Uniwersytet Mikołaja Kopernika
Wydział Matematyki i Informatyki

Piotr Wicenty
nr albumu: 187464

Praca magisterska
na kierunku Informatyka

# Natywny typ XML w relacyjnych bazach danych

Opiekun pracy dyplomowej
dr hab. Krzysztof Stencel, prof. UMK
Zakład Baz Danych

Toruń 2010

# Spis treści

# Rozdział I
# Wprowadzenie

# 1. Wprowadzenie do XML

## 1.1. Czym jest XML

Rozszerzalny język znaczników, czyli XML (ang. *Extensible Markup Language*) to prosty, szybki i elastyczny format tekstowy wywodzący się z SGML (ang. *Standard Generalized Markup Language*). XML jest otwartym, wolnym od opłat licencyjnych standardem rekomendowanym przez organizację World Wide Web Consortium (W3C)[1] . Mimo swojej nazwy nie jest językiem lecz raczej metajęzykiem – służącym do konstruowania innych języków oraz opisującym zasady ich tworzenia. Został zaprojektowany jako niezależny od platformy format przechowywania, wymiany oraz reprezentacji danych.

## 1.2. Rozwój i znaczenie formatu XML

Początki XML sięgają roku 1996. Wtedy to pojawiła się pierwsza, robocza wersja specyfikacji. Prace nad nią trwały jeszcze 2 lata, aż w końcu, 10 lutego 1998 roku XML w wersji 1.0 został oficjalnie zarekomendowany przez organizację W3C. Od tego czasu jego znaczenie stale rośnie - technologia XML stała się wszechobecna. Ze względu na swoją wszechstronność i zdolność wymiany danych między różnymi systemami, aplikacjami oraz urządzeniami znajduje zastosowanie w niemal każdej branży. XML dzięki swojej samoopisującej budowie oraz zdolności przechowywania danych o różnej strukturze stał się uniwersalnym standardem wymiany informacji.

Wraz z rozwojem Internetu nastąpił również wyraźny wzrost znaczenia XML. Stał się on podstawowym formatem wymiany bądź opisu danych dla wielu nowych języków, protokołów oraz technologii internetowych, takich jak AJAX, XHTML, RSS, XUL, SOAP, WSDL.

O tym jak elastycznym i uniwersalnym formatem danych jest XML może świadczyć choćby fakt pojawienia się w ostatnich latach bardzo wielu różnorodnych aplikacji języka XML. Niemal w każdej dziedzinie nauki znajdziemy aplikację opisującą właściwe dla niej dane. Wśród najpopularniejszych wymienia się VoiceXML, MathXML, XBRL, FpML, HL7 DockBook, BPEL, NewsML. Wzrost popularności XML spowodował, że na rynku pojawiło

1 World Wide Web Consortium (W3C) – Organizacja pracująca nad rozwojem standardów sieci Internet. Oficjalna witryna organizacji http://www.w3.org

się wiele nowych narzędzi wspomagających pracę z tym formatem danych. Oprócz dużych pakietów narzędziowych takich jak Altova XML Spy[2], Liquid XML Studio[3] czy Stylus Studio[4] dostępnych jest także szereg darmowych programów pozwalających na różny sposób przetwarzać dokumenty XML.

Szybko rosnąca popularność XML wymusiła od czołowych producentów systemów zarządzania bazami danych (SZDB) zaaplikowanie do swoich produktów mechanizmów wspierających ten język. Początkowo wsparcie to miało postać funkcji wbudowanych, później pojawił się również natywny typ XML pozwalający na przechowywanie i przetwarzanie w relacyjnych bazach danych dokumentów XML.

## 1.3. Tradycyjne metody zarządzanie dokumentami XML

Zanim SZBD wzbogacone zostały o typ XML zarządzanie danymi w tym formacie było znacznie utrudnione. Brak natywnego typu oznacza konieczność przechowywania dokumentów poza relacyjną bazą danych. Można to robić na kilka sposobów – na przykład umieszczając je w systemie plików. Rozwiązanie to jednak cechuje szereg istotnych wad. Przede wszystkim jest mało efektywne, gdy mamy do czynienia z dużą liczbą dokumentów, nie daje możliwości równoległego dostępu do danych oraz wygodnego zarządzania ich bezpieczeństwem.

Innym sposobem jest wykorzystanie natywnej bazy danych XML – przeznaczonej do przechowywania danych w tym formacie. W ostatnich latach pojawiło się wiele implementacji i bazy takie zdobywają coraz większą popularność. Zalet płynących z zastosowania takich rozwiązań jest kilka. Bazy XML-owe są zoptymalizowane pod obsługę danych o strukturze XML i dobrze radzą sobie z dużą liczbą dokumentów. Są jednak wciąż młode i w dużych projektach informatycznych często nie znajdują zastosowania ze względu na wątpliwą stabilność i małe w porównaniu z relacyjnymi bazami doświadczenie programistów w pracy z takimi systemami.

