



MONASH University

Information Technology

# FIT3176 Advanced Database Design

Topic 9: XML in Oracle

Dr Minh Le  
Minh.Le@monash.edu

**algorithm** distributed systems **database**  
systems **computation** knowledge ma  
**design** e-business **model** data mining **int**  
distributed systems **database** software  
**computation** knowledge management **an**

\*Adapted from slides developed by A/Prof David Taniar and Lindsay Smith

# This unit covers...

## 1. Advanced Database Design

- E/R is not complete enough
- EER, covering superclass and subclass

## 2. SQL and PL/SQL

- Trigger, Procedures/Functions

## 3. XML and XML DB

- XML and XML Schema
- **XML in Oracle** (*this week*)

## 4. Physical DB

- Relational Algebra (week 10)
- Query Optimization (week 11)

# Learning Objectives

*By the end of this week you should be able to:*

- Use XPath to locate elements and attributes within an XML document
- Describe the features of Oracle XML DB including the three storage options that are available in Oracle 11g and their relative uses/advantages
- Register and Deregister a schema in Oracle
- Create XMLType tables and columns
- Access XML data in an XMLType table or column using
  - extract/extractvalue
  - XMLQuery (including getStringVal()), and
  - XMLTable

# References

- Carey, P., *New Perspectives on XML 2<sup>nd</sup> Edition Comprehensive*, 2007, Thomson Course Technology, Tutorial 6 (Available in the library)
- Oracle XML DB Developers Guide 11g:  
[https://docs.oracle.com/cd/B28359\\_01/appdev.111/b28369/title.htm](https://docs.oracle.com/cd/B28359_01/appdev.111/b28369/title.htm)
- Oracle XML DB Developers Guide 12c:  
<https://docs.oracle.com/database/121/ADXDB/title.htm>
- W3Schools: [https://www.w3schools.com/xml/xquery\\_intro.asp](https://www.w3schools.com/xml/xquery_intro.asp)

# XML + Relational Data in Oracle

- Oracle Database is originally a pure RDBMS. Since Oracle8i, the database has shifted to become an ORDBMS. The generation of Oracle9i and above aims to become an XML-enabled DBMS
- To achieve full integration of XML and (object-)relational data, two aspects of functionality need to be addressed by Oracle
  - From a **data storage perspective**, both XML and relational data should be supported in the same database
  - From a **query language perspective**, both SQL-based and XML-based query should be available

# Storing XML in Oracle

- Oracle strategy for XML storage will depend on the nature of the XML document. The earlier approach has used the following techniques:
  - For **data-centric**, the XML documents can be stored in one or more tables
  - For **document-centric**, Oracle original solution is by storing the document in unstructured LOBs (Large Objects).
- However, the latest technique in Oracle has used the following method:
  - For **data-centric**, the XML documents can be **stored in a column** of a table. This column is defined as of XMLType format (column based XMLType).
  - For **document-centric**, the XML documents are **stored in a table** of XMLType format (table based XMLType).

# 1. Storing XML in Oracle

# Storing XML in Oracle

- **Step 1:** Register the schema into Oracle
- **Step 2:** Create the required table to store the XML data (e.g., XMLType column for data centric, and XMLType table for document centric). The default storage for XML data is BINARY.
- **Step 3:** Populate/Insert the XML data into the table



# Step 1: Register the Schema into Oracle

```
BEGIN DBMS_XMLSCHEMA.registerSchema(  
  SCHEMAURL=>'http://homepage.monash.edu.au/sample1',  
  SCHEMADOC ' ... schema in here ...',  
  LOCAL=>TRUE,  
  GENTYPES=>FALSE,  
  GENTABLES=>FALSE,  
  FORCE => FALSE,  
  options => DBMS_XMLSCHEMA.REGISTER_BINARYXML);  
END;  
/
```

