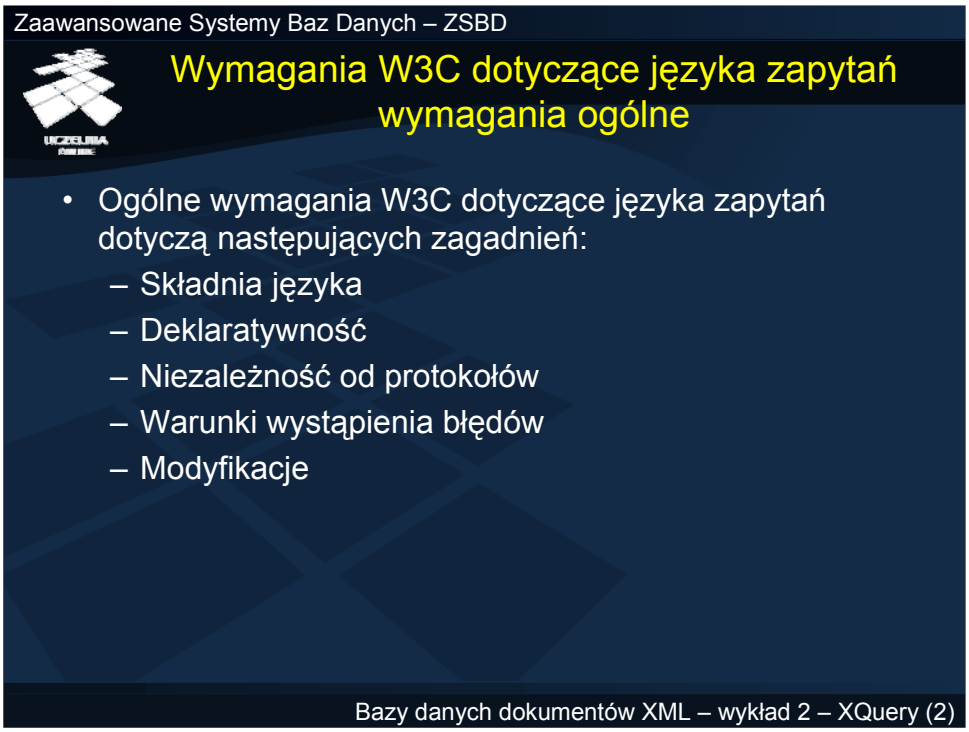
Drugi wykład z baz danych dokumentów XML zostanie poświęcony językowi zapytań XQuery. W roku 2005 język XQuery stał się standardem zatwierdzonym przez W3C.



W dokumencie "XML Query Requirements" organizacja W3C zebrała podstawowe wymagania dotyczące języków zapytań przeznaczonych do przetwarzania dokumentów XML.

Wymagania te dotyczą następujących zagadnień: składnia języka, deklaratywność, niezależność od protokołów, warunki wystąpienia błędów, modyfikacje.

I tak dla przykładu wymagania dotyczące składni języka są następujące:

- język zapytań **może** mieć więcej niż jedną składnię,

- **musi** być zrozumiały w czytaniu i pisaniu przez człowieka,

- co najmniej jedna składnia **musi** być wyrażona w formacie XML.

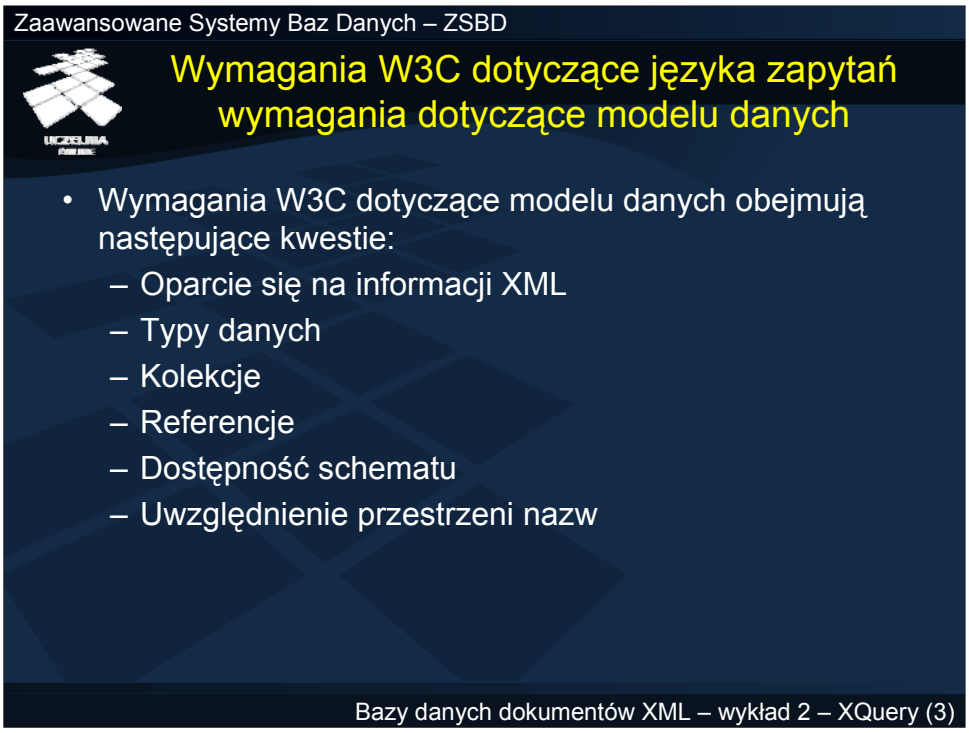
Ponadto język przeznaczony do przetwarzania dokumentów XML:

- **musi** być deklaratywny,

- **musi** być niezależny od wykorzystywanych protokołów,

- **musi** definiować zestawy standardowych informacji o błędach, które pojawiają się podczas ewaluacji zapytania,

- **nie może**, wykluczać możliwości rozszerzania go o klauzule przeznaczone do definiowania modyfikacji dokumentów XML



Wymagania W3C dotyczące modelu danych XML wykorzystywanego przez zapytania obejmują następujące kwestie: oparcie się na informacji XML, typy danych, kolekcje, referencje, dostępność schematu, uwzględnienie przestrzeni nazw.

Przykładowe ogólne wymagania W3C są następujące:

- Model danych opiera się na informacjach udostępnianych przez procesory XML i procesory schematów i musi gwarantować, że żadna dodatkowa informacja nie będzie wymagana.

- Model danych powinien umożliwiać tworzenie wszystkich elementów informacyjnie istotnych. W przypadku elementów pominiętych powinien udostępniać informacje o powodzie ich pominięcia.

- Model danych musi posiadać reprezentacje zarówno dla ciągów znaków (XML 1.0) jak i prostych i złożonych typów danych zgodnie ze specyfikacją XML Schema.

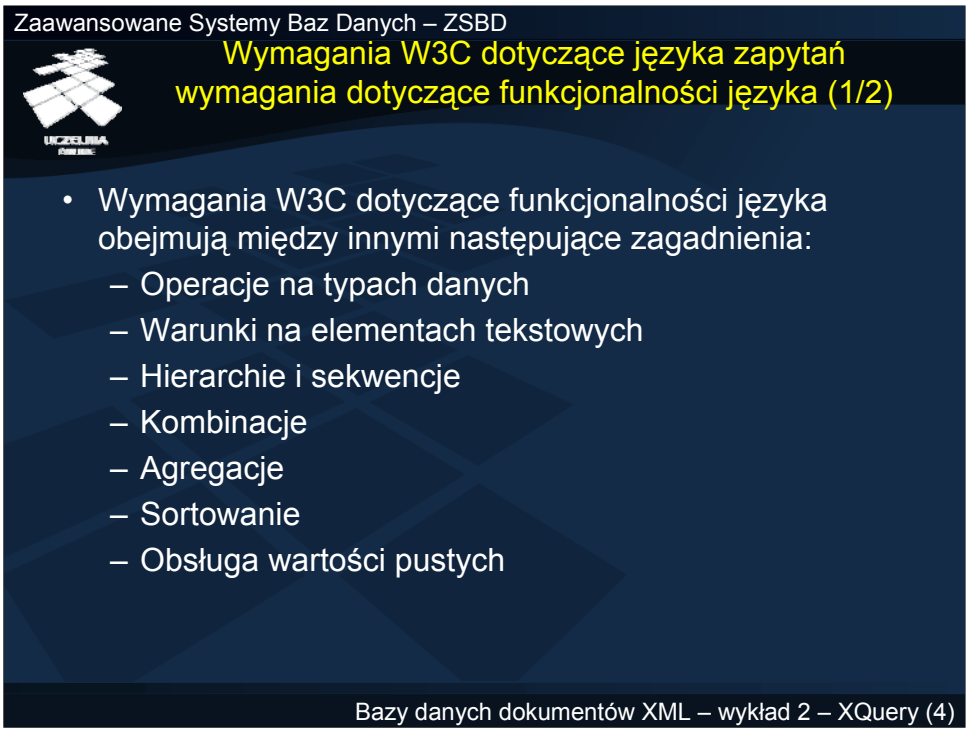
- Model danych musi posiadać reprezentacje kolekcji dokumentów oraz kolekcji wartości prostych i złożonych.

- Model danych musi wspierać referencje występujące zarówno wewnątrz dokumentu jak i pomiędzy dokumentami.

- Zapytania powinny być możliwe niezależnie od tego czy schematy dla przepytywanych dokumentów są dostępne czy też nie.

- Jeżeli schemat jest dostępny, wówczas model danych musi reprezentować każdy z elementów zgodnie z definicją schematu. Dotyczy to dla przykładu domyślnych atrybutów, encji lub typów danych. Elementy te nie są prezentowane jeśli schematy nie istnieją.

- Model danych musi uwzględniać istnienie przestrzeni nazw.



Wymagania W3C dotyczące funkcjonalności języka obejmują między innymi następujące zagadnienia: operacje na typach danych, warunki na elementach tekstowych, hierarchie i sekwencje, kombinacje, agregacje, sortowanie, obsługę wartości pustych.

Przykładowe wymagania dotyczące funkcjonalności języka zapytań są następujące:

- Język zapytań musi uwzględniać operacje na wszystkich typach danych dostępnych w ramach modelu danych.

- Zapytania muszą umożliwiać definiowanie prostych warunków dotyczących tekstu i jego zawartości.

- Operacje na kolekcjach muszą wspierać kwantyfikatory ilościowe oraz kwantyfikatory istnienia.

- Zapytania muszą umożliwiać wykonywanie operacji na hierarchiach oraz sekwencjach występujących w strukturze dokumentu.

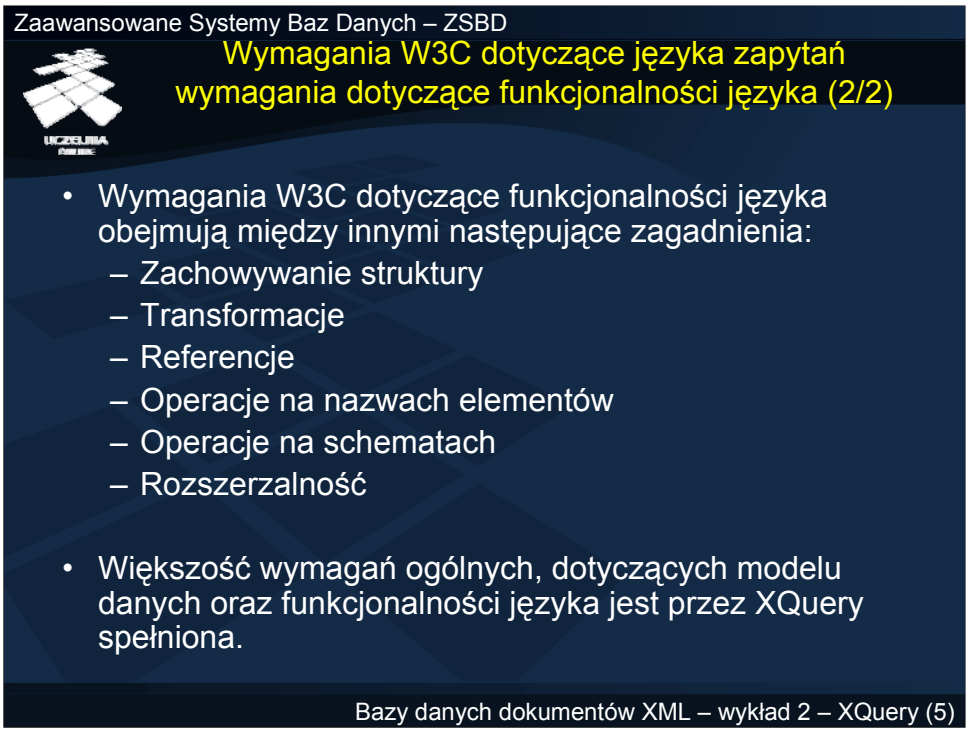
- Język zapytań przeznaczony do przetwarzania dokumentów XML musi umożliwiać łączenie powiązanej informacji znajdującej się w różnych częściach dokumentu lub wielu różnych dokumentów.

- Musi umożliwiać obliczanie podsumowań w oparciu o grupy elementów.

- Musi umożliwiać porządkowanie generowanych wyników.

- Musi umożliwiać tworzenie wyrażeń, które będą składały się ze złożonych operandów, będących dla przykładu zapytaniami.

- Musi uwzględniać możliwość występowania wartości pustych, dotyczy to również operatorów logicznych.



Wymagania W3C dotyczące funkcjonalności języka obejmują dodatkowo między innymi następujące zagadnienia: zachowywanie struktury, transformacje, referencje, operacje na nazwach elementów, operacje na schematach, rozszerzalność.

