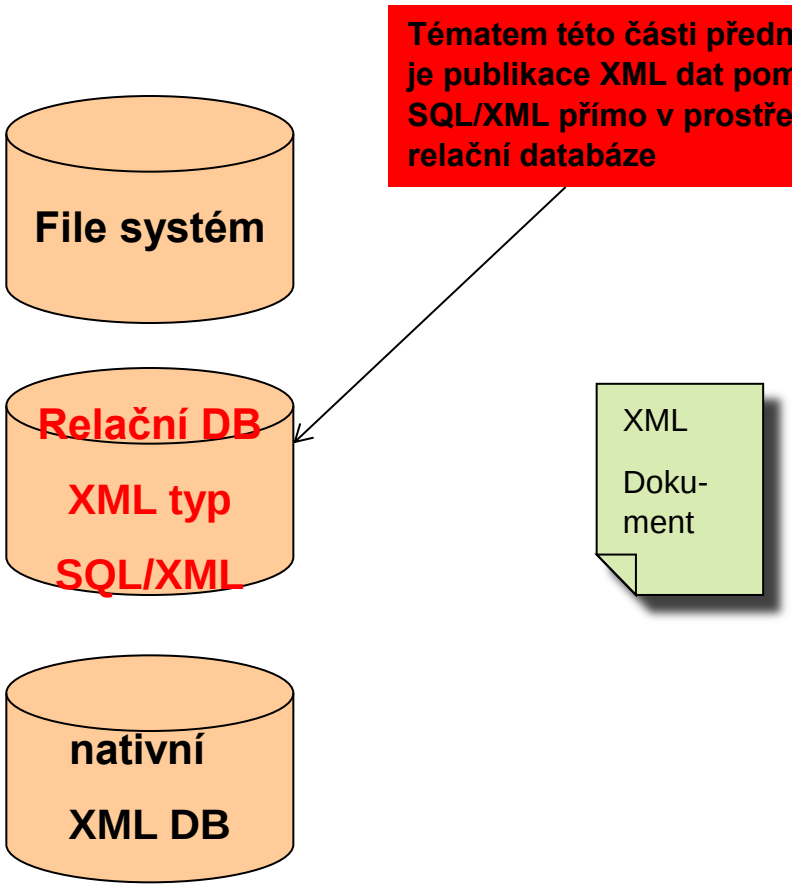
Technologie SQL/XML

SQL/XML - publikace XML dat

Jiří Měska (jiri.meska@gmail.com)

MIB008, Datové a procesní modely, MFF UK Praha, 2020

Perzistence dat



Technologie zpracování

Adresace	XPath
Dotazování	XQuery XQuery Update
Transformace	XSLT
Validace	DTD XML Schema Relax NG Schematron
Parsing	SAX, DOM, StAX, LINQ

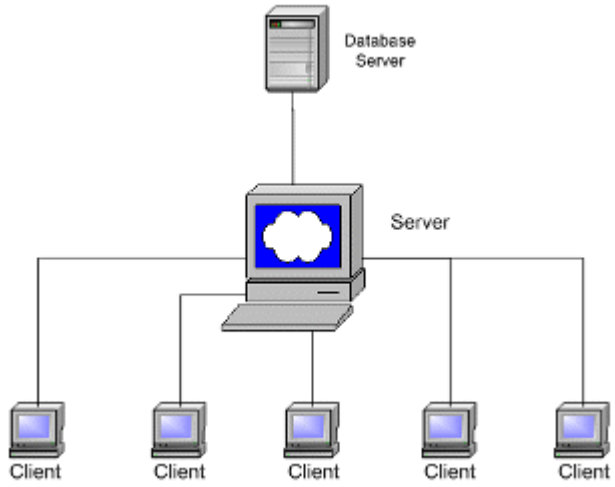
Uživatelské případy zpracování databázových dat:

Aplikační server provádí zpracování dat (výpočty, vyhodnocování aj.) v tomto případě se obvykle mapují persistentní data **do objektového modelu** (např. Java, C++, Hibernate, JPA). Téma přednášky aplikačního programování.

