

XII Seminarium PLOUG
Warszawa
Marzec 2006

XQuery – konkurencja dla SQL?

Tomasz Traczyk

Politechnika Warszawska

Abstrakt. Tutorial prezentuje szybko zyskujący popularność język zapytań XQuery. Omówiono, wspierając się licznymi przykładami, podstawowe konstrukcje języka. Przedstawiono możliwości wykorzystania Xquery oraz środowiska w których można użyć tego języka, ze szczególnym uwzględnieniem technologii Oracle.

Tutorial jest adresowany przede wszystkim do użytkowników relacyjnych baz danych i języka SQL, dlatego też cechy Xquery porównano z cechami SQL; podjęta też została próba odpowiedzi na tytułowe pytanie.

Informacja o autorze. Dr inż. Tomasz Traczyk – adiunkt w Instytucie Automatyki i Informatyki Stosowanej na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. Specjalizuje się w projektowaniu systemów informacyjnych z bazami danych oraz w zastosowaniach technologii XML. Jest twórcą lub współtwórcą wielu systemów informacyjnych dla przemysłu, edukacji i nauki. Prowadzi działalność naukowo-dydaktyczną w zakresie baz danych, inżynierii oprogramowania i systemów internetowych, a także szkolenia (m.in. w ośrodku szkoleniowym Oracle Polska) dotyczące projektowania baz danych, technologii CASE, XML i UML.

XQuery – konkurencja dla SQL?

- ❖ Wprowadzenie – czym jest XQuery?
- ❖ Podstawy składni języka XQuery
- ❖ Użycie XQuery
- ❖ XQuery w Oracle 10g r. 2
- ❖ Oracle XML Query Service
- ❖ Podsumowanie



Wprowadzenie – czym jest XQuery



Po co XQuery?

Po co język zapytań do XML?

- Wyszukiwanie w
 - dokumentach XML
 - » z plików
 - » z baz danych
 - innych źródłach w XML-u
 - » strony WWW
 - » usługi sieciowe (*Web services*)
 - » bazy danych
 - » itp.
- Przetwarzanie danych z tych źródeł
 - obliczenia
 - łączenie danych
 - kształtowanie formatu wyniku
 - przetwarzanie w stylu bliższym programistom (w porównaniu do XSLT)



Język zapytań dla XML

Czego oczekujemy od języka zapytań?

1. Powinien umożliwiać zapisanie złożonych warunków wyszukiwania
2. Powinien być deklaratywny
3. Powinien zwracać wyniki w tej samej formie, w jakiej otrzymuje dane źródłowe
4. Powinien umożliwiać sortowanie
5. Powinien umożliwiać wykonywanie przekształceń wyszukiwanej informacji, w tym agregacji

Dlaczego nie wystarczy XPath?

- Języki zapytań do baz danych (np. SQL, OQL) spełniają te oczekiwania
- XPath spełnia tylko 1. i 2.
- Ale na bazie XPath można zbudować język zapytań mający pozostałe pożądane cechy



Czym jest XQuery?

XQuery (XML Query)

- Deklaratywny język zapytań
 - przeznaczony do wyszukiwania w źródłach XML-owych
 - bazujący na XPath 2.0
 - umożliwiającą konstruowanie dokumentów XML
 - o możliwościach porównywalnych z językami zapytań dla baz danych
- Język funkcyjny
 - wolny od efektów ubocznych
 - z możliwością silnego typowania
- Składnia XQuery
 - ma być czytelna dla człowieka
 - » nie jest dialektem XML



Przykład

Przeszukiwany dokument:

```
<?xml version = '1.0' encoding = 'windows-1250'?>
<publikacje>
  <publikacja id="360" typ="REFERAT">
    <tytul>Język XML w aplikacjach z bazami danych</tytul>
    <autorzy>
      <autor afiliacja="WEiTI">
        <nazwisko>Traczyk</nazwisko><imiona>Tomasz</imiona>
      </autor>
      <autor afiliacja="WEiTI">
        <nazwisko>Macewicz</nazwisko><imiona>Włodzimierz</imiona>
      </autor>
    </autorzy>
    <cechy>
      <cecha id="TYTUŁ_KONFER">
        IV Konferencja użytkowników i developerów Oracle
      </cecha>
      <cecha id="MIEJSCE_KONF">Kościelisko</cecha>
      <cecha id="ORGANIZATOR">PLOUG</cecha>
      <cecha id="STRONY">133-146</cecha>
    </cechy>
    <rok>1998</rok>
  </publikacja>
  <publikacja>
    ...
  </publikacja>
  ...
</publikacje>
```

Uproszczony fragment danych wyjściowych z systemu informacji o publikacjach na Wydziale Elektroniki i Techniki Informatycznej Politechniki Warszawskiej



Przykład, c.d.

Zapytanie XQuery:

– wyszukiwanie publikacji z roku 1998

`<odpowiedz>` Zmienna \$b przebiega sekwencję węzłów będącą wynikiem ścieżki XPath

```
{
  for $b in doc("/public/publikacje.xml")//publikacja
  where $b//rok="1998"
  return
    <referat rok="{ $b/rok}" miejsce="{ $b//cecha[@id='MIEJSCE_KONF']}">
      { $b/tytul }
    </referat>
}
```

`</odpowiedz>`



Przykład, c.d.

Zapytanie XQuery:

– wyszukiwanie publikacji z roku 1998

`<odpowiedz>` Warunek where pozwala dodatkowo ograniczyć wynik

```
{
  for $b in doc("/public/publikacje.xml")//publikacja
  where $b//rok="1998"
  return
    <referat rok="{ $b/rok}" miejsce="{ $b//cecha[@id='MIEJSCE_KONF']}">
      { $b/tytul }
    </referat>
}
```

`</odpowiedz>`



Przykład, c.d.

Zapytanie XQuery:

– wyszukiwanie publikacji z roku 1998

```
<odpowiedz>
{
  for $b in doc("/public/publikacje.xml")//publikacja
  where $b//rok="1998"
  return
    <referat rok="{ $b/rok}" miejsce="{ $b//cecha[@id='MIEJSCE_KONF']}">
      { $b/tytul }
    </referat>
}
</odpowiedz>
```

Klauzula `return` kształtuje postać wyniku;
w treści wyniku można użyć wyrażeń odwołujących się do zmiennych



Przykład, c.d.

Zapytanie XQuery:

– wyszukiwanie publikacji z roku 1998

```
<odpowiedz>
{
  for $b in doc("/public/publikacje.xml")//publikacja
  where $b//rok="1998"
  return
    <referat rok="{ $b/rok}" miejsce="{ $b//cecha[@id='MIEJSCE_KONF']}">
      { $b/tytul }
    </referat>
}
</odpowiedz>
```

Cały wynik zapytania można „wmontować” w strukturę XML



Przykład, c.d.