Przykładowe wymagania dotyczące funkcjonalności języka zapytań dotyczące tych zagadnień są następujące:

Zapytania muszą być zdolne do zachowywania (w wyniku) oryginalnej hierarchii lub sekwencji elementów w źródle.

Zapytania muszą umożliwiać transformacje struktur XML, muszą także umożliwiać tworzenie nowych struktur XML.

Zapytania muszą umożliwiać poruszanie się wewnątrz dokumentu, oraz pomiędzy różnymi dokumentami na podstawie referencji.

Zapytania muszą być zdolne do zachowywania tożsamości elementów w ramach modelu danych.

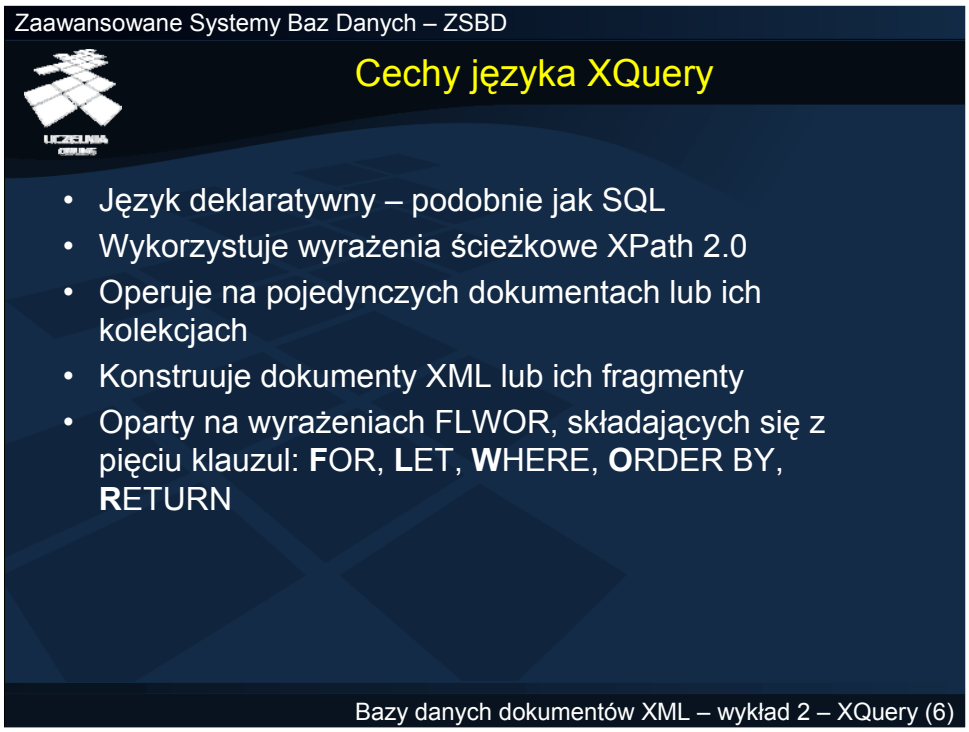
Zapytania muszą umożliwiać wykonywanie prostych operacji na nazwach. Dla przykładu porównywanie nazw, testowanie nazw elementów i atrybutów itp..

Język zapytań powinien umożliwiać wykorzystanie zewnętrznie zdefiniowanych funkcji operujących na dowolnych typach modelu danych. Interfejs takich funkcji powinien być definiowany za pomocą języka zapytań. Powinna być możliwość rozróżnienia funkcji zewnętrznych od funkcji zdefiniowanych w ramach języka zapytań.

Należy w tym momencie zaznaczyć, że znakomita większość wymagań W3C jest przez język XQuery spełniona. Z wymagań funkcjonalnych nie są spełnione wymagania dotyczące operacji na schematach. Wymagania te są następujące:

Zapytania powinny umożliwiać dostęp do schematów XML lub do DTD jeśli tylko one istnieją dla określonego dokumentu.

Jeśli schemat jest zdefiniowany w oparciu o DTD może być wymagane odpowiednie mapowanie na schemat XML.



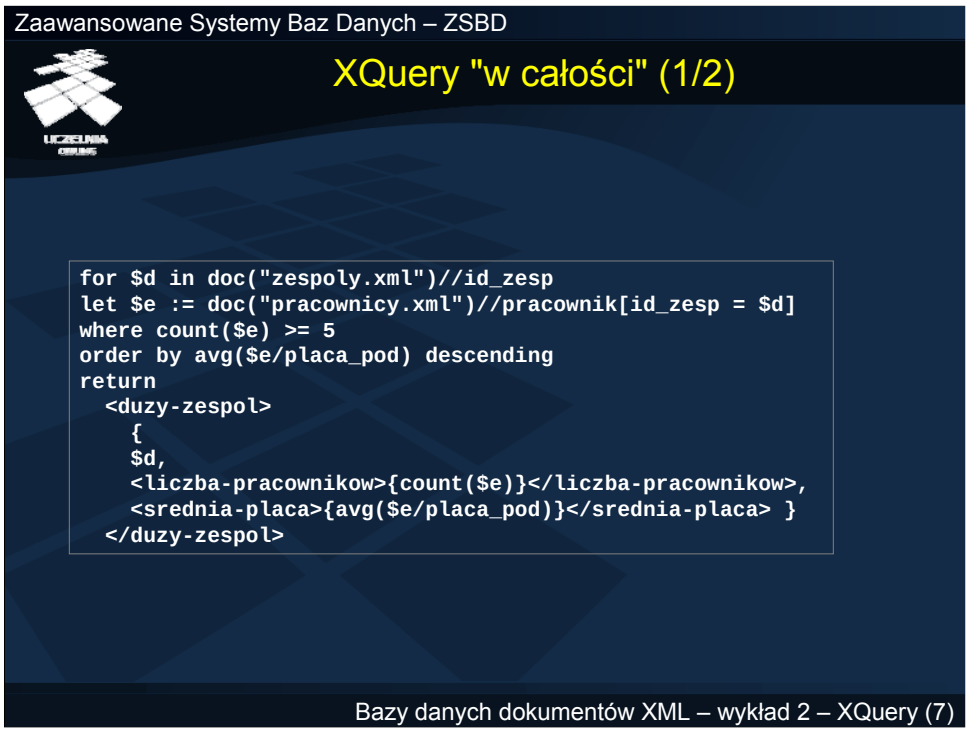
Zapytania XQuery pozwalają na definiowanie zapytań w sposób deklaratywny – podobnie jak SQL.

Wykorzystują wyrażenia ścieżkowe zgodne ze standardem XPath 2.0

Operują na pojedynczych dokumentach lub ich kolekcjach. Specjalne funkcje takie jak: input, collection, document lub doc pozwalają na wskazanie danych bazowych, na których zapytanie będzie operowało.

Polecenia XQuery konstruują dokumenty XML lub fragmenty takich dokumentów – wynik zapytania nie koniecznie musi być poprawnym dokumentem XML w sensie well-formed.

Polecenia XQuery oparte są na wyrażeniach nazywanych w skrócie FLWOR. Zbudowane są one z użyciem pięciu klauzul: FOR, LET, WHERE, ORDER BY, RETURN. Na kolejnych slajdach zostaną omówione poszczególne klauzule.



Przykładowe zapytanie XQuery przedstawione na slajdzie wykorzystuje wszystkie możliwe klauzule. Ich znaczenie w zapytaniu jest następujące

- Klauzula **FOR** wyszukuje w dokumencie zespoly.xml elementy wskazywane przez wyrażenie ścieżkowe //id_zesp. Każdy dopasowany element jest przypisywany zmiennej $d i powoduje utworzenie tzw. krotki. Klauzula FOR działa podobnie jak klauzule for w innych językach programowania wykorzystuje zmienne do iteracji po określonych wartościach.

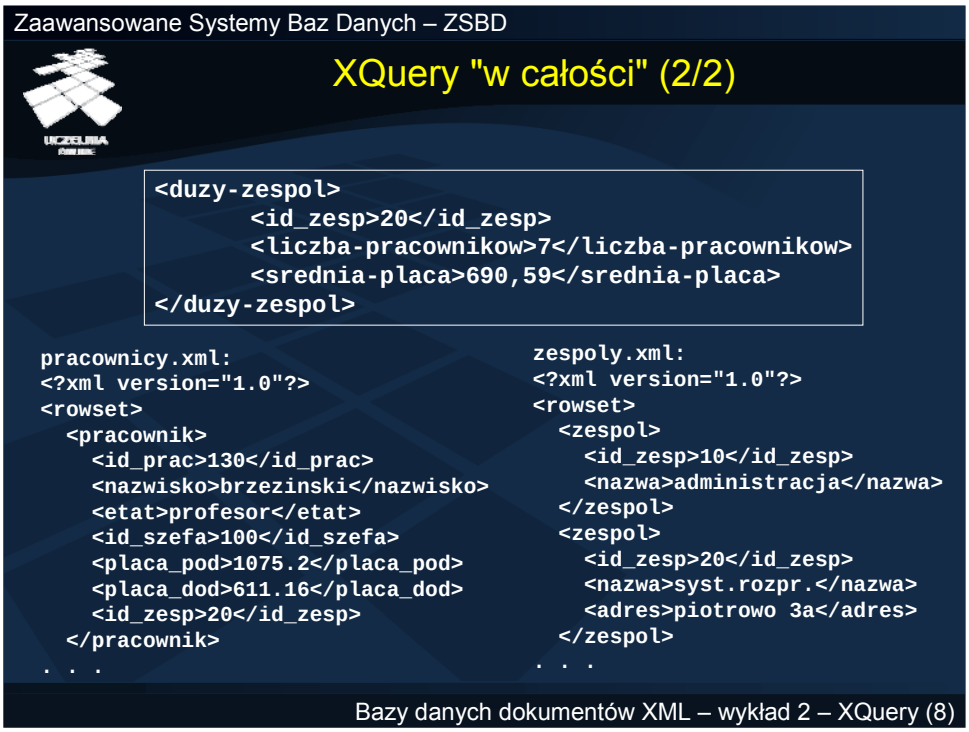
- Dla każdego przypisania zmiennej $d klauzula **LET** przypisuje zmiennej $e zbiór takich elementów pracownik, które posiadają podelement id_zesp o zawartości identycznej z wartością zmiennej $d. Elementy pracownik wyszukiwane są z dokumentu o nazwie pracownicy.xml.

- W wyniku działania klauzul FOR i LET otrzymujemy tzw. strumień krotek. Liczba krotek zależy od klauzuli FOR, natomiast liczba zmiennych w krotce zależy od klauzul FOR i LET. W naszym przykładzie liczba krotek jest równa liczbie elementów id_zesp w dokumencie zespoly.xml. Każda krotka zawiera parę zmiennych $d i $e posiadających określone wcześniej wartości.

- Zadaniem klauzuli **WHERE** jest selekcja strumienia krotek utworzonych za pomocą klauzul FOR i LET. W naszym przykładzie klauzula WHERE filtruje ten strumień pozostawiając tylko te krotki, w których zbiór elementów pracownik przypisanych do zmiennej $e składa się, z co najmniej 10 elementów.

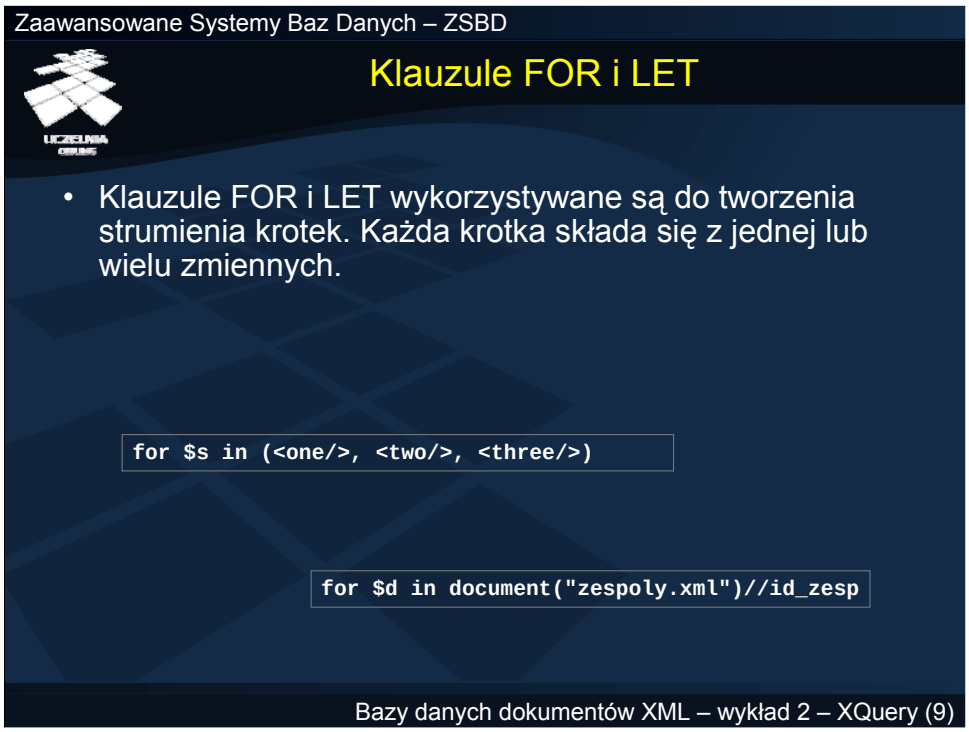
- Zadaniem klauzuli **ORDER BY** jest odpowiednie posortowanie wynikowych krotek. W powyższym przykładzie krotki zostaną posortowane wg średniej wartości elementów placa_pod znajdujących się w elementach pracownik przypisanych do zmiennej $e.

