

Integration of XML Data

Deise de Brum Saccol¹ and Carlos Alberto Heuser²

¹Centro Universitário Luterano de Palmas
Av. Teotônio Segurado, 1501 SUL, Palmas, TO, Brazil
deise@ulbra-to.br

²Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500, Bloco IV, Porto Alegre, RS, Brazil
heuser@inf.ufrgs.br

Abstract. Various XML instances from different data sources can model the same object of the real world. Query processing or view definition over these sources demands instance integration. In this context, integration means to identify which data instances represent the same object of the real world, as well as to solve ambiguities of representation of this object. The entity identification problem in XML is more complex than in structured databases. XML data, as originally considered, necessarily do not have the identification notion of primary key or object identifier. Thus, it is necessary the adoption of a mechanism that identifies the instances at the moment of data integration. This paper presents a proposal for identifiers attribution to XML instances, based on the use of *Skolem* functions and *XPath* recommendation, as proposed by W3C.

1 Introduction

The available amount of electronic data has grown so fast in the last years. These data come in several formats, from completely unstructured file systems to structured information stored in relational database systems.

In order to integrate data from different sources, it is necessary to identify data instances representing the same object of the real world. The bibliography refers to this problem by the terms *object identification*, *instance identification* or *entity identification* [2].

However, not all the applications can be satisfied with the relational, object-relational or object-oriented models; examples are found in the WEB, for which the scheme at the moment they are created is not known. To support the integration of these types of applications, models of semistructured data have been considered. XML is being recognized as an important approach to represent semistructured data. XML data, as originally considered, do not demand identifiers. Even when they exist, the identifiers restrict to distinguish the instances inside a document, not being useful to the integration between different documents. At the same time, the lack of structure in these data makes impracticable many of the techniques used in the

integration of instances in structured data sources. Trying to decrease this deficiency, this paper presents a proposal of an approach of identification of XML instances.

The proposal of identification of XML instances presented in this paper is part of a Master Thesis developed at Federal University of Rio Grande do Sul [19], which focus on the access and integration of heterogeneous sources. XML data are extracted from the sources, integrated and materialized in a relational database. End user queries are posed directly over this database, using a language based on SQL.

For the materialization of XML data, two main activities must be carried through:

- the generation of the relational logical scheme (based on an ontology that describes the problem domain of the sources);
- the instantiation and incremental maintenance of this database, in order to keep the data consistency in relation to the sources.

The materialization module is showed in Fig.1.

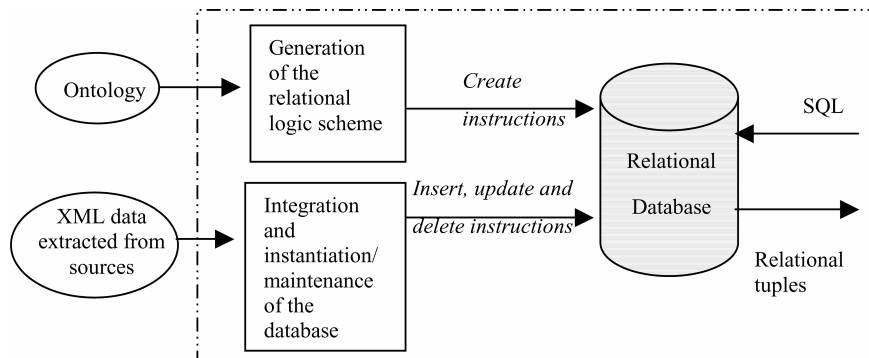


Fig.1. Materialization Module

The problem of instance identification arises at the moment of integration of the extracted data from the sources. This problem has been studied in the heterogeneous database area, which presents some proposals for solving it. Integration of XML data is a recent research area and here is the focus of this paper.

The present paper is organized as it follows. Section 2 presents the State of the Art about instance identification in heterogeneous database systems, along with the approach here proposed. Section 3 presents four approaches used on the properties values conflict resolution of a real world object, as well as the proposal adopted here.

2 Instance Identification

In this section, the main approaches used in heterogeneous databases to solve the problem of instance identification are pointed, as well as presented the approach proposed in this work.

