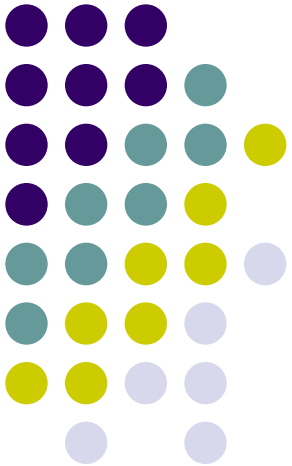
NPRG039

Advanced Aspects and New Trends in XML (and Related) Technologies

RNDr. Irena Holubová, Ph.D.

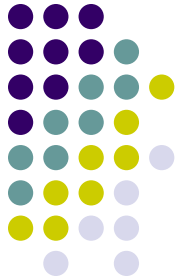
holubova@ksi.mff.cuni.cz



Lecture 5. XML Support in IBM DB2 and MS SQL Server, Comparison with Oracle DB

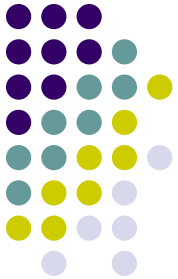
http://www.ksi.mff.cuni.cz/~svoboda/courses/171-NPRG039/

XML support in Oracle DB

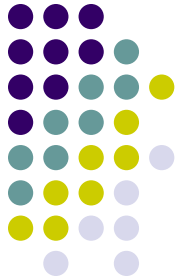


- XMLType storage
  - CLOB – unstructured
  - Object-relational – structured
    - Shredding into relations
    - Schema annotations
      - `SQLName`, `SQLType`, `maintainDOM`, `SQLInline`, ...
  - Binary – native
- XML publishing of relational data
  - SQL/XML
    - `XMLELEMENT`, `XMLATTRIBUTES`, `XMLFOREST`, `XMLCONCAT`, ...

XML support in Oracle DB

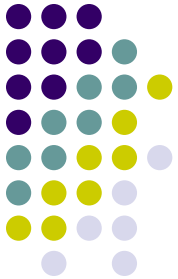


- XML retrieval using XPath, XQuery
  - SQL/XML
    - `XMLQuery`, `XMLTable`
  - XMLType functions
    - `existsNode`, `extract`, ...
- Indexing – depending on the storage
  - B-tree, XMLIndex, function-based, text-based
- XML data updates
  - `updateXML`, `insertXMLbefore`, `deleteXML`, ...
- XML Schema evolution
  - Copy-based
  - In-place – backward compatibility
- XML full-text
  - `CONTAINS`, `ora:contains`



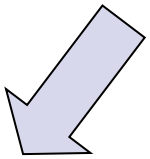
IBM DB2

IBM DB2



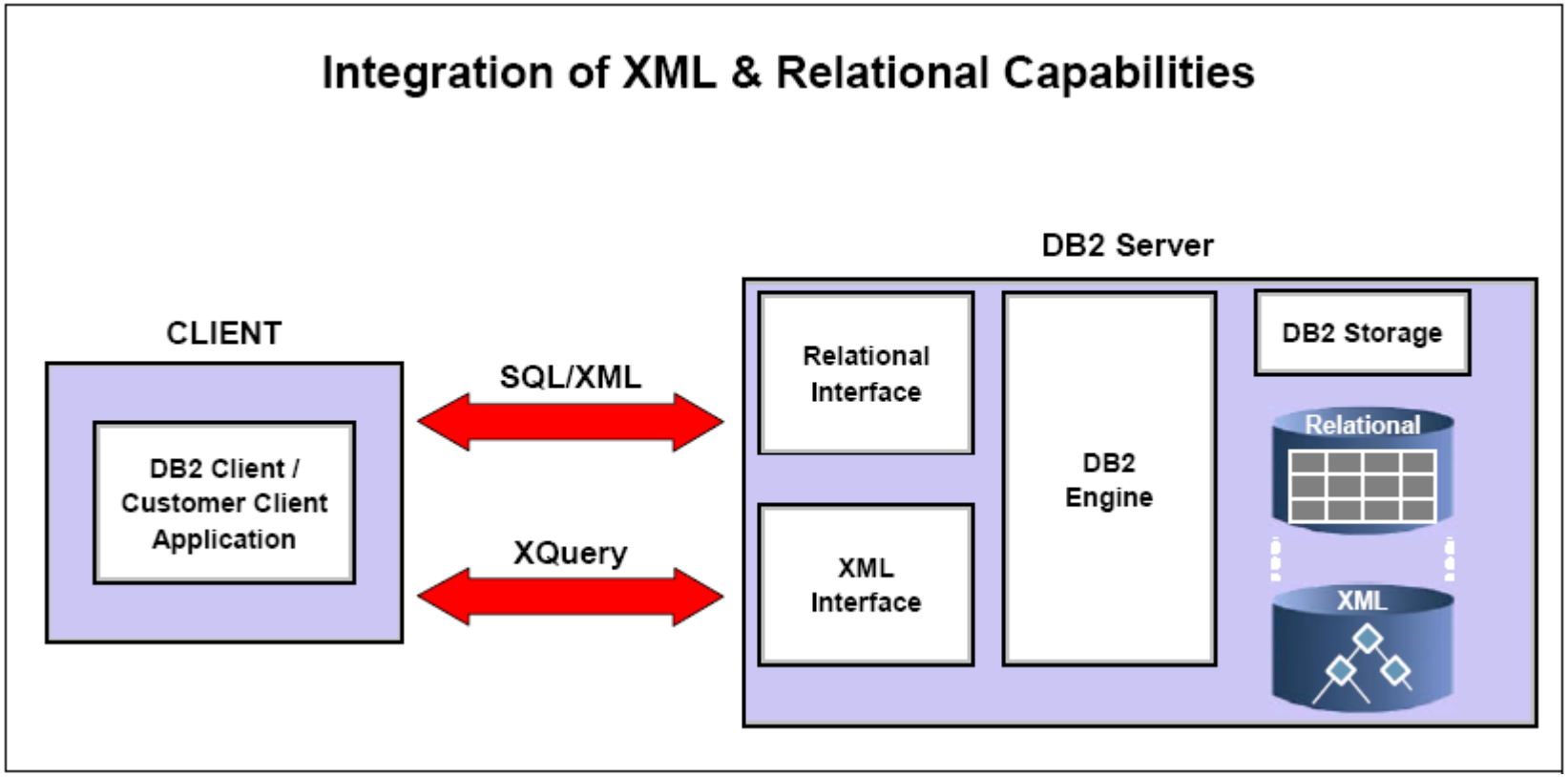
- http://www-01.ibm.com/software/data/db2/
- OS: Linux, UNIX, Windows
- Editions: Enterprise Server, Workgroup Server, Express-C, Express Edition
  - 90 days Data Server trial
  - Express-C – free, relational and XML data server
- pureXML
  - Native XML support in DB2
  - Earlier called IBM DB2 XML Extender