- Na zakończenie klauzula **RETURN** konstruuje wynik zapytania. W naszym przykładzie każda krotka za pomocą klauzuli RETURN stworzy element duzy-zespol, którego zawartością będzie kolejno: element $d (id_zesp), element liczba-pracowników którego zawartością będzie liczba będąca ewaluacją wyrażenia count($e) oraz element srednia-placa z zawartością wynikającą z wyrażenia avg($e/placa_pod)



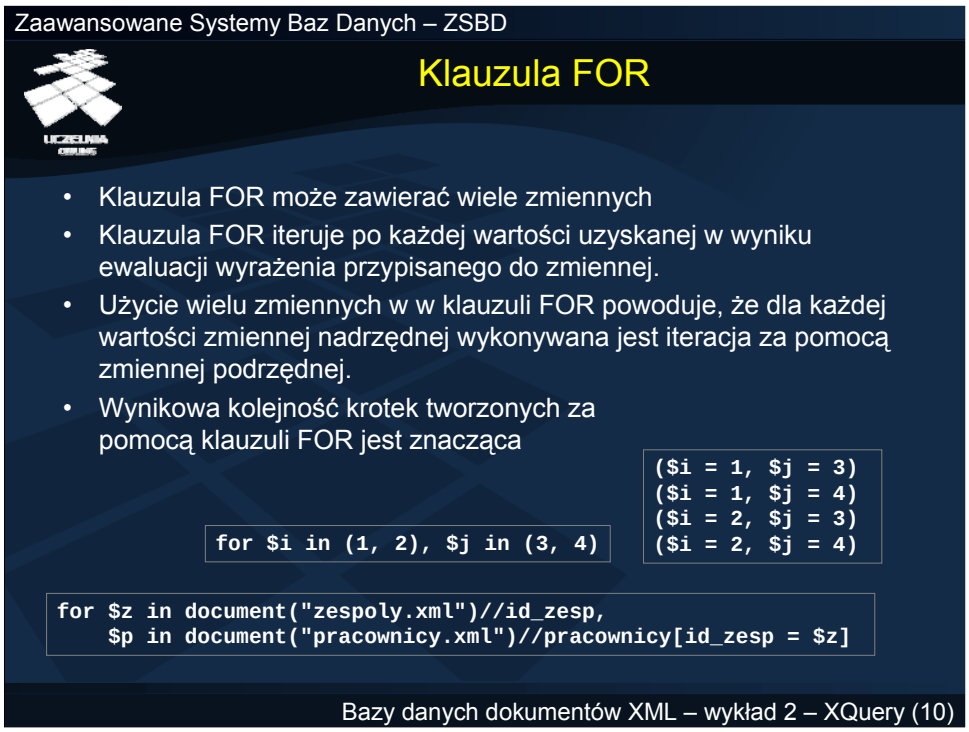
Przykładowym wynikiem zapytania z poprzedniego slajdu mógłby być dokument przedstawiony na slajdzie bieżącym w ramce. Dokumenty źródłowe mogłyby mieć natomiast postać przedstawioną poniżej.

Przejdziemy teraz do szczegółowego omówienia poszczególnych klauzul.



Tak jak wspomnieliśmy wcześniej klauzule FOR i LET wykorzystywane są do utworzenia strumienia krotek. Za liczbę krotek odpowiada klauzula FOR. Każdy węzeł dopasowany w wyniku ewaluacji wyrażenia ścieżkowego w klauzuli FOR jest przyporządkowywany odpowiedniej zmiennej użytej w tej klauzuli i stanowi podstawę do utworzenia kolejnej krotki. Zadaniem klauzuli LET jest ewentualne rozszerzenie krotek o zmienne zawierające dodatkowe informacje.

W pierwszym przykładzie ze slajdu klauzula FOR stworzy trzy krotki. W drugim przykładzie stworzy tyle krotek ile elementów id_zesp będzie w dokumencie zespoly.xml. W obu przypadkach każda krotka będzie składała się tylko z jednej zmiennej.



Klauzula FOR może zawierać wiele zmiennych. Każda zmienna może być związana z odrębnym wyrażeniem.

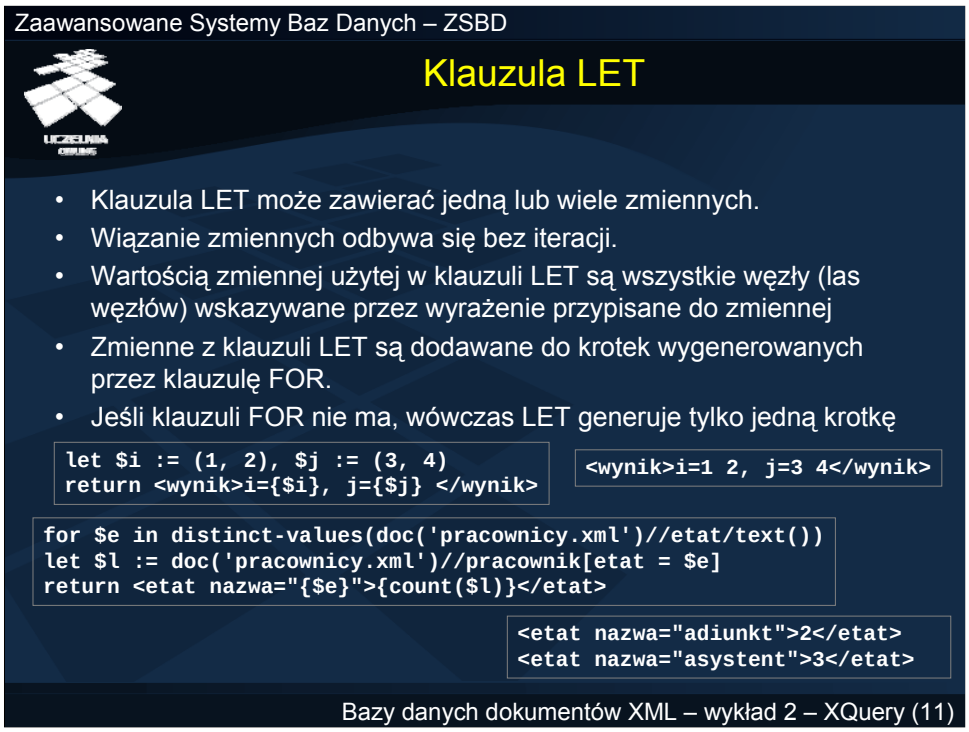
Klauzula FOR iteruje po każdej wartości uzyskanej na podstawie ewaluacji wyrażenia przypisanego do zmiennej. Oznacza to, że jeżeli wyrażenie przypisane do zmiennej $x wskazuje na 10 wartości w dokumencie źródłowym, to klauzula FOR utworzy 10 iteracji, w każdej z nich do zmiennej $x zostanie przypisana kolejna wartość z dokumentu źródłowego.

Użycie wielu zmiennych ma podobny skutek do zagnieżdżenia pętli w językach programowania. Dla każdej wartości zmiennej nadrzędnej wykonywana jest iteracja dla zmiennej podrzędnej.

Dla przykładu w pierwszym zapytaniu na slajdzie mamy dwie zmienne $i i $j. Zmienna $i jest zmienną nadrzędną – wystąpiła wcześniej w klauzuli FOR, natomiast $j jest zmienną podrzędną. A zatem dla każdego z dopasowań zmiennej $i zostanie wykonana iteracja przy użyciu zmiennej $j. Wyrażeniem przypisanym zmiennej $i jest (1, 2), natomiast wyrażeniem przypisanym zmiennej $j jest (3, 4). W efekcie wynikiem będzie strumień czterech krotek w postaci przedstawionej po prawej stronie zapytania.

W wyrażeniach przypisanych do zmiennych podrzędnych można używać zmiennych nadrzędnych tak jak ma to miejsce w przypadku drugiego zapytania umieszczonego na slajdzie. Zmiennej $z przypisane jest wyrażenie doc("zespoly.xml")//id_zesp. A zatem dla każdego elementu id_zesp w dokumencie zespoly.xml zostanie wykonana iteracja przy użyciu zmiennej $z. Zmiennej $p natomiast przypisane zostało wyrażenie, które wykorzystuje zmienną $z. W naszym przypadku dla każdej iteracji z użyciem zmiennej $z zostanie wykonana iteracja z użyciem zmiennej $p, której zostaną przypisywane elementy pracownik posiadające podelement id_zesp identyczny z wartością zmiennej $z.

Wynikowa kolejność krotek tworzonych przez klauzulę FOR jest znacząca, co oznacza, że nie może być zmieniona samoistnie. Do zmiany kolejności krotek służy klauzula ORDER BY.



Klauzula LET także może zawierać jedną lub wiele zmiennych. Różnica w stosunku do FOR polega na tym, że wiązanie zmiennych odbywa się bez iteracji na podstawie wyrażenia przypisanego zmiennej. Innymi słowy wartością zmiennej użytej w klauzuli LET są wszystkie węzły wskazywane przez wyrażenie przypisane do zmiennej.

Przykładowo pierwsze zapytanie na slajdzie wygeneruje wynik przedstawiony po jego prawej stronie. Zwróćmy uwagę na to, że nie doszło do żadnych iteracji. Do zmiennych $i i $j zostały przypisane odpowiednio wyrażenia 1,2 i 3,4.

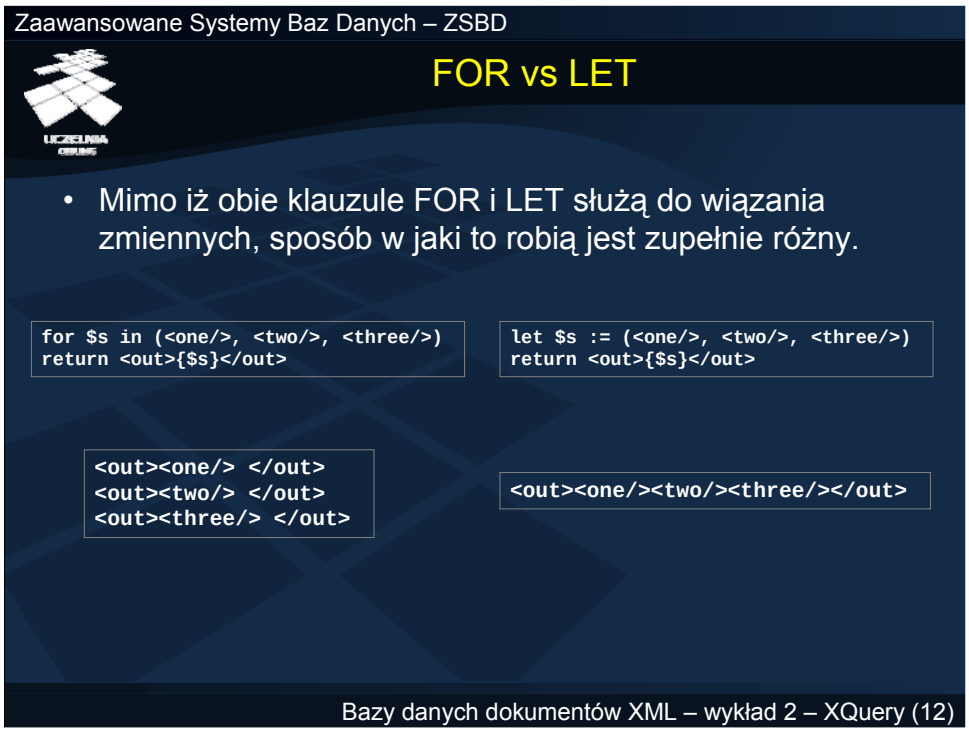
Wynika to z faktu, iż zarówno do zmiennej $i jak i $j przypisany został las węzłów określony na podstawie wyrażeń przypisanych zmiennym.

Zwróćmy uwagę, że w omawianym zapytaniu nie występuje klauzula FOR. Takie zapytania oczywiście są możliwe i generują zawsze jedną krotkę.

Zmienne z klauzuli LET są dodawane do krotek wygenerowanych przez klauzulę FOR.

Przeanalizujmy, przykładowo drugie zapytanie istniejące na slajdzie.

Klauzula FOR utworzy krotki ze zmiennymi $e, których wartością będą ciągi znaków umieszczone w elementach etat w dokumencie pracownicy.xml. Krotek będzie tyle ile różnych wartości tekstowych w elementach etat w dokumencie pracownicy.xml. Dla każdej wartości zmiennej $e, klauzula LET przypisze do zmiennej $l las elementów pracownik, posiadających w podelemencie etat wartość zgodną z ciągiem znaków przypisanym do zmiennej $e. W efekcie ewaluacji powyższego zapytania możemy otrzymać wynik w postaci przedstawionej na dole slajdu.

