MI-PDB, MIE-PDB: **Advanced Database Systems**

http://www.ksi.mff.cuni.cz/~svoboda/courses/2015-2-MIE-PDB/

Lecture 7:

XML Databases, SQL/XML, Sedna

5. 4. 2016

Lecturer: **Martin Svoboda**
svoboda@ksi.mff.cuni.cz

Author: **Martin Svoboda**
Faculty of Mathematics and Physics, Charles University in Prague
Course NPRG036: **XML Technologies**
Course NPRG039: **Advanced Aspects and New Trends in XML**

Outline

- **XML databases**
  - Introduction
  - **SQL/XML**
    - Extension to SQL for XML data
    - Used in XML-enabled (object-)relational database systems
  - **Sedna**
    - Native XML database system

XML Databases

Motivation

- **Why XML databases?**
  - We have high volumes of data
  - We want to work with them efficiently
    - Modeling, acquisition, creation, organization, evolution
    - Retrieval, querying, processing, updates
    - Administration, security, concurrency, recovery
    - …

Documents vs. Databases

- **World of documents**
  - Many small and usually static documents
  - Implicit structure given by tagging
  - Suitable for humans
  - Operations: editing, printing, retrieval, searching, …
- **World of databases**
  - Several huge and usually dynamic databases
  - Explicit structure given by a schema
  - Suitable for machines
  - Operations: storing, querying, cleansing, updating, …

XML Data

- Semi-structured data
  - **Document-oriented**
    - Created and processed by humans
    - Irregular and less structured
    - Mixed contents, CDATA sections, …
    - Important order of sibling elements
  - **Hybrid**
  - **Data-oriented**
    - Created and processed by machines
    - Regular and fully structured
    - Unimportant order of sibling elements

Storage Strategies

- **General classification**
  - File system
  - (Object-)relational database systems
    - XML-enabled or not
  - Native XML database systems

Storage Strategies

- **(Object-)relational DBMS**
  - BLOB/CLOB
    - The highest level of round tripping
    - No available XML-specific operations
  - Mapping techniques
    - XML data shredded into relations
  - **XML-enabled systems**
    - Extensions allowing to work with XML data
    - XML datatype and SQL/XML

Storage Strategies

- **Native XML DBMS**
  - Advantages
    - Logical model – directly based on XML documents
    - Query languages such as XPath, XQuery, …
    - DOM/SAX interfaces
    - …
  - Disadvantages
    - Implementations from scratch

Current Database Systems

- **XML-enabled ORDBMS**
  - Oracle DB
  - IBM DB2
  - PostgreSQL
  - Microsoft SQL Server
- **Native XML DBMS**
  - Sedna
  - BaseX
  - eXist-DB

SQL/XML

Introduction

- **SQL/XML**
  - **Extension to SQL for XML data**
    - XML datatype
    - Functions, constructors, mappings, XQuery embedding, …
- Standards
  - **SQL:2011-14** (ISO/IEC 9075-14:2011)
    - Older versions 2003, 2006, 2008

Sample XML Document

```
<?xml version="1.0"?>
<library>
   <book id="1" catalogue="c1" language="en">
      <title>Red</title>
      <author>John</author>
      <author>Peter</author>
   </book>
   <book id="2" catalogue="c1">
      <title>Black</title>
      <price>25</price>
   </book>
   <book id="3" catalogue="c2" language="en">
      <title>White</title>
      <author>John</author>
   </book>
</library>
```

Sample Relational Schema

- Table: books

id	catalogue	title	details	language
1	c1	Red	<author>John</author> <author>Peter</author>	en
2	c1	Black	<price>25</price>	NULL
3	c2	White	<author>John</author>	en

- Table: languages

code	name
en	English
cs	Czech

Sample Query

- Query statement

```
SELECT
   id,
   XMLELEMENT(
      NAME "book",
      XMLELEMENT(NAME "title", title),
      details
   ) AS book
FROM books
WHERE (language = "en")
ORDER BY title DESC
```

