Technologie SQL/XML

SQL/XML - typ dat xml, integrace xpath

Jiří Měska (jiri.meska@gmail.com)

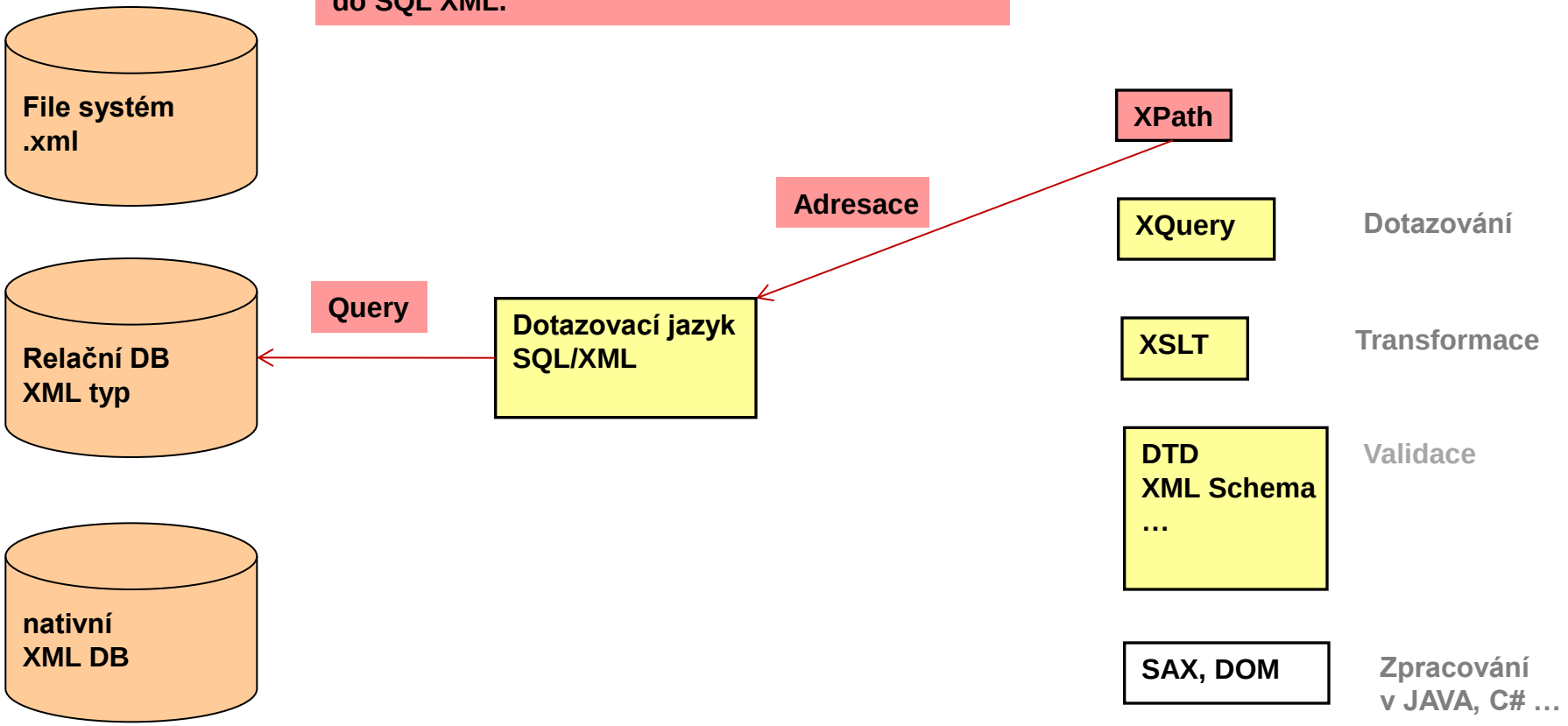
MIB008, Datové a procesní modely

MFF UK Praha, 2020

Perzistence XML dat

Zpracování XML dat

V této prezentaci integrujeme technologii XPath do SQL XML.



