

# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF INFORMATION SYSTEMS

## SUPPORT FOR XML IN ORACLE DATABASE SERVER

DIPLOMOVÁ PRÁCE

MASTERS'S THESIS

AUTOR PRÁCE

AUTHOR

DAVID SAN LEÓN GRANADO

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF INFORMATION SYSTEMS

## SUPPORT FOR XML IN ORACLE DATABASE SERVER

DIPLOMOVÁ PRÁCE

MASTERS'S THESIS

AUTOR PRÁCE

AUTHOR

DAVID SAN LEÓN GRANADO

VEDOUCÍ PRÁCE

SUPERVISOR

DR. ING. JAROSLAV ZENDULKA

BRNO 2007

## **Abstract**

This Mcs Thesis studies some XML technologies, focuses in the XML Databases. The XML databases are studied in general and are centred in Oracle 10g. Oracle 10g provides a lot of features to manage retrieve and process the XML data. To show some features to manage XML in Oracle Database Server is made a simple application that can create XML DB with structure and without it and to show the search features, in a particular XML Database (recipes for meals). Also is able to make different queries using Xpath to find recipes by title, by ingredients, by preparation and by nutritional information.

## **Keywords**

Oracle, XML, database, query, XSchema, XMLType, SQL, XQuery,UML.

# **SUPPORT FOR XML IN ORACLE DATABASE SERVER**

## **Declaration**

I declare that I have solved this Masters's Thesis by myself.

I have mentioned all information resources used in the thesis.

.....  
David San León Granado  
10-6-2007

## Acknowledgement

I would like to thank my supervisor, **Doc. Ing. Jaroslav Zendulka**, for his supervision and his help with me and this MSc. Thesis.

Thanks to all people that make the days of stay in Czech Republic more comfortable, especially to Erasmus people. Thanks to all friends in other countries who have been with me in the good times and the bad.

Finally, thanks to my family that has looking after me all my life with affection and confidence.

# Index

Index .....	1
Table of images .....	4
Table of examples.....	4
Introduction .....	5
PART I.....	7
1 Introduction.....	8
2 XML.....	9
2.1 What is XML?.....	9
2.2 History.....	9
2.3 Features of XML .....	10
2.4 XML does not DO Anything.....	12
2.4.1 XML is Free and Extensible .....	12
2.4.2 XML is a Complement to HTML .....	13
2.4.3 XML in Future Web Development .....	13
2.4.4 XML can separate Data from HTML .....	13
2.4.5 XML is Used to Exchange Data .....	13
2.4.6 XML and B2B (Business To Business).....	14
2.4.7 XML Can be Used to Share Data.....	14
2.4.8 XML Can be Used to Store Data .....	14
2.4.9 XML Can Make your Data More Useful.....	14
2.4.10 XML Can be Used to Create New Languages.....	14
2.5 XML document structure .....	14
2.5.1 Standards, extensions and schemas.....	15
3 Databases .....	17
3.1 Definition .....	17
3.2 Relational Model.....	17
4 XML Databases .....	19
4.1 Native XML Databases .....	19
5 Oracle XML DB .....	21
5.1 Features .....	21
5.2 Features .....	22
5.2.1 XMLType Datatype.....	22
5.2.2 The XMLType API.....	22
5.2.3 XML Schema Support .....	23

5.2.4	XML/SQL Duality .....	23
5.2.5	SQL/XML INCITS Standard SQL Functions .....	23
5.2.6	XPath Rewrite .....	24
5.3	Storing and retrieving XML Data in Oracle XML DB .....	24
5.3.1	XMLType Operations .....	24
5.4	XML Schema Storage and Query .....	27
5.4.1	XML Schema .....	27
5.4.2	XML Schema-Related Methods of XMLType .....	29
5.4.3	XMLType Tables and Columns Based on XML Schema .....	30
5.4.4	Oracle XML Schema Annotations .....	30
5.4.5	Generating XML Schemas with DBMS_XMLSCHEMA.GENERATESCHEMA .....	31
5.4.6	Creating constraints .....	32
5.4.7	Out-of-line Storage .....	32
5.4.8	Fully Qualified XML Schema URLs .....	33
5.4.9	Mapping XML Fragments to Large Objects (LOBs) .....	33
5.4.10	ComplexType Extensions and Restrictions in Oracle XML DB .....	34
5.4.11	Circular Schema Dependencies .....	34
5.5	Oracle XPath Extension .....	36
5.5.1	Functions to examine Type Information .....	36
5.5.2	Using XML Schema with Oracle XML DB .....	36
5.6	XPath Rewrite .....	37
5.6.2	Checking XPath rewrite .....	43
5.6.3	Examples of Rewrite of SQL Functions .....	44
5.6.4	Bind Variables .....	45
5.7	XML Schema Evolution .....	45
5.7.1	Style Sheet to update existing instance documents .....	46
5.8	Full-Text Search Over XML .....	48
5.8.1	The function CONTAINS .....	49
5.8.2	The score sql function .....	49
5.8.3	Structure operators .....	49
5.8.4	Context index .....	50
PART II	.....	51
6	Introduction .....	52
7	Analysis and design .....	52
7.1	Global vision .....	52
7.2	Requirements .....	52
7.2.1	Functional requirements .....	52

7.3	Use cases .....	54
7.3.1	Use case diagram .....	54
7.3.2	Use cases specification .....	54
7.3.3	The structure of the database .....	58
7.3.4	XML Schema selected for the database .....	58
7.3.5	Architecture.....	63
7.3.6	Structural diagram.....	63
7.3.7	User interface .....	65
7.3.8	Used technology.....	68
8	Conclusions and future works .....	70
9	Glossary .....	72
10	Bibliography and References .....	73
11	Appendices.....	75
11.1	Installation of the application.....	75
11.2	CD Contains .....	75



## Table of images

Figure 1 Example of out-of-line storage [6] .....	33
Figure 2 Example of mapping XML fragments to LOBs [6] .....	34
Figure 3 Crossed schema dependencies [6] .....	35
Figure 4 Self referencing [6] .....	35
Figure 5 Circular schema dependencies [6] .....	35
Figure 6 Use cases diagram .....	54
Figure 8 Structure of the recipe schema .....	61
Figure 9 Description of collection element .....	61
Figure 10 Description of recipe element .....	62
Figure 11 Description of ingredient element .....	62
Figure 12 Description of preparation element .....	62
Figure 13 Communication with the RDBMS [1] .....	63
Figure 14 Class diagram of the Simple Application .....	64
Figure 15 Manage XML databases .....	66
Figure 16 Add elements .....	67
Figure 17 Search Recipes .....	68

## Table of examples

Example 1 Simple XML document example .....	12
Example 2 XML Schema example .....	15
Example 3 Use of DBMS_SCHEMA.REGISTER SCHEMA [6] .....	29
Example 4 Use of DELETESHEMA[6] .....	29
Example 5 Use of GENERATESHEMA .....	32
Example 6 Use of REGISTERSCHEMA [6] .....	41
Example 7 Rewrite of query with XPath .....	43
Example 8 Use of CopyEvolve .....	48
Example 9 Fragment of the XML Schema that describe a recipe for meal .....	60

# Introduction

Nowadays, a lot of standards of the structure of documents and data are appearing, and this is a big problem when it is necessary to export from some structure to another one. But among those standards, XML appears and it solves some of these problems and it is widely used because it is open source. For a lot of people, XML is the future for the structured data; a lot of companies are developing XML applications for this reason.

For a long time the humans want to store the information that they get. With the new technologies, the quantity of information is bigger and it was necessary to find some place to put it without to use the traditional support (paper, stone,...). To solve this, the databases appeared using informatics technologies.

At the beginning, text and numbers was stored in the databases, but nowadays they store a lot of kinds of information, for example images, audio, videos or binary files. Due to this, a lot of kinds of databases appeared and of course, one of them was focused to XML data. Those databases are focused in the interchange of information using the XML characteristics. And after this, thanks to XML structure, in the text databases (databases that store text documents) the search is easier and faster. Now it is possible to find a XML databases and do queries like for example, find the books that contain a specific word in the preface. With the traditional text databases, it was very difficult and very slow, and then for instance, a lot of virtual libraries follow this way and translate the documents to specific XML structure to improve the searches.

Relational databases are the dominant data store by providing storage and processing mechanisms that offer both efficient techniques for storing structured data and high-performance query evaluation. On the other hand, XML is a newer, portable data format to exchange semi-structured data between disparate systems or applications. Businesses today require both XML and relational storage for easy exchange of data and added flexibility.

At present, the XML databases are not 'true' XML databases, the native databases. Oracle is near this; the Oracle 10g has native XML storage and retrieval technology. It fully absorbs the W3C XML data model into the Oracle Database, and provides new standard access methods for navigating and querying XML.

With the new database structure it is necessary to find another ways to make queries. For this, a lot of extensions appeared, like XPath and XQuery. These extensions helped to search and restrict the information. It is very important, for example, in the full text search. Other problem is the structure of

the XML, because XML only determines some rules for the document but not for the structure; at the beginning, it was not nothing to control this; because this, it was developed DTD, Schema and another extensions that create a specification, restrictions and rules for the document. For the XML databases it is very important to create dependencies between the different parts of the data and to validate the added elements in the database.

With the different formats it appears the problem of the difficulty of exchange data, XML solve this in a lot of situations. For instance, if some databases have to exchange data but the databases can have different structures and the data can have different formats, they have to have an intermediate structure for the data; normally is a XML structure and each database transforms their data in this format (all of them have to know this structure).

The actual databases have packages to transform XML in others formats, like HTML, XHTML or PDF. It is necessary if you want to show your information in net technologies like Internet. You can have the information in XML, create a XML style sheet and transform it to HTML, for instance, to put it on the web easily. This is very good because all web applications can have the same structure and only change the style sheet.

The structure of the document has two parts, the first part is for study deeply the different characteristics previously mentioned. The second part contains information about the design of the application, the engineering and specification. At the end of the thesis, the glossary, conclusions, references and bibliography and appendices (install manual and CD contains) appear.

# **PART I**

## **Background**

# 1 Introduction

This first part is divided in several chapters, the first one is the introduction, the second chapter is an overview of XML and XML extensions, XPath, Schema, XSLT, DTD... The third chapter contains talks about the databases in general. The fourth chapter talks about XML databases and their different kind of databases. Also, in the XML databases talks about natives XML databases and enabled XML databases.

The fifth chapter contains the studies over the Oracle XML DB features and explains the before chapter focused in Oracle Database Server.

## 2 XML

- The information presented in this chapter is based in [5], [9], and [11]

### 2.1 What is XML?

- XML stands for EXtensible Markup Language.
- XML is a markup language much like HTML.
- XML was designed to describe data.
- XML tags are not predefined. You must define your own tags.
- XML uses a Document Type Definition (DTD) or an XML Schema to describe the data.
- XML with a DTD or XML Schema is designed to be self-descriptive.
- XML is a W3C Recommendation.

### 2.2 History

The versatility of SGML for dynamic information display was understood by early digital media publishers in the late 1980s prior to the rise of the Internet.

By the mid-1990s some practitioners of SGML had gained experience with the then-new World Wide Web, and believed that SGML offered solutions to some of the problems the Web was likely to face as it grew. Dan Connolly added SGML to the list of W3C's activities when he joined the staff in 1995; work began in mid-1996 when Jon Bosak developed a charter and recruited collaborators. Bosak was well connected in the small community of people who had experience both in SGML and the Web. He received support in his efforts from Microsoft.

XML was designed by a working group of eleven members, supported by an (approximately) 150-member Interest Group. Technical debate took place on the Interest Group mailing list and issues were resolved by consensus or, when that failed, majority vote of the Working Group. The decision record was compiled by Michael Sperberg-McQueen on December 4<sup>th</sup> 1997. James Clark served as Technical Lead of the Working Group, notably contributing the empty-element “<empty/>” syntax and the name “XML”. Other names that had been put forward for consideration included “MAGMA”

(Minimal Architecture for Generalized Markup Applications), “SLIM” (Structured Language for Internet Markup) and “MGML” (Minimal Generalized Markup Language). The co-editors of the specification were originally Tim Bray and Michael Sperberg-McQueen. Halfway through the project Bray accepted a consulting engagement with Netscape, provoking vociferous protests from Microsoft. Bray was temporarily asked to resign the editorship. This led to intense dispute in the Working Group, eventually solved by the appointment of Microsoft’s Jean Paoli as a third co-editor.

The XML Working Group never met face-to-face; the design was accomplished using a combination of email and weekly teleconferences. The major design decisions were reached in twenty weeks of intense work between July and November of 1996, when the first Working Draft of an XML specification was published. Further design work continued through 1997, and XML 1.0 became a W3C Recommendation on February 10, 1998.

XML 1.0 achieved the Working Group’s goals of Internet usability, general-purpose usability, SGML compatibility, facilitation of easy development of processing software, minimization of optional features, legibility, formality, conciseness and ease of authoring.

Clarifications and minor changes were accumulated in published errata and then incorporated into a Second Edition of the XML 1.0 Recommendation on October 6, 2000. Subsequent errata were incorporated into a Third Edition on February 4, 2004.

Also published on the same day as XML 1.0 Third Edition was XML 1.1, a variant of XML that encourages more consistency in how characters are represented and relaxes restrictions on names, allowable characters, and end-of-line representations.

On August 16, 2006, XML 1.0 Fourth Edition and XML 1.1 Second edition were published to incorporate the accumulated errata.

Both XML 1.0 Fourth Edition and XML 1.1 Second Edition are considered current versions of XML.

## 2.3 Features of XML

If we speak about XML we have to speak about HTML, both are connected. We use XML and HTML in Web technology but we use them for different tasks.

XML and HTML are mark-up languages but are very different; the main difference is that XML was designed to carry data. Because of this XML is not a replacement for HTML, XML and HTML were designed with different goals:

- XML was designed to describe data and to focus on what data is.
- HTML was designed to display data and to focus on how data looks.

- HTML is about displaying information, while XML is about describing information.

XML provides a text-based means to describe and apply a tree-based structure to information. At its base level, all information manifests as text, interspersed with mark-up that indicates the information's separation into a hierarchy of character data, container-like elements, and attributes of those elements. In this respect, it is similar to the LISP programming language's S-expressions, which describe tree structures wherein each node may have its own property list.

The fundamental unit in XML is the character, as defined by the Universal Character Set. Characters are combined to form an XML document. The document consists of one or more entities, each of which is typically some portion of the document's characters, stored in a text file.

XML files may be served with a variety of Media types. RFC 3023 defines the types "application/xml" and "text/xml", which say only that the data is in XML, and nothing about its semantics. The use of "text/xml" has been criticized as a potential source of encoding problems but is now in the process of being deprecated. RFC 3023 also recommends that XML-based languages be given media types beginning in "application/" and ending in "+xml"; for example "application/atom+xml" for Atom. This page discusses further XML and MIME.