XML Data Storage



- Types:
  - LOB
  - Native XML
  - XML shredding
- Native:
  - Logical storage:
    - SQL XML data type: XML
  - Physical storage:
    - Native tree representation of XML data

Native XML Data Storage



Native XML Data Storage

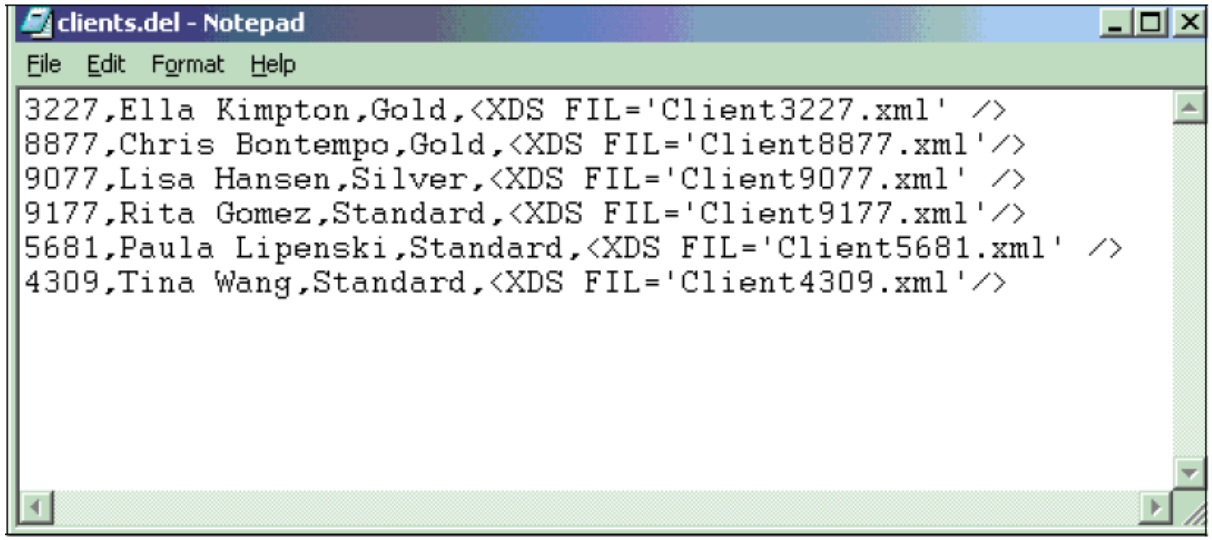
```
create table items (
  id int primary key,
  brandname varchar(30),
  itemname varchar(30),
  sku int,
  srp decimal(7,2),
  comments xml
  )
```

```
create table clients(
  id int primary key,
  name varchar(50),
  status varchar(10),
  contactinfo xml
  )
```