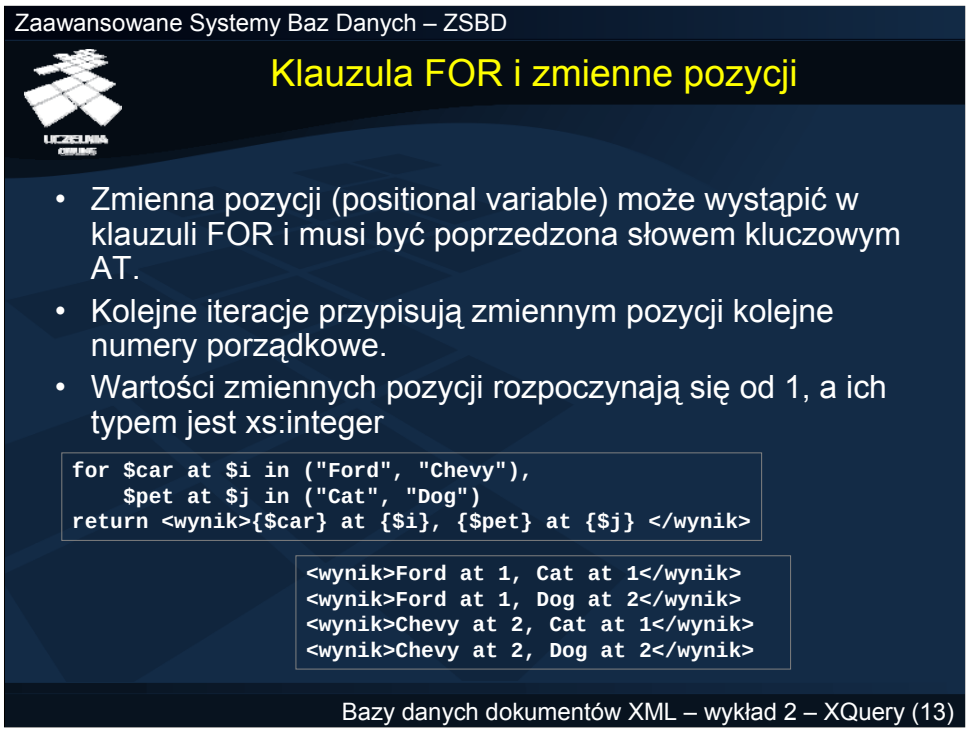


Mimo iż obie klauzule FOR i LET służą do wiązania zmiennych, sposób, w jaki to robią, jest zupełnie różny. Dobrze podsumowują ten fakt zapytania przedstawione na aktualnym slajdzie.

Lewe zapytanie skonstruuje trzy krotki. Do zmiennej $s w trzech kolejnych iteracjach zostaną przypisane trzy kolejne elementy z wyrażenia przypisanego zmiennej. Pod zapytaniem przedstawiono jego wynik.

Zapytanie przedstawione na slajdzie po prawej stronie skonstruuje tylko jedną krotkę. Wartością zmiennej $s będzie las elementów pochodzący z wyrażenia przypisanego zmiennej.

Poniżej przedstawiono wynik tego zapytania.



Zmienna pozycji (ang. *positional variable*) może wystąpić w klauzuli FOR i musi być poprzedzona słowem kluczowym AT.

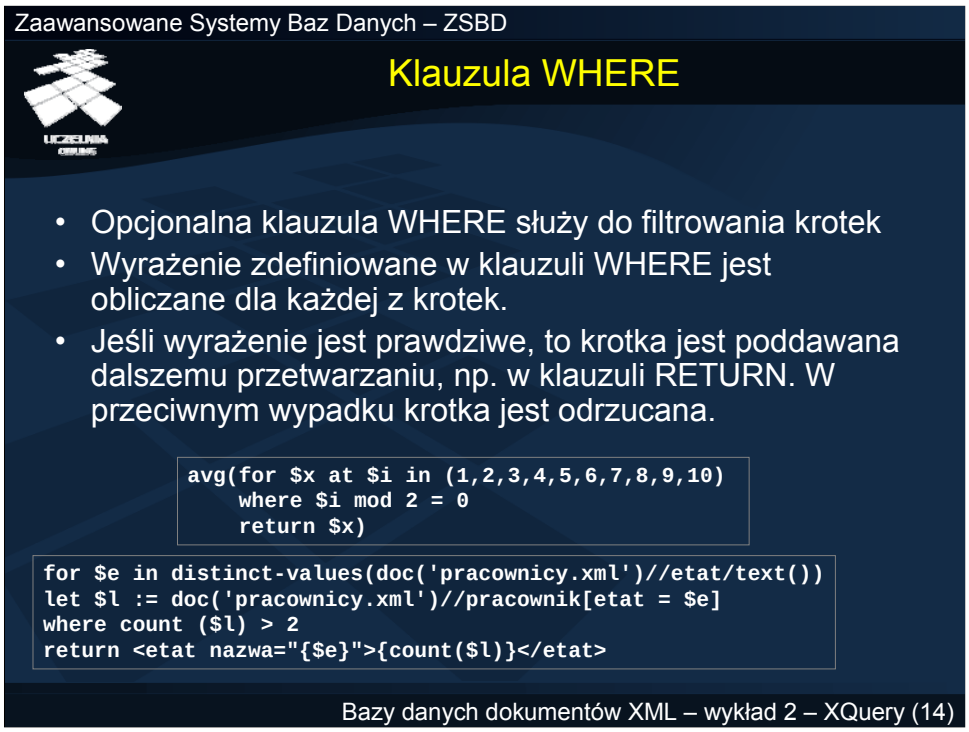
Kolejne iteracje przypisują zmiennym pozycji kolejne numery porządkowe.

Wartości zmiennych pozycji rozpoczynają się od wartości 1, a ich typem jest xs:integer.

Przykładowo zapytanie, które zostało przedstawione na slajdzie przyporządkuje zmiennym $i kolejno wartości 1 i 2, analogiczne wartości zostaną przypisane zmiennej $j. Wynik powyższego zapytania będzie miał postać przedstawioną poniżej zapytania.

Wartości zmiennych pozycji są niezależne od ewentualnego sortowania. I tak, gdyby przykładowo do naszego zapytania dodać klauzulę ORDER BY sortującą ostateczny wynik względem rosnącej wartości zmiennej $j, a następnie względem wartości zmiennej $car, to wygenerowany wynik byłby identyczny z przedstawionym na slajdzie z dokładnością do kolejności krotek.

Oczywiście zmienne pozycji mogą być wykorzystywane tak samo jak każde inne zmienne. Zwiększają one liczbę zmiennych w każdej z krotek generowanych przez klauzule FOR



Podobnie jak klauzule FOR i LET klauzula WHERE jest opcjonalna.

Zadaniem klauzuli WHERE jest selekcja krotek wygenerowanych za pomocą klauzul FOR i LET, czyli zaprzestanie przetwarzania (odrzucenie) krotek, które nie spełniają zadanego warunku.

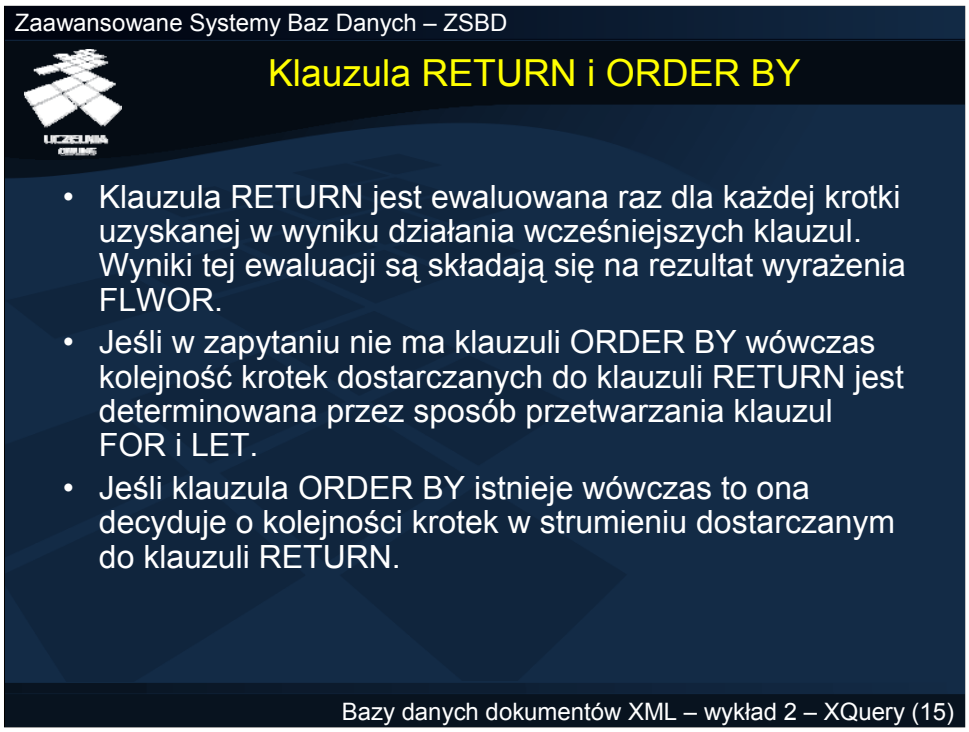
Wyrażenie zdefiniowane w klauzuli WHERE powinno być wyrażeniem logicznym i jest obliczane dla każdej z krotek. Jeśli wyrażenie jest prawdziwe, to krotka poddawana jest dalszemu przetwarzaniu, np. w ramach klauzuli RETURN lub ORDER BY. W przeciwnym wypadku krotka jest odrzucana.

Przykładowo klauzula WHERE w podzapytaniu umieszczonym we wnętrzu funkcji AVG (zapytanie to zostało przedstawione na slajdzie jako pierwsze) z dziesięciu krotek wygenerowanych przez klauzulę FOR, pozostawi tylko pięć, które dzięki klauzuli RETURN zostaną przekazane do dalszego przetwarzania za pomocą funkcji AVG. W naszym przypadku wartości te będą musiały spełnić warunek podzielności bez reszty przez 2, a zatem będą to wartości 2,4,6,8,10.

Zwróćmy uwagę, że w zapytaniu na poziomie funkcji AVG nie występują żadne z omówionych wcześniej klauzul.

Innym przykładem może być zapytanie przedstawione na slajdzie jako drugie.

W tym drugim przypadku elementy etat zostaną utworzone przez klauzulę RETURN tylko w przypadku, gdy na danym etacie znajduje się więcej niż dwóch pracowników. Odrzucenie pozostałych etatów – krotek jest realizowane za pomocą klauzuli WHERE weryfikującej liczbę elementów przypisanych do zmiennej $l.

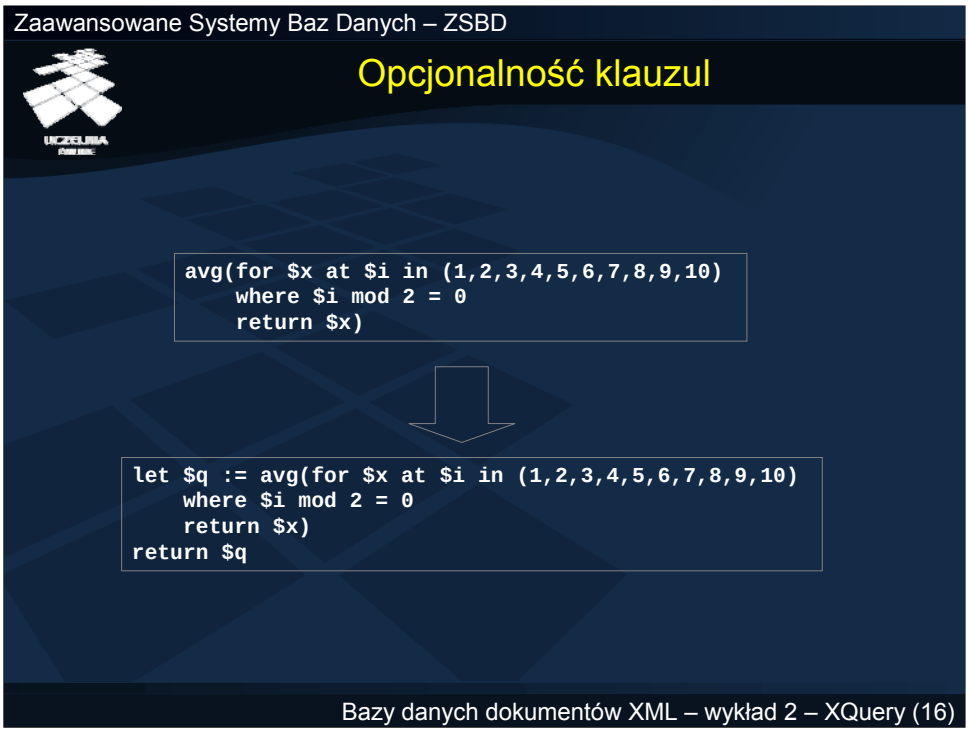


Zadaniem klauzuli RETURN jest konstruowanie wyniku zapytania XQuery. Wynik może mieć postać dokumentów XML (poprawnych w sensie well-formed) lub ich fragmentów (nie spełniających reguł poprawności w sensie well-formed) .

Klauzula RETURN jest ewaluowana raz, dla każdej krotki uzyskanej w wyniku działania wcześniejszych klauzul. Wyniki tej ewaluacji składają się na rezultat wyrażenia FLWOR.

Jeśli w zapytaniu nie ma klauzuli ORDER BY wówczas kolejność krotek dostarczanych do klauzuli RETURN jest determinowana przez sposób przetwarzania klauzul FOR i LET.