Zapytanie XQuery:

– wyszukiwanie publikacji z roku 1998

```
<odpowiedz>
{
  for $b in doc("/public/publikacje.xml")//publikacja
  where $b//rok="1998"
  return
    <referat rok="{ $b/rok}" miejsce="{ $b//cecha[@id='MIEJSCE_KONF']}">
      { $b/tytuł }
    </referat>
}
</odpowiedz>
```

Wynik:

```
<odpowiedz>
<referat rok="1998" miejsce="Kościelisko">
  <tytuł>
    Język XML w aplikacjach z bazami danych
  </tytuł>
</referat>
...
</odpowiedz>
```



Założenia XQuery

- ✓ Czytelność składni języka dla człowieka
- ✓ Deklaratywność
- ✓ Niezależność od protokołów i środowisk użycia
- ✓ Uwzględnienie XML-owego modelu danych, przestrzeni nazw i schematów XML
- ✓ Działanie także bez dostępu do schematów dokumentów
- ✓ Silne typowanie (typy proste i złożone)
- ✓ Istnienie kwantyfikatorów
- ✓ Możliwość operowania na hierarchiach i sekwencjach
- ✓ Możliwość łączenia informacji z wielu źródeł
- ✓ Możliwość wykonywania agregacji
- ✓ Transformacje dokumentów XML i możliwość tworzenia wynikowych struktur XML
- ✓ Możliwość nawigowania po odwołaniach (referencjach) do identyfikatorów



Standaryzacja XQuery

Ciało standaryzujące

- Standaryzacja jest prowadzona przez *XML Query Working Group* w ramach W3C
- Język XQuery jest silnie zależny od XPath 2.0 – standaryzacją zajmuje się ta sama grupa

Stan standaryzacji

- Główne dokumenty dotyczące standaryzacji XQuery mają status *Candidate Recommendation* (listopad 2005)
- Proces standaryzacyjny wciąż trwa, ale jest bliski ukończenia



Podstawy składni języka XQuery



Podstawowe zasady języka

Język funkcyjny

- Wszystkie konstrukcje języka są wyrażeniami zwracającymi wartość
- Można je zagnieżdżać
- Wartości zwracane przez wyrażenia są tzw. sekwencjami (w sensie XPath 2.0)

Charakter przetwarzanej informacji

- Na wejściu tzw. sekwencja
 - dokument lub fragment
 - kolekcja dokumentów lub fragmentów
- Na wyjściu też sekwencja
 - dokument lub fragment

Format języka

- Swobodny
- Wielkość liter ma znaczenie
- Słowa kluczowe pisze się małymi literami

Sekwencje

- Uporządkowany ciąg jednostek (*items*):
 - węzłów
 - wartości atomowych
- Pojedyncza jednostka jest tożsama z jednoelementową sekwencją
- Sekwencji nie można zagnieżdżać
 - wynik wyrażenia z zagnieżdżeniem zostaje „spłaszczony”
- Porządek w sekwencji
 - zgodny z tzw. porządkiem dokumentu
 - chyba że użyto klauzuli `order by`
- Jawny zapis sekwencji:
(*jednostka1, jednostka2, ...*)



Podstawowe elementy języka

Wyrażenia

- Wartości wszystkich wyrażeń są sekwencjami
- Wyrażenia zawierają
 - stałe i jawnie zapisane sekwencje
 - odwołania do zmiennych
 - operatory
 - wywołania funkcji wbudowanych i użytkownika
 - ścieżki XPath
 - wyrażenia FLWOR
 - wyrażenia warunkowe

Zmienne

- Oznacza się $\$nazwa$
- Mogą, ale nie muszą, być deklarowane
 - mogą być silnie typowane

Wartości logiczne

- Wartości `true` i `false`
- Operatory `or` i `and` oraz funkcja `not()`
- Efektywna wartość logiczna wyrażeń
 - fałsz jeśli:
 - » liczba 0 lub NaN
 - » pusta sekwencja
 - » pusty napis
 - prawda w przeciwnym wypadku
 - nie dotyczy to jawnych konwersji `cast as`

Komentarze

- Oznaczenie: `(: ... :)`
- Komentarze można zagnieżdżać



Podstawowe elementy języka, c.d.

Operatory i porównania

- Typowe operatory arytmetyczne i logiczne
- Porównania prostych wartości:
eq, ne, lt, le, gt, ge
- Porównania ogólne: =, !=, <, <=, >, >=
 - połączenie porównania elementów sekwencji z kwantyfikatorem *exists*
- Porównywanie tożsamości węzłów
 - identyczność: *is*
 - porządek: <<, >>
- Porównywanie całych gałęzi
(z węzłami podrzędnymi): *deep-equal()*

Kwantyfikatory

- Postać:
 - *every \$zmienna in sekwencja satisfies warunek*
 - *some \$zmienna in sekwencja satisfies warunek*
- Badają elementy sekwencji
- Stosowane np. w klauzuli *where*

Wyrażenia warunkowe

- Postać:
 - if warunek*
 - then wyrażenie1*
 - else wyrażenie2*
- Część *else* jest obowiązkowa



Podstawowe elementy języka, c.d.

Funkcje wbudowane

- Wielka liczba funkcji wbudowanych XPath 2.0
- Obejmują m.in.
 - operacje na sekwencjach: *insert-before()*, *remove()*, *reverse()*, *empty()*, *union()*, *intersect()*
 - operacje na węzłach: *node-before()*, *name()*, *local-name()*
 - operacje na wartościach: *string()*, *data()*
 - operacje arytmetyczne: *abs()*, *floor()*, *round()*
 - agregacje: *count()*, *sum()*, *min()*
 - operacje na tekstach: *concat()*, *substring()*, *string-length()*, *upper-case()*, *contains()*, *starts-with()*
 - wyrażenia regularne: *matches()*, *replace()*
 - » w Oracle10g r. 2 implementacja niezgodna ze specyfikacją
 - operacje na datach i czasie: *date-equal()*, *date-less-than()*, *time-equal()*, *time-less-than()*, *year-from-date()*, *day-from-date()*, *adjust-time-to-timezone()*
 - operacje na przedziałach czasu: *add-yearMonthDurations()*, *add-dayTimeDurations()*
 - obsługę błędów i śledzenie



Budowa zapytania XQuery

XPath

- Najprostszym zapytaniem XQuery jest sama ścieżka XPath
- Wszystkie zapytania bazują na ścieżkach XPath
 - ścieżka pozwala „dostać się” do właściwych danych w strukturze hierarchicznej
 - można w niej umieścić złożone warunki
- W XQuery można korzystać ze składni ścieżek nieco okrojonej w stosunku do XPath 2.0

Źródła danych

- Ścieżka XPath użyta w XQuery musi zaczynać się od źródła danych
 - zmiennej
 - funkcji – źródła danych
- Funkcje wbudowane udostępniające dane zewnętrzne:
 - `doc(URI)` – udostępnia dokument
 - » Oracle10g:
URI dokumentu w repozytorium XDB
 - `collection(URI)` – udostępnia kolekcję
 - » Oracle10g:
URI folderu w repozytorium XDB



Wyrażenia FLWOR

Wyrażenia FLWOR

Przykład: wypisanie w kolejności chronologicznej tytułów publikacji mających tylko jednego autora

```
for $p in doc(
  "/public/publikacje.xml")//publikacja
let $a := $p/autorzy/autor
where count($a) = 1
order by $p/rok
return $p/tytuł
```

Wynik (fragment, a nie dokument!):

```
...
<tytuł>Język XSL</tytuł>
<tytuł>Czy już warto używać XML?</tytuł>
<tytuł>Schematy XML</tytuł>
...
```

Zasady

- Wszystkie klauzule są opcjonalne
 - musi wystąpić `for` lub `let`
- Wyrażenia FLWOR można zagnieżdżać

Klauzula `for`

- Służy do organizowania iteracji
- Przypisuje do zmiennej kolejne elementy sekwencji – wyniku wyrażenia

Klauzula `let`

- Służy do wykonywania podstawień
- Przypisuje do zmiennej całą wartość wyrażenia na raz

Klauzula `where`

- Określa warunki filtrowania sekwencji zwróconej przez klauzule `for` i `let`

Klauzula `order by`

- Określa sortowanie tej sekwencji

Klauzula `return`

- Kształtuje postać wyniku



Wyrażenia FLWOR, c.d.