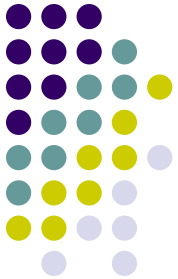
XML Data Inserting

```
insert into clients values (77, 'John Smith', 'Gold',
'<addr>111 Main St., Dallas, TX, 00112</addr>')
```

```
import from clients.del of del xml
  from C:/XMLFILES
insert into clients
```



XML Data Inserting with Validation



- By default no validation
  - Well-formed XML data
  - Validation must be explicitly called

URI

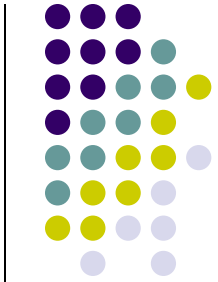
```
register xmlschema 'http://mysample.org'
  from 'C:/XMLFiles/ClientInfo.xsd'
  as user1.mysample complete
```

ID

location

```
import from clients.del of del xml
  from C:/XMLFILES
    xmlvalidate using xsd default user1.mysample
  insert into clients
```

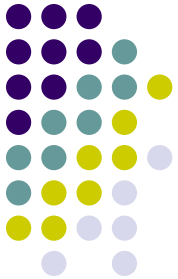
XML Data Inserting with Validation



- Implicit validation
  - Schema information is not passed by `INSERT` / `IMPORT`
- Needs:
  - `targetNamespace` in XML document
  - `xsi:scemaLocation` in XSD
  - URI in registration

```
insert into test values
xmlvalidate (xmlparse ( document
  '<XML document>‘ preserve whitespace ) )
```

Schema Annotations



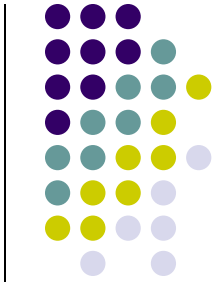
- XML-to-relational mapping
  - Different approach than in Oracle (i.e., not related to XML type)
- In some documentations marked as deprecated
  - `XMLTABLE` should be used instead
- Steps:
  1. Create tables
  2. Annotate schema
  3. Register schema
  4. Call `DECOMPOSE XML DOCUMENT`

```
REGISTER XMLSCHEMA 'd:/work/db2/sa.xsd'
  FROM D:\work\db2\sa.xsd
  AS admin.test_shr COMPLETE ENABLE DECOMPOSITION;
```

```
DECOMPOSE XML DOCUMENT d:/work/db2/s.xml
  XMLSCHEMA admin.test_shr
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1">
  <xsd:annotation>
    <xsd:appinfo>
      <db2-xdb:defaultSQLSchema>ADMIN</db2-xdb:defaultSQLSchema>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:element name="DeliveryTimes">
    <xsd:complexType>
      <xsd:sequence maxOccurs="unbounded">
        <xsd:element name="Entry">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Zip" type="xsd:string"
                           db2-xdb:rowSet="TRTIME"
                           db2-xdb:column="ZIP_CODE"/>
              <xsd:element name="Duration" type="xsd:integer"
                           db2-xdb:rowSet="TRTIME"
                           db2-xdb:column="DURATION"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

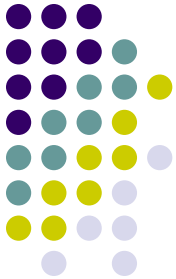
Advanced Annotations



- The data can be normalized, its white space manipulated, the data manipulated in an expression, or truncated before insertion.
  - `db2-xdb:expression`, `db2-xdb:normalization`, `db2-xdb:truncate`
- The data can be inserted conditionally
  - e.g., only values matching certain criteria should be decomposed into the table-column pairs
  - `db2-xdb:condition`, `db2-xdb:locationPath`
- Foreign key relationships can be described
- The same element or attribute can be inserted into multiple table-column pairs
  - `db2-xdb:rowSetMapping`
- Multiple elements or attributes can be inserted into the same table-column pair
  - `db2-xdb:table`
- …

SQL/XML Querying

XMLQuery, XMLExists

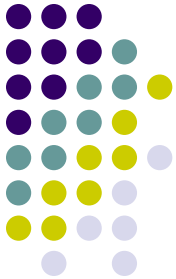


```
select name from clients
where xmlexists('$c/Client/Address[zip="95116"]'
  passing clients.contactinfo as "c")
```

```
select xmlquery(
  '$c/Client/email' passing contactinfo as "c")
from clients
where status = 'Gold'
```

```
select xmlquery('$c/Client/email[1]'
  passing contactinfo as "c")
from clients
where status = 'Gold'
and xmlexists('$c/Client/email' passing contactinfo as "c")
```

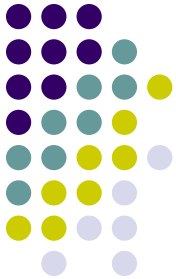
SQL/XML Querying
XMLTable