XML Typ dat

xml je datovým typem, který je rovnocenný ostatním typům jako jako varchar(), int, double ….

popis varchar(64) not null,

popisxml xml not null,

cenaKus double precision

hodnota typu xml je tvořena XML obsahem, který může být

- XML dokument (document) – to co chápeme dle standardu jako XML dokument
- XML fragment (content) – obecně část XML obsahu, tvořená množinou elementů

Pro vytvoření/serializaci hodnoty typu xml jsou k dispozici funkce:

- xmlParse() ... vytvoření hodnoty typu xml (vnitřní reprezentace) z textového řetězce
- xmlSerialize() ... vytvoření (serializace) textového řetězce z XML hodnoty typu xml (vnitřní reprezentace)

Pomocí publikačních funkcí jsme obvykle vytvářeli: _____ ?

Jaká publikační funkce vytváří XML dokument na výstupu: _____ ?

XML Typ dat – XMLParse()

Pro vytvoření hodnoty typu xml je třeba použít funkci xmlparse:

XMLPARSE ({ DOCUMENT | CONTENT } value)

Příklad 1: Parsování XML Dokumentu

```
XMLPARSE (DOCUMENT '<?xml
    version="1.0"?><book><title>Manual</title><chapter>...</chapter></book>')
```

Příklad 2: Parsování XML fragmentu

```
XMLPARSE (CONTENT 'abc<foo>bar</foo><bar>foo</bar>')
```

Poznámka:

XML typ nevaliduje obsah XML ani vzhledem k DTD ani k XML schématu.

XML Typ dat – vytvoření příkazem INSERT

V tabulce ZboziKatalog najdete sloupec popisxml, který je typu xml.

V tabulce ZboziKatalog najdete sloupec popisjson, který je typu jsonb. Zatím ignorujte.

Při vládání hodnoty je možné použít funkci xmlparse().

XML string, který je argumentem pro xmlparse() nesmí obsahovat RETURN!

Může proběhnout implicitní konverze podle: SET xmloption TO { DOCUMENT | CONTENT };

```
create table ZboziKatalog
( idZbozi int not null primary key,
  popisxml xml null,
  popisjson jsonb null
 );
```

```
insert into ZboziKatalog  values(
        5001,
        xmlparse( DOCUMENT
'<?xml version="1.0" encoding="UTF-8"?>
<zbozi part_number="RUK-1-1-1">
<jmeno>zimní rukavice Elvis</jmeno>
<material>vlna</material>
<zaruka>12</zaruka>
<dostupnost>na objednávku do 3 dnů</dostupnost>
<popis>vlněné zimní rukavice prstové</popis>
</zbozi>'),  null);
```

XML Typ dat – aktualizace hodnoty příkazem UPDATE

Sloupec popisxml lze aktualizovat standardně pomocí UPDATE

Soubor: **xmlupdatezbozi1.sql**

```
update ZboziKatalog
set popisxml = XMLPARSE( content  '<?xml version="1.0" encoding="UTF-8"?>
<zbozi part_number="RUK-1-1-1">
<jmeno>zimní rukavice Elvis1</jmeno><material>vlna</material>
<zaruka>12</zaruka><dostupnost>na objednávku do 3 dnů</dostupnost>
<popis>vlněné zimní rukavice prstové</popis><cena>491</cena></zbozi>')
where idzboziKatalog = 5001;
```

XML Typ dat – XMLSerialize()

Inverzní operací k XMLParse je XMLSerialize():

XMLSerialize ({ DOCUMENT | CONTENT } value AS type)

Funkce vytváří hodnoty typu „type“ z XML typu dat, kterými může být:

varchar, text

Příklad 1: Serializace XML dokumentu (výstup vlevo s XML deklarací)

SELECT xmlserialize(document popisxml as text) from zboziKatalog;