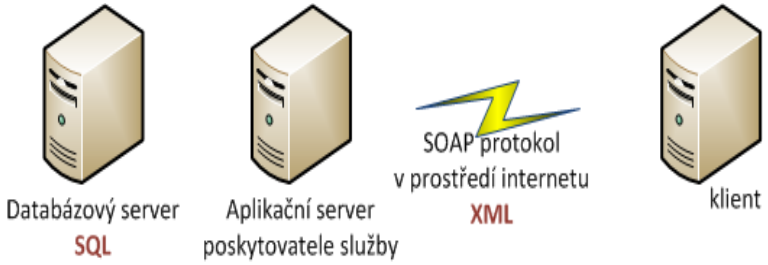
Aplikační server distribuuje data prostřednictvím webových služeb v tomto případě je lepší mapování **do XML**, které je nativním formátem WS. Zde je výhodné použití **SQL/XML**

Aplikační server distribuuje data pro JavaScriptovou aplikace tomto případě je lepší mapování **do JSON**. Zde je výhodné použití **SQL/JSON**

C2B Architektura:



B2B Architektura:



Stromový datový model versus relační datový model.

Relační (SQL) model je postaven na dvojdimensionálních tabulkách, u kterých není definována žádná hierarchie ani pořadí. Tyto tabulky jsou provázány vzájemnými odkazy zajištěnými referenční integritou.

Stromový (XML) model je postaven na hierarchické stromové struktuře uspořádání dat,

- kde vertikální zařazení je podstatné,
- kde horizontální pořadí je podstatné.

Realita:

- Většina informací je dodnes persistentně uložena v relačních databázích. Často i nerelační databáze jsou vnitřně postavené na relačních databázích.
- Informace prezentovaná uživateli (např. HTML) je obvykle hierarchicky strukturována. V hlavě si neukládáme informaci v relační struktuře.
- Informace předávaná webovými službami/mezi programy je obvykle definována nějakým standardem nebo proprietárním formátem obvykle opět v hierarchické struktuře.
- Nakonec i struktura informace na této stránce má stromovou povahu ☺

Rozdíl v modelech dat mezi relační databází a XML si ukážeme na příkladu prezentace obchodníků a jejich objednávek, které si zjednodušíme pro účely této prézentace pomocí dvou pohledů:

```
create view Obchodnik_1 as
select
        idObchodnik,
        jmenoObchodnika,
        kraj
from
        Obchodnik join Region Using(idRegion)
```

```
1002;"Novák";"Statutární město Praha"
1001;"Charvát";"Statutární město Praha"
1003;"Ryšavý";"Statutární město Brno"
1004;"Pilný";"Západní čechy - plzeňsko"
1005;"Horvát";"Bratislava"
1006;"Ujen";"Západní čechy - karlovarsko„
…
```

```
create view Objednavka_1 as
select
        idObjednavka,
        idObchodnik,
        datum,
        popis
from
        Objednavka join Zbozi Using(idZbozi)
```

```
3001;1007;"2006-10-03";"zimní rukavice"
3002;1002;"2006-10-03";"závodní kolo"
3025;1006;"2008-10-06";"kolečkové brusle"
3024;1006;"2008-10-06";"kolečkové brusle"
3023;1006;"2008-10-06";"kolečkové brusle"
3022;1006;"2008-10-04";"kolečkové brusle"
3021;1006;"2008-10-04";"kolečkové brusle"
3020;1006;"2008-10-04";"kolečkové brusle„
…
```

