H <\mbox{ADT\_name}> ((X=Op)\land(T=TOp))\} ; \\
&B_s=\{ts/((ts \mbox{ compliant with } H_s)\land(<Conditions>\equiv \mbox{True}))\}
\end{align*}
\begin{align*}
&\text{with } <\mbox{Conditions}> \text{ an expression on the body } B_s
\end{align*}

Let the relation Neighbourhoods in the section (4.4.4) :
\begin{align*}
\mbox{Neighbourhoods} = \mbox{RELATION}\{ & Id\_Point \mbox{ char}(10), \mbox{ Abscissa real}, \mbox{ Ordinate real}, \\
& \mbox{ OP Near\_of(P .char(10), v .real) .Set (integer)} \\
& \mbox{ KEY } (\mbox{Id\_Point}) ;
\}
\end{align*}

\begin{verbatim}
SHOW ADT V\_Proches ON Neighbourhoods
/* Query SHOW without conditions. The following query expresses 
All the abstract data types defining, possible Neighborhood. */
\end{verbatim}

---

6. Relation S is the union of relations Points and Neighbourhoods
7. Attributes Abscissa and ordinate result exclusively from one of two relations
5. **A*, an object and relational algebra**: The DBMS based on relational model is widely used today. This is essentially due to the integration, in these systems, of a query language based on the relational algebra [5]. Relational algebra has undergone several extensions to improve the relational model. Thus, new algebras are defined to meet the requirements of advanced database designers and users. These algebras, qualified as extended, contain besides the basic operations (i.e.: Union, Difference, Cartesian product, Projection, Restriction, Join) other derived operations (i.e.: Intersection, Division, Complement, Unflat, External Join, Transitive closure, extension or others). Consequently, relational object algebra \(A^*\) consists of relational operators based on the logic first order and algebraic extension operators. The possibilities of rewriting the operators of algebra \(A^*\) are shown in terms of those in Codd's relational algebra, when possible. Moreover, the algebraic operators in \(A^*\) are also compared to operators of standard object model (i.e. ODMG system).

5.1 **Propositions**: In order to build the algebra \(A^*\) closer to Codd's relational algebra, it is important to specify every operator \(<Op,TOp,Op(p_1t_1,p_2t_2,\ldots,p_nt_n)>\) as a relation \(r\) with \(n+1\) attributes where for every parameter \(p_i\ i\in[1,\ldots,n]\) we associate an attribute \(a_i\) and where the attribute \(a_i+1\) is the operator \(Op\) of type \(TOp\). Consequently, the operators of the algebra \(A^*\) can be considered, in certain cases, as relational operations.

**Proposition 1**: A scalar operator is an operator \(Op(x_1,x_2,\ldots,x_n):y\) which returns one or several scalar values \(y\) from \(n\) parameters \(x_1,x_2,\ldots,x_n\).

**Proposition 2**: Every scalar operator \(Op(x_1,x_2,\ldots,x_n):y\) is a relation \(r\) of the heading \(H_{Op}\)={\(x_1,t_1\),\(<x_2,t_2>,\ldots,<x_n,t_n>,<Op,y>\)} of body \(B_{Op}={t/t\ est\ conforme\ à\ H_{Op}}\);

The attributes \(x_1,x_2,\ldots,x_n\) determine the \(y\) attribute and are consequently a key for the relation.

**Proposition 3**: In an object relational database scheme, a relation \(r\) of the heading \(H\) and body \(B\) is defined by:

\(H={\langle a_1,T_1\rangle,\langle a_2,T_2\rangle,\ldots,\langle a_n,T_n\rangle}\) with \(Op(x_1,x_2,\ldots,x_n):y\)

\(B={t/t\ is\ compliant\ with\ à\ H}\);

and where \(t={\langle a_1,T_1,v_1\rangle,<Op,TOp,Op(p_1t_1,p_2t_2,\ldots,p_t)t_{n+1}>,\ldots,<an,Tn,vn}\};\)

The relation \(r\), in that case, can be considered as a functional structure binding a part of the heading \(H\) towards the rest of the heading \(H\). The subset of the attributes in \(H\) that determines the rest of the heading is called a key of the relation \(r\). In the case of a functional association \(A_{a},A_{b},\ldots,A_{n-1}\rightarrow A_{n}\) the relation \(r\) expresses a function \(f(A_{a},A_{b},\ldots,A_{n-1})=A_{n}\).

