

# ZTB: XML

Igor Wojnicki

Katedra Informatyki Stosowanej, Akademia Górniczo-Hutnicza w Krakowie

6 października 2016

# Problem

- XML.
- DB.
- DB  $\rightarrow$  XML.
- XML  $\rightarrow$  DB.
- XML  $\leftrightarrow$  DB.

I. Wojnicki, ZTB:XML

# Spis Treści

- 1 Intro
- 2 PostgreSQL
  - Wstęp
  - Eksport
  - Porównywanie własności
  - Mapowanie tabel i XML
  - Przetwarzanie XML
- 3 Natywne BD XML
  - EXist
  - BaseX
- 4 Wyszukiwanie i transformacje
  - XPath
  - XQuery

# Możliwości

- Eksport do XML.
- Typ danych XML.
- XPath do wyszukiwania.
- Integracja z SQL.

I. Wojnicki, ZTB:XML

# Jak użyć XML

Typ danych: xml

```
INSERT INTO test VALUES (  
    XMLPARSE (DOCUMENT '<doc>...</doc>'),...  
)  
  
SELECT XMLSERIALIZE (DOCUMENT data AS varchar)  
FROM test;
```

```
INSERT INTO test VALUES (  
    '<doc>...</doc>',...  
)
```

```
SELECT data FROM test;
```

Domyślnie przetwarzane jako CONTENT, a nie DOCUMENT – pozwala na las.

```
SET XML OPTION { DOCUMENT | CONTENT };
```

# Cechy

- Brak operatorów porównania.
- Do pobierania/indeksowania należy użyć:
  - rzutowania na tekst (`text`/`varchar`),
  - wyrażeń XPath.

I. Wojnicki, ZTB:XML

# Spis Treści

- 1 Intro
- 2 PostgreSQL
  - Wstęp
  - Eksport
  - Porównywanie własności
  - Mapowanie tabel i XML
  - Przetwarzanie XML
- 3 Natywne BD XML
  - EXist
  - BaseX
- 4 Wyszukiwanie i transformacje
  - XPath
  - XQuery

# Elementy I

Generacja element(ów) z wiersza danych.  
Rozwiązanie czołgowe.

```
SELECT '<klient id="' || idklienta || '">' || nazwa || '</klient>'  
FROM klienci;
```

I. Wojnicki, ZTB:XML



# Elementy II

Po prostu rozwiązanie.

```
xmlelement(name name
  [, xmlattributes(value [AS attname] [, ... ])]
  [, content, ...])
```

Przykłady:

- ```
SELECT xmlelement(name customer);
xmlelement
-----
<customer/>
```
- ```
SELECT xmlelement(name customer, 'dane klienta');
xmlelement
-----
<customer>dane klienta</customer>
```

# Elementy III

- ```
SELECT xmlelement(name customer, 'dane','klienta');
           xmlelement
```

---

```
<customer>daneklienta</customer>
```
- ```
SELECT xmlelement(name customer, xmlelement(name name,'nazwa klienta'));
           xmlelement
```

---

```
<customer><name>nazwa klienta</name></customer>
```
- ```
SELECT xmlelement(name customer,
           xmlelement(name name,'nazwa klienta'),
           xmlelement(name address,'adres klienta'));
           xmlelement
```

---

```
<customer>
  <name>nazwa klienta</name>
  <address>adres klienta</address>
</customer>
```

# Elementy IV

- SELECT

```
xmlelement(name customer,  
  xmlelement(name name,  
    xmlelement(name fn,'imie'),  
    xmlelement(name ln,'nazwisko')  
  ),  
xmlelement(name address,'adres klienta')));
```

```
xmlelement
```

```
-----  
<customer>  
  <name>  
    <fn>imie</fn>  
    <ln>nazwisko</ln>  
  </name>  
  <address>adres klienta</address>  
</customer>
```

# Elementy V

- SELECT

```
xmlelement(name customer,  
  xmlattributes('a10' as id),  
  'dane klienta');  
          xmlelement
```

