University of Vienna, Sommersemester 2015

INTEROPERABILITY
- XML and Databases -

Univ. Prof. Dr. Stefanie Rinderle-Ma
Workflow Systems and Technology
Faculty of Computer Science
University of Vienna
stefanie.rinderle-ma@univie.ac.at

Outline

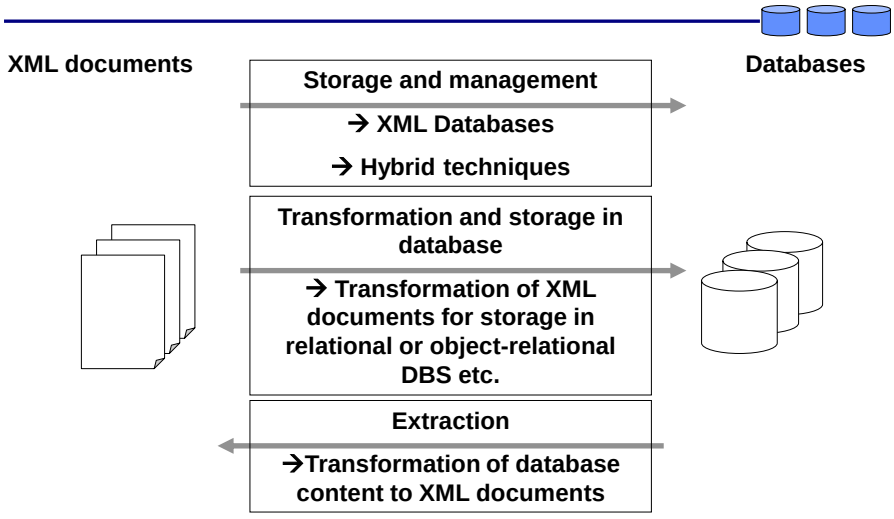
2.1 Motivation

- Databases employed in almost any enterprise
- Massive data exchange over the internet
- Resulting challenge: data stored and maintained within databases should be extracted for exchange
- Example:
  - Relational database for storing course information
  - Contents are to be published as XML documents
  - Exchange documents with other applications
  - Web-based course application

2.1 Motivation

- However, also the other way round is important
  - Huge amount of data is stored within XML documents
  - How to store and maintain theses documents?
- Important: characteristics of the XML document
  - Data-centric
  - Document-centric
  - Hybrid / mixed content
- Requirements
  - Maintaining /restoring the structure
  - Efficient querying
    - Simple selects
    - Joins
    - Within one document
    - Over several documents
    - Restoring documents (or parts of them)

2.1 Motivation



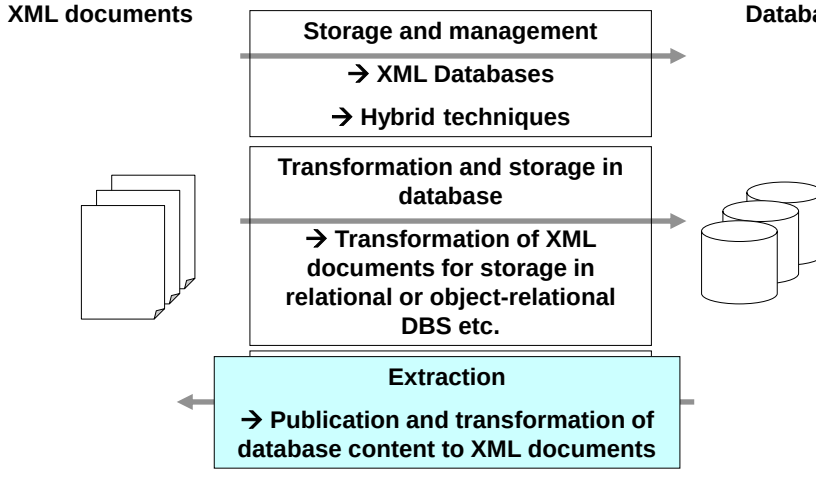
2.1 Motivation

Teaching objectives:

- Requirements and challenges for publishing database content in XML
- Basic publishing concepts (+ tool exercise)
- Introducing SQL/XML standard
- Discussing different techniques for storing XML documents in databases
- Introducing native XML databases
- Discussing different mapping techniques from XML to relational databases

Outline

2.2 Publishing database content in XML



Different questions:

- Which kind of database system?
  - *Relational*: we focus on relational databases due to their pratical importance
  - *Object-relationa*l: basically interesting for XML mappings due to
    - different data types (e.g., lists)
    - user-defined types
    - OO concepts (e.g., inheritance)
- What do we want to publish?
  - Complete databases
  - Query results or views

- How to publish?
  - Access to relational databases via JDBC/ODBC together with SAX or DOM
    - User-defined implementation
    - Introduction of basic concepts
  - XML extensions in databases
  - SQL/XML
    - Offers publishing functions
    - Implemented within the most important commercial systems (e.g. Oracle und DB2)
    - We will use DB2 Express within the exercises
  - XQuery

Relational database content to XML - Basic Considerations -

Relational database: CourseDB

Participant	PNr	Name	Location
	143	Schmidt	Bremen
	145	Huber	Augsburg
	146	Abele	Senden
	149	Kircher	Bochum
	155	Maier	Stuttgart
	171	Möller	Ulm
	173	Schulze	Stuttgart
	177	Mons	Essen
	185	Meier	Heidelberg
	187	Karstens	Hamburg
	194	Gerstner	Ulm

Level 1: Database names

Level 2: Relation names

Level 3: Attribute names

Relational database: CourseDB

Participant	PNr	Name	Location
	143	Schmidt	Bremen
	145	Huber	Augsburg
	146	Abele	Senden
	149	Kircher	Bochum
	155	Maier	Stuttgart
	171	Möller	Ulm
	173	Schulze	Stuttgart
	177	Mons	Essen
	185	Meier	Heidelberg
	187	Karstens	Hamburg
	194	Gerstner	Ulm

Derived XML document

```
<?xml version="1.0" encoding="utf-8"?>
<CourseDB>
        <Participant>
                <PNr>143</PNr>
                <Name>Schmidt</Name>
                <Loc>Bremen</Loc>
        </Participant>
        <Participant>
                <PNr>145</PNr>
                <Name>Huber</Name>
                <Loc>Augsburg</Loc>
        </Participant>
        <Participant>
                <PNr>146</PNr>
                <Name>Abele</Name>
                <Loc>Senden</Loc>
        </Participant>
</CourseDB >
```

E1: Database name as root element

E2: Relation name

E3: Attributes

Resulting XML document is regularly structured (data-centric)

Defining corresponding schema description using XML schema

Using data types of XML schema

- Example:

```
  <xs:element minOccurs="0" maxOccurs="unbounded"
name="Participant">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" name="PNr" type="xs:integer"/>
        <xs:element minOccurs="0" name="Name" type="xs:string" />
        <xs:element minOccurs="0" name="Location" type="xs:string"
/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

2.2 Publishing database content in XML

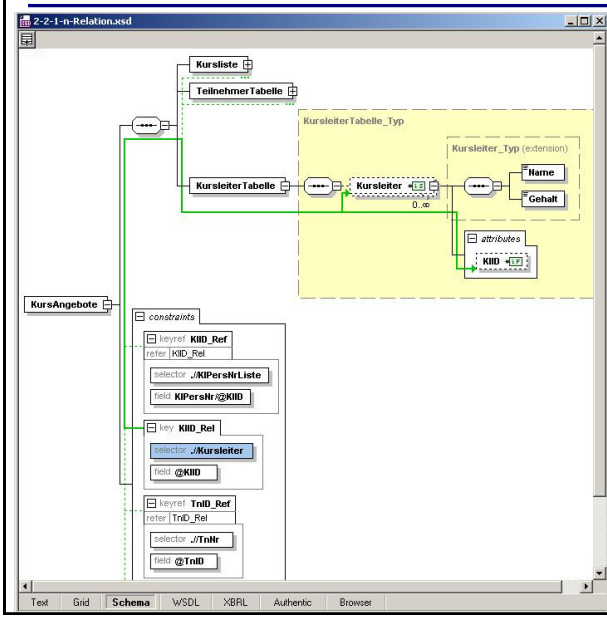
- Preserving integrity constraints in XML schema using XPath
- definition of `keys` in XML schema
- Example:

```
<xs:key name="PNr_key">
  <xs:selector xpath="/CourseDB/Participant"/>
  <xs:field xpath="PNr"/>
</xs:key>
```

- According to `key` definition a table is generated for resolving the subsequent `keyref` definitions
- Example:

```
<xs:keyref name="hat_gebucht_PNr" refer "PNr_key">
  <xs:selector xpath="/CourseDB/Participant"/>
  <xs:field xpath=„@PNr"/>
</xs:keyref>
```


2.2 Publishing database content in XML



ALTOVA XMLSpy (http://www.altova.com/)

SQL/XML Standard

Summary:

- Basic definition in SQL/XML [SQLXML06]
  - Mapping SQL character sets to XML character sets
  - Mapping SQL identifier to XML names
  - Mapping pre-defined SQL data types to XML schema
  - Treating NULL values
- Additional support by publication functions
  - Support the automatic generation of an XML document from database contents
  - Generating XML schema

2.2 Publishing database content in XML

- Publication functions of SQL/XML standard generate XML documents from relation data
- In a first step, the relational data is transformed into a database-internal XML value
- Then the internal XML value can be converted into external formal (→ **serializing**)
- *Remark*: Publication functions are also called **constructor functions**.

In this lecture we use the SQL/XML functions of DB2. However, publication functions are also part of, for example, Oracle [Oracle03]. A comparison of different database systems will be presented at the end of this chapter.

2.2 Publishing database content in XML

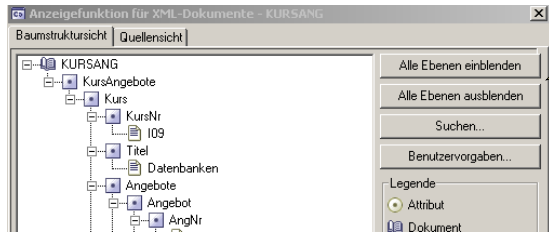
Performance considerations comparing the usage of XML data type and type CLOB later when introducing indices for XML documents.

- New SQL data type *XML*
  - Instead of *CLOB*
  - Can be used for defining columns, variables, and parameters
  - Possible XML values
    - documents
    - Elements
    - Element lists
    - Text nodes
    - Mixed content