Relační reprezentace dat

Chceme z databáze vybrat všechny objednávky obchodníků:

select * from Obchodnik_1 join Objednavka_1 using(idObchodnik)

```
1002;"Novák";"Statutární město Praha";3008;"2006-10-05";"stan"
1002;"Novák";"Statutární město Praha";3007;"2006-10-04";"diabolky"
1002;"Novák";"Statutární město Praha";3005;"2006-10-03";"fotbalový míč"
1002;"Novák";"Statutární město Praha";3011;"2006-10-06";"vybaveni tělocvičny"
1002;"Novák";"Statutární město Praha";3002;"2006-10-03";"závodní kolo"
1001;"Charvát";"Statutární město Praha";3010;"2006-10-06";"běžky Fischer"
1001;"Charvát";"Statutární město Praha";3003;"2006-10-03";"kolečkové brusle"
1006;"Ujen";"Západní čechy - karlovarsko";3009;"2006-10-04";"pinpongový stul"
1006;"Ujen";"Západní čechy - karlovarsko";3004;"2008-10-03";"kolečkové brusle"
1006;"Ujen";"Západní čechy - karlovarsko";3020;"2008-10-04";"kolečkové brusle"
1006;"Ujen";"Západní čechy - karlovarsko";3021;"2008-10-04";"kolečkové brusle„
…
```

```
<objednavka id="3004">
        <datum>2008-10-03</datum><popis>kolečkové brusle</popis>
        <obchodnik id="1006"/><jmeno>Ujen</jmeno><region>Západní čechy - karlovarsko</region>
</objednavka>
```

Jak to chce vidět uživatel?

Obrazovka Obchodníků:

Obchodník	Region	Počet objednávek
Novák	Praha	5
Charvát	Praha	2
Ujen	Západní Čechy	8

A pro obchodníka chce vidět jeho objednávky, např. „rozklikne“ Chorvát a očekává:

Zboží	Datum
běžky Fischer	6.10.2006
kolečkové brusle	3.10.2006

Které údaje ve výstupní sestavě na předchozí stránce jsou pro tento výstup redundantní?

Je to vhodná forma pro prezentaci uživateli?

XML reprezentace stejných dat:

```
...
<obchodnik id="1003"><jmeno>Ryšavý</jmeno><region>Statutární město Brno</region><objednavky/></obchodnik>
<obchodnik id="1004"><jmeno>Pilný</jmeno><region>Západní čechy - plzeňsko</region><objednavky/></obchodnik>
<obchodnik id="1005"><jmeno>Horvát</jmeno><region>Bratislava</region><objednavky/></obchodnik>
<obchodnik id="1006"><jmeno>Ujen</jmeno><region>Západní čechy - karlovarsko</region>
        <objednavky>
                <objednavka id="3004"><datum>2008-10-03</datum><popis>kolečkové brusle</popis></objednavka>
                <objednavka id="3009"><datum>2006-10-04</datum><popis>pinpongový stul</popis></objednavka>
                <objednavka id="3020"><datum>2008-10-04</datum><popis>kolečkové brusle</popis></objednavka>
                <objednavka id="3021"><datum>2008-10-04</datum><popis>kolečkové brusle</popis></objednavka>
                <objednavka id="3022"><datum>2008-10-04</datum><popis>kolečkové brusle</popis></objednavka>
                <objednavka id="3023"><datum>2008-10-06</datum><popis>kolečkové brusle</popis></objednavka>
                <objednavka id="3024"><datum>2008-10-06</datum><popis>kolečkové brusle</popis></objednavka>
                <objednavka id="3025"><datum>2008-10-06</datum><popis>kolečkové brusle</popis></objednavka>
        </objednavky>
</obchodnik>
...
```

V čem je úloha aplikačního serveru, jestliže má poskytnout klientskému programu tuto reprezentaci dat na základě předchozího SQL dotazu?

Jak je to možné udělat?

Dnes si ukážeme jak pomocí SQL/XML získáme požadovaná data elegantnějším způsobem, a to jediným příkazem.

Na závěr si ukážeme dotaz, kterým získáme požadované XML

SQL/XML je XML rozšíření SQL ve standardu SQL 2003.

V rámci standardu **SQL 2003** je definována následující funkcionalita pro podporu práce s XML:

- XML Publikační funkce – vytváření XML výstupu jako výsledek dotazu do relační databáze.

  Vhodné pro vytváření XML dat z hodnot uložených v relační databázi.

