



# **XQuery in DB2, Oracle and SQL Server**

**Danilo Montesi**

`danilo.montesi@unibo.it`

`http://www.unibo.it/docenti/danilo.montesi`



# Introduction

---

- We will look at the XQuery standard support provided by most known DBMSs, highlighting any difference between the standard and its implementations.
- We will take in consideration the following DBMS:
  - DB2 Express-C (Version 10.1)
  - Oracle XML DB (Version 11.2)
  - MS SQL Server 2012
- We will show some queries executed on these three systems, highlighting the differences. In addition, we will look in particular at the unique features of the three systems, and the techniques that they provide to integrate XML files with relational databases.
- Finally we will make some references to Oracle Berkeley XML, a set of libraries dedicated to the management of XML databases.



# Example documents

---

- We will show several XQuery queries executed on different DBMSs.
- The documents that we are going to query are **Employees.xml** and **Departments.xml**, which represent respectively the employees of a company and the departments where they work.
- The two documents are 'linked' by the attribute *deptno*, that will be used to carry out eventual join operations.



# Example document: Departments

---

## Departments.xml

```
<depts>
  <dept deptno="10"  dname="Administration"/>
  <dept deptno="20"  dname="Marketing"/>
  <dept deptno="30"  dname="Purchasing"/>
  <dept deptno="40"  dname="Publishing"/>
  <dept deptno="50"  dname="Transport"/>
</depts>
```



# Example document: Employees

## Employees.xml

```
<emps>
  <emp empno="1" deptno="10" ename="John" salary="21000"/>
  <emp empno="2" deptno="10" ename="Jack" salary="310000"/>
  <emp empno="3" deptno="20" ename="Jill" salary="100001"/>
  <emp empno="4" deptno="30" ename="Andrew" salary="50000"/>
  <emp empno="5" deptno="30" ename="Smith" salary="123000"/>
  <emp empno="6" deptno="30" ename="James" salary="34000"/>
  <emp empno="7" deptno="40" ename="Tom" salary="210000"/>
  <emp empno="8" deptno="40" ename="Derek" salary="223000"/>
  <emp empno="9" deptno="50" ename="Alice" salary="120000"/>
  <emp empno="10" deptno="50" ename="Sandra" salary="12000"/>
</emps>
```

# **XQuery on DB2**





# DB2, XML and XQuery

---

- DB2 allows you to insert columns of **XML** type into relational tables, where you can then enter data using normal INSERT commands combined with **XMLPARSE** function, which takes as input a string and (if possible), converts it into an XML fragment that DBMS can handle.
- Because you can use the XML type, the database must use the **UTF-8** encoding (you can choose this option to its creation).
- XQuery queries are then inserted in the SELECT clause of SQL queries, using the **XMLQUERY** function, and will return XML fragments.
- It is also possible to use other functions to combine relational query and XQuery queries: **XMLEXISTS** can be used in the WHERE clause, and **XMLTABLE** is used in the FROM clause.



# DB2: Path expression

DB2 provides full support for all axes of the path expression, all node types and all tests required by the XPath standard.

## Esempi:

- `/descendant-or-self::book[attribute::year>1992]`  
(select all the elements 'book' belonging to the axe 'descendant-or-self' that have an attribute 'year' with value greater than 1992)
- `/child::bib/child::book/child::text()`  
(select all the elements of type text children of an element 'book' which in turn is child of an element 'bib')





## DB2: Creation of an XML table

---

To start our series of examples, we have to **create the tables** able to contain XML data: To do this, we simply specify **XML** as a data type of a column. In our case, we want two tables, each of which will contain one of the documents seen previously.

### Database examples

```
CREATE TABLE employees (data XML)
CREATE TABLE departments (data XML)
```



## DB2: Inserting XML data in a table

In order to insert data in the tables we just built, we will use **the function XMLPARSE** within a normal INSERT clause, passing **a string representing the XML document** we want to insert in the table. It is possible to choose **whether or not to keep the whitespace**.

