Przegląd języków zapytań

Eklektyczny i stronniczy przegląd języków zapytań do baz danych: obiektowych, semistrukturalnych i XML.

Podziękowania

- Dla Grzegorza Enzo Dołęgowskiego za wpisanie moich notatek do komputera.

Przykładowy schemat obiektowej bazy danych

K = {Osoba, Małżeństwo, Zameldowanie}

```
typ(Osoba) =            [
                            PESEL : String;
                            Imię : String;
                            Nazwisko : String;
                            Ojciec : Osoba;
                            Matka : Osoba;
                            Dzieci : {Osoba}
                        ]

typ(Małżeństwo) =       [
                            Mąż : Osoba;
                            Żona : Osoba;
                            Data_zawarcia : Date;
                            Data_ustania : Date;
                            Dzieci : {Osoba}
                        ]

typ(Zameldowanie) =     [
                            Obywatel : Osoba;
                            Adres :    [
                                           Ulica : String;
                                           Nr_domu : String;
                                           Nr_miesz : String;
                                           Miejscowość : String
                                       ]
                            Od_kiedy : Date;
                            Do_kiedy : Date;
                            Rodzaj : String
                        ]
```

Język zapytań Nasz

select *wyrażenie*
  from *ciąg deklaracji zmiennych zakresowych*
  where *warunek logiczny*

wyrażenie:

1. element **W***,
2. agregacje na zbiorach (*count*, *sum*, *avg*, *flatten*, *element* — koercja kolekcji na jej jedyny element),
3. zmienne zakresowe.

Pierwszy przykład

Nasz:

```
select [ Nazwisko : x.Nazwisko, Imię : x.Imię ]
  from x ∈ Osoba
  where x.Imię = 'Adolf';
```

O_2 (ODMG OQL):

```
select struct(Nazwisko : x.Nazwisko, Imię : x.Imię)
  from x in Osoba
  where x.Imię = 'Adolf';
```

SQL, *ad hoc*, „brudny”:

```
select Nazwisko, Imię
  from Osoba
  where Imię = 'Adolf';
```

SQL, aplikacyjny, „czysty”:

```
select x.Nazwisko, x.Imię
  from Osoba x
  where x.Imię = 'Adolf';
```

W języku 3GL:

```
from   →   for x ∈ Osoba do
where  →     if x.Imię = 'Adolf' then
select →       output [ Nazwisko : x.Nazwisko, Imię : x.Imię ]
```

Złączenie zależne (ewaluacje od lewej do prawej)

Nasz:

```
select [ Nazwisko : x.Nazwisko, Rodzic : x.Imię, Dziecko : y.Imię ]
  from x ∈ Osoba, y ∈ x.Dzieci
  where x.Nazwisko = y.Nazwisko;
```

W języku 3GL:

```
for x ∈ Osoba do
  for y ∈ x.Dzieci do
    if x.Nazwisko = y.Nazwisko then
      output [ Nazwisko : x.Nazwisko, Rodzic : x.Imię, Dziecko : y.Imię ]
```

W stylu SQL, złączenie niezależne:

```
select x.Nazwisko, x.Imię as Rodzic, y.Imię as Dziecko
  from Osoba as x, Osoba as y
  where x.Nazwisko = y.Nazwisko
    and y in x.Dzieci;
```

Brak potrzeby GROUP BY

Nasz:

```
select [ Imię : x.Imię, Nazwisko : x.Nazwisko, Potomstwo : count(x.Dzieci) ]
  from x ∈ Osoba
  where count(x.Dzieci) > 5;
```

SQL87:

```
select x.Imię, x.Nazwisko, count(*)
  from Osoba x, Osoba y
  where x.PESEL = y.PESEL_matki or x.PESEL = y.PESEL_ojca
  group by x.PESEL, x.Imię, x.Nazwisko
  having count(*) > 5;
```

SQL99 (bez GROUP BY):

```
select x.Imię, x.Nazwisko,
       (select count(*)
          from Osoba y
          where x.PESEL=y.PESEL_ojca
             or x.PESEL=y.PESEL_matki) as Potomstwo
  from Osoba x
  where Potomstwo > 5;
```

Przetwarzanie kolekcji

Wnuki

```
select [ Imię : x.Imię, Nazwisko : x.Nazwisko,
         Wnuki : flatten(select y.Dzieci from y ∈ x.Dzieci ) ]
  from x ∈ Osoba;
```

Jedynaki i ich rodzice

```
select [ Nazwisko : y.Nazwisko, Rodzic : x.Imię, Jedynak : y.Imię ]
  from x ∈ Osoba, y ∈ x.Dzieci
  where count(x.Dzieci) = 1;
```

Wyrażenia ścieżkowe

```
select [ Nazwisko : m.Mąż.Nazwisko, ImięŻony : m.Żona.Imię, ImięMęża : m.Mąż.Imię ]
  from m ∈ Małżeństwo
  where Data_zawarcia > #16.03.2006#;
```

Wyrażenia są:

1. *skalarne*, gdy nie ma atrybutów zbiorowych na ścieżce,
2. *zbiorowe*, gdy jest choć jeden atrybut zbiorowy na ścieżce.