- View local schemas you own

```
select * from user_xml_schemas;
```

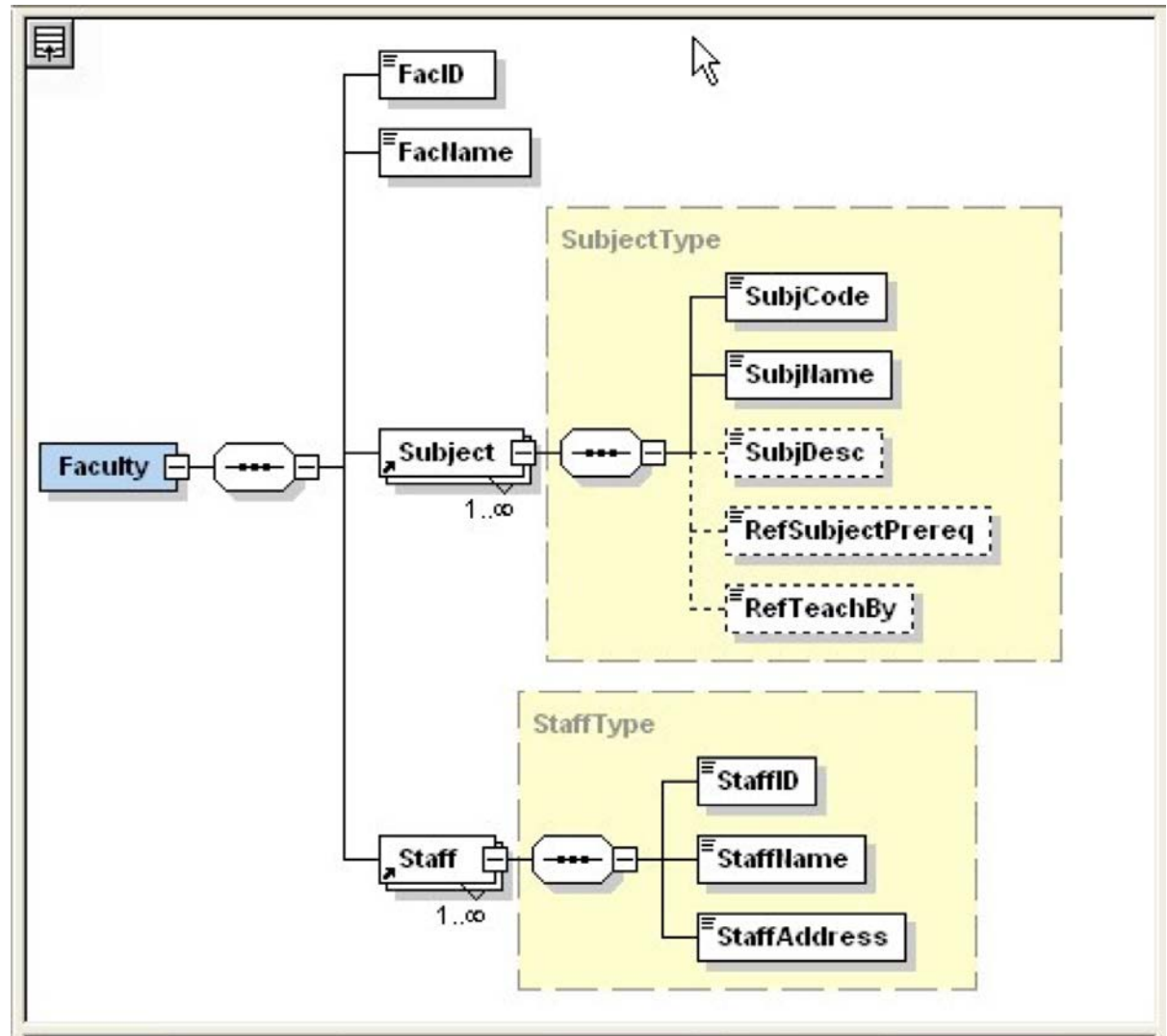
## NOTES:

**LOCAL => TRUE** indicates schema is registered as local (no sharing), if defines as FALSE then it is a global/sharable schema

**GENTYPES, GENTABLES** indicates whether or not types and tables are to be created automatically by oracle (=TRUE) or to be defined manually by the user (=FALSE).

# Step 1: Register the Schema into Oracle

## A Faculty Schema Case Study



# Step 1: Register the Schema into Oracle

```
BEGIN DBMS_XMLSCHEMA.registerSchema(  
  SCHEMAURL=>'http://fit3176.monash.edu/faculty_schema.xsd',  
  SCHEMADOC=>'<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
    <xs:element name="Faculty" >  
      <xs:complexType>  
        <xs:sequence>  
          <xs:element name="FacID" type="xs:ID" minOccurs="1" maxOccurs="1"/>  
          <xs:element name="FacName" type="xs:string" minOccurs="1" maxOccurs="1"/>  
          <xs:element ref="Subject" minOccurs="0" maxOccurs="unbounded"/>  
          <xs:element ref="Staff" minOccurs="0" maxOccurs="unbounded"/>  
        </xs:sequence>  
      </xs:complexType>  
    </xs:element>  
    <xs:complexType name="StaffType">  
      <xs:sequence>  
        <xs:element name="StaffID" type="xs:ID" minOccurs="1" maxOccurs="1"/>  
        <xs:element name="StaffName" type="xs:string" minOccurs="1" maxOccurs="1"/>  
        <xs:element name="StaffAddress" type="xs:string" minOccurs="1" maxOccurs="1"/>  
      </xs:sequence>  
    </xs:complexType>  
    <xs:complexType name="SubjectType">  
      <xs:sequence>  
        <xs:element name="SubjCode" type="xs:ID" minOccurs="1" maxOccurs="1"/>  
        <xs:element name="SubjName" type="xs:string" minOccurs="1" maxOccurs="1"/>  
        <xs:element name="SubjDesc" type="xs:string" minOccurs="0" maxOccurs="1"/>  
        <xs:element name="RefSubjectPrereq" type="xs:IDREF" minOccurs="0" maxOccurs="1"/>  
        <xs:element name="RefTeachBy" type="xs:IDREF" minOccurs="0" maxOccurs="1"/>  
      </xs:sequence>  
    </xs:complexType>  
    <xs:element name="Staff" type="StaffType"/>  
    <xs:element name="Subject" type="SubjectType"/>  
  </xs:schema>',  
  LOCAL=>TRUE, GENTYPES=>FALSE, GENTABLES=>FALSE, FORCE => FALSE, options =>  
    DBMS_XMLSCHEMA.REGISTER_BINARYXML);
```

END;

## To un-register (delete) a registered schema:

```
BEGIN
    dbms_xmlschema.deleteSchema('http://homepage.monash.edu/sample1',
    dbms_xmlschema.delete_cascade_force);
END;
/
```

## Step 2: Create Table

There are two kinds of tables (*choose one method*):

### 1. XMLType Column

```
CREATE TABLE table_name
  (attribute_name attribute_type,
    ...
    other attributes inserted here
    ...
    XML_attribute XMLTYPE)
XMLType COLUMN xml_attribute
XMLSCHEMA " 'http://homepage.monash.edu/sample1' ELEMENT "Root_element";
```

### 2. XMLType Table

```
CREATE TABLE table_name of XMLTYPE
XMLSCHEMA "http://homepage.monash.edu/sample1" ELEMENT "Root_element";
```