5.2 **Comparative description**: Let two relations \(r_1=(H_{r1},B_{r1})\) and \(r_2=(H_{r2},B_{r2})\) defined in an extended object relational database by:

\(H_{r1}={\langle a_1,t_1\rangle,\langle a_2,t_2\rangle,\ldots,\langle Opi,TOpi\rangle,\ldots,<an,t_n}\};\)

\(H_{r2}={\langle b_1,t_1\rangle,\langle b_2,t_2\rangle,\ldots,\langle Opj,TOpj\rangle,\ldots,<bm,t_m\}};\)

with \(Opi(p_1t_1,p_2t_2,\ldots,p_t)TOpi\); \(Opj(q_1t_1,q_2t_2,\ldots,q_t)TOpj\);  

The operator \(Opi\) and \(Opj\) have as input parameters \(p_1, p_2,\ldots, p; q_1, q_2, q\) (respectively as results \(Opi, Opj\)) of types \(t_1, t_2, t\) (respectively of types \(TOpi, TOpj\)). We note also that the extended relational object scheme evolves following the nature of the extension (ie descriptions in figure 3. are expressed according to the propositions (1,2,3) above). Finally,
the relational operators of the algebra $A^*$ has been moved closer in the Codd's [5] relational algebra while the extension algebraic operators has been compared on the one hand to the specification language of object scheme, ODL [6,7] and to the object relational language SQL3 [46] on the other hand.

Relational operators in $A^*$: Compared to the Codd's relational algebra, algebra $A^*$ contains five basic operators and two derived operators. In what follows we have elaborated a comparative study between the Codd's relational operators and those of the algebra $A^*$ defined in section 4 below (see Table 1.). The set of the basic operators of the algebra $A^*$ contains operators based on the first order logic (eg: $And^*$, $Now^*$, $Not^*$) besides a deletion operator $remove^*$ and an other one to rename the attributes $rename^*$. The derived operators $Tclose^*$ and $Compose^*$ are based on the composition and transitive closure operations, defined by Codd in [5].

![Diagram of various types of extension to the operators Op.]

Figure 3. Various types of extension to the operators Op.
<table>
<thead>
<tr>
<th>A* Relational operators</th>
<th>Relational algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>(r remove</em> ap+1,...,an)</td>
<td>If (∀ i ∈ {p+1,...,n}, ai is a basic type or an user type different from the type OP) then Π a1,...,ap (r) Else (∃ i ∈ {p+1,...,n}, ai is of type Operator); begin ∀ j ∈ {p+1,...,n} / aj is an operator Op do remove* OP; Π a1,...,ap (r); endElse</td>
</tr>
<tr>
<td>R And* s</td>
<td>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 natjoin s) Else (∃ i ∈ {p+1,...,n}, ai is of type Operator); begin ∀ j ∈ {p+1,...,n} / aj is an operator Op do ∪ Op; (r And* s)=(r x s)=res; res=res union ( The operators Op); res=res remove* a1,...,ap; endElse</td>
</tr>
<tr>
<td>r Or* s</td>
<td>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) remove* a1,...,ap) Else (∃ i ∈ {p+1,...,n}, ai is of operator type); begin ∀ 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</td>
</tr>
<tr>
<td>Not* (r)</td>
<td>If (Hr=Hs) Then (r + s)=(r And* (Not* (s))) ;</td>
</tr>
<tr>
<td>r Rename* (a,b)</td>
<td>Non - existant</td>
</tr>
</tbody>
</table>

(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) remove* a1,...,ap); Else(∃ i ∈ {p+1,...,n}, ai is of operator type); begin ∀ j ∈ {p+1,...,n} / aj is an 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 / (ai=aj)) Then begin r+= Tclose* (r) ; r+=r union ( Operators ) ; end |

