

X Konferencja PLOUG  
Kościelisko  
Październik 2004

# Język XML Query

Tomasz Traczyk

*Politechnika Warszawska  
e-mail: ttraczyk@ia.pw.edu.pl*

## **Abstrakt**

Kolejny referat z cyklu prezentującego technologię XML przedstawia XML Query (XQuery) – język zapytań do struktur XML-owych. Wyjaśniono, dlaczego potrzebny jest język zapytań dla XML i czemu nie wystarcza język XPath. Omówiono założenia leżące u podstaw języka i przedstawiono stan prac standaryzacyjnych. Pokazano przykłady zapytań i opisano główne sposoby użycia języka. Zaprezentowano przykładowe implementacje języka, w tym produkt firmy Oracle.



## Po co XML Query?

Stale rosnąca popularność języka XML sprawia, że pojawia się coraz więcej zasobów informacji dostępnej w tym języku. Szeroko stosowane są liczne standardowe narzędzia umożliwiające zapis i odczyt informacji w XML oraz jej przetwarzanie: parsery, procesory XSL, różnorodne narzędzia ułatwiające użycie XML w systemach z bazami danych. Potrzebne są jednak także środki, które umożliwiłyby wygodne wyszukiwanie i przetwarzanie informacji w zasobach dokumentów XML-owych.

Dość oczywistym rozwiązaniem jest stworzenie odpowiedniego języka zapytań. Języki zapytań są bowiem od dawna z powodzeniem używane do wyszukiwania i przetwarzania informacji i zostały powszechnie zaakceptowane. Wydawać by się mogło, że odpowiednie narzędzie już istnieje – że jest nim język XPath. Jednak XPath nie spełnia wielu wymagań zwykle stawianych językom zapytań.

Czego oczekujemy od języka zapytań? Jak się wydaje, kluczowe są następujące cechy.

1. Język powinien umożliwiać zapisanie złożonych warunków wyszukiwania.
2. Język powinien być deklaratywny, tzn. nie wymagać określenia algorytmu realizacji zapytania.
3. Język powinien zwracać wyniki w tej samej formie, w jakiej istnieją dane źródłowe (w tym wypadku w XML).
4. Język powinien umożliwiać wykonywanie przekształceń wyszukanej informacji, w tym różnorodnych agregacji (np. sumowań, uśrednień itp.).

Powszechnie używany język zapytań SQL spełnia oczywiście te wymagania, w szczególności wynikiem zapytania do tabel relacyjnych jest w nim zawsze tabela.

Język XPath spełnia pierwsze dwa z wymagań: formułuje zapytania deklaratywnie, pozwalając zapisywać złożone warunki wyszukiwania w strukturach drzewiastych w sposób pomysłowy, skuteczny i intuicyjny. Niestety, XPath nie spełnia pozostałych warunków. Wynikiem wyszukiwania jest pewna struktura danych (zbiór węzłów w XPath 1.0, sekwencja w XPath 2.0), która nie jest dokumentem XML. Węzły zwracane są w takiej postaci, w jakiej istnieją w dokumentach źródłowych; nie jest możliwe ich przekształcanie ani tym bardziej agregowanie.

Sam język XPath nie może być uznany za pełnowartościowy język zapytań, jednak na jego podstawie można taki język zbudować, obudowując wyrażenia XPath w odpowiednią składnię, która pozwala dodać brakujące właściwości.

## Czym jest XML Query?

Język XML Query, zwany także skrótowo XQuery, jest tak właśnie zbudowanym językiem zapytań. Skonstruowano go na bazie języka XPath 2.0. Ścieżki XPath obudowano w sposób dość typowy dla języków zapytań, dodając zmienne pomocnicze oraz klauzule służące do przeszukiwania wyników wyrażen ścieżkowych, podstawiania wyników obliczeń i przekształceń, filtrowania wyników oraz sortowania.

### Przykład

Typowym zasobem informacji wykorzystywanym w instytucjach naukowych są dane bibliograficzne opisujące publikacje stworzone przez daną instytucję i jej pracowników. Informacje takie są wykorzystywane na różne sposoby, m. in. do sprawozdawczości, powinny także być udostępniane przez WWW. O ile budowa danych na temat publikacji jest stosunkowo dobrze określona, o tyle trudno przewidzieć wszelkie potencjalne zastosowania tej informacji, a w szczególności możliwe warunki wyszukiwań oraz potrzebne formaty wyników.