```
connect to xmldbl;

CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY, Info XML);

INSERT INTO Customer (Cid, Info) VALUES (1000,
'<customerinfo xmlns="http://posample.org" Cid="1000">
    <name>Kathy Smith</name>
  <addr country="Canada">
        <street>5 Rosewood</street>
        <city>Toronto</city>
    <prov-state>Ontario</prov-state>
        <pcode-zip>M6W 1E6</pcode-zip>
```



Contrary to CLOB, the database „understands“ content of attribute of type XML (e.g., by using XQuery)

Based on [Moos08]

2.2 Publishing database content in XML

Name	Purpose
XMLAGG	Column (set) function: aggregates all values of a column to a single XML value
XMLATTRIBUTES	Generates XML attributes of an XML element based on relational data
XMLCOMMENT	Generates XML comments based on relational data
XMLCONCAT	Concatenates a number of scalar XML values to a single XML value
XMLDOCUMENT	Generates a document root node; an XML document of database-internal XML format must begin with such a node; document is tree-structured

Based on [Moos08]

2.2 Publishing database content in XML

Name	Purpose
XMLELEMENT	Generates an XML element based on relational data
XMLFOREST	Generates a sequence of XML elements based on relational data
XMLNAMESPACES	Generates XML name space declaration
XMLPI	Generates XML processing instruction (for parsers)
XMLSERIALIZE	Converts XML document of internal XML type to external XML docuemnt or BLOB (serializing XML document)
XMLTEXT	Generates XML text value based on relational data

Based on [Moos08]

2.2 Publishing database content in XML

Generally:

- Possible successor nodes of an XML element:
  - Element nodes
  - Text nodes
  - processing instructions
  - Comment nodes
- Possible predecessor nodes of an XML element:
  - Element nodes
  - Document nodes
- Syntactical structure:

```
XMLELEMENT(  name   elementName, elementContentExpression
                        [, elementContentExpression ] …
                    [OPTION {NULL ON NULL | EMPTY ON NULL}])
```


Based on [Moos08]

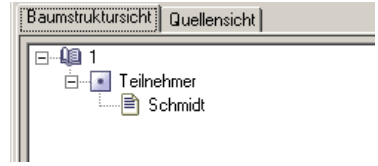
2.2 Publishing database content in XML

Example:

- Generate the following XML document:
  - `<Participantname>Schmidt</Participantname>`
- In XML/SQL (with VALUES as execution environemnt for DB2-SQL):

```
VALUES (
  XMLDOCUMENT(
       XMLELEMENT( NAME "Participant", 'Schmidt')));
```

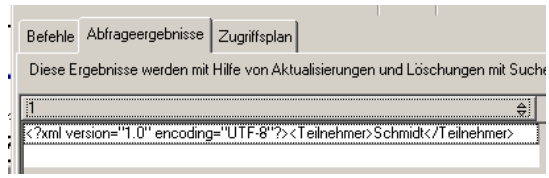
- Remark: instruction `XMLDOCUMENT` must be used to generate root node of XML document
- Generates the following (database-internal) XML values:



2.2 Publishing database content in XML

- To publish the external XML document the database-internal XML values must be serialized using XMLSERIALIZE
- Example:

```
VALUES (XMLSERIALIZE(
    XMLDOCUMENT(
        XMLELEMENT( NAME "Participant", 'Schmidt'))
            AS CLOB VERSION '1.0' INCLUDING
                XMLDECLARATION ));
```



2.2 Publishing database content in XML

Repetition: So far we published complete database tables. However, often it is also desired to publish query results on the database as XML documents.

- Basic idea in SQL/XML: Structure of SQL query result is transformed into corresponding XML document.
- Example:
  - Publish the result of the following SQL query as XML document

```
SELECT * FROM Participant
WHERE Location='Ulm';
```

.

PNr	Name	Location
171	Moeller	Ulm
194	Gerstner	Ulm

Based on [Moos08]

2.2 Publishing database content in XML

```
SELECT XMLSERIALIZE(
            XMLDOCUMENT(
                  XMLELEMENT( NAME "Participant",
                        XMLELEMENT( NAME "PNr", T.PNr
                        ),
                        XMLELEMENT( NAME "Name", T.Name
                        ),
                        XMLELEMENT( NAME "Location", T.Location
                        )
                  )
            )
            AS CLOB VERSION '1.0' INCLUDING XMLDECLARATION)
FROM Participant AS T
WHERE T.Location='Ulm';
```

Columns of result set are transformed into XML elements.

Serializing generates two XML documents → aggregration later

```
'<?xml version=""1.0"" encoding=""UTF-8""?><Teilnehmer><TnNr>171</TnNr><Name>Moeller, H.</Name><Ort>Ulm</Ort></Teilnehmer>"
'<?xml version=""1.0"" encoding=""UTF-8""?><Teilnehmer><TnNr>194</TnNr><Name>Gerstner, M.</Name><Ort>Ulm</Ort></Teilnehmer>"
```

Based on [Moos08]

2.2 Publishing database content in XML

Treating NULL values in the result set:

- Basic possibilities: `NULL ON NULL` and `EMPTY ON NULL`
- Example:

```
SELECT XMLSERIALIZE(
            XMLDOCUMENT(
                  XMLELEMENT( Name "Location", H.Location
                        OPTION EMPTY ON NULL
                  )
            )
            AS CLOB VERSION '1.0' INCLUDING
XMLDECLARATION
  ) AS Result
  FROM Hotel AS H;
```

HotelID	Name	Location
1	Stern	NULL

- Result: `<?xml version="1.0" encoding="UTF-8"?><Location/>`

28

2.2 Publishing database content in XML

Generating multiple XML elements by using `XMLFOREST`

```
XMLFOREST (elementContentExpression [AS elementName]
        [, elementContentExpression [AS elementName]] …
                [OPTION {NULL ON NULL | EMPTY ON NULL}])
```

```
XMLDOCUMENT(
              XMLELEMENT( NAME "Participant",
                    XMLFOREST( T.PNr AS "PNr",
                               T.Name AS "Name",
                               T.Location AS "Location"
                             )…
```

AS optional

- Generates the same XML document as for the expression on slide 35
- However, using `XMLFOREST` mostly results in more compact queries

2.2 Publishing database content in XML

- So far: leaf nodes of the logical XML tree are published as elements
- However, leaf nodes can be also represented as attributes
- In practice, mostly a hybrid approach is used
- Attributes are generated by using `XMLATTRIBUTES()`
- Embedding `XMLATTRIBUTES` into `XMLELEMENT()`:

```
XMLATTRIBUTES( attributeValueExpression [AS attributeName]
        [, attributeValueExpression [AS attributeName]] …
```

```
XMLELEMENT( NAME elementName
      [, XML attributeFunction] …
    [OPTION {NULL ON NULL | EMPTY ON NULL}])
```

Based on [Moos08]

2.2 Publishing database content in XML

The example on slide 35 with attribute-centered represenation:

```
SELECT XMLSERIALIZE(
            XMLDOCUMENT(
                  XMLELEMENT( NAME "Participant",
                        XMLATTRIBUTES(
                        T.PNr,
                        T.Name,
                        T.Location
                        )
                  )
            )
            AS CLOB VERSION '1.0' INCLUDING XMLDECLARATION)
FROM Participant AS T
WHERE T.Location='Ulm';

<Participant PNR="194" NAME="Gerstner, M." Location="Ulm"/>…
```


Based on [Moos08]

2.2 Publishing database content in XML

Example of slide 35 with hybrid representation:

```
SELECT XMLSERIALIZE(
            XMLDOCUMENT(
                  XMLELEMENT( NAME "Participant",
                        XMLATTRIBUTES(
                              T.PNr),
                        XMLFOREST (
                              T.Name,
                              T.Location
                        )
                  )
            )
            AS CLOB VERSION '1.0' INCLUDING XMLDECLARATION)
FROM Participant AS T
WHERE T.Location='Ulm';

<Participant PNR=""171""><NAME>Moeller, H.</NAME>
                              <Location>Ulm</Location></Participant>
```

Based on [Moos08]

2.2 Publishing database content in XML

- `XMLCOMMENT ( comment text )` inserts a comment into the resulting XML document
- `XMLTEXT ( Text)` inserts a text into the resulting XML document → important when generating document-centered or hybrid XML documents
- Example:

```
XMLDOCUMENT(
                XMLELEMENT( NAME "Participant",
                XMLELEMENT( NAME "PNr", T.PNr),
                XMLELEMENT( NAME "Name", XMLTEXT (,The
                      Participant is called:' || T.Name)),
                XMLELEMENT( NAME "Location", T.Location
                      )…

<Participant><PNr>171</PNr><Name>The Participant is called:
Moeller, H.</Name><Location>Ulm</Location></Participant>
```


Based on [Moos08]

2.2 Publishing database content in XML

- `XMLNAMESPACES` generates XML name spaces

```
XMLNAMESPACES( nameSpace-URI AS nameSpace-prefix
                [nameSpace-URI AS nameSpace-prefix]
                  [[,]{DEFAULT nameSpace-URI | NO
                    DEFAULT}])
```

```
...
XMLELEMENT( NAME "Participant",
        XMLNAMESPACES(
              'http://www.CourseOnline.de/Participant' AS
              CourseDB),
…
```

2.2 Publishing database content in XML

- Using `XMLCONCAT` several values of type XML can be concatenated to one XML value
- In other words: the values of several XML attributes (in the database) are concatenaed within one XML element in the XML result document

`XMLCONCAT( XMLExpression [, XMLExpression ] … )`

- Example on the next slide
- If necessary, `XMLCAST` can be used for type casts
- Example:

```
XMLCAST('<E>hi</E>' AS XML)
```


2.2 Publishing database content in XML

```
CREATE TABLE Info(S1 XML, S2 XML);
INSERT INTO Info VALUES(
            '<Element_1>XMLand</Element_1>',
            '<Element_2>databases</Element_2>'
            );
SELECT XMLSERIALIZE(
            XMLDOCUMENT(
                  XMLELEMENT( NAME "Info",
                        XMLCONCAT( S1, S2)
                  )
            )
      AS CLOB VERSION '1.0' INCLUDING XMLDECLARATION)
FROM Info;
```

Info	S1	S2
	XMLand	databases

Result:

```
<Info>
            <Element_1>XMLand</Element_1>
            < Element_2>databases </Element_2>
</Info>
```

Based on [Moos08]

2.2 Publishing database content in XML

- XMLCONCAT concatenates entries of several columns
- XMLAGG aggregates entries of one column to one XML value
- Syntactical structures as for XMLCONCAT
- Example:

```
CREATE TABLE Info(S1 XML);
INSERT INTO Info VALUES('<Element_1>A</Element_1>');
INSERT INTO Info VALUES('<Element_1>B</Element_1>');
INSERT INTO Info VALUES('<Element_2>A</Element_2>');
SELECT XMLSERIALIZE(
  XMLDOCUMENT(
       XMLELEMENT( NAME "Info",
            XMLAGG( S1))…
FROM Info;
```

Result:

```
<Info>
      <Element_1>A</Element_1>
      <Element_1>B</Element_1>
      <Element_2>A</Element_2>
</Info>
```

37

Based on [Moos08]

2.2 Publishing database content in XML

Using XMLAGG one resulting XML document can be generated:

```
SELECT XMLSERIALIZE(
     XMLDOCUMENT(
          XMLELEMENT( NAME "All_Participants",
               XMLAGG(
                    XMLELEMENT( NAME "Participant",
                         XMLFOREST(T.PNr,
                                   T.Name,
                                   T.Location
)…
```

```
<All_Participants>
   <Participant>
      <PNR>171</PNR>
      <NAME>Moeller, H.</NAME>
      <Location>Ulm</Location>
   </Participant>
   <Participant>
      <PNR>194</PNR>
      <NAME>Gerstner, M.</NAME>
      <Location>Ulm</Location>
   </Participant>
</All_Participants>"
```

38

Exercise: Write the SQL/XML expression in DB2 which takes the information from table *Info* and creates the following XML document (coplumns S1 and S2 are defined as XML type) :

Info	S1	S2
	Web and	databases
	XML and	databases

```
<Info>
      <S1>Web and</S1>
      <S2>databases</S2>
      <S1>XML and</S1>
      <S2>databases</S2>
</Info>
```

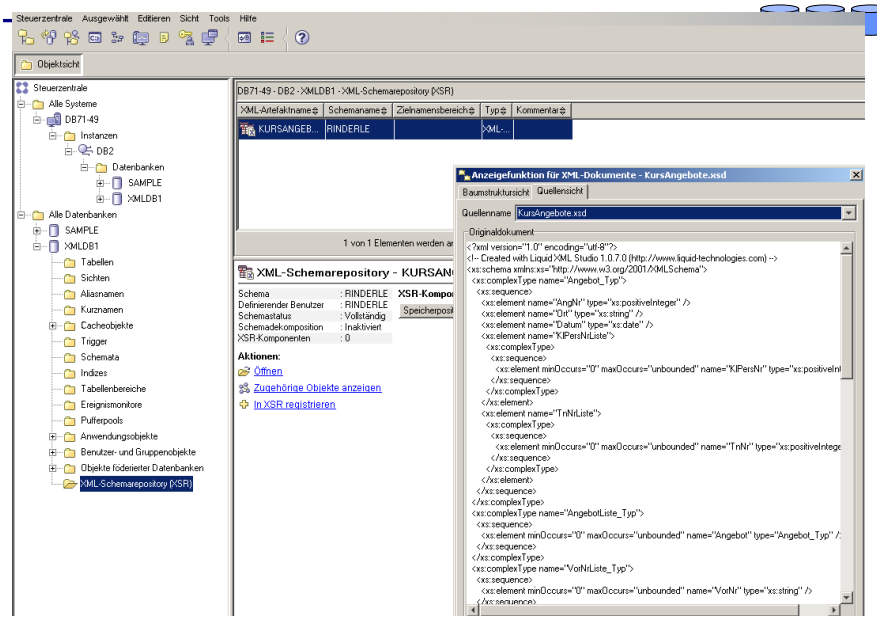

Wrap up:

- Using SQL/XML functions, relational content can be published as XML documents
- Using SELECT * FROM… queries, tables can be published (complete publication).
- Different tables can be joined using SQL and result sets can be publihsed as XML documents
- Question: how can we derive the associated schema information from the resulting XML documents?
- Manual generation of schema information
- Alternatives?

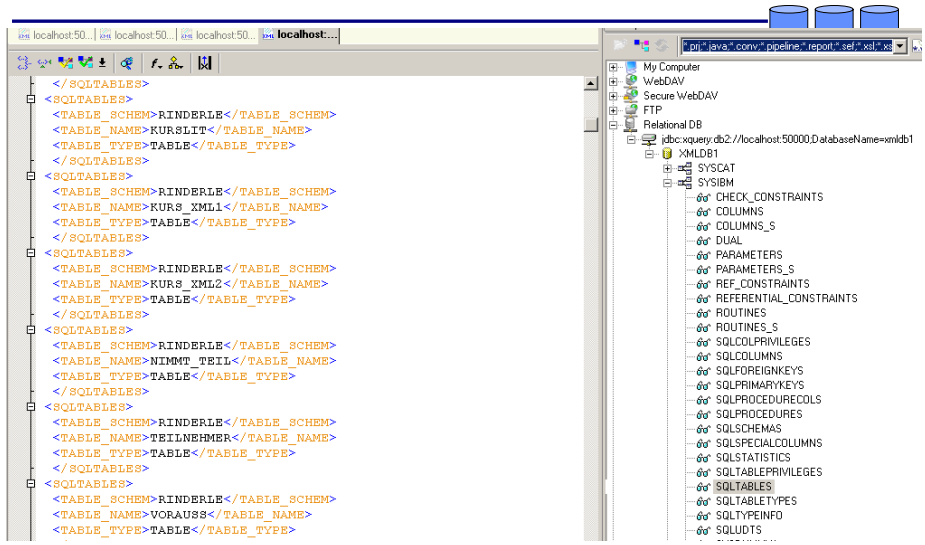
2.2 Publishing database content in XML

- As discussed in [Bourret], the generation of XML documents is a design time task, i.e., when publishing database contents as XML, schema information should be already available.
- If generating XML documents from database contents is mostly of ad-hoc nature or randomly, a better solution could be to use a native XML database.
- Nevertheless commercial systems offer automatic derivation techniques:
  - DB2: XML schemas can be registered within the XML repository (can be also used for later validation).
  - In StylusStudio and Altova MapForce different information on database tables and database statistics can be transformed into XML. This might yield valuable met information for the documentation process and the generation of XML schema.

2.2 Publishing database content in XML



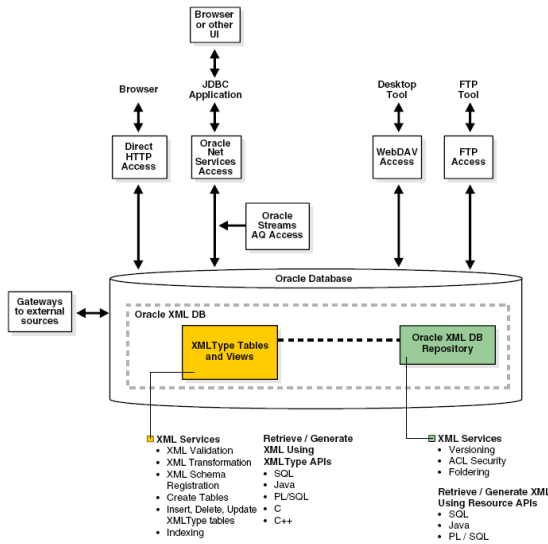
2.2 Publishing database content in XML



2.2 Publishing database content in XML

XML Support in Other Database Systems

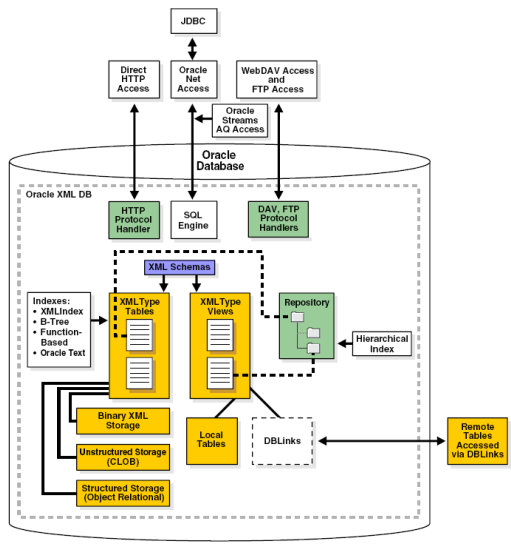
2.2 Publishing database content in XML



source: Oracle XML DB Developer's Guide 11g Release 1 (11.1) October 2007

ORACLE 11g

2.2 Publishing database content in XML



Bildquelle: Oracle XML DB Developer's Guide 11g Release 1 (11.1) October 2007

ORACLE 11g

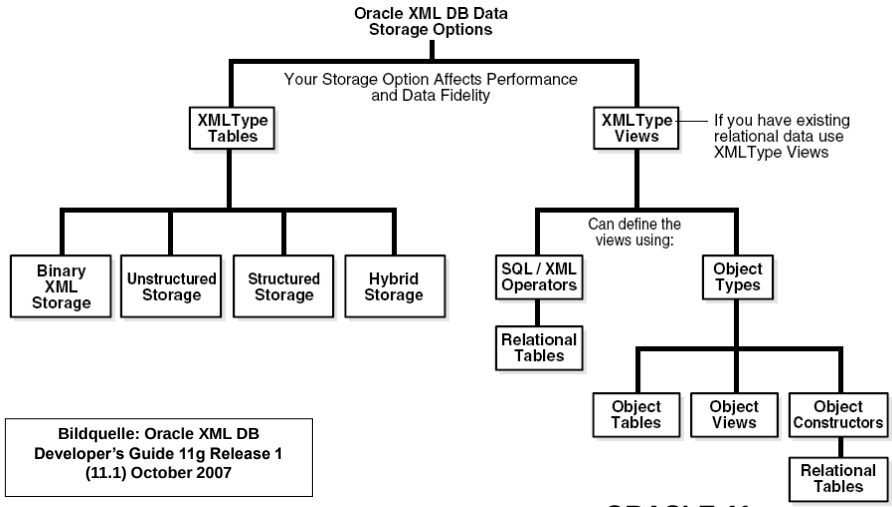
2.2 Publishing database content in XML

Support of XML in Oracle

- Data type `XMLType`: native data type for XML data
  - Enables validation against XML Schema
  - Can be used as data type for columns
  - `XMLType` is also object type → can be used to define tables of type `XMLType`
  - Tables can be validated against XML schema
- The following operations are defined on data of type `XMLType`:
  - `extract()`: extracts subset of an XML instance
  - `existsNode()`
  - `schemaValidate()`
  - `transform()`: transforms the content of an XML instance through XSL

ORACLE 11g

2.2 Publishing database content in XML



Bildquelle: Oracle XML DB Developer's Guide 11g Release 1 (11.1) October 2007

ORACLE 11g

Create tables and columns of type XML:

- column:

```
CREATE TABLE mytable1
(key_column VARCHAR2(10) PRIMARY KEY, xml_column
  XMLType);
```

- table:

```
CREATE TABLE mytable2 OF XMLType;
```

- Diverse Möglichkeiten zum Laden von XML-Daten (z.B. auch mittels Massenlader SQL*Loader)

ORACLE 11g

```
<PurchaseOrder>
        <Reference>SBELL-20021009123336O1PDT</Reference>
        <Actions><Action><User>SVOLLMAN</User></Action></Actions>
        <Reject/>
        <Requestor>Sarah J. Bell</Requestor>
        <User>SBELL</User>
        <CostCenter>S30</CostCenter>
        <ShippingInstructions>
                <name>Sarah J. Bell</name>
                <address>400 Oracle Parkway Redwood Shores CA 94065 USA</address>
                <telephone>650 506 7400</telephone>
        </ShippingInstructions>
        <SpecialInstructions>Air Mail</SpecialInstructions>
        <LineItems>
                <LineItem ItemNumber="1">
                        <Description>A Night to Remember</Description>
                        <Part Id="715515009058" UnitPrice="39.95" Quantity="2"/>
                </LineItem>
                <LineItem ItemNumber="2">
                        <Description>The Unbearable Lightness Of Being</Description>
                        <Part Id="37429140222" UnitPrice="29.95" Quantity="2"/>
                </LineItem>
        </LineItems>
</PurchaseOrder>
```

Beispieldokument aus [Oracle07]

ORACLE 11g

2.2 Publishing database content in XML

In Anlehnung an [Oracle07]

Queries in Oracle 11g:

- Useage of Xpath expression in `extract()` function, example:

```
SELECT extract(OBJECT_VALUE, '/PurchaseOrder/Reference')
FROM purchaseorder;
```

- yields

```
<Reference>SBELL-2002100912333601PDT</Reference>
```

- query

```
SELECT extract(OBJECT_VALUE,
'/PurchaseOrder/LineItems/LineItem[1]')
FROM purchaseorder;
```

- yields

```
<LineItem ItemNumber="1">
<Description>A Night to Remember</Description>
<Part Id="715515009058" UnitPrice="39.95" Quantity="2"/>
</LineItem>
```

ORACLE 11g

2.2 Publishing database content in XML

In Anlehnung an [Oracle07]

Relational „access“ on XML data by creating views, example:

```
CREATE OR REPLACE VIEW
  purchaseorder_master_view(reference, requestor, userid,
  costcenter,ship_to_name, ship_to_address, ship_to_phone,
  instructions)
AS SELECT
  extractValue(OBJECT_VALUE, '/PurchaseOrder/Reference'),
  extractValue(OBJECT_VALUE, '/PurchaseOrder/Requestor'),
  extractValue(OBJECT_VALUE, '/PurchaseOrder/User'),
  extractValue(OBJECT_VALUE, '/PurchaseOrder/CostCenter'),
  extractValue(OBJECT_VALUE,
        '/PurchaseOrder/ShippingInstructions/name'),
  extractValue(OBJECT_VALUE,
        '/PurchaseOrder/ShippingInstructions/address'),
  extractValue(OBJECT_VALUE,
        '/PurchaseOrder/ShippingInstructions/telephone'),
  extractValue(OBJECT_VALUE,
        '/PurchaseOrder/SpecialInstructions')
FROM purchaseorder;
```

ORACLE 11g

Support of the following SQL/XML publication functions:

- `XMLElement`: creates XML element
- `XMLAttributes`: adds attribute to element
- `XMLForest`: creates forest of elements
- `XMLAgg`: creates an element of a collection of elements

Further possibilites:

- Indices
- Specific functions on data type `XMLType`
- Good introduction [Oracle07]
- Also: Hands-on!

ORACLE 11g

In Anlehnung an [SQLServer09]

Basically:

- Supporting XML data type
  - example: `CREATE TABLE T1(Col1 int primary key, Col2 xml)`
- Xquery on columns of type XML, examples:
  - *exist (XQuery)*
  - *query ('XQuery')*
  - *modify (XML_DML)*
  - *nodes (XQuery) as Table(Column)*
  - *value (XQuery, SQLType*)
- Xquery extension functions in SQL Server
  - *sql:column()*
  - *sql:variable()*

  As in DB2: relational Non-XML-Data is included in XML

SQL Server

2.2 Publishing database content in XML

In Anlehnung an [SQLServer09]

- Support of shredding (mapping of XML data on relational tables) by *OPENXML*, example:

```
SELECT * FROM OPENXML (@XmlDocumentHandle,
'/ROOT/Customer/Order/OrderDetail',2)
WITH (   OrderID int '../@OrderID',
         CustomerID varchar(10) '../@CustomerID',
         OrderDate datetime '../@OrderDate',
         ProdID int '@ProductID', Qty int '@Quantity')
```

```
<Customer CustomerID="VINET" ContactName="Paul Henriot">
         <Order OrderID="10248" CustomerID="VINET" EmployeeID="5"
OrderDate="1996-07-04T00:00:00">
                  <OrderDetail ProductID="11" Quantity="12"/>
                  <OrderDetail ProductID="42" Quantity="10"/>
         </Order> </Customer>
<Customer CustomerID="LILAS" ContactName="Carlos Gonzlez">
         <Order OrderID="10283" CustomerID="LILAS" EmployeeID="3"
OrderDate="1996-08-16T00:00:00">
                  <OrderDetail ProductID="72" Quantity="3"/>
          </Order>
</Customer>
```


Example document taken from [SQLServer09]

2.2 Publishing database content in XML

- Mapping of database content on XML:
  - Statement *FOR XML* together with *RAW|AUTO|EXPLICIT|PATH*
- Possibility to create different indices (value, path, full text)
- Update possibilities by using XML DML (insert, delete, update)

Publishing object-relational data in XML

- **Object relational databases support non-atomic, complex columns [TuSa06]**
  - tuples
  - collections
  - objects
  - references
- **Existing structure should be transferrred to the XML document**
  - Instance level
  - Schema level

list **set**

HotelID	Name	<Adress>				{Phone}
		ZipCode	Location	Street	Number	
H1	Stern	89075	Ulm	Hirschstrasse	4	{07310000, 07310001}

tuple

Based on [Klet08]

First challenge: adequate mapping of complex attributes (tuples, sets, lists) at instance level

Im Beispiel:

```
create row type Adresse_t
( ZipCode    INTEGER,
  Location   VARCHAR(25),
  Street     VARCHAR(20),
  Nr         INTEGER
);
create table Hotel
( HotelID INTEGER  NOT NULL,
  Name     VARCHAR(20) NOT NULL,
  Adresse  Adresse_t,
  Phone  SET(INTEGER NOT NULL)
);
```

```
<HotelInformation>
    <Hotel>
        <HotelID>H1</HotelID>
        <Name>Stern</Name>
        <Adress>
            <ZipCode>89075</ZipCode>
            <Location>Ulm</Location>
            <Strasse>Hirschstrasse</Strasse>
            <Nr>4</Nr>
        </Adress>
        <Phone>07310000</Phone>
        <Phone>07310001</Phone>
    </Hotel>
</HotelInformation>
```

HotelID	Name	<Adress>				{Phone}
		ZipCode	Location	Street	Number	
H1	Stern	89075	Ulm	Hirschstrasse	4	{07310000, 07310001}

Based on [Klet08]

Second challenge: adequate mapping of complex attributes (tuples, sets, lists) at schema level

Example:

```
create row type Adress_t
( ZipCode    INTEGER,
  Location   VARCHAR(25),
  Street     VARCHAR(20),
  Nr         INTEGER
);
create table Hotel
( HotelID    INTEGER  NOT NULL,
  Name       VARCHAR(20) NOT NULL,
  Adress     Adress_t,
  Phone      SET(INTEGER NOT NULL)
);
```