2.1 State of the Art

Universal Key This is the simplest method for data integration. It is based on the existence of a common key between the instances to be integrated [2, 14, 16, 17, 23]. However, this approach is restricted, since the sources not always have a common key, as XML data.

Key Equivalence Specified by the User This approach requires that the user specifies equivalence between the instances, for example, using a mapping table of the local identifiers from each source to the global identifiers in the integrated system. This technique is used in [1, 18, 20]. The disadvantage of this technique is that the mapping table can be considerable and present difficult maintenance, handled by the database administrator, not in an automatic way.

Attributes Equivalence Not having a common key to integrate the instances, some proposals suggest using common attributes. For each pair of records from two sources, a matching value can be processed, which measures the degree of similarity between two instances. All the common attributes or only some of them (specified by the integrated system administrator) can be used to determine the object equivalence [4, 15, 22, 24].

Queries Defined for the User A query can be defined for each type of object of the real world, in order to construct a unique identification property for objects of this type. [8] uses this technique. Always when two instances of the same type return the same result for the query, they are considered the same object of the real world.

Inference Rules Lim [13] presents one technique for derivation of absent identifiers based on inference rules; the proposal uses extra semantic information to automate the process of identification between relations that do not share a common key. It considers the use of an extended key to identify instances, which can be defined as the union of keys of the participant data sources.

Boolean Functions Defined for the Designer When a common key between the data to be integrated does not exist, and when attributes in common that can be used in the identification of the instances do not exist, the designer can define a function for such intention. An example of this technique is presented in [24], which uses a function *close_names()*; this function returns a boolean value, depending on the similarity between two strings characters passed as arguments. This function could, for example, ignore names abbreviations in the process of instances identification, and consider that “Marcos Santos” and “Marcos A. Santos” refers to the same object of the real world.

The proposals found in literature for integration of XML data usually presume that the elements already have identifiers. The proposal of this paper is based on OWA (Open World Assumption) approach, since it aims to integrate data from the WEB. The OWA allows adding objects and improving knowledge of objects already stored. That is, the integrated system is open for new objects from WEB. In this way, the techniques based on the existence of an identifier in the data source (*Key Equivalence Specified for the User*, *Inference Rules*) are also excluded.

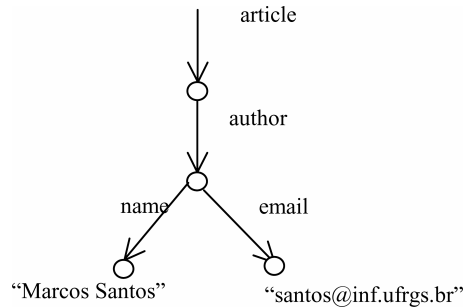
The technique *Attributes Equivalence* is applicable to the semistructured data, but it was not used in this paper for demanding great intervention of the user in the specification of the similarity degree of values.

As presented in section 2, one of the techniques for instance identification available in literature is the submission of queries over these data, in order to return the value of one or either a set of properties that uniquely identify it [8]. In this way, the proposed approach in this article aims to submit a query to the data sources, which returns the identifier of the instances. This means that two instances that model different objects of the real world never return the same result for this query. The technique here proposed combines the techniques *Queries Defined for the User* and *Boolean Functions Defined for the Designer*.

2.2 Technique Proposal for Identifying XML Instances

The data model adopted consists of a tree, representing the XML document, composed for nodes and edge. Nodes represent objects of the document, while the labels are represented in the edges. Complex objects point out other objects (for instance: object *author* in the tree below); atomic objects contain values of a basic type, represented for a string of bits (for instance: object *name* in the tree below). For example, the following XML document is represented in the tree below:

```
<article><author><name>Marcos Santos </name>
<email>santos@inf.ufrgs.br</email></author></article>
```



In object-oriented data [10], as well as in semistructured data (XML-QL [7]), the notion of instance identification has appeared in the form of *Skolem* functions. An alternative approach is the use of an *id function*, as introduced in [11].

The term *Skolem function* comes from the Logic of Predicates [9] and is used to assign a function that, from a given object, constructs its identifier.