Bardzo dogodnym sposobem zapisu i przekazywania informacji bibliograficznej jest właśnie XML. Informacja ta bowiem jest zwykle stosunkowo niewielkich rozmiarów (nawet w dużych in-

stytucjach liczba publikacji zamyka się w pojedynczych tysiącach rocznie), za to jej struktura jest dość skomplikowana. Nawet jeśli informacja źródłowa jest przechowywana w relacyjnej bazie danych, pozyskanie jej w postaci XML nie powinno sprawiać trudności.

Na Wydziale Elektroniki i Technik Informatycznych Politechniki Warszawskiej działa od niedawna system gromadzenia informacji o publikacjach, który ma możliwość prezentacji informacji bibliograficznej w postaci XML. Dane źródłowe przechowywane są w relacyjnej bazie danych (Oracle 9i). Zbudowano odpowiednie perspektywy XML-owe (wykorzystując rozszerzenia XSQL wbudowane w bazę), które pozwalają uzyskać dane bibliograficzne w postaci dokumentu XML.

Poniżej przedstawiono przykładowy (uproszczony) fragment XML-owego zapisu tych danych (w pliku publik.xml):

```
<?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>
```

W danych tych wyszukamy referaty wygłoszone w roku 1998 za pomocą zapytania XQuery:

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

Jako rezultat zapytania otrzymamy:

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

```
</tytul>
</referat>
</odpowiedz>
```

Jak widać, w zapytaniu użyto pomocniczej zmiennej  $\$b$ . Zmienna ta przyjmuje wartości kolejnych węzłów zwracanych przez wyrażenie ścieżkowe XPath. Wynik zapytania jest dodatkowo ograniczony klauzulą **where**, zaś postać wyniku jest kształtowana przez zawartość klauzuli **return**. Wynikowe struktury mogą być wstawiane w „szablon” dokumentu XML; służy do tego ujmowanie wyrażen XQuery w nawiasy klamrowe.

## Standaryzacja XML Query

Prace nad językami zapytań dla XML rozpoczęły się wkrótce po pojawieniu się samego XML. Powstało wiele propozycji (XQL, XML-QL, Lorel, Quilt); równoległe opracowano język XPath, początkowo związany głównie z XSL.

Opracowaniem standardu języka zapytań dla XML zajęła się *XML Query Working Group*, działająca w ramach *World Wide Web Consortium*; ta sama grupa pracuje także nad specyfikacją języka XPath 2.0. Przy konstrukcji języka XQuery wykorzystano intensywnie XPath 2.0 oraz elementy wcześniejszych propozycji. Wpływ na postać języka miały też oczywiście znane języki zapytań SQL i OQL.

Prace nad językiem XQuery nie są jeszcze zakończone. Opracowano aż 12 dokumentów (patrz <http://www.w3.org/XML/Query>), opisujących w sposób sformalizowany m.in. założenia języka, przypadki użycia języka, model danych oraz składnię i semantykę, a także rozszerzenia do przetwarzania pełnotekstowego. Propozycje specyfikacji języków XQuery 1.0 oraz XPath 2.0 zawarto w dokumentach [W3q04, W3p04]; mają one wciąż jednak status *Working Draft*, co oznacza, że prace nad standardami trwają i możliwe są jeszcze daleko idące zmiany.

## Wymagania dla XQuery

By tworzony język zapytań dla XML mógł spełnić oczekiwania przyszłych użytkowników, w sposób formalny spisano stawiane mu wymagania [W3r03]. Oto najważniejsze z nich:

- przynajmniej jedna ze składni języka<sup>1</sup> powinna być czytelna dla człowieka;
- język musi być deklaratywny (tzn. w zapytaniu nie trzeba kodować algorytmu jego wykonania);
- język musi być niezależny od protokołów i środowisk użycia;
- język powinien uwzględniać model danych XML, przestrzenie nazw i schematy;
- zapytania muszą działać nawet jeśli schematy przeszukiwanych dokumentów nie są dostępne;
- język musi być typowany, typowanie musi obejmować typy proste i złożone;
- niezbędne jest istnienie kwantyfikatorów (ogólnego i szczegółowego);
- wymagana jest możliwość operowania na hierarchiach i sekwencjach struktur dokumentów;
- musi istnieć możliwość łączenia informacji z wielu źródeł;
- niezbędna jest możliwość wykonywania agregacji;
- język musi umożliwiać transformacje dokumentów XML i możliwość tworzenia wynikowych struktur XML;
- potrzebna jest możliwość nawigowania po odwołaniach (referencjach) do identyfikatorów.

