

NDBI001: Query Languages I

<http://www.ksi.mff.cuni.cz/~svoboda/courses/221-NDBI001/>

Lecture

SQL/XML

Martin Svoboda

martin.svoboda@matfyz.cuni.cz

3. 1. 2023

Charles University, Faculty of Mathematics and Physics

Introduction

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

Example

- **Table: books**

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

- **Table: languages**

code	name
en	English
cs	Czech

Example

- Query

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

Example

- **Result**

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

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, ...
 - But we also allow fragments without exactly one root element
 - » This means that XML values may not be XML documents
 - NULL

Parsing XML Values

- XMLPARSE

- **Creates an XML value from a string**

- DOCUMENT – well-formed document with exactly one root
 - CONTENT – well-formed fragment

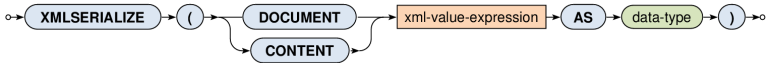


```
SELECT XMLPARSE (  
    DOCUMENT "<book><title>Red</title></book>"  
) AS result
```

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

Serializing XML Values

- XMLSERIALIZE
 - Exports an XML value to a string

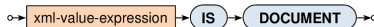


```
SELECT
  id, title,
  XMLSERIALIZE(CONTENT details AS VARCHAR(100)) AS export
FROM books
```

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`



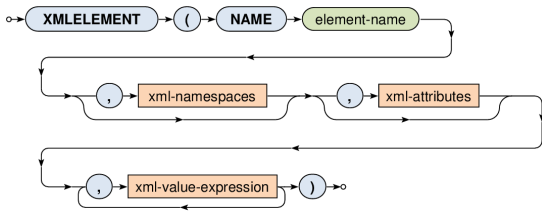
Constructors

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

Elements

- XMLEMENT

- **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>Green</book>
3	<book>Blue</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	<pre><book> <title>Red</title> <language>en</language> </book></pre>
...	...

Elements: Example 3: Mixed Content

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

id	description
1	<code><info></code> Book <code><title>Red</title></code> with identifier equal to 1. <code></info></code>
...	...

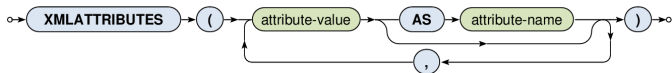
Elements: Example 4: Subqueries

```
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 attributes



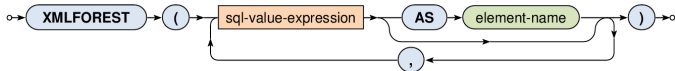
Attributes: Example

```
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
 - 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 elements



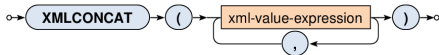
Element Sequences: Example

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

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

Concatenation

- XMLCONCAT
 - **Creates a sequence from a list of 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 values



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 super row**

- These rows can first be optionally sorted (**ORDER BY**)

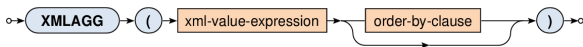
- For each row an XML value is generated as described

- Individual rows evaluated to `NULL` values are ignored

- All the generated XML values are then concatenated

- If all the rows are evaluated to `NULL`, then `NULL` is returned

- **Output: XML value with a sequence of items**



XML Aggregation: Example

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

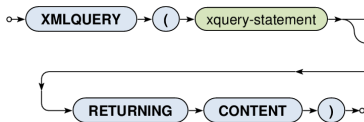
catalogue	list
c1	<book id="1">Red</book> <book id="2">Green</book>
c2	<book id="3">Blue</book>

Querying

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

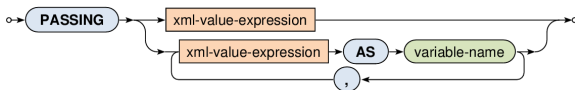


XQuery Statements

- XMLQUERY

- Input data

- When **only one input value** is specified...
 - its content is accessible via / inside the XQuery statement
 - 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**

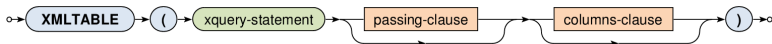
- Output:

- When **COLUMNS** clause is specified...

- Table containing the XQuery result being shredded into individual rows and columns according to the description

- Otherwise...

- Table containing the XQuery result being shredded into individual rows with only one XML value column



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>
...

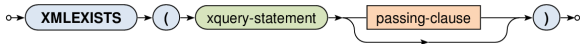
XML Tables: Example 2

```
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	Blue	<author>John</author>	en