-----  
<customer id="a10">dane klienta</customer>

- SELECT

```
xmlelement(name customer,  
  xmlattributes('a10' as id, 'a11' as ref), 'dane klienta');  
          xmlelement
```

-----  
<customer id="a10" ref="a11">dane klienta</customer>

# Elementy VI

- Dane z tabeli...

```
SELECT
  xmlelement(name customer, xmlelement(name name,nazwa))
FROM klienci;
```

xmlelement

```
-----
<customer><name>Magdalena Sowinska</name></customer>
<customer><name>Malgorzata Babik</name></customer>
```

- SELECT

```
  xmlelement(name customer,
    xmlattributes(idklienta as id),
    xmlelement(name name,nazwa))
FROM klienci;
```

xmlelement

```
-----
<customer id="msowins"><name>Magdalena Sowinska</name></customer>
<customer id="mbabik"><name>Malgorzata Babik</name></customer>
```

# Elementy VII

- Atrybuty na podstawie kolumn

```
SELECT xmlelement(name customer,  
    xmlattributes(idklienta), nazwa)  
FROM klienci;
```

-----  
xmlelement

```
<customer idklienta="msowins">Magdalena Sowinska</customer>  
<customer idklienta="mbabik">Malgorzata Babik</customer>
```

- Znaki niedozwolone

```
SELECT xmlelement(name customer, '<>&');  
xmlelement
```

-----  
<customer>&lt;&gt;&amp;</customer>

# Wiele elementów I

```
xmlforest(content [AS name] [, ...])
```

## Przykłady:

- `SELECT xmlforest('Nowak' as customer, 'Wielicka 25' as address);`  
`xmlforest`

```
-----  
<customer>Nowak</customer><address>Wielicka 25</address>
```

- `SELECT xmlforest(idklienta, nazwa) FROM klienci;`  
`xmlforest`

```
-----  
<idklienta>msowins</idklienta><nazwa>Magdalena Sowinska</nazwa>  
<idklienta>mbabik</idklienta><nazwa>Malgorzata Babik</nazwa>
```

## Wiele elementów II

- zwykle w połączeniu z `xmlelement()`:

```
SELECT xmlelement(name customer,xmforest(idklienta, nazwa)) FROM klienci;  
xmlelement
```

---

```
<customer>  
  <idklienta>msowins</idklienta>  
  <nazwa>Magdalena Sowinska</nazwa>  
</customer>  
<customer>  
  <idklienta>mbabik</idklienta>  
  <nazwa>Malgorzata Babik</nazwa>  
</customer>
```



# Root

```
xmlroot(xml, version text | no value [, standalone yes|no|no value])
```

## Przykład

```
SELECT xmlroot(xmlelement(name test,'content'), version 1.1,standalone yes);  
                xmlroot
```

```
-----  
<?xml version="1.1" standalone="yes"?><test>content</test>
```

Wartości: `version` i `standalone` modyfikują dane w elemencie `<?xml>`.

# Łączenie

```
xmlconcat(xml[, ...])
```

## Przykład

```
SELECT xmlconcat('<abc/>', '<bar>foo</bar>');
```

```
xmlconcat
```

```
-----  
<abc/><bar>foo</bar>
```

# Agregacja XML

Łączy dane XML z wielu wierszy.

```
xmlagg(xml)
```

Problem: generacja całego dokumentu za pomocą 1-go zapytania.

```
SELECT xmlroot(xmlelement(name customer,nazwa), version 1.1) FROM klienci;
```

xmlroot

---

```
<?xml version="1.1"?><customer>Magdalena Sowinska</customer>  
<?xml version="1.1"?><customer>Malgorzata Babik</customer>
```

Rozwiązanie:

```
SELECT xmlroot(xmlelement(name data,  
    xmlagg(xmlelement(name customer,nazwa))),  
    version 1.1) FROM klienci;
```

```
<?xml version="1.1"?>  
<data>  
  <customer>Magdalena Sowinska</customer>  
  <customer>Malgorzata Babik</customer>  
  ...  
</data>
```