Zagnieżdżanie wyrażen FLWOR

- Pozwala tworzyć wielopoziomowe struktury

```
<spis>
  {
    for $p in doc("/public/publikacje.xml")//publikacja
    return
      <pozycja tytul="{ $p/tytul}">
        {
          for $a in $p//autor
          return
            <autor> { $a/nazwisko/text() } {" "} { $a/imiona/text() } </autor>
        }
      </pozycja>
  }
</spis>
```

Wynik:

```
<spis>
  <pozycja tytul="Język XML w aplikacjach z bazami danych">
    <autor>Traczyk Tomasz</autor>
    <autor>Macewicz Włodzimierz</autor>
  </pozycja>
  ...
</spis>
```



Budowa zapytania XQuery, c.d.

Konstruowanie struktur wynikowych

- Umieszczanie (literalnie) elementów wynikowej struktury („szablonu”) w treści zapytania
 - wartości wyrażen XQuery wstawia się do „szablonu” przez ujęcie wyrażen w { }
- Stosowanie *computed constructors*
 - konstruktory:
 - » element, attribute, text
 - » pi, comment
 - umożliwiają wyliczanie
 - » nazw
 - » wartości wstawianych jednostek

Przykład:

```
for $p in doc("/public/publikacje.xml")//publikacja
return
  <publikacja tytul="{ $p/tytul/text()}">
    {
      for $c in $p//cecha
      return
        element {lower-case($c/@id)}
        {
          (
            if ($c/@id="DATA_KONFER") then
              attribute rok { $p/rok/text() }
            else
              nil
          )
          ; $c/text()
        }
    }
  </publikacja>
```

Wynik:

```
<publikacja
  tytul="Język XML w aplikacjach z bazami danych">
  <tytuł_konfer>
    IV Konferencja użytkowników i developerów Oracle:
    Ewolucja systemów informatycznych: dane, sprzęt,
    oprogramowanie i aplikacje
  </tytuł_konfer>
  <miejsce_konf>Kościelisko</miejsce_konf>
  <data_konfer rok="1998">XI 1998</data_konfer>
  <organizator>PLOUG</organizator>
  <strony>133-146</strony>
</publikacja>
```



Łączenie danych

Łączenie danych z różnych źródeł

- XQuery dobrze nadaje się do łączenia danych z różnych źródeł
- Można wykonywać np.
 - operacje podobne do złączeń relacyjnych
 - wyszukiwania zagnieżdżone

Przykład:

- Dodatkowy dokument:

```
<instytucje>
  <instytucja kod="WEiTI">Wydział Elektroniki i Technik Informacyjnych</instytucja>
  <instytucja kod="IAiIS">Instytut Automatyki i Informatyki Stosowanej</instytucja>
  ...
</instytucje>
```

- Łączone będą dane
 - publikacji
 - instytucji, w których afiliowani są autorzy



Łączenie danych, c.d.

Złączenie danych z różnych źródeł

```
for $p in doc("publik.xml")//publikacja,
    $i in doc("instyt.xml")//instytucja
where $i/@kod = $p//autor/@afiliacja
return
  <pozycja>
    {$p/tytul}
    {$i}
  </pozycja>
```

Wynik:

```
<pozycja>
  <tytul>Język XML w aplikacjach z bazami danych</tytul>
  <instytucja kod="WEiTI">Wydział Elektroniki i Technik Informacyjnych</instytucja>
</pozycja>
<pozycja>
  <tytul>Język XML w aplikacjach z bazami danych</tytul>
  <instytucja kod="IAiIS">Instytut Automatyki i Informatyki Stosowanej</instytucja>
</pozycja>
<pozycja>
  <tytul>Język XML w aplikacjach z bazami danych - po roku</tytul>
  <instytucja kod="WEiTI">Wydział Elektroniki i Technik Informacyjnych</instytucja>
</pozycja>
...
```

- Uzyskano powtórzenia, jak to przy złączeniu



Łączenie danych, c.d.

Zagnieżdżenie danych z różnych źródeł

```
for $p in doc("/public/publikacje.xml")//publikacja
return
  <pozycja>
    {$p/tytul}
    {
      for $i in doc("/public/institucje.xml")//instytucja
      where $i/@kod = $p//autor/@afiliacja
      return $i
    }
  </pozycja>
```

Wynik:

```
<pozycja>
  <tytul>Język XML w aplikacjach z bazami danych</tytul>
  <instytucja kod="WEiTI">Wydział Elektroniki i Technik Informatycznych</instytucja>
  <instytucja kod="IAiIS">Instytut Automatyki i Informatyki Stosowanej</instytucja>
</pozycja>
<pozycja>
  <tytul>Język XML w aplikacjach z bazami danych - po roku</tytul>
  <instytucja kod="WEiTI">Wydział Elektroniki i Technik Informatycznych</instytucja>
</pozycja>
...
```



Grupowanie i agregacje

Grupowanie

- Brak specyficznych konstrukcji językowych do grupowania
- Efekt grupowania można uzyskać np. przez odpowiednie zagnieżdżanie wynikowych elementów XML

```
for $a in distinct-values( doc("/public/publikacje.xml")//autor/@afiliacja )
return
  <instytucja kod="{ $a }">
    {
      for $p in doc("/public/publikacje.xml")//publikacja[./autor/@afiliacja = $a]
      return <publikacja>{$p/tytul/text()}</publikacja>
    }
  </instytucja>
```

Wynik:

```
<instytucja kod="IAiIS">
  <publikacja>Język XML w aplikacjach z bazami danych</publikacja>
</instytucja>
<instytucja kod="WEiTI">
  <publikacja>Język XML w aplikacjach z bazami danych</publikacja>
  <publikacja>Język XML w aplikacjach z bazami danych - po roku</publikacja>
  ...
</instytucja>
...
```



Grupowanie i agregacje, c.d.