```
select t.Comment#, i.itemname, t.CustomerID, Message
from items i,
  xmltable('$c/Comments/Comment' passing i.comments as "c"
    columns
      Comment#    integer path 'CommentID',
      CustomerID  integer path 'CustomerID',
      Message XML by ref  path 'Message') as t
```

Allowed

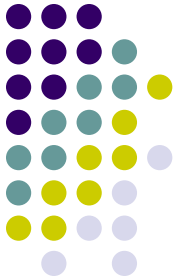
Relational Views of XML Data



```
create view commentview
  (itemID, itemname, commentID, message, mustrespond)
as select i.id, i.itemname,
          t.CommentID, t.Message,
          t.ResponseRequested
from items i,
  xmltable('$c/Comments/Comment' passing i.comments as "c"
    columns
      CommentID integer path 'CommentID',
      Message varchar(100) path 'Message',
      ResponseRequested varchar(100) path
        'ResponseRequested') as t;
```

- No column indexes over XML data
  - Inefficient queries
- Solution: Highly restrictive predicates involving indexed SQL types

Joining SQL and XML Data



```
select clients.name, clients.status
from items, clients
where xmlexists('$c/Comments/Comment[CustomerID = $p]'
  passing items.comments as "c",
          clients.id as "p")
```

XML

SQL

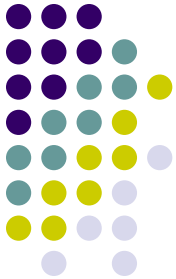
Advanced XQuery Queries

- XQuery as a top-level query language
  - Not embedded in SQL
  - But needs source of data

```
xquery db2-fn:xmlcolumn
  ('ITEMS.COMMENTS')/Comments/Comment/Message
```

```
xquery for $y in db2-fn:xmlcolumn
  ('ITEMS.COMMENTS')/Comments/Comment return ($y/Message)
```

Advanced XQuery Queries



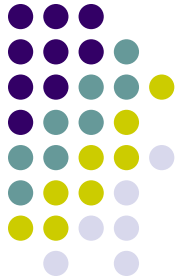
- XQuery queries with embedded SQL
  - Source of data can be SQL

SQL/XML: embedding XML queries into SQL

```
xquery db2-fn:sqlquery
  ('select comments
    from items where srp > 100')/Comments/Comment/Message
```

```
xquery
  for $y in
    db2-fn:sqlquery
      ('select contactinfo from clients
        where status=''Gold'' ')/Client
  where $y/Address/city="San Jose"
  return (
    if ($y/email) then <emailList>{$y/email}</emailList>
    else $y/Address
  )
```

SQL/XML Publishing



- SQL/XML statements
  - `XMLSERIALIZE` – converts an XML value into a string
  - `XMLPARSE` – converts a string into an XML value
  - `XMLELEMENT` – constructs a named XML element
  - `XMLFOREST` – constructs a sequence (forest) of XML element nodes
  - `XMLATTRIBUTES` – constructs one or more XML attribute nodes
  - `XMLCONCAT` – concatenates two or more XML values
  - `XMLAGG` – aggregates XML values as sequence items into a single resulting XML value
  - `XMLNAMESPACES` – constructs XML namespace declarations

SQL/XML Publishing

```
<tr>
  <td>000010</td>
  <td>CHRISTINE</td>
  <td>HAAS</td>
  <td>3978</td>
  <td>A00</td>
  <td>SPIFFY COMPUTER SERVICE </td>
</tr>
<tr>
  ...
</tr>
```

```
select xmlserialize (
  content xmlelement (
    name "tr",
    xmlelement (name "td", e.empno),
    xmlelement (name "td", e.firstnme),
    xmlelement (name "td", e.lastname),
    xmlelement (name "td", e.phoneno),
    xmlelement (name "td", d.deptno),
    xmlelement (name "td", substr(d.deptname, 1, 24)
    )
  ) as clob(120)
) as "result"
from employee e, department d
where e.workdept = d.deptno and year(hiredate) < 1970;
```

XML Data Updates

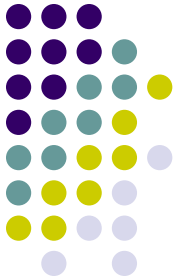
- SQL `UPDATE`, `DELETE`
  - Document level update

```
update clients set contactinfo = (
xmlparse(document '<email>newemail@someplace.com</email>' ))
where id = 3227
```