Sample Query

- Query result

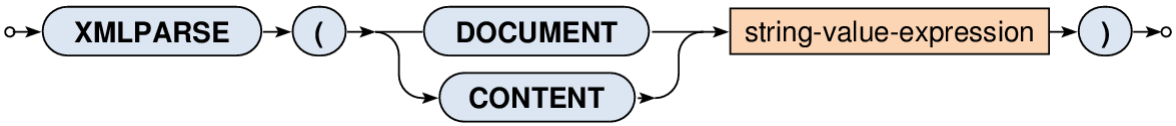
id	book
3	<book> <title>White</title> <author>John</author> </book>
1	<book> <title>Red</title> <author>John</author> <author>Peter</author> </book>

XML Datatype

- Traditional types
  - BLOB, CLOB, VARCHAR, …
- **Native XML type**
  - Collection of information items
    - Based on XML Information Set (**XML Infoset**)
      - Elements, attributes, processing instructions, …
      - Fragments with more root elements are allowed as well
  - NULL

Parsing XML Values

- XMLPARSE
  - **Creates an XML value from a string**
    - DOCUMENT – well-formed document with right one root
    - CONTENT – well-formed fragment

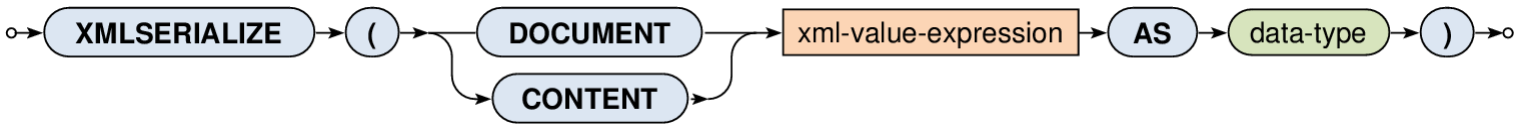



```
SELECT XMLPARSE(
   DOCUMENT "<book><title>Red</title></book>"
) AS result
```

result
<book> <title>Red</title> </book>

Serializing XML Values

- XMLSERIALIZE
  - **Exports an XML value to a string**




```
SELECT
   id, title,
   XMLSERIALIZE(CONTENT details AS VARCHAR(100)) AS export
FROM books
```

id	title	export
1	Red	<author>John</author><author>Peter</author>
...	...	...

Well-Formedness Predicate

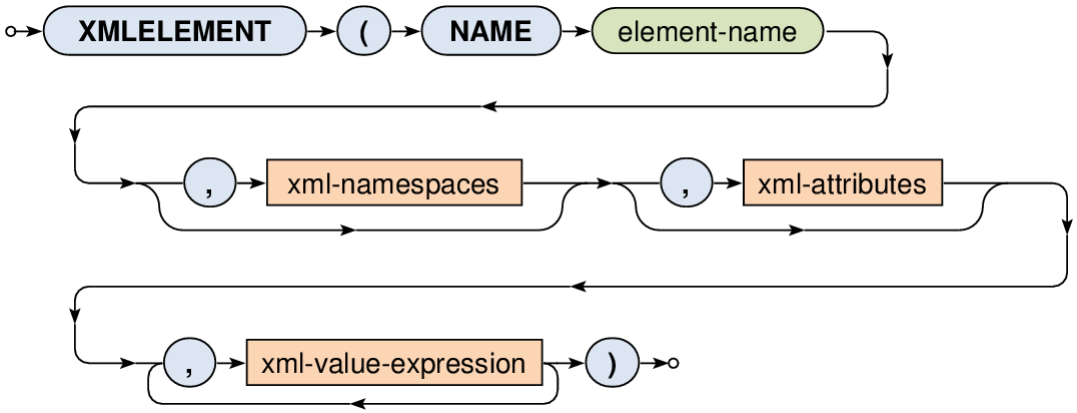
- IS DOCUMENT
  - **Tests whether an XML value is an XML document**
    - Returns `TRUE` if there is right one root element
    - Otherwise `FALSE`

Constructors

- Functions for construction of XML values…
  - **XMLELEMENT** – elements
  - **XMLNAMESPACES** – namespace declarations
  - **XMLATTRIBUTES** – attributes
  - **XMLCOMMENT** – comments
  - **XMLPI** – processing instructions
  - **XMLFOREST** – sequences of elements
  - **XMLCONCAT** – concatenations of values
  - **XMLAGG** – aggregates