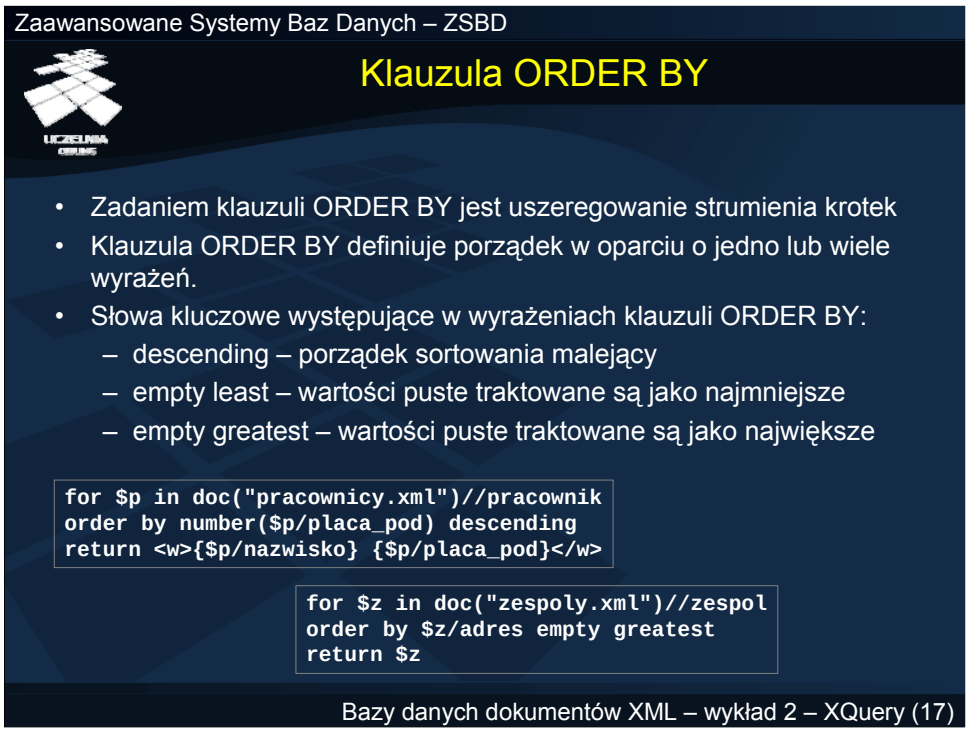
Jeśli klauzula ORDER BY istnieje wówczas to ona decyduje o kolejności krotek w strumieniu dostarczanym do klauzuli RETURN.



Zapytania XQuery mogą nie posiadać żadnej z omawianych klauzul.

Jako przykład przeanalizujmy wcześniejszy przykład z użyciem funkcji AVG. (został on ponownie przedstawiony na bieżącym slajdzie). Zapytanie na poziomie funkcji AVG nie posiada żadnej zmiennej, żadnej klauzuli. W przypadku takich zapytań zakłada się, że wyrażenie XQuery będące treścią zapytania zostaje przypisane za pomocą niejawnej klauzuli LET do niejawnej zmiennej, która następnie jako jedyna użyta jest w niejawnej klauzuli RETURN.

Innymi słowy zapytanie przedstawione w górnej części slajdu jest skrótem zapytania przedstawionego poniżej.



Zadaniem klauzuli ORDER BY jest uszeregowanie strumienia krotek wg określonego porządku. Porządek ten definiuje się w sposób bardzo podobny jak to jest przypadku klauzuli ORDER BY w języku SQL.

Klauzula ORDER BY może zawierać definicje porządku opartą na jednym lub wielu wyrażeniach.

Porządek wyznaczany jest poprzez sortowanie krotek w oparciu o wyrażenia kolejno od lewej do prawej.

Słowa kluczowe, jakie mogą wystąpić przy wyrażeniach znajdujących się w klauzuli ORDER BY to:

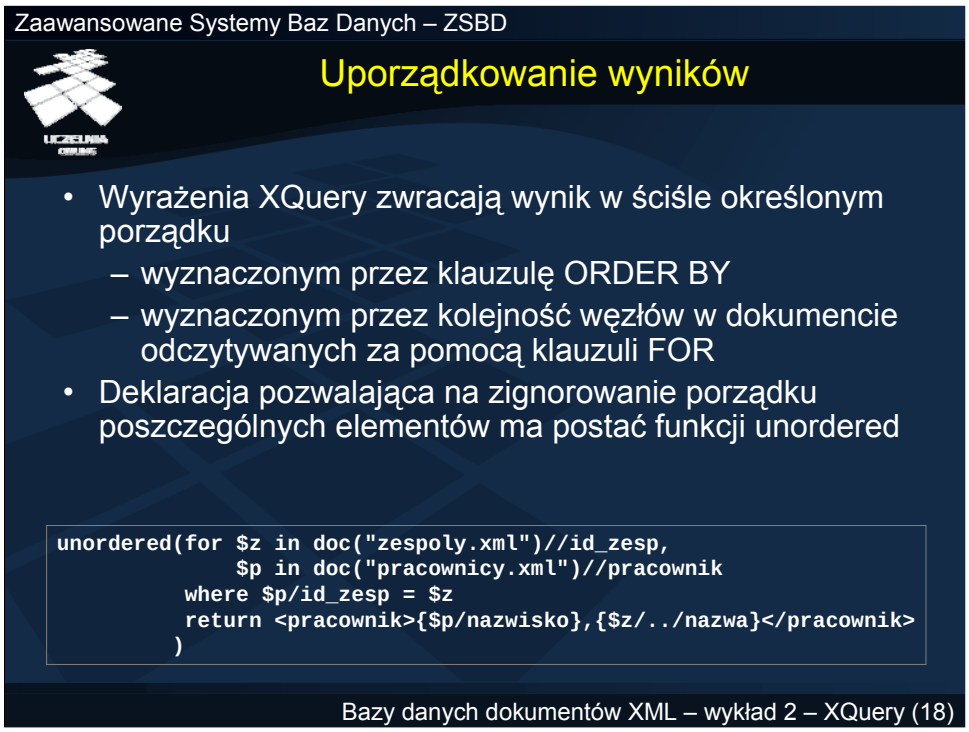
descending – ustala porządek sortowania malejący, domyślnym porządkiem jest porządek rosnący

empty least – powoduje, że wartości puste ustawiane są na końcu

empty greatest – w tym przypadku wartości puste traktowane są jako największe

stable – gwarantuje zachowanie oryginalnego porządku krotek (wynikającego z przetwarzania klauzul FOR i LET) w przypadku takiej samej wartości wyrażenia. W przeciwnym wypadku kolejność może być dowolna np.. zależna od sposobu realizacji operacji sortowania.

Przykładowo zapytanie przedstawione na slajdzie jako pierwsze, uszereguje strumień krotek generowany przez klauzule FOR i LET zgodnie z malejącymi wartościami elementów placa_pod. Natomiast zapytanie przedstawione na slajdzie jako drugie, uszereguje strumień krotek alfabetycznie względem elementu adres, przy czym wartości puste zostaną potraktowane jako największe.



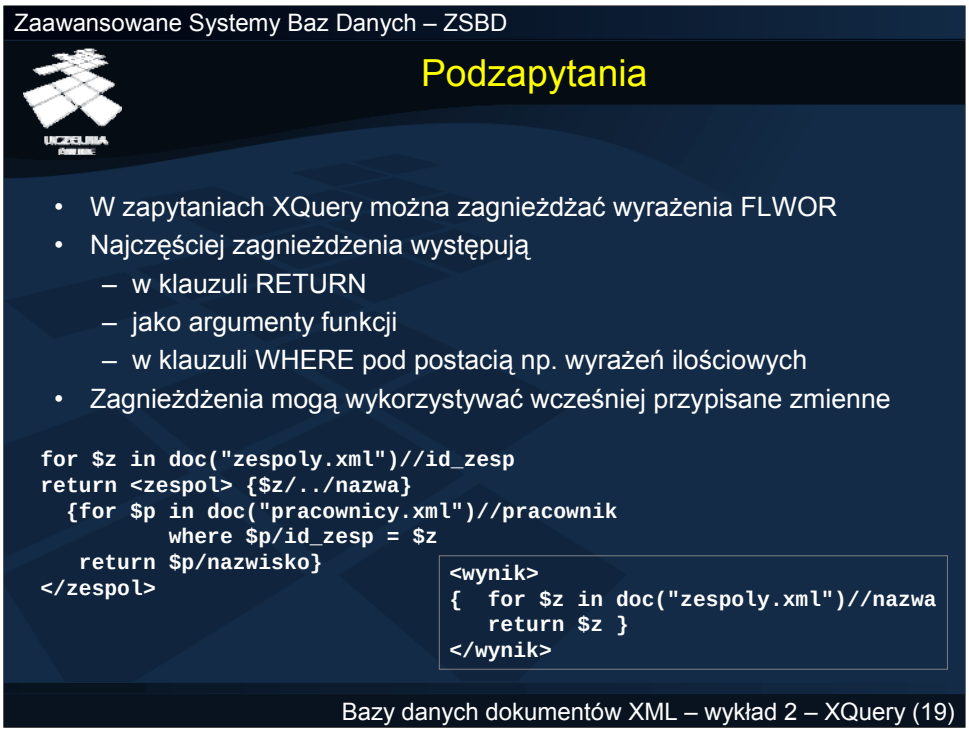
W ogólnym przypadku wyrażenia XQuery zwracają wynik w ściśle określonym porządku. Wynika on albo z wyrażeń umieszczonych w klauzuli ORDER BY albo z oryginalnego porządku węzłów odczytywanych za pomocą klauzuli FOR.

Mimo iż dokumenty XML stanowią uporządkowane zbiory tekstowe, użytkownika może nie interesować kolejność zwracanych przez zapytanie XQuery elementów. Ze względów efektywnościowych można, zatem pozwolić sobie na pobieranie poszczególnych elementów bez zachowywania ich oryginalnego porządku.

Korzyści z takiego rozwiązania mogą być znaczące dla przykładu w sytuacjach, gdy dokumenty w rzeczywistości składowane są dla przykładu w relacyjnych lub obiektowych bazach danych i odtwarzanie dokumentów XML z uwzględnieniem oryginalnej kolejności elementów może być operacją kosztowną.

Deklaracja pozwalająca na zignorowanie porządku wynikowego ma postać funkcji unordered. Argumentem funkcji jest zapytanie, dla którego wynikowe krotki mogą być uzyskane w dowolnej kolejności.

Przykładowe zapytanie z wykorzystaniem funkcji unordered zostało przedstawione na slajdzie. Gdyby funkcji unordered nie było zbiór wynikowych elementów pracownik powinien się rozpoczynać od pracowników należących do pierwszego zespołu umieszczonego w dokumencie zespoly.xml. Użycie funkcji unordered może dla przykładu spowodować wykonanie operacji połączenia informacji pochodzącej z dokumentu zespoly.xml i pracownicy.xml w dowolny sposób, a następnie wygenerowanie odpowiedzi bez dodatkowego sortowania.



W zapytaniach XQuery można zagnieżdżać wyrażenia FLWOR.

Najczęściej zagnieżdżenia występują:

- w klauzuli RETURN

- jako argumenty funkcji

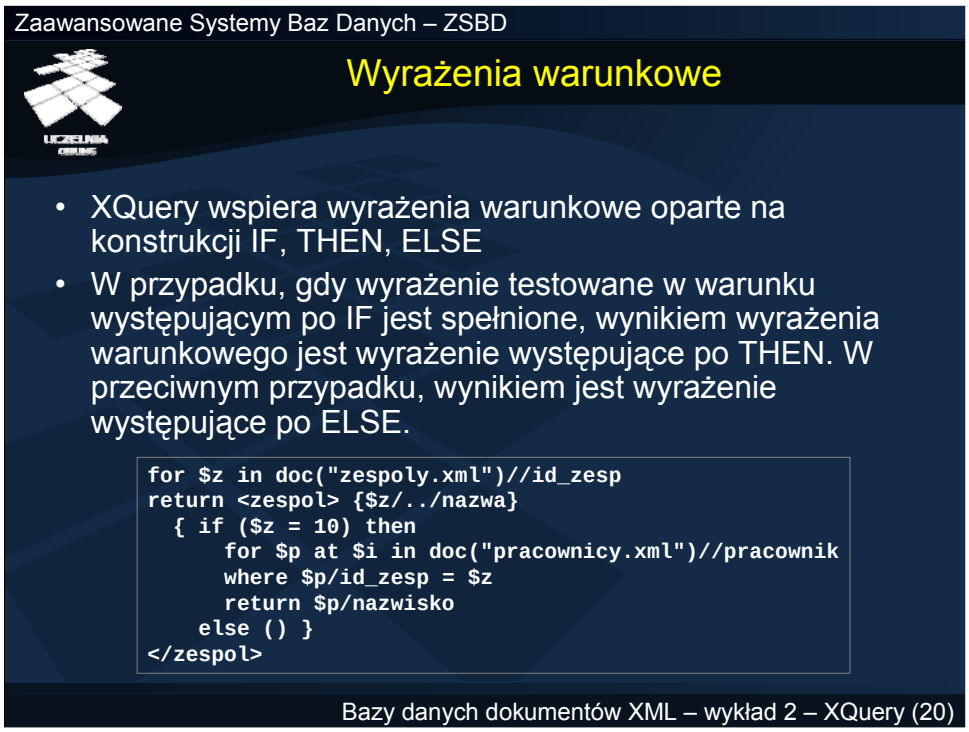
- w klauzuli WHERE pod postacią np. wyrażeń ilościowych

Zagnieżdżenia mogą wykorzystywać wcześniej przypisane zmienne. Zagnieżdżenia w klauzuli RETURN, umieszczone pomiędzy ciągami znaków tworzącymi szablon wyniku, ograniczamy nawiasami klamrowymi. W podobny sposób postępujemy z wymagającymi ewaluacji odwołaniami do wartości zmiennych.

Przykładowe zapytania z zagnieżdżonym wyrażeniem FLWOR umieszczono na slajdzie.

W pierwszym przypadku zapytanie zewnętrzne odwołuje się do dokumentu zespoly.xml i wydobywa z niego elementy id_zesp, które przypisuje do zmiennej $z, każdorazowo tworząc nową krotkę. Dla każdej krotki klauzula RETURN tworzy element zespol, w którego wnętrzu zostanie umieszczony element nazwa (ewaluacja zmiennej wymaga użycia nawiasów klamrowych) oraz wynik zagnieżdżonego wyrażenia FLWOR – las elementów nazwisko. W podzapytaniu wykorzystano zewnętrzną zmienną $z do ograniczenia krotek (wydobycia pracowników należących do danego zespołu) co oczywiście skutkuje tym że podzapytanie utworzy odpowiednio wyselekcjonowane elementy nazwisko.