- XML Typ dat – XML ukládáme jako XML datový typ, databáze s těmito daty potom umí pracovat jako s XML, například pomocí XPath, jak uvídíme příště.

  Použitelné pro ukládání XML dat do relační databáze, aniž bychom museli vytvářet relační model nad těmito daty. Vhodné pro data u nichž neznáme v době vytváření databáze úplnou strukturu.

- Mapovací pravidla
  - Mapování tabulek do XML
  - Mapování struktury tabulek (typ dat) do XML Schematu

  Vhodné pro jednoduchý export databáze v XML formátu.

Ilustrační příklad XML publikačních funkcí

Tabulka ZBOZI

"idzbozi";"popis";"cenakus"

5001;"zimní rukavice";490

5002;"závodní kolo";95000

5003;"kolečkové brusle";1000

5004;"vybaveni tělocvičny";110000

....

```
select idzbozi,
xmlelement(name "Zbozi",
      xmlattributes( idzbozi AS "id" ),
      xmlelement(name "popis", popis ),
      xmlelement(name "cenazakus", cenakus )
      ) as xmlzbozi
from zbozi
```

Publikační funkce:
xmlelement(…)
xmlattributes(…)

Typ dat: XML

Výstupní tabulka o dvou sloupcích

"idzbozi";"xmlzbozi"

5001;"<Zbozi id=""5001""><popis>zimní rukavice</popis><cenazakus>490</cenazakus></Zbozi>"

5002;"<Zbozi id=""5002""><popis>závodní kolo</popis><cenazakus>95000</cenazakus></Zbozi>"

5003;"<Zbozi id=""5003""><popis>kolečkové brusle</popis><cenazakus>1000</cenazakus></Zbozi>"

5004;"<Zbozi id=""5004""><popis>vybaveni tělocvičky</popis><cenazakus>110000</cenazakus></Zbozi>"

...

XML Publikační funkce přehled:

- **xmlelement()** ... Pomocí této funkce vytváříme XML element
- **xmlattributes()** ... Pomocí této funkce vytváříme XML atributy elementu
- **xmlforest()** ... Pomocí této funkce lze vytvořit xml elementy ze sloupců tabulky
- **xmlconcat()** ... Pomocí této funkce lze spojit několik XML fragmentů do jednoho
- **xmlagg()** ... Pomocí této funkce lze agregovat více řádků jednoho selectu do jednoho elementu nebo do jedné hodnoty
- **xmlparse()** ... Pomocí této funkce lze načíst znakový řetězec a vytvořit z něj XML datový typ (rozlišit XML uložený jako řetězec nebo XML datový typ)
- **xmlroot()** ... Pomocí této funkce vytvoříme root element XML dokumentu
- **xmlcomment()** ... Pomocí této funkce lze vytvořit poznámku v XML
- **xmlpi()** ... Pomocí této funkce lze vytvořit procesní instrukci v XML

Funkce: XMLElement

Syntaxe: xmlelement(name e_name [, xmlattributes(value_ expression [AS attr_name] [, ...])] [, value_expression, ...])

Cíl: Pro každou řádku tabulky zboží vytvoření jednoduchého xml fragmentu tvořeného elementem „nabídka zboží“ obsahujícího hodnotu ze sloupce popis zboží

Příklad: xmlelement1.sql

```
select
    idzbozi,
    xmlelement( name nabidka_zbozi, popis) as xmlzbozi
from zbozi
```

XML content

```
"idzbozi";"xmlzbozi"
5001;"<nabidka_zbozi>zimní rukavice</nabidka_zbozi>"
5002;"<nabidka_zbozi>závodní kolo</nabidka_zbozi>"
5003;"<nabidka_zbozi>kolečkové brusle</nabidka_zbozi>"
5004;"<nabidka_zbozi>vybaveni tělocvičky</nabidka_zbozi>"
5005;"<nabidka_zbozi>fotbalový míč</nabidka_zbozi>"
5006;"<nabidka_zbozi>volejbalový míč</nabidka_zbozi>"
5007;"<nabidka_zbozi>diabolka</nabidka_zbozi>„ ...
```

Funkce: XMLElement + atribut

Cíl: Předchozí příklad rozšíříme o vytvoření atributu id a hodnotou vytvářeného elementu „cenik“ bude nějaký text, popis zboží a cena zboží.

