Extension of a Semi Structured Algebra for XML to Procedures Linked Operators

Safia Nait Bahloul and Djilali Benhamamouch
Computer Department, - Faculty of Science - Es-Sénia University
P. O. Box 1524 El Ménouar, Oran 31000, Algeria

Abstract

In this paper, we treat requests in XQuery in a system of mediation "every-XML".These requests use the operators of an algebra of semi-structured data which already exists and to which. we add a procedural aspect that treats the data of type document XML A mixed approach based on relational operators and models of data under shape of graphs, using timekeepers indexed in the process of reconstruction and course of road, is retained in this paper. One of algebras adopting this approach is algebra deended by G.Gardarin and T.T.D.Ngoc called XAlgebra. This XAlgebra contains at the same time: relational operations to treat the tables of XTuples and the operations of navigation for trees XML. Idea aims to extend this XAlgebra by procedures linked operators.. A type of operator accompanied by its procedures of syntax: *Xcext.. add.. by (procedures)* is integrated into the functional language XQuery and transforms expression FLWR into FLX$_C$WOR . This extension operates on the table of the XTuples and on the XOperations, extend consequently the XRelation formed by the XTuples and the associated trees. Extension becomes then a extended XOperator and is added to the existing XOperators to treat the XRelation. Syntax, semantics as well as the grammar of the FLX$_C$WOR are studied. The treatment of requests FLX$_C$WOR passes by a syntactic analysis, the construction of a plan of execution as well as a process of evaluation. The construction of a plan of execution passes by several stages, and in the stage of decomposition we adapt the rules of (normalization, canonization and atomisation) to the new expression FLX$_C$WOR. The evaluation of the procedure linked operator requires the conception of a new SAX.

Keywords: Semi-structured data, XML, semi-structured algebra, extension, XML-trees, procedures linked operator, Expression FLX$_C$WOR.

1. INTRODUCTION

The data semi-structured are data structure of which, unknown in priori, must be defined by deducing it, from the data it even. An example of data semi-structured are

documents XML (eXensible Markup Language) [1]. The models semi-structured [2, 3, 4] were introduced to answer two type of need application software : (1) to supply a model allowing to represent naturally very diverse information within the framework of the integration of data [5] and, (2) to be able to represent information little structured and evolving quickly that one finds in the databases of the genome [6] or on the canvas [7]. This information is treated then restructured from sources of multiple and diverse data. XML became then the standard for the exchange and the manipulation of these data on Web, success certainly due to its easily legible side, and to its auto-descriptive nature which allows to mix at the same moment the contents of the document and its own structure. The documents XML are typed through their DTD, but most of widely used languages are not, or little (XSLT, XPath, …). The ceaselessly increasing place of data semi-structured of which dominant standard is size XML led to define systems of data base adapted for this type of data. To evaluate a request on semi-structured data imply to navigate through the structure by examining at the same moment the values of elements and the auto-descriptive name of the element throughout the course. Several languages were proposed to make requests on data XML, XQuery [8] was retained as standard by W3C (World Wide Web consortium). The mediators taking into account semi-strutured data adopt all the architecture DARPA I3 with base of adapters [9], then with XML's advent the structures of data migrated to the model of data XML. XML became the ideal model of integration in architecture of access to diverse distributed data. Language XQuery offers the effective interrogation of requests on XML and the sheets of style XSL allowing to present data to the user. In an mediator 'every-XML' [10], XQuery is used as language of requests, XML as model of data in all the levels allow a modularity of constituents, XMLschema describes metadata and transitions are to be make by the SAX.
In this paper, we treat requests in XQuery in a system of mediation " every- XML " for which an algebra for data semi-structured already exists and to which we add a procedural aspect, not still exploite in XQuery, which treats data (documents XML) existing to create it and to add of news by means of the operators of extension.. In the section 2, we give an outline on XML and a typology on the languages of interrogation of documents XML as well as the various algebras, notably the XAlgèbre [10, 11]. In the section 3 are given syntax and semantics of an procedures linked procedure, an operator of extension. In the section 4 is exploited this operator in the language XQuery and in the algebra XAlgèbre where expression FLX_CWOR is studied. In the section 5 let us give the treatment of the request XQuery extended FLX_CWOR in particular the construction of the plan of execution, the section 6 studies the process of evaluation of the same expression and we end by the section 7.

2. THE LANGUAGE OF REQUESTS XML

One of the manners to represent a semi-structured data is to represent it under shape of a graph. Among a multitude of models, OEM [5] is the common model of semi-structured data. XML (eXtensible *Markup Language*)[4] is a standard of representation of documents, specified with the W3C. The W3C presented various objectives for a language of requests for XML [12]. A XMLSchema [13, 14] is one standards allowing to define and to typify a document XML in a formalism close to that of databases. The interrogation of semi-structured data requires the definition of a specific language of

requests. Abiteboul [15] defined the specificities of this new language as for example, existence of the operators of requests on databases (projection, selection, joint,), navigation in data (road, graph, forest), construction of the result (to foresee a mask for an answer), etc. Several languages of requests on semi-structured data, Xpath [16], XML-QL [17], XQL [18] were proposed before ending in the language of requests XQuery [8]. A comparative study of numerous languages of requests was made [19] which put in evidence the importance to decompose a request into three parts: a motive, a filter and a constructor. XQuery [8, 20, 21] was born from these recommendations.

XQuery : the language was conceived to allow easily understandable precise requests (FLWR), expression of the shape *for.. Let.. where... Return*. XQuery is Quilt's [22] by-product, borrowing numerous features of XPATH, XQL, XML-QL, SQL and OQL. It is a language of strongly typified functional type where every request is an expression of various types (expression of road, expressions FLWR, conditional expressions, functions) of which : for, Let, Where, Return.
Here is the expression of FLWR :

(1) for $var **in** *forest* [,$var **in** *forest*]*
(2) let $var := sub-tree
(3) Where *condition*
(4) return *result*

Let us give an example of request in XQuery containing an expression FLWR, an expression XPath, an imbrication (for imbricating for) and a function aggregate (contains) :

```
For j in collection ("customers") / customer /* collection indicates collection of documents* /
For i in collection (" commands.xml ") / command
  Where
       $j/name = “Lola”
         and
       Contains ($i/Sendingadress/ City /CityNamee, ’’Paris’’)
  Return
   <customers>
     <name>$i/name/text ( )</name>
     < First name >$i/ First name / text ( )</ First name >
     <telephone>$i/telephone/ text ( )</telephone>
     <city> $i/Sendingadress/city/cityname/text ( )</city>
   </customers>
```