```
delete from clients
where xmlexists('$c/Client/Address[zip="95116"]'
  passing clients.contactinfo as "c")
```

- `XMLUPDATE` stored procedure
  - What to update, how to update, other update parameters, …

XQuery Update Facility

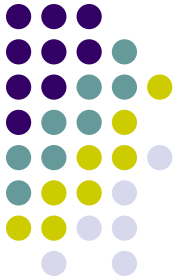


- Since DB2 9.5
- We can:
  - Replace the value of a node
  - Replace a node with a new one
  - Insert a new node
    - Before/after a given node
  - Delete a node
  - Rename a node
  - Modify multiple nodes in a document in a single `UPDATE` statement
  - Update multiple documents in a single `UPDATE` statement

http://www.w3.org/TR/xquery-update-10/

XQuery Update Facility

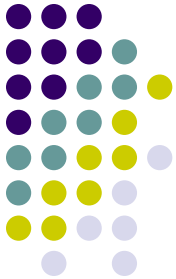
Sample Data



```
create table xmlcustomer (cid bigint, info XML)
```

```
insert into xmlcustomer values (1000,
'<customerinfo>
   <name>John Smith</name>
   <addr country="Canada">
     <street>Fourth</street>
     <city>Calgary</city>
     <state>Alberta</state>
     <zipcode>M1T 2A9</zipcode>
   </addr>
   <phone type="work">963-289-4136</phone>
 </customerinfo>')
```

XQuery Update Facility
How it Works



```
update xmlcustomer set info = xmlquery(
  'copy $new := $INFO
   modify do delete $new/customerinfo/phone
   return  $new') where cid = 1000;
```

```
...
   modify do rename $new/customerinfo/addr as "address"
...
```

```
...
   modify do replace $new/customerinfo/phone with
     <phone type="home">416-123-4567</phone>
...
```

```
...
   modify do insert <phone type="cell">777-555-3333</phone>
     into $new/customerinfo
...
```

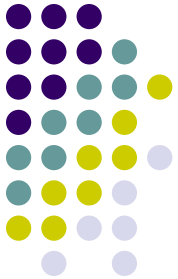
XQuery Update Facility
Where to Use

```
update xmlcustomer set info = xmlquery(
  `...')
where cid = 1000;
```

```
select xmlquery(
  'copy $new := $INFO
   modify do replace value of
   $new/customerinfo/phone with "905-xxx-xxxx"
   return $new ')
from xmlcustomer where cid = 1000;
```

```
update xmlcustomer set info = xmlvalidate(xmlquery(
  'copy $new := $INFO
   modify do replace value of $new/customerinfo/phone
   with "905-477-9011"
   return  $new ')
  according to xmlschema id "cust.custschema")
  where cid = 1000;
```

XML Data Indexing



- XML region index
  - Stores the locations of each XML document
  - XML document is stored in one or more regions
  - Created automatically
- XML column path index
  - For each XML column
  - Provides mappings of unique XML paths to their IDs
  - Created automatically
- XML column index
  - Enhances query performance by indexing XPath expressions

```
create index myindex on items(comments)
  generate key using xmlpattern
    '/Comments/Comment/CommentID' as sql double
```

XML Full-Text

```
select appl_id from feedback where
CONTAINS(comment,
  'section("/feedback/entry/comment") "good"
   in same sentence as "quick response"')=1
```

- Proximity searching, Boolean operators, fuzzy searching, stemming, wildcards, …

XML Schema Evolution

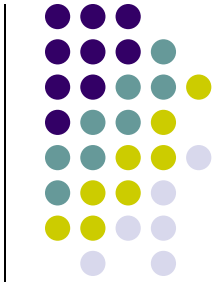
- **XSR_UPDATE**
  - Evolves an existing XML schema in XML schema repository
    - Repository of registered schemas
- Only with backward compatibility
  - XML schema changes so that it can be used to validate both already existing and newly inserted XML documents
- Both schemas must be registered

```
CALL SYSPROC.XSR_UPDATE(
  'STORE', 'PROD', 'STORE', 'NEWPROD', 1)
```

- Schema `STORE.PROD` is updated with the contents of `STORE.NEWPROD` which is dropped

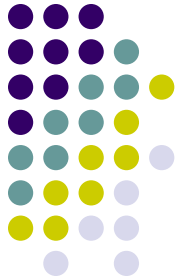
References

- pureXML for Dummies
  - https://www14.software.ibm.com/webapp/iwm/web/signup.do?lang=en_US&source=sw-infomgt&S_PKG=book-pureXML-dummies
- Managing XML Data – Best Practices
  - http://download.boulder.ibm.com/ibmdl/pub/software/dw/dm/db2/bestpractices/DB2BP_XML_0508I.pdf
- DB2 9: pureXML Overview and Fast Start
  - www.redbooks.ibm.com/abstracts/sg247298.html
- DB2 9 pureXML Guide
  - http://www.redbooks.ibm.com/abstracts/sg247315.html
- DB2 pureXML enablement wiki
  - https://www.ibm.com/developerworks/wikis/display/db2xml/Home



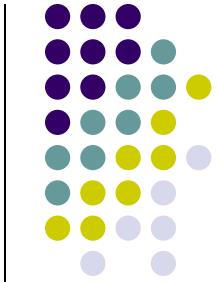
Microsoft SQL Server

Microsoft SQL Server



- http://www.microsoft.com/sqlserver/2008/en/us/default.aspx
- OS: Windows XP, Windows Vista or Windows 7
- Editions:
  - http://www.microsoft.com/sqlserver/2008/en/us/editions.aspx
    - Core – Enterprise, Standard
    - Specialized – Workgroup, Web, Developer
    - Free – Express, Compact 3.5
- No special name XML support package

Storing XML Data

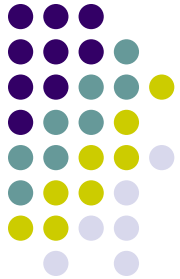


- LOB data type
- XML data type
  - Native XML repository