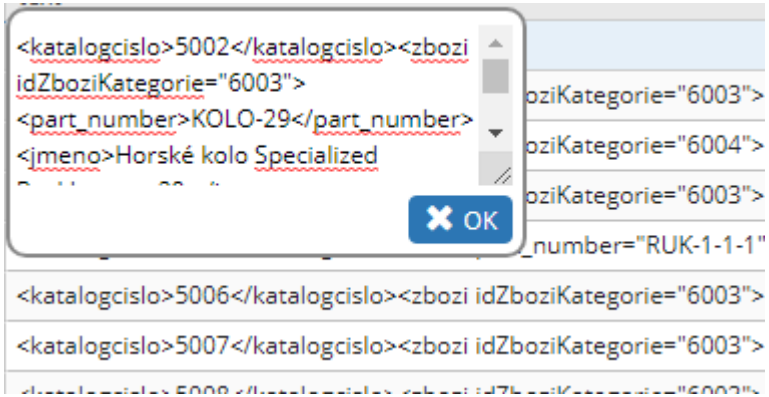
Příklad 2: Serializace XML fragmentu (výstup vpravo bez XML deklarace)

SELECT xmlserialize(content xmlconcat(xmlelement(name KatalogCislo, idZboziKatalog), popisxml) as text) from ZboziKatalog;

Výstupem je vždy pro každou řádku textová hodnota XML dokumentu

"<?xml version="1.0" encoding="UTF-8"?><zbozi part_number="KOLO-29"><jmeno>Horské kolo Specialized opper 29 </jmeno><zaruka>24</zaruka><dostupnost>skladem </dostupnost><velkosti><velkost>XL</velkost><velkost>L</velkost></velkosti><popis>Specialized Rockhoppe (...)"

...



XML Typ dat – export XML pomocí publikačních funkcí

Sloupec popisxml lze zahrnout do exportu jako obsah elementu nabídka_zbozi.

Soubor: **xmlupdatezbozi2.sql**

```
SELECT XMLRoot(
        XMLElement( name nabidka,
        XMLAgg( XMLElement(name nabidka_zbozi, popisxml))),
        VERSION '1.0', STANDALONE YES)
  AS xmlroot from zboziKatalog;
```

```
SELECT XMLRoot(
        XMLElement( name nabidka,
        XMLAgg( popisxml)),
        VERSION '1.0', STANDALONE YES)
  AS xmlroot from zboziKatalog;
```

Vyzkoušet na cvičení.

Nutno uložit jako UTF-8.

Při exportu z csv je nutno nahradit dvojí apostrofy u atributů za jednoduché.

XML predicates

XML predikáty:

- xml IS DOCUMENT
- XMLEXIST(xpath)
- xml_is_well_formed(text),
- xml_is_well_formed_document(text),
- xml_is_well_formed_context(text) (postgres nepodporuje)

.

XML predicate: is document

Syntax: **xml is document**

Sémantika:

Predicate is document vraci true, jestliže XML obsah je XML document

Predicate is document vraci false, jestliže XML obsah je null nebo není XML document, jedná se o fragment (content)

Výpis zboží, kde je k dispozici popisxml jako xml dokument.

Soubor: **xmlpredicate1.sql**

```
select * from zboziKatalog where popisxml is document
```

```
5001; "<zbozi part_number="RUK-1-1-1"><jmeno>zimní rukavice
Elvis</jmeno><material>vlna</material><zaruka>12</zaruka><dostupnost>na
objednávku do 3 dnů</dostupnost><popis>vlněné zimní rukavice
prstové</popis></zbozi>";490

5002;"závodní kolo";"<zbozi part_number="KOLO-29"><jmeno>Horské kolo Specialized
opper 29
</jmeno><zaruka>24</zaruka><dostupnost>skladem</dostupnost><velkosti><velkost>
XL</velkost><velkost>L</velkost></velkosti><popis>Specialized Rockhopper 29 v
černo-bílém provedení, jako spo (...)";null

...
```

XML predicate/function: xmlexists

Syntax: **XMLEXISTS(text PASSING [BY REF] xml [BY REF])**

Sémantika: Predikát vrací true jestliže XPath výraz (prvý parametr) vrací nějaký uzel (element, atribut atd.,). V opačném případe vrací false**.** By ref je z kompatibilních důvodů.

Výpis řádků zboží, kde v xml sloupci popisxml se vyskytuje element cena.