Algebra for Semi-Structured Data: Several algebras were defined to allow the interrogation of semi-structured data. Some base themselves on the structure of graph as IBM [23], the others on the structure of tabular type, algebra Yat [24] is conceived specifically in the case of a system of integration based on XML. Algebra LORE [25] is based on the transformation of a request in a plan of execution using algebraic operators (Select, Project,..), algebra d'AT&T [26] is directly inspired by SQL, OQL and imbricated relational algebra. A mixed approach based on extented relational operators and model of data under shape of graph using indexed timekeepers is used by G.Gardarin and T.T.D.Ngoc in a frame of the evaluation of semi-structured diverse requests and in architecture of mediation "every- XML" [10, 11], it is about the XAlgèbre. The approach adopted by [10 , 11] is mixed. It uses extented relational operators while treating data of treelike type enriched by timekeepers indexed under tabular shape. This structure of algebra is based on streams of standard data SAX, and

contains a mechanism of indexation of trees arranged in a tabular way to treat the trees of semi-structured data. The adopted model of data is based on the notions of tree, navigation on trees (forest), bow, road, prefixes (sub-road), reference, XTuple, XAttribute and XOperators. From these notions is formed a XRelation. A XTuple is (constituted with the XAttributs necessary for the operators corresponding to roads in a tree and to collection of trees (called forest). The XAttributs is XPath making in reference nodes in trees XML. Knowing that the XAlgebra is conceived within the framework of an mediator, the construction of the XTuples is made by means of stream SAX, once adapters communicate results under shape XML to the mediator The XAlgebra is a physical algebra in the sense where algebraic expressions are used to treat streams XML and where algorithms treat operators. Several operators are defined in the XAlgebra, one finds: XRestriction, XProduit, XUnion, XOrderBy, XDiff, …An operator of departure allowing exists also to treat a source of data XML, Xsource Generate a set of Xtuples. Let us give an example of Xoperator :

Xprojection (π): π_X : $\chi \rightarrow \chi$, the operator retains only the nodes of every tree of the set of departure the road of which belongs to the set of roads X specified. The set of arrival is a set of Xtuples. The operator takes in argument a XRelation as well as a set of roads Xpath, X = {/ a / b, / a / c} for example, on which projection must be made, makes in exit a XRelation containing only data concerned with wanted Xpath. Some aspects of this algebra and XQuery were given so that we can introduce a procedural aspect necessary to enrich requests XQuery, missing aspct in the XAlgebra. This aspect takes offence by means of an operation of extension called also ***procedure linked operator*** which we define in the following section.

3. PROCEDURE LINKED OPERATOR

Procedures linked operators for relational databases and oriented objects can be introduced with standard operators into a language algebraic as SQL, SQL3 [27] and XQuery. Idea is inspired by the abstract types of data and the operation of extension defined by certain authors as Abiteboul [28] and Spaccaspietra [29], in relational and oriented systems object. In [28] is exposed a type of algebraic language for the structured objects including several classes of operators of which constants, operators on the sets, operations of filtering and extension . Another type of operation of extension was exploited in the works of [29, 30] to increase algebra MADS [31], ann algebra multi-attributes, where the operators treat so many types of object as types of association. The procedures linked operators allow to import a procedural aspect to extend or to transform a relation or an object. Implied procedures are written with the DBA or designers of generalized procedures. We introduce so our procedure linked operator as we call **extension**.

3.1. Definition of extension and call to the extension

Syntax

- ***Administrator***

ext < formal relname > **add** < formal attname > with <type> **by** < procname > [(< formal

parameters list > with <types>)] b**egin** <comment>;<code language>; <procbody> **end;**

- ***User of procedure***

cext <actual relname> **add** <actual attname> [with <type>] **by** <procname> [(<actual parameters list> [with <types>])];

< type formal attname>:= Simple or complex type

Semantics

An expert can write a procedure P (denoted by < procname>) with or without formal parameters (accompanied with their appropriate type) to form the values of the new attribute denoted by < formal attname > with ADT). The body of the procedure is written in any programming language (denoted <code language>, if there is an interpreter or compiler) and is accompanied with a comment. Procedures <procname> define so the contents of the attributes which one wants to add to the relation temporarily. A procedure may be applied to several relations and even in various classes of databases. The call of the extension is made later the choice of a procedure (explained in the comment) by an user according to its necessities by writing cext with the same name of procedure but by replacing <formal relname >, < formal parameters list> by their current correspondents with the compatible TDAs and syntactic word add allows addition. The to use does not have to rewrite The text of "**begin**" until "**end**" and does not need to understand the text, except the comment.

3.2. Examples

Let us show on an example of figures (the point) the introduction of the extension in SQL3 [27], for example the database :

```
Create type as object point
    ( number integer,
      abscissa interger,
      Ordinate integer,
      primary key pk-point number );
```

create points table of point;

Q1:

```
select  color
from points
where
(cext points add color: char(2) by colorer()) and p.color     = green)
```

Q2:

```
select  dist, p.number
from (cext points add dist: real by distance (points p, points q): real), points
where (dist<=0.5) and (p.number=10);
```

4. XQuery EXTENDED TO THE PROCEDURE LINKED OPERATORS: EXPRESSION FLX$_C$WOR

4.1. Signature of the operator extension: Xext et Xcext

· ***Definition***
Xext < formal collection> **add** < formal document > with <type> > **by** < procname > [(< formal parameters list > with <types>)] b**egin** <comment>;<code language>; <procbody> **end;**

· ***To the Call***
Xcext <actual collection> add <actual document> [with <type>] **by** <procname> [(<actual parameters list> [with <types>])];

<type formal document>:= simple type (Road) ou complex type (Tree)

4.2. Formal definitions

Our purpose is to be as well near as possible of the extented relational algebra while allowing the manipulation of trees and orderly collections of trees effectively. We introduce the procedure linked operators in the XRelation, which can be considered as a special case of the relations of objects, a domain being trees XML. Classically, a tree XML is a set of labelled orderly trees. Besides, crossed links can be supported as special edge. Let us redefine the concepts of Xalgebre extented to the procedure linked operators.