Příklad: xmlelement6.sql

```
select idzbozi,
xmlelement(name "cenik",
    xmlattributes( idzbozi AS id ),
    'zde později něco doplníme',
    xmlelement( name popis, popis),
    cenakus
    ) as xzbozi
from zbozi
```

```
5001;"<cenik id="5001">zde později něco doplnime<popis>zimní rukavice</popis>490</cenik>"
5002;"<cenik id="5002">zde později něco doplnime<popis>závodní kolo</popis>14867</cenik>"
...
```

Funkce: XMLElement + vnořený select + atribut

Cíl: V následujícím příkladu ukážeme, jak lze použít poddotaz při tvorbě hodnoty XML elementu. Chceme pro každého zákazníka vytvořit fragment „zakaznik“ obsahující nejen údaje o zákazníkovi z tabulky zákazník, ale i celkový počet jeho objednávek.

Soubor: xmlelement3.sql

```
select idzakaznik,
        xmlelement(name zakaznik,
                xmlattributes( idzakaznik AS id ),
                xmlelement(name popis_zakaznika, jmenozakaznika, ' z ',
                        (select mesto from adresa a where z.idadresa=a.idadresa) ),
                xmlelement(name pocetobjednavek,
                        (select count(*) from objednavka o where o.idzakaznik=z.idzakaznik ) )
) as xzakaznik from zakaznik z
```

```
<zakaznik id="2001">
        <popis_zakaznika>Matušičová z Plzeň </popis_zakaznika>
        <pocetobjednavek>2</pocetobjednavek>
</zakaznik>
```

Funkce: XMLElement + namespaces

Cíl: vytvoření elementů pro cenovou nabídku zboží v konkrétním jmenném prostoru.

Pomocí attributu xmlns:si jsme vytvořili alias „si“ pro namespase 'http://www.silicium.cz‘.

Soubor: xmlelement5.sql

```
select idzbozi,
xmlelement(name "si:cenik",
    xmlattributes( 'http://www.silicium.cz' AS "xmlns:si" ),
    xmlelement(name popis, popis ),
    xmlelement(name cenazakus, cenakus )
    ) as xzbozi
from zbozi
```

```
"idzbozi";"xzbozi"
5001;"<si:cenik xmlns:si="http://www.silicium.cz" id="5001"><popis>zimní rukavice</popis><cenazakus>490</cenazakus></si:cenik>"
5002;"<si:cenik xmlns:si="http://www.silicium.cz" id="5002"><popis>závodní kolo</popis><cenazakus>14867</cenazakus></si:cenik>"
```

Funkce: XMLForest (vytvoří seznam elementů)

Syntaxe PostgreSQL: **XMLForest(content [AS name] [, ...])**

Cíl: Vytvoření ceníku z tabulky zbozi. Funkce XMLForest vytvoří z vybraných položek/sloupců elementy.

Soubor: xmlforest1.sql

```
select      idzbozi,
        xmlelement(name zbozi,
                xmlforest( popis,
                           cenakus)
        ) as xzbozi
from zbozi
```

Alternatiivní zápis bez pomocí xmlforest:

```
select      idzbozi,
        xmlelement(name zbozi,
            xmlelement(name popis, popis),
            xmlelement(name cenakus, cenakus)
         ) as xzbozi
from zbozi
```

```
"idzbozi";"xzbozi"
5001;"<zbozi><popis>zimní rukavice</popis><cenakus>490</cenakus></zbozi>"
5002;"<zbozi><popis>závodní kolo</popis><cenakus>95000</cenakus></zbozi>"
...
```

