Big Data and Data Mining

Relational Data and XML

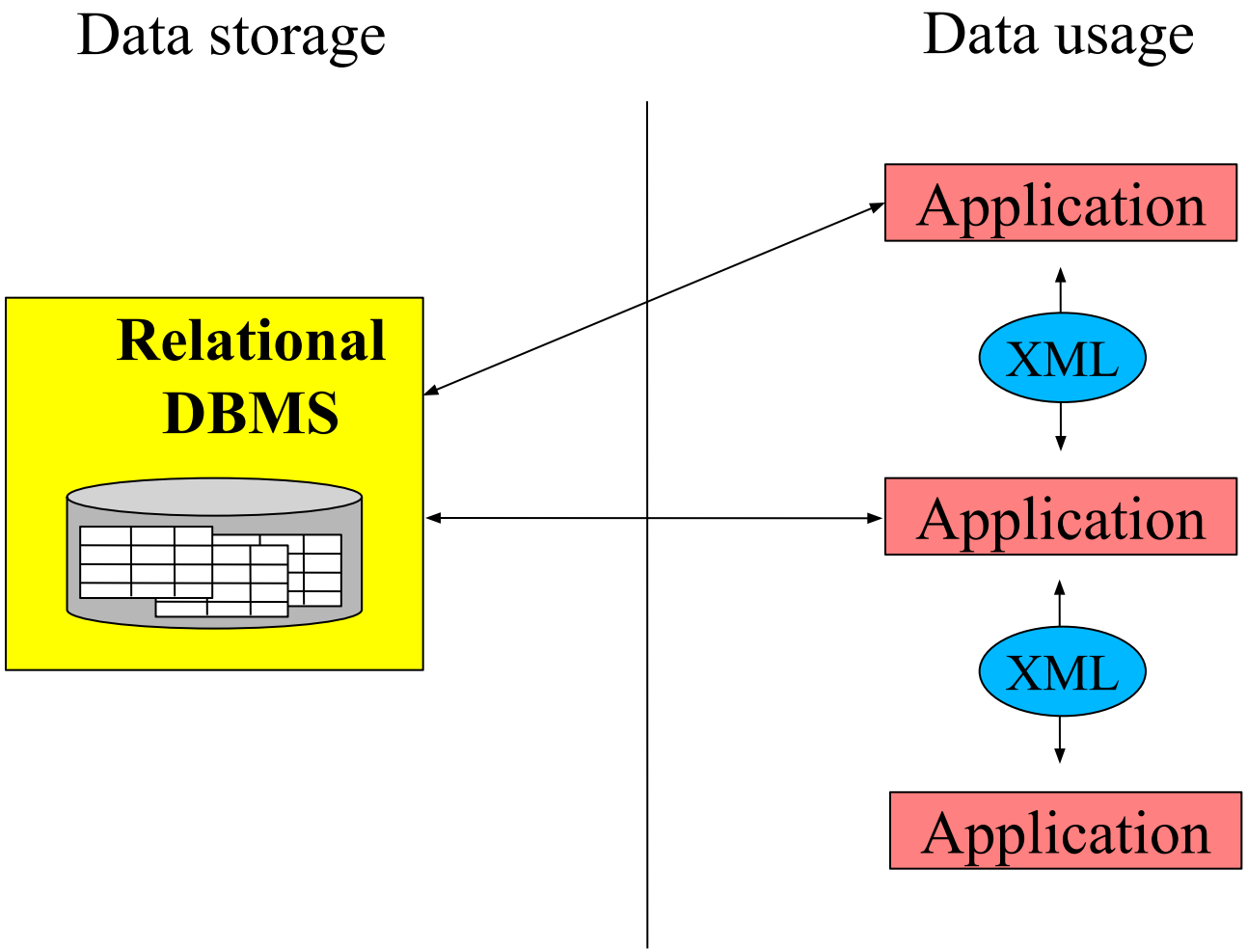
Flavio Bertini

flavio.bertini@unipr.it

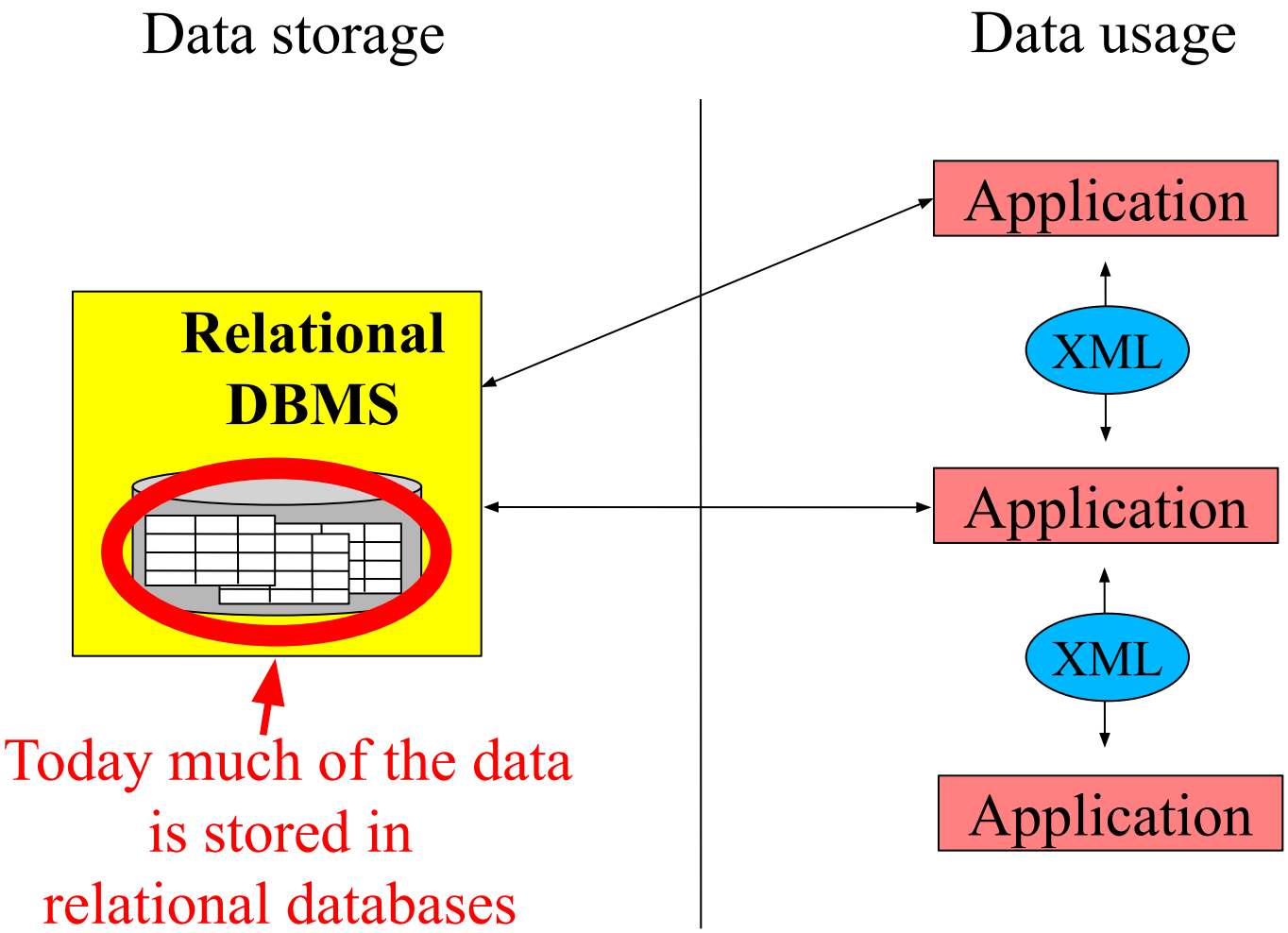
Relational Data and XML - Outline

- Combining relational data and XML
  - From SQL to XML
  - From XML to SQL
- The SQL/XML language
  - XMLELEMENT
  - XMLFOREST
  - XMLCONCAT
  - XMLAGG
  - XMLGEN

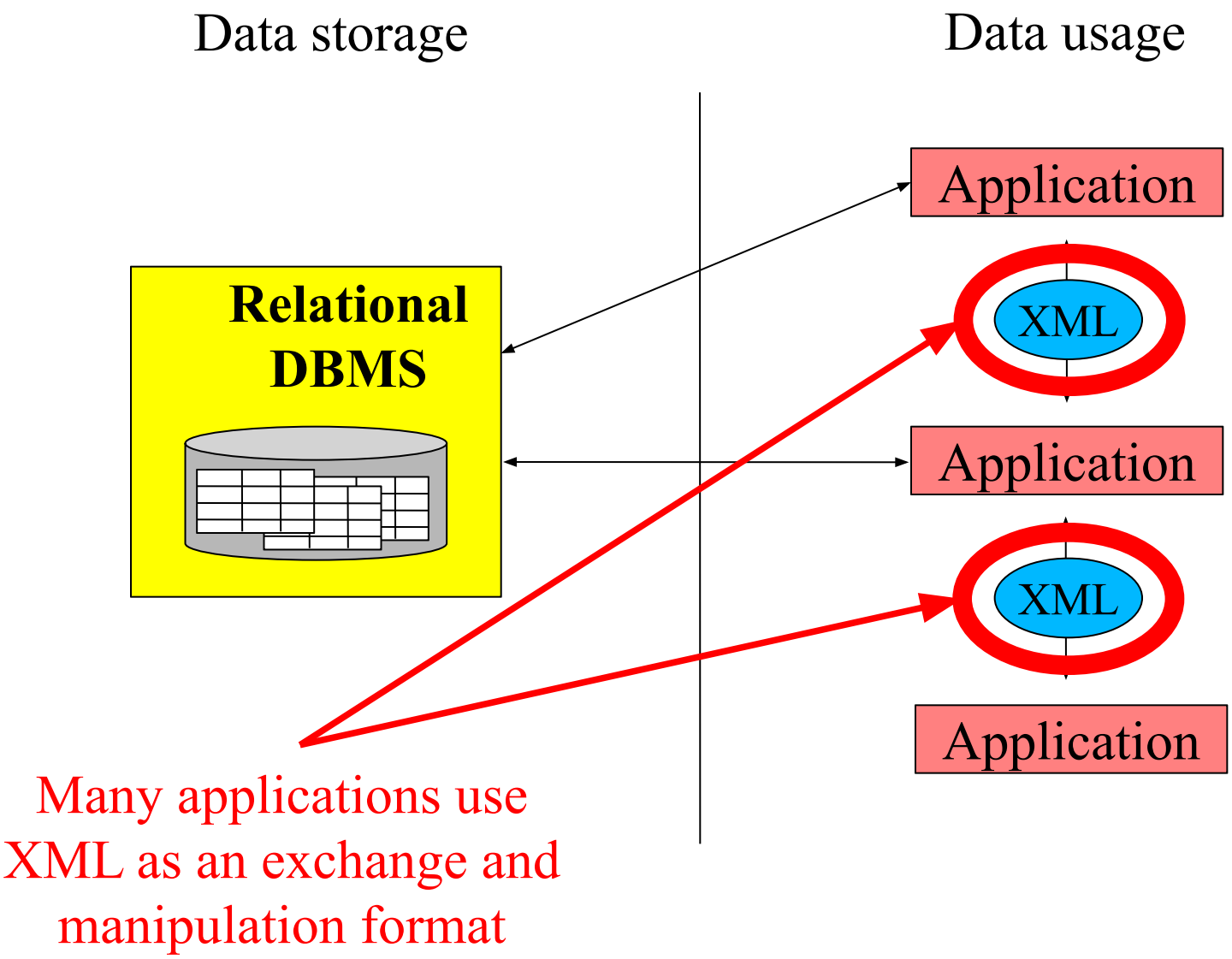
Introduction 1/4



Introduction 2/4



Introduction 3/4