**Table 1. Comparison A* / relational Algebra.**

**Extensions operators in A*:** The extension operators of the object relational scheme are classified into three extension types; that is to say, new type extensions, to operators (global or local) extension operators and to extension relation operators and (or) to abstract data type generation. The six extension operators defined in the section 4.4 are compared with those of the language ODL (eg, standard object scheme specification) and in those of the language SQL3 (eg, standard language for object relational database).
<table>
<thead>
<tr>
<th>A* Extension Operators</th>
<th>Object / Relational DBMS (SQL3)</th>
<th>Object DBMS (ODMG ODL / OQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) EXT r ADD an+1 BY Op(p1:t1,p2:t2,...,p:tn):tn+1</td>
<td>• 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]); Commit; /* Relation Br being already created. */</td>
<td>• 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:tn+1):an+1); } /* Keywords in bold correspond to ODL and OQL languages of the ODMG system. The inheritance exploitation (ie: Hext Hr) has allowed to realize this algebraic extension. It is important to note that the calculated 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 limited by the weaknesses of object models concerning the data interrogation language */</td>
</tr>
<tr>
<td>(2) EXT r ADD an+1tm+1 BY Op(p1:t1,p2:t2,...,p:tn+1)</td>
<td>• Add attribute an+ by assigning values determined as Null to [an+1]. • Query SQL3: ALTER TABLE Br ADD an+1 ; UPDATE Br SET Br.an+1=null; Commit ; • 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.</td>
<td>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); }</td>
</tr>
<tr>
<td>(3) EXT r BY Op(p1:t1,p2:t2,...,pt):tn+1</td>
<td>The table changement operator ALTER TABLE above allowing to add an operator Op in a relation is not available in the SQL3 database language. We propose the following possibility: • CREATE FUNCTION Op(p1 IN t1,p2 IN t2,..., p IN t) RETURN tn+1 BEGIN &lt;Op_body&gt; END; • The nature of the operator Op in that case determines the nature of the extension EXT. Indeed, Op application to the Hext domain allows either to specialize the latter ( see section 4.4.3 (a)) or to redefine it by regrouping certain types contained in Hext ( see section 4.4.3 ( b)) or unflat the type Hext into its elements or basic domains ( see section 4.4.3 ( c)). We note in that case that the scheme manipulations are not possible in the object model OMG who’s the definition language is ODL. 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.</td>
<td>• interface Hext :Hr (extent Bext) { attribute tn+1 Op; tn+1 Op(p1,t1,p2,t2,...,pt); } [BEGIN &lt;Op_body&gt; END.]</td>
</tr>
<tr>
<td>(4) EXT ALL r BY Op[(p1:t1,p2:t2,...,pt):tn+1]</td>
<td>• ALTER TABLE Br ADD Op tn+1; DECLARE CURSOR c IS SELECT * FROM Br; Row1 %ROWTYPE ; CREATE FUNCTION Op(p1 IN t1,p2 IN t2,..., p IN t) RETURN tn+1 IS Op tn+1;</td>
<td>• 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) {</td>
</tr>
</tbody>
</table>
Table 2. Comparison A* / SQL3 / (ODL, OQL).

| Remark: Algebra A* is composed of operators called relational, expressed in first order logic and algebraic extension operators. In fact, the A* completeness study and consequently any language based on A* (ie : ERA* Language ) depends on the object relational queries space to be expressed. From a purely relational point of view, algebra A* is complete in the sense where it offers the possibility of expressing any relational query (see comparison in the Table1.): in an object or object relational context, the algebra A* completeness and (or) any language based on A* should be proved. In fact, the expression of any object relational query requires an object relational query language being able to contain and realize any possible program. In our article, the purpose is to sketch a language prototype called ERA* (see section 6 below) which can be the kernel of a strong and powerful query language to manipulate logically (or) symbolically represented complex objects.

6. Language ERA*: language ERA* (Extended Relational Algebra * ) is an algebraic language based on the operators of algebra A*. Indeed, realization and support of the algebraic operators of extension offer 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] allows on the one hand to enrich the object
relational model seen by Melton [46] and Gardarin in [4]. On the other hand, it reinforces the data interrogation language by sophisticated operators for the resolution of some query classes. Consequently, the functionality integration offered by the language ERA* in SQL3 [46] improves the latter in its modeling and interrogation aspects.