```
<!ELEMENT HotelInformation (Hotel*)>
<!ELEMENT Hotel (HotelID, Name,
                 Adress, Phone+)>
<!ELEMENT HotelID (#PCDATA)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Adress (ZipCode, Location,
                  Street, Nr)>
<!ELEMENT ZipCode (#PCDATA)>
<!ELEMENT Location (#PCDATA)>
<!ELEMENT Street (#PCDATA)>
<!ELEMENT Nr (#PCDATA)>
<!ELEMENT Phone (#PCDATA)>
```

DTD

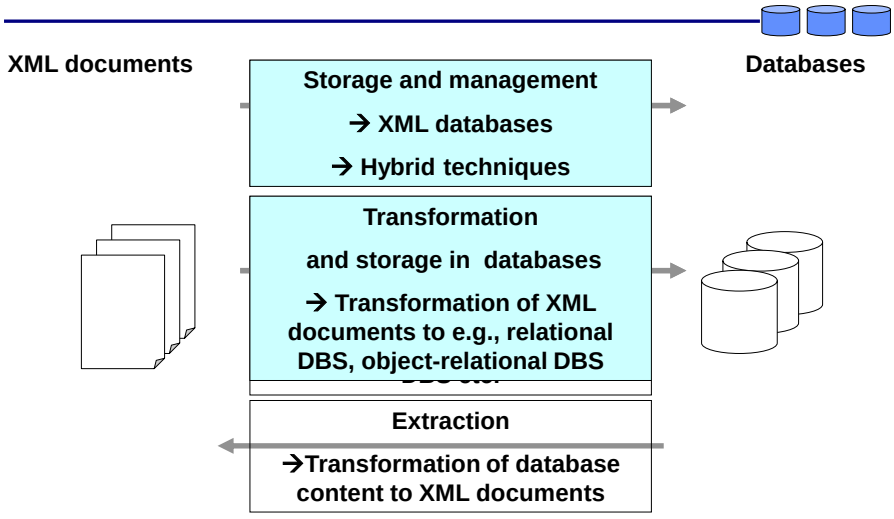
HotelID	Name	<Adress>				{Phone}
		ZipCode	Location	Street	Number	
H1	Stern	89075	Ulm	Hirschstrasse	4	{07310000, 07310001}

2.2 Publishing database content in XML

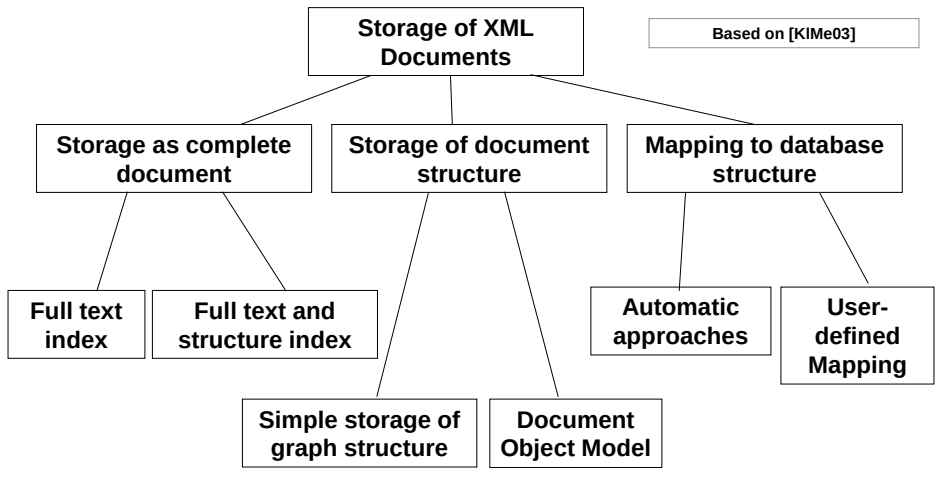
- Mapping of database content to XML documents is practially relevant
  - Web-based publishing
  - Exchange of content in XML format
- Different techniques
  - Access to relational data via JDBC / ODBC together with XML parsers SAX or DOM
  - XML extensions in databases
  - SQL/XML
  - XQuery
- Focus is on relational and object-relational databases
- Better support when using XQuery

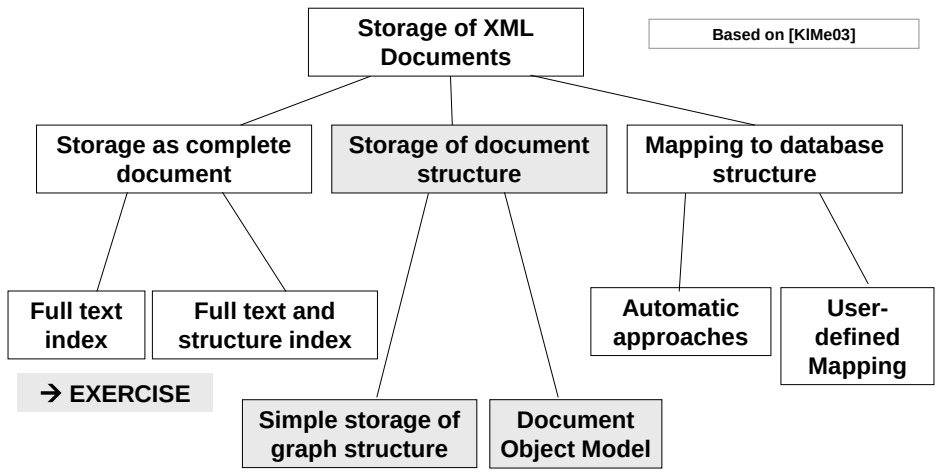
Outline

2.3 Storage of XML documents in databases



2.3 Storage of XML documents in databases





Simple storage of graph structure – basic idea [Brad98]:

- Storing graph structure of XML documents in relations
- Relations: elements with their tags and values
- Each document has a unique document identifier
- Each element has an identifier
- Mapping of element hierarchy of XML document by storing identifier of predecessor node
- Additional (artificial) attribute ChildNo sibling relation between elements can be expressed

Based on [KlMe03]

2.3 Storage of XML documents in databases

Based on [KIMe03]

DocID	Element name	ID	Predecessor	ChildNo	Value
h001	hotel	101		1	
h001	hotelname	102	101	1	Hotel Stern
h001	adress	103	101	2	
h001	zip	104	103	1	89073
h001	loc	105	103	2	Ulm
h001	street	106	103	3	Hirschstrasse
h001	desc.	108	101	3	From Stuttgart…
h002	hotel	201		1	
h002	hotelname	202	201	1	Sonne

```
<hotel>
    <hotelname>Hotel Stern</hotelname>
    <adress>
        <zip>89073</zip>
        <loc>Ulm</loc>
        <street>Hirschstrasse</street>
    </adress>
    <desc>
        From Stuttgart by train…
    </desc>
</hotel>
```

```
<hotel>
    <hotelname>Sonne</hotelname>
    <adress>
        <zip>89075</zip>
        <loc>Ulm</loct>
        <street>Baumstrasse</street>
    </adress>
    <desc>
          From Stuttgart by train…
    </desc>
</hotel>
```

2.3 Storage of XML documents in databases

EXERCISE:

Take the table on the slide before. Provide the SQL query which selects all names of hotels in Ulm.

2.3 Storage of XML documents in databases

Based on [KIMe03]

Dealing with attributes

DocID	Attribute name	ElementID	Value
h001	state	105	california
h001	city	106	LA
h001	url	108	http://www...
h002	…	…	…

2.3 Storage of XML documents in databases

Based on [KIMe03]

Extensions:

- Maintaining seperate relations for values of elements and attributes
- Advantage: for each data type a separate relation can be created
- This enables, for example, sorting of values

DocID	Elementname	ID	predecessor	ChildNo	Value
h001	hotel	101		1	
h001	hotelname	102	101	1	string01
h001	adresse	103	101	2	
h001	zip	104	103	1	int01
h001	ort	105	103	2	string02
h001	strasse	106	103	3	string03

Table of string values

RefID	Value
string01	Hotel Stern
string02	Ulm
string03	Hirschstrasse

Table of integer values

RefID	Wert
int01	89073

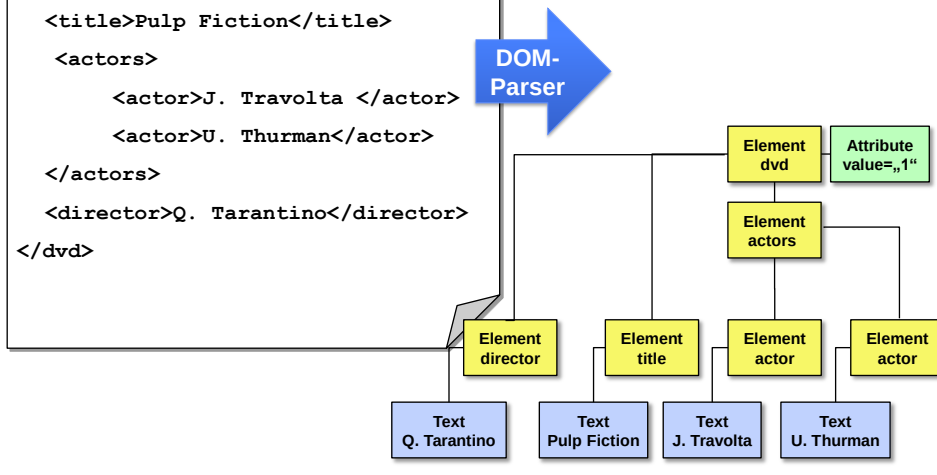
Based on [KIMe03]

Evaluating simple storage of graph structure:

- No schema necessary (+)
- Only few relations with fixed structure necessary (+)
- Enables storing documents with irregular structure (+)
- Queries for more than one element or attribute become very expensive (self joins!)
- Reconstructing the whole document is possible, but very expensive
- SQL queries can be used
- Query rewriting enables the use of XQuery as well

Repetition Document Object Model (DOM)

```
<dvd value="1">
  <title>Pulp Fiction</title>
   <actors>
        <actor>J. Travolta </actor>
        <actor>U. Thurman</actor>
  </actors>
  <director>Q. Tarantino</director>
</dvd>
```



Based on [KIMe03]

Mapping DOM structure to (relational) database tables:

„structure table“

node_id	node_type	doc_id	parent	p_sib	n_sib
01	Element	d01	NULL	NULL	NULL
02	Element	d01	01	04	NULL
03	Element	d01	01	NULL	04
04	Element	d01	01	03	02
05	Element	d01	02	NULL	06
06	Element	d01	02	05	NULL
07	Attribute	d01	01	NULL	NULL

Element table

node_id	tag_name	text
01	dvd	
02	actors	
03	director	Q. Tarantino
04	title	Pulp Fiction
05	actor	J. Travolta
06	actor	U. Thurman

Attribute table

node_id	attr_name	attr_value
07	value	1

Based on [KIMe03]

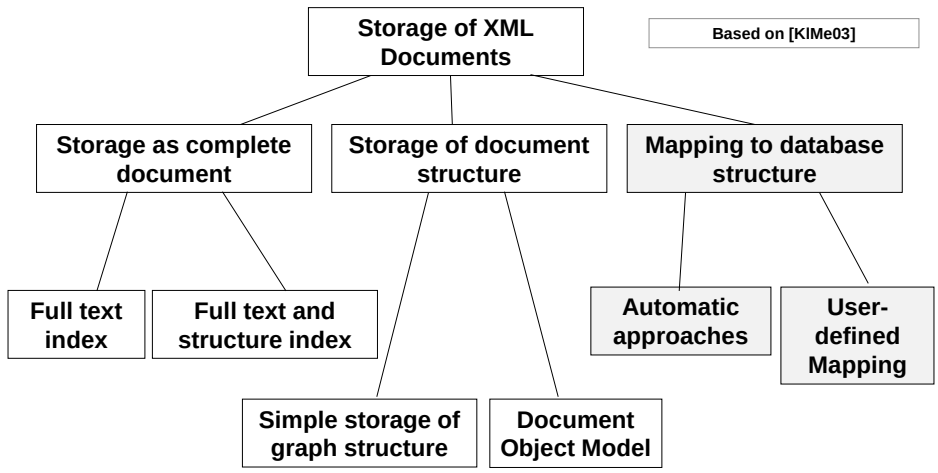
- Queries can be formulated using DOM methods
  - Methods of class *Node* for structural table
  - Methods of class *Element* for element table
  - Methods of class *Attribute* for attribute table
- Queries using XPath or XQuery possible (query rewriting or already supported by the database system, e.g., XQuery in DB2)
- Due to relational storage, we can also use SQL queries

Based on [KIMe03]

Evaluation of storage using DOM:

- No schema necessary (+)
- DOM information persistent (+)
- Updates by using DOM methods (+)
- Reconstructing document structure is possible, but expensive (strongly fragmented storage) (-)
- It is possible to use different query languages (+)
  - SQL
  - DOM methods
  - XQuery / XPath
- Relational storage of graph structure of XML documents is also called **model-based native storage**.

Based on [KIMe03]



Automatic approaches:

- No generic database structures, but specifically designed database structures for each XML document
- Kind and type of document is often defined by schema description (DTD or XML schema)
- Thus: derivation of database structures based on schema description

Based on [KIMe03]

Rules for deriving database schema from a DTD

- Elements
  - XML elements → Attribute of a relation
  - Sequence of elements → Attribute of a relation
  - Choice of elements → Attribute of a relation
  - Elements with quantification? → Attribute, NULL poss.
  - Elements with quantification + or * → SET or LIST (objectrel.)
  - Complex-structured element → ROW (objectrel.)
- Attribute
  - XML attribut → Attribute of a relation
    #IMPLIED → NULL allowed
  - #REQUIRED → NULL not allowed
  - Default value → Default value

Based on [KlMe03]

Challenges of automatic approaches:

- Recursion (possible within schema description)
- Examle ([KlMe03]):