Elements

- XMLELEMENT
  - **Creates an XML element** with a given name and…
    - optional **namespace declarations**
    - optional **attributes**
    - optional **element content**



Elements: Example 1

```
SELECT
   id,
   XMLELEMENT(NAME "book", title) AS result
FROM books
ORDER BY id
```

id	result
1	<book>Red</book>
2	<book>Black</book>
3	<book>White</book>

Elements: Example 2: Subelements

```
SELECT
   id,
   XMLELEMENT(
      NAME "book",
      XMLELEMENT(NAME "title", title),
      XMLELEMENT(NAME "language", language)
   ) AS records
FROM books
```

id	records
1	<book> **<title>Red</title>** **<language>en<language>** </book>
…	…

Elements: Example 3: Mixed Content

```
SELECT
   id,
   XMLELEMENT(
      NAME "info",
      "Book ", XMLELEMENT(NAME "title", title),
      " with identifier ", id, "."
   ) AS description
FROM books
```

id	description
1	<info> **Book <title>Red</title> with identifier 1.** </info>
…	…

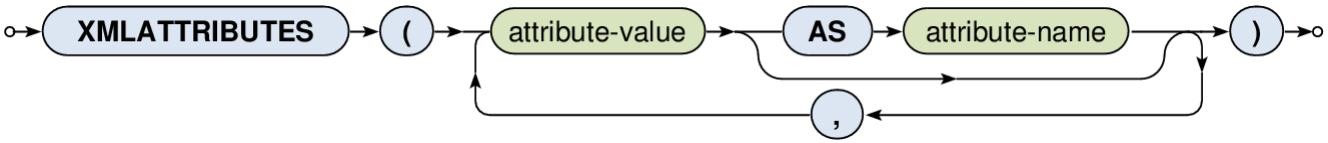
Elements: Example 4: Subqueries

```
SELECT
   id,
   XMLELEMENT(NAME "title", title) AS book,
   XMLELEMENT(
      NAME "language",
      (SELECT name FROM languages WHERE (code = language))
   ) AS description
FROM books
```

id	book	description
1	<title>Red</title>	<language>**English**</language>
…	…	…

Attributes

- XMLATTRIBUTES
  - **Creates a set of attributes**
  - Input: list of values
    - Each value must have an **explicit / implicit name**
      - It is used as a name for the given attribute
      - Implicit names can be derived, e.g., from column names
  - Output: XML value with a set of attribute nodes



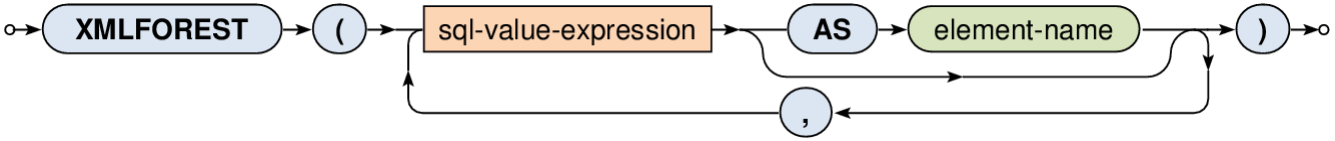
Attributes: Example

```
SELECT
   id,
   XMLELEMENT(NAME "book",
      XMLATTRIBUTES(
         language, catalogue AS "location"
      ),
      XMLELEMENT(NAME "title", title)
   ) AS book
FROM books
```

id	book
1	<book language="en" location="c1"> <title>Red</title> </book>
…	…

Element Sequences

- XMLFOREST
  - **Creates a sequence of XML elements**
    - **… by encapsulating provided values into elements**
  - Input: list of SQL values
    - Individual content expressions evaluated to `NULL` are ignored
    - If all the expressions are evaluated to `NULL`, then `NULL` is returned
    - Each content value must have an **explicit / implicit name**
      - It is used as a name for the given element
  - Output: XML value with a sequence of element nodes



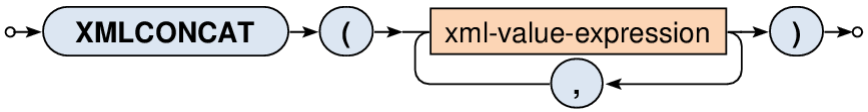
Element Sequences: Example

```
SELECT
   id,
   XMLFOREST(
      title, language, catalogue AS location
   ) AS book
FROM books
```

id	book
1	<title>Red</title> <language>en</language> <location>c1</location>
2	<title>Black</title> <location>c1</location>
...	...