6.1 Definition : A grammar G is a quadruplet G=<S,T,N,P> where S : Axiom ; T : Terminals ; N : Not terminals; P : language production rules

\[
T=\{\text{ext} | \text{add} | \text{by} | \text{in} | \text{out} | \text{var} | \text{begin} | \text{end} | \text{all} | \text{extract} | \text{from} | \text{adt} | \text{show} | \text{instance} | \text{on} | \text{;} | \text{,} | \text{.} | \text{)} | \text{(} | \text{vide} | \text{C} | \text{Pascal} | \text{SQL3} | \text{ERA} | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | \text{real} | \text{integer} | [ | ] | \text{array} | \text{relation} | \text{SET} | < | > | = | \text{and} | \text{or} | \neq \}
\]

\[
N=\{<\text{Axiome}>,<A>,<R>,<SR>,<ADT>,<T>,<Op>,<Parametres>,<C>,<B>,<LSR>,<ADTS>,<relation>,<attribute>,\text{Operator},\text{type},\text{Input\_Output},\text{Comm},\text{Type}C,\text{Code},\text{C\_language},\text{Pascal\_language},\text{Database\_language},\text{SQL3\_language},\text{ADT\_name},\text{instance},\text{digits},\text{alphabet},\text{Conditions(1)},\text{Conditions(2)}\}
\]

6.2 ERA* syntax:

\(1\) \(<\text{Axiome}>::=\text{not}<\text{relation}> | \text{remove}<\text{attribute}> | \text{remove}<\text{Operator}> | \text{Tclose}<\text{relation}> | \text{ext}<\text{R}> | \text{extract}<\text{SR}> | \text{show}<\text{ADT}> | <\text{Database\_language}\\
\)
\(2\) \(<\text{R}>::=<\text{A}>\text{add}<\text{T}> | <\text{A}>\text{by}<\text{Op}\\
\)
\(3\) \(<\text{A}>::=<\text{relation}> | \text{all}<\text{relation}>\\
\)
\(4\) \(<\text{T}>::=<\text{attribute}><\text{type}> | <\text{attribute}>\text{by}<\text{Op}\\
\)
\(5\) \(<\text{Op}>::=<\text{Operator}><\text{Parameters}><\text{type}> | <\text{Operator}><\text{Parameters}><\text{type}C | <\text{Operator}><\text{Parameters}\\
\)
\(6\) \(<\text{Parameters}>::=<\text{Input\_Output}><\text{attribute}><\text{type}> | <\text{Input\_Output}><\text{attribute}> | <\text{Input\_Output}><\text{attribute}><\text{type}><\text{Parameters}> | \text{empty}\\
\)
\(7\) \(<\text{Input\_Output}>::=\text{in} | \text{out} | \text{var} | \text{empty}\\
\)
\(8\) \(<\text{C}>::=\text{begin}<\text{B}>\text{end.}\\
\)
\(9\) \(<\text{B}>::=\text{Comm};<\text{Type}C;<\text{Code}; | \text{empty}\\
\)
\(10\) \(<\text{SR}>::=<\text{relation}> <\text{LSR}>\text{by}<\text{Op}\\
\)
\(11\) \(<\text{LSR}>::=<\text{relation}> | <\text{relation}><\text{LSR}> | \text{empty}\\
\)
\(12\) \(<\text{ADT}>::=\text{all}\ <\text{ADTS}> \text{where} <\text{Conditions(1)}> | <\text{ADTS}> \text{where} <\text{Conditions(2)}>\\
\)
\(13\) \(<\text{ADTS}>::=\text{adt}\text{-}\text{nom}\_\text{ADT}\text{on}\text{relation} | \text{instance}\text{-}\text{instance}\text{on}\text{nom}\_\text{ADT}\\
\)
\(14\) \(<\text{Conditions(1)}>::=<\text{expression} <\text{between} <\text{expression} (* \text{conditions on the headings Hs}*)\\
\)
\(15\) \(<\text{between}>::=<| > | = | \neq | \text{and} | \text{or}\\
\)
\(16\) \(<\text{expression}>::= (* \text{expressions on the tuples ts or headings Hs} *)\\
\)
\(17\) \(<\text{Conditions(2)}>::= (* \text{conditions on the tuples ts of Bs} *)\\
\)
\(18\) \(<\text{relation}>::=<\text{identifier}\\
\)
\(19\) \(<\text{attribute}>::=<\text{identifier}\\
\)
\(20\) \(<\text{Operator}>::=<\text{identifier}\\
\)
7. Discussions: Let us consider the database BASE* defined in section (3.3) and the set of the queries of the illustrative example of the section 1. Our objective in what follows is summed up in the presentation of the solutions of some types of queries by using both SQL3 [46] and ERA*.