Opracowano także zbiór przykładów [W3u03], które pokazują typowe przypadki użycia XQuery; są one pomocne np. w określaniu stopnia zgodności poszczególnych implementacji ze standardem.

---

<sup>1</sup> Dopuszczono istnienie wielu alternatywnych składni.

## Wprowadzenie do składni języka XQuery

Język XQuery jest konsekwentnie zaprojektowanym językiem funkcyjnym. Konstrukcje języka składają się zatem z wyrażeń przyjmujących pewne wartości. Wszystkie wyrażenia XQuery zwracają tzw. sekwencje. Podstawowym wyrażeniem „dostającym się” do danych jest ścieżka XPath – w wersji 2.0 tego języka wynikiem ścieżki jest właśnie sekwencja (por. [Tra03]). W XQuery dostępna jest niemal pełna składnia ścieżek XPath 2.0, z nielicznymi wyjątkami.

### Podstawowe elementy języka

#### Sekwencje

Sekwencja jest to uporządkowany ciąg węzłów lub wartości atomowych (prostych). Pojedyncza wartość atomowa jest utożsamiana z jednoelementową sekwencją; oznacza to, że nie są tu potrzebne jawne konwersje. Sekwencje nie mogą być zagnieżdżane. Porządek w sekwencji zwracanej przez wyrażenie jest – inaczej niż w SQL – ściśle określony; jest to tzw. porządek dokumentu, tzn. elementy występują w takiej kolejności, w jakiej były w przetwarzanym dokumencie. Kolejność tę można zmienić żądając sortowania za pomocą klauzuli **order by**.

Sekwencje mogą być wynikiem wyrażeń, można je też zapisać jawnie ujmując elementy, oddzielone przecinkami, w nawiasy zwykłe. Pustą sekwencję zapisuje się w postaci pary nawiasów: **()**.

#### Wyrażenia

Wszystkie konstrukcje języka XQuery składają się z wyrażeń. Wyrażenia zwracają sekwencje i mogą być zagnieżdżane. Wyrażenia mogą zawierać stałe, jawnie zapisane sekwencje, odwołania do zmiennych, wywołania funkcji wbudowanych i funkcji użytkownika, ścieżki XPath, tzw. wyrażenia FLWOR (omawiane dalej) i warunkowe oraz operatory.

#### Zmienne

Nazwy zmiennych poprzedzane są znakiem dolara. Zmienne mogą być deklarowane, ale nie jest to obowiązkowe.

#### Operatory i porównania

XQuery dostarcza zestawu typowych operatorów arytmetycznych i logicznych. Nietypowe są natomiast operatory porównania.

Operatory porównujące proste wartości zapisuje się **eq**, **ne**, **lt**, **le**, **gt**, **ge**. Jeśli operandem jest jednoelementowa sekwencja, to następuje niejawnie tzw. atomizacja, tj. konwersja na wartość atomową.

Ogólne operatory porównań, zapisywane **=**, **!=**, **<**, **<=**, **>**, **>=**, porównują całe sekwencje stanowiąc kombinację zwykłego porównania wartości z kwantyfikatorem szczegółowym (*exists*). Oznacza to, że wynik ogólnego porównania sekwencji jest prawdą, jeśli tylko istnieją takie elementy porównywanych sekwencji, które spełniają żądane porównanie.

Operatory porównywania węzłów: **is**, **<<**, **>>**, badają identyczność (w sensie tożsamości) węzłów oraz ich porządek w dokumencie.

Do porównywania całych gałęzi (sekwencji wraz z węzłami podrzędnymi) służy funkcja wbudowana **deep-equal ()**.

#### Funkcje wbudowane

W XQuery można korzystać z wielkiej liczby funkcji wbudowanych. Funkcje te obejmują m.in. operacje na sekwencjach, węzłach i wartościach, operacje arytmetyczne, agregacje, operacje na tekstach, wyrażenia regularne, operacje na datach i czasie oraz obsługę błędów i śledzenie.