Unikalność

- Funkcja `distinct-values()`
 - eliminuje powtórzenia z wyniku zapytania
 - używana np. w części `in` klauzuli `for`
 - działa na danych prostych typów

Funkcje agregujące

- `sum()`, `avg()`, `count()`, `max()` i `min()`

Przykład: zliczenie publikacji związanych z poszczególnymi instytucjami

```
for $i in doc("/public/instytucje.xml")//instytucja
let $p := doc("/public/publikacje.xml")//publikacja[./autor/@afiliacja = $i/@kod]
return
  <instytucja liczba="{count($p)}">
    {$i/text()}
  </instytucja>
```

Wynik:

```
<instytucja liczba="7">Wydział Elektroniki i Technik Informatycznych</instytucja>
<instytucja liczba="1">Instytut Automatyki i Informatyki Stosowanej</instytucja>
```



Modularyzacja

Moduły

- Kod może być podzielony na moduły
 - zapytań
 - biblioteczne
- Moduły biblioteczne
 - zawierają obowiązkowo deklarację `module`
 - włączane do modułów zapytań deklaracją `import module`
- Zmienne zewnętrzne (`external`)
 - pełnią rolę parametrów
 - wartości są dostarczane przez środowisko wykonania
- Oracle10g r. 2
 - nie zaimplementowano modularyzacji
 - » wszystko w module głównym

Budowa modułu

- Prolog
 - deklaracja `xquery` (opcjonalna)
 - » wersja
 - » strona kodowa
 - deklaracja `module` (w modułach bibliotecznych)
 - » docelowa przestrzeń nazw
 - deklaracje `import schema`
 - deklaracje `import module`
 - deklaracje przestrzeni nazw
 - deklaracje zmiennych
 - deklaracje funkcji
- Ciało
 - zapytanie (wyrażenie XQuery)



Modularyzacja, c.d.

Funkcje użytkownika

- Możliwe jest
 - definiowanie funkcji użytkownika
 - wykorzystanie ich w zapytaniach
- Funkcje mogą mieć parametry
- Parametry i wartość funkcji mogą być typowane
- Nie można
 - przeciążać nazw funkcji użytkownika
 - definiować funkcji o zmiennej liczbie parametrów
- Funkcje lokalne muszą być umieszczone w predefiniowanej przestrzeni nazw `local`

Przykład:

```
declare function
  local:liczba-autorow($param)
  { count($param//autor) }
;

for $p in doc("/public/publikacje.xml"
) // publikacja
let $c := local:liczba-autorow($p)
return
  <pozycja autorow="{ $c }">
    { $p / tytul }
  < / pozycja >
```

Wynik:

```
<pozycja autorow="2">
  <tytul>Język XML w aplikacjach
    z bazami danych</tytul>
< / pozycja >
<pozycja autorow="1">
  <tytul>Język XML w aplikacjach
    z bazami danych - po roku</tytul>
< / pozycja >
<pozycja autorow="1">
  <tytul>Język XSL</tytul>
< / pozycja >
...
```



Typowanie

Typy

- Gdzie przydaje się typowanie
 - sortowanie i porównywanie
 - operacje wymagające konwersji
 - wykrywanie błędów w zapytaniach
- Typowanie w XQuery
 - w oparciu o schemat dokumentu
 - gdy brak schematu
 - » jawne deklaracje typu
 - » domyślne i jawne konwersje
 - » w ostateczności: przetwarzanie reprezentacji tekstowej

Dostępne typy

- Typy predefiniowane XML Schema
- Typy predefiniowane XPath 2.0, np. `xd:untyped`, `xd:untypedAtomic`, `xd:anyAtomicType`
- Typy ze schematu powiązanego z danym dokumentem

Deklaracje typów

- Import schematów: `import schema`
- Jawne deklaracje typów
 - zmiennych
 - parametrów i wyników funkcji
- Jawne określenie typu wyrażeń: `treat as` (statyczne, bez konwersji)



Typowanie c.d.

Typowanie dynamiczne

- Walidacja na podstawie schematu
 - wyrażenie `validate`
 - » działa na węźle
 - » sprawdza zgodność ze schematem
 - » przypisuje dynamicznie typy
 - tryby walidacji
 - » `strict` – elementy spoza schematu powodują błąd
 - » `lax` – elementy spoza schematu pozostają nieznanego typu
- Sprawdzenia w czasie wykonania
 - wykonalności operacji
 - poprawności jawnych i domyślnych konwersji
 - zgodności typów z deklaracjami

Operacje na typach

- Konstrukcje XPath 2.0 pozwalające wyszukać elementy/atributy określonego typu, np. `element(*, nazwa_typu)`
- Jawne konwersje typów: `cast as`
- Badanie czy element sekwencji jest danego typu: `instance of`
- Przetwarzanie zależne od typu: `typeswitch... case... default...`
- Wydobywanie wartości:
 - znakowej: `fn:string()` – daje konkatenację reprezentacji znakowych
 - typowanej: `fn:data()` – daje sekwencję wartości typowanych

Typowanie statyczne

- Opcjonalne (*Static Typing Feature*)
- Polega na sprawdzaniu poprawności typów w czasie kompilacji zapytania XQuery
- Zaimplementowane w Oracle 10g r. 2



XQuery a przestrzenie nazw

Predefiniowane przestrzenie i prefiksy

- `xs` – typy danych XML Schema
- `xsi` – prefiks atrybutu wskazującego instancję schematu
- `xdt` – typy danych XPath 2.0
- `fn` – funkcje wbudowane
- `local` – funkcje użytkownika zdefiniowane lokalnie w module zapytania
- `ora` – funkcje rozszerzające dodane przez Oracle

Użycie przestrzeni nazw

- Przetwarzanie XQuery wykorzystuje przestrzenie nazw
- Użycie przestrzeni nazw w definicjach funkcji użytkownika jest obowiązkowe
- Deklaracje przestrzeni nazw
 - `declare namespace` – definiuje inne przestrzenie nazw (i prefiksy)
 - `declare default element namespace` – określa przestrzeń domyślną
 - deklaracje działają i na dane wejściowe, i wyjściowe

```
declare namespace p="urn://ploug.org";
for $i in doc("/public/instytucje.xml")//instytucja
return <p:instytucja>{$i/text()}</p:instytucja>

<p:instytucja xmlns:p="urn://ploug.org">
  Wydział Elektroniki i Technik Informatycznych
</p:instytucja>
...
```



XQueryX

Co to takiego?

- Reprezentacja XQuery w składni XML
- Nieczytelna dla człowieka, ale łatwa do przetwarzania za pomocą narzędzi XML-owych

Przykład:

```
for $t in doc("/public/publikacje.xml")/publikacje/publikacja/tytul return $t
```

```
<xq:query xmlns:xq="http://www.w3.org/2001/06/xquery">
  <xq:flwr>
    <xq:forAssignment variable="$t">
      <xq:step axis="CHILD">
        <xq:function name="doc">
          <xq:constant datatype="CHARSTRING">publik.xml</xq:constant>
        </xq:function>
        <xq:identifier>publikacje</xq:identifier>
      </xq:step>
      <xq:step axis="CHILD">
        <xq:identifier>publikacja</xq:identifier>
      </xq:step>
      <xq:step axis="CHILD">
        <xq:identifier>tytul</xq:identifier>
      </xq:step>
    </xq:forAssignment>
    <xq:return>
      <xq:variable>$t</xq:variable>
    </xq:return>
  </xq:flwr>
</xq:query>
```



Użycie XQuery



Do czego XQuery?