The ubiquity of text file authoring software facilitates rapid XML document authoring and maintenance. Prior to the advent of XML, there were very few data description languages that were general-purpose, Internet protocol-friendly, and very easy to learn and author. In fact, most data interchange formats were proprietary, special-purpose, "binary" formats (based foremost on bit sequences rather than characters) that could not be easily shared by different software applications or across different computing platforms, much less authored and maintained in common text editors.

By leaving the names, allowable hierarchy, and meanings of the elements and attributes open and definable by a customizable schema, XML provides a syntactic foundation for the creation of custom, XML-based mark-up languages. The general syntax of such languages is rigid — documents must adhere to the general rules of XML, assuring that all XML-aware software can at least read (parse) and understand the relative arrangement of information within them. The schema merely supplements the syntax rules with a set of constraints. Schemas typically restrict element and attribute names and their allowable containment hierarchies, such as only allowing an element named 'birthday' to contain 1 element named 'month' and 1 element named 'day', each of which has to contain only character data. The constraints in a schema may also include data type assignments that affect how information is processed; for example, the 'month' element's character data may be defined as being a month according to a particular schema language's conventions, perhaps meaning that it must not only be formatted a certain way, but also must not be processed as if it were some other type of data.



In this way, XML contrasts with HTML, which has an inflexible, single-purpose vocabulary of elements and attributes that, in general, cannot be repurposed. With XML, it is much easier to write software that accesses the document's information, since the data structures are expressed in a formal, relatively simple way.

XML makes no prohibitions on how it is used. Although XML is fundamentally text-based, software quickly emerged to abstract it into other, richer formats, largely through the use of data type-oriented schemas and object-oriented programming paradigms (in which the document is manipulated as an object). Such software might treat XML as serialized text only when it needs to transmit data over a network, and some software doesn't even do that much. Such uses have led to "binary XML", the relaxed restrictions of XML 1.1, and other proposals that run counter to XML's original spirit and thus garner criticism.

## 2.4 XML does not DO Anything

XML was not designed to DO anything, XML was created to structure, store and to send information.

The following example is a note to John from Tom, stored as XML:

```
<note>
<to>John</to>
<from>Tom</from>
<heading>Reminder</heading>
<body>Send me an email!</body>
</note>
```

### Example 1 Simple XML document example

The note has a header and a message body. It also has sender and receiver information. But still, this XML document does not DO anything. It is just pure information wrapped in XML tags. Someone must write a piece of software to send, receive or display it.

### 2.4.1 XML is Free and Extensible

The tags used to mark up HTML documents and the structure of HTML documents, are predefined. The author of HTML documents can only use tags that are defined in the HTML standard (like <p>, <h1>, etc.).

XML allows the author to define his own tags and his own document structure.

The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are “invented” by the author of the XML document.

## **2.4.2 XML is a Complement to HTML**

It is important to understand that XML is not a replacement for HTML. In future Web development it is most likely that XML will be used to describe the data, while HTML will be used to format and display the same data.

## **2.4.3 XML in Future Web Development**

XML is going to be everywhere.

We have been participating in XML development since its creation. It has been amazing to see how quickly the XML standard has been developed and how quickly a large number of software vendors have adopted the standard.

We strongly believe that XML will be as important to the future of the Web as HTML has been to the foundation of the Web and that XML will be the most common tool for all data manipulation and data transmission.

## **2.4.4 XML can separate Data from HTML**

When HTML is used to display data, the data is stored inside your HTML. With XML, data can be stored in separate XML files. This way you can concentrate on using HTML for data layout and display, and be sure that changes in the underlying data will not require any changes to your HTML.

XML data can also be stored inside HTML pages as “Data Islands”. You can still concentrate on using HTML only for formatting and displaying the data.

## **2.4.5 XML is Used to Exchange Data**

In the real world, computer systems and databases contain data in incompatible formats. One of the most time-consuming challenges for developers has been to exchange data between such systems over the Internet.

Converting the data to XML can greatly reduce this complexity and create data that can be read by many different types of applications.

## **2.4.6 XML and B2B (Business To Business)**

XML is going to be the main language for exchanging financial information between businesses over the Internet. A lot of interesting B2B applications is under development.

## **2.4.7 XML Can be Used to Share Data**

Since XML data is stored in plain text format, XML provides a software- and hardware-independent way of sharing data.

This makes it much easier to create data that different applications can work with. It also makes it easier to expand or upgrade a system to new operating systems, servers, applications, and new browsers.

## **2.4.8 XML Can be Used to Store Data**

XML can also be used to store data in files or in databases. Applications can be written to store and retrieve information from the store, and generic applications can be used to display the data.

## **2.4.9 XML Can Make your Data More Useful**

Since XML is independent of hardware, software and application, you can make your data available to other than only standard HTML browsers.

Other clients and applications can access your XML files as data sources, like they are accessing databases. Your data can be made available to all kinds of agents, and it is easier to make your data available for blind people, or people with other disabilities.

## **2.4.10 XML Can be Used to Create New Languages**

XML is the mother of WAP and WML.

The Wireless Markup Language (WML), used to markup Internet applications for handheld devices like mobile phones, is written in XML.

## **2.5 XML document structure**

A XML document has several defined parts and these parts are composed from other parts called elements that are signed with tags.

## 2.5.1 Standards, extensions and schemas

There are a lot of languages to manipulate XML data, to create schemas. The oldest schema format for XML is the Document Type Definition (DTD), inherited from SGML.

### 2.5.1.1 DTD Document Type Definition

A DTD is a set of rules that define the allowable structure of an XML document. DTDs are text files that derive their format from SGML and can be associated with an XML document either by using the DOCTYPE element or by using an external file through a DOCTYPE reference.

### 2.5.1.2 XML Schema

The most used language is XML Schema. XML Schema can be used to define a schema: a set of rules to which an XML document must conform in order to be considered 'valid' according to that schema. However, unlike most other schema languages, XML Schema was also designed with the intent of validation resulting in a collection of information adhering to specific data types, which can be useful in the development of XML document processing software.

A simple XML Schema Definition:

```
<xs:schema
xmlns:xs="http://www.chanle.org/XMLSchema">
<xs:element name="country" type="Country"/>
<xs:complexType name="Country">
<xs:sequence>
<xs:element name="name" type="xs:string"/>
<xs:element name="population" type="xs:decimal"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
```

#### Example 2 XML Schema example

### 2.5.1.3 Extensions

XML was created to structure, store and to send information, but sometimes it is not enough, we can need to find some information or leak it. XML don't support this and we need some extensions. Some extensions are: XPath, XQuery...

#### 2.5.1.3.1 XPath (XML Path Language)

XPath is an expression language for addressing portions of an XML document, or for computing values (strings, numbers, or Boolean values) based on the content of an XML document.

The XPath language is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria.

The most common kind of XPath expression is a path expression. A path expression is written as a sequence of steps to get from one XML node (the current ‘context node’) to another node or set of nodes. The steps are separated by “/” (i.e. path) characters.

#### 2.5.1.3.2 *XQuery*

XQuery is a query language (with some programming language features) that is designed to query collections of XML data. It is semantically similar to SQL.

The mission of the XML Query project is to provide flexible query facilities to extract data from real and virtual documents on the World Wide Web, therefore finally providing the needed interaction between the Web world and the database world. Ultimately, collections of XML files will be accessed like databases

XQuery provides the means to extract and manipulate data from XML documents or any data source that can be viewed as XML, such as relational databases or office documents.

XQuery uses XPath expression syntax to address specific parts of an XML document. It supplements this with a SQL-like “FLWOR expression” for performing joins. A FLWOR expression is constructed from the five clauses after which it is named: FOR, LET, WHERE, ORDER BY, RETURN.

The language also provides syntax allowing new XML documents to be constructed. Where the element and attribute names are known in advance, an XML-like syntax can be used; in other cases, expressions referred to as dynamic node constructors are available. All these constructs are defined as expressions within the language, and can be arbitrarily nested.

The language is based on a tree-structured model of the information content of an XML document, containing seven kinds of node: document nodes, elements, attributes, text nodes, comments, processing instructions, and namespaces.

The type system of the language models all values as sequences (a singleton value is considered to be a sequence of length one). The items in a sequence can either be nodes or atomic values. Atomic values may be integers, strings, Booleans, and so on: the full list of types is based on the primitive types defined in XML Schema.

## 3 Databases

- The information presented in this chapter is based in [1]

### 3.1 Definition

A database is an organized collection of data. One possible definition a database can be defined as a structured collection of records or data that is stored in a computer so that a program can consult it to answer queries. The records retrieved in answer to queries become information that can be used to make decisions. The computer program used to manage and query a database is known as a database management system (DBMS).

The term "database" originated within the computing discipline. Database-like records have been in existence since well before the industrial revolution in the form of ledgers, sales receipts and other business related collections of data.

The central concept of a database is that of a collection of records, or pieces of knowledge. Typically, for a given database, there is a structural description of the type of facts held in that database: this description is known as a schema. The schema describes the objects that are represented in the database, and the relationships among them. There are a number of different ways of organizing a schema, that is, of modelling the database structure: these are known as database models (or data models). The model in most common use today is the relational model, which in layman's terms represents all information in the form of multiple related tables each consisting of rows and columns (the true definition uses mathematical terminology). This model represents relationships by the use of values common to more than one table. Other models such as the hierarchical model and the network model use a more explicit representation of relationships but were given up because the relational model is better.

### 3.2 Relational Model

The relational model was introduced in an academic paper by E. F. Codd in 1970 as a way to make database management systems more independent of any particular application. It is a mathematical model defined in terms of predicate logic and set theory.

The products that are generally referred to as relational databases in fact implement a model that is only an approximation to the mathematical model defined by Codd. The data structures in these products are tables, rather than relations that are how it must be if we follow Codd's model: the main differences being that tables can contain duplicate rows, and that the rows (and columns) can be

treated as being ordered. SQL language which is the primary interface to these products treats Codd relations as tables as well. There has been considerable controversy, mainly due to Codd himself, as to whether it is correct to describe SQL implementations as "relational": but the fact is that the world does so, and the following description uses the term in its popular sense.

A relational database contains multiple tables, each similar to the one in the "flat" database model. Relationships between tables are not defined explicitly; instead, keys are used to match up rows of data in different tables. A key is a collection of one or more columns in one table whose values match corresponding columns in other tables: for example, an Employee table may contain a column named Location which contains a value that matches the key of a Location table. Any column can be a key, or multiple columns can be grouped together into a single key. It is not necessary to define all the keys in advance; a column can be used as a key even if it was not originally intended to be one if it has the correct properties like being unique for example. This property obeys to all values of this column to be different so it is possible to identify a row with the value of this column. This row is called unique key. If we use this column to refer to row it is called primary key. A key that has an external, real-world meaning (such as a person's name and surname, a book's ISBN, or a car's serial number), is sometimes called a "natural" key. If no natural key is suitable (think of the many people named Brown), an arbitrary key can be assigned (such as by giving employees ID numbers). In practice, most databases have both generated and natural keys, because generated keys can be used internally to create links between rows that cannot break, while natural keys can be used, less reliably, for searches and for integration with other databases. (For example, records in two independently developed databases could be matched up by social security number, except when the social security numbers are incorrect, missing, or have changed.)

XML database is a data persistence software system that allows data to be imported, accessed and exported in the XML format.

## 4 XML Databases

- The information presented in this chapter is based in [3], [4] and [5].

XML database is a data persistence software system that allows data to be imported, accessed and exported in the XML format. For the creation of this kind of database O'Connell gives one reason for the use of XML in databases: the increasingly common use of XML for data transport, which has meant that "data is extracted from databases and put into XML documents and vice-versa". It may prove more efficient (in terms of conversion costs) and easier to store the data in XML format.

There are two kinds of XML Databases:

- XML enable database.
- Native XML database.

The XML enabled databases are databases that hold data in some format other than XML. An interface is provided, however, so that XML can be presented to an application even though the data is stored in some other format. Often, these databases may have existing data that is now needed to be presented using XML. An XML-enabled database might be a relational database, object-relational database, or an object-oriented database. Some object-relational mapping tools are also designed to work with XML.

The native XML databases are a type of database allows XML data to be stored directly. Usually these mean populating a new database with the XML data. Native XML databases are likely to perform better than XML-enabled databases since there is little need for converting the data or that the conversion is minor. The data conversion in an enabled database is almost always going to be more significant and time consuming than with a native database.(for more information see 4.1 Native XML Databases)

The difference between XML-enabled and native storage is that XML-enabled storage uses schema-specific structures that must be mapped to the XML document at design time. Native XML storage uses generic structures that can contain any XML document.

### 4.1 Native XML Databases

The formal definition from the XML, DB consortium states that a native XML database:



- Defines a (logical) model for an XML document (as opposed to the data in that document) and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order. Examples of such models include the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX 1.0.
- Has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage.

It does not need to have any particular underlying physical storage model. For example, NXDs can use relational, hierarchical, or object-oriented database structures, or use a proprietary storage format (such as indexed, compressed files).

Additionally, many XML databases provide a logical model of grouping documents, called “collections”. Databases can set up and manage many collections at one time. In some implementations, a hierarchy of collections can exist, much in the same way that an operating system’s directory-structure works.

All XML databases now support at least one form of querying syntax. Minimally, just about all of them support XPath for performing queries against documents or collections of documents. XPath provides a simple pathing system that allows users to identify nodes that match a particular set of criteria.

In addition to XPath, many XML databases support XSLT as a method of transforming documents or query-results retrieved from the database. XSLT provides a declarative language written using an XML grammar. It aims to define a set of XPath filters that can transform documents (in part or in whole) into other formats including Plain text, XML, HTML, or PDF.

Not all XML databases support XQuery to perform querying. Some XML databases support an API called the XML: DB API (or XAPI) as a form of implementation-independent access to the XML data store. In XML databases, XAPI resembles ODBC as used with relational databases.

# 5 Oracle XML DB

- The information presented in this chapter is based in [6], [10], [12], [13] and [14]

Oracle XML DB is a feature of the Oracle Database. It provides a high-performance, native XML storage and retrieval technology. It fully absorbs the W3C XML data model into the Oracle Database, and provides new standard access methods for navigating and querying XML. Oracle XML DB gets all the advantages of relational database technology plus the advantages of XML.

## 5.1 Features

In general, the features of Oracle XML DataBases are:

- Support for the World Wide Web Consortium (W3C) XML and XML Schema data models and standard access methods for navigating and querying XML. The data models are incorporated into Oracle Database.
- Ways to store, query, update, and transform XML data while accessing it using SQL.
- Ways to perform XML operations on SQL data.
- A simple, lightweight XML repository where you can organize and manage database content, including XML, using a file/folder/URL metaphor.
- A storage-independent, content-independent and programming language-independent infrastructure for storing and managing XML data. This provides new ways of navigating and querying XML content stored in the database. For example, Oracle XML DB Repository facilitates this by managing XML document hierarchies.
- Industry-standard ways to access and update XML. The standards include the W3C XPath recommendation and the ISO-ANSI SQL/XML standard. FTP, HTTP(S), and WebDAV can be used to move XML content into and out of Oracle Database. Industry-standard APIs provide programmatic access and manipulation of XML content using Java, C, C++ and PL/SQL.