Podzapytanie bywa też najczęstszym sposobem na generowanie wyników w postaci dokumentów poprawnych w sensie well-formed. Zapytanie drugie jest tego przykładem.

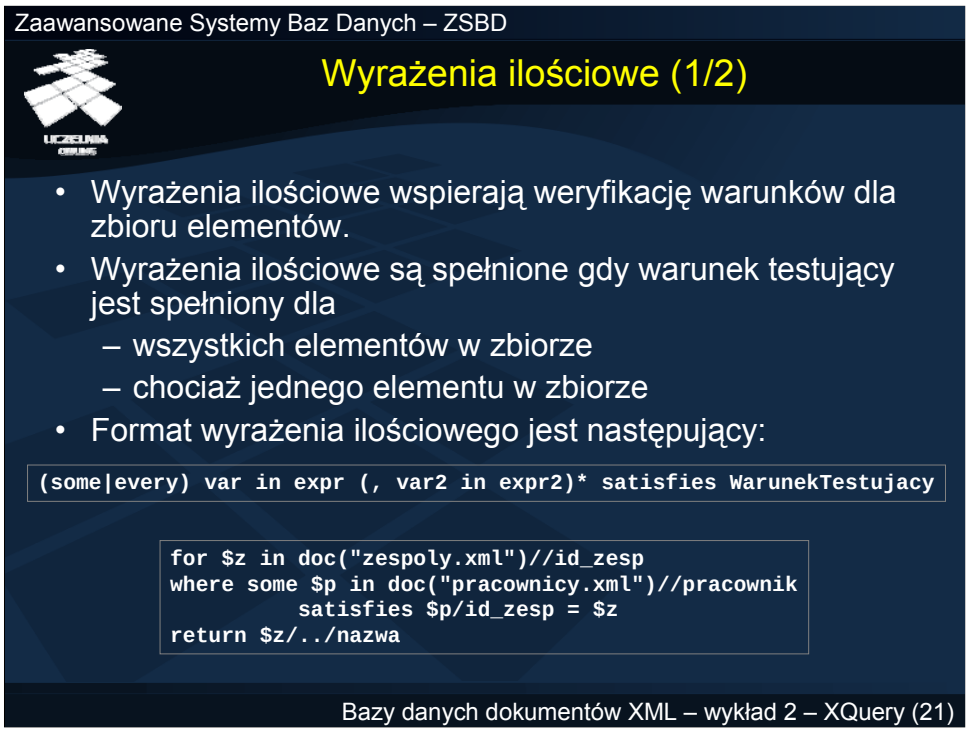


XQuery wspiera także wyrażenia warunkowe oparte na konstrukcji IF, THEN, ELSE. Wykorzystuje się je najczęściej do warunkowego tworzenia fragmentów wyników.

W przypadku, kiedy wyrażenie testowane w warunku występującym po słowie kluczowym IF jest spełnione, wynikiem wyrażenia warunkowego jest wyrażenie występujące po słowie kluczowym THEN. W przeciwnym przypadku wynikiem wyrażenia warunkowego jest wyrażenie występujące po słowie kluczowym ELSE.

Przykład zapytania z użyciem klauzuli IF THEN ELSE został przedstawiony na slajdzie. W zapytaniu tym nazwiska pracowników są generowane tylko dla zespołu o identyfikatorze 10-tym. Dla pozostałych zespołów element wynikowy zespol będzie składał się tylko z jednego podelementu nazwa.

Zwróćmy uwagę na obowiązkową kompletność wyrażenia IF, THEN, ELSE (konieczność użycia wszystkich słów kluczowych)



Wyrażenia ilościowe wspierają weryfikację warunków dla zbioru elementów.

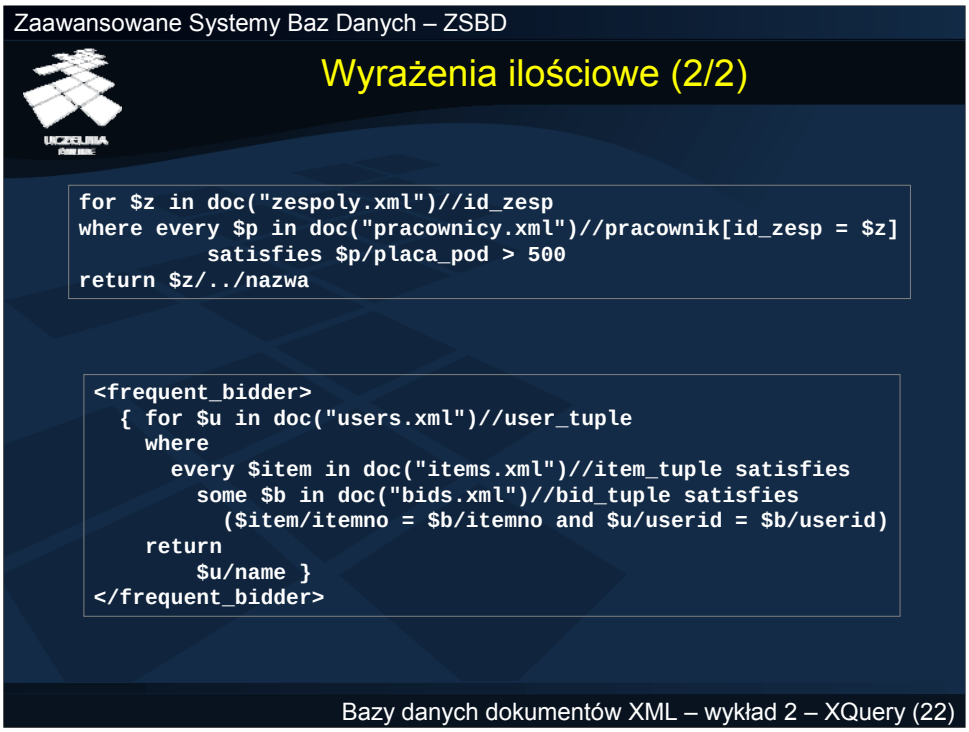
Wyrażenia ilościowe są spełnione, gdy warunek testujący jest spełniony dla:

- wszystkich elementów w zbiorze (every),

- chociaż jednego elementu w zbiorze (some).

Format wyrażenia ilościowego został przedsatwiony na slajdzie. Po słowie kluczowym satisfies umieszczamy warunek testujący który w zależności od użytego słowa kluczowego (some lub every) będzie musiał być spełniony dla choćby jednego lub dla każdego elementu w zbiorze tworzonym przez wyrażenie ilościowe. Z reguły warunek testujący wykorzystuje zmienne zastosowane w wyrażeniu ilościowym.

Przykład prostego zapytania wykorzystującego wyrażenie ilościowe został zamieszczony na slajdzie. W zapytaniu tym poszukiwane są zespoły posiadające co najmniej jednego pracownika.



Aby bliżej zapoznać się z możliwościami wyrażeń ilościowych przeanalizujmy dwa kolejne zapytania.

W pierwszym zapytaniu chcemy poznać nazwę zespołów, w których wszyscy pracownicy zarabiają powyżej 500 zł.

Nieco bardziej złożonym przypadkiem jest drugie zapytanie. Wykorzystuje ono trzy dokumenty:

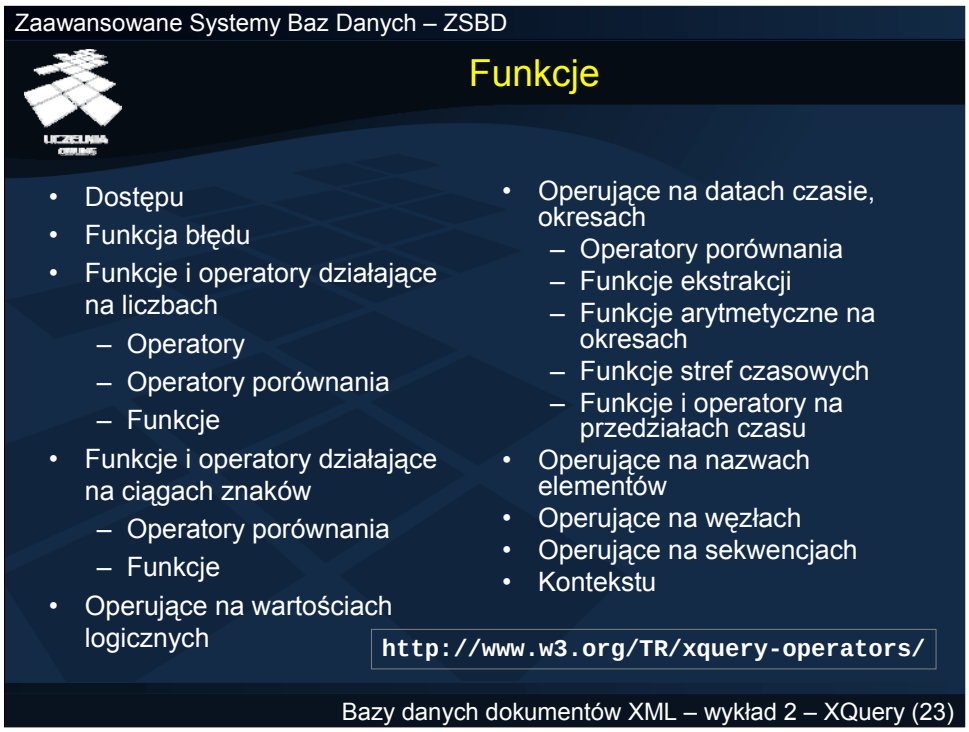
- users.xml – zawiera on informację o użytkownikach serwisu aukcyjnego

- items.xml – zawiera informacje o przedmiotach wystawionych na aukcjach

- bids.xml – zawiera informacje o poszczególnych ofertach składanych przez użytkowników na rzecz wystawionych przedmiotów.

Przejdźmy teraz do analizy naszego zapytania. Element frequent_bidder został użyty w celu zagwarantowania wyniku poprawnego w sensie well-formed. W jego wnętrzu zapytanie przegląda informacje dotyczące użytkowników z dokumentu users.xml. Klauzula WHERE sprawdza czy dla każdego przedmiotu (o którym informacja istniejące w dokumencie items.xml) użytkownik (dostępny za pomocą zmiennej $u) złożył chociaż jedną ofertę (informacja o tej ofercie znajduje się w dokumencie bids.xml).

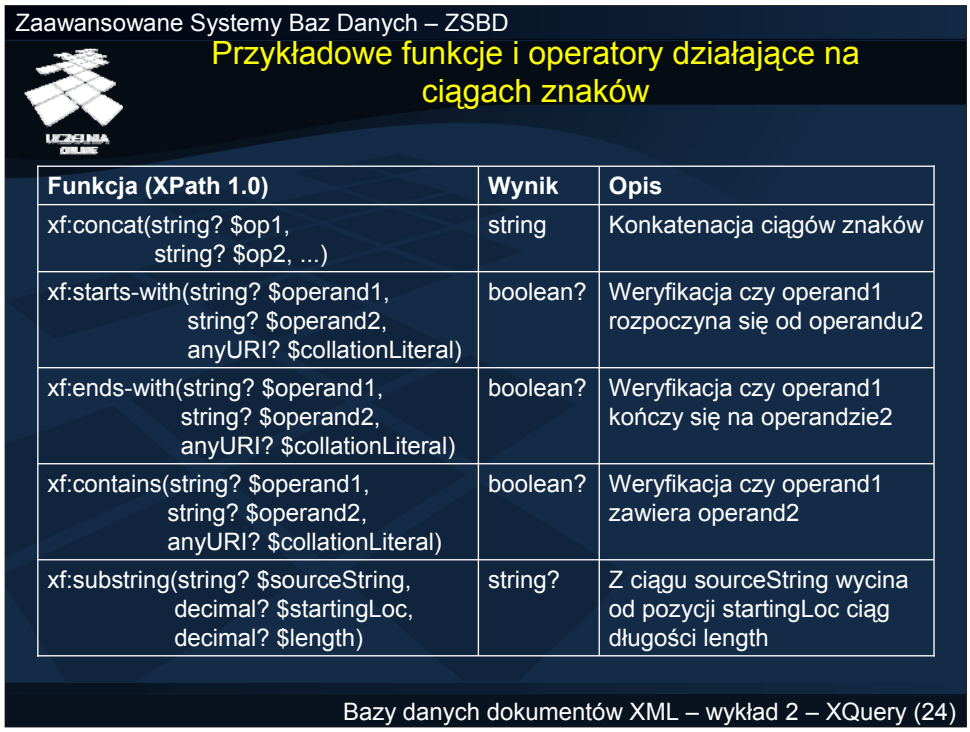
Przykład ten jest bardzo popularny i w znakomity sposób pokazuje możliwości wyrażeń ilościowych.



XQuery jest bardzo bogaty we wszelkiego rodzaju funkcje. Oprócz funkcji predefiniowanych, użytkownik może skorzystać z funkcji definiowanych samodzielnie.