Funkcje

- Wyszukiwanie
 - w źródłach XML-owych
 - w źródłach mieszanych
- Transformacje
 - przekształcanie XML
 - scalanie XML z wielu źródeł
- Agregacje i inne obliczenia
- Integracja danych heterogenicznych
- Usługi sieciowe (*Web Services*)
- Dynamiczne serwisy WWW
 - w połączeniu z XSLT i CSS

Przetwarzane informacje

- Dokumenty XML
 - w tym informacja semistrukuralna
- Dane z baz danych
 - przekształcone do XML
- Usługi sieciowe (*Web Services*)

Środowiska

- Systemy inter- i intranetowe
- Systemy wielowarstwowe
 - warstwa DBMS
 - warstwa *mid-tier*
- Systemy rozproszone
 - zwłaszcza bazujące na SOAP
- Systemy EDI i *e-business*
 - zwłaszcza B2B



XQuery a XSLT

XSLT

- Programowanie deklarytywne (szablony) i proceduralne (np. iteracje)
- Mało podatny na optymalizację zapytań
- Przeznaczony pierwotnie do formatowania
- Przetwarza w zasadzie jedno źródło
- Dość trudny do nauczenia
- Dość nienaturalny sposób programowania
- Rozwlekły i mało czytelny kod

XQuery

- Programowanie deklarytywne
- Bardziej podatny na optymalizację zapytań
- Przeznaczony do przetwarzania danych
- Może przetwarzać wiele źródeł
- Łatwiejszy do nauczenia
- Sposób programowania bliższy programistom, zwłaszcza znającym inne języki zapytań
- Kod dość zwarty, ale czytelny



XQuery a SQL

XQuery

- Deklaratywny
- Podatny na optymalizację zapytań
- Tylko zapytania
- Przetwarza dane hierarchiczne
- Zwraca sekwencje (z węzłami)
- Kolejność danych w odpowiedzi zgodna z kolejnością dokumentu (jeśli bez sortowania)
- Dobry do przetwarzania informacji strukturalnej i semistukturalnej

SQL

- Deklaratywny
- Podatny na optymalizację zapytań
- Zapytania, DML i DDL
- Przetwarza dane relacyjne
- Zwraca relacje (tabele)
- Kolejność danych w odpowiedzi losowa (jeśli bez sortowania)
- Przetwarza tylko informację strukturalną



Implementacje XQuery

Najbardziej znane implementacje

- *Open-source* itp.
 - Saxon
 - Qizx/open
 - Qexo (GNU)
- Komercyjne
 - Oracle 10g r. 2
 - MS SQL Server 2005
 - BEA AquaLogic Data Services Platform

BumbleBee

- Testy implementacji XQuery
- Rozbudowany zestaw zapytań testowych
- Narzędzia automatyzujące testowanie

Narzędzia do XQuery

- Altova XMLSpy XQuery Development
- Stylus Studio XQuery Tools
- Oracle JDeveloper 10g



Konteksty użycia XQuery

Zapytania interaktywne

- Mogą być składane za pomocą „konsol” XQuery
- Służą do zapytań *ad-hoc*, nauki języka, testowania zapytań itp.

Usługi WWW (*Web Services*)

- XQuery zwraca XML, jest więc idealnym językiem zapytań dla *WebServices*
- Oracle10g r. 2 udostępnia wyniki zapytań XQuery przez protokół SOAP

Przetwarzanie w DBMS

- XQuery w postaci SQL/XML daje się zanurzać w PL/SQL

Przetwarzanie w warstwie *mid-tier*

- Interfejsy API do języka Java

XQuery a Java

- Istniejące implementacje mają własne API
 - OJXQI (*Oracle Java XQuery API*)
- Można użyć JDBC i funkcji SQL/XML
- Opracowywany jest standard interfejsu programistycznego XQJ
 - zaangażowane Oracle, IBM, BEA, Sun, Sybase i in.
 - „ideologia” zgodna z JDBC
 - „stylizacja” API zgodna z JDBC i JAXP
 - nawigacyjny mechanizm dostępu („kursory”)
 - przewidziana jest obsługa transakcji



XML i XQuery w integracji danych

XML jako format integracji

- Bardzo odpowiedni format do integracji
- Duża siła wyrazu
 - może reprezentować dowolne dane mające postać tekstową
- Elastyczność
- Łatwo dostępne narzędzia
- Własny język wielu źródeł danych
 - serwisy w XML, XHTML
 - *Web Services*
- Stosunkowo łatwa konwersja innych źródeł
 - bazy danych
 - arkusze kalkulacyjne
 - HTML
 - itp.

XQuery jako narzędzie do integracji

- Dobre narzędzie do integracji danych XML
- Cechy języka zapytań XQuery
 - deklaratywny
 - » łatwe pisanie zapytań
 - nie trzeba kodować algorytmów
 - uniwersalny
 - » łatwe łączenie dane z wielu źródeł
 - » dowolne formatowanie wyniku
 - wydajny
 - » zapytania podlegają optymalizacji
- XQuery umożliwia
 - integrację źródeł danych
 - » w czasie rzeczywistym
 - » bez replikowania informacji
 - efektywność
 - » dobry stosunek wydajności i jakości wyniku do nakładów
 - elastyczność integracji
 - » łatwe rozszerzanie
 - » łatwa adaptacja do zmian danych źródłowych



XQuery w Oracle 10g r. 2



XQuery w technologii Oracle

Oracle10g release 2

- Wbudowane rozwiązania zgodne z *XQuery 1.0 Working Draft April 2005*
 - z niewielkimi odstępstwami
- Nowe funkcje SQL/XML (wg SQL:2005)
 - XMLQuery
 - » zwraca w wyniku fragment dokumentu (o typie XMLType)
 - XMLTable
 - » elementy wynikowej sekwencji umieszcza w wierszach wirtualnej tabeli, która może być wykorzystana w klauzuli FROM zapytań SQL
- Dodatkowe funkcje rozszerzające XPath
- Nowe polecenie SQL*Plus: xquery
 - Przykłady zapytań XQuery, umieszczone w niniejszej prezentacji, zostały sprawdzone przy użyciu tej właśnie implementacji XQuery

Oracle JDeveloper 10g release 3

- Wsparcie dla składni XQuery (w tym wbudowany schemat)
- Wykonywanie zapytań XQuery wprost z IDE

Oracle Application Server 10g release 3

- Technologia XQS (*XML Query Service*, dawniej XDS) w AS10g r. 3



Konsola XQuery

SQL*Plus 10g r. 2

- Nowe polecenie xquery umożliwia interaktywne wykonywanie zapytań XQuery

```
SQL*Plus: Release 10.2.0.1.0 - Production
Pojłączono z:
Personal Oracle Database 10g Release 10.2.0.1.0 - Production
SQL> xquery doc("/public/publikacje.xml")//publikacja[id="354"]//autor
2 /
Result Sequence
-----
<autor afiliacja="WEITI"><nazwisko>Traczyk</nazwisko><imiona>Tomasz</imiona></autor>
```