- XML-specific memory management and optimizations.
- Enterprise-level Oracle Database features for XML content: reliability, availability, scalability, and security.

## 5.2 Features

The major features of Oracle XML DB are these:

### 5.2.1 XMLType Datatype

XMLType is a native server datatype that lets the database understand that a column or table contains XML information. Datatype XMLType also provides methods that allow operations such as XML validation and XSL transformations on XML content.

The datatype that Oracle uses to store the content of the document as XML is the Character Large Object (CLOB). This datatype allows flexibility to store XML structures in a single table or column.

An important point in the features of XMLType is XMLType tables or columns can be constrained and conform to a XML schema. This is important because the database will ensure that only XML documents that validate against the XML schema can be stored in the column or table moreover Oracle XML DB with the information of the schema can provide more intelligent query and update processing of the XML.

### 5.2.2 The XMLType API

Datatype XMLType provides the following:

#### 5.2.2.1 Constructors

These allow creating XMLType from a VARCHAR, CLOB, BLOB or BFILE value.

#### 5.2.2.2 Methods

- **Extract()**: Extract a subset of nodes from an XMLType.
- **existsNode()**: check whether or not node exist in the XMLType.
- **schemaValidate()**: Validate the contents of the XMLType against an XML schema.

- **Transform()**: XLS Transformation.

## 5.2.3 XML Schema Support

Oracle XML DB support for the Worldwide Web Consortium XML Schema Recommendation.

XML Schema unifies both document and data modelling. In Oracle XML DB, you can create tables and types automatically using XML Schema. The user can create XML schema-based XMLType tables and columns and specify. This specifies conform to pre-registered XML schemas and are stored in structured storage format specified by the XML schema (this maintains DOM).

Oracle XML DB can store an XMLType object as an XML object that is based on an XML schema (LOBs or structure storage) or not based on an XML schema (LOBs).

## 5.2.4 XML/SQL Duality

XML/SQL duality means that the XML programmer can use the power of the relational model when working with XML content and the SQL programmer can use the flexibility of XML when working with relational content.

Oracle XML DB allows the relational and XML metaphors become interchangeable, the same data can be showed as rows in a table and manipulated using SQL or showed as nodes in an XML document and manipulated using DOM or XSL transformation, moreover access and processing techniques are independent of the underlying storage format. This allows that relational data can quickly and easily be converted into HTML pages, or get all of the information in XML documents without the overhead of converting back between different formats or text, spatial data, and multimedia operations can be performed on XML Content.

## 5.2.5 SQL/XML INCITS Standard SQL Functions

SQL/XML functions:

- **existNode:** Return true or false depending on whether or not the document contains a node that matches the XPath expression.
- **Extract:** Return the nodes that match the expression. If the return is multiple, the result will be a document fragment.
- **extractValue:** This operation takes an XPath expression and returns the leaf node.
- **updateXML:** This operation allows partial updates to be made to an XML document, moreover allows multiple updates to be specified for a single XML document.

- **XMLSequence:** This allow expose the members of a collection as a virtual table.

## 5.2.6 XPath Rewrite

Oracle XML DB can rewrite SQL statements that contain XPath expressions to purely relational SQL statements. The database optimizer simply processes the rewritten SQL statement in the same manner as other SQL statements.

XPath rewrite is not possible in all situations, only is possible when a SQL statement contains SQL/XML SQL functions or XMLType methods that use XPath expressions to refer to one or more nodes within a set of XML documents, an XMLType column or table containing the XML document is associated with a registered XML schema, an XMLType column or table uses structured storage techniques to provide the associated storage model.

If possible using the XPath, the process will have the following steps:

- Identify the set o XPath expressions included in the SQL statement.
- Translate each XPath expression into an object relational SQL expression than references the tables, types, and attributes of the underlying SQL.
- Rewrite the SQL statement to an equivalent object relational SQL statement.
- Take the new SQL statement to the database optimizer for plan generation and query execution.

## 5.3 Storing and retrieving XML Data in Oracle XML DB

### 5.3.1 XMLType Operations

#### 5.3.1.1 Selecting and querying XML Data

Oracle allows getting XML data from XMLType columns in different ways:

- Select XMLType columns in SQL, PL/SQL.
- Query XMLType columns directly or using XMLType methods `extract()` and `existsNode()`.
- Use Oracle Text operators.
- Use the XQuery language.

### 5.3.1.2 Searching XML Documents with XPath Expressions

XPath Construct	Description
/	Denotes the root of the tree in an XPath expression. Also used as a path separator to identify the children node of any given node.
//	Used to identify all descendants of the current node.
*	Used as a wildcard to match any child node.
[ ]	Used to denote predicate expressions. XPath supports a rich list of binary operators such as OR, AND, and NOT.  Brackets are also used to denote an index into a list.
Functions	XPath supports a set of built-in functions such as substring(), round(), and not(). In addition, XPath allows extension functions through the use of namespaces.  In the Oracle namespace, <a href="http://xmlns.oracle.com/xdb">http://xmlns.oracle.com/xdb</a> , Oracle XML DB additionally supports the function ora:contains(). This function behaves like the equivalent SQL function.

### 5.3.1.3 Selecting XML Data

Oracle allows selecting XMLType using the XMLType methods, the methods are the following:

- **getClobVal():** retrieve XML data as a CLOB value.
- **getStringVal():** retrieve XML data as a VARCHAR value.
- **getNumerVal():** retrieve XML data as a NUMBER value.
- **getBlobVal(csid):** retrieve XML data as a BLOB value.

#### 5.3.1.4 Querying XMLType Data with SQL Functions

- **Existnode(xmlType\_instance, XPath\_string,[namespace\_string]):** This function return 1 whether the given XPath path references at least one XML element node or text node, otherwise, it returns 0.
- **extract(XMLType\_instance, XPath\_string, [namespace\_string]):** This extracts the node (element, attribute or text node) or a set of nodes from the document identified by the XPath expression.
- **extractValue (XMLType\_instance, XPath\_expression,[namespace\_string]):** It returns a scalar value corresponding to the result of the XPath evaluation on the XMLType instance.
  - **Updating XML Instances and XML Data in Tables**
- **updateXML(XMLType\_instance,[XPath\_string,value\_expr], [namespace\_string]) :** Replace XML nodes of any kind.
- **insertChildXML(XMLType\_instance,XPath\_string,child\_expr,value\_expr,[namespace\_string]):** Insert XML element or attribute nodes as children of a given element node.
- **insertXMLbefore(XMLType\_instance, XPath\_expression,value\_expr,[namespace\_string]):** Insert XML nodes of any kind immediately before a given node.
- **appendChildXML(XMLType\_instance, XPath\_expression,value\_expr,[namespace\_string]):** Insert XML nodes of any kind as the last child nodes of a given element node.
- **deleteXML(XMLType\_instance,XPath\_expression,[namespace\_string]):** Delete XML nodes of any kind.

#### 5.3.1.5 Indexing XMLType Columns

##### 5.3.1.5.1 XPath Rewrite for indexes on singleton elements or attributes

Oracle XML DB attempts to rewrite the XPath expressions provided to SQL function `extractValue` into `CREATE_INDEX` statements that operate directly on the underlying objects.

##### 5.3.1.5.2 Creating Function-Based Indexes on XMLType Tables and Columns

A function-based index is created by evaluating the specified functions for each row in the table. It can be useful when the XML content is not managed using structured storage. In this case, instead of `CREATE INDEX` statement being rewritten, the index is created by invoking the function on the XML content and indexing the result.

##### 5.3.1.5.3 CTXXPATH Indexes on XMLType Columns

CTXXPATH index is based on Oracle Text technology and the functionality provided in the HASPATH and INPATH operators provided by the Oracle Text contains function. The HASPATH and INPATH operators allow high performance

XPath-like searches to be performed over XML content.

The CTXXPATH index is designed to rewrite the XPath expression supplied to SQL function existsNode into HASPATH and INPATH operators, which can use the underlying text index to quickly locate a superset of the documents that match the supplied XPath expression. Each document identified by the text index is then checked, using a DOM-based evaluation, to ensure that it is a true match for the supplied XPath expression. Due to the asynchronous nature of the underlying Oracle Text technology, the CTXXPATH index will also perform a DOM-based evaluation of all un-indexed documents, to see if they also should be included in the result set.

The creation of the CTXXPATH indexes follows the same way the creation of the Oracle Text indexes, using the following syntax:

```
CREATE INDEX [schema.]index
ON [schema.]table(XMLType column)
INDEXTYPE IS CTXSYS.ctxpath [PARAMETERS(paramstring)];
Paramstring: [storage storage_pref] [memory memsize] [populate
| nopopulate]
```

## 5.4 XML Schema Storage and Query

### 5.4.1 XML Schema

Oracle XML DB uses annotated XML Schemas as metadata, that is, the standard XML Schema definitions along with several Oracle XML DB-defined attributes. These attributes control how instance XML documents get mapped to the database. Because these attributes are in a different namespace from the XML Schema namespace, such annotated XML Schemas are still legal XML Schema documents.

The XML schema URL in Oracle is associated with parameter *schemaur*l of PL/SQL procedure DBMS\_XMLSCHEMA.registerSchema, the XML schema URL identifies the XML schema in the database.

Oracle XML DB provides XML Schema support for the following tasks:

Registering any W3C-compliant XML schemas.



- Validating your XML documents against registered XML schema definitions.
- Registering local and global XML schemas.
- Generating XML schemas from object types.
- Referencing an XML schema owned by another user.
- Explicitly referencing a global XML schema when a local XML schema exists with the same name.
- Generating a structured database mapping from your XML schemas during XML schema registration.
- Specifying a particular SQL type mapping when there are multiple legal mappings.
- Creating XMLType tables, views and columns based on registered XML schemas.
- Performing manipulation (DML) and queries on XML schema-based XMLType tables.
- Automatically inserting data into default tables when schema-based XML instances are inserted into Oracle XML DB Repository using FTP, HTTP(S)/WebDAV protocols and other languages.

If we speak about the XML Schema we have to speak about DTD, Oracle XML DB have supports it. In addition, to supporting XML Schema, which provides a structured mapping to object- relational storage, Oracle XML DB also supports DTD specifications in XML instance documents. Though DTDs are not used to derive the mapping, XML processors can still access and interpret the DTDs.

An important point in management of XML schemas is that before an XML schema can be used by Oracle XML DB, it must be registered with Oracle Database. It is possible register an XML schema using the PL/SQL package DBMS\_XMLSCHEMA. In this package, there are two important methods:

- **registerSchema**: this registers XML Schema. The main arguments are:
  - schemaURL: Identifier for the XML schema within Oracle XML DB.
  - schemaDoc: the XML schema source document. (VARCHAR, CLOB, BLOB, BFILE, XMLType or URIType).
  - CSID: the character-set ID of the source-document encoding, when schemaDoc is a BFILE or BLOB value.

An example of registering an XML Schema with  
DBMS\_XMLSCHEMA.REGISTERSCHEMA:

```
BEGIN
DBMS_XMLSCHEMA.registerSchema(
SCHEMAURL =>
'http://xmlns.oracle.com/xdb/documentation/purchaseOrder.xsd',
SCHEMADOC => bfilename('XMLDIR','purchaseOrder.xsd'),
```

```

CSID => nls_charset_id('AL32UTF8');
END;
/

```

### Example 3 Use of DBMS\_SCHEMA.REGISTER SCHEMA [6]

- **deleteSchema:** The first one registers one XML Schema and the second delete a previously created XML Schema.

Example:

```

BEGIN
DBMS_XMLSCHEMA.deleteSchema(
SCHEMAURL =>
'http://xmlns.oracle.com/xdb/documentation/purchaseOrder.xsd',
DELETE_OPTION => dbms_xmlschema.DELETE_CASCADE_FORCE);
END;
/

```

### Example 4 Use of DELETESchema[6]

- **copyEvolve:** Update a registered XML schema.

After XML schema is registered, is possible manage it, storing, accessing and manipulating XML instances that conform to the XML schema, then you can create types or default tables.

Also you can register a XML schema as local or global. The local XML schema is visible only to the owner and global schema is visible by all database users. In the registerSchema method there is a parameter where you can choose the kind of schema you prefer.

## 5.4.2 XML Schema-Related Methods of XMLType

- **isSchemaBased():** Returns TRUE if the XMLType instance is based on an XML schema, FALSE otherwise.
- **getSchemaURL(), getRootElement(), getNamespace():** Return the XML schema URL, name of root element, and the namespace for an XML schema-based XMLType instance, respectively.
- **schemaValidate(), isSchemaValid(), isSchemaValidated(), setSchemaValidated():** An XMLType instance can be validated against a registered XML schema using these validation methods.

### 5.4.3 XMLType Tables and Columns Based on XML Schema

Using Oracle XML DB, developers can create XMLType tables and columns that are constrained to a global element defined by a registered XML schema. After an XMLType column has been constrained to a particular element and a particular XML schema, it can only contain documents that are compliant with the schema definition of that element. An XMLType table column is constrained to a particular element and a particular XML schema by adding the appropriate XMLSCHEMA and ELEMENT clauses to the CREATE TABLE operation.

Syntax:

```
CREATE [GLOBAL TEMPORARY] TABLE [schema.] table OF XMLType
[(object_properties)] [XMLType XMLType_storage] [XMLSchema_spec]
[ON COMMIT {DELETE | PRESERVE} ROWS] [OID_clause] [OID_index_clause]
[physical_properties] [table_properties];
```

The data associated with an XMLType table or column that is constrained to an XML schema can be stored in two different ways:

- Shared the contents of the document and store it as a set of objects. This is known as structured storage.
- Stored the contents of the document as text, using a single LOB column. This is known as unstructured storage.

### 5.4.4 Oracle XML Schema Annotations

The schema annotation of Oracle XML DB gives application developers the ability to influence the objects and tables that are generated by the XML schema registration process. Annotation involves adding extra attributes to the *complexType*, *element*, and *attribute* definitions that are declared by the XML schema. The attributes used by Oracle XML DB belong to the namespace <http://xmlns.oracle.com/xdb>.

The most commonly used annotations are the following:

- **defaultTable**: Used to control the name of the default table generated for each global element when the GENTABLES parameter is FALSE. Setting this to the empty string "" will prevent a default table from being generated for the element in question.
- **SQLName**: Used to specify the name of the SQL attribute that corresponds to each element or attribute defined in the XML schema.