```
CREATE TABLE person (
  id INT PRIMARY KEY,
  desc XML);
```

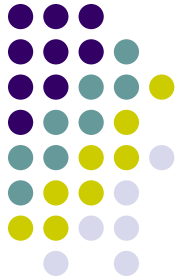
- Shredding into tables
  - Annotated XML schema

Schema Annotations



- Again different approach
  - Annotated XSD = XML view over relational data
  - Posing XML query over XSD → posing SQL query over relations and getting the result in XML
- Again denoted as deprecated
- Default mapping:
  - Element of complex type → table with the same name
  - Element/attribute of simple type → column with the same name in the table

Schema Annotations



- **`sql:relation`** – maps an XML item to a database table
- **`sql:field`** – maps an XML item to a database column
- **`sql:is-constant`** – creates an XML element that does not map to any table but occurs in query result
- **`sql:mapped`** – allows schema items to be excluded from the result
- **`sql:key-fields`** – allows specification of column(s) that uniquely identify the rows in a table
- **`sql:max-depth`** – allows you to specify depth in recursive relationships that are specified in the schema
- …

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">
  <xsd:element name="Contact" sql:relation="Person.Contact" >
   <xsd:complexType>
     <xsd:sequence>
        <xsd:element name="FName" type="xsd:string"
                     sql:field="FirstName" />
        <xsd:element name="LName" type="xsd:string"
                     sql:field="LastName" />
     </xsd:sequence>
     <xsd:attribute name="ContactID" type="xsd:integer" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

schema

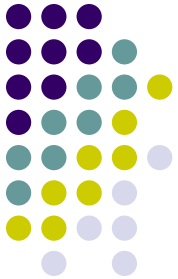
```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:xpath-query mapping-schema="MySchema-annotated.xml">
    /Contact
  </sql:xpath-query>
</ROOT>
```

query

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
 <Contact ContactID="1">
    <FName>Gustavo</FName>
    <LName>Achong</LName>
 </Contact>
  .....
</ROOT>
```

result

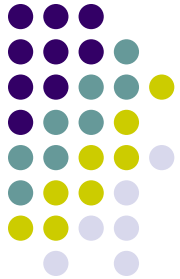
Indexing XML Data



- XML index for XML data type
- **Primary XML index**
  - Indexes all tags, values and paths over the XML instances in the column
    - Exploiting the ORDPATHS schema
- **Secondary indexes**
  - `PATH` index
    - B+ tree on (path, value) pair of each XML node
    - Usage of path expressions – e.g., `exists`

```
select * from xmlinvoice
where invoice.exist('/Invoice/@InvoiceID[. = "1003"]') = 1
```

Indexing XML Data



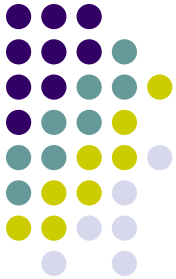
- PROPERTY index
  - B+ tree clustered on the (PK, path, value) tuple
    - PK is the primary key of the base table
  - Retrieval of multiple values from individual XML instances

```
select * from xmlinvoice
where invoice.exist('//Invoice/@InvoiceID[. = "1003"]') = 1
```

- VALUE index
  - B+ tree on (value, path) pair of each node in document order
    - Reverse order than in PATH
  - Looking for values at unknown position

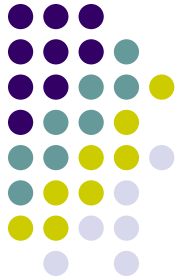
```
//author[last-name="Howard"]/book [@* = "novel"]
```

Creating and Indexing XML Data



```
CREATE TABLE person (id INT PRIMARY KEY, desc XML);

INSERT INTO person VALUES (1, '<desc>...</desc>');

CREATE PRIMARY XML INDEX idx_desc on person (desc);

CREATE XML INDEX idx_desc_path on person (desc)
  USING XML INDEX idx_desc FOR PATH;

CREATE XML INDEX idx_desc_property on person (desc)
  USING XML INDEX idx_desc FOR PROPERTY;
```