XML-QL uses this type of function to transform data, for example, from a DTD to another one [7]. To illustrate this, consider the following part of a DTD that defines a type *author*:

```
<!ELEMENT author (firstname, lastname)>
```

One can write a XML-QL query that transforms data from this DTD into data that are in accordance with another DTD, for example:

```
<!ELEMENT person (lastname, firstname, address?,phone?,
publicationtitle*)>
<ATTLIST person ID ID #REQUIRED>
```

The query below extracts authors and transforms them into elements <person>.

```
WHERE
<$><author><firstname>$fn/><lastname>$ln/></></>
IN "www.a .b.c/bib.xml",
CONSTRUCT <person ID=PersonID ($fn, $ln)
<firstname>$fn/> <lastname>$ln/> </>
```

Always when an element <person> is produced, its ID becomes PersonID (\$fn, \$ln). PersonID is a *Skolem* function, and its intention is to generate a new identifier for each distinct value of firstname and lastname. A more formal description of *Skolem* Functions can be found in [9]. XML-QL does not specify as the *Skolem* functions are implemented.

The mechanism here considered consists:

- to use *Skolem* functions to identify objects;
- to implement the *Skolem* functions through queries in a XML language, specifically *XPath* [3].

It follows some examples of *Skolem* functions codified in *XPath*. It assumes that the first and last names of an author are enough to uniquely identify a data instance of this type. Thus, the query over this element must return these two strings of characters in order to identify them. Always when the same values are returned from the query, it will be considered that they are the same object of the real world; different results mean that the two data instances model two distinct objects.

The *Skolem* function could, for example, concatenate the two strings of characters, convert them in capital letter, eliminate the white spaces (if they exist), map them in a numerical value etc; the final result would be the identifier of this instance (ID).

Considering the same case and assuming two instances of XML data, from different sources, with distinct structures:

```
<person><lastname>Santos</lastname>
<firstname>Marcos</firstname></person>
```

```
<author><firstname>Marcos</firstname><middlename>Albert
</middlename><lastname>Santos</lastname></author>
```

Considering that the information contained in the labels <firstname> and <lastname> is enough to identify the instance; the idea is to submit one query to each data source in order to return the content from these labels, concatenate the two strings of characters and convert them in capital letter, generating the following identification: ID = MARCOSSANTOS. With this identifier, one can have the knowledge that both XML instances refers to the same object of the real world. So, this instance will be considered just once, at the moment of the data integration.

The example above was simplified for understandable needs. The necessary information for identification was represented directly in the labels <firstname> and <lastname>. A simple query in a XML language would return the expected values. However, when referring to semistructured data, this it is not always correct. The

necessary information for instance identification could be found inside a string of characters of one specific label; in this case, it is necessary a processing in order to extract it before the generation of the identifier. In another case, it could be necessary to extract subparts of some labels of the XML document, concatenate them and convert them in a specific format. For example, one could consider that a scientific event is identified by its name and the year of accomplishment. Such information is available in the following instance XML:

```
<article><author><name>Marcos Alberto Santos</name>
<email>santos@inf.ufrgs.br</email></author>

<title>Caching XML Data</title>

<event>Brazilian Simposium                               on
Databases, RJ, july 2001</event></article>
```

To generate the identifier of this event, it is necessary to extract the name of the event and the year of its accomplishment. Although this information is visible to a person, it is not represented in a directly way for a XML query language. The name and the year of the event are inside the label `<event>`, and must be extracted in order to generate the identifier of the instance.

To reach this goal, the *XPath* recommendation is used in the article, considered by W3C. *XPath* it is a language that allows to refer to parts of a XML document and to supply basic facilities to handle strings of characters, boolean and numbers [3]. This recommendation models a XML document as a tree of nodes. The basic syntactic constructor in *XPath* is the *expression*, a string of characters that consists of instructions to select an element, attribute, other markup structures, or a string of text [3]. They identify an item for its location in the hierarchic structure of the document. For example, the expression *author* selects children of the current element that has the name *author*. For current element, it is meant the current position in the tree of XML document. Another example is the expression *book/title*, which selects the title of an element *book*.