- **SQLType:** For complexType definitions, SQLType is used to specify the name of the SQL object type that corresponds to the complexType definitions. For simpleType definitions, SQLType is used to override the default mapping between XML schema datatypes and SQL datatypes. A very common use of SQLType is to define when unbounded strings should be stored as CLOB values, rather than VARCHAR(4000) CHAR values (the default).
- **SQLCollType:** Used to specify the name of the varray type that will manage a collection of elements.
- **maintainDOM:** Used to determine whether or not DOM fidelity should be maintained for a given complexType definition.
- **storeVarrayAsTable:** Specified in the root element of the XML schema. Used to force all collections to be stored as nested tables. A nested table is created for each element that specifies maxOccurs > 1. The nested tables are created with system-generated names.

## 5.4.5 Generating XML Schemas with

### DBMS\_XMLSCHEMA.GENERATESCHEMA

Oracle in the package DBMS\_XMLSCHEMA, offers us the functions generateSchema and generateSchemas, with those functions it is possible to generate a XML schema from an object-relational type automatically using a default mapping.

- **generateSchema:** This function returns an XMLType containing an XMLschema. It can optionally generate XML schema for all types referenced by the given object type or restricted only to the top-level types.
- **generateSchemas:** It returns an XMLSequenceType value. This is a varray of XMLType instances, each of them is an XML schema that corresponds to a different namespace. It also takes an additional optional argument, specifying the root URL of the preferred XML schema location

The following example shows the use of the generateSchema:

If we have the object type:

```
CREATE TYPE employee AS OBJECT(id NUMBER(10), name
VARCHAR2(200), salary NUMBER(10,2)):
```

Using generateSchema:

```
SELECT DBMS_XMLSCHEMA.generateSchema('EM','EMPLOYEE' FROM
DUAL;
```

This returns a schema for the type employee. The schema declares an element named EMPLOYEE and a complexType called EMPLOYEEType and adds other information from <http://xmlns.oracle.com/xdb>.

```
DBMS_XMLSCHEMA.GENERATESCHEMA('T', 'EMPLOYEE')
<xsd:schema targetNamespace="http://ns.oracle.com/xdb/T"
xmlns="http://ns.oracle.com/xdb/T1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xdb="http://xmlns.oracle.com/xdb"
xsi:schemaLocation="http://xmlns.oracle.com/xdb
http://xmlns.oracle.com/xdb/XDBSchema.xsd">
<xsd:element name="EMPLOYEE" type="EMPLOYEEType"
xdb:SQLType="EMPLOYEE" xdb:SQLSchema="T"/>
<xsd:complexType name="EMPLOYEEType">
<xsd:sequence>
<xsd:element name="EMPNO" type="xsd:double" xdb:SQLName="id"
xdb:SQLType="NUMBER"/>
<xsd:element name="ENAME" type="xsd:string" xdb:SQLName="name"
xdb:SQLType="VARCHAR2"/>
<xsd:element name="SALARY" type="xsd:double" xdb:SQLName="salary"
xdb:SQLType="NUMBER"/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

#### Example 5 Use of GENERATESCHEMA

### 5.4.6 Creating constraints

After creating an XMLType table based on an XML schema it is possible to create constraints, for instance, to create constraints on elements that occur more than once, it stores the varray as a table (Ordered Collections in Tables (OCT)) and after it will create the constraints.

### 5.4.7 Out-of-line Storage

In some scenarios the out-of-line storage offers better performance than the default storage. In those scenarios the SQLInline attribute can be to false, then Oracle XMLDB generates an object type with embedded REF attribute. REF points to a XML fragment that gets stored out-of-line.

For example:

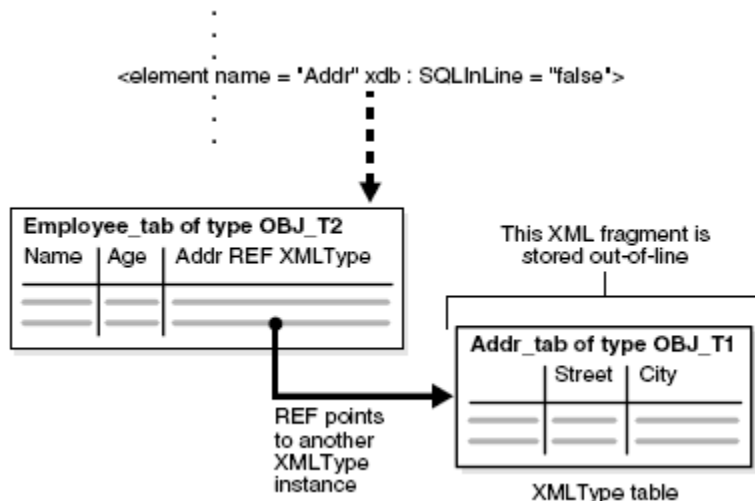


Figure 1 Example of out-of-line storage [6]

It shows the mapping complexType to SQL for Out-of-Line Storage, the attribute Addr is a REF XMLType and points to another XMLType that is a stored out-of-line XML fragment.

The advantage of this model is that it lets you query the out-of-line table (addr\_tab) directly, to look up the address information. Also if it has a list items to be stored out of line, instead of a single REF column, the parent elements will contain a varray of REF values that point to the items. If it has a large number of items is a good idea use this technique using an intermediate table (The intermediate table can be created by setting `xdb:storeVarrayAsTable="true"` in the XMLSchema definition).

### 5.4.8 Fully Qualified XML Schema URLs

Oracle supports fully qualified XML Schema URLs, this means that the name of the database user owning the XML schema is also specified as part of the schema URL, except that such XML schema URLs belong to the Oracle XML DB namespace:

<http://xmlns.oracle.com/xdb/schemas/<database-user>/<schemaURL-minus-protocol>>

### 5.4.9 Mapping XML Fragments to Large Objects (LOBs)

In Oracle it is possible to specify the SQLType for a complex element as CLOB or BLOB value. This is useful when parts of the XML document are seldom queried but are mostly retrieved and stored as single pieces.

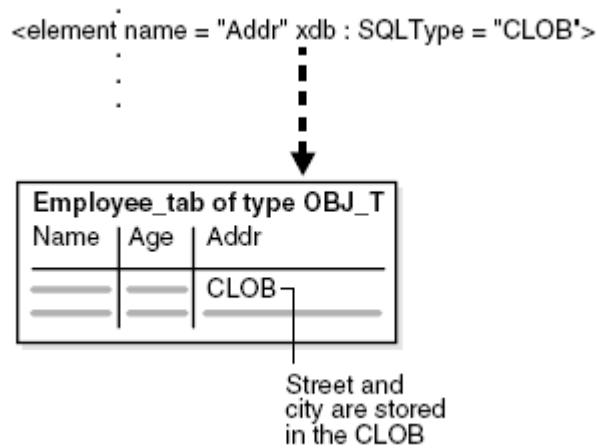


Figure 2 Example of mapping XML fragments to LOBs [6]

### 5.4.10 ComplexType Extensions and Restrictions in Oracle XML DB

In XML Schema, complexType values are declared based on:

- simpleContent: is declared as an extension of a simpleType.
- complexContent: is declared as Base type, or complexType extension on complex restriction.

With ComplexType, Oracle allows handling inheritance, for example to extend other complextypes or to restrict other complexTypes.

### 5.4.11 Circular Schema Dependencies

Oracle XML DB supports XML schemas that define the circular schema structure (with recursive references). This detects the cycles, breaks them and stores the recursive elements as rows in separate XMLType table. For this, is important that the genTables parameter is set to TRUE when registering an XML schema that defines this kind of structure.

Examples of circular schema structure are:

Cross Referencing Between Different complexTypes in the Same XML Schema

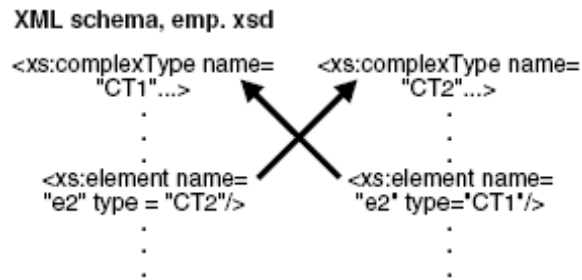


Figure 3 Crossed schema dependencies [6]

complexType self referencing within an XML Schema

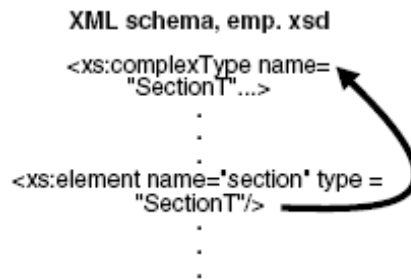


Figure 4 Self referencing [6]

Another kind of structure is the structure where the references are between different schemas. In this case sometimes Oracle XML DB can't break the cycles and give a fail.



Figure 5 Circular schema dependencies [6]



## 5.5 Oracle XPath Extension

### 5.5.1 Functions to examine Type Information

The functions `instanceof` and `instanceof-only` are in namespace `http://xmlns.oracle.com/xdb` and they have the prefix “ora”.

- **ora: instanceof-only:** it is used to restrict the result set to nodes of a certain datatype.

Syntax: `ora:instanceof-only(nodeset-expr, typename [, schema-url])`

- **nodeset-expr:** XPath expression.
- **Typename:** string.
- **Schema-url:** indicates the schema location URL for the dataType to be matched.

On XML schema-based data, the function evaluates XPath expression `nodeset-expr` and determines the XML Schema datatype for each of the resultant nodes. It returns true if the datatype of any of the nodes exactly matches datatype `typename`, and return false in otherwise.

- **ora: instanceof:** it is used to restrict the result set to nodes of a certain datatype or its subtypes.

Syntax:

```
ora:instanceof(nodeset-expr, typename [, schema-url])
```

This function returns true if the datatype of any of the matching nodes exactly matches a subtype of datatype `typename`.

### 5.5.2 Using XML Schema with Oracle XML DB

When you use a bind variable, Oracle Database rewrites the queries for the cases where the bind variable is used in place of a string literal value. You can also use the `CURSOR_SHARING` set to force Oracle Database to always use bind variables for all string expressions.

## 5.6 XPath Rewrite

The XPath expressions that are rewritten by Oracle XML DB are a subset of those that are supported by Oracle XML DB. Whenever you can do so without losing functionality, use XPath expressions that can be rewritten.

For example:

We have a query that obtains the Book element and compare it with the literal '1980':

```
SELECT OBJECT_VALUE FROM mybooks b
WHERE extractValue(OBJECT_VALUE, '/Library/Book') = '1980';
```

If the table *mybooks* was created with XML schema-based structured storage, extract value will be rewritten to the relational column that stores the book information. The final query will be:

```
SELECT VALUE(b) FROM mybooks b WHERE b.xmldata.Book = '1980';
```

XPath rewriter can change other kinds of queries but only an XPath expression can be rewritten if all of the following are true:

- The XML function or method is rewritable. SQL functions `extract`, `existsNode`, `extractValue`, `updateXML`, `insertChildXML`, `deleteXML`, and `XMLSequence` are rewritten. Except method `existsNode()`, none of the corresponding `MLType` methods are rewritten.
- The XPath expression uses only the descendent axis. Expressions involving axes (such as parent and sibling) other than descendent are not rewritten. Expressions that select attributes, elements, or text nodes can be rewritten. XPath predicates are rewritten to SQL predicates.
- The XML Schema constructs for the XPath expression are rewritable. XML Schema constructs such as complex types, enumerated values, lists, inherited (derived) types, and substitution groups are rewritten. Constructs such as recursive type definitions are not rewritten.
- The storage structure chosen during XML-schema registration is rewritable. Storage using the object-relational mechanism is rewritten. Storage of complex types using CLOBs is not rewritten.

With those guidelines, this is a list with XPath constructs that can be rewritten:

- Simple XPath traversals

- Predicates and index accesses
- Oracle-provided extension functions on scalar values
- SQL Bind variables
- Descendant axis (XML schema-based data only): Rewrites over the descendant axis (//) are supported if:
  - There is at least one XPath child or attribute access following the //
  - Only one descendant of the children can potentially match the XPath child or attribute name following the //. If the XML schema indicates that multiple descendants of the children potentially match, and there is no unique path that the // can be expanded to, then no rewrite is done.
  - None of the descendants have an element of type xsi:anyType
  - There is no substitution group that has the same element name at any descendant.
- Wildcards (XML schema-based only). Rewrites over wildcard axis (/\*) are supported if:
  - There is at least one XPath child or attribute access following the /\*
  - Only one of the grandchildren can potentially match the XPath child or attribute name following the /\*. If the XML schema indicates that multiple grandchildren potentially match, and there is no unique path that the /\* can be expanded to, then no rewrite is done.
  - None of the children or grandchildren of the node before the /\* have an element of type xsi:anyType
  - There is no substitution group that has the same element name for any child of the node before the /\*.

And the not supported list:

- XPath functions other than those listed earlier. The listed functions are rewritten only if the input is an element with scalar content.
- XPath variable references.
- All axes other than the child and attribute axes.
- Recursive type definitions with descendent axes.
- UNION operations.

The following Scheme constructs are supported:

- Collections of scalar values where the scalar values are used in predicates.
- Simple type extensions containing attributes.
- Enumerated simple types.
- Boolean simple type.

- Inheritance of complex types.
- Substitution groups.

And no supported Scheme constructs:

- XPath expressions accessing children of elements containing open content, namely any content. When nodes contain any content, then the expression cannot be rewritten, except when the any targets a namespace other than the namespace specified in the XPath. The any attributes are handled in a similar way.
- Datatype operations that cannot be coerced, such as addition of a Boolean value and a number.

The following storage constructs are supported:

- Simple numeric types mapped to SQL RAW datatype.
- Various date and time types mapped to the SQL TIMESTAMP\_WITH\_TZ datatype.
- Collections stored inline, out-of-line, as OCTs, and as nested tables.
- XML functions over schema-based and non-schema-based XMLType views and SQL/XML views also get rewritten.

And not supported:

- CLOB storage

#### 5.6.1.1 Xpath rewrite can change comparison semantics

In XPath 1.0, the operators `<`, `>`, `<=` and `>`, use only numeric comparison and if we use it and one of the operands fails the comparison returns false because before the comparison the operands are converted to numeric values. That can happen with for instance if we manage dates. With XPath rewrite, however, this predicate is translated to a SQL date comparison.

#### 5.6.1.2 Example about how XPath expressions are rewritten

##### *Creating XML Schema-Based Purchase-Order Data*

```
DECLARE
doc VARCHAR2(2000) :=
`<schema
targetNamespace="http://xmlns.oracle.com/xdb/documentation/purchaseOrder.xsd"
xmlns:po="http://xmlns.oracle.com/xdb/documentation/purchaseOrder.xsd"
xmlns="http://www.w3.org/2001/XMLSchema"
```

```

elementFormDefault="qualified">
<complexType name="PurchaseOrderType">
<sequence>
<element name="PONum" type="decimal"/>
<element name="Company">
<simpleType>
<restriction base="string">
<maxLength value="100"/>
</restriction>
</simpleType>
</element>
<element name="Item" maxOccurs="1000">
<complexType>
<sequence>
<element name="Part">
<simpleType>
<restriction base="string">
<maxLength value="20"/>
</restriction>
</simpleType>
</element>
<element name="Price" type="float"/>
</sequence>
</complexType>
</element>
</sequence>
</complexType>
<element name="PurchaseOrder" type="po:PurchaseOrderType"/>
</schema>' ;
BEGIN
DBMS_XMLSCHEMA.registerSchema(
'http://xmlns.oracle.com/xdb/documentation/purchaseOrder.xsd', doc);
END;
/

```

The registration creates the internal types. We can now create a table to store the XML values and also create a nested table to store the items.