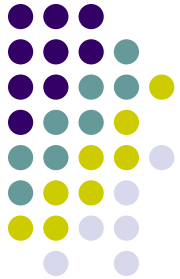
XML Full-Text



- SQL function `CONTAINS`
  - Token match using stemming
  - Note: XQuery `contains` = substring match
- Full-text index
  - Can be used also for other SQL types

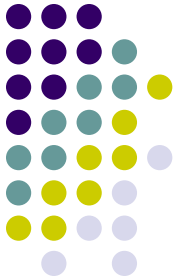
```
CREATE FULLTEXT CATALOG ft AS DEFAULT

CREATE FULLTEXT INDEX ON person (desc) KEY INDEX
  PK_docs_023D5A04

SELECT * FROM person
  WHERE CONTAINS (desc, 'mff.cuni.cz')
```

XML Data Querying



- XQuery
  - **`Exist`** – existential checks on an XML instance
    - Checks existence of nodes
  - **`Value`** – extracts a scalar value from an XML instance
    - Returns SQL value
  - **`Query`** – extracts parts of an XML instance
    - Returns XML data type result
  - **`Nodes`** – returns one row for each node that matches the query

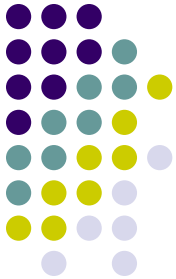
XML Data Querying



```
SELECT xCol.query('/doc[@id = 123]//section')
FROM   docs
WHERE  xCol.exist('/doc[@id = 123]') = 1
```

```
SELECT pk, xCol.query('
   for $s in /doc[@id = 123]//section
   where $s/@num >= 3
   return <topic>{data($s/title)}</topic>')
FROM docs
```

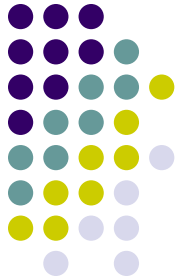
Data Binding in SQL Queries



- To ease usage of both SQL and XML data in queries
- **`sql:variable`**
  - Mapping SQL values to XML values
- **`sql:column`**
  - Accessing SQL columns in XQuery

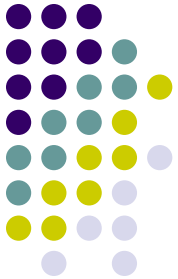
```
DECLARE @date varchar(20)
SET @date = '13/1/2003'
SELECT desc
FROM person
WHERE desc.exist ('/reord[date = sql:variable("@date")]')
= 1
```

SQLXML



- Not SQL/XML standard!
  - Similar idea: to bridge the gap between SQL and XML data
- **`OPENXML`** – SQL view of XML data
  - Explicit
  - Using a general mapping strategy
- **`FOR XML`** – XML view of SQL relations using modes
  - **`RAW`** – generates a single `<row>` element per each row in the rowset that is returned by the `SELECT` statement; its columns are mapped either to attributes or subelements depending on other parameters
  - **`AUTO`** – generates nesting in the resulting XML using heuristics based on the `SELECT`
  - **`EXPLICIT`** – allows more control over the shape of the XML view using a set of special directives
  - **`PATH`** – flexibility of the `EXPLICIT` mode in a simpler manner – using paths and nested queries

SQLXML
OPENXML



```
SELECT *
FROM OPENXML (@docHandle, '/record')
WITH (PersonName varchar(10) 'name',
      PersonID int '@id',
      PersonEmail varchar(10) 'email')
```

- Like XMLTABLE

SQLXML
FOR XML

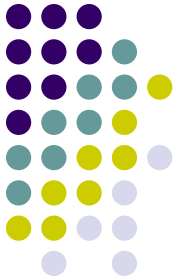
```
SELECT ProductModelID, Name
FROM Production.ProductModel
WHERE ProductModelID=122 or ProductModelID=119
FOR XML RAW
```

```
<row ProductModelID="122"
     Name="All-Purpose Bike Stand" />
<row ProductModelID="119"
     Name="Bike Wash" />
```

```
SELECT ProductModelID, Name
FROM Production.ProductModel
WHERE ProductModelID=122 or ProductModelID=119
FOR XML RAW, ELEMENTS;
```

```
<row>
  <ProductModelID>122</ProductModelID>
  <Name>All-Purpose Bike Stand</Name>
</row>
<row>
  <ProductModelID>119</ProductModelID>
  <Name>Bike Wash</Name>
</row>
```

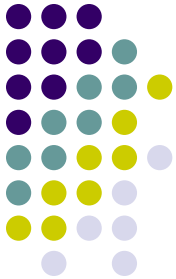
SQLXML
FOR XML



```
SELECT Cust.CustomerID,
       OrderHeader.CustomerID,
       OrderHeader.SalesOrderID,
       OrderHeader.Status,
       Cust.CustomerType
FROM Sales.Customer Cust, Sales.SalesOrderHeader
OrderHeader
WHERE Cust.CustomerID = OrderHeader.CustomerID
ORDER BY Cust.CustomerID
FOR XML AUTO
```

```
<Cust CustomerID="1" CustomerType="S">
  <OrderHeader CustomerID="1" SalesOrderID="43860" Status="5" />
  <OrderHeader CustomerID="1" SalesOrderID="44501" Status="5" />
  <OrderHeader CustomerID="1" SalesOrderID="45283" Status="5" />
  <OrderHeader CustomerID="1" SalesOrderID="46042" Status="5" />
</Cust>
...
```