Funkce: XMLForests

Cíl: Vytvoření ceníku z tabulky zbozi se zadanými elementy popiszbozi, cenazbozi

Soubor: xmlforest2.sql

```
select      idzbozi,
        xmlforest(      popis AS popiszbozi,
                        cenakus AS cena,
                        'a' AS a,
                        0.9*cenakus AS dicount )
      as xzbozi
from zbozi
```

```
<popiszbozi>zimní rukavice</popiszbozi><cena>490</cena><a>a</a><dicount>441</dicount>
<popiszbozi>závodní kolo</popiszbozi><cena>14867</cena><a>a</a><dicount>13380.3</dicount>"
…
```

V tomto případě jsme xml hodnotou v řádce není správně formátovaný xml dokument, který musí mít jediný root, ale fragment XML, který může být zřetězením elementů a textových polí. Obsahem xml elementu v databázi může být z praktických důvodů xml dokument nebo fragment xml. Více příště.

Funkce: XMLConcat

Syntaxe PostgreSQL: XMLConcat(xml fragment[, ...])

Funkce XmlConcat() spojuje několik XML fragmentů do jednoho XML fragmentu.

Cíl: Vytvořit informaci o obchodnících tvořenou dvěma elementy, první element „obchodnik“ bude obsahovat informaci o obchodnikovi, druhý element „adresa“ bude obsahovat informaci o adrese obchodníka.

Soubor: xmlconcat1.sql

```
select
xmlconcat(
    xmlelement( name obchodnik, 'Jméno: ' || jmenoobchodnika || ', Provize: ' ||
                                                provize),
    xmlelement( name adresa,  mesto || ', ' || ulice )
    ) as xobchodnik
from obchodnik join adresa using(idAdresa)
```

```
"xobchodnik"
"<obchodnik>Jméno: Novák, Provize: 0.2</obchodnik><adresa>Praha, Kavkova
324</adresa>"
...
```

Otázka: Je XML hodnota řádku XML dokumentem?

Funkce: XMLConcat

Cíl: Variace na předchozí téma s použitím xmlforest.

Soubor: xmlconcat2.sql

```
select
xmlconcat(
      xmlelement( name obchodnik, 'Jméno: ' || jmenoobchodnika || ', Provize: ' ||
                                          provize),
      xmlforest(  mesto, ulice )
      ) as xobchodnik
from obchodnik join adresa using(idAdresa)
```

```
"<obchodnik>Jméno: Novák, Provize: 0.2</obchodnik>
<mesto>Praha</mesto>
<ulice>Kavkova 324</ulice>"
```

Funkce: XMLAgg (agregace celé tabulky)

Syntaxe PostgreSQL: XMLAgg(xml)

XMLAgg je agregační funkce. Na rozdíl od předchozího XMLAgg nepracuje s jednou řádkou, ale s celou tabulkou nebo se skupinou řádek seskupených podle group by

Cíl: Vytvořit jeden fragment xml obsahující kompletní informaci o tabulce zboží.

Soubory: xmlagg0a.sql, xmlagg0b.sql

Výstupem je tabulka, kde každá řádka reprezentuje jednu položku zboží

```
select
    xmlelement( name zbozi,
    xmlforest( idzbozi, popis, cenakus)
    ) as xzbozi
from zbozi
```

```
"<zbozi><idzbozi>5001</idzbozi><popis>zimní rukavice</popis><cenakus>490</cenakus></zbozi>"
"<zbozi><idzbozi>5002</idzbozi><popis>závodní kolo</popis><cenakus>14867</cenakus></zbozi>„
...
```

Výstupem je jedna řádka obsahující všechny položky agregované do jednoho XML

```
select
    xmlelement( name zbozi,
    xmlagg(xmlforest( idzbozi, popis, cenakus)
    )) as xzbozi
from zbozi
```

```
"<zbozi><idzbozi>5001</idzbozi><popis>zimní rukavice</popis><cenakus>490</cenakus>
<idzbozi>5002</idzbozi><popis>závodní kolo</popis><cenakus>14867</cenakus>
<idzbozi>5003</idzbozi><popis>kolečkové
..."</zbozi>"
```