### Insert example

```
INSERT INTO departments(data) VALUES
(XMLPARSE
  (document cast
    ('<depts>
      <dept deptno="10" dname="Administration"/>
      ....
      <dept deptno="50" dname="Transport"/>
    </depts>' as clob)
    preserve whitespace)
  )
```




## DB2: XQuery in SELECT

Now that we've entered the data in the tables, we begin to make some queries. In this first example, we want to extract **the names of all the departments** and insert them in **XML elements <deptName>**. We use the function XMLQUERY to enter the XQuery query in an SQL expression.

### XQuery 1

```
SELECT XMLQUERY
  ('for $d in $list//dept
   return <deptName>{ $d/@dname }</deptName>'
   passing dtable.data as "list")
FROM departments dtable
```

The *passing* clause is used to specify which XML fragment we are working on. In this case we **pass the content of the departments table as "list"**: in this way **a variable \$list will be created** that we can use within the query to refer to the XML fragment that we have previously inserted into the table.



## DB2: XQuery with **where** and **order by** in SELECT

Let's complicate the query a little bit: now we want to extract the names and the salaries of all employees with a salary greater than 50.000. In addition, we want the results to be inserted into <empSalary> elements and sorted by salary.

### XQuery 2

```
SELECT XMLQUERY
  ('for $e in $list//emp
   where $e/@salary > 50000
   order by $e/@salary
   return <empSalary>{ $e/@ename }{$e/@salary}</empSalary>'
   passing etable.data as "list")
FROM employees etable
```



## DB2: : XQuery with **let** and aggregation operators in SELECT

We want to extract for each employee his name, his salary and the average salary of the department where he works.

### XQuery 3

```
SELECT XMLQUERY
  ('for $e in $list//emp
   let $avgsal := avg($list//emp[@deptno = $e/@deptno]/@salary)
   return
  <averages>{$e/@ename}{$e/@salary}{$avgsal}</averages>'
   passing etable.data as "list")
FROM employees etable
```

The **let** clause will be executed at each for cycle, computing for each <emp> the average of @salary value of all the <emp> elements with the same @deptno.





## DB2: Conditional operators

In this example we want to create a <salaries> element that is **empty if the employee John earn more than the employee Smith**, and **that contains the salary of Smith otherwise**.

### XQuery 4

```
SELECT XMLQUERY
  ('let $john := $list//emp[@ename="John"]
   let $smith := $list//emp[@ename="Smith"]
   return
     if ($john/@salary > $smith/@salary)
     then <salaries/>
     else <salaries>{$smith/@salary}</salaries>'
   passing etable.data as "list")
FROM employees etable
```





## DB2: Sets operators

In this example we want to extract the names of all the employees who work in the department with @deptno 30, or of those who earn less than 100.000.

### XQuery 5

```
SELECT XMLQUERY
  ('let $emps30 := $list//emp[@deptno = 30]
let $empspoor := $list//emp[@salary < 100000]
for $e in ($emps30 union $empspoor)
return <empName>{$e/@ename}</empName>'
  passing etable.data as "list")
FROM employees etable
```

In this case, the **let** clause is outside the **for** clause, therefore it's computed just one time.



## DB2: Quantifiers

In this example we want to extract the names and the salaries of all the employees, but only if there is at least one employee who has a salary greater than 100000.

### XQuery 6

```
SELECT XMLQUERY
  ('if
    (some $emp in $list//emp satisfies
      ($emp/@salary > 100000))
    then
      <results>
        {for $e in $list//emp
          return <emp>{$e/@ename}{$e/@salary}</emp>}
        </results>
    else
      <results/>'}
  passing etable.data as "list")
FROM employees etable
```



# DB2: Join

In this example, we want to select the name of all the employees **who work in the department named "Purchasing"**.

## XQuery 7

```
SELECT XMLQUERY
  ('let $purchasingdep := $dlist//dept[@dname = "Purchasing"]
for $e in $elist//emp
where $e/@deptno = $purchasingdep/@deptno
return <empName>{$e/@ename}</empName>'
  passing etable.data as "elist", dtable.data as "dlist")
FROM employees etable, departments dtable
```

In order for the join to work, we use the *passing* clause to pass both the *employees* content and *departments* content.



## DB2: Join, **let** clause and aggregation operators

Now we want to select, **for each department**, its name and **the sum of the salaries of the employees who works in it**.