## Step 2: Create Table

In this case study, we are going to use **XMLType Column**.  
Hence the Create Table statement is:

```
CREATE TABLE Faculty_Lab  
(faculty_doc XMLTYPE)  
XMLType COLUMN faculty_doc  
XMLSCHEMA "http://fit3176.monash.edu/faculty_schema.xsd" ELEMENT  
"Faculty";
```

### Notes:

If you are choosing an **XMLType Table** method, the create table is as follows:

```
CREATE TABLE Faculty_Lab OF XMLTYPE  
XMLSCHEMA "http://fit3176.monash.edu.au/faculty_schema.xsd"  
ELEMENT "Faculty";
```

# Step 3: Insert Record

## 1. XMLType Column

```
INSERT INTO table_name VALUES (other_attribute_values, XMLType(
'<Root_element>
... the rest of the XML document goes here ...
</Root_element>').CreateSchemaBasedXML('http://homepage.monash.edu
.au/sample1'));
```

## 2. XMLType Table

```
INSERT INTO table_name VALUES (XMLType(
'<Root_element>
... the rest of the XML document goes here ...
</Root_element>').CreateSchemaBasedXML('http://homepage.monash.edu
.au/sample1'));
```

## Step 4: Insert Record

### Inserting three records using **XMLType** Column:

```
INSERT INTO Faculty_Lab
VALUES (XMLType(
'<Faculty>
  <FacID>FSTE</FacID>
  <FacName>Science, Technology and Engineering</FacName>
  <Subject>
    <SubjCode>CSE21DB</SubjCode>
    <SubjName>Database System</SubjName>
    <RefTeachBy>RH</RefTeachBy>
  </Subject>
  <Subject>
    <SubjCode>CSE42ADB</SubjCode>
    <SubjName>Advanced Database System</SubjName>
    <RefSubjectPrereq>CSE21DB</RefSubjectPrereq>
    <RefTeachBy>WR</RefTeachBy>
  </Subject>
  <Staff>
    <StaffID>WR</StaffID>
    <StaffName>Dr.Wenny Rahayu</StaffName>
    <StaffAddress>5 ABC St, Melbourne 3000</StaffAddress>
  </Staff>
  <Staff>
    <StaffID>RH</StaffID>
    <StaffName> Dr. Rob Haley </StaffName>
    <StaffAddress> 10 Plenty Road, Bundoora 3086</StaffAddress>
  </Staff>
</Faculty>' ).CreateSchemaBasedXML('http://fit3176.monash.edu/faculty_schema.xsd'));
```



## Step 4: Insert Record

The second record:

```
INSERT INTO Faculty_Lab
VALUES( XMLType(
'<Faculty>
  <FacID>FLB</FacID>
  <FacName>Law and Business</FacName>
  <Subject>
    <SubjCode>LAW41ML</SubjCode>
    <SubjName>Maritime Law</SubjName>
    <RefTeachBy>MH</RefTeachBy>
  </Subject>
  <Staff>
    <StaffID>MH</StaffID>
    <StaffName>Dr.Mark Hummel</StaffName>
    <StaffAddress>3 Red Gum St, Northcote 3070</StaffAddress>
  </Staff>
</Faculty>' ).CreateSchemaBasedXML('http://fit3176.monash.edu/facul
ty_schema.xsd') );
```

## Step 3: Insert Record

The third record:

```
INSERT INTO Faculty_Lab
VALUES( XMLType(
'<Faculty>
  <FacID>FE</FacID>
  <FacName>Education</FacName>
  <Subject>
    <SubjCode>EDU11TM</SubjCode>
    <SubjName>Teaching Methodology</SubjName>
    <RefTeachBy>MD</RefTeachBy>
  </Subject>
  <Staff>
    <StaffID>MD</StaffID>
    <StaffName>Dr.Michael Doulton</StaffName>
    <StaffAddress>56 Brown St, Fairfield 3078</StaffAddress>
  </Staff>
</Faculty>' ).CreateSchemaBasedXML('http://fit3176.monash.edu/facul
ty_schema.xsd' ) );
```

## 2. XPath

# Relative Paths

- With a relative path, the location of the node is indicated relative to a specific node in the tree called the context node
- Note that the path nomenclature is similar to UNIX OS pathnames
- For example if stock is the current context node:
  - . = the stock element
  - .. = the portfolio element
  - category = the child element category

Figure 6-8

Relative path expressions

Relative path	Description
.	Refers to the context node
..	Refers to the parent of the context node
<i>child</i>	Refers to the child of the context node named <i>child</i>
<i>child1/child2</i>	Refers to the <i>child2</i> node, a child of the <i>child1</i> node beneath the context node
<i>../sibling</i>	Refers to a sibling of the context node named <i>sibling</i>
<i>../descendant</i>	Refers to a descendant of the context node named <i>descendant</i>

# Using XPATH To Reference A Node

- For an absolute path, XPath begins with the root node, identified by a forward slash and proceeds down the levels of the node tree
  - An absolute path: /child1/child2/child3/...
  - /portfolio/stock/
- To reference an element without regard to its location in the node tree, use a double forward slash with the name of the descendant node
  - //descendant
    - //name - references all name nodes in the node tree (name node set)
- Referencing Groups of Elements
  - XPath allows you to refer to groups of nodes by using the wildcard character (\*)
    - /portfolio/stock/\* = all of the elements of the stock element
  - To select all of the nodes in the node tree, you can use the path: //\*
    - The (\*) symbol matches any node, and the (//)symbol matches any level of the node tree