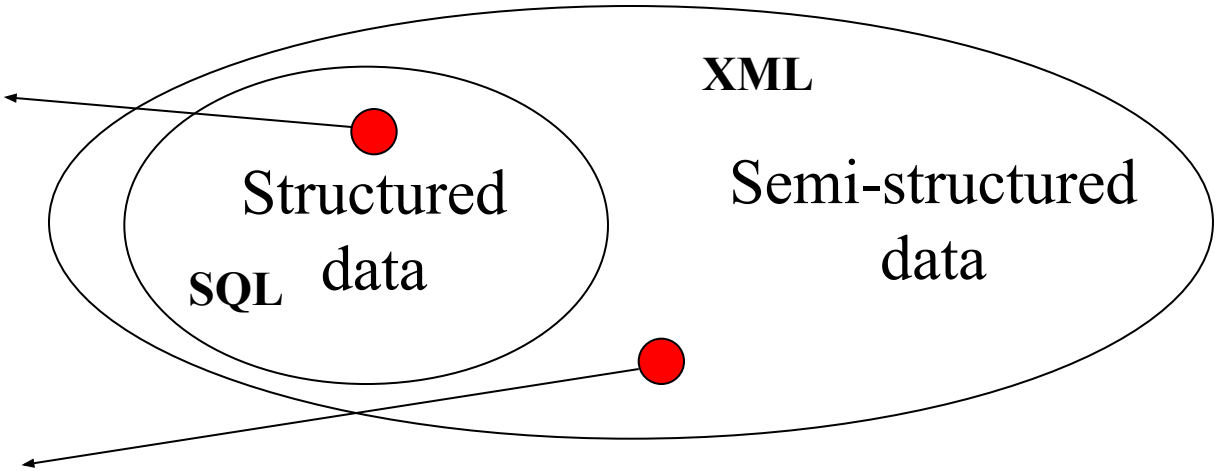
Introduction 4/4

- XML is used primarily for two types of data management activities:
  - Representation of **semi-structured data**
  - **Exchange of data** between applications
- When the data exchanged between different applications are locally stored in relational databases, a “bridge” is needed between these two formats
- SQL/XML, which is a standard extension to SQL, provides a common language to convert relational data to XML