Funkcje predefiniowane można podzielić w następujący sposób:

- Funkcje dostępu
- Funkcja błędu
- Funkcje i operatory działające na liczbach
  * Operatory
  * Operatory porównania
  * Funkcje
- Funkcje i operatory działające na ciągach znaków
  * Operatory porównania
  * Funkcje
- Funkcje operujące na wartościach logicznych
- Funkcje operujące na datach czasie, okresach
  * Operatory porównania
  * Funkcje ekstrakcji
  * Funkcje arytmetyczne na okresach
  * Funkcje stref czasowych
  * Funkcje i operatory na przedziałach czasu
- Funkcje operujące na nazwach elementów
- Funkcje operujące na węzłach
- Funkcje operujące na sekwencjach
- Funkcje kontekstu



Na bieżącym slajdzie przedstawiono przykładowe funkcje i operatory działające na ciągach znaków. I tak:

- funkcja concat łączy ciągi znaków przekazane jej jako parametry,

- operator start-with sprawdza czy pierwszy parametr rozpoczyna się od ciągu znaków przekazanego pod postacią drugiego parametru,

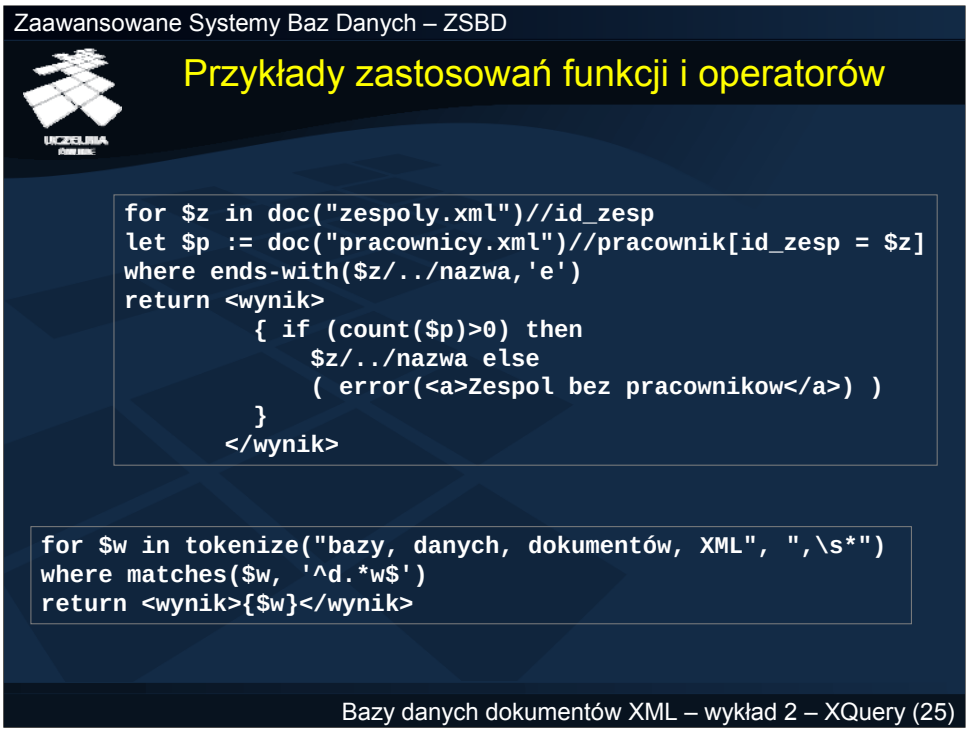
- operator ends-with sprawdza czy pierwszy parametr kończy się zestawem znaków przekazanym pod postacią drugiego parametru,

- operator contains sprawdza czy pierwszy parametr zawiera zestaw znaków przekazany jako drugi parametr,

- a funkcja substring pozwala na wycięcie dowolnego fragmentu z podanego ciągu znaków.

Pełen zestaw funkcji wchodzących w zakres standardu XQuery jest przedstawiony na stronie organizacji W3C (http://www.w3.org/TR/xquery-operators/)

Na następnym slajdzie zostanie przedstawionych kilka zapytań wykorzystujących funkcje i operatory wchodzące w zakres standardu XQuery.

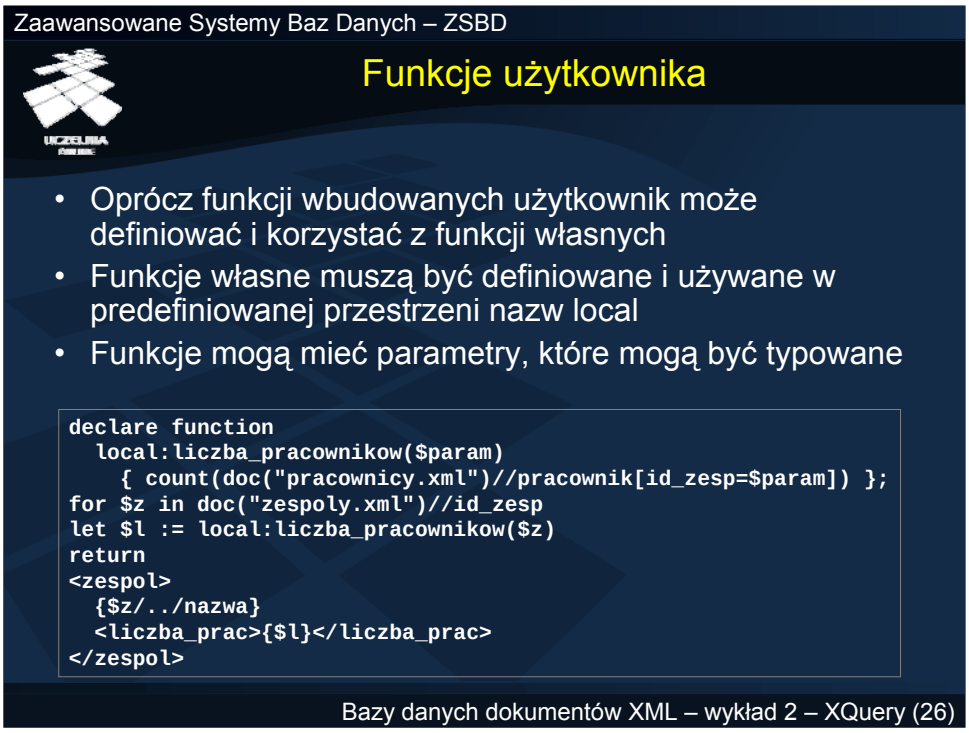


Zapytania przedstawione na bieżącym slajdzie zawierają przykłady zastosowań funkcji i operatorów.

Zapytanie pierwsze przegląda elementy id_zesp znajdujące się w dokumencie zespoly.xml i przypisuje je do zmiennej $z. Dla każdego znalezionego elementu poszukiwane są elementy pracownik posiadające element id_zesp o wartości identycznej z wartością zmiennej $z. Wszystkie znalezione elementy pracownik są przypisywane zmiennej $p. Warunek umieszczony w klauzuli WHERE został zbudowany w oparciu o operator ends-with. Warunek nakłada filtr na generowany strumień krotek – do dalszego przetwarzania zostaną skierowane te zespoły, których nazwa kończy się na literę e.

Klauzula RETURN w pierwszym zapytaniu wykorzystuje przetwarzanie warunkowe. Jeżeli liczba pracowników w danym zespole jest mniejsza lub równa 0 wówczas wykonywanie zapytania zostanie przerwane i zgłoszony zostanie błąd o treści: Zespol bez pracownikow. Zgłoszenie błędu będzie możliwe dzięki zastosowaniu funkcji błędu – error.

Drugie zapytanie za pomocą funkcji tokenize dzieli ciąg znaków "bazy, danych, dokumentów, XML" na fragmenty przy założeniu, że separatorem wyznaczającym poszczególne składowe jest znak ",". Kolejne fragmenty przypisywane są w kolejnych krotkach do zmiennej $w. Wartość zmiennej $w w klauzuli WHERE jest analizowana za pomocą funkcji matches. Funkcja matches porównuje zgodność ciągu znaków ze wzorcem. W naszym przypadku sprawdza, czy zawartość zmiennej $w rozpoczyna się od litery d i kończy na literze w, tylko takie fragmenty będą podlegały dalszemu przetwarzaniu za pomocą klauzuli RETURN.



Oprócz funkcji wbudowanych użytkownik może definiować i korzystać z funkcji własnych – lokalnych.

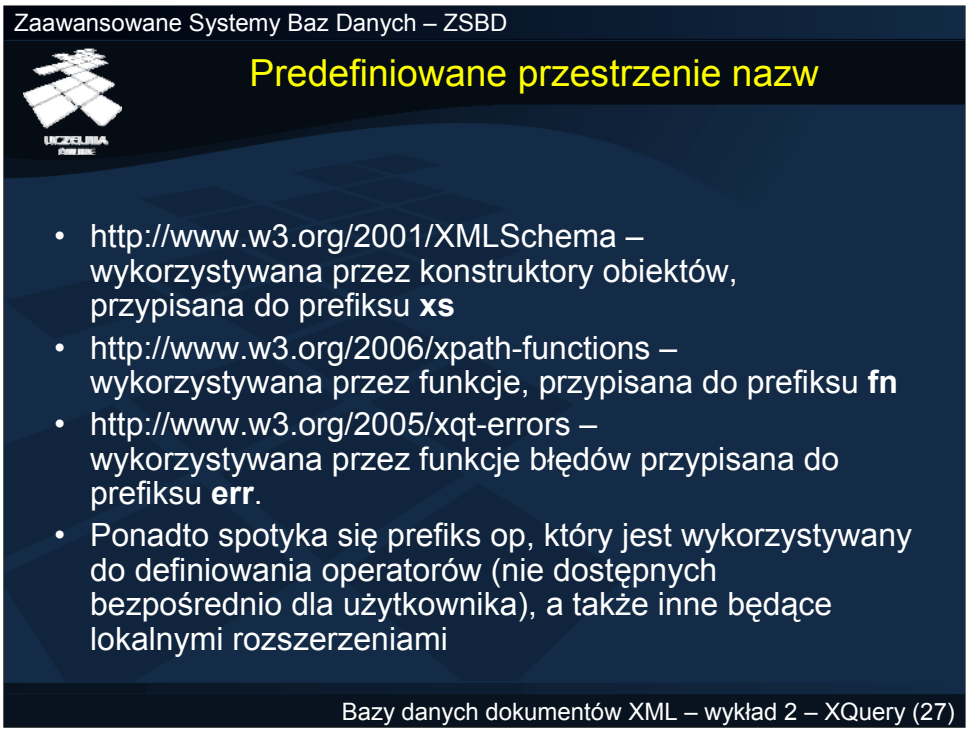
Funkcje lokalne muszą być definiowane i używane w predefiniowanej przestrzeni nazw local.

Funkcje mogą mieć parametry, które mogą być typowane. Nie można przeciążać nazw funkcji użytkownika ani definiować funkcji o zmiennej liczbie parametrów.

Przykładowa definicja funkcji i jej użycie zostało przedstawione na slajdzie.

Funkcja o nazwie liczba_pracowników ma jeden parametr $param. Zgodnie z regułą dotyczącą przestrzeni nazw, funkcja ta została zdefiniowana przy użyciu przestrzeni nazw do której odwołaliśmy się za pomocą prefiksu local. Wynikiem funkcji jest wyrażenie, które obliczy liczbę elementów pracownik w dokumencie pracownicy.xml posiadających podelement id_zesp o wartości równej wartości parametru $param.

Funkcja iczba_pracowników została użyta w zapytaniu do wygenerowania zawartości elementu liczba_prac.



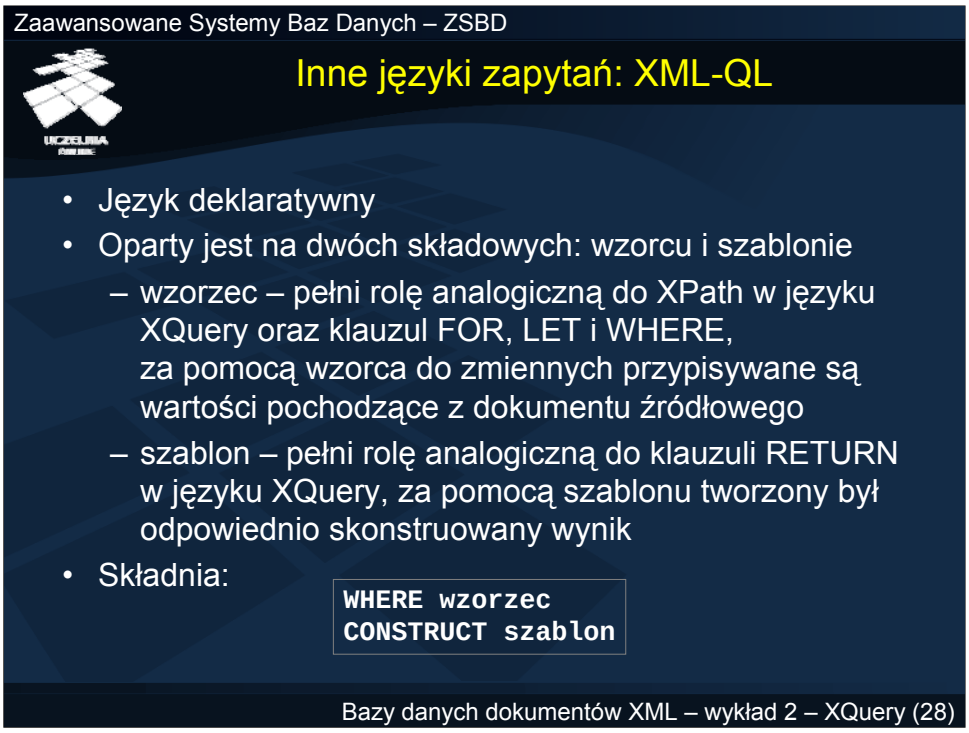
Oprócz przestrzeni nazw o prefiksie lokal w XQuery istnieje jeszcze kilka predefiniowanych przestrzeni nazw.