```

CREATE TABLE mypurchaseorders OF XMLType
XMLSchema
'http://xmlns.oracle.com/xdb/documentation/purchaseOrder.xsd'
ELEMENT "PurchaseOrder"
VARRAY xmldata."Item" STORE AS TABLE item_nested;

```

Now, we insert a purchase order into this table.

```

INSERT INTO mypurchaseorders
VALUES(XMLType( '<PurchaseOrder
  xmlns="http://xmlns.oracle.com/xdb/documentation/purchaseOrder.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation
= "http://xmlns.oracle.com/xdb/documentation/purchaseOrder.xsd
http://xmlns.oracle.com/xdb/documentation/purchaseOrder.xsd">
<PONum>1001</PONum>

```

```

<Company>Oracle Corp</Company>
<Item>
<Part>9i Doc Set</Part>
<Price>2550</Price>
</Item>
<Item>
<Part>8i Doc Set</Part>
<Price>350</Price>
</Item>
</PurchaseOrder>' ));

```

#### Example 6 Use of REGISTERSchema [6]

Because the XML schema did not specify anything about maintaining the ordering, the default is to maintain the ordering and DOM fidelity. Hence the types have the SYS\_XDBPD\$ (PD) attribute to store the extra information needed to maintain the ordering of nodes and to capture extra items such as comments, processing instructions and soon.

The SYS\_XDBPD\$ attribute also maintains the existential information for the elements (that is, whether or not the element was present in the input document). This is needed for simpleType elements, because they map to simple relational columns. In this case, both empty and missing simpleType elements map to NULL values in the column, and the SYS\_XDBPD\$ attribute can be used to distinguish the two cases. The XPath rewrite mechanism takes into account the presence or absence of the SYS\_XDBPD\$ attribute, and rewrites queries appropriately.

This table has a hidden XMLDATA column of type purchaseorder\_t that stores the actual data.

### 5.6.1.3 Mapping types and Path expressions

#### 5.6.1.3.1 Mapping for a Single XPath:

XPath Expression	Maps to
/PurchaseOrder	column XMLDATA
/PurchaseOrder/@PurchaseDate	column XMLDATA."PurchaseDate"
/PurchaseOrder/PONum	column XMLDATA."PONum"
/PurchaseOrder/Item/Part	attribute "Part" in the collection XMLDATA."Item"

#### 5.6.1.3.2 Mapping for a simpleType Elements

An XPath expression can contain a text() node test, which targets the text node (content) of an element. When rewriting, this maps directly to the underlying relational columns. For example, the

XPath expression `"/PurchaseOrder/PONum/text()"` maps directly to the SQL column `XMLDATA."PONum"`.

A NULL in the PONum column implies that the text value is not available: either the text() node test is not present in the input document or the element itself is missing.

If the column is NULL, there is no need to check for the existence of the element in the `SYS_XBDPDP$` attribute.

The XPath `"/PurchaseOrder/PONum"` also maps to the SQL attribute `XMLDATA."PONum"`. However, in this case, XPath rewrite must check for the existence of the element itself, using attribute `SYS_XBDPDP$` in column `XMLDATA`.

#### 5.6.1.3.3 Mapping predicates

Predicates are mapped to SQL predicate expressions. Since the predicates are rewritten to SQL, the comparison rules of SQL are used instead of the XPath 1.0 semantics. The following example shows this (using the previous schema example) :

The predicate in the XPath expression:

`/PurchaseOrder[PONum=1001 and Company = "Oracle Corp"]`

maps to the SQL predicate:

`(XMLDATA."PONum" = 20 AND XMLDATA."Company" = "Oracle Corp")`

XPath expressions can involve relational collection expressions. In XPath 1.0, these are treated as existential checks: if at least one member of the collection satisfies the expression, then the expression is true.

#### 5.6.1.4 Document ordering with collection traversals

Most of the rewrite preserves the original document ordering. However, because SQL does not guarantee ordering on the results of sub queries when selecting elements from a collection using SQL function `extract`, the resultant nodes may not be in document order. To fix this problem it can use collection traversals, for instance:

```
SELECT extract(OBJECT_VALUE,
  '/PurchaseOrder/Item[Price>2100]/Part' )
FROM mypurchaseorders p;
```

This query is rewritten to use a sub query:

```
SELECT (SELECT XMLAgg(XMLForest(x."Part" AS "Part"))
```

```
FROM table(XMLDATA."Item") x WHERE x."Price" > 2100)
FROM mypurchaseorders p;
```

#### **Example 7 Rewrite of query with XPath**

In most cases, the result of the aggregation is in the same order as the collection elements, but this is not guaranteed. So, the results may not be in document order.

#### ***Collection position***

An XPath expression can also access an element at a position of a collection. If the collection is stored as a varray, this operation retrieve the nodes in the same order as in the original document but if the collection as a nested table, it is impossible to know the order.

#### ***Xpath expressions that can not be satisfied***

An XPath expression can contain references to nodes that cannot be present in the input document. Such parts of the expression map to SQL NULL values during rewrite.

#### ***Namespace handling***

Namespaces are handled in the same way as function-based evaluation. For schema-based documents, if the function (such as existsNode or extract) does not specify any namespace parameter, then the target namespace of the schema is used as the default namespace for the XPath expression.

#### ***Data Format conversions***

Date datatypes such as DATE, gMONTH, and gDATE have different format in XML Schema and SQL. If an expression has a string value for columns of such datatypes, then the rewrite automatically provides the XML format string to convert the string value correctly. Thus, the string value specified for a DATE column must match the XML date format, not the SQL DATE format.

## **5.6.2 Checking XPath rewrite**

There are some ways to check if your XPath expressions are rewritten:

- **Using Explain Plan whit XPath rewrite:** With the explained plan, if the plan does not pick applicable indexes and shows the presence of the SQL function (such as existsNode or extract), then you know that the rewrite has not occurred. You can then use the events described later to understand why the rewrite did not happen.
- **Using events with XPath Rewrite:** Events can be set in the initialization file or can be set for each session using the ALTER SESSION statement. The XML events can be used to turn off functional evaluation, turn off the XPath rewrite mechanism and to print diagnostic traces.



## 5.6.3 Examples of Rewrite of SQL Functions

We use the following XPath expression:

```
/PurchaseOrder[PONum =2100]/@PurchaseDate
```

### 5.6.3.1 EXISTSNODE:

- Mapping for EXISTSNODE with Document Ordering Preserved (SYS\_XDBPD\$ exists and maintainDOM="true" is present in the schema document):

Maps to:

```
CASE WHEN XMLDATA."PONum"=2100
AND node_exists1(XMLDATA.SYS_XDBPD$, 'PurchaseDate')
THEN 1 ELSE 0 END
```

- Mapping for EXISTSNODE without Document Ordering Preserved (SYS\_XDBPD\$ not exists and maintainDOM="false" is present in the schema document):

Map to:

```
CASE WHEN XMLDATA."PONum" = 2100
AND XMLDATA."PurchaseDate" NOT NULL
THEN 1 ELSE 0 END
```

### 5.6.3.2 EXTRACTVALUE

Maps to:

```
(SELECT x.XMLDATA."PurchaseDate") FROM DUAL
WHERE x."PONum" = 2100)
```

### 5.6.3.3 EXTRACT

- With Document Order Maintained

Maps to:

```
SELECT      CASE      WHEN      node_exists1(XMLDATA.SYS_XDBPD$,
'PurchaseDate')
THEN XMLElement(" ", XMLDATA."PurchaseDate")
ELSE NULL END
FROM DUAL WHERE XMLDATA."PONum" = 2100
```

- Without Maintaning Document Order

Maps to:

```
SELECT XMLForest(XMLDATA."PurchaseDate" AS "PurchaseDate ")
FROM DUAL WHERE XMLDATA."PONum" = 2100
```

## 5.6.4 Bind Variables

When bind variables are used as string literals in XPath, the expression can be rewritten to use the bind variables. The bind variable must be used in place of the string literal using the concatenation operator (||), and it must be surrounded by single-quotes (') or double-quotes (") inside the XPath string.

### 5.6.4.1 Setting CURSOR\_SHARING to FORCE

With XPath rewrite, Oracle Database changes the input XPath expression to use the underlying columns. This means that for a given XPath there is a particular set of columns or tables that is referenced underneath. This is a compile-time operation, because the shared cursor must know exactly which tables and columns it references. This cannot change with each row or instantiation of the cursor.

Hence if the XPath expression is itself a bind variable, Oracle Database cannot do any rewrites, because each instantiation of the cursor can have totally different XPaths.

This is similar to binding the name of the column or table in a SQL query.

When CURSOR\_SHARING is set to FORCE, by default each string constant including XPath becomes a bind variable. When Oracle Database then encounters SQL functions `extractValue`, `existsNode`, and so on, it looks at the XPath bind variables to check if they are really constants. If so, it uses them and rewrites the query. Hence there is a large difference depending on where the bind variable is used.

## 5.7 XML Schema Evolution

XML schema evolution is the process of updating a registered XML schema. Oracle XML DB supports XML schema evolution by providing PL/SQL procedure `copyEvolve` as part of PL/SQL

package DBMS\_XMLSCHEMA. With that procedure you can evolve a registered XML schema in such a way that existing XML instance documents continue to be valid. But this procedure has some limitations, for instance: the indexes, triggers, constraints, and other metadata related are deleted and these must be re-created after evolution. Other limitation is that it can't be used if there is a table with an object-type column that has an XMLType attribute that is dependent on any of the schemas to be evolved.

To use DBMS\_XMLSCHEMA.COPYEVOLVE, it necessary follows the next steps:

1. Identify the XML schemas that are dependent on the XML schema that is to be evolved. You can acquire the URLs of the dependent XML schemas using the following query, where `schema_to_be_evolved` is the schema to be evolved, and `owner_of_schema_to_be_evolved` is its owner (database user).

In many cases, no changes may be necessary in the dependent XML schemas. But if the dependent XML schemas need to be changed, you must also prepare new versions of those XML schemas.

2. If the existing instance documents do not conform to the new XML schema, you must provide an XSL style sheet that, when applied to an instance document, will transform it to conform to the new schema. This needs to be done for each XML schema identified in the first step. The transformation must handle documents that conform to all top-level elements in the new XML schema.
3. Call procedure DBMS\_XMLSCHEMA.copyEvolve, specifying the XML schema URLs, new schemas, and transformations.

### 5.7.1 Style Sheet to update existing instance documents

After it modifies a registered XML schema, it must update any existing XML instance documents that used the old version of the schema. It does it by applying an XSLT style sheet to each of the instance document. The style sheet represents the difference between the old and new schemas.

Syntax:

```
procedure copyEvolve(schemaURLs IN XDB$STRING_LIST_T,  
    newSchemas IN XMLSequenceType,  
    transforms IN XMLSequenceType := NULL,  
    preserveOldDocs IN BOOLEAN := FALSE,
```

```
mapTabName IN VARCHAR2 := NULL,
generateTables IN BOOLEAN := TRUE,
force IN BOOLEAN := FALSE,
```

- **schemaURLs:** Varray of URLs of XML schemas to be evolved (varray of VARCHAR2(4000). This should include the dependent schemas as well. Unless the force parameter is TRUE, the URLs should be in the dependency order, that is, if URL A comes before URL B in the varray, then schema A should not be dependent on schema B but schema B may be dependent on schema A.
- **newSchemas:** Varray of new XML schema documents (XMLType instances). Specify this in exactly the same order as the corresponding URLs. If no change is necessary in an XML schema, provide the unchanged schema.
- **Transforms:** Varray of XSL documents (XMLType instances) that will be applied to XML schema based documents to make them conform to the new schemas. Specify these in exactly the same order as the corresponding URLs. If no transformations are required, this parameter need not be specified.
- **preserveOldDocs:** If this is TRUE the temporary tables holding old data are not dropped at the end of schema evolution. See also "Using Procedure DBMS\_XMLSCHEMA.COPYEVOLVE".
- **mapTabName:** Specifies the name of table that maps old XMLType table or column names to names of corresponding temporary tables.
- **generateTables:** By default this parameter is TRUE; if this is FALSE, XMLType tables or columns will not be generated after registering new schemas. If this is FALSE, preserveOldDocs must be TRUE and mapTabName must not be NULL.
- **Force:** If this is TRUE errors during the registration of new schemas are ignored. If there are circular dependencies among the schemas, set this flag to TRUE to ensure that each schema is stored even though there may be errors in registration.
- **schemaOwners** Varray of names of schema owners. Specify these in exactly the same order as the corresponding URLs.

An example of using DBMS\_XMLSCHEMA.COPYEVOLVE to update an XML Schema:

This example evolves XML schema purchaseOrder.xsd to revisedPurchaseOrder.xsd using XSL style sheet evolvePurchaseOrder.xsl.

```
BEGIN
  DBMS_XMLSCHEMA.copyEvolve(
```

```

        xdb$string_list_t('http://localhost:8080/source/schemas/
        poSource/xsd/purchaseOrder.xsd'),
        XMLSequenceType(XDBURITYPE('/source/schemas/poSource/revi
        sedPurchaseOrder.xsd').getXML()),
        XMLSequenceType(XDBURITYPE('/source/schemas/poSource/evol
        vePurchaseOrder.xsl').getXML()));
END;
SELECT extract(object_value, '/PurchaseOrder/LineItems/LineItem[1]')
LINE_ITEM
FROM purchaseorder
WHERE existsNode(object_value, '/PurchaseOrder[@Reference="SBELL-
2003030912333601PDT"]') = 1
/

```

#### Example 8 Use of CopyEvolve

## 5.8 Full-Text Search Over XML

To use XML documents it is very important when you want to do full-text search since you can use the XML structure of the document to restrict the search. Also if these XML documents are of type XMLType, then you can project the results of the query using the XML structure of the document.

This kind of search differs from structured search or substring search in the following ways:

- The substring search looks for whole words rather than substrings. For example if we search the string “neighbour” it might return “neighbourhood”, however a full-text search will not, because search words.
- A full-text search supports some language-based and word-based searches, the substring searches can't. For example you can use it to find synonymous (language-based search) or to find all the comments that contain the word “lawn” within 5 words of “wild”.
- A full-text search generally involves some notion of relevance.

To do a search that includes full-text search and XML structure, you can:

Include the structure inside the full-text predicate, using *contains* SQL function:

```

... WHERE contains(doc, 'electric INPATH
(/purchaseOrder/items/item/comment)') > 0

```

Or to include the full-text predicate inside the structure, using the ora:contains XPath Function.