Definition1 : Tree$_{_Xcext}$

Let T_{Xcext} (r, $S_{\_Xcext}$, $E_{\_Xcext}$, $A_{\_Xcext}$, $P_{\_Xcext}$) the tree representing the model of data of documents XML. $S_{\_Xcext}$ is the set of the node of the graph, node can be of type element or element$_{_Xcext}$ or of type value ($S_{ement} \cup S_{String} \cup S_{Int} \cup \ldots, \cup$ Procname (P_1, P_2, …) with Pi actual parameters of the procedure), $E_{\_Xcext}$ represents the set of the edge of element and edge of element$_{_Xcext}$, $A_{\_Xcext}$ the set of the edge of attributes of element and the edge of attributes of element$_{_Xcext}$, $P_{\_Xcext}$ the set of the edge of element or of element$_{_Xcext}$ of referencing (timekeepers). r is the particular node ($r \in S$) corresponding in the root. A edge (edge$_{_Xcext}$) in the tree $T_{\_Xcext}$ is parent's relation $\in S_{\_Xcext\_element}$ towards s*on* $\in S_{\_Xcext}$ with a name *name* $\in T_{nom}$ where T_{nom} the set of character strings. One distinguishes three types of edge:

- The edge of element $E_{\_Xcext}$: one edge is said edge of element or edge of element$_{_Xcext}$ if the node source is a node element or of element$_{_Xcext}$ and the node destination can be a node element or node of element$_{_Xcext}$ or a node value ;

- The edge of attributes $A_{\_Xcext}$: are edge connecting an element or element$_{_Xcext}$ with its attributes ;
- The edge of referencing $P_{\_Xcext}$: a edge is said edge or edge$_{_Xcext}$ of referencing if the node source references the node destination of this edge.

Nodes representing element or element$_{_Xcext}$ and node representing values are ordered according to their appearance in the document XML. Now, attributes have no orders.

Definition 2 : edge neighboring $_{\_Xcext}$

A edge is called neighboring$_{_Xcext}$ to the other one if the node of element or of element$_{_Xcext}$ destination of the one is the node of element or of element$_{_Xcext}$ source of the other one.

Definition 3: Road$_{_Xcext}$

A road $C_{\_Xcext}$ is an orderly sequence of edge neighboring $_{\_Xcext}$ going from a node of element or of element$_{_Xcext}$ in the other one. A road from a node to the other one is not necessarily unique. A road is noted by the syntax XPath.

Définition 4 : Xtuple$_{_Xcext}$

One Xtuple$_{_Xcext}$ X is a couple (R_x, T_x) constituted by a set of references R_x= $\{R_1 \cup ..... \cup R_n \cup R_{\_Xcext}\}$ with R_i ($C_{\_Xcext}$), the function which from a road $C_{\_Xcext}$ sends the set of references associated to this road in the tree $T_{\_Xcext}$, and of a set of tree referenced $T_x = \{T_1 \cup ..... \cup T_n \cup T_{\_Xcext}\} \subset T_{\_Xcext}$; with $i \in [1 \ldots n, n+1]$, $R_i \subset C_{Ti}$.(the set of the roads of a tree Ti)

Definition 4 : Construction of Xtuple$_{_Xcext}$

The analysis by the mediator of the document XML by means of the SAX1 AND SAX2 (to see section 6.3) build the tree associated DOM. The structure of these trees results is called extented Xtuple or Xtuple$_{_Xcext}$.

Definition 5 : Xalgebre$_{_Xcext}$

The Xalgebre$_{_Xcext}$ is a collection of XTuples$_{_Xcext}$

Definition 6: The operators of Xalgebre$_{_Xcext}$ are used in the manipulation of extented requests XQuery, Request of the shape FLX_CWOR

4.2. General expression of FLX_CWOR

Generally, requests XQuery run within the framework of an expression FLWOR (**for**… **let**… **where**…**order by**… r**eturn**). We propose an extension of this expression by a clause **Xcext**. This clause extends temporarily a document XML or a collection of documents XML by a sub-tree, the values of the added nodes are calculated by means of a procedure (procname : being able to be written in language Java). The expression FLWOR of XQuery is extended to the operator of appeal of extension by a clause

Xcext. A new shape named FLX$_C$WOR represents the new expression of XQuery. The general shape of a FLX$_C$WOR :

for *$var* in *Forest*
Xcext *$var* **add** <RoadAdd (*$var*)> <TreeAdd(schema?) >
by { procName(arg1,…, argk) }
return R(*$var*)

$var* :** Variable sets has every iteration a tree of the forest ***forest
***RoadAdd*:** A declared expression of road XPath is which connects TreeAdd
TreeAdd: Document XML, data chosen by the user to be added to a *collection* of documents XML.

The clause Xcext is executed for every Xtuple produced with the clause for. The clause Xcext allows to add a sub-tree (TreedAdd) to the road XPath (RoadAdd). The sub-tree ArbreAdd is a document XML. We propose below grammar corresponding to the expression FLX$_C$WOR .

4.3. Grammar of FLX$_C$WOR

:General grammatical expression of the extented request Xquery

FLX$_C$WOR::= FLexpr WORexpr | FLexpr FOR (FOR| LET)* **Xcext** WORexpr
FLexp ::= (FOR | LET)+
WORexpr ::= WHERE? ORDERBY? "return" Expr

***Grammar of the fragment containing* Xcext :**

Xcext_Clause ::= "**Xcext**" "$" NameVar "**add**" "$"NameVar RoadPath "doc " " ("URL")""**by**"
MethodInvocation
MethodInvocation ::= IdentifyProced "("ArgumentList ")"
ArgumentList ::= Expression ArgumentList | Expression | ArgumentRoad
ArgumentRoad ::= "$" NameVar RoadPath ","ArgumentList|"$"NameVar RoadPath

The clause Xcext must be preceded with at least a clause for and symbols '+','*',' ?', possess same semantics that that of the grammar of the standard expression FLWOR

Semantics of Xcext : The treatment of requests is made in an mediator 'every XML' in the same way as the works of [10, 11]. A request user XQuery arrives so at the mediator, contains in our case Xcext, and undergoes several operations in the schema mediator/ adapter, to build a plan of execution. At the level of the mediator, Xcext is taken into account by the administrator of the plans of execution of a request in the same way as XUnion, XProjection or among the other XOperators. Xext allows to enrich libraries of functions used in XQuery by procedures compiled and ready for

execution formulated by an expert (by using Xext) according to the necessities of the users. During the decomposition of the request (decomposition which we shall develop in the section 5.2), initial request is decomposed into sub-requests and, on the meeting of the keyword add, the mediator loads source(s) target porter(s) of the collection to be extended to form sub-Tree (to add) by the SAX. Extension Xcext is different from both the XProduit and the XJointure defined in the XAlgebra because it is an operation which allows at first the creation of a sub-tree dynamically before adding it to a given tree. Xcext does not so operate on two trees in advance defined. So the XRelation is extended for each of its Xtuples by Xattribute generated with the procName and the procedure linked operator (Xcext) becomes an operator of the XAlgebra