Features: SQL → XML and XML → SQL

- To use a combination of relational and XML data requires two main functions:
  - **Extracting XML** from one or more relational tables
  - XML **storage** in one (or more) relational tables
- The first feature is conceptually simpler, since XML was created in order to represent both semi-structured and structured data
- The second is more complex in general, for the same reason

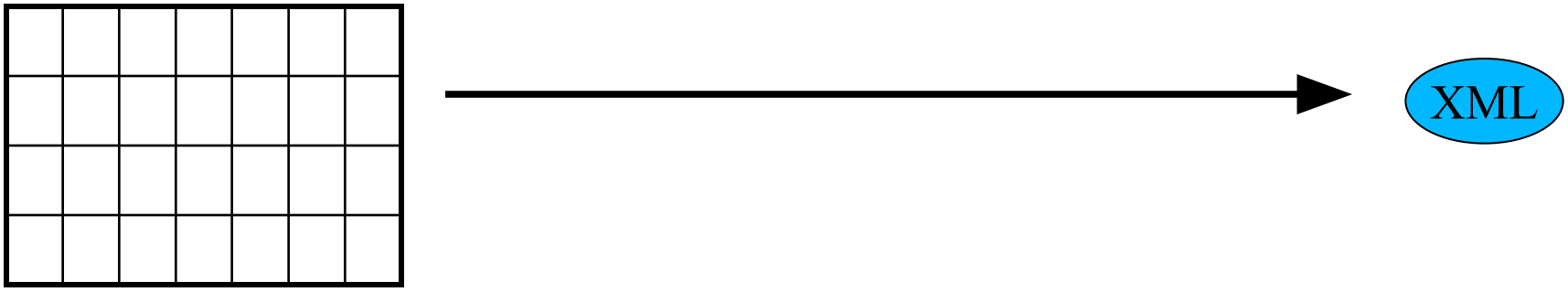
Data that can be represented both in relational and XML



Data that can be hardly represented in relational

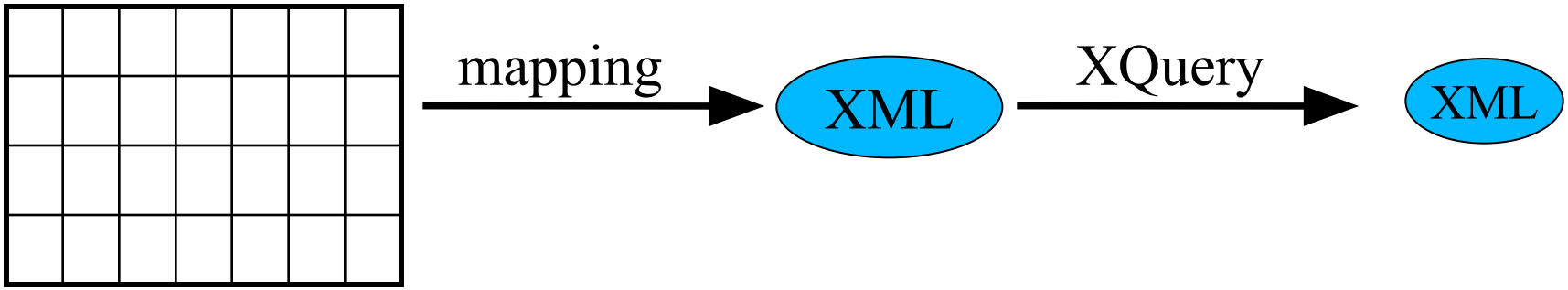
Extracting XML from a table

- Extracting XML data from a table is simple, because any type of data that can be represented in a table can be also represented in XML
  - Let's see two alternative ways to implement the extraction

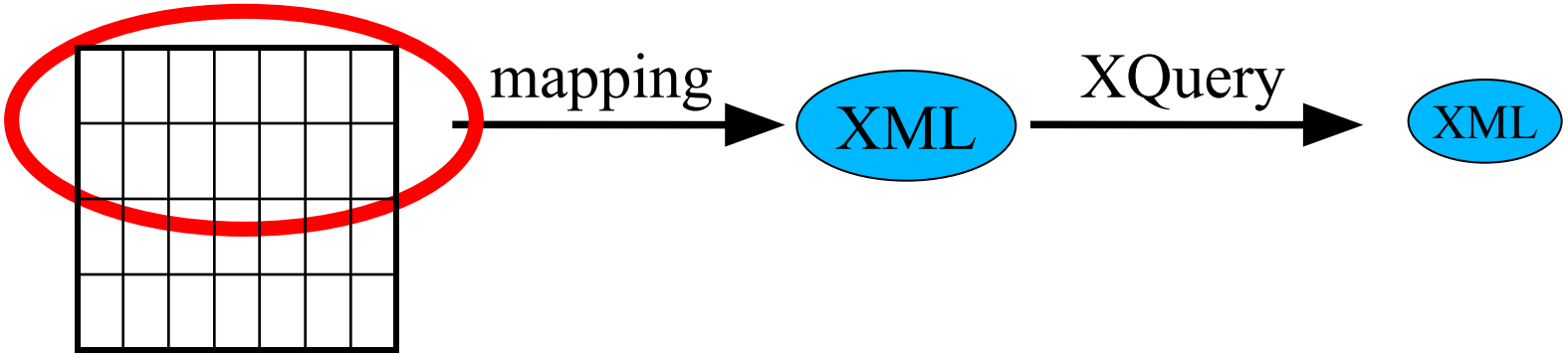


1: Extracting XML from a table (XML + XQuery)

- Represent the table in XML (mapping)
- Extract data using XML technologies (XQuery)



1: Extracting XML from a table (XML + XQuery)