After evaluating an expression, the result is a set of selected nodes. Expressions can contain, recursively, other expressions used to filter sets of nodes, through the use of predicates: `author[email=santos@inf.ufrgs.br]`. The evaluation of the expression above returns the author whose email is the one specified in the string of characters.

Moreover, the *XPath* recommendation propose a varied set of functions for the handling of position elements tests (select the first author), values of attributes (select the attribute *edition* of the element *book*), manipulation of strings of characters (select the paragraph that starts with '*XML is a standard*'). A full description of *XPath* can be found in [3]. Basically, the proposal of the paper uses expressions and functions that make possible the manipulation of strings of characters: extraction of substrings of characters contained in elements, concatenation, conversion for upper/lower case, elimination of white spaces etc.

Returning to the example about the identification of an event for its name and year from accomplishment. The name of the event is available from the beginning of the label and extends to the first comma. Assuming that this is a regular standard in all the labels `<event>` of a specific XML document. One can take advantage of this fact, and using *XPath*, to extract the information of the event name, using the following

function: *string substring_before(string, string)*. This function returns the substring of the first argument that precedes the first occurrence of the second argument in the first argument. For example: *substring_before*("Brazilian Simposium on Databases, RJ, july 2001", ",") returns "Brazilian Simposium on Databases" (the several white spaces are on purpose). The year of the event also is necessary for identification; assume that this is always available in the four last positions of the label <event>. To extract it, one can use the function *substring*("Brazilian Simposium on Databases, RJ, july 2001", *string_length*("Brazilian Simposium on Databases, RJ, july 2001")-3,4).

This function returns the substring from the first argument, starting in the position specified in the second argument with the size specified in the third argument. Assuming that the function *string_length* returns the number of characters in the string passed as parameter, the expression above returns 2001. (for optimizations about the use of these functions, see [3]). Now, making use of the name of the event and its year of accomplishment, the identification of the instance could be generated. The *Skolem* function can carry through a processing on the name and the year of the event. The type of this processing is defined by the integrated system administrator. Consider that the generated identifier is a string of characters, resulted of the concatenation of the two arguments, previously converted to capital letter and without the presence of multiple white spaces between the words. Using the *XPath* functions, the steps are shown below. The partial results are attributed to temporary variables for better understanding:

1) eliminating white spaces: *a = normalize-space* ("Brazilian Simposium on Databases")

a = "Brazilian Simposium on Databases"

2) converting to capital letter:

b = translate (a, "abcdefghijklmnopqrstuvwxy", "ABCDEFGHIJKLMNOPQRSTUVWXYZ")

b = "BRAZILIAN SIMPOSIUM ON DATABASES"

The generated final identifier will be the result of the concatenation of the string of characters "b" with the year of the event: *ID = concat* (b, " ", "2001")

ID = "BRAZILIAN SIMPOSIUM ON DATABASES 2001"

Putting all the steps together, the responsible function for the identifier generation of an event in a specific XML source that follows the standards above would have the following style:

ID = (concat(translate(normalize-space(substring_before(event, ",")), "abcdefghijklmnopqrstuvwxy", "ABCDEFGHIJKLMNOPQRSTUVWXYZ"), " ", substring(event, string_length(event)-3,4))

ID = "BRAZILIAN SIMPOSIUM ON DATABASE 2001"

It must be pointed out that the *XPath* language includes an ample variety of functions that allow the manipulation of XML documents. This ample flexibility

allows the most varied handlings of XML instances, and the generated identifier can be resulted from complex processings in the structure of the document.

In the example above, one assumed that an event was identified for its name and year of accomplishment. Such assumption comes upon a study of the problem domain, and it is responsibility of the system designer that integrates the sources. This external knowledge of the domain is essential to make the instance identification.