Remark: FLX$_C$WOR is applied for one or several collections of trees

Note : The clause Xcext in our implementation extented request XQuery has to respect the following syntax:

Xcext *$variable* ***add*** RoadAdd ***doc*** *("uri")*
by { ***import*** *Name_Package.Name_Class ;*
Name_Method_of_class (param$_1$,param$_2$,…, param$_n$) ***}***

4.4. Extended XQuery: Example of expression FLX$_C$WOR

Example of request *:* One wants to calculate final price and sum the prices for the product for every customer of whom quantity is superior to a criterion given with reduction of the product, then to display information about the customers with product and the sum prices.

To answer this request, it is necessary to have appeal to procedures which do not exist necessarily in the catalog of functions XQuery. In this example, we need a procedure which calculates the reduction of the price according to the fixed criterion. The procedure linked operator Xext allows the expert to write procedure called calculate-reduction in XQuery, by explaining its sense in the comment and prepare all the informal parameters for an ultimate appeal. Xcext.

Request prepared by the DBA :

Xext forest **add** Road [<<Tree><Schema>] ***/****in the forest ''forest'' add a tree '' Tree'' following road ''road''*/* **by** calculate-reduction ($arg1:string, $arg2:string, $arg3:string,$arg4:string)

{
for $k **in** collection(''product'')/products
if $arg2= $k//code then if $arg1>$k /product/ Criterion-reduction then
let $r = ($k/product/price*$k/product/Rate-reduction)/100
let $s = $k/product/price - $r
let $som = $s * $arg1

```
text(arg4) = $s
text(arg3) = $som
}
```

In the appeal and on the example, the user can write his request as follows:

```
<command>
{
for $c in collection("commands")/command
Xcext $c add $c  doc (reduction.xml)
  by  {calculate-reduction ($c/quantity, $c/idproduct, $c/ reduction /sum, $c/reduction/price-reduce) }
where $c/reduction/sum > 20000
  return
   <result>
  <date-command>$c//date-command</date-command><price>$c/reduction/price-reduce</price>
  <sum>$c/reduction /sum</sum>
 for $v in collection(''products'')/product
 for $p in collection(''customers'')/customer
   where  $p//id = $c//id-customer and $v// code = $c//idproduct
    return
        <Display-customer>
         <name-customer>$p/name<name-customer>
         <firstname-customer>$p/firstname/firstname-customer>
         <product>$v//name- Commercial </product>
   </Display-customer>
< /result >
}
</command>
```

Interpretation of the example

Procedure **calculate- reduction** allows to calculate values associated to the document "reduction", **Xext** is associated to a single forest and the user can choose extension according to his need in consulting the comment. In the appeal, in this example, **Xcext** makes two actions :

1. by the syntactic word **add** the extension, the SAX creates the added sub-tree named reduction but with empty sheets (not still filled), to the exit of an adapter where is the collection in question to extend. At the level of the mediator is executed the procedure calculate- reduction in the plan of execution with the other operators of the XAlgebra. In the execution, the values of sub-tree "reduction" are calculated, this same procedure complies one or several times if need be.

2. takes as entrance the collection "command ("command"), the road collection ("commands')/ command and the document reduction.xml. In the collection "command", is added a tree of root "reduction" (associated to the document

reduction.xml) following road collection ("commands")/command. by the operation calculate-reduction (by calculate reduction (…)), Xcext calculates the reduced price for the product if the criteria of reduction are respected, and sum it prices for the same product associated to the same customer; then these values are affected in associated nodes ($c / reduction / price-reduce, c / reduction /sum).

5. TREATMENT OF THE XQuery REQUEST EXTENDED FLX$_C$WOR

The evaluation of a request XQuery within the mediator passes by the following steps:

1- *Analysis of the request*
2- *Creation of plan of execution*
3- *Evaluation and reorganization of results*

5.1. Analysis of the expression FLX$_C$WOR

The classic analyzer can not treat requests FLX$_C$WOR because the analysis of the request is to be made with regard to the interrogated types of data (XML-Schema). Now a request FLX$_C$WOR allows to reach data which do not appear any more in sources and also do not appear in the metadata (XMLSchema), semantic errors (lack of information) can arise. Our proposition consists in adding a *software* layer allowing to adapt the treatment of the request FLX$_C$WOR at the level of the mediator. This layer allows to constitute the first level of the analysis of the request. It consists of main modules:

- *Analyzer-Xcext* : Analyze request XQuery to verify if request contains the clause Xcext.
- *Check-RoadAdd* : This module consists in verifying if the clause Xcext contains the road on which one is going to add a given document and verifies its existence with regard to the metadata (XML-Schema).
- *Fusion-schema* : This module allows to merge two given schemas.
- *Error :* It is an administrator of errors.

If Analyser-Xcext finds a clause Xcext, Check-RoadAdd verifies the road which is after the keyword **Add** in the metadata. If this road answers the interrogated type of data, the Check-Roadadd module appeals to the Fusion-Schema module with the aim of merging two schemas corresponding respectively to the interrogated type of data and the document generated by Xcext. The rest of the request will undergo a traditional treatment expressed in the Analyzer-Mediator module [10, 11]. Check- RoadAdd is going to call upon the administrator of errors expressed with the Error module to generate a semantic error, in case the road corresponds to the interrogated type of data.

.

5.2. Construction of the plan of execution of a request FLX$_C$WOR

The simple construction of a plan of execution is to be made according to several steps [10, 11]. In the case of the extension Xcext, we are going to change the rules of the

normalization, the canonization and the atomisation defined [10, 11] to apply them to extended requests XQuery.

Let us l remind the definitions of the three phases of decomposition [10, 11].

5.2.1. Definitions

Definition 1. Normalization of a request: consists in eliminating as much as possible the imbrications.

Definition 2. Canonization of a request consists in regrouping the various blocks of evaluation of the request in sub-request named QDBi not containing reconstruction, and by generating a request of reconstruction called QMem.