Validity Checking

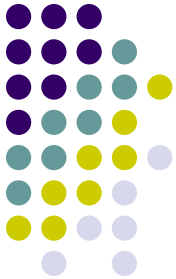


- Validity checking
  - `SCHEMA COLLECTION` – store one or more schema
- XML column or variable can be associated with schema collection → validity checking is ensured

```
CREATE XML SCHEMA COLLECTION MyColl AS
'<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://personschema1.xsd">
  ...
</schema>'
```

```
CREATE TABLE person (
  id INT PRIMARY KEY,
  desc XML (MyColl) )
```

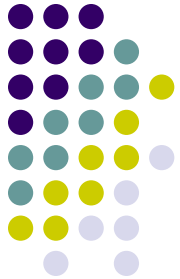
Updating XML Data



- **modify**
  - Built-in function
  - Parameters: Data + required operation
- Subtrees can be inserted
  - Before/after a specified node, as the leftmost/rightmost child
- Attribute/element/text nodes can be inserted
- Subtrees can be deleted
- Scalar values can be replaced

```
UPDATE person SET desc.modify('
  insert <dg>PhD</dg>
  after (/record/name[.="Irena Mlynkova"])')
```

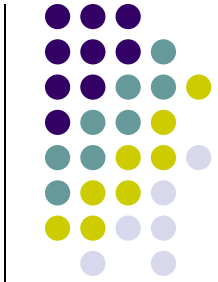
XML Schema Evolution



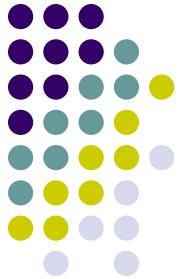
- Restricted
  - Backward compatibility
- Implicitly ensured using `SCHEMA COLLECTIONS`
  - Can be extended with new schemas
- Respective XML column/variable can contain data valid against both old and new XML schemas in the collection

```
ALTER XML SCHEMA COLLECTION MyColl ADD
'<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://personschema2.xsd">
  ...
</schema>'
```

References



- What's New for XML in SQL Server 2008
  - http://www.microsoft.com/sqlserver/2008/en/us/wp-sql-2008-whats-new-xml.aspx
- SQL Server Best Practices
  - http://msdn.microsoft.com/en-us/sqlserver/bb671432.aspx
- XML Best Practices for Microsoft SQL Server 2005
  - http://msdn.microsoft.com/cs-cz/library/ms345115(en-us,SQL.90).aspx



System Comparison

Comparison of Key XML Features

Feature	Oracle DB	DB2	SQL Server
XML data type	XMLType	XML	XML
	Structured, binary, unstructured	LOB, native, (structured with different usage)	LOB, native, (structured with different usage)
Mapping	User-defined	User-defined	User-defined
	Names, data types and storage strategies (VARRAY vs. LOB)	Relations, columns, conditions, expressions	Relations, columns, keys, relationships
Indexing	B-tree, XMLIndex, function-based, text-based	Region index, column path index, XML index, full-text index	Primary, secondary (PATH, PROPERTY, VALUE), full-text index

Comparison of Key XML Features

Feature	Oracle DB	DB2	SQL Server
Querying	XQuery, SQL/XML	XQuery, SQL/XML, SQL embedded to XQuery	XQuery, SQLXML (OPENXML, FOR XML)
Other operations	Validity checking, XSL transformations	Validity checking, XSL transformations	Validity checking, XSL transformations only via an external tool
Updating	Own functions for inserting, replacing, deleting nodes	XQuery Update facility	Own function with a parameter for inserting, replacing, deleting nodes
Evolution	With/without backward compatibility	Backward compatibility must be ensured	Backward compatibility using SCHEMA COLLECTIONs

General Comparisons of DBs

- http://www.microsoft.com/sqlserver/2008/en/us/compare.aspx
  - From Microsoft point of view
  - SQL Server 2008 is better than Oracle DB in:
    - Performance and Scalability, Security, Business Intelligence tools
  - SQL Server 2008 is better than DB2 in:
    - Performance and scalability, higher availability, industry-leading security, easier manageability, enhanced developer productivity, leading Business Intelligence and data warehousing capabilities, a platform for mainframe OLTP, and SAP integration

General Comparisons of DBs

- http://www.mssqlcity.com/Articles/Compare/Compare.htm
  - General comparisons of various DBs
  - Platform, performance, price, features, …
  - Not up-to-date