As already stated, XML data are semistructured data. The necessary information for identification of one specific object (for example, event) can be settled in several ways when we go from one source to the others. In this way, a different *Skolem* function must be written to each source when the structure is not the same one. For example, assuming that in another XML document, the name of the event and its year of accomplishment were stored in separated labels, as shown below:

```
<event><nameEvent>Brazilian Simposium          on
Databases</nameEvent><yearEvent>2001</yearEvent></event>
```

The *Skolem* function for generation of the identifier of event for this source would be of the style:

$$ID = (\text{concat}(\text{translate}(\text{nameEvent}, \text{"abcdefghijklmnopqrstuvwxy"},$$

$$\text{"ABCDEFGHIJKLMNPOQRSTUVWXYZ"})), \text{" "}, \text{yearEvent})$$

$$ID = \text{"BRAZILIAN SIMPOSIUM ON DATABASES 2001"}$$

It notices that, despite the structure of the XML document vary between the sources, it is assumed that inside of a source it remains regular. In the example above, it is considered that the necessary information for the generation of the identifier was available in the labels `<nameEvent>` and `<yearEvent>`. If this rule is not satisfied in the entire XML document, the generated identifier will not be the expected one.

From the considerations above, it can be evidenced that the integrated system administrator is responsible for the definition of which information is necessary for identification of the entities to be integrated. Moreover, a *Skolem* function must be written to each XML source (assuming that its structures, labels and disposals of the necessary information are not the same ones) for each type of object to be identified; all these functions must return the same identification, when referring to the same entity of the real world.

3 Attributes Values Conflict Resolution

After two instances were identified as the same object of the real world, the values of its properties can present conflicting values, for example, two different addresses for the same author. By properties, we mean the attributes of an object. In this paper it is considered that object properties are represented in XML for atomic elements or attributes, or either, for leaves of the representative tree of an XML instance. For the handling of this type of conflict, some approaches had been proposed in available literature for structured databases. Along with the proposed approach for XML, these techniques are presented below

3.1 Conflict Resolution State of the Art

Aggregation Functions In this approach, the aggregation functions of the query languages are used (minimum, maximum, average, etc.) to solve conflicts in attributes values. For instance, considering the case that the value of the price of a book is represented with different values in two data sources; it could be used a minimum function and be considered only the minor value for the property *price* in the integrated system. This technique is proposed in [2, 5, 15, 18]. The disadvantage of this technique is that the used functions can restrict the type of attribute to which can be applied; as example, a minimum function can only be applied to numerical values.

Partial values When different values of a property cannot be mapped to only one value in the integrated system, a partial value can be generated. This partial value is a set of values, from which only one is considered correct [6]. This only value is resulted of the intersection of some conflicting values in this property. For example, considering that three XML instances model the same object *author* and present the following values for the property address: address 1 = “X Street, 110, Bananeiras District”, address 2 = “X Street, 110, Downtown” and address 3 = “X Street, 110”. The value considered correct in the integrated system would be the result of the intersection between the conflicting values, or either, “X Street, 110”.

Probabilistic partial values In this proposal, probabilities are attributed to the partial values. Extended selection operations can filter tuples that do not satisfy the query condition with the expected probability [21]. [14] proposes an extension to the relational model to store these conflicting values of the attributes. For example, considering the previous case, some method of attribution of probabilities to the conflicting values of the property *address* can be applied: address 1 = probability 0.66; address 2 = probability 0.33. A query specifying an expected minimum probability of 0.5 would return only the value from address 1 to the user.

Storage of all attributes values [15] considers a global object model that stores all the conflicting values of the attributes found in the data sources; the model allows user queries on attributes with the original values of the sources and with the solved values in the integrated model (for example, using aggregation functions for the resolution, whenever it is possible). Differently of the previous proposal, this technique does not assign probabilities to the conflicting values of the attributes; it allows to query all the values found in the sources for the same property of a real world object, as well as the solved value in the integrated system, whenever it is possible.

Analyzing these techniques, it is noticed that they do not have a generalized application, depending on the application and the property that is being considered.

3.2 Technique Proposal for Attributes Values Conflict Resolution

In this paper, we looked for a generic and applicable solution for data sources from WEB. This proposal joins two ideas, normalization of values and time stamps.

For *normalization of values*, we mean the transformation of the original values from the sources in a normalized value; this normalized value can be compared with the value from another source.

For instance, considering two data sources. The first one stores the information about ‘days of the week’ in Portuguese language format. The second one stores them in numerical format. To normalize the values, data from these sources could be converted to a canonic set of values. In this example, the values from both sources could be transformed in English language format. We propose to normalize these values using XSLT language.