Concatenation

- XMLCONCAT
  - **Creates a sequence of XML nodes**
    - **… by concatenating provided XML values**
  - Input: list of XML values
    - Individual content expressions evaluated to `NULL` are ignored
    - If all the expressions are evaluated to `NULL`, then `NULL` is returned
  - Output: XML value with a sequence of nodes



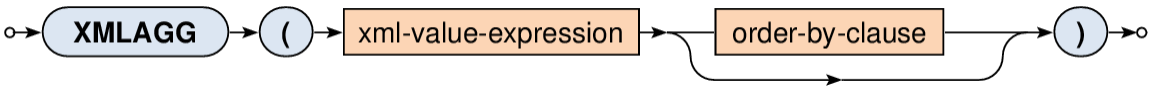
Concatenation: Example

```
SELECT
   id,
   XMLCONCAT(
      XMLELEMENT(NAME "book", title),
      details
   ) AS description
FROM books
```

id	description
1	<book>Red</book> <author>John</author> <author>Peter</author>
...	...

XML Aggregation

- XMLAGG
  - **Aggregates rows within a given super row**
    - I.e. acts as a standard aggregate function (like SUM, AVG, …)
  - Input: rows within a given aggregation group
    - These rows can first be optionally sorted (**ORDER BY**)
    - For each row an XML value is constructed as described
      - Individual constructed items evaluated to `NULL` values are ignored
    - All the generated XML values are then concatenated
      - If all the constructed items are evaluated to `NULL`, then `NULL` is returned
  - Output: XML value with a sequence of nodes



XML Aggregation: Example

```
SELECT
   catalogue,
   XMLAGG(
      XMLELEMENT(NAME "book", XMLATTRIBUTES(id),
         title)
      ORDER BY id
   ) AS list
FROM books
GROUP BY catalogue
```

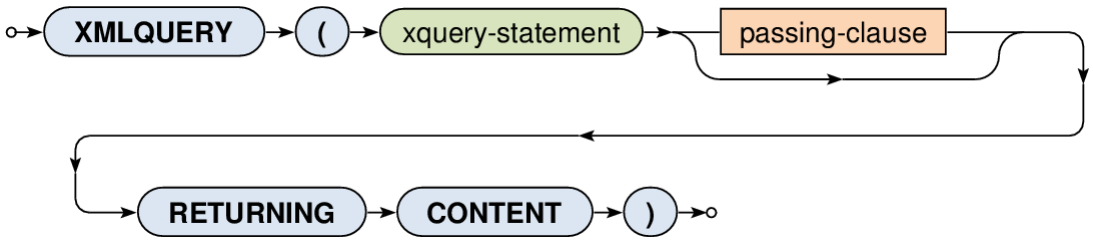
catalogue	list
c1	<book id="1">Red</book> <book id="2">Black</book>
c2	<book id="3">White</book>

Querying

- Query constructs
  - Based on XQuery language
  - **XMLQUERY** – returns query result
    - Usually used in SELECT clauses
  - **XMLTABLE** – decomposes query result into a table
    - Usually used in FROM clauses
  - **XMLEXISTS** – tests query result for non-emptiness
    - Usually used in WHERE clauses

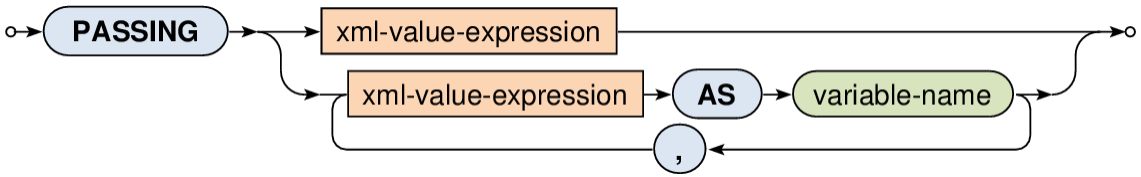
XQuery Statements

- XMLQUERY
  - **Evaluates an XQuery statement and returns its result**
  - Input:
    - XML values declared in an optional **PASSING** clause
  - Output: XML value