Definition 3. Atomisation of request :consists in separating request in sub-requests concerning each a single collection of documrent and a request of connection sub-requests are called atomic requests.
In our case, appeal to the extension will undergo different ***canonization and atomisation***, we give the suited rules of transformation and we shall apply them like the section 4.3.

5.2.2. Rules of normalization, canonization and atomisation of a request FLX_CWOR

We leaned on the rules of normalization, canonization and of atomisation [10, 11] based on the rules of transformation detailed in the article of [32] and those made in the society e-XMLMedia. Indeed, a request XQuery with the clause Xcext is a written expression generally under the shape (and abbreviated shape) described in table1 :

FLX_CWOR	Simplified FLX_CWOR
for in E1, x2 **in** E2,…, xi **in** Ei,…,xn **in** En **Xcext** xi **add** <RoadAdd(xi)><TreeAdd(schema?) > **by {** procName(arg1,…, argk) **}** **where** C (x1,…, xi, …, xn) **return** R(x1,…, xi, …, xn)	**for** x **in** E **Xcext** xi **add** <RoadAdd (xi)><TreeAdd(schema?) > **by {** procName(arg1,…, argk) **}** **where** C (x) **return** R**(**x)
where n > 0 k >= 0	

Table1: Expression FLX_CWOR simplified

With the following notations: x, y, z indicate variable XQuery ; E, C, R requests XQuery.

- **Normalization**

The imbrications are eliminated in best and clauses let in an expression FLX_CWOR are considered as variable temporary of definition, are eliminated with the rule of normaliza described in table2 :

Rule 1

FLX_CWOR	**Normalized FLX_CWOR**
for x **in** E1 **let** y = E2(x) **for** z **in** E3(x, y) **Xcext** xi **add** <RoadAdd (xi)> <TreeAdd(schema?) > **by {** procName(arg1,…, argk) **}** **where** C (x, y, z) **return** R**(**x, y, z)	**for** x **in** E1, z **in** E3(x, E2(x)) **Xcext** xi **add** <RoadAdd (xi)> <TreeAdd(schema?) > **by {** procName(arg1,…, argk) **}** **where** C (x, y, z) **return** R**(**x, y, z)

Table 2: Rule of normalization

- **Canonization**

In this step we separate the blocks of request XQuery in several under requests as in the case of a request FLWR and in one of the QDBi exists one or several operator Xcext, to see table 3.

Rule2

FLX_CWOR	**Canonized FLX_CWOR**	
for x **in** E **Xcext** xi **add** <RoadAdd (xi)> <TreeAdd(schema?)> **by {procName** (arg1,…, argk) **}** **where** C (x) **return** R**(**x)	**let** T := **for** v1 **in** E **Xcext** xi **add** <RoadAdd (xi)><TreeAdd(schema?) > **by {procName** (arg1,…, argk)**}** **where** C (v1) **return** (v1)	**QDB1**
where v1 = x	R**(**x)	**QMem**

And

for x **in** E1 **Xcext** xi **add** <RoadAdd (xi)><TreeAdd(schema?) > **by {procName** (arg1,…, argk) **}** **where** C1(x) **return** R**(for** y **in** E2 **where** C2(x , y) **return** R(x , y **))**	**let** T1:= **for** v1 **in** E1 **Xcext** xi **add** <RoadAdd (xi)> TreeAdd(schema?) > **by {procName** (arg1,…, argk) **}** **where** C1(v1) **return** (v1)	**QDB1**
	let T2:=**for** v2 **in** T1 **for** v3 **in** E2 **w here** C2(v2, v3) **return** R(v2, v3)	**QDB2**
v1 = x where v2= x ∩ y v3= y \ x	R(v1 , v2, v3)	**Qmem**

Table 3: Rule of canonization

- **Atomisation**

Once canonized request, it is a question then of separating every request QDBi in sub-request concerning each a single collection of documents XML and, a request of connection QDEP. The atomisation is made according to the rule of atomisation [10, 11] containing the clause Xcext followed necessarily by the above-mentioned rule 4. Rules 3, 4 described respectively in tables 4 and 5 describe the phase of the atomisation**.**

Rule3

FLX$_C$WOR	**Atomised FLX$_C$WOR**	
for x **in** E Xcext xi **add** < RoadAdd (xi)> TreeAdd(schema?) **by {procName** (arg1,…, argk)**}** **where** C(x) **return** (x)	**for** x1 **in** E1 (x1) **where** Cx1 (x1) **return (**x1**)**	**REQ-A1**
	for x2 **in** E2 (x2) **where** Cx2 (x2) **return (**x2**)**	**REQ-A2**
	. . . .	
	for xi **in** Ei (xi) **Xcext** xi **add** < RoadAdd (xi)> <TreeAdd(schema?)> **by { procName** (arg1,…, argk) **}** **where** Cxi (xi) **return (**xi**)**	**REQ-Ai**

	. . . . .	
	for xn **in** En (xn) **where** Cxn (xn) **return (**xn**)**	**REQ-An**
	where C(x1, x2,…,xi,…,xn)	**QDEP**
	R (k1, k2,…, kn)	**Qmem**

Table 4: Rule 3 of Atomisation

On the example, we are going to apply two steps :

Step1 : normalization and canonization

QDB1 :
Let t1 = for $c **in** collection(‘’ commands’’)/command
Xcext$c**add**$c**doc**("reduction.xml")**byc**alculate-reduction($c//quantity,$c//idproduct,$c//description/sum,$c//description/price-reduce)
where $c//description/sum > 20000
return($c//date-command,$c//description/price-reduce,$c//description/sum,$c//idcustomer, $c//idproduct

QDB2 :
Let t2 = for $t **in** $t1
For $v in collection(‘’products’’)/product
For $p in collection(‘’customers’’)/customer
where $p/id = $t/idcustomer and
$ v//code = $t//idproduct
return
($p/name,$p/firstname,$v/namecmmand,$p/id,$v/code)

So, a request of reconstruction called Qmem is generated.