### Wartości logiczne

Wynikiem porównań jest jedna z dwóch wartości logicznych: **true** lub **false**. Operacje logiczne wykonywać można za pomocą operatorów **or** i **and** oraz funkcji **not()**.

Ciekawą koncepcją jest tzw. efektywna wartość logiczna wyrażenia. Otóż jeśli wyrażenie o typie różnym od logicznego występuje w kontekście, w którym oczekiwana jest wartość logiczna (a nie dokonano jawnej konwersji na wartość logiczną za pomocą wyrażenia **cast as**), to jest ono uznawane za fałszywe jeśli jego wynik jest sekwencją pustą, pustym napisem, liczbą o wartości zero lub wartością NaN; w przeciwnym wypadku jest uznawane za prawdziwe.

### Inne elementy języka

W języku XQuery wielkość liter ma znaczenie; wszystkie słowa kluczowe pisane są małymi literami. W tekście można umieszczać komentarze o dość zabawnej składni (*: komentarz :*) i komentarze te można zagnieźdzać.

### Budowa zapytania XQuery

Najprostszym zapytaniem w XQuery jest po prostu wyrażenie ścieżkowe XPath. Zastosowanie wobec dokumentu z wcześniejszego przykładu zapytania:

```
<tytul>
  { doc("publik.xml")//tytul }
</tytul>
```

daje w wyniku dokument o następującej postaci:

```
<tytul>
  <tytul>Język XML w aplikacjach z bazami danych</tytul>
  <tytul>Język XML w aplikacjach z bazami danych - po roku</tytul>
  ...
</tytul>
```

gdyż ścieżka XPath zwraca sekwencję elementów `<tytul>`.

W zapytaniu tym odwołano się do zewnętrznego dokumentu jako do źródła danych. Tak proste zapytanie daje stosunkowo niewielkie możliwości, dlatego do konstruowania praktycznie przydatnych zapytań najczęściej używa się wyrażań FLWOR.

### Źródła danych

Dostęp do źródeł danych dla zapytań zapewniają dwie funkcje wbudowane:

- **doc(uri)**<sup>2</sup> zwraca (jako węzeł) dokument znajdujący się pod wskazanym adresem URI;
- **collection(uri)** zwraca sekwencję węzłów będącą wynikiem odwołania się do wskazanego adresu URI – może to być sposób odwoływania się do baz danych.

### Wyrażenia FLWOR

Najbardziej charakterystyczną konstrukcją języka XQuery są wyrażenia FLWOR<sup>3</sup>. Wcześniej podano już przykład wykorzystujący tę konstrukcję. Oto kolejny przykład:

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

Zapytanie to zwraca sekwencję elementów `<tytul>` zawierających tytuły tych publikacji, które miały tylko jednego autora, posortowane według roku publikacji.

<sup>2</sup> W starszych wersjach propozycji standardu funkcja ta nazywa się **document()**.

<sup>3</sup> Czytane jak angielskie *flower*.

Wynikiem całego wyrażenia FLWOR jest oczywiście sekwencja. Poszczególne klauzule wyrażenia FLWOR mają następujące znaczenie.

- Klauzula **for** organizuje przypisuje kolejno do zmiennej kolejne pozycje sekwencji zwróconej przez wyrażenie umieszczone po słowie **in**. Kolejne postawienia stanowią podstawę wyliczenia (na podstawie klauzuli **return**) kolejnych elementów sekwencji wynikowej.
- Klauzula **let** dokonuje jednokrotnego podstawienia do zmiennej. Jeśli wynik wyrażenia po lewej stronie podstawienia jest sekwencją, to zmienna przybiera (jednokrotnie!) wartość całej tej sekwencji.
- Klauzula **where** określa warunki „filtrowania”.
- Klauzula **order by** pozwala posortować wynikową sekwencję.
- Klauzula **return** definiuje postać elementów wynikowej sekwencji.

Wszystkie klauzule są opcjonalne, ale musi wystąpić przynajmniej jedna z klauzul **for** lub **let**. Wyrażenia FLWOR można zagnieżdżać, jak w zapytaniu:

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

### Konstruktory

Wynikowe struktury XML można – podobnie jak w XSLT – konstruować na dwa sposoby:

- umieszczając wprost w treści zapytania elementy i atrybuty („szablon”) wynikowego dokumentu XML, a w miejscach w których mają być podstawione wyniki zapytania wstawiając odpowiednie wyrażenia XQuery ujęte w nawiasy klamrowe (patrz przykłady wyżej);
- używając specjalnych słów kluczowych: **element**, **attribute**, **text**, **pi** (*processing instruction*), **comment** – pozwala to wstawiać nie tylko wartości, ale i nazwy wyliczane przez wyrażenia.