# SPECIAL TYPES OF NODES

- Referencing attribute nodes
  - XPath uses different notation to refer to attribute nodes
  - The syntax for an attribute node is:
    - @attribute - where attribute is the name of the attribute
      - /portfolio/stock/sName/@symbol - absolute
      - sName/@symbol – relative based on stock context node
      - @symbol – relative based on the name context node
- Referencing the contents of an element (Text Nodes)
  - The text contained in an element node is treated as a text node
  - The syntax for selecting a text node is:
    - text()
      - /portfolio/stock/sName/text()
  - To match all text nodes in the document, use - //text()

**Q1. The XPATH listed below:  
/student[@gender="Female"]/givenname/text()  
would yield:**

- A. Mary, Phyllis
- B. Female
- C. Nothing
- D. <givenname>Mary</givenname>,  
<givenname>Phyllis</givenname>

```
<students>
  <student studid="S12345678" gender="Female">
    <givenname>Mary</givenname>
    <familyname>Jones</familyname>
    <crscore>3334</crscore>
    <dateenrolled>2012-02-27</dateenrolled>
    <currentload totalpoints="30">5</currentload>
  </student>
  <student studid="S22223344" gender="Male">
    <givenname>Sung Hoon</givenname>
    <familyname>Xiue</familyname>
    <crscore>3830</crscore>
    <dateenrolled>2013-07-02</dateenrolled>
  </student>
  <student studid="S33451087" gender="Female">
    <givenname>Phyllis</givenname>
    <familyname>Olson</familyname>
    <crscore>2380</crscore>
    <dateenrolled>2014-02-16</dateenrolled>
  </student>
</students>
```

**Q2. The XPATH listed below:**

**`//student[@gender="Female"]/givenname/text()`  
would yield:**

- A. Mary, Phyllis
- B. Female
- C. Nothing
- D. `<givenname>Mary</givenname>`,  
`<givenname>Phyllis</givenname>`

```
<students>
  <student studid="S12345678" gender="Female">
    <givenname>Mary</givenname>
    <familyname>Jones</familyname>
    <crscore>3334</crscore>
    <dateenrolled>2012-02-27</dateenrolled>
    <currentload totalpoints="30">5</currentload>
  </student>
  <student studid="S22223344" gender="Male">
    <givenname>Sung Hoon</givenname>
    <familyname>Xiue</familyname>
    <crscore>3830</crscore>
    <dateenrolled>2013-07-02</dateenrolled>
  </student>
  <student studid="S33451087" gender="Female">
    <givenname>Phyllis</givenname>
    <familyname>Olson</familyname>
    <crscore>2380</crscore>
    <dateenrolled>2014-02-16</dateenrolled>
  </student>
</students>
```



### Q3. The XPATH to find the date of enrolment (dateenrolled) for all students in course code 3334

- A. `//student[crscode="3334"]/dateenrolled/text()`
- B. `/students/student[crscode="3334"]/dateenrolled/text()`
- C. `//dateenrolled[../crscode="3334"]/text()`
- D. a, b and c

```
<students>
  <student studid="S12345678" gender="Female">
    <givenname>Mary</givenname>
    <familyname>Jones</familyname>
    <crscode>3334</crscode>
    <dateenrolled>2012-02-27</dateenrolled>
    <currentload totalpoints="30">5</currentload>
  </student>
  <student studid="S22223344" gender="Male">
    <givenname>Sung Hoon</givenname>
    <familyname>Xiue</familyname>
    <crscode>3830</crscode>
    <dateenrolled>2013-07-02</dateenrolled>
  </student>
  <student studid="S33451087" gender="Female">
    <givenname>Phyllis</givenname>
    <familyname>Olson</familyname>
    <crscode>2380</crscode>
    <dateenrolled>2014-02-16</dateenrolled>
  </student>
</students>
```

# 3. Using XPath to Query XML Tables

# XPATH like Operators for XMLType

- Oracle provides XPath-based methods to ease the handling of XML Type documents
- Table creation and DML (insert, update, delete) using XML Type tables or columns are allowed with this new data type
- This XMLType data type has member functions that take arguments with XPATH-like syntax to return fragments of XML in the XMLType
- Some common XPath functions:
  1. **extract**
  2. **extractValue**
  3. **existsNode**

# 1. The **extract** Function

- The **extract function** takes an XPath expression and returns the nodes that match the expression, as an XML document or fragment.
- If only a **single node** matches the XPath expression, then the result is a well-formed XML document.
- If **multiple nodes** match the XPath expression, then the result is a document fragment.
- The extract function can be used in **SELECT** clause or **WHERE** clause.

## a. Simple XPath traversals

```
SQL> SELECT extract(faculty_doc, 'Faculty/FacName').GETSTRINGVAL()  
"Faculty Name"  
2 FROM Faculty_Lab;
```

Faculty Name

-----

```
<FacName>Science, Technology and Engineering</FacName>  
<FacName>Law and Business</FacName>  
<FacName>Education</FacName>
```

```
SQL> SELECT extract(faculty_doc, 'Faculty/Subject').GETSTRINGVAL() "Subject"
2 FROM Faculty_Lab;
```

Subject

-----  
<Subject>

<SubjCode>CSE21DB</SubjCode>

<SubjName>Database System</SubjName>

<RefTeachBy>RH</RefTeachBy>

</Subject>

<Subject>

<SubjCode>CSE42ADB</SubjCode>

<SubjName>Advanced Database System</SubjName>

<RefSubjectPrereq>CSE21DB</RefSubjectPrereq>

<RefTeachBy>WR</RefTeachBy>

</Subject>

<Subject>

<SubjCode>LAW41ML</SubjCode>

<SubjName>Maritime Law</SubjName>

<RefTeachBy>MH</RefTeachBy>

</Subject>

<Subject>

<SubjCode>EDU11TM</SubjCode>

<SubjName>Teaching Methodology</SubjName>

<RefTeachBy>MD</RefTeachBy>

</Subject>

## b. Filtering conditions

```
SQL> SELECT extract(faculty_doc, 'Faculty[FacID="FSTE"]/FacName').GETSTRINGVAL()  
"Faculty Name"  
2 FROM Faculty_Lab;
```