Wyrażenia ścieżkowe z wyrażeniami regularnymi zawsze są zbiorowe:

_	Dowolna krawędź grafu
(*x*)?	0 albo 1 wystąpienie krawędzi *x*
(*x*)*	Dowolna liczba (również 0) wystąpień krawędzi *x*
(*x*)+	Dowolna dodatnia liczba wystąpień krawędzi *x*
(*x* | *y*)	Jedno wystąpienie krawędzi *x* albo jedno wystąpienie krawędzi *y*

```
select [ Imię : z.Imię, Nazwisko : z.Nazwisko ]
  from  x ∈ (select y
             from y ∈ Osoba
             where y.Imię = 'Krzysztof and y.Nazwisko = 'Stencel'),
        z ∈ x._*.(Ojciec|Matka)
```

albo (z dokładną specyfikacją ścieżki):

```
select [ Imię : z.Imię, Nazwisko : z.Nazwisko ]
  from  x ∈ (select y
             from y ∈ Osoba
             where y.Imię = 'Krzysztof and y.Nazwisko = 'Stencel'),
        z ∈ x.(Ojciec|Matka)*.(Ojciec|Matka)
```

Zmienne ścieżkowe

Na zmiennej ścieżkowej możemy zapamiętać ścieżkę, która doprowadziła do przetwarzanego obiektu:

(ścieżka)@Z

Do zmiennej odwołujemy się poprzez @Z. Ścieżki i nazwy pól są obywatelami I kategorii.

Pokrewieństwo:

```
select [ Imię : y.Imię, Nazwisko : y.Nazwisko, Pokrewieństwo : @P ]
  from x ∈ Osoba, y ∈ x.((Ojciec|Matka)+)@P
  where x.Imię = 'Krzysztof' and x.Nazwisko = 'Stencel';
```

Jako etykieta pola struktury:

```
select [ Imię : z.Imię, Nazwisko : z.Nazwisko, @P: u.Imię ]
  from z ∈ Osoba, u ∈ z.(Ojciec|Matka)@P;
```

Aplikacja ścieżki do innego obiektu:

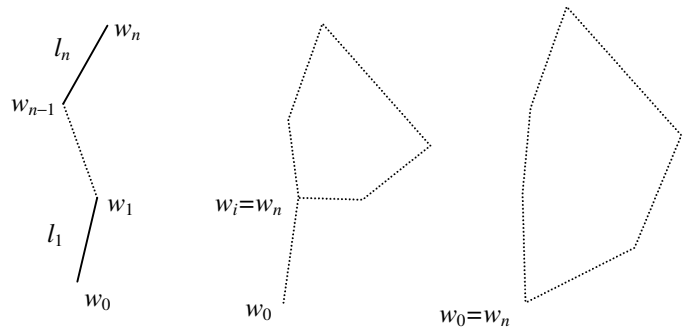
```
select [  Imię : z.Imię, Nazwisko : z.Nazwisko, pokrewieństwo : @P,
          TenSamPoziomPokrewieństwaUKorbsdaja : y.@P.Imię ]
  from x ∈ Osoba, z ∈ x.((Ojciec|Matka)*)@P, y ∈ Osoba
  where x.Imię = 'Krzysztof' and x.Nazwisko = 'Stencel'
    and y.Imię = 'Marcin' and y.Nazwisko = 'Korbsday';
```

Ewaluacja ścieżek

Rozpatrujemy tylko ścieżki bez cyklu, a dokładniej bez powtórzeń węzłów. Co najwyżej jeden węzeł może się wystąpić dwa razy i wtedy jest on także węzłem ostatnim. Wyrażenia ścieżkowe ewaluuje się do zbioru ścieżek.