- Dodano kilka nowych komend set, sterujących działaniem polecenia xquery



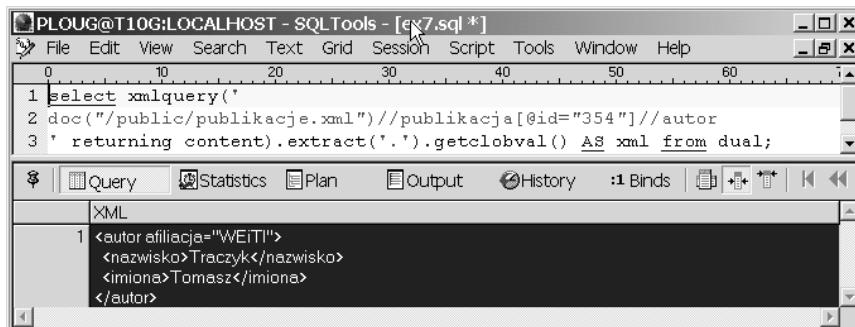
XQuery w SQL/XML

Funkcja xmlquery()

- Przyjmuje tekstowo zapytanie XQuery
- Zwraca wynikowy XML w postaci XMLType
 - klauzula returning content jest obowiązkowa
 - w Oracle10g nie ma opcji returning sequence
- W celu otrzymania wyniku tekstowego należy użyć metody getclobval() lub getstringval()
- Dodatkowo można użyć metody extract() by sformatować wynik

```
select xmlquery('doc("/public/publikacje.xml")//publikacja'
  returning content).extract('.').getclobval() as xml
from dual;
```

- W ten sposób można użyć dowolnej konsoli SQL jako konsoli XQuery



XQuery w SQL/XML, c.d.

Funkcja `xmltable()`

- Przyjmuje tekstowo zapytanie XQuery
- Do deklarowania przestrzeni nazw w służy specjalne wyrażenie `xmltable(xmlnamespaces('urn://ploug.org' as "p"), ...)`
- Kolejne elementy wynikowej sekwencji umieszcza w wierszach wirtualnej tabeli
- Tej wirtualnej tabeli można użyć w klauzuli `from` zapytania SQL

```
select x.column_value.getclobval()
from xmltable('
  for $p in doc("/public/publikacje.xml")//publikacja
  return
    <publikacja>
      {$p/tytul}
      <autor1>
        {$p//autor[1]/nazwisko/text(), " ",
         $p//autor[1]/imiona/text()}
      </autor1>
    </publikacja>
') x;
```

<publikacja> <tytul>Język XML w aplikacjach z bazami danych</tytul> <autor1>Traczyk Tomasz</autor1> </publikacja>
<publikacja> <tytul>Język XML w aplikacjach z bazami danych - po roku</tytul> <autor1>Traczyk Tomasz</autor1> </publikacja>
...



XQuery w SQL/XML, c.d.

Funkcja `xmltable()`, c.d.

- Wynik zapytania może być interpretowany jako zbiór kolumn
 - sterowanie klauzulami `columns` i `path`
 - jeśli źródło jest w postaci „kanonicznej”, klauzulę `path` można pominąć
- Nadaje się więc do transformacji XML w dane relacyjne

```
select x.*
from xmltable('
  for $p in doc("/public/publikacje.xml")//publikacja
  return
    <publikacja>
      {$p/tytul}
      <autor1>
        {($p//autor[1]/nazwisko/text(), " ",
         $p//autor[1]/imiona/text())}
      </autor1>
    </publikacja>'
  columns tytul VARCHAR2(100) path '//tytul' ,
          autor1 VARCHAR2(50) path '//autor1'
) x;
```

<u>TYTUL</u>	<u>AUTOR1</u>
Język XML w aplikacjach z bazami danych	Traczyk Tomasz
Język XML w aplikacjach z bazami danych - po roku	Traczyk Tomasz
...	...



Źródła danych dla XQuery w Oracle 10g r. 2

Repozytorium XML DB

- `doc(URI)` – udostępnia dokument przez URI dokumentu w repozytorium XDB
- `collection(URI)` – udostępnia kolekcję przez URI folderu w repozytorium XDB

Przykład: wypisanie nazwy elementu głównego

```
doc("/public/publikacje.xml")/name(./*)
```

zwraca:

publikacje

```
collection("/public")/name(./*)
```

zwraca:

instytucje publikacje



Źródła danych dla XQuery w Oracle 10g r. 2, c.d.

Tabela/perspektywa

- Funkcja `ora:view` rozszerzająca XPath
 - udostępnia dane z tabeli lub perspektywy w postaci „kanonicznej” XML
 - » sekwencja elementów `<ROW>`, zawierających elementy nazwane tak jak kolumny
 - » nazwy wielkimi literami
 - dla danych relacyjno-obiektowych tworzy odpowiednie struktury zagnieżdżone

Przykłady:

Tabela: `CREATE TABLE instytucje (kod VARCHAR2(12), nazwa VARCHAR2(100));`

Zapytanie: `ora:view("instytucje")/.`

```
<ROW>
  <KOD>WEiTI</KOD>
  <NAZWA>Wydział Elektroniki i Technik Informatycznych</NAZWA>
</ROW>
<ROW>
  <KOD>IAiIS</KOD>
  <NAZWA>Instytut Automatyki i Informatyki Stosowanej</NAZWA>
</ROW>
```

Zapytanie: `ora:view("instytucje")//NAZWA`

```
<NAZWA>Wydział Elektroniki i Technik Informatycznych</NAZWA>
<NAZWA>Instytut Automatyki i Informatyki Stosowanej</NAZWA>
```

Tabela r-o: `CREATE TABLE instytucje_xml (kod VARCHAR2(12), opis XMLType)`

Zapytanie: `ora:view("instytucje_xml")/.`

```
<ROW>
  <KOD>WEiTI</KOD>
  <OPIS>
    <instytucja kod="WEiTI">Wydział Elektroniki i Technik Informatycznych</instytucja>
  </OPIS>
</ROW>
```



Źródła danych dla XQuery w Oracle 10g r. 2, c.d.

Obiekty XMLType

- Pochodzące np. z
 - wierszy tabel obiektów, kolumn tabel relacyjnych,
 - perspektyw XML-owych
- Dostęp za pomocą klauzuli `passing as` funkcji `xmlquery` lub `xmltable`

Przykłady:

Tabela r-0: `CREATE TABLE instytucje_xml (kod VARCHAR2(12), opis XMLType)`

```
select xmlquery('$i/instytucja' passing opis as "i"
              returning content)
from instytucje_xml;
```

```
<instytucja kod="WEiTI">Wydział Elektroniki i Technik Informatycznych</instytucja>
<instytucja kod="IAiIS">Instytut Automatyki i Informatyki Stosowanej</instytucja>
```

```
select ro.kod, xml.column_value.getclobval()
from   instytucje_xml ro,
       xmltable('$i/instytucja' passing opis as "i") xml;
```

WEiTI	<instytucja kod="WEiTI">Wydział Elektroniki i Technik Informatycznych</instytucja>
IAiIS	<instytucja kod="IAiIS">Instytut Automatyki i Informatyki Stosowanej</instytucja>



Źródła danych dla XQuery w Oracle 10g r. 2, c.d.