### Wyrażenia warunkowe i kwantyfikatory

Często przydatne jest wyrażenie warunkowe **if warunek then wyrażenie1 else wyrażenie2**. Warto zwrócić uwagę na to, że ponieważ każde wyrażenie musi zawsze zwracać jakąś wartość, to część **else** wyrażenia warunkowego jest w XQuery obowiązkowa.

W konstruowaniu warunków **where** ważną rolę odgrywają kwantyfikatory: ogólny **every... in... satisfies** i szczegółowy **some... in... satisfies**.

### Unikalność

Jak wiadomo, w języku zapytań przydatna jest konstrukcja, która umożliwia wyeliminowanie powtórzeń z wyniku. W XQuery zapewnia to specjalna funkcja **distinct-values()**, używana np. w części **in** klauzuli **for**.

### Łączenie danych z różnych źródeł

Za pomocą wyrażeń FLWOR można w dość łatwy sposób wykonywać „złączenia” danych z różnych źródeł. Załóżmy, że jest dostępny dodatkowy dokument `instyt.xml`:

```
<instytucje>
  <instytucja kod="WEiTI">Wydział Elektroniki</instytucja>
  ...
</instytucje>
```

Poniższe zapytanie:

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

tworzy dla każdej pozycji spisu publikacji listę wszystkich instytucji, których pracownicy byli autorami danej publikacji.

### Grupowanie i agregacje

W języku nie ma specjalnych konstrukcji przeznaczonych do grupowania (podobnych do *group by* z SQL). Efekt grupowania uzyskać można dzięki zagnieżdżaniu wyrażeń FLWOR.

Grupowanie przydaje się szczególnie w przypadku obliczania agregacji. XQuery dostarcza kilku funkcji agregujących: `sum()`, `avg()`, `count()`, `max()` i `min()`. Użycie agregacji pokazuje przykład:

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

Zapytanie zwraca liczbę publikacji napisanych przez pracowników kolejnych instytucji.