Faculty Name

-----  
<FacName>Science, Technology and Engineering</FacName>

```
SQL> SELECT extract(faculty_doc, 'Faculty/Subject[RefTeachBy="RH"]/SubjName')  
2 FROM Faculty_Lab;
```

EXTRACT(FACULTY\_DOC, 'FACULTY/SUBJECT[REFTEACHBY="RH"]/SUBJNAME')

-----  
<SubjName>Database System</SubjName>

## c. Index elements

```
SQL> SELECT extract(faculty_doc, 'Faculty/Subject[1]') .GETSTRINGVAL( )  
2 FROM Faculty_Lab;
```

```
EXTRACT(FACULTY_DOC, 'FACULTY/SUBJECT[1]')
```

---

```
<Subject>  
  <SubjCode>CSE21DB</SubjCode>  
  <SubjName>Database System</SubjName>  
  <RefTeachBy>RH</RefTeachBy>  
</Subject>
```

```
<Subject>  
  <SubjCode>LAW41ML</SubjCode>  
  <SubjName>Maritime Law</SubjName>  
  <RefTeachBy>MH</RefTeachBy>  
</Subject>
```

```
<Subject>  
  <SubjCode>EDU11TM</SubjCode>  
  <SubjName>Teaching Methodology</SubjName>  
  <RefTeachBy>MD</RefTeachBy>  
</Subject>
```



## c. Index elements

```
SQL> SELECT extract(faculty_doc, 'Faculty/Staff[2]').GETSTRINGVAL()  
2 FROM Faculty_Lab;
```

```
EXTRACT(FACULTY_DOC, 'FACULTY/STAFF[2]')
```

```
-----  
<Staff>
```

```
  <StaffID>RH</StaffID>
```

```
  <StaffName> Dr. Rob Haley </StaffName>
```

```
  <StaffAddress> 10 Plenty Road, Bundoora 3086</StaffAddress>
```

```
</Staff>
```

## d. Going up traversals

```
SQL> SELECT extract(faculty_doc,  
'Faculty/Staff[2]/../FacName').GETSTRINGVAL()  
2 FROM Faculty_Lab;
```

```
EXTRACT(FACULTY_DOC, 'FACULTY/STAFF[2]/../FACNAME' )
```

```
-----  
<FacName>Science, Technology and Engineering</FacName>
```

## e. Checking if a node exists

```
SQL> SELECT extract(faculty_doc, 'Faculty/Subject[RefSubjectPrereq]/SubjName').  
GETSTRINGVAL()  
2 FROM Faculty_Lab;
```

```
EXTRACT(FACULTY_DOC, 'FACULTY/SUBJECT[REFSUBJECTPREREQ]/SUBJNAME' )
```

```
-----  
<SubjName>Advanced Database System</SubjName>
```

## 2. The **extractValue** Function

- The **extractValue** function takes an XPath expression and returns the corresponding leaf node.
- The XPath expression passed to `extractValue` should identify a single attribute or an element that has precisely **one text node child**.
- The result is returned in the appropriate SQL data type.
- The function `extractValue` is essentially a shortcut for `extract` plus either `getStringVal()` or `getNumberVal()`.

## a. **extract** versus **extractValue**

```
SQL> SELECT extract(faculty_doc, 'Faculty/Subject').  
GETSTRINGVAL()  
2 FROM Faculty_Lab  
3 WHERE extract(faculty_doc,  
'Faculty/Subject/RefTeachBy/text()').getStringVal() = 'MH';
```

```
EXTRACT(FACULTY_DOC, 'FACULTY/SUBJECT' )
```

---

```
<Subject>  
  <SubjCode>LAW41ML</SubjCode>  
  <SubjName>Maritime Law</SubjName>  
  <RefTeachBy>MH</RefTeachBy>  
</Subject>
```

Alternatively, the above query can be written as follows:

```
SQL> SELECT extractValue(faculty_doc, 'Faculty/Subject[RefTeachBy="MH"]/SubjName')
      2 FROM Faculty_Lab;
```

```
EXTRACTVALUE(FACULTY_DOC, 'FACULTY/SUBJECT[REFTEACHBY="MH"]/SUBJNAME' )
-----
```

Maritime Law

```
SQL> SELECT extractValue(faculty_doc, 'Faculty/Subject[SubjCode="CSE21DB"]/SubjName')
      2 FROM Faculty_Lab;
```

```
EXTRACTVALUE(FACULTY_DOC, 'FACULTY/SUBJECT[SUBJCODE="CSE21DB"]/SUBJNAME' )
-----
```

Database System

```
SQL> SELECT extractValue(faculty_doc, 'Faculty/Staff[StaffID="MD"]/StaffName')
      2 FROM Faculty_Lab;
```

```
EXTRACTVALUE(FACULTY_DOC, 'FACULTY/STAFF[STAFFID="MD"]/STAFFNAME' )
-----
```

Dr.Michael Doulton

Can you see the difference between the above three extractValue function results?