```

... '/purchaseOrder/items/item/comment[ora:contains(text(),
"electric")>0]' ...

```

## 5.8.1 The function CONTAINS

SQL function `contains` returns a positive number for rows where `[schema.]column` matches `text_query`, and zero otherwise. It is a user-defined function, a standard extension method in SQL. It requires an index of type `CONTEXT`. If there is no `CONTEXT` index on the column being searched, then `contains` throws an error.

**Syntax:**

```
contains([schema.]column, text_query VARCHAR2 [,label NUMBER])  
RETURN NUMBER
```

The argument `text_query` support combinations of `AND`, `OR`, and `NOT`.

## 5.8.2 The score sql function

SQL function `contains` has a related function, `score`, which can be used anywhere in the query. It is a measure of relevance, and it is especially useful when doing full-text searches across large document sets. `score` is typically returned as part of the query result, used in the `ORDER BY` clause, or both.

**Syntax:**

```
score(label NUMBER) RETURN NUMBER
```

## 5.8.3 Structure operators

Also Oracle has 3 structure operators to restrict `contains` queries using XML structure:

### 5.8.3.1 Within

The `WITHIN` operator restricts a query to some section within an XML document. Section names are case-sensitive; also you can search within attributes, with others operands.

An example of use of `within`:

```
SELECT id FROM purchase_orders  
WHERE contains(doc, 'lawn AND electric WITHIN comment') > 0;
```

That query return the comments that has inside the words “lawn” and “electric”

### 5.8.3.2 INPATH

Operator INPATH takes a text\_query on the left and a Text Path, enclosed in parentheses, on the right

For example:

```
SELECT id FROM purchase_orders
WHERE contains(doc, 'electric INPATH
(/purchaseOrder/items/item/comment)') > 0;
```

The query finds purchaseOrders that contain the word “electric” in the path.

### 5.8.3.3 HASPATH

This operator finds documents that contain a particular section in a particular path, possibly with an “=” predicate.

```
SELECT id FROM purchase_orders
WHERE contains(doc, 'HASPATH
(/purchaseOrder//item/EUPrice="120.5")') > 0;
```

This query finds purchaseOrders that have a items with a EUPrice that text-equal to “120.5”

## 5.8.4 Context index

Context index is a full-text index. To create a default full-text index, use the regular SQL CREATE INDEX command, and add the clause INDEXTYPE IS CTXSYS.CONTEXT. This index is necessary to use the function contains. If it isn’t used, Oracle returns error.

## **PART II**

### **Sample application**



- The information presented in this part is based in [2], [8], [14], and [15].

## 6 Introduction

This part contains the documentation for a simple application that shows some features from Oracle 10g for management, store and retrieve XML documents. In that documentation appear the analysis, design UML documentation, a user manual and some important information about the application.

The designed application is a simple application to show the different features of Oracle XML DB. The application has different parts, one of them shows the possibilities of the XMLSCHEMA in general, it can create, delete and register schemas also creates tables with this schemas or without them. Moreover the application can add new elements with XML structure to the selected tables. At the last, the application contains a searcher to show the search features in XML Documents, in this part it is used a specific XML database, this database is related with a XML Schema that describe the structure of recipes for meals or drinks, also it is possible to add recipes through the form . Moreover the user can choose different options of search to show the use of XPath features and search feature in text from oracle.

In resume, the application has two parts, one of them is a schema independent and other is schema dependent (dependent of the XML schema of recipes).

## 7 Analysis and design

### 7.1 Global vision

This point starts showing the most important requirements for the application. After it will appear the use case diagram with the use case specification. Also it will appear the structure diagram and behaviour diagrams.

### 7.2 Requirements

#### 7.2.1 Functional requirements

##### 7.2.1.1 Rq.1

##### *Description*

The system must show the support of Oracle Database for XML structures.

#### **7.2.1.2 Rq.2**

##### ***Description***

The system must manage XML documents.

#### **7.2.1.3 Rq.3**

##### ***Description***

The application must allow connecting with a DB. (FIT DB server)

#### **7.2.1.4 Rq.4**

##### ***Description***

The application must create XML documents.

#### **7.2.1.5 Rq. 5**

##### ***Description***

The system must allow searching in XML DataBase.

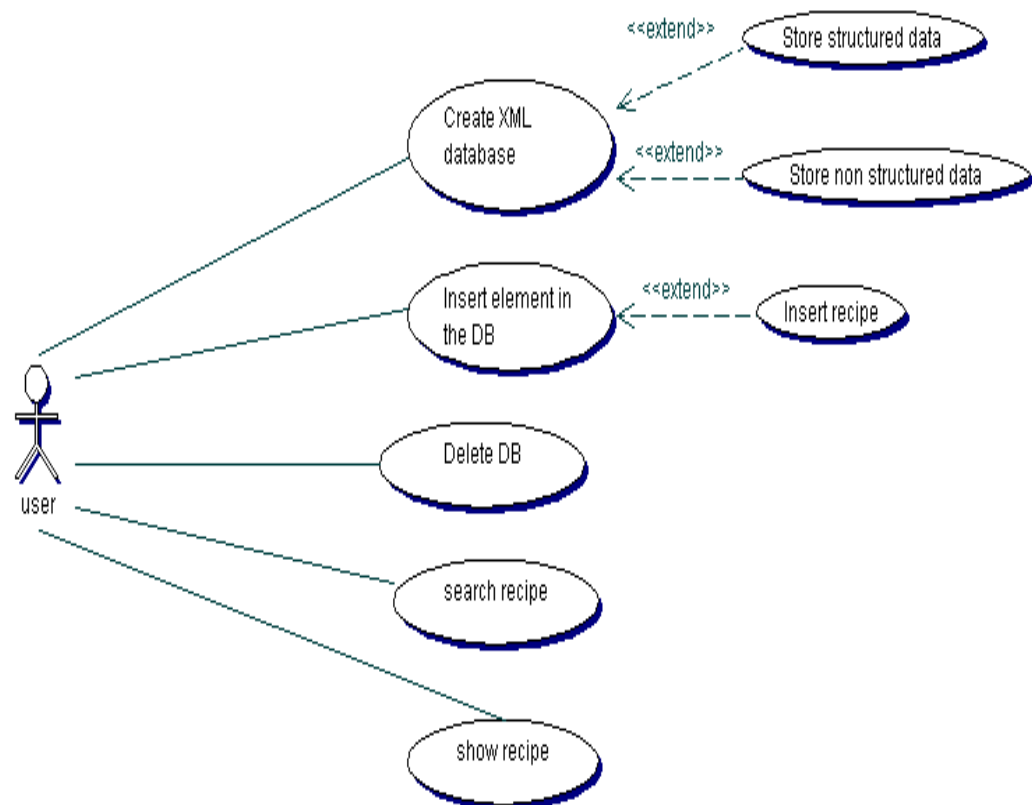
#### **7.2.1.6 Rq. 6**

##### ***Description***

The system must allow inserting elements in a XML DataBase.

## 7.3 Use cases

### 7.3.1 Use case diagram



**Figure 6 Use cases diagram**

The **Figure 6** shows the use case diagram, it is possible see that the application

### 7.3.2 Use cases specification

#### 7.3.2.1 Create XML Database

##### *Preconditions*

- The user has to be introduced in the system.

##### *Basic course of events*

1. The use case starts when the user wants to create XML DB.
2. The user selects the option "Create XML DB".
3. The system shows a form for the options for the DB.

4. The user selects the action to do. *Point of extension*.
5. The system stores the changes.
6. The case use finish.

***Postconditions***

- A database or table is created.
- The database or table created appears in the correspondent list.

**7.3.2.2 Create Structured DataBase**

***Precondition***

***Basic course of events***

1. The case use starts when the user wants to create a Database using Document XSD (schema).
2. The system show a dialog where the user can choose a xsd file (XML Schema Definition)
3. The user selects the file.
4. The system checks it.
5. The system validates and registers the Schema.
6. The system creates the correspondent XML tables with aN XMLType column for the data.
7. The use case finishes.

***Postcondition***

- A new DB is created.

**7.3.2.3 Create a No Structured XML DataBase**

***Precondition***

***Basic course of events***

1. The use case starts when the user wants to store data in a non structured XML Database.
2. The system shows a dialog where the user can choose the name of the table to create.
3. The system creates a XML table with an XMLType column for the element.
4. The system shows a dialog to choose the xml file with the data.
5. The system stores the file like a GLOB type.
6. The use case finishes.

***Postconditions.***

- The DB contains the new data.

#### **7.3.2.4 Insert element**

##### ***Preconditions***

- It might exist a database.

##### ***Basic course of events***

1. The case use starts when the user wants to insert an element in a XML DB.
2. The system shows the available XMLDB. *Point of extension.*
3. The user selects a XMLDB.
4. The system shows the tables of the XMLDB.
5. The user selects the table where he wants to add the element (document XML).
6. The system checks the element.
7. The use case finishes.

##### ***Alternative course***

##### ***Postconditions***

- The element is added to the database.

#### **7.3.2.5 Insert recipe**

##### ***Preconditions***

- It might exist the database XMLRECIPE.

##### ***Basic course of events***

1. The case use starts when the user wants to insert a recipe in the database XMLRECIPE.
2. The system shows a form to write the necessary information. Title, ingredients, preparation and nutritional information.
3. The user introduces the information.
4. The user selected the option add recipe.
5. The system checks the introduced data.
6. The system adds the element to the database.
7. The use case finishes.

##### ***Postconditions***

- The database is modified with the selected changes.
- The database contains an element with the introduced information.

### **7.3.2.6 Search Recipe**

#### ***Preconditions***

- It might exist a database (XMLRECIPE).
- It might exist at least one element in the database.

#### ***Basic course of events***

1. The case use starts when the user wants to access an element in a XML DB.
2. The system shows a form where the user can introduce the word or words to search.
3. The user selects the different options (search by title, by ingredients, by preparation, by nutritional contains).
4. The user selects the option search.
5. The system shows the titles of the recipes of the query.
6. The use case finishes.

#### ***Alternative course***

#### ***Postconditions***

- The system shows the elements required.

### **7.3.2.7 Show recipe**

#### ***Preconditions***

- It might exist a XML database (XMLRECIPE).
- It might exist at least one element in the database.
- The user has to do a search.

#### ***Basic course of events***

1. The case use stars when the user wants to see a recipe.
2. The user selected a recipe.
3. The system transforms a XML document with the elements to readable format.
4. The system shows the recipe.
5. El use case finishes.

#### ***Postconditions***

- The system shows the element required.

### 7.3.3 The structure of the database

Oracle to store the XMLDatabase needs to store in a relational database, in this case the database of recipes will be the next structure:

- The database of recipes only contains one table. This table is related with a XML schema with the structure of a recipe for meals. The followed code shows the creation.

```
CREATE TABLE xmlrecipes OF XMLTYPE
XMLSCHEMA recipes.xsd
ELEMENT collection;
```

The type of data base is XMLTYPE, it is related to the schema “recipes.xsd” and it is associated to the global element, collection in this case.

### 7.3.4 XML Schema selected for the database

This is the schema selected for the database, the recipes not only contains information about the ingredients or preparation but nutritional information too. In the next fragment (**Example 9**) of the schema is possible to see the different elements of the recipe (The elements are between <element> and </element> marks). Also it appears information about the multiplicity, for example <element ref="r:ingredient" minOccurs="0" maxOccurs="unbounded"/>, it means that the element ingredient can be reference to 0 o more ingredients.

- Title: contains a string with the recipe title.
- Date: contains information about the date of creation of the recipe
- Ingredient: contains the information about the ingredients, it has information like:
  - Name: name of the ingredient.
  - Amount: The necessary quantity, it has a restriction, it not possible to put a negative number.
  - Unit: Type of unit of amount. For example, grams, cups, litres...It is an optional attribute (use= “optional”).

Also an ingredient can be composed, and contains another ingredients and preparation.

- Preparation: Contains the information to prepare the recipe. Can be composed by a several steps.
- Nutrition: contains nutritional information like, percentage of calories, of carbohydrates, of fat, of proteins or of alcohol. The last one it is optional, only if the recipe is for a alcoholic drink.

.  
.  
.

```

<element name="recipe">
  <complexType>
    <sequence>
      <element name="title" type="string"/>
      <element name="date" type="string"/>
      <element ref="r:ingredient" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="r:preparation"/>
      <element name="comment" type="string" minOccurs="0"/>
      <element ref="r:nutrition"/>
      <element ref="r:related" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="NMTOKEN"/>
  </complexType>
</element>

<element name="ingredient">
  <complexType>
    <sequence minOccurs="0">
      <element ref="r:ingredient" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="r:preparation"/>
    </sequence>
    <attribute name="name" use="required"/>
    <attribute name="amount" use="optional">
      <simpleType>
        <union>
          <simpleType>
            <restriction base="r:nonNegativeDecimal"/>
          </simpleType>
          <simpleType>
            <restriction base="string">
              <enumeration value="*"/>
            </restriction>
          </simpleType>
        </union>
      </simpleType>
    </attribute>
    <attribute name="unit" use="optional"/>
  </complexType>
</element>

```



```

</complexType>
</element>
<element name="preparation">
  <complexType>
    <sequence>
      <element name="step" type="string" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

<element name="nutrition">
  <complexType>
    <attribute name="calories" type="r:nonNegativeDecimal" use="required"/>
    <attribute name="protein" type="r:percentage" use="required"/>
    <attribute name="carbohydrates" type="r:percentage" use="required"/>
    <attribute name="fat" type="r:percentage" use="required"/>
    <attribute name="alcohol" type="r:percentage" use="optional"/>
  </complexType>
</element>