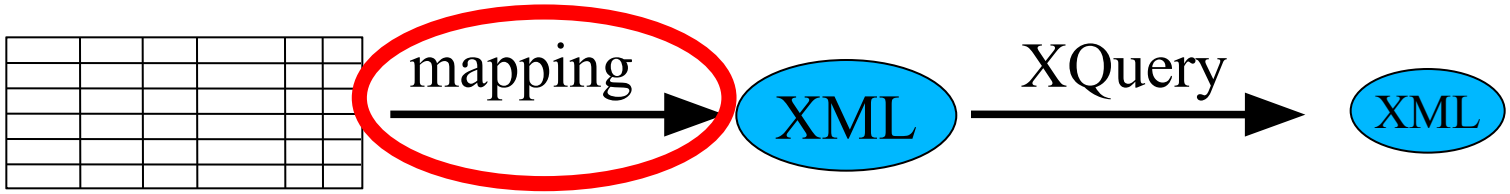


EMPLOYEES

ID	NAME	SURNAME	SALARY
emp0001	John	Doe	20000
emp0002	Jack	Black	18000

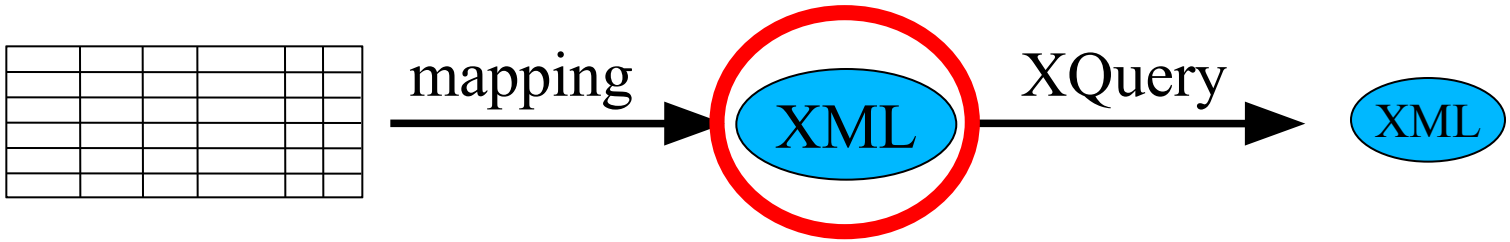
We start from a relational table

1: Extracting XML from a table (XML + XQuery)



- The mapping proceeds in the following way:
  1. The name of the table becomes the name of the document
  2. Each row is included in an element <row>
  3. Each value (column) is included in an element with the name of the (SQL) attribute
  4. Null values are represented using the attribute xsi:nil="true"

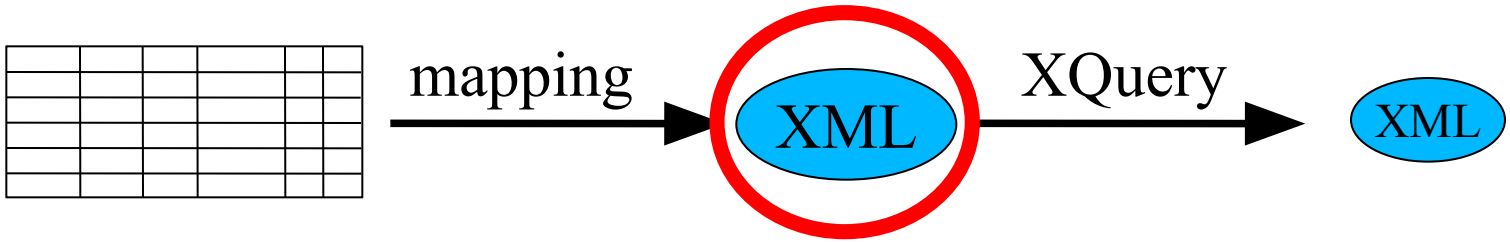
1: Extracting XML from a table (XML + XQuery)



This mapping is defined by SQL/XML

```
<EMPLOYEES>
  <row>
    <ID>emp0001</ID>
    <NAME>John</NAME>
    <SURNAME>Doe</SURNAME>
    <SALARY>20000</SALARY>
  </row>
  <row>
    <ID>emp0002</ID>
    <NAME>Jack</NAME>
    <SURNAME>Black</SURNAME>
    <SALARY>18000</SALARY>
  </row>
</EMPLOYEES>
```

1: Extracting XML from a table (XML + XQuery)



- The standard defines also an XML schema with the definitions of each type of data in SQL and each XML element
  - For instance, CHARACTER(6) produces:

```
<xsd:sempleType name="CHAR_6">
   <xsd:restriction base="xsd:string">
      <xsd:length value="6" />
   </xsd:restriction>
</xsd:sempleType>
```

Fragment of XML Schema, provided as an example (it is not necessary to remember its details)

1: Extracting XML from a table (XML + XQuery)

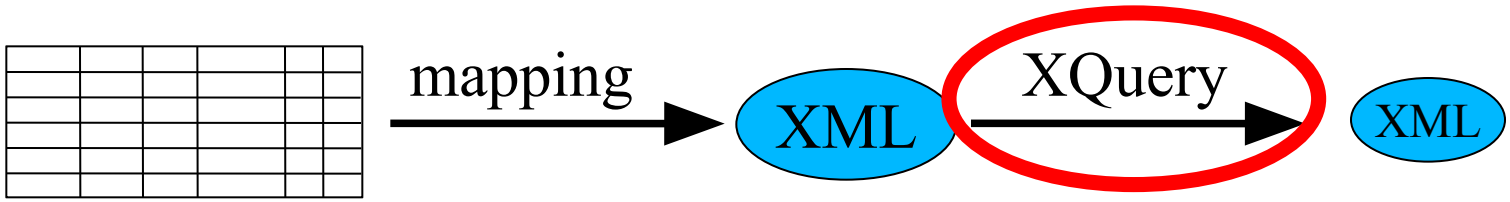
XML schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Address">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Recipient" type="xs:string" />
        <xs:element name="House" type="xs:string" />
        <xs:element name="Street" type="xs:string" />
        <xs:element name="Town" type="xs:string" />
        <xs:element name="County" type="xs:string"
                    minOccurs="0" />
        <xs:element name="PostCode" type="xs:string" />
        <xs:element name="Country">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="FR" />
              <xs:enumeration value="DE" />
              <xs:enumeration value="ES" />
              <xs:enumeration value="UK" />
              <xs:enumeration value="US" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  </xs:schema>
```

XML file