Qmem
<result>
<date-command>$c//date-command</date-command>
<price>$c//description/price-reduce </price>
<sum>$c//description/sum</sum>
<resultcustomer>
<Displaycustomert>
<name-customert>$p//name</name-customert>
<firstname-customer>$p//firstname</firstnamecustomer>
<product>$v//name-commercial</product>
</Displaycustomert>
</resultcustomert>
< /result >

Step2 : Atomisation

After the rule 3, for the QDBi containing Xcext, we apply the rule 4

Treatment of an atomic request with the extension : After the atomisation of the request, we add a procedure consisting in decomposing every atomic request which contains the clause Xcext (named **Qcext**) in two sub-requests **Qext1** and **Qext2**, this decomposition is to make following, to see table 5 :

Rule 4

Atomised FLX$_C$WOR		**Atomisation of Xcext in atomised FLX$_C$WOR**
Qcext =	**for** x **in** E **Xcext** xi **add** < RoadAdd (v2)> <TreeAdd(schema?) > **by {procName** (arg1,…, argk)**}** **where** C(x) **return (**x**)**	**let** ext1:= **for** v1 **in** E **return(r)** = **Qext1**
		let ext2**:= for** v2 **in** ext1 **Xcext** v2 **add** <RoadAdd (v2)> <treeAdd(schema?) > **by {ProcName**(arg1,…,argk)**}** **where** C(v2) = **Qext2**
	where v1 = x V1 = v2	

Table 5: Rule of atomisation (Left containing Xcext)

We apply rules 3, 4 of the atomisation to the continuation of the example :

Req-A$^1_1$
Let ext1:= **for** $c **in** collection (‘’commands’’)/command
return($c//date-command, $c//idcustomer, $c//idproduct, $c/quantity)

with **Req-A$^1_1$=Qext1**

Req-A$^1_2$
let ext2:= **for** $var **in** $ext1
Xcext $var **add** $var **doc**("reduction.xml")
by{calculate-reduction($var//quantity,$var//idproduct,$var/reduction /sum, $var/reduction/price-reduce) }
where $var/reduction/sum > 20000
return($var//date-command,$var//idcustomer,$var//idproduct,$var/quantity, $var/reduction/price-reduce, $var/ reduction /sum)

with **Req-A$^1_2$ = Qext2**

Req-A_3[1]
Where $c/reduction/sum > 20000

Req-connection2
No Req-connection : because we bring in only a single collection which is 'command'

Second request QDB2 not containing extension will follow the atomisation as in [10, 11] is so decomposed into 4 sub-requests of which one of the connection.

5.3. Atomisation of a request canonized with Xcext

Every atomic request containing the clause Xcext (named **Qcext**) must be decomposed into sub-request according the rule 4 before the creation of the plan of execution of the global request. Request Qcext returns a set of roads XPath (denoted by x) necessary to the execution of the global request (for example : roads used in logical expressions, of predicates which express joints, roads used as parameters of the procedure. Etc.). After the evaluation of this atomic request (Qcext), result can be sent under shape of a set of XTuples (XRelation). A XTuple is conmposed of two parts, the left part of the XTuple gives the XAttributes, that is the references to the node of the right part associated to roads XPath. Roads returned by the request Qcext build the right part of the XTuples. Furthermore, this set of roads x is constituted by two sets :

- The first set contains roads necessary for the execution of the global request and belonging to the collection **C** (the set of roads returned by the request Qext1);
- The second set contains the roads which must be added by the clause Xcext.

Atomic request Qext1 brings in only a single collection of documents XML **C**. Request Qcext1 returns a set of roads XPath (denoted by **r**).
The algorithm of construction of the set r
For every road into the set of roads X :

1- to verify if it belongs to the set of roads of collection C;
2- if the condition of the stage 1 is verified then to add this road to the set of roads r;
3- otherwise, not to make anything.

The mediator sends back atomic request Qext1 to the adapter. This last one estimates it locally and sends back in exit the result of this request under shape of stream SAX.
Request Qext2 should be taken into account by the mediator that is at the level of mediator this request is *translated* in certain one numbers of operators XOperatorr of the algebra used by the mediator, the *XAlgebra*. These operators enter the plan of execution of global request as being nodes of the corresponding tree at this plan of execution.

6. XQuery FLXCWOR QUERY ESTIMATION PROCESSES

The operator Xcext allows to add a sub-tree as soon as the XTuple are created by the analysis of stream of events SAX. This stream is sent after the execution of the request at the level of the adapter.

6.1.Evaluation of the operator Xcext

The procedure of evaluation of the operator Xcext is decomposed into two phases :
First phase
This phase allows to subject request Qext1 (to see the treatment of a request with the extension) to a collection of documents XML in one or several sources. A collection is identified with its name. A source is reached by way of its adapter and identified with its name. This stage sends back the result of the request Qext1 made with the adapter under shape of stream of events SAX
Second phase
This phase allows to get back the result of the request Qext1 (the result of the first phase) under shape of stream of events SAX. This stream SAX is read in the air to be transformed into tree referenced DOM (XTuple).The principle which we propose is to analyze the streams of events by means of a first analyzer SAX (that we shall name afterward ***SAX1***) which allows to build the tree DOM with all the roads (XAttributes) which will be necessary for the execution of the global request. These road XPath is determined with an analysis of the global request (given with the clause **return** of the request Qcext). This analyzer allows also to activate the appeal of a second analyzer SAX (that we shall name afterward ***SAX2***) after the construction of road RoadAdd given by the request Qcext**.** The analyzer SAX2 allows to analyze the document XML (TreeAdd) given by the user (document added temporarily) and to add the sub-tree corresponding to this document in the road RoadAdd.

In the second phase the analyzer SAX1 allows to build roads XAttributes. After the construction of road RoadAdd (given by the request Qcext), we activate the appeal of the analyzer SAX2 which allows to analyze a document XML and to build the sub-tree corresponding to this document and inserted into the tree progressively to be built

6.2. Call of procedure

All the nodes text (*Text_Node, to see 6.3*) of the added sub-tree must be empty. The values of these nodes are calculated by means of a procedure written in any *programming language*. The execution of this procedureis to be made at once after the creation of a XTuple (operation not blocking) to give the $Xtuple_{Xcext}$.

The execution of the procedure for every Xtuple (XTuple i)) follows the following stages:

- To identify the XAttributes used in the procedure;
- For every XAttribut:
  1-Made out a will if references a node in XTuple (i);
  2-Si the condition of the stage 1 is verified then to get back the value of node and to replace it in the procedure;
  3- else to treat the tuple XTuple (i+1);
  4- if selected attribute corresponds to the last parameter of the procedure then to execute procedure and pass in the XTuple (i+1).