Ścieżką nazwiemy ciąg $(w_0, l_1, w_1, l_2, w_2, \ldots, w_{n-1}, l_n, w_n)$. Dla takiej ścieżki zmienna ścieżkowa ma wartość:

$$@P = l_1, l_2, \ldots, l_n$$



Zbiorowe i skalarne wyrażenia ścieżkowe

select [Imię : x.Imię, Nazwisko : x.Nazwisko, Wnuki : flatten(**select** y.Dzieci **from** y ∈ x.Dzieci)]
from x ∈ Osoba;

Przyjrzymy się wyrażeniu:

flatten(**select** y.Dzieci **from** y ∈ x.Dzieci)

i typom jego podwyrażeń:

$$typ(\text{y.Dzieci}) = \{\text{Osoba}\}$$

$$typ(\textbf{select}\ \text{y.Dzieci}\ \textbf{from}\ \text{y} \in \text{x.Dzieci}) = \{\{\text{Osoba}\}\}$$

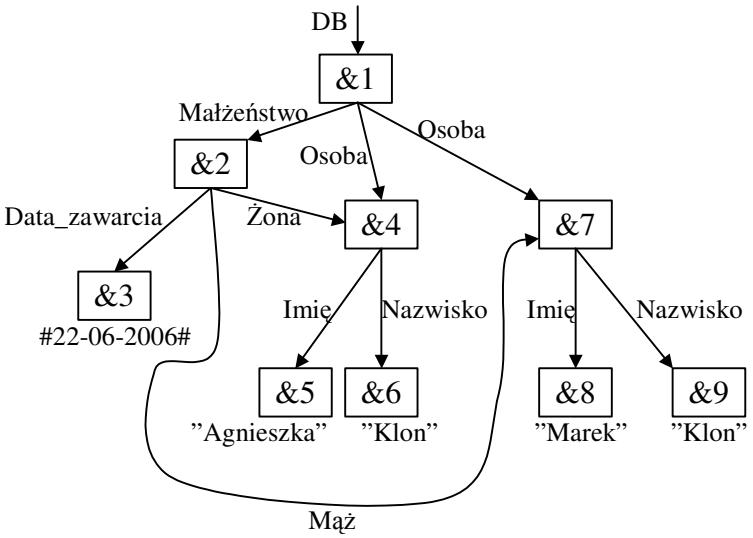
$$typ(\text{flatten}(\textbf{select}\ \text{y.Dzieci}\ \textbf{from}\ \text{y} \in \text{x.Dzieci})) = \{\text{Osoba}\}$$

W językach „strukturalnych”, gdy nie wolno korzystać ze zbiorowych wyrażeń ścieżkowych konieczne jest *flatten* i podzapytanie **select**. W językach „semistrukturalnych” za pomocą zbiorowych wyrażeń ścieżkowych możemy napisać to zapytanie znacznie prościej.

select [Imię : x.Imię, Nazwisko : x.Nazwisko, Wnuki : x.Dzieci.Dzieci]
from x ∈ Osoba;

Dane semistrukturalne

Grafy bez typów, np. OEM (Object Exchange Model) to zbiór krotek (oid, typ, wartość). Jeśli typ węzła jest złożony, to mogą z niego wychodzić krawędzie skierowane do innych węzłów (dowolnych). Tu najlepiej pasują te konstrukcje ścieżkowe, ponieważ chodzimy po grafie.



Wierzchołki w grafie są:

1. atomowe (mają wartość), tu: &3, &5, &6, &8, &9.
2. złożone (mają wychodzące krawędzie), tu: &1, &2, &4, &7.

Lorel i UnQL

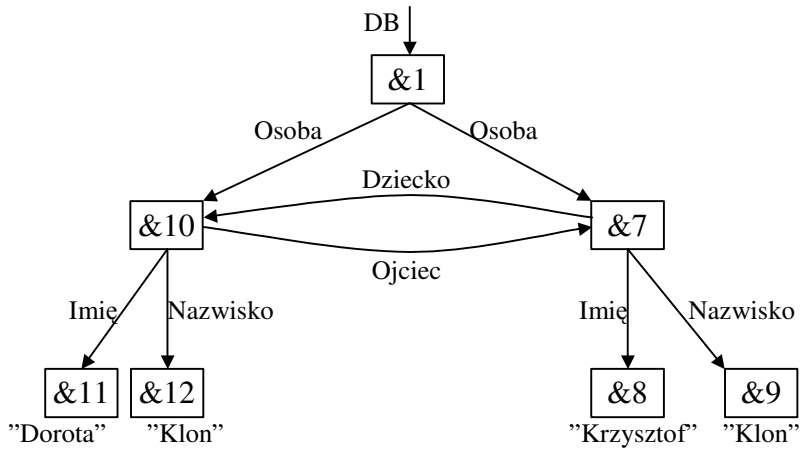
```
select x
    from DB.Osoba x
    where exists y in x.Dziecko : y.Imię = 'Józef';

select wnuk : x
    from DB.Osoba.Dziecko.Dziecko x;

select wnuki : (select y from x.Dziecko y)
    from DB.Osoba.Dziecko x;

select potomki : (select y from x(.Dziecko)* y)
    from DB.Osoba x;

select osobaipotomki : [ x, potomki : (select y from x(.Dziecko)* y) ]
    from DB.Osoba x;
```