# XML Processing Instructions

```
xmlpi(name target [, content])
```

Przykład:

```
SELECT xmlpi(  
    name "xml-stylesheet",  
    'type="text/xsl" href="style.xsl"',  
);  
  
          xmlpi  
-----  
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

# Komentarze

```
xmlcomment(text)
```

Przykład:

```
SELECT xmlcomment('dane orientacyjne');
       xmlcomment
-----
<!--dane orientacyjne-->
```

# Spis Treści

- 1 Intro
- 2 PostgreSQL
  - Wstęp
  - Eksport
  - **Porównywanie własności**
  - Mapowanie tabel i XML
  - Przetwarzanie XML
- 3 Natywne BD XML
  - EXist
  - BaseX
- 4 Wyszukiwanie i transformacje
  - XPath
  - XQuery

# Czy jest dokumentem?

```
xml IS DOCUMENT
```

Przykład:

```
SELECT '<a></a>' IS DOCUMENT;  
?column?
```

```
-----  
t
```

```
SELECT '<a></a><b></b>' IS DOCUMENT;  
?column?
```

```
-----  
f
```

I. Wojnicki, ZTB:XML

# Czy jest poprawnie sformułowany

Czy podany tekst jest dokumentem XML? Czy rzutowanie na typ XML będzie działać.

```
xml_is_well_formed(text)
```

Przykład:

```
SELECT xml_is_well_formed_document(  
    '<pg:foo xmlns:pg="http://postgresql.org/stuff">bar</my:foo>');  
xml_is_well_formed_document
```

f



# Czy dane istnieją?

Czy wyrażenie *XPath* zwróci jakieś dane?

```
XMLEXISTS(text PASSING [BY REF] xml [BY REF])
```

Przykład:

```
SELECT xmlexists('//town[text() = ''Toronto'']',  
  PASSING BY REF  
  '<towns><town>Toronto</town><town>Ottawa</town></towns>');
```

```
xmlexists
```

```
-----  
t
```

Nawiasy kwadratowe w XPath oznaczają użycie predykatu; określenie warunku.

# Spis Treści

- 1 Intro
- 2 PostgreSQL
  - Wstęp
  - Eksport
  - Porównywanie własności
  - **Mapowanie tabel i XML**
  - Przetwarzanie XML
- 3 Natywne BD XML
  - EXist
  - BaseX
- 4 Wyszukiwanie i transformacje
  - XPath
  - XQuery

# Tabele, zapytania i XML I

```
table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text)
query_to_xml(query text, nulls boolean, tableforest boolean, targetns text)
cursor_to_xml(cursor refcursor, count int, nulls boolean,
              tableforest boolean, targetns text)
```

**nulls** wartości null reprezentowane przez element  
`<nazwaKolumny xsi:nil="true"/>`

**tableforest** każdy wiersz osobnym elementem,  
**targetns** namespace.

# Tabele, zapytania i XML II

Przykład:

```
SELECT table_to_xml('odbiorcy', false, false, '');
           table_to_xml
```

```
-----
<odbiorcy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
<row>+
  <idodbiorcy>1</idodbiorcy>+
  <nazwa>Slawomir Zegane</nazwa>+
  <miasto>Krakow</miasto>+
  <kod>30-059</kod>+
  <adres>Al A. Mickiewicza 4/3</adres>+
</row>+
<row>+
  <idodbiorcy>2</idodbiorcy>+
  <nazwa>Dorota Pszczolka</nazwa>+
  <miasto>Slomniki</miasto>+
  <kod>32-090</kod>+
  <adres>ul. Wiosenna 8</adres>+
</row>+
...+
</odbiorcy>+
```

# Tabele, zapytania i XML III

Przykład:

```
SELECT table_to_xml('odbiorcy', false, true, '');
           table_to_xml
```