.
.
.


```


#### Example 9 Fragment of the XML Schema that describe a recipe for meal.

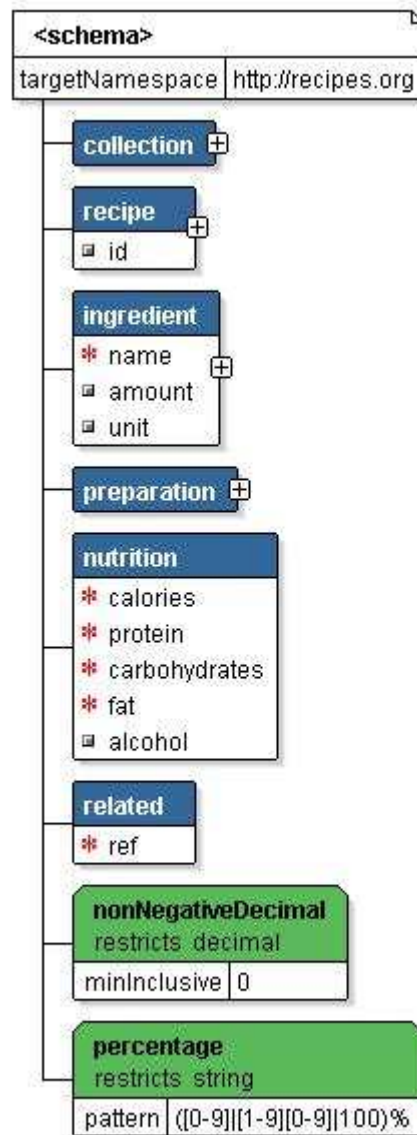
The full description is in the CD, in the file recipes.xsd but to better understand it is showed the diagram presented in Jdeveloped and describe the file recipes.xsd.

The full structure is represented in the **figure 8**, to explain this figure it is necessary know which is the mean of the different rectangles, symbols and other notations:

- The blue rectangles represent an element from the schema, if appear a  symbol, it means the element it is composed by others, also if bellow a white rectangle appears, means the element has attributes (\* if is optional and  for required attribute).

It can be appearing , this means the element reference other element.

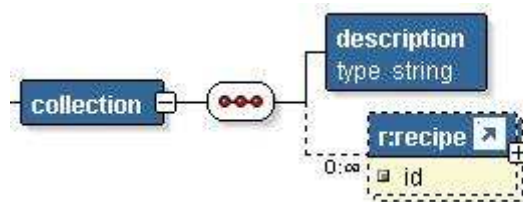
Other element that appears is . It means sequence of elements



**Figure 7 Structure of the recipe schema**

Also in the structure appear some restrictions about the data (green squares in the **Figure 8**), for example not used negative decimal or that the percentage of the nutritional information only can be a number between 0 and 100.

This structure allows create a collection of recipes (**Figure 9**) where each recipe contains ingredients, a preparation, and nutritional information (**Figure 10**). In the followed figures is possible to see the different elements of the Schema and the attributes.



**Figure 8 Description of collection element**

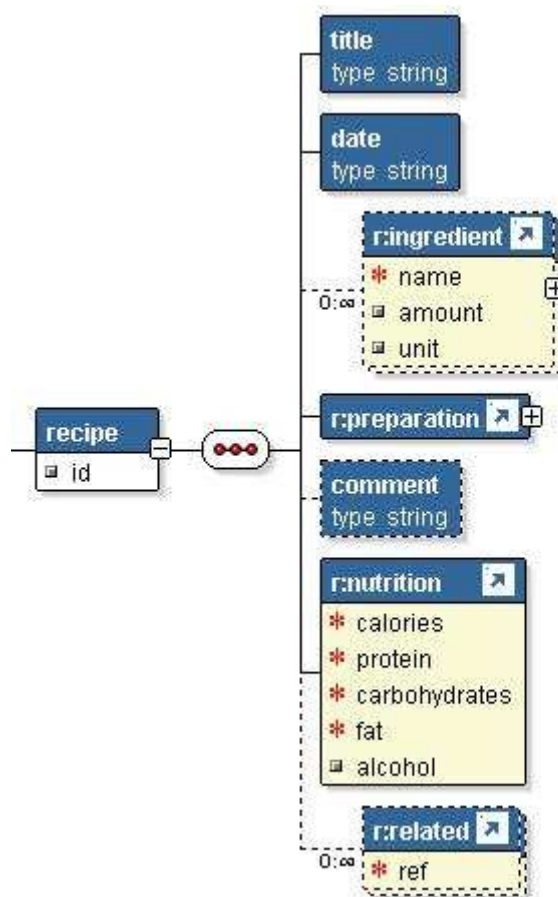


Figure 9 Description of recipe element

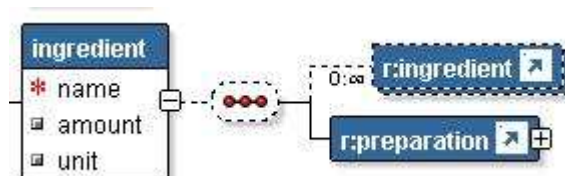


Figure 10 Description of ingredient element

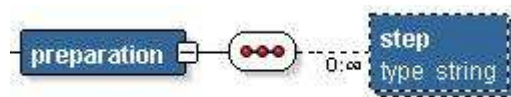


Figure 11 Description of preparation element

The ingredients can be composite by other ingredients and a preparation (**Figure 11**). This allows more flexibility because a lot of ingredients contain others and sometimes is important store this information. The preparation element is composed by several steps (**Figure 12**) to differentiate the parts of the cooking method.

The XML file with the recipes has to have the follow head:

```
<?xml version="1.0" encoding="UTF-8"?>
<?dsd href="recipes.dsd"?>
```

```

<collection xmlns="http://recipes.org"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://recipes.org recipes.xsd">
  <description>
    A description.
  </description>

```

### 7.3.5 Architecture

The architecture selected is 3 layers one: Application, Business, and DB access. The communication with the database is realized through JDBC. The client is a java application that makes a JDBC connection with thin drivers (**Figure 13**). Oracle's JDBC Thin driver uses Java sockets to connect directly to Oracle. It provides its own TCP/IP version of Oracle's Net8 (SQL\*Net) protocol. Because it is 100% Java, this driver is platform independent and can also run from a Web Browser (applets) if the platform changes in the future.

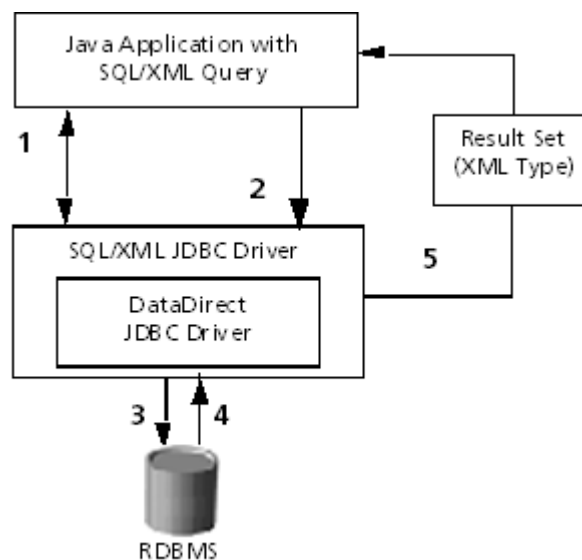
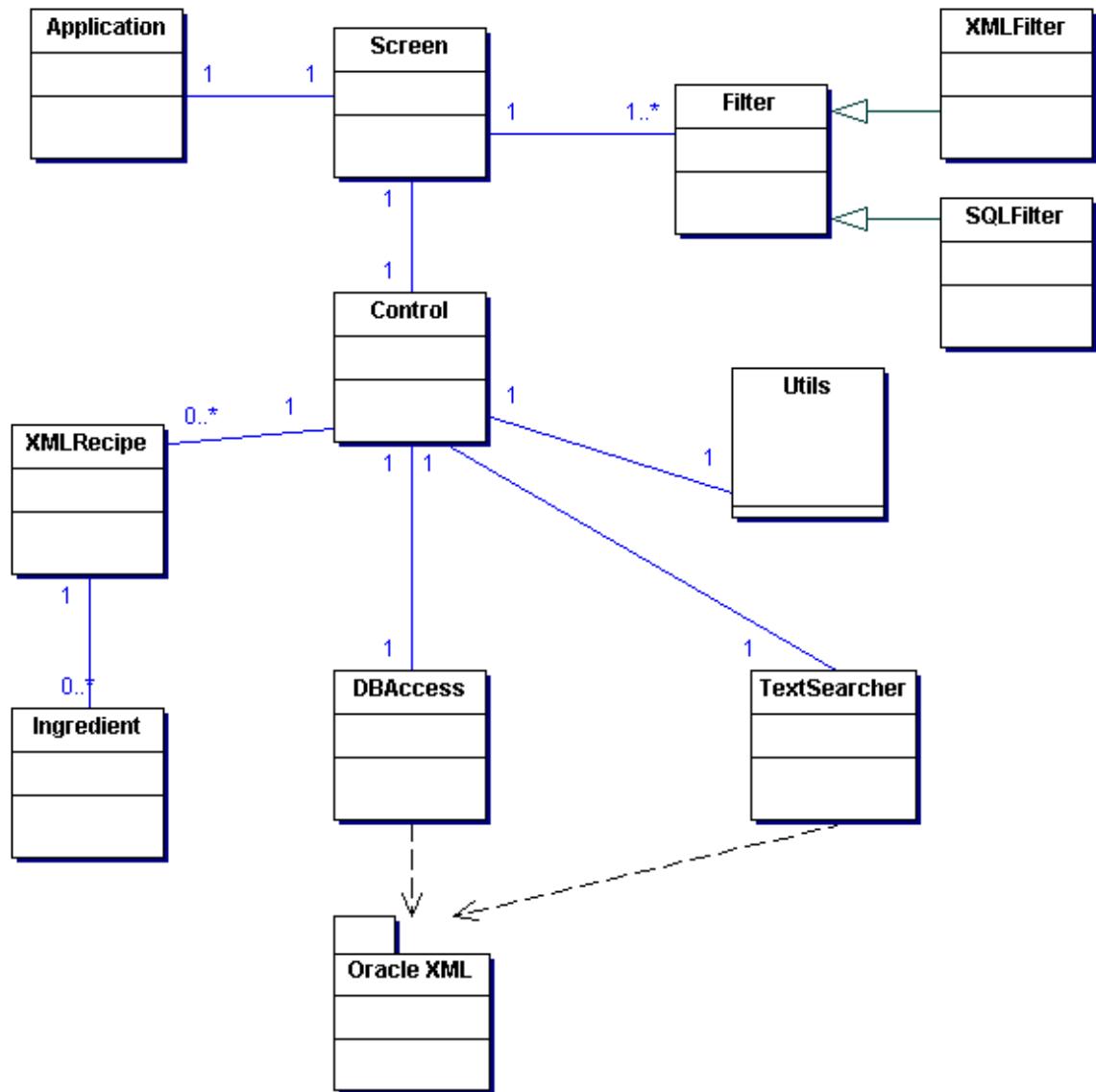


Figure 12 Communication with the RDBMS [1]

### 7.3.6 Structural diagram

The structural diagram shows the class diagram. Attributes and methods are not shown in the diagram to make easy to understand.



**Figure 13 Class diagram of the Simple Application**

- Class Application: It executes the application.
- Class Screen: It contains the elements and functionality of the user interface.
- Class Filter: It filters the files in the dialog for choose a file.
- Class XMLFilter: It filters the list of files in a choose dialog and only allows XML files.
- Class SQLFilter: It filters the list of files in a choose dialog and only allows SQL files.
- Class Control: It manages the introduced information by the user and the database information.
- Class DBAccess: It contains the functionality to access to DB.
- Class TextSearcher: It contains the functionality to search in the DB.
- Class XMLRecipe: It models a recipe element.
- Class Ingredient: It models an ingredient element.

- Class Utils: It contains several adjacent functions to manage the data.
- Package Oracle.XML: It contains the functions to manage, store and retrieve XML.

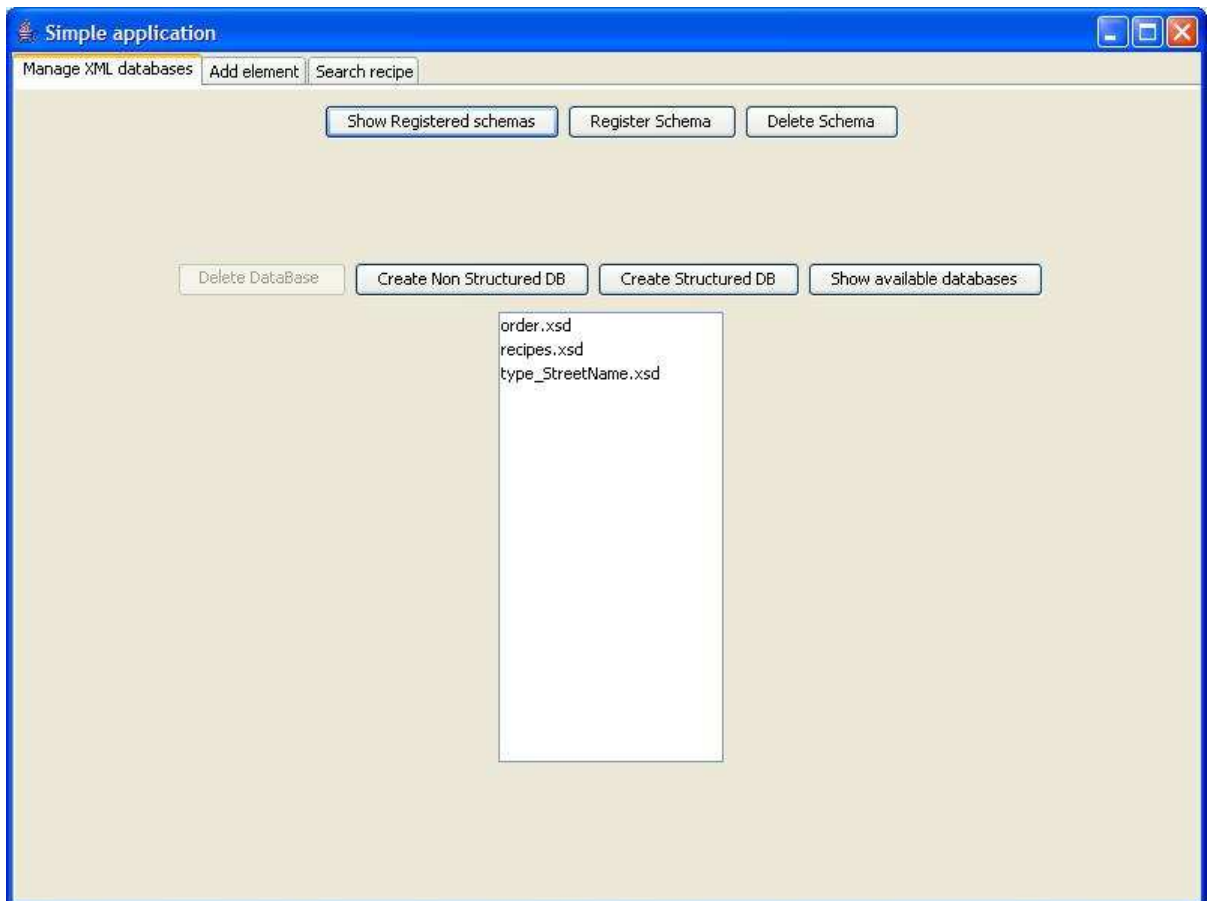
### 7.3.7 User interface

The user interface was implemented with JAVA (Swing), but the application is designed so that the IU is independent of the rest of application, then in the future for example it is possible use JSP or another languages to implement.

The user interface structure has three parts, each implement in a tab panel:

- First pane (Manage XML Databases): contains the follow options.
  - Create a structured XML Database: The user can create a structured XML Database choosing a schema.
  - Create a no structured XML Database: The user can create a no structured XML Database.
  - Register Schema: The user can register a new schema.
  - Delete Schema: The user can delete a selected schema.
  - Delete XML Database: The user can delete a selected XMLDatabase.
  - Show schemas: With this option the user can show the available schemas.
  - Show XMLDataBases: This option shows the available databases.

The **figure 15** shows the first parts of the application, in this picture, the option Show Registered schemas has been selected, and the system shows the available schemas:



**Figure 14 Manage XML databases**

- Second pane (Add Element): It contains the option to add elements to an XMLDataBase choosing a XML Document or writing a recipe.
  - Add element: The system allows select an XMLDatabase and a XML file to add her.

The user has to follow the next steps user to introduce a recipe:

1. Write the title of the recipe.
2. Write the name of the ingredient, the quantity and the unit and click in Add ingredient. The user can insert all of the ingredients that he wants.
3. Write the preparation.
4. Write the nutritional information. Quantity of calories, of fat, of proteins, of carbohydrates and of alcohol if the recipe has alcohol.
5. Click in the Add recipe button.

The **figure 16** shows the second pane, a recipe is been created using the form.

Simple application

Manage XML databases Add element Search recipe

Add a element DDD

Title: Spanish omelette Ingredients: Salt 1 nch Add ingredient

Eggs 4.0 Unit  
Onion 1.0 Unit  
Olive Oil 100.0 ml  
Salt 1.0 Pinch

Preparation:

en colour, remove from the pan and put the potato mixture either in a sieve or kitchen paper, so that as much oil as possible drains away or is absorbed.

2: Beat the eggs well with a pinch of salt, and add to the potatoes. Mix well.

3: Put two small spoonfuls of olive oil in the frying pan, so that the bottom of the pan is covered with a thin layer of oil. Once the oil is hot, add the potato and egg mixture. Tip: shake the pan gently as you move the mixture, so that none sticks to the bottom. Once the omelette seems to be cooked, use the lid of the frying pan (or a large plate) to tip the omelette out of the pan, add a little more oil and slide the omelette in again, this time putting the less cooked side first into the pan. If you need to repeat this step, so that the omelette is perfectly cooked and golden on both sides, you may do so. This omelette is delicious hot or cold.

Nutrition: :Calories :Fat :CarboHydrates :Alcohol :Proteins Add Recipe

**Figure 15 Add elements**

- Third pane (search recipe): It is the pane with the search options, the user can search recipes by:
  - Title
  - Preparation
  - Nutritional information
  - Ingredients.

If the user wants to search some recipe, he has to follow the next steps:

1. Write the word or words to search using the commands “and”, “or” if necessary.
2. Click in Search button.
3. Click the required recipe from the list that the system shows.

The **figure 17** shows the third pane, in this case, the user introduced the word “beef”, the option Search by ingredients is selected and the application show the available recipe “Beef Parmesan with Garlic Angel Hair Pasta” and the preparation.



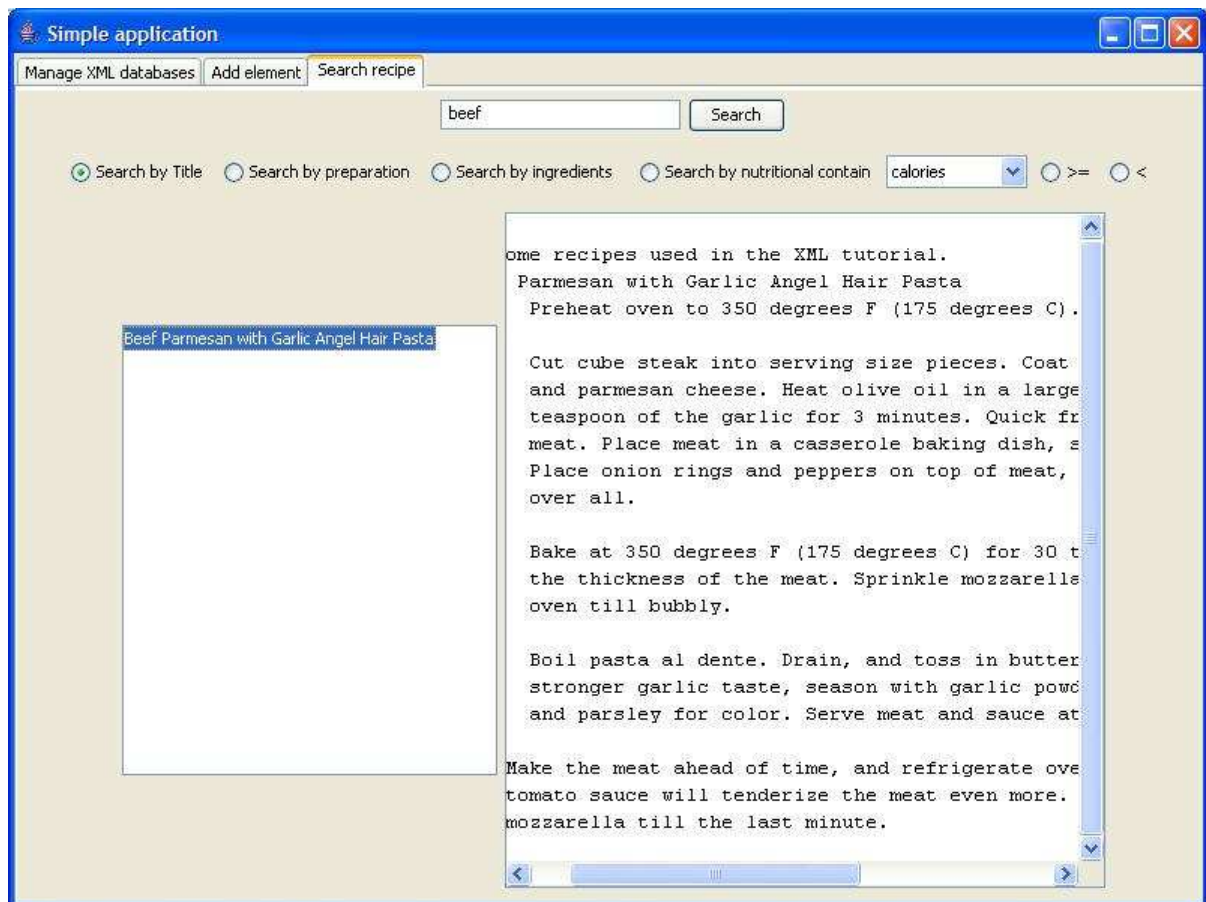


Figure 16 Search Recipes

### 7.3.8 Used technology

To implement this application was used some Schema features from the package DBMS\_SCHEMA (Section 5.4), using some functions like registerSchema, and deleteSchema with several options to show some features of XML Schema in Oracle. Also to the search information was necessary use XPath (Section 5.5) with XType functions and full text functions like contains, ora:contains or extractValue (Section 5.8) and existNode. That functions are necessary to make the correct queries. To show the recipes ,it was necessary too, XSL functions like XMLTransform that use a XSL (XML Stylesheet) to show the information with a specific style.

Not all important features that were named in the Background could be used in the application, for example the characteristics of the XPath rewrite (Section 5.5) can't be showed with this application, so you need use sqlplus of another tool to see the transformations that Oracle make in the queries with XPath.

During the implementation appeared the problem to use the function createSchema, it needed administration privileges to use the option “CREATE DEFAULT TABLE”, that create the XML tables related with the Schema automatically. So the action to create the XML tables is making the tables in a second step using CREATE TABLE with the option XMLSCHEMA to refer to schema.

## 8 Conclusions and future works

Since the XML Databases appeared, with the enabled XML Databases, it is developing in the native XML databases, several databases are nearly native but it is necessary improved the technology.

But now the question appears, are the native XML databases the future? About this, in the Arun Gaikad's article [4], he said: "An XML database system is something which you may think is unnecessary but once you start using it, you wonder how you would survive without it", maybe is true, because nowadays to store, search and manage XML is very important in the actual information technologies.

At present, the XML databases are mixed with the relational databases, this generates several problems because different packages appear to access to the different elements of the database, making difficult the application developing. One of the future works is to solve this; for example Oracle is developing XQS (Oracle XML Query Service). XQS is an OC4J service built upon XQuery to provide a simplified, declarative mechanism for creating integrated views of enterprise data. Without a service like XQS, XQuery is limited to accessing XML documents. With XQS you can also retrieve another kind of data (non-XML documents, relational databases...) through access mechanisms such as JDBC or Web services.

Working in the simple application gave me the opportunity of test the several features and functions to store and retrieve of XML data. This allows me to know how Oracle can manage this kind of databases. The XML data is easily stored, and if you have a schema, the system can create the database from the schema automatically (only with administrator privileges), so that the user only needs to develop the schema and after that, the system does the rest. This is only if you have schema, without schema it is more difficult, because the system doesn't make the database, don't validate the elements to insert, however is a good option if you only want to store elements like CLOB and not to access to specific parts of the documents. To insert elements in XML database with schema is easier, the system before to insert uses the schema to validate first the element to insert, the problem is that problems appear when the schema has some recursive elements, the system don't solve automatically this, and the user have to solve manually, and sometimes aren't a good solution.

The search in the XML documents is very important and the system has a lot of options and functions but sometimes you have to use CONTEX INDEX if you need velocity in the search (function contains) and to use SQL package for text search, it is the best option if the user doesn't add a lot of times because in each added the index have to be rebuilt. The other used option was the ora:contains function. This option is slower than the other option but it isn't necessary create an index. With the database that I created for the application the differences don't exist because it is very

small, with small elements and small number of elements. But for example in digital libraries where the number of elements and information is huge, the election of the type of search is critical.

In my opinion it is necessary to continuous improving this kind of databases, because the problem about to store information in different format is big and transforming this information to XML is fast and allows having only one format. When that problem (to have different standard data format) disappears and appear only one standard format, the XML databases can be a good point to store the information, since XML databases are suitable to search in this information.

In my point of view, it is necessary increase the functionality of these databases because there are a lot of options for text but not for others kind of data. For instance, you have functions to search in text but if you want to relate the text with a video (for example subtitles) there isn't functionality.

A good future work will be having more functionality to integrate the different types of data.

## 9 Glossary

**JDBC:** Java Database Connectivity. An application program interface specification for retrieving and manipulating data in a database. Calls are used to execute SQL operations.

**SQL (Structured Query Language):** It is a computer language used to create, retrieve, update and delete data from relational database management systems.

**UML (Unified Modelling Language):** It is a standardized specification language for object modelling. UML is a general-purpose modelling language that includes a graphical notation used to create an abstract model of a system, referred to as a UML model.

**XML:** The Extensible Markup Language (XML) is a general-purpose markup language. Its primary purpose is to facilitate the sharing of data across different information systems, particularly via the Internet.

**XPath:** It is an expression language for addressing portions of an XML document, or for computing values (strings, numbers, or boolean values) based on the content of an XML document. The XPath language is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria. In popular use (though not in the official specification), an XPath expression is often referred to simply as an XPath.

**XQuery:** It is a query language (with some programming language features) that is designed to query collections of XML data. It is semantically similar to SQL.

**XSLT (Extensible Stylesheet Language Transformations):** It is an XML-based language used for the transformation of XML documents. XSLT is designed to transform XML documents into other XML documents.

# 10 Bibliography and References

## Digital Bibliography

- [1] Base de datos en castellano; *Api para Bases de datos*. Last access (20-5-2007). Available at <http://www.programacion.net/bbdd/articulo/xmlapisdata/#xmlapisbasico>
- [2] Java Technology; Last access (1-6-2007). Available at <http://java.sun.com/>
- [3] López Sanz, Marcos; *Bases de datos XML* [PDF Document]. Last access (20-5-2007) Available at : [kybele.escet.urjc.es/MIGJRV/GIJRV/%5BGIJRV-2006-2007%5DTema2-Bases%20de%20Datos%20XML.pdf](http://kybele.escet.urjc.es/MIGJRV/GIJRV/%5BGIJRV-2006-2007%5DTema2-Bases%20de%20Datos%20XML.pdf)
- [4] Gaikkad, Arun. *Introduction to XIndex. An Open source native XML database system*. Last access (4-6-2007). Available at <http://www.ibm.com/developerworks/web/library/wa-xindex.html>
- [5] Moller, Anders; Schwartzbach Michael I. *The XML Revolution, Technologies for the future Web*. Last revision October 2003. Last access (20-5-2007). Available at <http://www.brics.dk/~amoeller/XML/index.html>
- [6] Oracle 10g Database Online Documentation, last access (20-5-2007). Available at [http://www.oracle.com/pls/db102/portal.portal\\_db?selected=7](http://www.oracle.com/pls/db102/portal.portal_db?selected=7).
- [7] PSOUG Oracle Morgan's Library SQL PL/SQL. Last access (1-6-2007). Available at <http://www.psoug.org/library.html>
- [8] The Code Project-Free Source Code and Tutorials. Last access (1-6-2007). Available at <http://www.codeproject.com>
- [9] XML documentation, last access (20-5-2007). Available at <http://www.w3.org/XML/>.
- [10] Oracle9i XML Developer's Kits Guide – XDK Release 2 (9.2), last access (20-5-2007). Available at <http://www.lc.leidenuniv.nl/awcourse/oracle/appdev.920/a96621/index.htm>.

- [11] W3Schools Online Web Tutorials. Last access (20-5-2007). Available at <http://www.w3schools.com/>

## **Bibliography**

- [12] Chandran, Sharat. *University Oracle XML: Develop Applications Instructor Guide*. First Edition 2001.
- [13] Lapis, George. *XTech 2005: XML, the Web and beyond. XML and relational Storage*.
- [14] Muench, Steve. *Building Oracle XML Applications*. First Edition September 2000. Ed. O'Reilly.
- [15] Roy-Faderman, Avrom; Koletzke, Peter; Dorsey, Paul. *Oracle Jdeveloper 10g Handbook*, First edition 2004 Ed. Mc Graw Hill.

# 11 Appendices

## 11.1 Installation of the application

The information about the installation appears in the CD, in Readme.txt file

- Before to install the application it is necessary install Oracle Database with XML packages to manage the XML. The Oracle database server and the information to install it appear in:  
<http://www.oracle.com/pls/db102/homepage>

It is necessary to register in that page to download the Oracle database server.

Other way to use it is use the Database of FIT pcuifs1.fit.vutbr.cz:1521:stud (only if the user have a account).

## 11.2 CD Contains

The information in this section appears in the CD in the Readme.txt file.

The CD contains several directories:

- Documentation: contains this document in electronic version. McThesis.pdf
- Sample application: contains the application (application directory), the code source (source directory) .