## b. Multi-columns, pipe, and concatenate operator

Displays Subject Code and Subject Name:

```
SQL> SELECT
  2      extractValue(faculty_doc,
'Faculty/Subject[RefTeachBy="MH"]/SubjCode') "Subject Code",
  3      extractValue(faculty_doc,
'Faculty/Subject[RefTeachBy="MH"]/SubjName') "Subject Name"
  4  FROM Faculty_Lab;
```

Subject Code

-----

Subject Name

-----

LAW41ML

Maritime Law

Another way of displaying more than one element is to use '|' (pipe), but you will need to use 'extract' rather than 'extractValue' for this.

```
SQL> SELECT extract (faculty_doc, 'Faculty/Subject[RefTeachBy="MH"]/SubjCode |  
2 Faculty/Subject[RefTeachBy="MH"]/SubjName' ).GETSTRINGVAL( )
```

```
3 FROM Faculty_Lab;
```

```
EXTRACT( FACULTY_DOC , ' FACULTY/SUBJECT[ REFTEACHBY= "MH" ] /SUBJCODE | FACULTY/SUBJECT[ R  
-----
```

```
<SubjCode>LAW41ML</SubjCode><SubjName>Maritime Law</SubjName>
```

If you need to get rid of the tags, then you can concatenate operator (||) to concatenate two extractValue function values:

```
SQL> SELECT extractValue(faculty_doc, 'Faculty/Subject[RefTeachBy="MH"]/SubjCode' ) ||  
2 extractValue(faculty_doc, 'Faculty/Subject[RefTeachBy="MH"]/SubjName' )  
3 FROM Faculty_Lab;
```

```
EXTRACTVALUE( FACULTY_DOC , ' FACULTY/SUBJECT[ REFTEACHBY= "MH" ] /SUBJCODE ' ) || EXTRACTVA  
-----
```

```
LAW41MLMaritime Law
```



## c. Errors when using **extractValue**

Cannot use `extractValue` to retrieve non-leaf node or multiple nodes:

```
SQL> SELECT extractValue(faculty_doc, 'Faculty/Subject[RefTeachBy="MH"]')  
2 FROM Faculty_Lab;
```

```
SELECT extractValue(faculty_doc, 'Faculty/Subject[RefTeachBy="MH"]')
```

ORA-19025: EXTRACTVALUE returns value of only one node

19025. 00000 - "EXTRACTVALUE returns value of only one node"

\*Cause: Given XPath points to more than one node.

\*Action: Rewrite the query so that exactly one node is returned.

```
SQL> SELECT extractValue(faculty_doc, 'Faculty[FacID="FSTE"]/Subject/SubjName')  
2 FROM Faculty_Lab;
```

## Using extract is ok:

```
SQL> SELECT extract(faculty_doc,  
'Faculty[FacID="FSTE"]/Subject/SubjName').GETSTRINGVAL() "Subject  
Name"  
FROM Faculty_Lab;
```

Subject Name

-----  
<SubjName>Database System</SubjName>  
<SubjName>Advanced Database System</SubjName>

## c. **extractValue** in the Where clause

```
SQL> SELECT extractValue(faculty_doc, 'Faculty/Subject/RefSubjectPrereq')  
      FROM Faculty_Lab;
```

```
EXTRACTVALUE ( FACULTY_DOC, ' FACULTY / SUBJECT / REFSUBJECTPREREQ ' )
```

```
-----  
CSE21DB
```

```
SQL> SELECT extract(faculty_doc, 'Faculty/Subject').GETSTRINGVAL()  
      FROM Faculty_Lab  
      WHERE extractValue(faculty_doc, 'Faculty/Subject/RefSubjectPrereq') = 'CSE21DB';
```

```
EXTRACT ( FACULTY_DOC, ' FACULTY / SUBJECT ' ) . GETSTRINGVAL ( )
```

```
-----  
<Subject>
```

```
  <SubjCode>CSE21DB</SubjCode>
```

```
  <SubjName>Database System</SubjName>
```

```
  <RefTeachBy>RH</RefTeachBy>
```

```
</Subject>
```

```
<Subject>
```

```
  <SubjCode>CSE42ADB</SubjCode>
```

```
  <SubjName>Advanced Database System</SubjName>
```

```
  <RefSubjectPrereq>CSE21DB</RefSubjectPrereq>
```

```
  <RefTeachBy>WR</RefTeachBy>
```

```
</Subject>
```

## c. **extractValue** in the Where clause

```
SQL> SELECT extract(faculty_doc, 'Faculty/Subject')  
      FROM Faculty_Lab  
      WHERE extractValue(faculty_doc, 'Faculty/Subject/RefSubjectPrereq') = 'CSE21DB';
```

```
EXTRACT(FACULTY_DOC, 'FACULTY/SUBJECT') .GETSTRINGVAL( )
```

```
-----  
<Subject>  
  <SubjCode>CSE21DB</SubjCode>  
  <SubjName>Database System</SubjName>  
  <RefTeachBy>RH</RefTeachBy>  
</Subject>  
<Subject>  
  <SubjCode>CSE42ADB</SubjCode>  
  <SubjName>Advanced Database System</SubjName>  
  <RefSubjectPrereq>CSE21DB</RefSubjectPrereq>  
  <RefTeachBy>WR</RefTeachBy>  
</Subject>
```

### **WARNING:**

This XPath returns both subjects, because path traversal is based on nodes, not based on instances. Because this record has RefSubjectPrereq CSE21DB, the Subject node of this record (which consists of two subjects) will be returned as the result of the query.


```
SQL>SELECT extract(faculty_doc, 'Faculty/Subject').GETSTRINGVAL( )  
FROM Faculty_Lab  
WHERE extract(faculty_doc,  
'Faculty/Subject/RefSubjectPrereq/text()').getStringVal() = 'CSE21DB';
```

```
EXTRACT(FACULTY_DOC, 'FACULTY/SUBJECT' ).GETSTRINGVAL( )
```

```
-----  
<Subject>  
  <SubjCode>CSE21DB</SubjCode>  
  <SubjName>Database System</SubjName>  
  <RefTeachBy>RH</RefTeachBy>  
</Subject>  
<Subject>  
  <SubjCode>CSE42ADB</SubjCode>  
  <SubjName>Advanced Database System</SubjName>  
  <RefSubjectPrereq>CSE21DB</RefSubjectPrereq>  
  <RefTeachBy>WR</RefTeachBy>  
</Subject>
```