```
-----
<odbiorcy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
  <idodbiorcy>1</idodbiorcy>                                     +
  <nazwa>Slawomir Zegane</nazwa>                                 +
  <miasto>Krakow</miasto>                                       +
  <kod>30-059</kod>   +
  <adres>Al A. Mickiewicza 4/3</adres>                             +
</odbiorcy>  +
<odbiorcy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
  <idodbiorcy>2</idodbiorcy>                                     +
  <nazwa>Dorota Pszczolka</nazwa>                                 +
  <miasto>Slomniki</miasto>                                       +
  <kod>32-090</kod>   +
  <adres>ul. Wiosenna 8</adres>                                   +
</odbiorcy>  +
```

# Tabele, zapytania i XML IV

Przykład:

```
SELECT query_to_xml('SELECT idodbiorcy,nazwa FROM odbiorcy',  
                    false, false, '');  
                    query_to_xml
```

```
-----  
<table xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+  
+  
<row>+  
  <idodbiorcy>1</idodbiorcy>+  
  <nazwa>Slawomir Zegane</nazwa>+  
</row>+  
+  
<row>+  
  <idodbiorcy>2</idodbiorcy>+  
  <nazwa>Dorota Pszczolka</nazwa>+  
</row>+  
...
```

# Schematy XML i schematy bazy I

```
table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean,  
                  targetns text)  
query_to_xmlschema(query text, nulls boolean, tableforest boolean,  
                  targetns text)  
cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean,  
                  targetns text)
```

# Schematy XML i schematy bazy II

Przykład:

```
SELECT table_to_xmlschema('odbiorcy', false, false, '');
                                table_to_xmlschema
```

```
-----
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:simpleType name="INTEGER">
    <xsd:restriction base="xsd:int">
      <xsd:maxInclusive value="2147483647"/>
      <xsd:minInclusive value="-2147483648"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="VARCHAR">
    <xsd:restriction base="xsd:string">
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="CHAR">
    <xsd:restriction base="xsd:string">
```



# Schematy XML i schematy bazy III

```

</xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="RowType.wojnicki.public.odbiorcy">
  <xsd:sequence>
    <xsd:element name="idodbiorcy" type="INTEGER" minOccurs="0"></xsd:element>
    <xsd:element name="nazwa" type="VARCHAR" minOccurs="0"></xsd:element>
    <xsd:element name="miasto" type="VARCHAR" minOccurs="0"></xsd:element>
    <xsd:element name="kod" type="CHAR" minOccurs="0"></xsd:element>
    <xsd:element name="adres" type="VARCHAR" minOccurs="0"></xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TableType.wojnicki.public.odbiorcy">
  <xsd:sequence>
    <xsd:element name="row" type="RowType.wojnicki.public.odbiorcy"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="odbiorcy" type="TableType.wojnicki.public.odbiorcy"/>

```

# Schematy XML i schematy bazy IV

```
</xsd:schema>
```

I. Wojnicki, ZTB:XML

# I jeszcze kilka innych funkcji

```
table_to_xml_and_xmlschema(tbl regclass, nulls boolean, tableforest boolean,  
                             targetns text)
```

```
query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean,  
                             targetns text)
```

```
schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text)
```

```
schema_to_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)
```

```
schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean,  
                             targetns text)
```

```
database_to_xml(nulls boolean, tableforest boolean, targetns text)
```

```
database_to_xmlschema(nulls boolean, tableforest boolean, targetns text)
```

```
database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text)
```

# Spis Treści

- 1 Intro
- 2 PostgreSQL
  - Wstęp
  - Eksport
  - Porównywanie własności
  - Mapowanie tabel i XML
  - **Przetwarzanie XML**
- 3 Natywne BD XML
  - EXist
  - BaseX
- 4 Wyszukiwanie i transformacje
  - XPath
  - XQuery

# Przetwarzanie danych XML

```
xpath(xpath, xml [, nsarray])
```

Zwraca tablice wybranych elementów, zgodnych z wyrażeniem *XPath*.