### XQuery 8

```
SELECT XMLQUERY
  ('for $d in $dlist//dept
   let $sumsalary :=
     sum($elist//emp[@deptno = $d/@deptno]/@salary)
   return <deptCost>{$d/@dname} {$sumsalary}</deptCost>'
   passing etable.data as "elist", dtable.data as "dlist")
FROM employees etable, departments dtable
```



## DB2: Last example of XQuery in SELECT

We want to select for each department the number of its employees and their average salary. We want the results to be sorted by the total cost (the sum of the salaries) of each department, and that only the departments with more than one employee must be included in the answer.

### XQuery 9

```
SELECT XMLQUERY
  ('for $d in $dlist//dept
   let $emps := $elist//emp[@deptno = $d/@deptno]
   where count($emps) > 1
   order by sum($emps/@salary) descending
   return
     <big-dept>
       {$d/@dname}
       <headcount> {count($emps)}</headcount>
       <avgsal>{avg($emps/@salary)}</avgsal>
     </big-dept>'
   passing etable.data as "elist", dtable.data as "dlist")

FROM employees etable, departments dtable
```



# **XQuery in Oracle DB**





# Oracle DB and XQuery

---

- Oracle DB provide full support to XQuery, in a way very similar to DB2: as we will see in the examples, most functions are the same.
- Oracle XML DB has been designed with particular focus on the coexistence between XQuery and SQL.



# Oracle DB: Unsupported features

---

Even if Oracle DB implements all the main features of XQuery, some features of the standard are not supported:

- **version encoding**: it is not possible to specify the encoding used inside an XQuery expression;
- **xml:id**: if this feature is used, an error is thrown;
- **xs:duration**: this type of data is not supported; it is possible to use xs:yearMonthDuration or xs:DayTimeDuration as an alternative.
- **schema validation feature** (optional in the standard);
- **module feature** (optional in the standard).



# Oracle DB e XPath 2.0

---

OracleDB provides full support to the functions and operators included in XPath 2.0 (as the operations between sets and quantifiers we've already seen for DB2), with some exceptions:

- you can not use the standard functions for the definition of regular expressions, but you have to use those built-in inside the DBMS;
- The functions **fn:id** and **fn:idref** are not supported;
- the function **fn:collection** without arguments is not supported.



# Oracle DB: Creation of an XML table

As in DB2, in Oracle DB we can enter columns of XML type in relational tables, using the keyword **XMLType**. It is moreover possible to create tables which are of XML type themselves.

## Examples:

*Creation of an XML table:*

```
CREATE TABLE employees OF XMLType
```

*Creation of a relational table with an XML column:*

```
CREATE TABLE departments  
  (id NUMBER(4),  
   data XMLType)
```

In the following examples we will assume that both *employees* and *departments* are tables of XML type.



# Oracle DB: Implementations of XMLType

---

The XMLType is an abstract construct, which can be implemented by the DBMS in two ways:

- Inside a **LOB (Large Object)**, that is, like a simple string.
- In a **Structured Storage**: choosing this option, the DBMS will build automatically tables and views that follow the XML schema of the document, and it will use them to contain the single nodes that the document contains, keeping the compliance to its *DOM (Document Object Model)*.

The choice of the implementation has no consequence on the methods the user will use to handle the *XMLTypes*; it has however consequences on the performance of the queries. It is not possible to determine which storage method is “superior”: in order to obtain the best performance you need to try case by case.





# Oracle DB: Entering XML data in a table

In order to enter data inside an XML table of column, we use the function **XMLType** within an INSERT command, passing the XML fragment we want to insert.

## Esempio:

```
INSERT INTO departments VALUES  
(XMLType  
( '<depts>  
  <dept deptno="10"  dname="Administration"/>  
  <dept deptno="20"  dname="Marketing"/>  
  <dept deptno="30"  dname="Purchasing"/>  
  <dept deptno="40"  dname="Publishing"/>  
  <dept deptno="50"  dname="Transport"/>  
</depts>' ) )
```





# Oracle DB: Example of query and differences with DB2

We show now a query we already ran in DB2: the differences are really minimal. In this example we want to extract **the name and the salary** of all the employees with a salary greater than 50000, ordered from the less paid to the most paid.