Funkce: XMLAgg (agregace pomocí group by)

Cíl: Přehled všech obchodníků - managerů s informací o všech jejich podřízených

Soubor: xmlagg1.sql

```
select idManager,
    xmlelement( name manager,
    xmlattributes( jmenomanagera as jmenomanagera),
    xmlagg( xmlelement( name obchodnik, jmenoobchodnika))
    ) as xobchodnik
from ( select
    nad.idObchodnik as idManager,
    nad.jmenoobchodnika as jmenomanagera,
    pod.jmenoobchodnika as jmenoobchodnika
    from obchodnik nad, obchodnik pod
    where pod.idmanager=nad.idobchodnik ) AS foo
group by jmenomanagera, idManager
```

SQL/XML - Publikační funkce

"idManager";"xobchodnik"

1005; <manager jmenomanagera="Horvát">
<obchodnik>Ferencz</obchodnik>
</manager>

1002; <manager jmenomanagera="Novák">
<obchodnik>Charvát</obchodnik>
<obchodnik>Ryšavý</obchodnik>
<obchodnik>Pilný</obchodnik>
<obchodnik>Ujen</obchodnik>
<obchodnik>Brutus</obchodnik>
</manager>

1006; <manager jmenomanagera="Ujen">
<obchodnik>Nhac</obchodnik>
</manager>

"<manager jmenomanagera="Horvát"><obchodnik>Ferencz</obchodnik></manager>"

Poddotaz před group by

1002;"Novák";"Charvát"

1002;"Novák";"Ryšavý"

1002;"Novák";"Pilný"

1002;"Novák";"Ujen"

1006;"Ujen";"Nhac"

1005;"Horvát";"Ferencz"

1002;"Novák";"Brutus"

Funkce: XMLAgg (Seskupení objednávek podle obchodníků)

Cíl: Přehled objednávek podle obchodníků

Soubor: xmlagg2.sql

```
select idobchodnik,
       xmlelement( name obchodnik,
        xmlattributes( idobchodnik as idObchodnika),
        xmlagg( xmlelement( name objednavka,
                    xmlattributes( idobjednavka as idobjednavka),
                    cenakus*mnozstvikus ))
      ) as xobchodnik
from objednavka
group by idobchodnik
```

SQL/XML - Publikační funkce

```
1002;<obchodnik idObchodnika="1002">
        <objednavka idobjednavka=""3002"">190000</objednavka>
        <objednavka idobjednavka=""3005"">504000</objednavka>
        <objednavka idobjednavka=""3007"">7000</objednavka>
        <objednavka idobjednavka=""3008"">67000</objednavka>
        <objednavka idobjednavka=""3011"">100000</objednavka>
    </obchodnik>
1007;<obchodnik idObchodnika="1007">
        <objednavka idobjednavka=""3001"">490</objednavka>
        <objednavka idobjednavka=""3006"">135300</objednavka>
        <objednavka idobjednavka=""3026"">7600</objednavka>
        <objednavka idobjednavka=""3027"">7800</objednavka>
        <objednavka idobjednavka=""3028"">23400</objednavka>
        <objednavka idobjednavka=""3029"">31200</objednavka>
    </obchodnik>
....
```

Funkce: XMLAgg (Seskupení objednávek podle obchodníků)

Cíl: Modifikace předchozího příkladu**.**

- Setřídit objednávky podle celkové ceny objednávky.
- U elementu obchodník zobrazte součet cen všech objednávek obchodníka

Soubor: xmlagg3.sql

```
select idobchodnik,
      xmlelement( name obchodnik,
        xmlattributes( idobchodnik as idObchodnika, sum(cenakus*mnozstvikus) as
      celkovasumma),
        xmlagg( xmlelement( name objednavka,
                    xmlattributes( idobjednavka as idobjednavka),
                    cenakus*mnozstvikus )
                    ORDER BY cenakus*mnozstvikus DESC )
      ) as xobchodnik
from objednavka
group by idobchodnik
```

Otázka: Jaké jsou zde použity agregační funkce?