7.1 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?

<table>
<thead>
<tr>
<th>Number of the query</th>
<th>SQL3</th>
<th>ERA*</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Select Abscissa, Ordinate</td>
<td>Ext ALL Points BY</td>
</tr>
<tr>
<td></td>
<td>From Points</td>
<td>InTown(Id_Point:integer,(A,C)):Boolean;</td>
</tr>
<tr>
<td></td>
<td>Where (Points.Id_Point='A') OR (Points.Id_Point='C')</td>
<td>Points REMOVE* Id_Point;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHOW ALL INSTANCE ON Points;</td>
</tr>
<tr>
<td>(2)</td>
<td>Select Distance(P1, P2)</td>
<td>Ext Points BY</td>
</tr>
<tr>
<td></td>
<td>From Points P1, Points P2</td>
<td>Eval_dist(Departure:char(10),</td>
</tr>
<tr>
<td></td>
<td>Where P1.Id_Point='A' AND P2.Id_Point='E'</td>
<td>Arrival:char(10)):real;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHOW ADT ON Points</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Where Eval_dist(x,y) AND x='A' AND y='E';</td>
</tr>
</tbody>
</table>

Remarks:

- 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:integer, 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 [18] in the language SQL3.
- We notice that the solution of the query (1) in language ERA*, besides the resolution, allows to enrich the relation Points by the operator InTown below.
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?

7.2 Queries of aggregates:

(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?

<table>
<thead>
<tr>
<th>Number of the query</th>
<th>SQL3</th>
<th>ERA*</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3)</td>
<td>Select P1.Id_Point From Points P1 Where (1&lt;P1.Abscissa&lt;20) AND (2&lt;P1.OOrdinate&lt;7)</td>
<td>(3.1) : Res = Extract Point_in_zone From Points By inzone(p:char(10),S):integer ;&lt;br&gt;(3.2) : Res = Res Remove* Abscissa, Ordinate ; ShowAll ADT On Res Where true;</td>
</tr>
<tr>
<td>(4)</td>
<td>Select P1.Id_Point From Points P1 Where Not_inzone(P1.Abscissa, P1.OOrdinate);</td>
<td>(4.1) : Res(<em>) = NOT</em>(Res);&lt;br&gt;(4.2) : ShowAll ADT On Res(*) Where not_inzone(p:char(10), S):boolean;</td>
</tr>
</tbody>
</table>

Remarks:

- The S system represents the zone Z of the figure 2 where S=\{1< x <20 and 2< y <7\}.
- The operators inzone(p :char(10),S :char(10)) :integer and not_inzone(p :char(10),S :char(10)) :boolean ; express the following two propositions (p \in S) and (p \notin S)

(A) Treatment of the query (3)

(3.1) : Res=RELATION\{ Id_Point:integer,<br>Abscissa:integer,<br>OOrdinate:integer,<br>OP inzone(C:char(10),S:char(10)):integer<br>\} KEY\{ Id_Point \}

(3.2) : Res=RELATION\{ Id_Point:integer,<br>OP inzone(C:char(10),S:char(10)):integer<br>\} 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>>\};

B_Res is shown by:

<table>
<thead>
<tr>
<th>Id_Point</th>
<th>Inzone</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
</tr>
</tbody>
</table>