```
<!ELEMENT loc (name, trip*)>
<!ELEMENT trip (loc, distance)>
```

- Here, the recursion has to be resolved by modeling `loc` not as attribute of relation `trip`, but as explicit reference to separate relation `loc`.

Based on [KlMe03]

Challenges (continued)

- Mixed Content:
- Example:

```
<Description>
  You can reach our hotel from different directions:
      <train>per train: ca. 300 m from main train station </train>
      <car>by car: highway, exit 5 </car>
      <plane>by plane: 80 km from Stuttgart </plane>
      You can find us right in the city center
</Description>
```

ID	PCDATA	train	car	plane
1	You can …			
2		train: ca…		
3			By car: …	
4				By plane …
5	You can find…			

Based on [KlMe03]

Challenges (continued)

- ANY in DTD: cannot be represented
- Document order is not preserved → Introducing additional database attribute
- Mapping of alternatives from DTD to relational database:
  - Exampe in DTD:

```
<!ELEMENT publication (book|journal|proceeding)*>
```

  - Which alternative are conceivable?

Example document:

```
<publication>
  <book>
    <booktitle>XML and databases
</booktitle>
    <publisher>dpunkt</publisher>
  </book>
  <journal>
    <title>Process Mining</title>
    <journal>DKE</journal>
  </journal>
  <proceeding>
    <title>Process Patterns</title
    <conference>BPM</conference>
  </proceeding>
</publication>
```

2.3 Storage of XML documents in databases

Example document:

```
<publication>
  <book>
    <booktitle>XML and databases
</booktitle>
    <publisher>dpunkt</publisher>
  </book>
  <journal>
    <title>Process Mining</title>
    <journal>DKE</journal>
  </journal>
  <proceeding>
    <title>Process Patterns</title
    <conference>BPM</conference>
  </proceeding>
</publication>
```


2.3 Storage of XML documents in databases

Example document::

```
<publication>
  <book>
    <booktitle>XML and databases
</booktitle>
    <publisher>dpunkt</publisher>
  </book>
  <journal>
    <title>Process Mining</title>
    <journal>DKE</journal>
  </journal>
  <proceeding>
    <title>Process Patterns</title
    <conference>BPM</conference>
  </proceeding>
</publication>
```

Evaluation of automatic approaches:

- Functionality of SQL can be used (+)
  - Joins
  - Aggregation
  - Query optimization
- XPath, XQuery can be used (+)
- Schema description necessary for design (-)
  - However, there are some approaches in literature which use data mining instead fo requiring a schema, e.g., STORED [DFS99])
- Database structures are variant (changes of the XML document affect the table structures)
- Reconstructing document only partly possible (-)
- Altogether: applicable for data-centric, stable XML documents

User-defined mappings:

- Also called **shredding**
- Example for mapping tool XML-DBMS (http://www.rpbourret.com/xmldbms/index.htm) by Ronald Bourret:

```
<Orders>
  <SalesOrder SONumber="12345">
  <Customer CustNumber="543">
     <CustName>ABC Industries</CustName>
     <Street>123 Main St.</Street>
     <City>Chicago</City>
     …
  </Customer>
   <OrderDate>981215</OrderDate>
   …
  </SalesOrder>
</Orders>
```

Mapping of classes (element types) onto tables:

```
<ClassMap>
  <ElementType Name="SalesOrder"/>
  <ToClassTable>
    <Table Name="Sales"/>
  </ToClassTable>
  ...property maps...
  ...related class maps...
</ClassMap>
```

2.3 Storage of XML documents in databases

- Example for mapping tool XML-DBMS (http://www.rpbourret.com/xmldbms/index.htm) by Ronald Bourret:

```
<Orders>
  <SalesOrder SONumber="12345">
  <Customer CustNumber="543">
     <CustName>ABC Industries</CustName>
     <Street>123 Main St.</Street>
     <City>Chicago</City>
     …
  </Customer>
   <OrderDate>981215</OrderDate>
   …
  </SalesOrder>
</Orders>
```

Mapping properties (attributes and element types) to columns:

```
<PropertyMap>
  <Attribute Name="SONumber"/>
  <ToColumn>
    <Column Name="Number"/>
  </ToColumn>
</PropertyMap>
<PropertyMap>
  <ElementType Name="OrderDate"/>
  <ToColumn>
  <Column Name="Date"/>
   </ToColumn>
</PropertyMap>
```


2.3 Storage of XML documents in databases

- Additionally:
  - Mapping of element hierarchy (referential integrity)
  - Eliminating of unnecessary root elements
  - Mapping of „Mixed Content“
  - Data transfer (XML→ DB, DB → XML)
- Evaluation:
  - Database query languages (+)
  - Flexible mappings (+)
  - Schema description necessary (-)
  - In case of schema changes mapping rules have to be changed as well (-)
  - Mapping rules are potentially errorneous (-)

Reading Exercise 2:

Read paper „Efficiently publishing relational data as XML documents“ [Shan+01] and answer the following questions:

a) How do the described approaches relate to the approaches presented in Florescu &Kossmann [FIKo99] (overall picture)?

b) Discuss the different approaches presented in the paper.

Redundant storage

- Corresponds to a combination of the previously introduced techniques.
- Example document reconstruction:
  - Storage of document structure can be used for partial queries
  - Additionally: redundant storage of XML documents (possibly using index) for document reconstruction
- However: increased storage space
- → depending on application scenario

2.3 Storage of XML documents in databases

Different possibilities for storing XML documents:

- Storage of entire documents
  - Full text index
  - Full text index and structural index
- Storage of document structure
  - Generic database structures
  - Or along graph structure of documents (e.g., DOM)
- Mapping on database structures
  - Specific database structure for single document types
  - Schema description usually necessary
- Not „black&white“: there is no „BEST“ solution
- Depending on application scenario (e.g., on kind of XML document, query profiles, change frequencies)

Outline

What are native XML databases (NXD)?

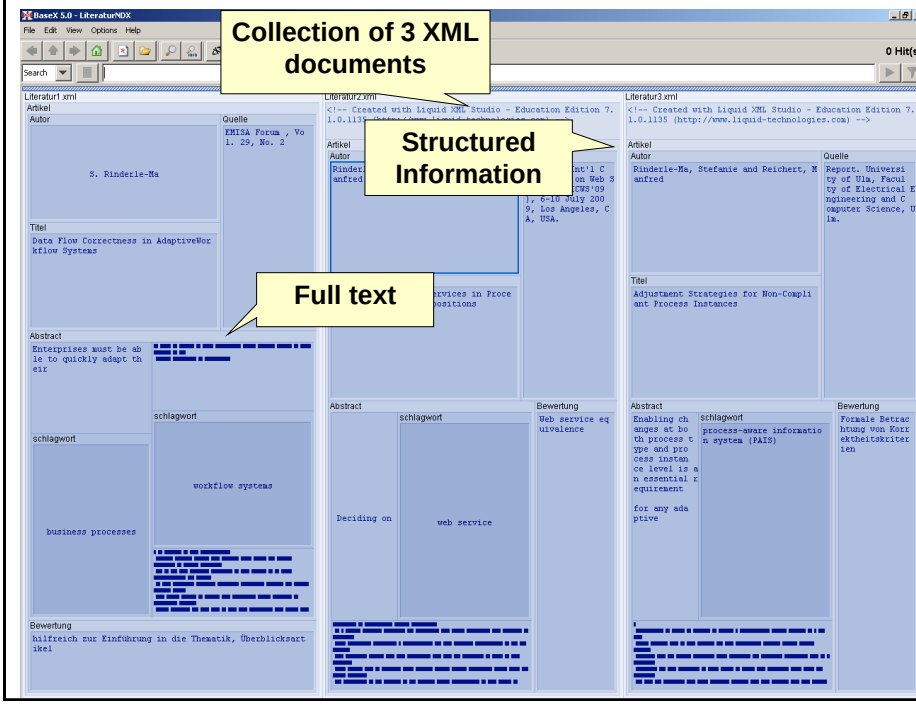
- Trying a definition [Bourret09,Staken01]:
  - Definition of a logical model for an XML document
  - NXD enable in accordance with logical model storing and retrieving documents
  - Model must contain at least elements, attributes, #PCDATA , and document order (e.g., Xpath data model, DOM, SAX)
  - Fundamental logical unit is the XML dokument (cf. tuple in relational databases)
  - Collection: set of related XML documents (cf. Tables as set of tuples in relational databases)
  - Basic physical model can vary:
    - Proprietary storage mechanism (e.g., indices, compressed files)
    - Based on databases

Why NDX?

- Managing a collection of XML documents
- Advantages:
  - Simplifying management
  - Performance of query processing
  - Distributed access
  - Security
  - Transactional concepts
- Typical application scenarios:
  - Storing and querying document-centered and semi-structured documents
  - Data integration
  - For data-centered documents probably mapping XML documents on relational databases (full SQL functionality, indices, query optimization, etc.)

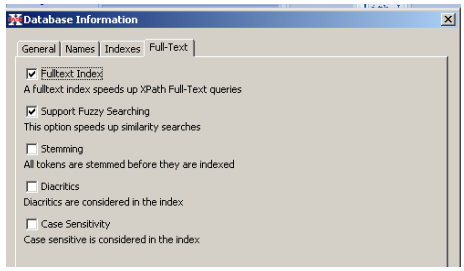
2.4 Advanced aspects

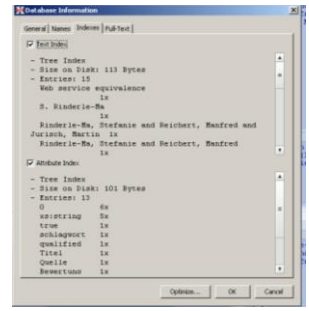
Native XML database (e.g., BaseX, http://basex.org/):



Questions:

- How are XML documents stored within a NDX?
- How can XML documents be queried?
  - Structured parts
  - Unstructured parts (e.g., full text)
- Indices?
- Performance?



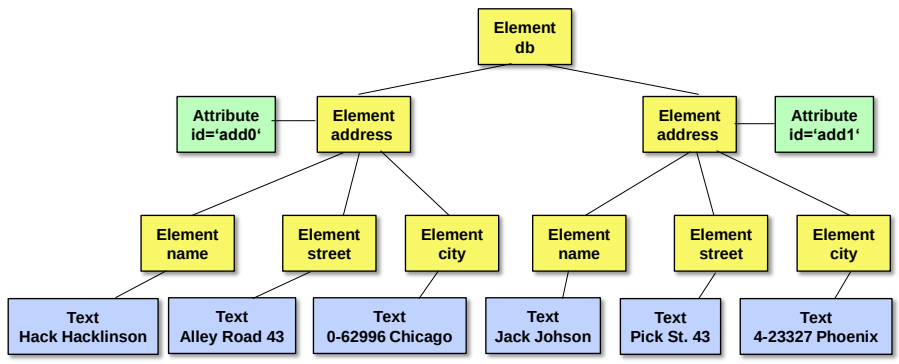


- NDX developed at University of Konstanz (http://basex.org/)
- Basic situation:
  - Different approaches which accelerate Xpath queries on relational databases
  - With Xpath acceleration storage and querying of XML documents in NDX becomes really competitive with relational databases
  - Prominent examples:
    - XPath Accelerator [GKT04] (implemented in MonetDB [Boncz+06], http://monetdb.cwi.nl)
    - X-Hive Persistent DOM (https://community.emc.com/community/edn/xmltech)
- Storage in BaseX is based on two data structures [Grün+06]
  - XML node table
  - Hash index

Example (from [Grün+06]):