### Funkcje użytkownika i modularyzacja

Język umożliwia tworzenie funkcji użytkownika i wykorzystywanie ich w zapytaniach. Na przykład poniższe zapytanie używa wcześniej zadeklarowanej funkcji użytkownika.

```
define function liczba-autorow($param)
  { count($param//autor) }

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

Nie ma możliwości przeciążania funkcji użytkownika ani definiowania funkcji o zmiennej liczbie parametrów.

W nowszych wersjach standardu zdefiniowano także środki do modularyzacji kodu. Moduły zapytań mogą, a moduły biblioteczne muszą zawierać prolog, w którym podaje się różne deklaracje. Moduły biblioteczne zawierają deklarację `module`, która definiuje m.in. nazwę modułu. Deklaracja `import module` pozwala włączać moduły biblioteczne do modułów zapytań.

## Typowanie i walidacja

XQuery jest językiem silnie typowanym. System typów wzięto z XML Schema, oprócz tego XQuery korzysta z dodatkowych „generycznych” typów danych XPath 2.0, np. **anySimpleType** czy **untypedAtomic**.

Typowanie wartości elementów i atrybutów jest niezbędne w wielu okolicznościach, np. przy porównywaniu czy sortowaniu wartości. Umożliwia też statyczne wykrywanie błędów w zapytaniach.

Z pełnym silnym typowaniem mamy do czynienia wtedy, gdy przetwarzane dokumenty są wyposażone w schematy XML Schema. Takie schematy można importować do modułu XQuery za pomocą umieszczanej w prologu deklaracji **import schema**. Przetwarzanie dokumentów odbywa się wtedy z pełnym typowaniem uwzględniającym typy danych zdefiniowane w schemacie. Elementy tworzone w wyniku zapytania także są sprawdzane z użyciem zaimportowanego schematu. Walidacją tą można sterować (wymuszać, wyłączać itp.) za pomocą instrukcji **validate**.

Jeśli nie mamy dostępu do schematu przetwarzanego dokumentu, wykonywanie zapytań nadal jest możliwe, lecz odbywa się z w oparciu o znakową reprezentację wartości elementów i atrybutów oraz jawne lub domyślne konwersje.

W kodzie XQuery można wykonywać pewne jawne operacje na typach danych, m.in.:

- deklarowanie typów zmiennych oraz parametrów i wyników funkcji użytkownika;
- jawne konwersje typów za pomocą operatora **cast as**;
- badanie czy element sekwencji jest określonego typu za pomocą operatora **instance of**;
- zwrotnica **typeswitch... case... default**, pozwalająca różnie przetwarzać element w zależności od jego typu.

## XQuery a przestrzenie nazw

Przetwarzanie w XQuery odbywa się z wykorzystaniem przestrzeni nazw.

XQuery wprowadza kilka predefiniowanych przestrzeni nazw i predefiniowanych prefiksów, np. dla typów danych XML Schema i XPath oraz funkcji wbudowanych XPath. Inne przestrzenie nazw można określać za pomocą deklaracji **declare namespace** umieszczanej w prologu modułu; podobnie określić można także przestrzeń domyślną (nie wymagającą prefiksów).

## Użycie XML Query

### Konteksty użycia XQuery

XQuery, jak każdego języka zapytań, użyć można w wielu kontekstach, interaktywnie lub przez zanurzenie w językach programowania. Ponieważ większość znanych implementacji języka napisano w Javie, możliwe jest wykonywanie zapytań we wszystkich warstwach systemów informacyjnych. Przeszukiwane dokumenty mogą być dostępne w postaci plików, mogą też być użyte dowolne zasoby XML-owe dostępne przez protokół HTTP i adres URI. Oznacza to, że można tworzyć zapytania do dynamicznych źródeł informacji w XML, takich jak strony JSP czy XSQL; w ten sposób można też z poziomu zapytania XQuery dostawać się do zawartości baz danych.

### Interaktywne zapytania XQuery

Najprostszym sposobem wykonania zapytania jest użycie odpowiedniego programu uruchamianego z linii poleceń. Dostępne implementacje XQuery z reguły dają taką możliwość. Takie wykorzystanie może służyć do nauki języka, testowania zapytań czy składania zapytań *ad hoc*.

### XML Query a Java

Zapytania XQuery mogą być wykonywane przez programy; w takim przypadku samo zapytanie należy zanurzyć w innym języku programowania. Jak to na ogół bywa z narzędziami XML-owymi, pierwszoplanowym językiem jest Java. Dlatego dla Jawy opracowuje się specjalny interfejs pro-

gramistyczny o nazwie XQJ (patrz <http://jcp.org/en/jsr/detail?id=225>), spójny z istniejącymi standardami JDBC i JAXP.

### Zapis zapytań w XML – XQueryX

O ile przedstawiony wyżej język XQuery jest czytelny i wygodny dla człowieka, o tyle same teksty zapytań niespecjalnie dobrze nadają się do przetwarzania maszynowego. W szczególności moduły XQuery nie są zapisane w XML, nie mogą więc być przetwarzane narzędziami XML-owymi. Stworzono więc alternatywną składnię języka, nazwaną XQueryX, będącą dialektem XML.

Na przykład takie proste zapytanie:

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

w XQueryX zapisać trzeba by tak:

```
<xq:query xmlns:xq="http://www.w3.org/2001/06/xqueryx">
  <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>
```

Jak widać, jest to zapis rozwlekły i zupełnie nieczytelny, jednak łatwy do przetwarzania narzędziami XML-owymi (łatwy też to translacji na zwykłą postać XQuery – można to zrobić np. za pomocą skryptu w XSLT).

### Implementacje XML Query

Mimo trwających wciąż prac standaryzacyjnych pojawiło się już sporo prób implementacji XQuery. Najbardziej znane są zapewne:

- implementacja wbudowana w popularny procesor XSLT Saxon (<http://sourceforge.net/projects/saxon/>);
- dostępna także na zasadach *open-source* implementacja Qizx/open (<http://www.xfra.net/qizxopen/>);
- rozpowszechniana na licencji GNU implementacja Qexo (<http://www.gnu.org/software/qexo/>).

Istnieje także specjalny pakiet zestaw o nazwie *BumbleBee*, stworzony specjalnie do testowania implementacji XQuery (<http://www.xquery.com/bumblebee/>). Zawiera on bardzo rozbudowany zestaw testowych zapytań (w tym przykłady przypadków użycia XQuery z [W3u03]), jest wyposażony w narzędzia automatyzujące testowanie, umożliwia także tworzenie i automatyczne wykonywanie dodatkowych testów.

## XML Query w Oracle

Także firma Oracle zbudowała prototyp narzędzia przetwarzającego zapytania w języku XQuery (<http://www.oracle.com/technology/tech/xml/xquery/index.html>). Jest on napisany w języku Java i wykorzystuje składniki znanego produktu *Oracle XML Developers Toolkit* (XDK).

Niestety, prototyp ten jest oparty na bardzo starej i już mocno nieaktualnej wersji propozycji standardu (z listopada 2002), w dodatku zrealizowanej tylko częściowo. Prototyp ma bardzo liczne ograniczenia, nawet w zakresie podstawowej składni XQuery, np. nie działa klauzula **order by**; zupełnie ułomna jest też dostarczana z produktem dokumentacja. Pewne problemy sprawiają niestety polskie litery: narzędzie przyjmuje zapytania jedynie w kodzie UTF-8 i tylko w nim tworzy wyniki; dokumenty źródłowe mogą jednak być kodowane inaczej. W stosunku do propozycji standardu dodano ciekawą funkcję o nazwie **sqlquery ()**, która pozwala użyć jako źródła danych zapytania SQL (wynik zapytania SQL przetwarzany jest na XML za pomocą biblioteki XSU z pakietu XDK).

Prototyp można wywołać z linii komendy. Polecenie

```
java oracle.xquery.XMLPlus plik
```

powoduje wykonanie zapytania zawartego w pliku. To samo polecenie bez parametru uruchamia interaktywną „konsolę” języka XQuery. Prototyp może też być wywoływany programowo z języka Java za pomocą opracowanego przez Oracle API.

## Podsumowanie

XQuery jest ciekawym językiem zapytań, dającym możliwości wyszukiwania informacji w źródłach XML-owych podobne jak w znanych językach zapytań dla baz danych.

Język jest jeszcze w stadium powstawania i daleko mu do stabilności. Na obecnym etapie rozwoju ma pewne braki, np. nie ma w nim konstrukcji służących do modyfikowania danych (odpowiednika SQL DDL). Prace standaryzacyjne posuwają się jednak naprzód, pojawiły się też pierwsze implementacje. Zainteresowanie językiem wyrażają zarówno nieformalne społeczności „świata” *open-source*, jak wielkie firmy, w tym BEA, IBM, Nokia, Oracle, Sybase i Sun.

Jak się więc wydaje, język i jego implementacje będą się rozwijać, gdyż potrzeba istnienia wygodnego i silnego języka zapytań do wyszukiwania i przetwarzania informacji XML-owej jest bezdyskusyjna, XQuery zaś stanowi tu najpoważniejszą propozycję.

## Bibliografia

- [Hun03] Hunter J.: X is for XQuery. Oracle Magazine, May/June 2003.
- [Hun04] Hunter J.: XQuery Tricks and Traps. Oracle Magazine, January/February 2004.
- [Ka+04] Katz H. et al.: XQuery from the Experts: A Guide to the W3C XML Query Language. Addison-Wesley, 2004. ISBN 0-321-18060-7.
- [Sin04] Singh D.: Essential XQuery – The XML Query Language. February 2004. <http://www.youkonxml.com/articles/xquery>
- [Tra03] Traczyk T.: XML – stan obecny i trendy rozwojowe. Materiały IX Konferencji użytkowników i developerów Oracle. Kościelisko, październik 2003.
- [W3r03] XML Query (XQuery) Requirements. W3C Working Draft, 27 June 2003. <http://www.w3.org/TR/2003/WD-xquery-requirements-20030627/>
- [W3u03] World Wide Web Consortium: XML Query Use Cases. W3C Working Draft, 12 November 2003. <http://www.w3.org/TR/xquery-use-cases/>
- [W3q04] World Wide Web Consortium: XQuery 1.0: An XML Query Language. W3C Working Draft, 23 July 2004. <http://www.w3.org/TR/xquery/>
- [W3p04] World Wide Web Consortium: XML Path Language (XPath) 2.0. W3C Working Draft, 23 July 2004. <http://www.w3.org/TR/xpath20/>