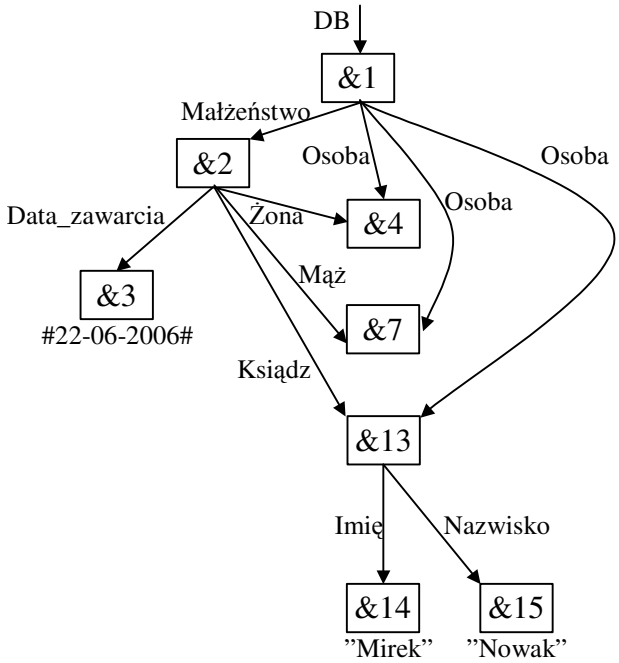


Zamiana etykiet na wartości

```
select małżonek : [ rola : @L, kto : x.Nazwisko ]
    from DB.Małżeństwo._@L x
    where @L = 'Mąż' or @L = 'Żona';
```

Wyłączenie pewnych etykiet (wyłączamy księdza i wszystkie węzły bez nazwiska)

```
select małżonek : [ rola : @L, kto : x.Nazwisko ]
    from DB.Małżeństwo. _@L x
    where @L <> 'Ksiądz'
        and count(x.Nazwisko) > 0;
```



XML

XML należy uważać za sposób zapisu i transmisji danych. XML jest dobrym sposobem zapisu danych seminstrukturalnych.

DTD (Document Type Definition) = Schemat danych

```
<!ELEMENT USC              (Osoba+)>

<!ELEMENT Osoba            (PESEL, Imię, Nazwisko, Zameldowanie*, Dzieci?)>
<!ELEMENT PESEL            (#PCDATA)>
<!ELEMENT Imię             (#PCDATA)>
<!ELEMENT Nazwisko         (#PCDATA)>
<!ELEMENT Dzieci           (Osoba+)>

<!ELEMENT Zameldowanie     (Adres, Data_od, Data_do?)>
<!ELEMENT Data_od          (#PCDATA)>
<!ELEMENT Data_do          (#PCDATA)>

<!ELEMENT Adres            (Miejscowość, Ulica?, Nr_domu, Nr_mieszkania?)>
<!ELEMENT Miejscowość      (#PCDATA)>
<!ELEMENT Ulica            (#PCDATA)>
<!ELEMENT Nr_domu          (#PCDATA)>
<!ELEMENT Nr_mieszkania    (#PCDATA)>
```

Przykładowy dokument XML

```
<?xml version="1.0" encoding="windows-1250" ?>

<USC>
    <Osoba>
        <PESEL>43534534534</PESEL>
        <Imię>Marek</Imię>
        <Nazwisko>Kwiatkowski</Nazwisko>
        <Dzieci>
            <Osoba>
                <PESEL>03243054854</PESEL>
                <Imię>Ludmiła</Imię>
                <Nazwisko>Kwiatkowska</Nazwisko>
            </Osoba>
            <Osoba>
                <PESEL>53453454235</PESEL>
                <Imię>Bogusława</Imię>
                <Nazwisko>Kwiatkowska</Nazwisko>
            </Osoba>
            <Osoba>
                <PESEL>02090475747</PESEL>
                <Imię>Natalia</Imię>
                <Imię>Kornelia</Imię>
                <Nazwisko>Kwiatkowska</Nazwisko>
            </Osoba>
        </Dzieci>
    </Osoba>
</USC>
```