## Example of XQuery

```
SELECT XMLQUERY
  ('for $e in //emp
   where $e/@salary > 50000
   order by $e/@salary
   return <empSalary>{ $e/@ename }{$e/@salary}</empSalary>'
   passing etable.object_value
   returning content)
FROM employees etable
```

Apart from some differences in the **passing** clause and the adding of the clause **returning content**, the query system is basically the same of DB2.



# Oracle DB: Example of join

We show a query with a join between two XML tables: even in this case, the syntax is extremely similar to the one used in DB2. The aim of the query is to extract all the employees who work in the department named "Purchasing".

## Example of XQuery with join

```
SELECT XMLQUERY
  ('for $e in $elist//emp
   let $d := $dlist//dept[@dname = "Purchasing"]
   where $e/@deptno = $d/@deptno
   return $e'
   passing etable.object_value as "elist",
          dtable.object_value as "dlist"
   returning content)
FROM employees etable, departments dtable
```



# Oracle: working directly on XML documents


If we don't want to enter data in the tables, it is possible to use the **doc() function** to launch queries directly on the XML files. Let's see an example of the same query we've just seen, but this time applied to the document `employees.xml`.

```
SELECT XMLQUERY
  ('for $e in doc("employees.xml")//emp
   where $e/@salary > 50000
   order by $e/@salary
   return <empSalary>{ $e/@ename }{$e/@salary}</empSalary>'
   returning content)
FROM DUAL
```

The **DUAL** table is present by default in all Oracle databases, and it's empty: it only serves to compensate for the fact that the FROM clause is mandatory in all SQL query, even in the case in which the data are taken by other means (such as, in this case, the function **doc()** ).



# **XQuery in SQL Server 2012**



# SQL Server: Supported XQuery version

---

- Microsoft SQL Server support only a subset of the XQuery features, being based on the 2004 Working Draft of the standard.
- We will see now which features are not implemented in this DBMS.





# SQL Server: Sequences and path

As for the **Sequences**, the following limitations are to be reported:

- The sequences must be homogeneous, so they can only be composed of **nodes** or **atomic values**;
- The construct to build the sequences **to** is not supported;
- You can not use the operators **union**, **intersect**, **except** for combining sequences formed by nodes.

The **Path Expressions** are supported, but not completely:

- It is possible to use the axes **child**, **descendant**, **parent**, **attribute**, **self**, **descendant-or-self**;
- Only the node types *comment()*, *node()*, *processing-instruction()*, *text()* are supported.





# SQL Server: FLWOR expressions

---

The **FLWOR** expressions are completely supported.

- It is important to note that in SQL Server 2008 R2 the expressions assigned to a variable through the LET clause will be inserted in the query **every time that the variable has been mentioned in it.**
- This means that the expression will be not executed one time only, instead **it will be recomputed each time that there is a reference to that variable.**



# SQL Server: other features

---

Other supported operators are:

- Conditional constructs **if-then-else**;
- Comparison operators;
- Logical operators (only **and** and **or**; *not* is not available as a function);
- All the operators **with the exception of idiv**;
- All the **aggregation functions**;
- Expressions with existential and universal quantifiers (**some/satisfies** and **every/satisfies**).



# SQL Server: Creation of an XML table

---

As for DB2, in order to handle XML data we first need to create a table to contain it. To do so, we just need to insert an XML column in it.

## Example of tables

```
CREATE TABLE employees(  
    coll int primary key,  
    data XML)
```

```
CREATE TABLE departments(  
    coll int primary key,  
    data XML)
```



# SQL Server: Inserting XML data in a table

In order to enter data in the table we just created, it is sufficient to enter a **string containing the XML fragment** in the INSERT operation.

## Example of INSERT

```
INSERT INTO departments VALUES  
(1,  
( '<depts>  
  <dept deptno="10"  dname="Administration"/>  
  <dept deptno="20"  dname="Marketing"/>  
  <dept deptno="30"  dname="Purchasing"/>  
  <dept deptno="40"  dname="Publishing"/>  
  <dept deptno="50"  dname="Transport"/>  
</depts>' ) )
```



# SQL Server: differences from DB2 and OracleDB

We show the same query we had done for DB2 and OracleDB: we want to extract the name and salary of employees earning more than 50,000, ranked by salary.