Dokumenty zewnętrzne

- Dokumenty zewnętrzne XML dostępne przez protokół http mogą być wykorzystane w zapytaniach XQuery
- Wykorzystuje się funkcję `httpuritype()`
- Umożliwia to integrację danych zewnętrznych w zapytaniach w DBMS
- Wydajność jest ograniczona
 - konieczność parsowania XML

```
select xmlquery('$i/.'
              passing
              xmlparse(
                document httpuritype('http://.../instytucje.xml').getclob()
              ) as "i"
              returning content)
from dual;
```



Oracle 10g r. 2 – funkcje rozszerzające XPath

Dostęp do danych

- Funkcja ora:view()

Wyrażenia regularne

- Funkcje ora:matches() i ora:replace()
- Wyrażenia regularne w Oracle10g nie są zgodne ze standardem XPath

Wyszukiwanie pełnotekstowe

- Funkcja ora:contains pozwala wyszukiwać z użyciem mechanizmów pełnotekstowych
- Używana zwykle w klauzuli where wyrażenia FLWOR

Obliczenia

- Funkcja ora:sqrt wylicza pierwiastek kwadratowy



Oracle10g r. 2 – wydajność zapytań XQuery

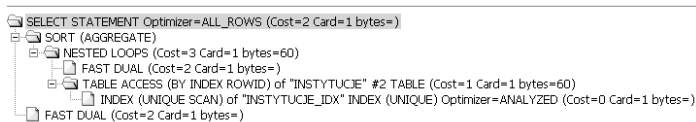
Optymalizacja zapytań XQuery

- Zapytanie XQuery jest optymalizowane
 - łącznie z otaczającym SQL
 - w podobny sposób
- Optymalizacja XQuery
 - zapytanie XQuery jest zamieniane na bloki podzapytań SQL ze specjalnymi operatorami działającymi na sekwencjach
 - te bloki są optymalizowane w kontekście całego zapytania SQL
 - eliminuje się zbędne pośrednie materializacje wyników
 - nieliczne konstrukcje XQuery, które nie dają się zamienić na podzapytania SQL, są interpretowane przez wbudowany interpreter
- Optymalizacja XPath
 - mechanizm *XPath rewrite*
 - » zamienia zapytania ścieżkowe XPath na zapytania relacyjne, jeśli można
 - » umożliwia wykorzystanie indeksów

Przykład:

```
Indeks: CREATE UNIQUE INDEX
instytucje_idx
ON instytucje(kod);
```

```
select xmlquery('
for $i in ora:view("instytucje")/ROW
where $i/KOD="WEITI"
return $i
' returning content).extract('.').getclobval() AS xml
from dual
```



Oracle XML Query Service



Oracle XQS

XML Query Service (dawniej XML Data Synthesis)

- Narzędzie do integracji danych
 - działające w warstwie *middle-tier*
 - napisane w Javie (aplikacja serwera OC4J)
 - używające XML jako wspólnego formatu danych
 - używające języka XQuery do definiowania sposobu integracji
- Dostępność
 - składnik AS10g r. 3



Źródła danych dla XQS

Możliwe źródła danych

- Dokumenty XML
- Dokumenty nie-XML
 - transformowane na bieżąco
 - opis transformacji w D3L
- Dane z baz relacyjnych
 - przez JDBC
- Dane uzyskiwane z *Web Services* przez SOAP
- XML zwracany przez klasę Java lub EJB
 - dostępne przez WSIF Provider (*Web Services Invocation Framework*)

Buforowanie danych

- XQS może buforować dane źródłowe
 - polepszenie wydajności
 - parametry buforowania ustawiane w pliku konfiguracyjnym

Dostęp do źródeł w XQuery

- Przez funkcje zastępujące standardową `doc()`
 - definiowane deklaratywnie (pliki konfiguracyjne XQS)

D3L (*Data Definition Description Lang.*)

- Część technologii *Oracle Application Server Integration InterConnect*
- Język opisu formatów nie-XML-owych
 - dialekt XML
 - określa
 - » format źródła
 - » sposób jego udostępnienia w XML
- Motor D3L wbudowano w XQS
 - definicja źródła danych może odwoływać się do URL opisu w D3L
 - XQS wykonuje translację *on-line* i udostępnia do integracji dane w XML



Wyniki XQS

Dostęp do wyników integracji

- Biblioteka znaczników JSP
- API klienta dla języka Java
- Komponenty sesyjne EJB
 - bezstanowy
 - zachowujący stan
 - » umożliwia pobieranie wyniku w wielu krokach
- Widoki XQuery udostępnione jako *Web Services*

Widoki XQuery

- Zapytania XQuery
 - zapamiętane w plikach `.xq`
 - » mogą być sparametryzowane za pomocą zmiennych zewnętrznych (`declare variable ... external`)
 - zdefiniowane w pliku konfiguracyjnym
- Użycie:
 - jako źródła danych w innych zapytaniach
 - jako usługi sieciowe



Przykład – zadanie

Dane wejściowe

- Spis repet studentów w poszczególnych semestrach
 - perspektywa w bazie danych Oracle
- Cennik opłat za powtarzanie przedmiotów, ceny w euro
 - plik Excela w formacie CSV
- Lista terminów wnoszenia opłat
 - plik XML
- Kursy walut
 - usługa sieciowa

Opis zadania

- Zestawić repety za ostatni semestr z podaniem terminu opłaty i kwoty opłaty w złotych

Rozwiązanie

- Cztery źródła danych
 - SQL
 - dokument tekstowy z konwersją przez D3L
 - dokument XML
 - *Web Service*
- Integracja w XQS przez zapytanie XQuery



Źródło danych: dokument XML

Konfiguracja w pliku xqs-config.xml

```
<xqs-sources>
  <document-source isCached="false">
    <function-name namespace="http://xmlns.oracle.com/ias/xqs">
      terminy_dokumentXML
    </function-name>
    <documentURL>http://.../terminy.xml</documentURL>
  </document-source>
  ...
</xqs-sources>
```



```
<terminy_platnosci>
  <termin typ_zaliczenia="E">2005-10-20</termin>
  <termin typ_zaliczenia="Z">2005-10-20</termin>
  <termin typ_zaliczenia="P">2005-09-25</termin>
  <termin typ_zaliczenia="L">2005-09-25</termin>
</terminy_platnosci>
```



Źródło danych: dokument CSV

Konfiguracja w pliku xqs-config.xml

```
<document-source isCached="false">
  <function-name namespace="http://xmlns.oracle.com/ias/xqs">
    ceny_dokumentCSV
  </function-name>
  <documentURL>http://.../ceny.csv</documentURL >
  <XMLTranslate method="D3L">
    <schema-file>http://.../ceny_CSV2XML.xml</schema-file>
  </XMLTranslate>
</document-source>
```