SQL/XML - Publikační funkce

1002;<obchodnik idobchodnika="1002" celkovasumma="868000">

<objednavka idobjednavka="3005">504000</objednavka>

<objednavka idobjednavka="3002">190000</objednavka>

<objednavka idobjednavka="3011">100000</objednavka>

<objednavka idobjednavka="3008">67000</objednavka>

…

1007; <obchodnik idobchodnika="1007" celkovasumma="205790">

<objednavka idobjednavka="3006">135300</objednavka>

<objednavka idobjednavka="3029">31200</objednavka>

<objednavka idobjednavka="3028">23400</objednavka>

<objednavka idobjednavka="3027">7800</objednavka>

…

....

Funkce: XMLRoot (rootový element XML dokumentu)

Syntaxe PostgreSQL: xmlroot(xml dokument, version text | no value [, standalone yes|no|no value])

Cíl: Vytvoření XML dokument/ obsahujícího XML deklaraci (procesní instrukce) a kořenový dokument.

Soubor: xmlroot1.sql

Poznámka: pseudoatribut standalone se týká DTD, pokud se nepoužívá je nezajímavý.

XML dokument v příkladu je vytvořen parsováním textového řetězce funkcí **xmlparse**.

```
SELECT XMLRoot( xmlparse( document '<my>my</my>'),
       VERSION '1.0', STANDALONE YES)
  AS xmlroot
```

```
"<?xml version="1.0" standalone="yes"?><my>my</my>"
```

Funkce: XMLRoot

Cíl: Vytvoření dokumentu agregujícího seznam zboží s kořenovým elementem nabídka

Soubor: xmlroot2.sql

```
SELECT
  XMLRoot(
        XMLElement( name nabidka,
             XMLAgg( XMLElement(name nabidka_zbozi, popis))),
        VERSION '1.0', STANDALONE YES)
  AS xmlroot
from zbozi
```

```
"<?xml version="1.0" standalone="yes"?>
<nabidka>
<nabidka_zbozi>zimní rukavice</nabidka_zbozi>
<nabidka_zbozi>závodní kolo</nabidka_zbozi>
...
</nabidka>"
```

Motivační příklad z úvodu – bez kořenového elementu

Cíl: Vytvoření seznamu obchodníků a jejich objednávek (řešeno poddotazem)

Soubor: ObjednavkyObchodniku.sql

```
Select
    xmlagg(
        xmlelement( name obchodnik,
            xmlattributes( ob.idObchodnik as id),
            xmlforest( ob.jmenoObchodnika as jmeno, ob.kraj as region ),
            xmlelement( name objednavky,
            ( select
                xmlagg( xmlelement( name objednavka,
                        xmlattributes( o.idObjednavka as id),
                        xmlforest( o.datum as datum, o.popis as popis)))
              from Objednavka_1 as o
              where ob.idObchodnik = o.idObchodnik ) ) ) )   as objednavky_obchodnika
from Obchodnik_1 as ob
```

Vyzkoušejte si na cvičení.

Motivační příklad z úvodu – s kořenovým elementem

Cíl: Vytvoření seznamu obchodníků a jejich objednávek (řešeno poddotazem)

Soubor: ObjednavkyObchodniku1.sql

```
Select
      xmlroot( xmlelement( name obchodnici,
      xmlagg( xmlelement( name obchodnik,
                xmlattributes( ob.idObchodnik as id),
                xmlforest( ob.jmenoObchodnika as jmeno, ob.kraj as region ),
                xmlelement( name objednavky,
                   ( select
                        xmlagg( xmlelement( name objednavka,
                        xmlattributes( o.idObjednavka as id),
                        xmlforest( o.datum as datum, o.popis as popis)))
                        from Objednavka_1 as o
                     where ob.idObchodnik = o.idObchodnik ) ) ) ) ),
               VERSION '1.0', STANDALONE YES)  as objednavky_obchodnika
from Obchodnik_1 as ob
```


Thanks