Podmínka je zadána xpath: '//cena'

Soubor: **xmlpredicate2.sql**

```
select * from zboziKatalog where xmlexists( '//cena' passing by ref popisxml)

select * from zboziKatalog where xmlexists( '//cena' passing popisxml)
```

```
"<zbozi part_number="KOLO-29"><jmeno>Horské kolo Specialized opper 29
</jmeno><zaruka>24</zaruka><dostupnost>skladem</dostupnost><velkosti><velkost>
XL</velkost><velkost>L</velkost></velkosti><popis>Specialized Rockhopper 29 v
černo-bílém provedení, jako sportovně-rekreační a nejdostupnější varianta tohoto
modelu na 29“ velkých kolech je postavena na hliníkovém rámu, jenž nabídne ...
</popis><barvy><barva>černo/bílá</barva></barvy><cena>14867</cena></zbozi>„

...
```

XML predicate/function: xmlexists

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

SQL/XML standard předepisuje jako argument **XQuery**, PostgreSQL podporuje pouze **XPath.**

Příklad: Zobrazení informace o zboží za předpokladu, že existuje v jeho xmlpopis element zboží s cenou větší než 1000,-

```
select * from zboziKatalog
where xmlexists('//cena[text() > 1000 ]' PASSING popisxml );
```

<zbozi idZboziKategorie="6003"><part_number>STAN-2</part_number><jmeno>Stan Husky Felen 2-3 - červený (2-3 osoby)</jmeno><cena>5630</cena><mena>Kč</mena><zaruka>24</zaruka><dostupnost>skladem</dostupnost><velikosti></velikosti><popis>Felen 2-3 je novinka v produkci Husky a nahrazuje starý model Flare. Použili jsme velmi úspěšnou konstrukci stanu Felen 3-4, protože nabízí velmi prostornou předsíňku a vyzkoušený systém fixace nosné konstrukce, která kombinuje textilní tunýlky a uchycení pomocí přezek. ... </popis><barvy><barva>tmavě zelený</barva><barva>béžový</barva></barvy><materialy></materialy><vaha></vaha></zbozi>

XML predicate/function: xml_is_well_formed_document

Syntax: **xml_is_well_formed_document(text)**

Sémantika: Predikát vrací true jestliže argumentem je XML dokument podle standardu XML. Jinak vrací false.

```
SELECT xml_is_well_formed_document('<>');

xml_is_well_formed_document

false

(1 row)
```

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

xml_is_well_formed_document

true

 (1 row)
```

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

xml_is_well_formed_document

false

 (1 row)
```

XML predicate: xml_is_well_formed_content

Syntax: **xml_is_well_formed (text)**

Sémantika: Predikát vrací true jestliže argumentem je XML content. Jinak vrací false.

```
SELECT xml_is_well_formed ('<>');
xml_is_well_formedt
false
(1 row)
```

```
SELECT xml_is_well_formed('<pg:foo xmlns:pg="http://postgresql.org/stuff">bar</pg:foo><a></a>');
xml_is_well_formed
true
 (1 row)
```

XPATH

XML funkce:

- xpath

XML/SQL: Funkce xpath()

Syntax: **path(xpath, xml [, nsarray])**

Funkce vyhodnocuje xpath výraz proti xml, které musí být well formed XML dokument.

Sekvence se vrací jako množina hodnot uzavřená v {}

Příklad: Výběr ceny z popisxml, za předpokladu, že je větší než 1000,-

Soubor: xmlxpath.sql

```
select xpath('//cena[text() > 1000 ]/text()', popisxml )
from zboziKatalog;
```

xpath xml[]
{14867}
{}
{1100...
{}
{}
{}
{}
{5630}
{10590}
{7500}
{2500}
{1900}
{1249}
{}
{}
{}

XML/SQL: Funkce xpath()

Příklad: Zobrazte prodejni cenu zbozi z tabulky ObjednavkaPolozka a současně odpovídající katalogovou cenu z tabulky ZboziKatalog. Jako katalogovou cenu zobrazte hodnotu prvního elementu <cena> z popisxml.

```
select
cenaKus as "prodejniCena",
(xpath('//cena[text()]/text()', popisxml ))[1] as "katalogovaCena"
from
ObjednavkaPolozka join zboziKatalog
using (idZboziKatalog);
```

prodejniCena double precision	katalogovaCena xml
490	491
2390	7500
79	1
95000	14867
5200	199
1000	879
1000	879
1050	449
1230	449
10	166
6700	5630
12000	10590
7400	7500
100000	110000
1000	879