XQuery Statements

- XMLQUERY
  - Input data
    - When **only one input value** is specified…
      - its content is accessible via `/` within the XQuery expression
    - When **one or more named variables** are specified…
      - their content is accessible via `$variable-name/`



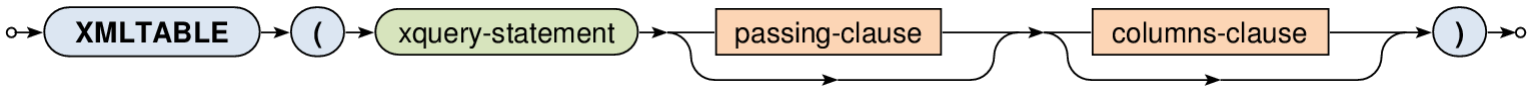
XQuery Statements: Example

```
SELECT
   id, title,
   XMLQUERY(
      "<authors>{ count($data/author) }</authors>"
      PASSING details AS data
      RETURNING CONTENT
   ) AS description
FROM books
```

id	title	description
1	Red	<authors>2</authors>
…	…	…

XML Tables

- XMLTABLE
  - **Decomposes an XQuery result into a virtual table**
  - Output:
    - When **COLUMNS** clause is specified…
      - Table containing the XQuery result is shredded into individual rows and columns according to the provided description
    - Otherwise…
      - Table with one row and one column with the XQuery result represented as an XML value



XML Tables: Example 1

```
SELECT
   id, title, result.*
FROM
   books,
   XMLTABLE(
      "<authors>{ count($data/author) }</authors>"
      PASSING books.details AS data
   ) AS result
```

id	title	result
1	Red	<authors>2</authors>
…	…	…

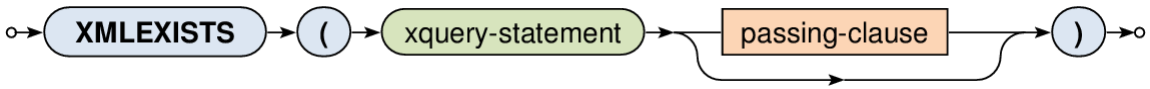
XML Tables: Example 2

```
SELECT
   id, title, result.count
FROM
   books,
   XMLTABLE(
      "<authors>{ count($data/author) }</authors>"
      PASSING books.details AS data
      COLUMNS
         count INTEGER PATH "authors/text()"
   ) AS result
```

id	title	count
1	Red	**2**
…	…	…

Exists Predicate

- XMLEXISTS
  - **Tests an XQuery statement result for non-emptiness**
  - Output: Boolean value
    - Returns `TRUE` for result sequences that are not empty
    - Otherwise `FALSE`



Exists Predicate: Example

```
SELECT books.*
FROM books
WHERE
   XMLEXISTS(
      "/author"
      PASSING details
   )
```

id	catalogue	title	details	language
1	c1	Red	<author>John</author> <author>Peter</author>	en
3	c2	White	<author>John</author>	en

Sedna

Introduction

- **Sedna**
  - http://www.sedna.org/
  - Free native XML database system
    - Open source under Apache License 2.0
    - Implemented in C/C++
    - Drivers for Java, C, PHP, Python, Ruby, Perl, C#, …
  - Platforms
    - Linux, Windows, Mac OS X, FreeBSD, Solaris

Introduction

- Basic features
  - **XQuery** language
    - SQL connection from XQuery
    - External functions implemented in C
  - Declarative node-level **updates**
  - **Indices** based on B-trees, **full-text** search indices
  - ACID **transactions**
  - Triggers
  - Incremental backup
  - Database security (users, roles, authentication)

Documents

- Document creation and removal
  - **`CREATE DOCUMENT`** `name`
  - **`DROP DOCUMENT`** `name`
- Bulk load
  - **`LOAD`** `location name`
    - Loads the specified file into a given standalone document
- Document retrieval
  - **`doc`**`($name)`