```
xpath_exists(xpath, xml [, nsarray])
```

Przykład:

```
SELECT id, name , (xpath('//comments/en/text()', description))[1]
FROM printer;
```

id	name	xpath
1	Canon-BJ-100.xml	Seems more or less compatible to the BJ-200.
2	Canon-BJ-10e.xml	Note that some report it working properly in epson mode only.

# Wyszukiwanie i indeksowanie XML

```
SELECT ... FROM ...  
WHERE (xpath('//element/text()', description))[1]::text LIKE 'value'
```

```
CREATE INDEX idx_xml_printer_des ON printer USING btree (  
  (xpath('//element/text()', description))[1]::text  
);
```

```
select * from (select unnest(ARRAY[1,5,3])) as a;  
create index xmltest_idx on xmltest  ( unnest(cast( (xpath('/d/text()','xml')) as text
```

# Spis Treści

- 1 Intro
- 2 PostgreSQL
  - Wstęp
  - Eksport
  - Porównywanie własności
  - Mapowanie tabel i XML
  - Przetwarzanie XML
- 3 Natywne BD XML
  - EXist
  - BaseX
- 4 Wyszukiwanie i transformacje
  - XPath
  - XQuery

# Spis Treści

- 1 Intro
- 2 PostgreSQL
  - Wstęp
  - Eksport
  - Porównywanie własności
  - Mapowanie tabel i XML
  - Przetwarzanie XML
- 3 Natywne BD XML
  - EXist
  - BaseX
- 4 Wyszukiwanie i transformacje
  - XPath
  - XQuery



# Spis Treści

- 1 Intro
- 2 PostgreSQL
  - Wstęp
  - Eksport
  - Porównywanie własności
  - Mapowanie tabel i XML
  - Przetwarzanie XML
- 3 Natywne BD XML
  - EXist
  - BaseX
- 4 Wyszukiwanie i transformacje
  - XPath
  - XQuery

# XPath

<http://www.w3.org/TR/xpath/> Wyszukiwanie informacji w dokumentach XML.

- Wyrażenia do wyszukiwania (zbiorów) węzłów,
  - podobne do ścieżek w systemach plików.
- Wbudowane funkcje: ekstrakcja i konwersja danych, porównywanie itp.

# Podstawy XPath

Węzłem jest:

- element,
- atrybut,
- tekst,
- przestrzeń nazw,
- instrukcja przetwarzania (PI, processing instruction),
- komentarz,
- dokument.

# Wybieranie węzłów

nazwa_węzła	węzeł o określonej nazwie
/	korzeń
//	lokalizacja nieistotna
.	aktualny węzeł
..	rodzic
@	atrybut

I. Wojnicki, ZTB:XML

# Predykaty stosowalności

Występują w nawiasach kwadratowych, wykorzystują funkcje.

```
//pcl[@level]          elementy pcl mające atrybut level
//node()[@level]      dowolne elementy mające atrybut level
//printer[.//@level]  elementy printer zawierające element z atr. level
//dpi/node()[last()]  ostatnie elementy w elementach dpi
//dpi/*
```

# Wiele ścieżek

```
//dpi | //make  
//city[population>8000000] [name="Moscow"]/(name | population)
```

I. Wojnicki, ZTB:XML

# XPath

```
//resolution[dpi/x>2000]  
//model[../resolution/dpi/x>2000]  
//model[resolution/dpi/x>2000]
```

podciągi, zobacz: <http://www.w3.org/TR/xpath-functions>

```
//printer[fn:starts-with(make,"B")]  
//printer[fn:matches(make,"B")]
```

# Spis Treści

- 1 Intro
- 2 PostgreSQL
  - Wstęp
  - Eksport
  - Porównywanie własności
  - Mapowanie tabel i XML
  - Przetwarzanie XML
- 3 Natywne BD XML
  - EXist
  - BaseX
- 4 Wyszukiwanie i transformacje
  - XPath
  - XQuery



# XQuery

```
for $p in //printer
where $p/mechanism/color
return ($p/make, $p/model)
```

```
for $p in //printer
where $p/mechanism/color
return <p> { $p/make, $p/model } </p>
```

```
for $p in //printer
where $p/mechanism[not(color)]
return <p> { $p/make, $p/model } </p>
```

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