Mapovací pravidla

Mapování tabulek do XML

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)

Návratovou hodnotou funkcí je xml.

table_to_xml maps the content of the named table, passed as parameter tbl. The regclass type accepts strings identifying tables using the usual notation, including optional schema qualifications and double quotes.

query_to_xml executes the query whose text is passed as parameter query and maps the result set.

cursor_to_xml fetches the indicated number of rows from the cursor specified by the parameter cursor. This variant is recommended if large tables have to be mapped, because the result value is built up in memory by each function.

Mapovací pravidla

Parametry:

nulls boolean … pro sloupec s null hodnotou

true … vygeneruje se v XML dokumentu <vekobchodnika xsi:nil="true"/>

false … do XML dokumentu se sloupce negeneruje vůbec

tableforest boolean …

true … množina dokumentů - řádků (forest) <row>

true … dokument se zastřešujícím elementem <table> obsahující <row>

targetns …. jméno cílového jmenného prostoru

Mapování: table_to_xml

Funkce umožňuje export databázové tabulky do xml dokumentu.

Každý řádek exportované tabulky je zabalen do elementu <row>, jednotlivé sloupce do elementů s jmény odpovídajícími sloupcům tabulky.

```
select table_to_xml('zboziKatalog', true, false, '');
```

```
"<zbozi xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance"">
<row>
        <idzbozi>5001</idzbozi>
        <popis>zimní rukavice</popis>
       <cenakus>490</cenakus>
</row>
<row>
       <idzbozi>5002</idzbozi>
       <popis>závodní kolo</popis>
       <cenakus>95000</cenakus>
</row>
...
```

Mapování: query_to_xml

Funkce umožňuje export výsledku SQL dotazu do XML dokumentu.

Soubor: **selecttoxml.sql**

```
SELECT QUERY_TO_XML('SELECT datum,
jmenoObchodnika,
(select sum(mnozstviKus*cenaKus) from ObjednavkaPolozka where
    objednavkaPolozka.idObjednavka= objednavka.idObjednavka) as cenaCelkem
FROM objednavka join obchodnik using(idObchodnik)',
TRUE, FALSE, '');
```

```
<table xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<row>
  <datum>2006-10-03</datum>
  <jmenoobchodnika>Nhac</jmenoobchodnika>
  <cenacelkem>2959</cenacelkem>
</row>
…
```

Mapovací pravidla

Mapování tabulek do XML Schematu

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)

Pokud chceme získat schéma k vygenerovanému XML předchozími funkcemi, je třeba použít stejných parametrů.

Společný export XML a XML schématu:

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)

Pro export celého schématu nebo databáze:

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)

Mapování: table_to_xmlschema

Funkce vygeneruje XML Schema pro zadanou tabulku

Soubor: tabletoschema2.sql

```
select table_to_xmlschema('Obchodnik', true, false, 'http://www.silicum.cz');
```

```
<xsd:schema  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.silicum.cz"  elementFormDefault="qualified">

….

<xsd:complexType name="RowType.cviceni_x0020_10.public.obchodnik">
  <xsd:sequence>
    <xsd:element name="idobchodnik" type="INTEGER" nillable="true"></xsd:element>
    <xsd:element name="idmanager" type="INTEGER" nillable="true"></xsd:element>
    <xsd:element name="jmenoobchodnika" type="VARCHAR" nillable="true"></xsd:element>
    <xsd:element name="idadresa" type="INTEGER" nillable="true"></xsd:element>
    <xsd:element name="idregion" type="INTEGER" nillable="true"></xsd:element>
    <xsd:element name="provize" type="DOUBLE" nillable="true"></xsd:element>
    <xsd:element name="vekobchodnika" type="INTEGER" nillable="true"></xsd:element>
  </xsd:sequence>
</xsd:complexType>….
```


Thanks