Konfiguracja D3L

```
<?xml version="1.0"?>
<!DOCTYPE message SYSTEM "d3l.dtd">
<message name="excelData" type="ceny_repet">
  <impparray id="wiersz">
    <struct>
      <field name="typ_zaliczenia"><termstring endchar=","/></field>
      <field name="nazwa_typu"><termstring endchar=","/></field>
      <field name="cena_w_euro"><number><termstring endchar="\n"/></number></field>
    </struct>
  </impparray>
  <struct id="ceny_repet">
    <field name="repet">
      <typeref type="wiersz"/>
    </field>
  </struct>
</message>
```

```
<ceny_repet>
  <repet>
    <typ_zaliczenia>E</typ_zaliczenia>
    <nazwa_typu>Egzamin</nazwa_typu>
    <cena_w_euro>150</cena_w_euro>
  </repet>
  ...
</ceny_repet>
```



Źródło danych: zapytanie SQL

Konfiguracja w pliku xqs-config.xml

```
<wsdl-source isCached="false">
  <function-name prefix="xqs">repety_zrodloSQL</function-name>
  <wsdlURL>http://.../repety_SQL.wsdl</wsdlURL>
  <operation>pobierzDaneSQL</operation>
  <port>portsSQL</port>
  <input-parameters>
    <part position="1" name="semestr">
      <schema-type prefix="xs">string</schema-type>
    </part>
  </input-parameters>
</wsdl-source>
```

Konfiguracja „usługi” w pliku WSDL (fragment)

```
<binding name="wiazaniesQL" type="oe:zrodloSQL_typ">
  <sql:binding/>
  <operation name="pobierzDaneSQL">
    <sql:operation XMLtransform="XSU-client">
      select * from repety where semestr = :1
    </sql:operation>
    <input name="sqlInput">
      <sql:input>semestr</sql:input>
    </input>
    <output name="sqlResult"/>
    <fault name="sqlFault"/>
  </operation>
</binding>
<service name="repety_SQL">
  <port name="portsSQL" binding="oe:wiazaniesQL">
    <sql:address data-source-location="jdbc/OracleCoreDS" user="..." password="..." />
  </port>
</service>
```

```
<ROW num="1">
  <ID_STUDENTA>120032</ID_STUDENTA>
  <NAZWISKO>Abacki</NAZWISKO>
  <IMIONA>Aleksander</IMIONA>
  <KOD_PRZEDMIOTU>BD1</KOD_PRZEDMIOTU>
  <SEMESTR>05L</SEMESTR>
  <TYP_ZALICZENIA>L</TYP_ZALICZENIA>
</ROW>
...
```



Źródło danych: usługa sieciowa (Web Service)

Konfiguracja w pliku xqs-config.xml

```
<wsdl-source isCached="true">
  <function-name prefix="xqs">kursywalut_webService</function-name>
  <wsdlURL>http://.../CurrencyExchangeService.wsdl</wsdlURL>
  <operation>getRate</operation>
  <service>CurrencyExchangeService</service>
  <port>CurrencyExchangePort</port>
  <input-parameters>
    <part position="1" name="country1">
      <schema-type prefix="xs">string</schema-type>
    </part>
    <part position="2" name="country2"/>
      <schema-type prefix="xs">string</schema-type>
    </part>
  </input-parameters>
</wsdl-source>
```

- Konfiguracja odwołuje się do „obcego” dokumentu WSDL, opisującego usługę



Zapytanie XQuery

```
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs";
declare function xqs:repety_zrodloSQL ($id_semestru as xs:string) external;
declare function xqs:ceny_dokumentCSV() external;
declare function xqs:kursywalut_webService($j1 as xs:string, $j2 as xs:string) external;
declare function xqs:terminy_dokumentXML() external;
```

```
let $t := xqs:terminy_dokumentXML(),
    $c := xqs:ceny_dokumentCSV(),
    $k := xqs:kursywalut_webService("euro", "poland")
return
<oplaty>
{
  for $r in xqs:repety_zrodloSQL('05L')//ROW
  let $ce := $c//repeta[typ_zaliczenia=$r//TYP_ZALICZENIA]/cena_w_euro/text(),
      $cz := $ce * $k,
      $te := $t//termin[@typ_zaliczenia=$r//TYP_ZALICZENIA]/text()
  order by $r//KOD_PRZEDMIOTU, $r//TYP_ZALICZENIA, $r//NAZWISKO, $r//IMIONA
  return
  <repeta>
  <opлата>
    {$r//KOD_PRZEDMIOTU}
    {$r//TYP_ZALICZENIA}
    <kwota waluta="EUR">{$ce}</kwota>
    <kwota waluta="PLN">{$cz}</kwota>
    <termin_wplaty>{$te}</termin_wplaty>
  </opлата>
  <student>
    {$r//NAZWISKO}
    {$r//IMIONA}
  </student>
</repeta>
}
</oplaty>
```

```
<oplaty>
  <repeta>
    <opлата>
      <KOD_PRZEDMIOTU>BD1</KOD_PRZEDMIOTU>
      <TYP_ZALICZENIA>E</TYP_ZALICZENIA>
      <kwota waluta="EUR">150</kwota>
      <kwota waluta="PLN">588,450014591217</kwota>
      <termin_wplaty>2005-10-20</termin_wplaty>
    </opлата>
    <student>
      <NAZWISKO>Cabacki</NAZWISKO>
      <IMIONA>Cezary</IMIONA>
    </student>
  </repeta>
  ...
</oplaty>
```



Podsumowanie



XQuery a SQL

Przewagi XQuery

- Bogatsza postać przetwarzanej informacji
- Dostosowanie do przetwarzania informacji semistrukturalnej
- Bardziej konsekwentna składnia języka
- Bardzo dobre dopasowanie do potrzeb środowiska WWW, w tym *Web Services*

Przewagi SQL

- Prostota przetwarzanej informacji
- Dostosowanie do licznych narzędzi i interfejsów
- Wielka rzesza specjalistów
- Efektywne metody optymalizacji i szeroko znane sposoby strojenia
- DML i DDL

XQuery konkurencją dla SQL?

- Raczej uzupełnieniem w zastosowaniach związanych z XML, WWW, integracją danych itp.
- Alternatywą dla SQL/XML
- A konkurencją – jeszcze nie dziś...



Podsumowanie

Standard XQuery

- Wspierany przez wielkich producentów oprogramowania
- Bliski ukończenia

Braki XQuery

- Brak DML – nie ma instrukcji DML ani zarządzania transakcjami
- Brak mechanizmu refleksji (np. funkcji typu eval ())

Praktyczne znaczenie języka XQuery

- Szybko rośnie
- Czynniki wzrostu znaczenia XQuery
 - pojawienie się implementacji
 - » zintegrowanych z DBMS – spełnione
 - » zintegrowanych z wyszukiwarkami internetowymi – jeszcze brak
- Spodziewana rola XQuery
 - wygodny środek do jednolitego wyszukiwania informacji w Internecie, bazach danych i innych zasobach informacji
 - narzędzie do scalania i przekształcania informacji
 - język zapytań uzupełniający możliwości SQL



XQuery – konkurencja dla SQL?