```
<?xml version="1.0" encoding="utf-8"?>
<Address
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SimpleAddress.xsd">
  <Recipient>Mr. Walter C. Brown</Recipient>
  <House>49</House>
  <Street>Featherstone Street</Street>
  <Town>LONDON</Town>
  <PostCode>EC1Y 8SY</PostCode>
  <Country>UK</Country>
</Address>
```

1: Extracting XML from a table (XML + XQuery)

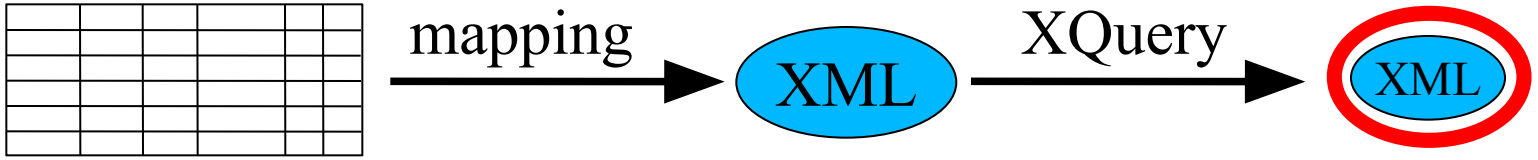


Non-standard XQuery function

```
<highSalaries>
   { for $e in table("EMPLOYEES")/EMPLOYEES/row
     where $e/salary > 18000
     return
        <employee>
           {$e/surname, $e/name}
        </employee>
   }
</highSalaries>
```

An XQuery query, provided as an example (we will not see XQuery in details in these slides)

1: Extracting XML from a table (XML + XQuery)



EMPLOYEES

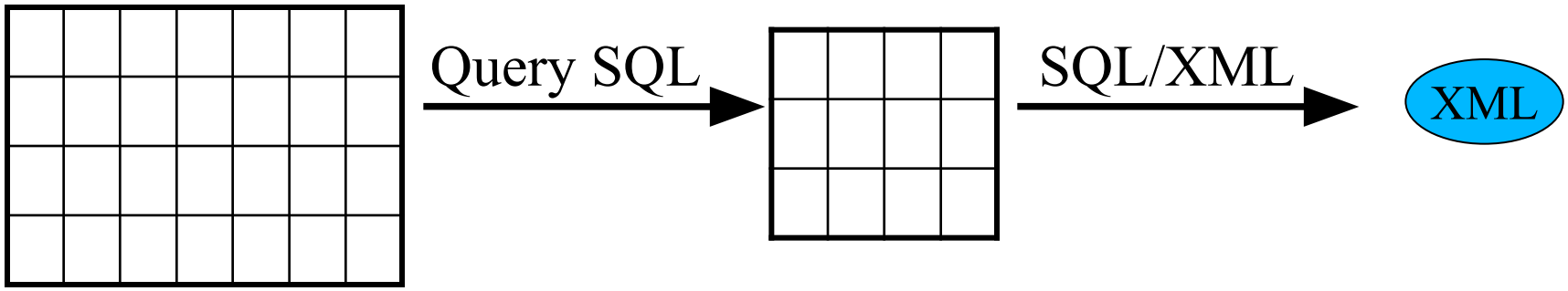
ID	NAME	SURNAME	SALARY
emp0001	John	Doe	20000
emp0002	Jack	Black	18000

- The result is the following:

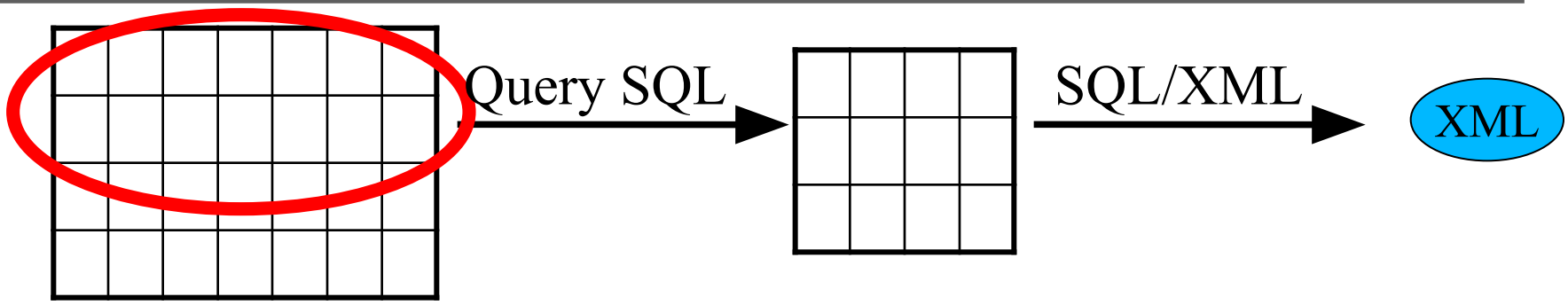
```
<highSalaries>
   <employee>
      <name>John</name>
      <surname>Doe</surname>
   </employee>
</highSalaries>
```

2: Extracting XML from a table (SQL/XML)

- Extract data from the table (SQL)
- Transform this data in XML (SQL/XML)



2: Extracting XML from a table (SQL/XML)

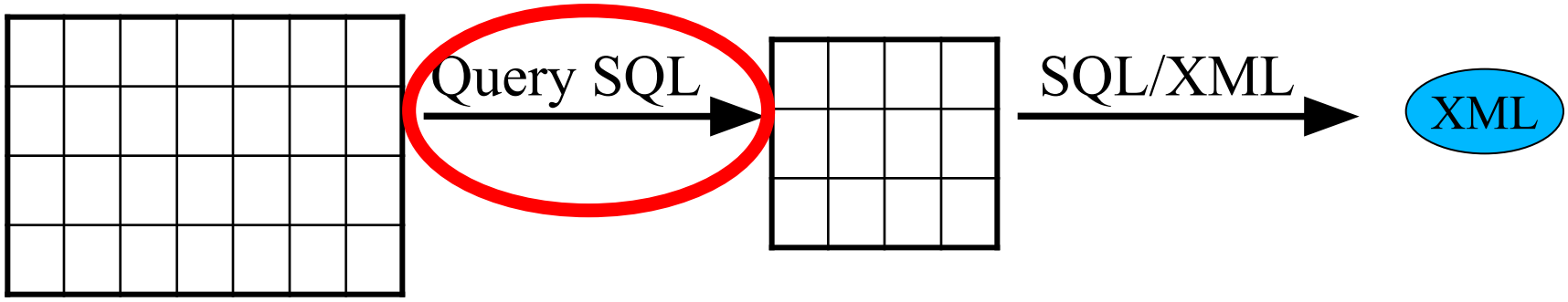


EMPLOYEES

ID	NAME	SURNAME	SALARY
emp0001	John	Doe	20000
emp0002	Jack	Black	18000

We start from the same table as in the previous example

2: Extracting XML from a table (SQL/XML)

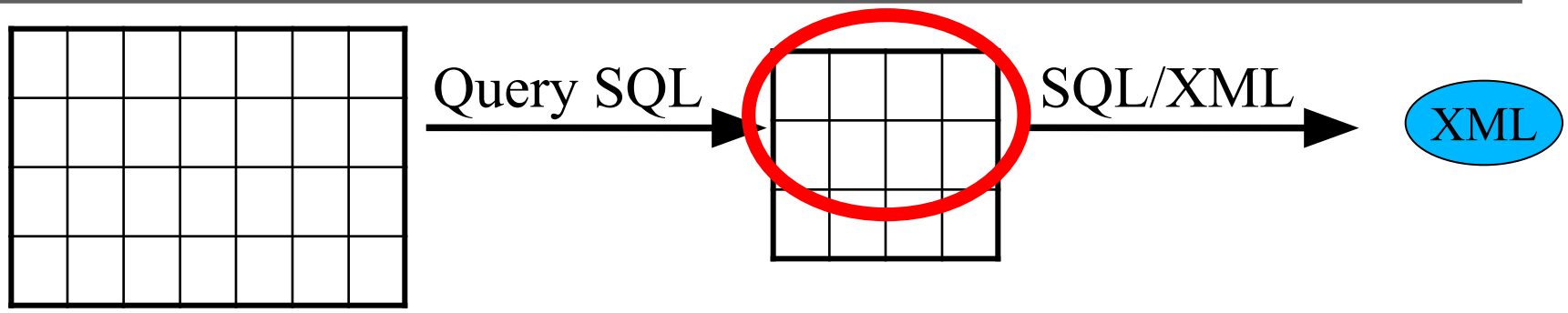