(B) Treatment of the query (4)

(4.1) : Res*=RELATION\{ Id_Point:integer,<br>OP inzone(C:char(10),S:char(10)):integer<br>\} KEY\{ Id_Point \}
The body $B_{\text{Res}}^*$ of the relation $\text{Res}^*$ above is:

$$B_{\text{Res}}^* = \{ \langle \text{Id\_Point}, \text{char}(10), A \rangle, \langle \text{inzone}, 3 \rangle, \langle \text{Id\_Point}, \text{char}(10), B \rangle, \langle \text{inzone}, 4 \rangle, \langle \text{Id\_Point}, \text{char}(10), D \rangle, \langle \text{inzone}, 5 \rangle \}$$

(4.2) : A tabular representation of the body $B_{\text{Res}}^*$

<table>
<thead>
<tr>
<th>Id_Point</th>
<th>Inzone</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
</tr>
</tbody>
</table>

7.3 Topological Queries :

(5) What is the transitive closure of the road set in zone Z does not intersect with points of the line $Y=3$? What is the nature of the possible polygons?

The steps of resolution of query (5), in the example above, are :

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

Indeed, roads in the zone Z 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 not passing via points of the line $Y=3$ represent the set $SS$ such that:

$$SS = \{ \{P1,C\}, \{P1, P2\} \} \ (\text{see figure above}) ; \ (b)$$

And so the transitive closure $SS^*$ is: $SS^* = SS \cup \{P2, C\} = \{ \{P1, C\}, \{P1, P2\}, \{P2, C\} \} ; \ (c)$
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 namely :

(1) Possibility of defining and extracting new domains from the concrete database already defined;

(2) The definition of infinite objects but Logically finite

(3) Definition of non-specified property in priori in the scheme of the Object relational data base (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 calculation 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 [18], 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 calculation in data domain in the case of complex queries.

The treatment of query (5) above is:

<table>
<thead>
<tr>
<th>Language</th>
<th>Query (5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL3</td>
<td>The PSM / SQL module does not supply the possibility of integrating procedures PSM which can estimate all the roads of a zone generally. Still the complexities of algorithms being able to generate new domains are difficult to realize. So, it is important to express this type of result under the form of operators Op in our extended relational object approach.</td>
</tr>
</tbody>
</table>
| ERA*      | Given that the relations Points and Roads in the database ROAD_BASES *

Language Query (5)

| TYPE Cs Inequations ; /* Type of data formed of inequalities */
| Cs=\{1< x <20 and 2x<y<7\} ; /* Definition of the zone z */
| Cs1=\{Y=3\} /* Definition of the line Y=3*/
| P1=\{[A,C], Right Y=7\} ; /* The definition of the point P1*/
| P2=\{[A,E], Right Y=7\} ; /* The definition of the point P2*/
| P \textless EXT Points BY Inzone(Id_Point : integer , C : inequalities) : Boolean ;
| EXTRACT PCs FROM P BY Inzone(Id_Point : integer , Cs)
| Res \textleft= PCs AND* (P1,P2)* /* The set of points in the zone Z + (P1, P2)*/
| EXT Res BY AllRoads(Cs) ;
| EXTRACT S FROM Res BY BEGIN
| SELECT * FROM Res WHERE Id_Point NOT IN (Inzone(Id_Point ,Cs1))
| END
| Res* = TCLOSE* Res
| SHOW ALL INSTANCE ON Res Where TRUE

8. 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 relational object data base from concrete relations. We have shown the interest of such an extension of algebraic operators, comparing to stored procedures or PSM [18], 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 [46], a functionality dealing with the complex algorithms of calculation, modeling and querying for new applications. Indeed, on the one hand, the operator $OP$ in the definition $Hs=\{<a1,t1>,<an tn>,<OP1,TOP>,<OPL,TOPL>\}$ 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 to deal with new queries in database.
References


http://ercyes.ces.cwru.edu:80/jlin/cikm.ps


[47]: A.Goldberg and D.Robson. Smalltalk: The language and its Implementation. Addition-Wesley, 1983.