Collections

- **Collections**
  - Documents can be organized into collections
    - However, standalone documents are permitted as well
    - Each document belongs to at most one collection
- Management of collections
  - **`CREATE COLLECTION`** `name`
  - **`DROP COLLECTION`** `name`
  - **`RENAME COLLECTION`** `oldName` **`TO`** `newName`

Collections

- Document creation and removal
  - **`CREATE DOCUMENT`** `name` **`IN COLLECTION`** `col`
  - **`DROP DOCUMENT`** `name` **`IN COLLECTION`** `col`
- Bulk load
  - **`LOAD`** `location name col`
- Document retrieval
  - **`doc`**`($name, $col)`
    - Returns a given document from the specified collection
  - **`collection`**`($col)`
    - Returns a sequence of all documents from a given collection

Updates

- Insert
  - **UPDATE INSERT** source (**INTO** | **PRECEDING** | **FOLLOWING**) target
    - Inserts zero or more nodes into a designated position
- Delete
  - **UPDATE DELETE** expression
    - Removes target nodes including their descendants

Updates

- Replace
  - **`UPDATE REPLACE`** `$var` **`IN`** `source` **`WITH`** `expression`
    - Replaces nodes with a new sequence of zero or more nodes
- Rename
  - **`UPDATE RENAME`** `target` **`ON`** `name`
    - Changes a name of target nodes

Value Indices

- Index creation
  - **`CREATE INDEX`** `title`
    `ON` `path1` **`BY`** `path2` **`AS`** `type`
    - title – unique name of the index to be created
    - path1 – absolute path to nodes to be indexed
    - path2 – relative path to keys of such nodes
    - type – atomic type to which keys should be cast
- Index removal
  - **`DROP INDEX`** `title`

Value Indices

- Index usage
  - Existing indices are not exploited automatically!
    - Special functions have to be used instead
  - **`index-scan`**`($title, $value, $mode)`
    - Traverses the given index and returns a sequence of nodes matching the provided atomic value
    - $title – name of the index to be used
    - $mode – 'EQ', 'LT', 'GT', 'GE', 'LE'
  - **`index-scan-between`**`($title, $value1, $value2, $mode)`

Full-Text Indices

- Index creation
  - **`CREATE FULL-TEXT INDEX`** `title`
    `ON` `path` **`TYPE`** `type`
    `WITH OPTIONS` `options`
    - title – unique name of the index to be created
    - path – absolute path to nodes to be indexed
    - type – how to construct text representations of nodes
      - "xml" – XML representation of nodes
      - "string-value" – concatenation of all descendant text nodes
      - …
    - options
      - Backend index implementation, stemming language, …

Full-Text Indices

- Index removal
  - **`DROP FULL-TEXT INDEX`** `title`
- Index usage
  - **`ftindex-scan`**`($title, $query, $options)`
    - Scans the given full-text index and returns a sequence of items which satisfy the query
    - Allowed query constructs:
      - Phrases – e.g. `'word1 word2'`
      - Operators – e.g. `'word1 OR word2'`
      - Stemming – e.g. `'word~'`
      - Contains – e.g. `'element CONTAINS word'`

Metadata

- **Metadata**
  - Accessible as special system documents / collections
    - They can be queried, but not updated
    - Their names start with $
  - `$documents`
    - Document with a list of all documents
  - `$collections`
    - Document with a list of all collections
  - `$indexes`
    - Document with a list of all indexes
  - …

Java API

- Sessions
  - `SednaConnection c = DatabaseManager.getConnection( url, dbname, user, password );`
    - Establishes an authenticated connection with Sedna server
      - url – server name including an optional port (default 5050)

Java API

- Statements and results
  - `SednaStatement s = c.createStatement();`
    - Creates a new statement object
  - `s.execute(query);`
    - Executes a given query statement
  - `SednaSerializedResult r = s.getSerializedResult();`
    - Retrieves the result of the last executed query
  - `r.next();`
    - Fetches the next item from the result
    - Otherwise `null` is returned

Java API

- Documents
  - `s.loadDocument(inputStream, name);`
    `s.loadDocument(inputStream, name, col);`
    - Loads contents of a given standalone / collection document
- Transactions
  - `c.begin();`
  - `c.commit();`
  - `c.rollback();`
- Exceptions