```
SELECT i.name AS employeeName

FROM EMPLOYEES i WHERE salary > 18000
```

We write an SQL query to get the data of interest

2: Extracting XML from a table (SQL/XML)

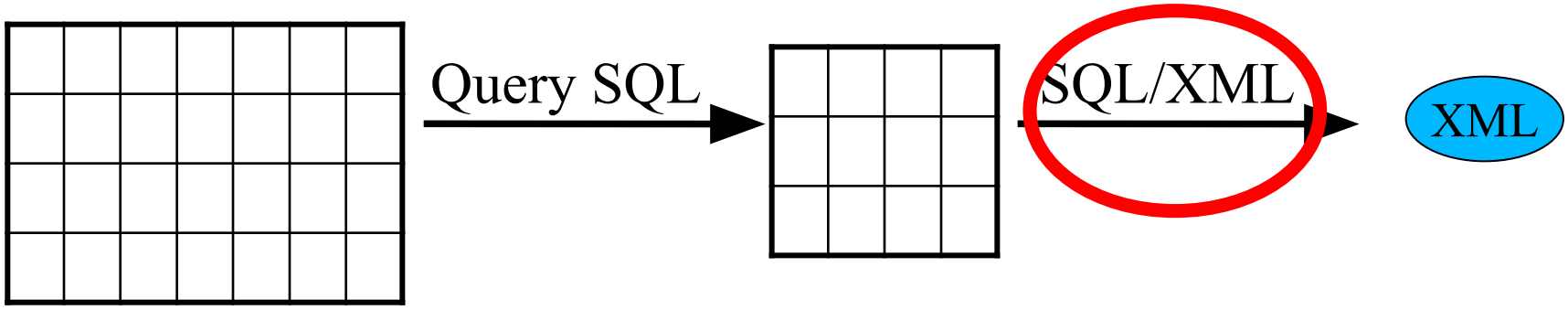


SELECT i.name AS employeeName

FROM EMPLOYEES i WHERE salary > 18000

employeeName
John

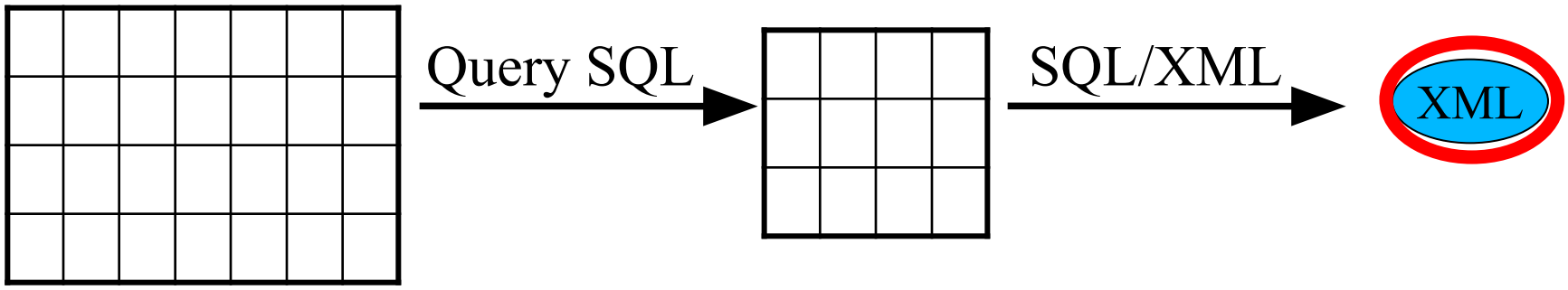
2: Extracting XML from a table (SQL/XML)




```
SELECT XMLELEMENT(
        NAME "emp",
        i.name ) AS employeeName
FROM EMPLOYEES i WHERE salary > 18000
```

We add the XML constructors

2: Extracting XML from a table (SQL/XML)