### **WARNING:**

The same problem with getStringVal().



```
SQL> SELECT extract(faculty_doc,  
'Faculty/Subject[RefSubjectPrereq="CSE21DB"]') .getStringVal()  
FROM Faculty_Lab;
```

```
EXTRACT ( FACULTY_DOC , ' FACULTY / SUBJECT [ REFSUBJECTPREREQ = " CSE21DB " ] ' )
```


```
-----  
<Subject>  
  <SubjCode>CSE42ADB</SubjCode>  
  <SubjName>Advanced Database System</SubjName>  
  <RefSubjectPrereq>CSE21DB</RefSubjectPrereq>  
  <RefTeachBy>WR</RefTeachBy>  
</Subject>
```

## NOTES:

This is correct; checking is done at an instance level, not at a node level.

### 3. The **existsNode** Function

- The **existsNode** function is used in the WHERE clause of a SQL statement to restrict the set of documents returned by a query.
- The **existsNode** function takes an XPath expression and applies it to an XML document.
- The function returns TRUE (1) or FALSE (0), depending on whether or not the document contains a node that matches the XPath expression.



```
SQL> SELECT extract(faculty_doc, 'Faculty/FacName').getStringVal()  
"Faculty Name"  
      FROM Faculty_Lab  
      WHERE existsNode(faculty_doc, 'Faculty/Subject[SubjName="Database  
System"]') = 1;
```

Faculty Name

-----  
<FacName>Science, Technology and Engineering</FacName>

## The same query WITHOUT existsNode:

```
SQL> SELECT extract(faculty_doc, 'Faculty/Subject[SubjName="Database  
System"]/../../FacName').getStringVal() "Faculty Name"  
      FROM Faculty_Lab;
```

Faculty Name

-----  
<FacName>Science, Technology and Engineering</FacName>



# 4. XQuery

# XQuery

```
SELECT XMLQuery
    ( '<XQuery Expression using FLOWR Expressions>
      for $a in //<element_node>
      where <selection_predicate[conditional_expression]>
      return $a/<projection_element>'
    PASSING <OBJECT_VALUE>
    RETURNING CONTENT)
FROM <table_name>;
```

- The keyword **PASSING** followed by one or more SQL expressions (expr) that each return an XMLType instance.
- **RETURNING CONTENT** indicates that the value returned by an application of XMLQuery is an instance of parameterized XML type XML(CONTENT).

# XQuery

SELECT **XMLQuery**

```
(`<XQuery Expression using FLOWR Expressions>
  for $a in //<element_node>
  where <selection_predicate[conditional_expression]>
  return $a/<projection_element>`
```

PASSING <OBJECT\_VALUE>

RETURNING CONTENT)

FROM <table\_name>;

- **FLWOR** is the most general expression syntax in XQuery. It stands for For, Let, Where, Order By, Return
  - **For**: bind one or more variables each to any number of values. For each variable, iterate, binding the variable to a different value for each iteration.
    - for \$a in //movies/directormovie
    - for \$i in (2,3,4)
  - **Let**: bind one or more variables
    - let \$b := \$a/film/title
  - **Where**: filter for and let to some conditions (similar to where statement in SQL)
  - **Order By**: sort the result of where filtering
  - **Return**: construct a result from the ordered, filtered values. This will return the result of the FLWOR expression as a whole in a flattened sequence.

# Example 1:

```
SELECT XMLQuery('for $a in /Faculty
                return $a/FacName'
                PASSING faculty_doc
                RETURNING CONTENT).getStringVal() "Faculty Name"
FROM Faculty_Lab;
```

Faculty Name

-----

```
<FacName>Science, Technology and Engineering</FacName>
<FacName>Education</FacName>
<FacName>Law and Business</FacName>
```

- PASSING faculty\_doc
- FROM Faculty\_Lab
- XMLQuery('...') "FacultyName"
- for \$a in /Faculty
- return \$a/FacName
- RETURNING CONTENT

## Example 2:

```
SELECT XMLQuery('for $a in /Faculty/Subject
                let $b:=$a[RefTeachBy="MH"]
                return $b'
        PASSING faculty_doc
        RETURNING CONTENT).getStringVal() "Subject"
FROM Faculty_Lab;
```

Subject

---

```
<Subject>
  <SubjCode>LAW41ML</SubjCode>
  <SubjName>Maritime Law</SubjName>
  <RefTeachBy>MH</RefTeachBy>
</Subject>
```

- for \$a in /Faculty/Subject
- let \$b := \$a[RefTeachBy="MH"]
- return \$b

## Example 3:

```
SELECT XMLQuery('for $a in /Faculty/Subject
                let $b:=$a[RefTeachBy="MH"]
                return $b/SubjName'
                PASSING faculty_doc
                RETURNING CONTENT).getStringVal() "Subject Name"
FROM Faculty_Lab;
```

Subject Name

-----

<SubjName>Maritime Law</SubjName>

- for \$a in /Faculty/Subject
- let \$b := \$a[RefTeachBy="MH"]
- return \$b/SubjName

## Example 4:

```
SELECT XMLQuery
      ('for $a in /Faculty/Subject[SubjName="Database System"]
       return $a/../../FacName'
      PASSING faculty_doc
      RETURNING CONTENT).getStringVal() "Faculty Name"
FROM Faculty_Lab;
```

Faculty Name

-----  
<FacName>Science, Technology and Engineering</FacName>

## Example 5:

```
SELECT extract(faculty_doc, 'Faculty/Subject/../FacName').getStringVal()  
FROM Faculty_Lab;
```

```
EXTRACT(FACULTY_DOC, 'FACULTY/SUBJECT/../FACNAME')
```

---

```
<FacName>Science, Technology and Engineering</FacName>
```

```
<FacName>Law and Business</FacName>
```

```
<FacName>Education</FacName>
```



# 5. XMLTable

# XMLTable

- Maps the result of an XML Query into relational rows and columns
- XMLTable function creates a table consisting of the columns specified by the COLUMNS clause
- All normal SQL operation can then be used – join, group, aggregate functions etc.