## Example of XQuery:

```
SELECT data.query
  ('for $e in //emp
   where $e/@salary > 50000
   order by $e/@salary
   return <employee>{ $e/@ename }{$e/@salary}</employee>')
AS result
FROM employees
```

Apart from a few variations in passing the data (instead of using a passing clause, apply directly **the query () function to the XML column "data"** contained **in the employees table**), the syntax is the same as always.



# **XQuery in Oracle Berkeley DB XML**





# Oracle Berkeley: a set of libraries

---

- Oracle Berkeley it's not a real DBMS, but more of a set of libraries that can be used from within most common programming languages, to handle XML databases with simplicity.
- Oracle Berkeley does not provide any support for relational databases.
- It is possible to operate also from a text terminal interface which directly evaluate the commands (as in the examples we will see in the following slides).



# Oracle Berkeley: Main operations

---

- The most fundamental object, in Oracle Berkeley, is the **container** (roughly comparable to the tables of relational DBMS).
- Once a container is created, it is possible to **insert** XML documents in it.
- It is possible to **recover** XML fragments from a container by executing an **XQuery query**.
- It is possible to use XML Schema to insert some **constraints** on the documents.
- It is also possible to create indices that increase the performances.



# Oracle Berkeley: Creating a container

---

We can choose between two different ways of storing the data:

- **by document:** the documents are stored exactly as they have been given to the system;
- **by nodes:** the documents are decomposed in the nodes they are made of, which then are stored separately.

Storing by nodes allows to modify the document in the future, and also grant better performances.

## Example (through the console):

```
dbxml> createContainer esempio.dbxml
```



# Oracle Berkeley: Inserting a document

To insert documents in the current container we use the operation "putDocument <namePrefix> <string> [f|s|q]".

- Putting "f" as third parameter denotes that **the string contains the path of the XML file** we want to insert.
- Putting "s" as third parameter denotes that **the string contains the XML fragment** we want to pass.

## Examples (through the console):

```
dbxml> putDocument book1  
'<book><title>Title1</title><author>Author1</author><pages>  
100</pages></book>' s  
  
dbxml> putDocument book1 'Document.xml' f
```



# Oracle Berkeley: query

In order to retrieve documents inserted in a container, XQuery queries are used. Oracle Berkeley will run a search on all the documents contained in the container.

## Examples (through the console):

*Retrieve the whole content of the collection:*

```
dbxml> query 'collection("esempio.dbxml")'
```

*Retrieve the title of all the books that have John as author:*

```
dbxml> query  
'collection("esempio.dbxml")/book[author="John"]/title'
```

*Retrieve the title of all the books with a price greater than 100:*

```
dbxml> query  
'for $book in collection("esempio.dbxml")  
where $book/price > 100  
return $book/title'
```



# Oracle Berkeley: constraints

---

- During the creation of a container, it is possible to bind a schema to an XML document; this way, only XML documents which satisfies the schema validation will be allowed.
- It is also possible to specify different constraints for different documents, within the same container.





# **Appendix: Install DBMS**



# Install DB2

---

- From the website <http://www-01.ibm.com/software/data/db2/> it is possible to download the software needed to execute DB2.
- The free version of the DBMS is called "**DB2 Express-C**".
- At the address <http://www.ibm.com/developerworks/downloads/im/data/> it is possible to download **IBM Data Studio**, a free graphic interface extremely useful to execute any kind of operations.



# Install Oracle XML

---

- From the website [www.oracle.com](http://www.oracle.com), in the Downloads section, it is possible to freely download **Oracle Database 11g**, which also contains Oracle XML.
- In this case too a graphic user interface is needed to easily interact with the DBMS: this is called **Oracle SQL Developer**, and this is also freely available in the Downloads section of [www.oracle.com](http://www.oracle.com).



# Install Microsoft SQL Server 2012

---

- From the website <http://www.microsoft.com/sqlserver> it is possible to download **MS SQL Server 2012 Express**, a free version of the DBMS.
- By selecting the download package **Express with Tools**, it will also install a graphic user interface named **SQL Server Management Studio Express**.