```
SELECT XMLELEMENT(
        NAME "emp",
        i.name ) AS employeeName
FROM EMPLOYEES i WHERE salary > 18000
```

employeeName
<emp>John</emp>

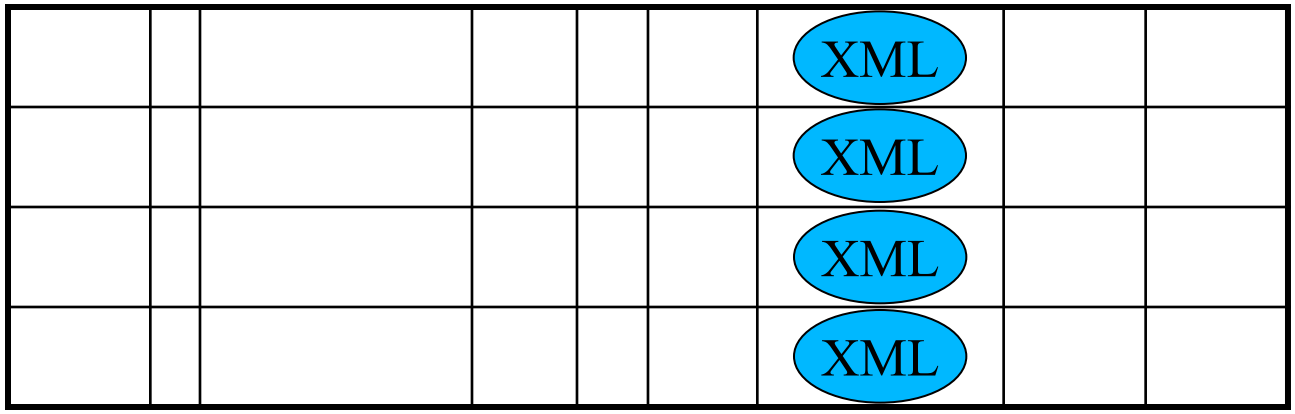
More properly, in this way we are not extracting XML code, but tables containing XML code, which can be then restored and used as XML

Storing XML in Relational DBs

- To provide this functionality, different systems use *ad hoc* techniques
- These can be traced to two main modes:
  - Using Object-Relational columns to store entire XML fragments in a single field
  - *Shredding* ("chopping") of the documents, for which different items are stored in different fields

Inserting XML 1/2

- An XML document can be viewed as a type of SQL data
  - In this case, an entire document is stored in an attribute (column) of a table

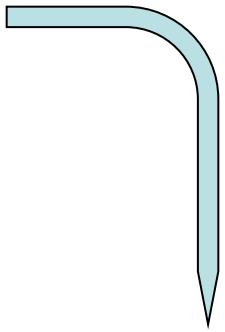


Inserting XML 2/2 (Shredding)

- With some limitations, an XML document can be splitted in fragments and stored in pieces

```
<employee>
   <ID>emp0001</ID>
   <SURNAME>Doe</SURNAME>
   <SALARY>20000</SALARY>
</employee>
<employee>
   <ID>emp0002</ID>
   <SURNAME>Black</SURNAME>
   <SALARY>18000</SALARY>
</employee>
```

FILE FOR MAPPING DEFINITION



ID	SURNAME	SALARY
emp0001	Doe	20000
emp0002	Black	18000

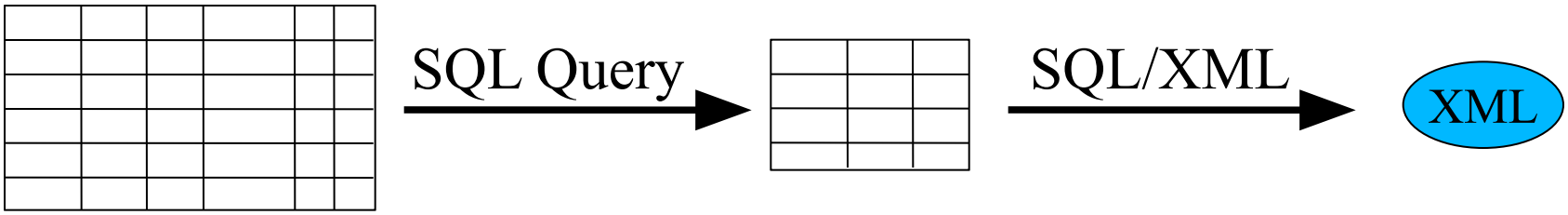
The SQL/XML language

SQL/XML: a SQL extension for XML

- SQL/XML is an extension of SQL
- It is a part of the Structured Query Language (SQL) specification
- It comprises of constructors, routines, functions to support manipulation and storage of XML in a SQL Database

Extracting XML with SQL/XML 1/2

- SQL/XML defines the following operators
  - XMLELEMENT
  - XMLFOREST
  - XMLCONCAT
  - XMLAGG
  - XMLGEN



https://oracle-base.com/articles/misc/sqlxml-sqlx-generating-xml-content-using-sql

Extracting XML with SQL/XML 2/2

- An SQL/XML query has the following structure:

  SELECT Attr1, Attr2, ..., *Constructor XML*

  FROM…

  WHERE...

- To compute its result
  - FIRST: only SQL is considered, and a table is computed as it was a SELECT *
  - THEN: the result is built, selecting the requested attributes (Attr1, Attr2...) and building the **XML code FOR EACH TUPLE**

Table used in the examples

EMPLOYEE

id	*name*	*surname*	*department*	*dismissed*	*salary*
emp0001	John	Doe	Sales	null	20000
emp0002	Jack	Black	Sales	null	18000
emp0003	Donald	Mason	Accounting	null	15000
emp0004	Samuel	Wood	Accounting	01/01/03	15000

XMLELEMENT

- XMLELEMENT allows to create an XML element
- It takes the following arguments:
  - The name of the element
  - An optional list of attributes
  - The content of the element

XMLELEMENT – example

```
SELECT i.id, XMLELEMENT(
        NAME "emp",
        i.name ) AS result
FROM EMPLOYEES i
```

XMLELEMENT – example

```
SELECT i.id, XMLELEMENT(
        NAME "emp",
        i.name ) AS result
FROM EMPLOYEES i
```

name of the element

content

XMLELEMENT – result

```
SELECT i.id, XMLELEMENT(
        NAME "emp",
        i.name ) AS result
FROM EMPLOYEES i
```

name of the element

content

id	result
emp0001	<emp>John</emp>
emp0002	<emp>Jack</emp>
emp0003	<emp>Donald</emp>
emp0003	<emp>Samuel</emp>

XMLELEMENT

- The content of an element can be built by concatenating more values of SQL
- The concatenation operator is ||

XMLELEMENT

```
SELECT i.id, XMLELEMENT(
        NAME "emp",
        i.name || '  ' || i.surname ) AS result
FROM EMPLOYEES i
```

id	result
emp0001	<emp>John Doe</emp>
emp0002	<emp>Jack Black</emp>
emp0003	<emp>Donald Mason</emp>
emp0003	<emp>Samuel Wood</emp>

XMLATTRIBUTES

- In order to declare a list of attributes, we use the operator XMLATTRIBUTES
- Each parameter of XMLATTRIBUTES is inserted in an attribute which, if not explicitly declared, takes the name of the relational column from which it has been selected

XMLELEMENT and XMLATTRIBUTES

```
SELECT i.id, XMLELEMENT(
        NAME "emp",
        XMLATTRIBUTES(i.salary as "sal"),
        i.name || ' ' || i.surname ) AS result
FROM EMPLOYEES i
```

id	result
emp0001	<emp sal="20000">John Doe</emp>
emp0002	<emp sal="18000">Jack Black</emp>
emp0003	<emp sal="15000">Donald Mason</emp>
emp0003	<emp sal="15000">Samuel Wood</emp>

XMLELEMENT

- In the content of an element it is possible to specify several objects, both elements and strings of characters
- Let’s try to create an element that contains other two elements and some text

XMLELEMENT – example

As we said before, the first parameter specifies the name of the resulting element