Jeśli potraktować dokument XML jak zwykły plik, wówczas jego przechowywanie nie stanowi nowego wyzwania dla bazy danych. Można go umieścić w tabeli z kolumną typu znakowego (VARCHAR, TEXT) bądź typu obiekt (BLOB, CLOB). Rozwiązanie jest

2 Altova XML Spy – oficjalna witryna produktu: http://www.altova.com

3 Liquid XML Studio – oficjalna witryna produktu: http://www.liquid-technologies.com

4 Stylus Studio – oficjalna witryna produktu: http://www.stylusstudio.com

właściwe i skuteczne tak długo jak zależy nam jedynie na przechowywaniu i odzyskiwaniu całości danych XML. Uniemożliwia jednak efektywne odpytywanie bazy danych o zawartość dokumentu, wyświetlanie jego fragmentów, konkretnych elementów czy atrybutów.

Innym dość często wykorzystywanym mechanizmem przechowywania danych XML jest ich dekompozycja i umieszczenie w relacyjnych strukturach bazy danych. Dokument jest dzielony na elementy oraz atrybuty a następnie umieszczany w wielu oddzielnych tabelach i kolumnach. Odtwarzanie takiego dokumentu wymaga wówczas wielu złączeń tabel i dość skomplikowanych zapytań. Takie podejście pozbawia dodatkowo dane XML ich wrodzonej elastyczności i powoduje, że dokumenty stają się nieczytelne.

Wszystkie wymienione powyżej rozwiązania posiadają istotne wady. Dlatego też uzasadnione i konieczne stało się rozszerzenie relacyjnych systemów baz danych o nowy typ: natywny XML.

# 2. Sprzęt i oprogramowanie wykorzystane w pracy

## 1.1. Systemy zarządzania bazami danych opisane w pracy

W pracy wykorzystano najpopularniejsze i najbardziej zaawansowane systemy zarządzania relacyjnymi bazami danych: system firmy IBM – DB2, Mirfosoft – SQL Server, Oracle Database firmy Oralce oraz przedstawiciela oprogramowania Open Source – system PostgreSQL. W pracy wykorzystano oprogramowanie w wersji jak na poniższej liście:

- IBM DB2 Express-C 9.7 (9.7.100.177), wersja dla Microsoft Windows (32-bit)
- Oracle Database 11g Release 1 (11.1.0.6.0) Standard Edition, wersja dla Microsoft Windows (32-bit)
- Microsoft SQL Server 2008 Express (10.0.25.31)
- PostgreSQL 8.4 (8.4.2), wersja dla Microsoft Windows (32-bit)

## 1.2. Pozostałe oprogramowanie wykorzystane w pracy

- IBM DB2 Control Center
- IBM DB2 Command Editor
- IBM DB2 Activity Monitor
- Oracle SQL Developer
- Oracle SQL Plus
- Oracle Database Manager
- Microsoft SQL Server Management Studio
- Microsoft SQL Server Profiler
- Microsoft Visual Studio
- pgAdmin

- SQL Shell (psql)
- Altova XML Spy

Powyższa lista zawiera jedynie najważniejsze i najczęściej wykorzystywane narzędzia w niniejszej pracy. Na liście pominięto oprogramowanie używane do celów niezwiązanych bezpośrednio z przedmiotem pracy. Oprócz wymienionego oprogramowania wykorzystano aplikacje oraz skrypty stworzone przez autora niniejszej pracy w celu przygotowania danych testowych, przygotowania skryptów testowych, automatyzacji procesu testowania, przetwarzania wyników testów itp.

## 1.3. Konfiguracja sprzętu wykorzystanego w pracy

| | |
|---|---|
| System operacyjny: | Microsoft Windows XP (5.1.2600) SP3 |
| CPU: | Intel Core2 Duo CPU E7400 @ 2.80 GHz 2.80 GHz |
| Dysk fizyczny: | Western Digital Caviar SATA 500 GB / 32MB Cache |
| Pamięć RAM: | DDR2 2048 MB Kingston |

## 3. Cel pracy

Intencją autora pracy jest nakreślenie zakresu i sposobu, w jaki poszczególne systemy zarządzania bazami danych obsługują format XML. W pracy wskazane zostaną możliwości każdego z SZBD, w szczególności opisane zostaną metody tworzenia i importowania danych typu XML oraz sposób ich składowania. Opisane zostaną funkcje pozwalające na dostęp do danych XML za pośrednictwem zapytań standardu XPath oraz XQuery, ich modyfikację, a także integrację danych XML z danymi składowanymi w postaci krotek tabel relacyjnych. Wskazane zostaną możliwości standardu SQL/XML oraz omówione będą różnice w implementacji wymienionych standardów.

Celem niniejszej pracy jest również zbadanie wydajności zapytań XPath/XQuery w zależności od struktury danych oraz wskazanie przypadków, w których wykorzystanie typu danych XML ma uzasadnienie. Oprócz tego, zamysłem autora jest omówienie indeksów XML, wskazanie ich budowy w poszczególnych SZBD, sposobu tworzenia, korzyści oraz kosztów jakie niesie ze sobą ich stosowanie.