XML-QL

where *oddzielone przecinkami warunki i deklaracje zmiennych zakresowych*
construct *wynik*

PESELE osób mających dzieci (zmienne poprzedzone dolarem)

```
where
    <Osoba>
        <PESEL>$P</PESEL>
```

```
        <Dzieci>$y</Dzieci>
    </Osoba> in 'abc.xml'
construct
    <result>$P</result>
```

Spłaszczenie struktury dokumentu

```
where
    <Osoba></> element_as $O in 'abc.xml'
construct
    $O
```

Osoby z takim samym imieniem i nazwiskiem (</> zamyka ostatni znacznik)

```
where
    <Osoba>
        <PESEL>$P</PESEL>
        <Imię>$I</>
        <Nazwisko>$N</>
    </> in 'abc.xml',
    $I = $N
construct
    <result> $P</>
```

Złączenie zależne

```
where
    <Osoba>
        <Nazwisko>$N</>
        <Dzieci>$D</>
    </> in 'abc.xml',
    <Osoba>
        <Nazwisko>$M</>
    </> in $D,
    $M != $N
construct
    <wynik>
        <Rodzic>$N</>
        <Dziecko>$M</>
    </>
```

Zagnieżdżone zapytanie (**element_as** przypisuje całość znacznika na zmienną, z którą jest związany, tu: wartością zmiennej $O będzie znacznik Osoba)

```
where
    <Nazwisko>$N</> in 'abc.xml'
construct
    <Lista>
        <Nazwisko>$N</>
        where
            <Osoba>
                <PESEL>$P</>
                <Nazwisko>$M</>
            </> element_as $O in 'abc.xml',
            $N = $M
        construct
            $O
    </>
```

Zmienne w znacznikach (wypisanie listy znaczników z dokumentu)

```
where
    <$P></> in 'abc.xml'
construct
    <Znacznik>$P</>
```

Ścieżki w znacznikach — symbole jak w DTD (wszyscy potomkowie osób)

```
where
    <Osoba>
        <(Dzieci.Osoba)+></> element_as $O
    </> in 'abc.xml'
construct
    <Potomek>$O</>
```

Wszystkie miejscowości i ulice

```
where
    <(Dzieci.Osoba)+.(Miejscowość|Ulica)>$A</> in 'abc.xml'
```

```
construct
    <Adres>$A</>
where
    <Osoba>
        <Dzieci>
            <Osoba[$I]>$O</>
        </>
    </>
construct
    <Dziecko>
        <Nr>$I</>
        <Osoba>$O</>
    </>
```

Numeracja znaczników

XSLT

XSLT też jest językiem zapytań, choć trochę nietypowym. XSLT odpowiada gramatyce atrybutywnej za akcjami. Arkusz stylów XSLT steruje zejściami rekurencyjnymi w głąb przetwarzanego dokumentu.

```
<?xml version="1.0" encoding="windows-1250" ?>

<xsl:stylesheet version="1.0"    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                                 xmlns:html="http://www.w3.org/TR/REC-html40">

    <xsl:template match="/">
        <html:html>
            <html:head>
                <html:title>Lista rodów</html:title>
            </html:head>
            <html:body>
                <xsl:apply-templates/>
            </html:body>
        </html:html>
    </xsl:template>

    <xsl:template match="/Osoba">
        <html:h1>Ród: <xsl:apply-templates select="Nazwisko"/></html:h1>
        <html:ul>
            <xsl:call-template name="wydruk-Osoby"/>
        </html:ul>
    </xsl:template>

    <xsl:template match="Osoba" name="wydruk-Osoby">
        <html:li>
            Nazwisko:    <xsl:apply-templates select="Nazwisko"/><html:br/>
            Imię:        <xsl:apply-templates select="Imię"/>
            <xsl:if test="Dzieci">
                <html:ul>
                    <xsl:apply-templates select="Dzieci/Osoba"/>
                </html:ul>
            </xsl:if>
        </html:li>
    </xsl:template>

    <xsl:template match="Nazwisko">
        <html:b> <xsl:value-of select="."/></html:b>
    </xsl:template>

    <xsl:template match="Imię">
        <html:b> <xsl:value-of select="."/></html:b>
    </xsl:template>

</xsl:stylesheet>
```