```
<db>
      <address id='add0'>
            <name title='Prof.'>Hack Hacklinson</name>
            <street>Alley Road 43</street>
            <city>0-62996 Chicago</city>
      </address>
      <address id='add1'>
            <name>Jack Johnson</name>
            <street>Pick St. 43</street>
            <city>4-23327 Phoenix</city>
      </address>
</db>
```


Tree representation



2.4 Advanced aspects

Model-based storage in relational table [Grün+06]:

pre	par	token	kind	att	attVal
0	0	Db	Elem		
1	1	Address	Elem	Id	Add0
2	2	Name	Elem	Title	Prof.
3	3	Hack Hacklinson	Text		
4	2	Street	Elem		
5	3	Alley Road 43	Text		
6	2	City	Elem		
7	3	0-65996 Chicago	Text		
8	1	Address	Elem	Id	add01
9	2	Name	Elem		
10	3	Jack Johson	Text		
11	2	Street	Elem		
12	3	Pick St. 43	Text		
13	2	City	Elem		
14	3	4-23329 Phoenix	Text		

2.4 Advanced aspects

Storage in BaseX (source [Grün+06]):

TagIndex

id	tag
…0000	db
…0001	address
…0010	name
…0011	street
…0100	city

Node Table

par	Child/token	attribute
…0000	0…..0000	nil
…0001	0…..0001	0000…0000
…0010	0…..0010	0001…0001
…0011	1…..0000	nil
…0010	0…..0011	nil
…0011	1…..0001	nil
…0010	0…..0100	nil
…0011	1…..0010	nil
…0001	0…..0001	0000…0010
…0010	0…..0010	nil
…	…	…

AttNameIndex

id	att
…0000	id
…0001	title

AttValIndex

id	attVal
…0000	add0
…0001	Prof.
…0010	add1

id	text
…0000	Hack Hacklinson
…0001	Alley Road 43
…0010	0-62996 Chicago
…0011	Jack Johnson
…0100	Pick St. 43
…0101	4-23327 Phoenix

TextIndex

2.4 Advanced aspects

Description of data structure:

- Node Table represents relational structure
- Further attributes / tables:
  - Node kind (`kind`): 0 means element, 1 means text node
  - Node content (`token`): Tag name of an element or value of a text node
  - Attribute: combination of attribute name and value or `nil`
- Text content (tag names, value of a text node, attribute name and attribute value) are managed within a hash structure and referenced by integer values
- Node table is completely coded with integer values (optimizing CPU)
- Reduction of storage space by combination of attributes:
  - `kind` and `token`: `kind` needs 1 Bit, `token` less than 32 Bit

2.4 Advanced aspects

Description of hash structure:

- Three arrays:
  - `TOKENS` references indexed tokens
  - `ENTRIES` references position of first token
  - `BUCKET`: Lookup in overflow area
- Approach:
  - Calculate hash value for input token
  - Entry over `ENTRIES`
  - `ENTRIES` yields pointer to `TOKEN` array
  - If pointer yields `nil` no token has been stored.
  - Otherwise compare token with input
  - If comparison fails, `BUCKET` points to the next token or yields `nil`, if no token with same hash value exists

Hash index (source [Grün+06]):

ArrayPos	0	1	2	3
TOKENS	nil	XML	Xpath	XSLT
BUCKETS	nil	nil	nil	1
ENTRIES	nil	2	nil	3

hashValue("XML") → 3

- **ENTRIES [3] → 3**
- **TOKENS [3] = " XML" ? → FALSE**
- **BUCKETS [3] → 1**
- **TOKENS [1] = "XML" ? → TRUE**

Summary:

- By coding of XML document trees compact storage can be achieved.
- Processing XQuery or Xpath queries on these structures is also efficient
- Visualization of hierarchically structured data by using tree map algorithm

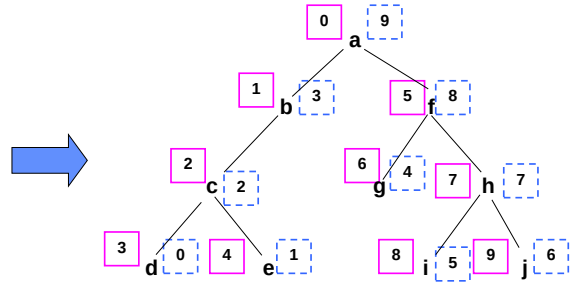
Support of XPath/XQuery in relational DBS

XPath Accelerator approach [GKT04]:

- Question: how can XML documents be stored and indexed within RDBMS such that XPath/Xquery are efficiently supported?
- Pre-consideration: Take tree structure of an XML document (abstract from attributes)
- Basic idea:
  - Annotate nodes of XML document tree with pre and post-order numbers
  - Pre- and post-order numbers of XML document nodes place them within two-dimensional coordinate system
  - Building indices can be done similar to B-tree or special spatial indicies (e.g., R-tree)

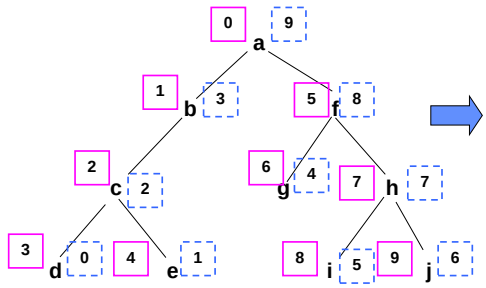
XPath Accelerator [GKT04], example:

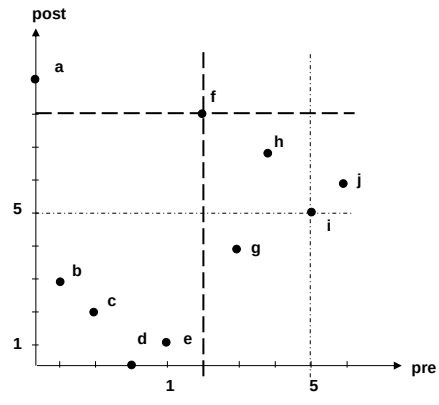
```
<a>
    <b>
        <c>
            <d/>
            <e/>
        </c>
    </b>
    <f>
        <g/>
        <h>
            <i/><j/>
        </h>
    </f>
</a>
```



3 Node numbers acc. preorder

3 Node numbers acc. Post-order





3 Node numbers acc. preorder

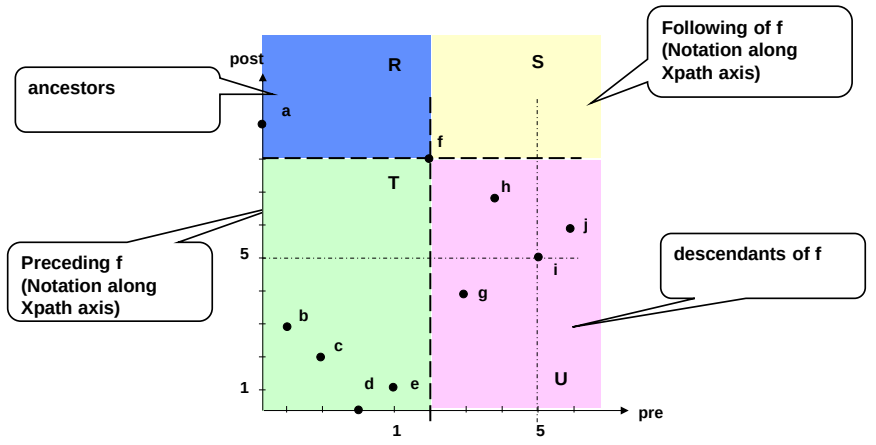
3 Node numbers acc. Post-order

Observation:

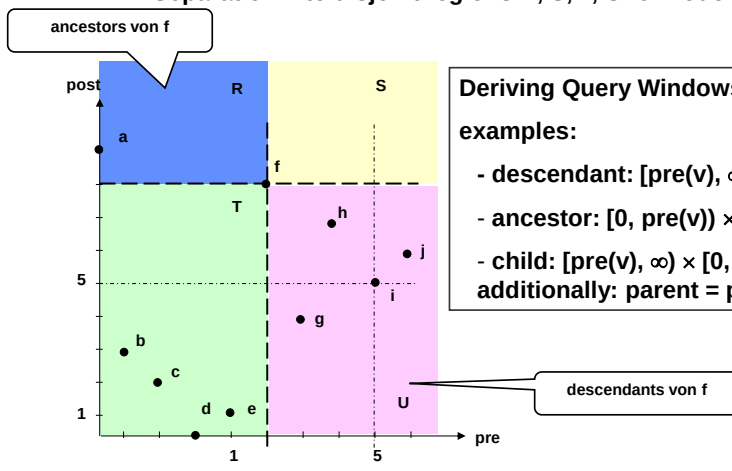
Node v‘ is descendant of node v

⇔ pre(v) < pre(v‘) ∧ post(v‘) < post(v)

Separation into disjoint regions R, S, T, U for node f



Separation into disjoint regions R, S, T, U for node f



Deriving Query Windows for axes:

examples:

- **descendant: [pre(v), ∞) × [0, post(v))**
- **ancestor: [0, pre(v)) × [post(v), ∞)**
- **child: [pre(v), ∞) × [0, post(v)) , additionally: parent = pre(v)**

2.4 Advanced aspects

- Additionally node and name test
- Altogether: complete description of node v by:
  - *desc(v) = ⟨pre(v), post(v), pre(v'), kind(v), name(v)⟩*
    (with v' = parent(v))
- Test on child axis:
  - *window(child, v) = ⟨ ♣, ♣, pre(v), elem, ♣ ⟩* (♣ corresponds to „don't care")
- example:
  - *desc(f) = ⟨ 5, 8, 9, elem, f ⟩*
  - Check g: *desc(g) = ⟨ 6, 4, 5, elem, g ⟩* → is child of f
  - Check j: *desc(j) = ⟨ 9, 6, 7, elem, j ⟩* → no child of f
- Additional optimization by restricting query window sizes

2.4 Advanced aspects

- Implementation of XPath Accelerator [GKT04]:
  - Purely relational in IBM DB2
  - In Monet [Boncz06]
- Relational implementation:
  - Storage structure is five-column table *accel(pre|post|par|kind|name)*
  - Separation of contents (cf. Separation vs. inlining): storing content in g in content tables: pre|text, pre|attr
  - **Rephrasing Xpath expressions**:
    - Expression $p = s_1/s_2/ \ldots / s_n$ is transformed into a sequence of region queries
    - Result of step s_i is input for step s_{i+1}
    - Context node for first step s_1 ?
    - Assuming an absoulte path starting from root node ($p=/s_1/ \ldots / s_n$) context table contains one tuple (e.g., *(0, 9, NULL, elem, a))*

- Then the corresponding SQL-Query turns out as