Moreover, it is assumed that each data source has a time stamp which identifies its last update. The proposed mechanism consists, in case of conflict of the normalized value, to use the value that has the most recent time stamp.

Assuming that the author identified by “MARCOSSANTOS” is represented in two data sources, as shown below:

```
<name>Marcos A. Santos</name><address>X
Street,110</address>
```

```
<firstname>Marcos</firstname><lastname>Santos
</lastname><address>Y Street, 200</address>
```

Considering that source1.xml had the last update (time stamp) on January, 1st 2000 and the source 2.xml on July, 3rd 2000. Thus, the responsible module for the source integration verifies the time stamps of the sources that refer to this author and considers the address “Y Street, 200” as the correct one, since the source 2 is the one that was modified latest. The information about time stamps and about which sources refer to which integrated instances is necessary to adopt this technique.

Let’s take a look at another example concerning to the conflict resolution of value of properties:

```
<title>Caching XML Data</title><edition>2</edition>
<title @edition = "2a" Caching XML Data</title>
```

The property *edition* of the book is represented with different values in two data sources; in this case, a normalized value can be generated in order to match them. For example, we can use the function *translate* (*edition*, “1234567890^a”, “1234567890”) to get the value “2” of the property *edition*, which is represented in source2.xml for “2^a”. The function *translate* returns the first argument with occurrences from characters in the second argument substituted by the characters (in the corresponding position) from the third argument. If there is a character in the second argument without character (in the corresponding position) in the third argument, then the occurrences of this character in the first argument are removed. Thus, the two labels reflect the same value (2); only the normalized value is returned from XML sources and this value is used for matching the value of the property.

4 Final Considerations

XML data integration is a new research area. However, the problem of integration of heterogeneous data sources has been extensively studied in the latest years. Taking advantage of that, this proposal for XML instance identification is based on a deep

study of the alternatives already known for heterogeneous database systems, as seen in sections 2.1 and 3.1.

The proposed approach seems to be flexible:

- 1) Objects, represented for complex objects of a XML document (for example *author*, in the examples above) are identified by *Skolem* functions, codified as *XPath* queries to the original data sources.
- 2) Object properties, represented for atomic elements and attributes of XML, (for example, the book edition) are identified by its content, by default. However, a normalized value can be used as identification, using XSLT language. In the occurrence of conflicts in the normalized values, it is used information from the source that has the most recent time stamp.

At this moment of development, it is demanded the knowledge and the use of XSLT/*XPath*. To make the proposal usable for users who do not dominate these techniques, we are implementing a graphical interface, in which the user can choose between preconstructed skeletons of *XPath* queries. These skeletons can be defined from a grouping of the types of identifications that are used more frequently in XML sources. Examples could be the content of a property, the concatenation of the content of some properties etc.

We have used the Xalan processor for executing the *XPath* queries. Xalan-Java is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. It implements the W3C Recommendations for XSL Transformations (XSLT) and the XML Path Language (*XPath*).

References

- [1] Ahmed, R. et al. The Pegasus Heterogeneous Multidatabase System. Computer, New York, v.24, n.12, p. 19-27, Dec. 1991.
- [2] Albert, J. Data Integration in the Rodin Multidatabase System. In: International Conference On Cooperative Information Systems, 1., 1996. Papers. [S.l.:s.n.], 1996. p. 48-57.
- [3] Bradley, Neil. The XML Companion. 2nd ed. Harlow: Addison-Wesley, 2000.
- [4] Chatterjee, A. et al. Data Manipulation in Heterogeneous Databases. Sigmod Record, New York, v.2, n.4, p. 64-68, Dec. 1991.
- [5] Dayal, U. Processing queries over generalized hierarchies in a multidatabase system. In: International Conference On Very Large Data Bases, 9., 1983, Florence, IT. Proceedings... Florence: VLDB Endowment, 1983. p. 342-353.
- [6] Demichiel, L. G. Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains. IEEE Transactions on Knowledge and Data Engineering, New York, v.1, n.2, p. 485-493, Dec. 1989.
- [7] Deutsch, A. et al. A query language for XML. Journal WWW8/ Computer Networks, [S.l.], v.31, n.11-16, p.1155-1169. Available at: <<http://www.research.att.com/~mff/files/final.html>>. Access: Apr. 24 th, 2002.
- [8] Gogolla, M. Identifying Objects by Declarative Queries. In: Chomicki, Jan; Saake, Gunter; Sernadas, Christina. The Role of Logics in Information Systems. [S.l.:s.n.], 1995. (Dagstuhl-Seminar-Report, n. 121).