```
SELECT i.id,
    XMLELEMENT(
        NAME "emp",
        XMLELEMENT(
            NAME "co",
            i.surname),
        ' of  department ',
        XMLELEMENT(
            NAME "dep",
            i.department)
    ) AS result
FROM EMPLOYEES AS i
```

XMLELEMENT – example

Here we list the contents. We specify an element <co>, then a text content “of department”, then a <dep> element

```
SELECT i.id,
    XMLELEMENT(
        NAME "emp",
        XMLELEMENT(
            NAME "co",
            i.surname),
        ' of  department ',
        XMLELEMENT(
            NAME "dep",
            i.department)
    ) AS result
FROM EMPLOYEES AS i
```

XMLELEMENT – result

id	result
emp0001	<emp><co>Doe</co> of department <dep>Sales</dep></emp>
emp0002	<emp><co>Black</co> of department <dep>Sales</dep></emp>
emp0003	<emp><co>Mason</co> of department <dep>Accounting</dep></emp>
emp0003	<emp><co>Wood</co> of department <dep>Accounting</dep></emp>

XMLFOREST

- XMLFOREST is a quick way to produce a list of simple elements
- The behaviour of its parameters is the same of XMLATTRIBUTES

XMLFOREST – example

```
SELECT i.id,
    XMLELEMENT(
        NAME "emp",
        XMLFOREST(
            i.name,
            i.surname AS "co",
            i.department AS "dip")
    ) AS result
FROM EMPLOYEES i
```

If this explicit name is missing, the name of the corresponding column is used instead

XMLFOREST – result

id	result
emp0001	<emp><name>John</name> <co>Doe</co><dip>Sales</dip></emp>
emp0002	<emp><name>Jack</name> <co>Black</co><dip>Sales</dip></emp>
emp0003	<emp><name>Donald</name><co>Mason </co><dip>Accounting</dip></emp>
emp0003	<emp><name>Samuel</name><co>Wood </co><dip>Accounting</dip></emp>

XMLCONCAT

- XMLCONCAT concatenates its arguments, producing a forest of elements
- XMLCONCAT can concatenate also elements built using XMLELEMENT

XMLCONCAT – example

```
SELECT i.id,
    XMLCONCAT(
        XMLELEMENT(
            NAME "co",
            i.surname),
        XMLELEMENT(
            NAME "dep",
            i.department)
    ) AS result
FROM EMPLOYEES i
```

XMLCONCAT – result

id	result
emp0001	<co>Doe</co> <dep>Sales</dep>
emp0002	<co>Black</co> <dep>Sales</dep>
emp0003	<co>Mason</co> <dep>Accounting</dep>
emp0003	<co>Wood</co> <dep>Accounting</dep>

XMLAGG

- If we need to group more tuples depending on one or more attributes, SQL use the GROUP BY construct
- XMLAGG allows to restore the tuples aggregated using GROUP BY

XMLAGG

id	*nome*	*surname*	*department*	*dismissed*	*salary*
emp0001	John	Doe	Sales	null	20000
emp0002	Jack	Black	Sales	null	18000
emp0003	Donald	Mason	Accounting	null	15000
emp0004	Samuel	Wood	Accounting	01/01/03	15000

```
<dip nome="Sales">
    <emp></emp>
    <emp></emp>
</dip>
```

```
<dip nome="Accounting">
    <emp></emp>
    <emp></emp>
</dip>
```

XMLAGG

```
SELECT XMLELEMENT(
        NAME "department",
        XMLATTRIBUTES(i.department AS "name"),
        XMLAGG(
           XMLELEMENT(
              NAME "emp",
              i.surname) )
     ) AS result
FROM EMPLOYEES i
GROUP BY department
```

XMLAGG

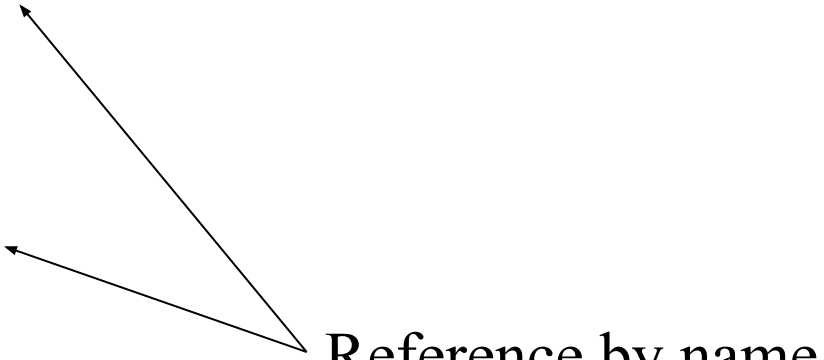
result
<department name="Sales"> <emp>Doe</emp> <emp>Black</emp> </department>
<department name="Accounting"> <emp>Mason</emp> <emp>Wood</emp> </department>

XMLGEN

- Another way to create XML is XMLGEN
- It is possible to directly specify XML code, inserting data through variables, expressed using “{” and “}”
- Variables can be also used for the names of the elements, which was not possible using XMLELEMENT, which instead requires that the names of the elements must be provided explicitly using a constant

XMLGEN

```
SELECT XMLGEN(
        '<employee>
            <name>{$name}</name>
            <salary>{$sal}</salary>
        </employee>',
        i.name,
        i.salary AS "sal"
    ) AS result
FROM EMPLOYEES i
WHERE salary > 15000
```



XMLGEN

result
<employee> <name>Doe</name> <salary>20000</salary> </employee>
<employee> <name>Black</name> <salary>18000</salary> </employee>

Specifications of SQL/XML

- ISO/IEC 9074-14:2003
  - Corresponding to the things presented in these slides
- ISO/IEC 9075-14:2006
  - It adds the possibility to execute queries in XQueries, whose data model is adopted, and to validate XML documents
- ISO/IEC 9075-14:2008
  - It adds some features like the TRUNCATE TABLE statement
- ISO/IEC 9075-14:2011
  - Most recent version with feature for new data types (e.g., temporal)

References

- Some articles on SQL/XML:
  - A description of the standard:
    https://dl.acm.org/doi/10.1145/565117.565141
  - An in-depth view of its new features:
    https://dl.acm.org/doi/10.1145/1031570.1031588
- Some of the systems supporting SQL/XML:
  - PostgreSQL
  - Oracle database server
  - IBM DB2
  - MS SQL Server