# XMLTable Example

```
SQL> Select a.FacultyName
2   From Faculty_Lab,
3   XMLTable('/Faculty'
4   PASSING faculty_lab.faculty_doc
5   COLUMNS
6   FacultyID    varchar2(10) PATH '/Faculty/FacID',
7   FacultyName  varchar2(60) PATH '/Faculty/FacName'
8   ) a
9   Where a.FacultyID = 'FSTE';
```

FACULTYNAME

-----  
Science, Technology and Engineering

- XMLTable function creates a table consisting two columns: FacultyID and FacultyName
- Then it passes the Faculty\_Lab table (line 2) to the XMLTable (line 4), and the XMLTable function (lines 3-8) creates a table with two columns (lines 6-7). The alias for this table is “a” (line 8). The Where clause filters the records.

```
SQL> Select a.SubjectName
      2 From Faculty_Lab,
      3 XMLTable('/Faculty'
      4 PASSING faculty_lab.faculty_doc
      5 COLUMNS
      6 SubjectName varchar2(30) PATH '/Faculty/Subject/SubjName',
      7 FacultyID   varchar2(10) PATH '/Faculty/FacID'
      8 ) a
      9 Where a.FacultyID = 'FSTE';
Select a.SubjectName
*
ERROR at line 1:
ORA-19279: XPTY0004 - XQuery dynamic type mismatch: expected
singleton sequence
- got multi-item sequence
```

- The query attempts to create an XMLTable with two columns: SubjectName and FacultyID. Because **one faculty has many subjects**, the above query fails.

```

SQL> Select a.SubjectName
      2 From Faculty_Lab,
      3 XMLTable('/Faculty/Subject'
      4   PASSING faculty_lab.faculty_doc
      5   COLUMNS
      6     SubjectName varchar2(30) PATH 'SubjName'
      7   ) a;

```

SUBJECTNAME

-----

Database System

Advanced Database System

Maritime Law

Teaching Methodology

- The query returns all subjects (regardless the faculty).

Suppose we would like to display the subjects offered by faculty FSTE, like in the previous attempt, the following query solves the above problem, by combining with an XPath's existsnode in the Where clause of the query:

```
SQL> Select a.SubjectName
      2 From Faculty_Lab,
      3 XMLTable('/Faculty/Subject'
      4 PASSING faculty_lab.faculty_doc
      5 COLUMNS
      6 SubjectName varchar2(30) PATH 'SubjName'
      7 ) a
      8 Where existsnode(faculty_doc, '/Faculty[FacID="FSTE"]') = 1;
```

SUBJECTNAME

-----

Database System

Advanced Database System

## 6. Storing Relational Data in an XML Format

# Creating XML from relational data

- Manually structure via string concat - ||

```
1 select
2 '<employee deptno="' || dept_no || '">
3   <empname>' || rtrim(emp_name) || '</empname>'
4 </employee>'
5 from employee;
```

Script Output x

Task completed in 0.003 seconds

```
<employee deptno="1">
  <empname>Employee 1</empname>
</employee>

<employee deptno="1">
  <empname>Employee 2</empname>
</employee>
```



# Creating XML from relational data

- Use the XMLELEMENT & XMLATTRIBUTE Oracle functions

```
7 select xMLELEMENT("employee", xmlattributes(dept_no as "deptno"),
8        xMLELEMENT("empname", rtrim(emp_name)))
9 from employee;
```

Script Output x

Task completed in 0.004 seconds

XMLELEMENT("EMPLOYEE",XMLATTRIBUTES(DEPT\_NOAS"DEPTNO"),XMLELEMENT("EMPNAME",RTRIM(EMP\_NAME)))

```
<employee deptno="1"><empname>Employee 1</empname></employee>
<employee deptno="1"><empname>Employee 2</empname></employee>
<employee deptno="2"><empname>Employee 3</empname></employee>
<employee deptno="2"><empname>Employee 4</empname></employee>
<employee deptno="2"><empname>Employee 5</empname></employee>
```

- Use the XMLFOREST function to create a 'forest' of child elements

```
7 select xMLELEMENT("employee", xmlattributes(dept_no as "deptno"),
8        xmlforest(rtrim(emp_name) as "empname", emp_salary as "salary"))
9 from employee;
```

Script Output x Query Result x

All Rows Fetched: 5 in 0.006 seconds

XMLELEMENT("EMPLOYEE",XMLATTRIBUTES(DEPT\_NOAS"DEPTNO"),XMLFOREST(RTRIM(EMP\_NAME)AS"EMPNAME",EMP\_SALARYAS"SALARY"))

```
1 <employee deptno="1"><empname>Employee 1</empname><salary>35000</salary></employee>
2 <employee deptno="1"><empname>Employee 2</empname><salary>40000</salary></employee>
3 <employee deptno="2"><empname>Employee 3</empname><salary>45000</salary></employee>
4 <employee deptno="2"><empname>Employee 4</empname><salary>50000</salary></employee>
5 <employee deptno="2"><empname>Employee 5</empname><salary>56000</salary></employee>
```