Przestrzeń nazw *http://www.w3.org/2001/XMLSchema* wykorzystywana jest przez konstruktory obiektów i przypisana została do prefiksu **xs**

Przestrzeń nazw *http://www.w3.org/2006/xpath-functions* wykorzystywana jest przez funkcje wbudowane i przypisana została do prefiksu **fn**

Przestrzeń nazw *http://www.w3.org/2005/xqt-errors* wykorzystywana jest przez funkcje błędów i przypisana została do prefiksu **err**

Ponadto spotyka się prefiks op, który jest wykorzystywany do definiowania operatorów (nie dostępnych bezpośrednio dla użytkownika), a także inne przestrzenie nazw będące lokalnymi rozszerzeniami dodawanymi przez twórców aplikacji w celu rozszerzenia wachlarza dostępnych funkcji. Przykładem może być prefiks ora możliwy do wykorzystania podczas wykonywania zapytań XQuery w bazie danych Oracle.



Przed językiem zapytań XQuery powstało bardzo wiele propozycji i rozwiązań języków zapytań przeznaczonych do przetwarzania baz danych dokumentów XML. Najbardziej znane z nich to XML-QL i Quilt.

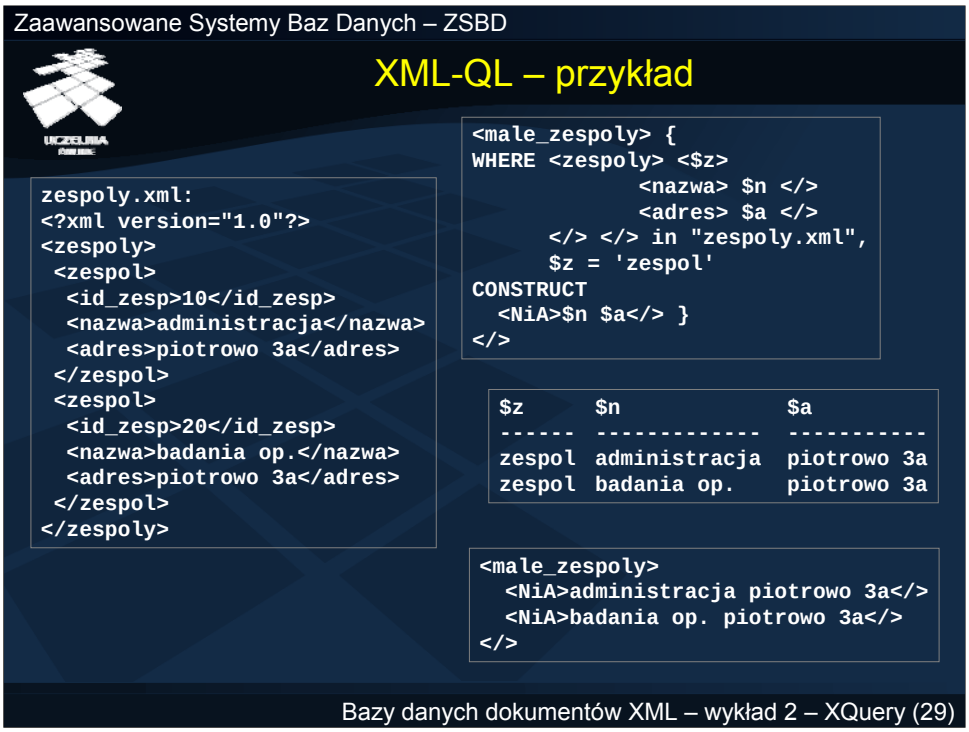
XML-QL jest językiem deklaratywnym, który umożliwia wykonywanie zapytań, konstruowanie, transformację oraz integrację danych XML. XML-QL wspiera zarówno uporządkowany jak i nieuporządkowany widok dokumentu XML. Dodatkowo, XML-QL spełnia większość wymagań nakładanych na języki zapytań dla XML przez konsorcjum W3C (http://www.w3.org/TR/xquery-requirements).

XML-QL oparty jest na dwóch składowych: wzorcu i szablonie.

Wzorzec pełni rolę analogiczną do XPath w języku XQuery oraz klauzul FOR, LET i WHERE, za jego pomocą do zmiennych przypisywane są wartości pochodzące z dokumentu źródłowego.

Szablon pełni rolę analogiczną do klauzuli RETURN w języku XQuery. Za pomocą szablonu XML-QL tworzony jest odpowiednio skonstruowany wynik w postaci dokumentów XML lub ich fragmentów.

Polecenia XML-QL składają się z dwóch klauzul: WHERE, po którym następuje wzorzec dopasowywany do dokumentu źródłowego i CONSTRUCT, po którym następuje szablon tworzący obiekt wynikowy. Na następnym slajdzie zostanie przedstawione przykładowe zapytanie w języku XML-QL.

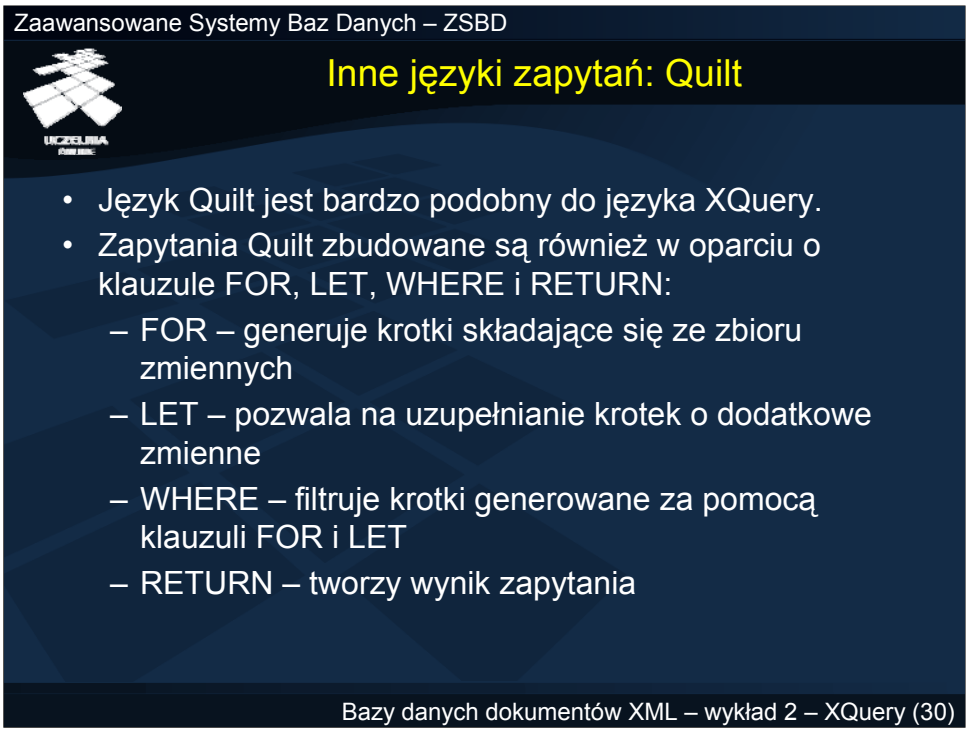


W przykładzie zapytania XML-QL zastosowano podzapytanie. Zostało ono ograniczone nawiasami klamrowymi. Jego zadaniem jest skonstruowanie elementu zewnętrznego male_zespoly (jest tu użyta niejawna klauzula CONSTRUCT). Podzapytanie składa się z wzorca rozpoczynającego się od elementu <zespoly>. Wzorzec składa się z dwóch części rozdzielonych przecinkiem. Zadaniem pierwszej części wzorca jest wielokrotne dopasowanie się do zawartości dokumentu zespoly.xml. W wyniku każdorazowego dopasowania zmienne $z, $n i $a uzyskują odpowiednie wartości. Jednocześnie, w wyniku każdego dopasowania, tworzona jest kolejna krotka podlegająca przetwarzaniu. Przy założeniu, że dokument zespoly.xml został przedstawiony na slajdzie, wzorzec użyty w zapytaniu utworzy 2 krotki zawierające każdorazowo trzy zmienne. Wartości zmiennych w kolejnych krotkach zostały przedstawione na slajdzie. Druga część wzorca sprawdza czy zawartość zmiennej $z to "zespol". Ten fragment wzorca funkcjonuje analogicznie do klauzuli WHERE w XQuery. Krotki wygenerowane za pomocą klauzuli WHERE zostają następnie przetwarzane za pomocą klauzuli CONSTRUCT. W naszym przykładzie tworzy ona elementy <NiA>, których zawartość zostanie zbudowana w oparciu o wartości zmiennych $n i $a.

Zwróćmy uwagę, że wzorzec wykorzystywany przez język XML-QL odpowiada klauzulom FOR i LET oraz wyrażeniom XPath używanym w języku XQuery. Sposób definiowania wzorca powoduje, że zwykle przyjmuje on duże rozmiary, ponadto nie jest on tak funkcjonalny jak wyrażenia XPath.

Podstawowymi wadami XML-QL są:

- brak możliwości korzystania z wyrażeń XPath i funkcji XPath
- brak obsługi przestrzeni nazw
- brak funkcji grupowych



Język Quilt jest bardzo podobny do języka XQuery. Wynika to z faktu iż XQuery jest następcą języka Quilt, a przy jego tworzeniu brały udział między innymi te same osoby.

Zapytania języka Quilt składają się z analogicznych klauzul.

FOR – generuje krotki składające się ze zbioru zmiennych.

LET – pozwala na uzupełnianie krotek o dodatkowe zmienne.

WHERE – filtruje krotki generowane za pomocą klauzuli FOR i LET.

RETURN – tworzy wynik zapytania.

Twórcy języka XQuery w dokumencie dotyczącym języka XQuery napisali:

XQuery wywodzi się z języka zapytań przeznaczonego do przetwarzania dokumentów XML o nazwie Quilt. Z kolei język Quilt zapożyczył swoje własności z kilku innych języków zapytań.

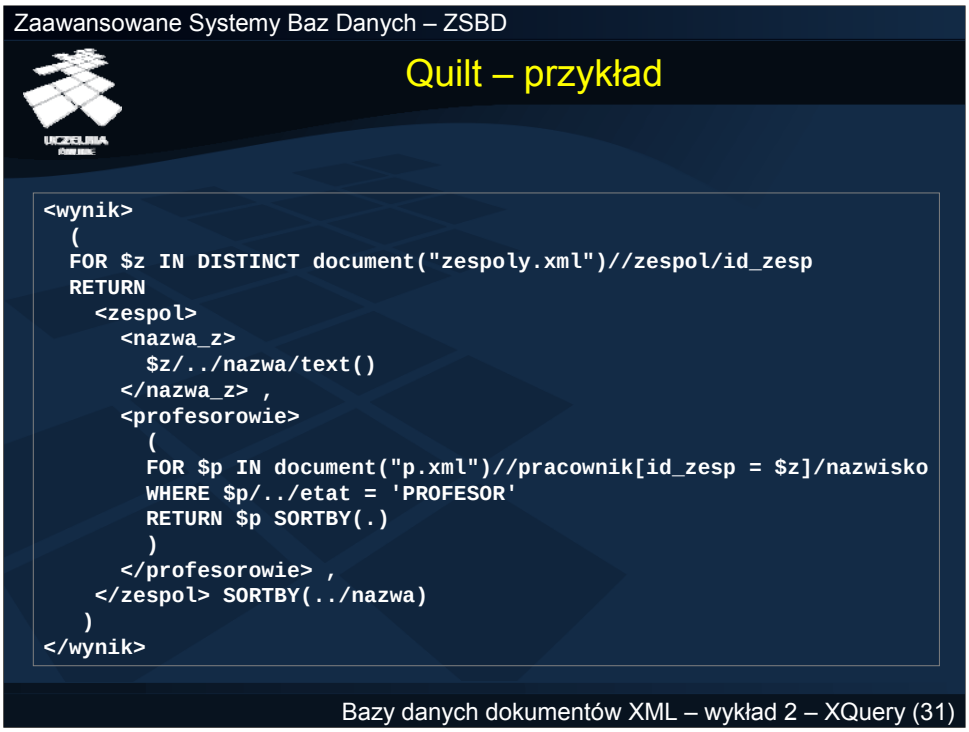
Ze standardu XPath oraz języka XQL zapożyczone zostały wyrażenia ścieżkowe właściwe dla dokumentów hierarchicznych.

Z języka XML-QL zapożyczone zostało pojęcie zmiennych wiązanych oraz wykorzystanie zmiennych do tworzenia nowych struktur.

Z języka SQL zapożyczona została idea zestawu klauzul opartych na słowach kluczowych, które udostępniają mechanizmy do transformacji danych.

Z języka OQL zapożyczone zostało pojęcie języka funkcjonalnego złożonego z wielu różnych rodzajów wyrażeń, które mogą być wielokrotnie zagnieżdżane z zachowaniem ogólności.

Ponadto na język Quilt miały wpływ także inne języki do przetwarzania dokumentów XML jakie jak Lorel i YATL.



Już na pierwszy rzut oka widać, że Quilt jest bardzo podobny do języka XQuery.

Powyższe zapytanie we wnętrzu elementu <wynik> generuje elementy <zespol> powstałe w oparciu o zawartość dokumentu "zespoly.xml".

Każdy element <zespol> zawiera element <nazwa_z> oraz element <profesorowie>, którego zawartość wynika z zawartości dokumentu pracownicy.xml.

Klauzule FOR, LET, WHERE i RETURN pełnią w powyższym zapytaniu role analogiczne jak w poleceniach XQuery.

Bibliografia

- Pełna specyfikacja XQuery http://www.w3.org/TR/xquery/
- Definicje funkcji i operatorów http://www.w3.org/TR/xquery-operators/
- Przykładowe edytory otwarte na XQuery
  - Oracle JDeveloper www.oracle.com/technology/products/jdev/
  - Altova XMLSpy www.altova.com