6.3. Algorithm of the extended SAX

The transformation of a stream SAX in Xtuple$_{_Xcext}$ is made by analyzing the stream of events by the analyzer **SAX1** which appeals to the analyzer **SAX2**. These two analyzer SAX base themselves on mechanisms of **callbacks**.

SAX1

```
/// Callbacks corresponding to the analyzer SAX1
Stack   stack-XPath ;
XTuple  Xtuple-current ;
String Road - current ;
String Text - current ;
String  Array  XAttributes [1,.., n] ;

/*The   callbacks)*/

//* At the beginning of document XML
   startDocument( )
Begin
      xtuple-current = create – initialize XTuple();
 end   startDocument.
 //* At the end of document XML
 endDocument( )
     begin
         end Xtuple (xtuple-current) ;
      end   endDocument.
//* At the beginning of element
    startElement( String URI, String Name, String qName, Attributes attributes)
       begin
         String  Array  tab [1, ..,2] ;
            Road-Current := find-Road( ) ;
            tab [0] := Road-Current ;
            tab [1] := Name
            To stack (stack-XPath, tab) ;
        end  startElement.
//* end element
    endElement(String  URI, String  Name, String qName)
      begin
           integer  i ;
           Node Node-Current ;
         If (equal (Road-Current, Prefix -Add) Then
```

```
begin
      SaxAnalyzerAnalyzer-Sax :=    CreateAnalyzerSax() ;
      Call-method-Analyzer-Sax(Analyser (Document-Add, HandlerAnalyzer2)) ;
 end ;
 For i=0 until  lengh (XAttributes)
      Begin
 IF (equal(Road-current, XAttributes[i]))  then
   begin
     Node-Current := Realize(qName, Text-current) ;
            Connect Tree(Node-Current) ;
            reference (Node-Current) ;
            To go to Etiq ;
        end;
    else
      begin
 if(prefix(XAttributes[i],road-current) or prefix(Road-current, XAttributes[i]))
        begin
      Node-Current := realize(qName, Text-current) ;
      connectTree(Node-Current) ;
       go to Etiq ;
         end ;
      end;
    end ;
   Etiq :   depilate (stack-XPath) ;
 end     endElement.
//* Text associates to a node
 Characters (String text)
   begin
     Text-current := text ;
  end    Characters
```

After the execution of the atomic request containing the clause Xcext by the adapter, This one sends back result under shape of fragments XML. Every fragment XML corresponds to a new Xtuple (Called $XTuple_{\_Xcext}$) which is then created during this event (startDocument) The necessary roads (*XAttribute*) in the execution of the global request are then built by callbacks startElement (), endElement (), and Characters (). During the course of the tree, the useful node (that is being prefix of at least a XAttribute or a XAttribute corresponding to a prefix of road-current) are realized under shape of node DOM and inserted into the tree during its construction. If the followed road corresponds exactly to RoadAdd (road XPath given by the request Qcext, to see section 5.3)**,** in that case the appeal of the analyzer SAX2 is activated.

So the analyzer SAX1 activates the appeal of an analyzer SAX2 at once after the construction of road RoadAdd. The analyzer SAX2 allows to analyze a document XML TreeAdd (given by the request Qcext) and to send events. Every event activates the appeal of an associated function ***(callbacks)***.

Following algorithm defines callbacks (corresponding to the analyzer SAX2) which allow to insert a sub-tree (corresponding to a document XML TreeAdd) into the tree being built.

SAX2

```
///  Handler Analyzer 2 ///
/// callbacks corresponding to the analyzer SAX2
String  array  XAttributs [1,.., n] ;
String Road-Current ;
String Text-current ;
String  RoadAdd ;
/ *    callbacks */
// begin of document XML
startDocument( )
   begin
   end    startDocument.
//* end of document XML
endDocument( )
   begin
   end    endDocument
//* begin of element
startElement( String URI, String Name, String qName, Attributes attributes )
 begin
        String  Array  XPath [1, ..,2] ;
    road-Current:=  FusionRoad(RoadAdd, find-Road( )) ;
              XPath [0] := road-Current ;
              XPath [1] := Name
              To stack (stack-XPath, XPath) ;
end    startElement.
//* end of element
endElement(String  URI, String  name, String qName)
  begin
      integer  j ;
      Node Node-Current ;
      for j=0 until  lengh(XAttributes)
           begin
            if (equal(road-current, XAttributes)) then
              begin
          Node-Current := Realize(qName, Text-current) ;
                   ConnectTree(Node-Current) ;
                   Reference(Node-Current) ;
                   Go to exit;
               end ;
           else
             begin
       if(prefix(XAttributes,road-current)or prefix(road-current, XAttributes))
               begin
          Node-Current := realize(qName, Text-current) ;
                  ConnectTree(Node-Current) ;
                   Go to exit;
                end ;
            end ;
         end for ;
       Exit :    depilate (stack-XPath) ;
  end  endElement.

//* text associated to a node
Characters (String text)
```

```
begin
    Text-current := text ;
end  Characters .
```

Remarks

1. SAX2 uses the same stack as SAX1 (the stack serves for building trees),
2. We quote a new SAX (SAX2) because we need to add a tree and not a road,
3. callbacks beginDocument (), EndDocument () and text () is empty,
4. Text () is empty because the added tree compotes no values and it is at the level of the mediator that procedure.is executed,
4. BeginDocument () and EndDocument () is empty because we take a new Xtuple.

Note

Several architectures of mediation exist and adopt the architecture DARPA. Some use the model of data based on a directed graph OEM (TSIMMIS, STRUDEL), and use as language of interrogation respectively (OEM-QL, STRUQL). With the birth of XML, the mediator base themselves the treelike internal model of data (YAT, MIX). In our case :

The classic analyzer can not treat requests FLX_CWOR. because a request FLX_CWOR allows to reach data which do not appear any more in sources The architecture allows to show the main constituents of our system and how to store documents XML. At the lowest level is situated a set of collections (repertories) of documents XML which are stored in a system of files. Every collection contains one or several documents XML of same natural associated to a schema XML (XML-Schema or DTD), a collection indicated with an unique name. The system Extented XQuery is (constituted of three constituents, ***analysis of the request***, ***execution of the request***, *and* ***generation of the result***.