- [9] Hein, J. Discrete Structures, Logic and Computability. [S.l.]: Jones&Bartlett Publishers, 1995. Preliminary Edition.
- [10] Hull, R. et al. ILOG: Declarative Creation and Manipulation of Object Identifiers. In: International Conference On Very Large Data Bases, 6., 1990, Brisbane, AU. Proceedings... Brisbane: VLDB Endowment, 1990. p. 455-468.
- [11] Kifer, M. et al. Querying Object-Oriented Databases. In: International Conference On Management Of Data, 1992, San Diego, California, USA. Proceedings... San Diego: ACM Sigmod, 1992, p. 393-402.
- [12] Liefke, H. et al. Efficient View Maintenance in XML Data Warehouses. [S.l.]: Department of Computer and Information Science, University of Pennsylvania. Available at: <<http://www.cis.upenn.edu/~liefke/papers/whax.ps.gz>>. Access: Jan. 20 th, 2002.
- [13] Lim, E. et al. Entity identification in database integration. In: International Conference On Data Engineering, 9., 1993, Viena, AU. Proceedings... Viena: [s.n.], 1993. p.294-301.
- [14] Lim, E. et al. Resolving attribute incompatibility in database integration: An evidential reasoning approach. In: International Conference On Data Engineering, 10., 1994, Houston, US. Proceedings... Houston: [s.n.], 1994. p.154-163.
- [15] Lim, E. et al. A Global Object Model for Accommodating Instance Heterogeneities. In: International Conference On Conceptual Modelling, 17., 1998, Singapore. Proceedings... Singapore: [s.n.], 1998. p. 435-448.
- [16] Manolescu, I. et al. Agora: Living with XML and Relational. In: International Conference On Very Large Data Bases, 26., 2000, Cairo, EG. Proceedings... Cairo: VLDB Endowment, 2000. p. 623-626.
- [17] Papakonstantinou, Y. et al. Object Fusion in Mediator Systems. In: International Conference On Very Large Data Bases, 22., 1996, Bombay. Proceedings... Bombay: VLDB Endowment, 1996. p. 413-424.
- [18] Reddy, M. P. et al. A Methodology for Integration of Heterogeneous Databases. IEEE Transactions on Knowledge and Data Engineering, New York, v.6, n.6, p. 920-933, Dec. 1994.
- [19] Saccol, D. B. Materialização de Visões XML. 2001. Master Thesis (Master Course in Computer Science) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.
- [20] Shoens, K. A. et al. The Rufus System: Information Organization for Semi-Structured Data. In: International Conference On Very Large Data Bases, 19., 1993, Dublin, IR. Proceedings... Dublin: VLDB Endowment, 1993. p. 97-107.
- [21] Tseng, F. S. et al. A Probabilistic Approach to Query Processing in Heterogeneous Database Systems. In: International Workshop On Research Issues On Data Engineering: Transaction And Query Processing, 2., 1992, Tempe, US. Proceedings... Tempe: [s.n.], 1992. p. 176-183.
- [22] Wang, Y.R. et al. The Inter-Database Instance Identification Problem in Integrating Autonomous Systems. In: International Conference on Data Engineering, 5, 1989, Los Angeles, US, Proceedings... Los Angeles [s.n.], 1989, p. 46-55

- [23] Wiener, J.L. et al. The WHIPS prototype for Data Warehouse Creation and Maintenance. In: International Conference on Data Engineering, 13, 1997, Birmingham, UK. Proceedings...
- [24] Zhou, G. et al. Using Object Matching and Materialization to Integrate Heterogeneous Databases. In: International Conference on Cooperative Information Systems, 3, 1995, Viena, AU. Proceedings... 1995. p. 4-18.