  SELECT DISTINCT v_n*.**
  FROM context c, accel v_1*, …, accel* v_n
  WHERE INSIDE(window(s_1*, c),* v_1*) AND … AND INSIDE(window(*s_n*,* v_{n-1}*),* v_n*)*
  ORDER BY v_n*.pre ASC*

- With function *INSIDE*

  INSIDE(⟨[pre_l*,* pre_h*], [*$post_l$*,* $post_h$*], p, k, n⟩, v) ≡*
  pre_l *< v.pre AND* pre_h *> v.pre AND* $post_l$ *< v.post AND* $post_h$ *> v.post*
  AND v.par = p AND v.kind = k AND v.name = n

Performance considerations:

- Given: XML documents having sizes 0.11 MB to 111.0 MB with 5257 to 5077531 nodes
- Test of 3 Xpath expression Query 1, Query 2, Query 3 along different Xpath axes (*descendant, ancestor, child, preceding-sibling*)
- Test of five different implementations:
  - relational in DB2
  - In Monet
  - With R-Tree
  - Native XML database
  - In DB2 with Edge Table Mapping
  - Results see [GKT04]

Outline

2.1 Motivation

2.2 Publishing database content in XML

2.3 Storage of XML documents in databases

2.4 Advanced aspects

2.5 Summary and outlook

References

2.5 Summary and outlook

- Challenges when storing XML documents in databases or publising database content as XML documents arise due to the following characteristics of the data;
  - XML data is hierarchically structured
  - Thus, object-relational databases are natural corresponding candidates for storage
  - However, relational databases practically much more relevant
  - Relational databases are flat → gap
  - Connected: choice of query language
  - Set-oriented (SQL) versus navigating (XPath, XQuery)

2.5 Summary and outlook

- Different possibilities to store XML documents in databases
  - Native XML databases (NDX)
  - Object-relational databases
  - Relational databases
    - Mapping to generic relations (e.g., Edge Table)
    - Specific mappings
  - User-defined mappings (shredding)
  - Hybrid storage (→ SQL/XML standard)
- Effect on query languages
  - XML query languages (XPath, XQuery)
  - On native XML databases
  - On relational databases (e.g., XPath Accelerator)
  - In connection with SQL (e.g., in SQL/XML standard)
  - SQL

2.5 Summary and outlook

- Strong research topic
- Commercial systems
- Important topic in the context of interoperability
- Databases are often used as data sources
- XML as exchange format
- → information integration (next chapter)

References

[Bourret09] R. Bourret: Going Native: Use Cases for native XML databases, http://www.rpbourret.com/xml/UseCases.htm (2009)

[FKM01] T. Fiebig, C.C. Kanne, G. Moerkotte: Natix – ein natives XML-DBMS, Datenbank-Spektrum 1 (2001)

[KM2000a] Kanne, C.-C.; Moerkotte, G.: Efficient Storage of XML Data. In: Int'l Conference on Data Engineering (ICDE'00), IEEE Computer Society Press, S. 198 (2000)

[KlMe03] M. Klettke, H. Meyer: Speicherung von XML-Dokumenten – eine Klassifikation. Datenbank-Spektrum 5: 40 – 50 (2003)

[Staken01] K. Staken: Introduction to Native XML Databases. XML.com (2001)

[Türker09] C. Türker: XML und Datenbanken. Vorlesung im Frühjahr, Universität Zürich (2009)

References

[Dada09] P. Dadam: Datenbanksysteme, Vorlesung im Wintersemester 2008/09, Universität Ulm (2009)

[Dada09a] P. Dadam: Objektrelationale und erweiterbare Datenbanken, Vorlesung im Sommersemester 2009, Universität Ulm (2009)

[Moos08] A. Moos: XQuery und SQL/XML in DB2-Datenbanken. Vieweg+Teubner (2008)

[MoSc06] Anders Møller, Michael Schwartzbach: An Introduction to XML and Web Technologies, Addison-Wesley, ISBN 0-321-26966-7 (2006)

[BBB00] R. Bourret, C. Bornhövd, A. Buchmann: A Generic Load/Extract Utility for Data Transfer between XML Document and Relational Databases. Int'l Workshop on Advanced Issues of E-Commerce and Web-based Information Systems (2000)

[GKT04] Torsten Grust, Maurice van Keulen, Jens Teubner: Accelerating XPath evaluation in any RDBMS. ACM Trans. Database Syst. 29: 91-131 (2004)

References

[EiMe01] A. Eisenberg, J.Melton: SQL/XML and the SQLX Informal Group of Companies. SIGMOD Record 30(3): 105-108 (2001)

[EiMe02] A. Eisenberg, J. Melton: SQL/XML is Making Good Progress. SIGMOD Record 31(2): 101-108 (2002)

[SQLXML06] Information technology — Database languages — SQL — Part 14: XML-Related Specifications (SQL/XML) SO/IEC JTC 1/SC 32 (2006)

[BOURRET] http://www.rpbourret.com/xml/XMLDBLinks.htm#Relational

[SQLX] www.sqlx.org

[Klet08] M. Klettke: XML und Datenbanken. Folien zum Buch „XML und Datenbanken“ (www.xml-und-datenbanken.de) (2008)

[Oracle03] SQL 2003 Standard Support in Oracle Database 10g. Oracle White Paper November 2003

[TuSa06] Türker, C.; Saake, G. Objektrelationale Datenbanken - Ein Lehrbuch. dpunkt.verlag, 2006

References

[BaUn77] R. Bayer, K. Unterauer: Prefix B-Trees. ACM Transactions on Database Systems 2(1): 11-26 (1977)

[FaCh87] C. Faloutsos, S. Christodoulakis: Optimal Signature Extraction and Information Loss. ACM Transactions on Database Systems, Vol. 12, No. 3, Sept. 1987, S. 395-428

[ReDa08] M. Reichert, P. Dadam: Speicherstrukturen und Zugriffspfade in Informationssystemen, Vorlesung, Universität Ulm (2008)

[Schek78] H.-J. Schek: The Reference String Indexing Method. Proc. on Information Systems Technology, Venice, Italy, 1978 (Lecture Notes in Computer Science 65, Springer-Verlag), S. 432-459

[Grün+06] Christian Grün, Alexander Holupirek, Marc Kramis, Marc H. Scholl, Marcel Waldvogel: Pushing XPath Accelerator to its Limits. ExpDB 2006

[HGS09] Alexander Holupirek, Christian Grün, Marc H. Scholl: BaseX & DeepFS joint storage for filesystem and database. EDBT 2009: 1108-1111

References (Further reading)

[FlKo99] D. Florescu, D. Kossmann: Storing and Querying XML Data using an RDMBS. IEEE Data Engineering Bulletin 22:27-34 (1999)

[Shan+01] J. Shanmugasundaram. E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, B. Reinwald: Efficiently publishing relational data as XML document. VLDB Journal 10:133-154 (2001)

[DFS99] A. Deutsch, M. Fernandez, D. Suciu: Storing Semistructured Data with STORED. ACM SIGMOD, S. 431-442 (1999)