7. CONCLUSIONS

In this paper, we based widely ourselves on the works of the XAlgebra as at the level XRelation as at the level of the XOperators.The procedure linked operator is integrated into requests XQuery and enriched its expression in FLX_CWOR. Several treatments were necessary both at mediator and adapter levels to estimate a FLX_CWOR. Sub-requests Xcext deal in one or several sources of the part of recovery of roads to form sub-trees to be added and execute procedure to find the data of sub-tree in question, where from a SAX which transforms documents into Xtuples. Xcext can be written several times in the same request and be treated in the same way. When identical collections are in different sources, the operator Xcext can be made in common in the plan of execution of the main request. The optimization of the plans of execution must be treated taking into account algebraic equivalences to be defined.

8. REFERENCES

[1] T.Bray, J. Paoli, C. M. Sperberg-McQueenc. *Extensible markup language (XML*), 1.0 (2nd edition). Technical Report REC-xml-20001006,World Wide Web Consortium, 2000.

[2] S. Abiteboul. *Quering Semi-Structured data*. In Proc.of Inter.Conf.on Database Theory (ICDT), PP 1-18, Delphi, Greece, 1997.

[3] P. Buneman. *Semistructured data.* in Proc. of ACM Symposium on Principles of Database Systems, PP 117-121, Tucson, Arisona, 1997.

[4] D. Suciu. *Semistructured data and XML*. In Inter.Conf.on Foundations of Data Organisation (FODO), Japan, 1998.

[5] Y. Papakconstantinou, H. Garcia-Molina, J.Windom. *Object exchange across heterogeneous information sources*. In Proc. of IEEE Inter. Conf. On Data Engineering (ICDE), PP 251-260, Taiwan 1995.

[6] P. Buneman, SB Davidson, D.Suciu. *Programming constructs for unstructured data*. In Proc.of In.Workshop on Database Programming Languages, Gubbio, Italie, 1995.

[7] MF Fernandez, D.Florescu, AY. Levy, D. Suciu. *A query language for a web-site management system.* In SIGMOD Conference, PP 414-425, 1998.

[8] W3C. *An XML Query language (Xquery 1.0*), 2001

[9] G. Wiederhold. *Mediators in the Architecture of Futeure Information* System.Computer. Computer, 25(3): PP 38-49, 1992

[10] G. Gardarin, F.Sha, T.-T.Tang. *XML-Based Components for federating Multiple Heterogeneous data Sources*. In 18 Inter. Conf. on Conceptual Modeling, volume 1728 of LNCS, pages 506-519, Paris, 1999.

[11] G. Gardarin, A. Mensch, T.-T. Dang-Ngoc, L. Smit, *Integrating Heterogeneous Data Sources with XML and XQuery.* Workshops DEXA , PP 839-846, 2002.

[12] D. Chamberlin, P. Fankhauser, M. Marchiori, JRobbie. *Xml query (xquery) requirements* Technical Report 20030627, World Wide Web Consortium, 2003.

[13] H.Thompson, D.Beech, M.Maloney, N.Mondelsohn *XML Schema Part 1: Structures*, 2001.

[14] P.Biron, A. Malhotra. *XML Schema Part 2 : Datatypes,* October 2000

[15] S. Abiteboul, D.Quass, J. McHugh, J.Windom, J.Wiener. *The Lorel Query language for semi-structured data*. Journal of the Digital Library.1(1), pp 68-88, 1997

[16] J. Clark, S. DeRose. *XML Path Language (Xpath)* Version 1.0, 1999.

[17] A. Deutsch, M. Fernadez, D. Florescu, A. Levy, D. Suciu. *XML-QL: A Query Language for XML*, 1998.

[18] J. Robbie, J.lapp, D.Schaech. *XML Query language (XQL)*, 1998.

[19] M. Fernández, J. Siméon, P. Wadler, S. Cluet, A. Deutsch, D. Florescu, A. Levy, D. Maier,J. McHugh, J. Robie, D. Suciu, and J.Widom. *Xml query languages: Experiences and exemplars*, 1999.

[20] M. Fernandez D. Florescu S. Boag, D. Chamberlin, J. Robie, J. Siméon, and M. Stefanescu *XQuery 1.0: An XML Query Language*, 2001.

[21] M. Fernandez, Jérome Siméon, and Philip Wadler.*An algebra for XML query*. *Lecture Notes in Computer Science*, 1974, 2000.

[22] Don Chamberlin, Jonathan Robie, and Daniela Florescu. *Quilt: An XML query language for heterogeneous data sources*. *Lecture Notes in Computer Science*, 2001.
[23] D. Beech, A. Malhotra, M.Rys. *A formal data and Algebra for XML*, septembre 1999
[24] V. Christophildes, S. Cluet, J.Simeon. *On Wrepping Laguage and efficient XML Integration*. In ACM SIGMOD Conference on Management of data, 2000
[25] J. McHugh, J.Windom. *Query optimisation for XML*. On Proc. Of the 25 th Inter. Conf. On very large databases, Ecosse, septembre 1999.
[26] M. Fernadez, J. Simeon, , P. Walder. *A semi-Monad for semi-structured data*. In Inter.Conf..on database theory, January 2001.
[27] SQL3. Available on Web, http://www.objs.com/x3b7/sql3.htm
[28] S.Abiteboul, S.Grumberch. *Bases de donnéeset objets structurés*. TSI, Vol. 6, No. 5, May 1987.
[29] S. Spaccapietra, C. Parent, C. Vangenot. *GIS databases: From multiscale to multiRepresentation*. Abstraction, Reformulation, and Approximation, LNAI 1864, Springer, Proc. 4th International Symposium, SARA-2000, Horseshoe Bay, Texas, USA, 26-29 July 2000.
[30] S.Spaccapietra, C.Parent, E.Zimanyi, C.Vangenot. *MurMur:A Research Agenda on MultipleRepresentation*. Proc. of the Inter. Symposium on Database Applications in Non Traditional Environments, DANTE'99, Kyoto, Japan, 28-30 November 1999.
[31] C. Parent, S. Spaccapietra, E. Zimanyi, P. Donini, C. Plazanet, C. Vangenot, N. Rognon, J.Pouliot, P.-A. Crausaz. *MADS: un modèle conceptuel pour des applications spatio-temporelles*.*Revue Internationale de Géomatique,* Vol. 7, No 3-4, 1997.
[32] I. Manolescu, D.Florescu, D.Kossmann : Answering XML Queries over heterogenous data Sources. In Proceeding of 27th International Conference on Very Large data Bases (VLDB), pp. 241-250, Rome, Italie, Septembre 2001.