# Rozdział II
# XML w relacyjnych bazach danych

# 1. Systemy zarządzania bazami danych wspierające format XML

## 1.4. IBM DB2

Rosnąca popularność standardu XML skłoniła firmę IBM do podjęcia prac nad wprowadzeniem do swego produktu mechanizmów pozwalających obsłużyć nowy typ danych. Pięć lat pracy kilkuset deweloperów, architektów oraz inżynierów przyniosło efekt w postaci pierwszego na rynku hybrydowego serwera bazy danych. Nowy produkt – IBM DB2 w wersji 9 ujrzał światło dzienne w lipcu 2006 roku. Spośród innych wyróżniało go to, że pozwalał na efektywne przechowywanie zarówno danych relacyjnych jak i hierarchicznych. Podczas, gdy inne serwery baz danych oraz wcześniejsze wersje DB2 umożliwiały przechowywanie dokumentów XML jedynie w strukturach relacyjnych, IBM zaproponował rozwiązanie pozwalające na umieszczanie danych również w zhierarchizowanej formie w postaci drzew i przetwarzanie zapytań zarówno w postaci SQL/XML[5] jak i w języku XQuery. Nowa technologia zyskała nazwę *pureXML*.

W 2009 roku IBM wydał kolejną, ostatnią jak dotąd stabilną wersję produktu - DB2 9.7 zawierającą szereg poprawek oraz usprawnień w stosunku do poprzedniczki. IBM DB2 z technologią *pureXML* może pracować pod kontrolą systemów operacyjnych z rodziny Linux, Windows, Solaris, AIX oraz HP-UX.

## 1.5. Microsoft SQL Server

Microsoft pierwsze usprawnienia dla XML wprowadził w SQL Server 2000. Polegały one na dodaniu do słownika serwera słów kluczowych FOR XML oraz OPENXML. FOR XML jest rozszerzeniem w składni SELECT pozwalającym na zwracanie wyników zapytania w postaci strumienia XML. Funkcja OPENXML działa w przeciwny sposób – wynik zapytania dla otrzymanego dokumentu XML przedstawia w standardowym, wierszowym zestawieniu.

Przykład:

5 SQL/XML jest rozszerzeniem SQL i stanowi część specyfikacji ANSI/ISO SQL 2003 (formalnie: ISO/ IEC 9075-14:2003)

```
SELECT ID, Nazwa
FROM Produkty Produkt
FOR XML AUTO
```

```
<Produkt ID=”1” Nazwa=”szafa”/>
<Produkt ID=”2” Nazwa=”komoda”/>
```

*Przykład 1: Zapytanie SQL oraz rezultat w formacie XML, serwer SQL Server.*

Przykład:

```
<Zamowienie ZamowienieID = "65002">
    <Produkt ProduktID="1" Liczba="3"/>
    <Produkt ProduktID="2" Liczba="16"/>
</Zamowienie>
```

```
SELECT * FROM
OPENXML (@xmlDoc, 'Zamowienie/Produkt', 1)
WITH
(ZamowienieID integer '../@ZamowienieID', ProduktID integer, Liczba integer)
```

| ZamowienieID | ProduktID | Liczba |
|---|---|---|
| 65002 | 1 | 3 |
| 65002 | 2 | 16 |

*Przykład 2: Zapytanie danych XML zwracające wynik w postaci SQL, serwer SQL Server.*

W SQL Server 2005 wprowadzono szereg poprawek oraz usprawnień. Nowa wersja umożliwiała m. in. zagnieżdżanie FOR XML, dzięki czemu zapytania mogły zwracać bardziej złożone wyniki. Największą nowością w wersji 2005 był jednak nowy typ danych – natywny XML pozwalający na tworzenie zmiennych oraz kolumn dla danych XML. Wraz z nowym typem pojawiało się również repozytorium dokumentów XSD oraz funkcja walidacji. Dodano obsługę zapytań XQuery oraz indeksowanie XML.

Przykład:

```
CREATE TABLE Zamowienia (
    Id INT PRIMARY KEY,
    Data DATETIME,
    KlientID INT,
    Uwagi XML (UwagiZamowieniaXSD))
```

*Przykład 3: Tworzenie tabeli z kolumną typu XML z nałożonym schematem XSD.*

SQL Server 2008 oferuje dodatkowe możliwości walidacji XSD, rozbudowane wsparcie dla zapytań XQuery oraz rozszerzoną funkcjonalność ułatwiającą operacje modyfikacji danych XML. Poza tym, bazuje głównie na rozwiązaniach zaproponowanych wersji 2005.

## 1.6. Oracle Database

Oracle jako pierwszy wprowadził do swoich produktów mechanizmy pozwalające na pracę z danymi XML. W wydanej w 1999 roku wersji 8i pojawiła się możliwość generowania XML bezpośrednio z bazy danych przy pomocy specjalnych zapytań, a także wypełniania bazy na podstawie zawartości dokumentu XML. Oracle dostarczył również XDK (Oracle XML Developer Kit ) zawierające m. in. parsery pozwalające na wykorzystanie XML w aplikacjach napisanych w języku PL/SQL[6], Java oraz C/C++.

Wśród komponentów Oracle Database 9i pojawił się XMLType – typ danych do przechowywania dokumentów XML, a wraz z nim szereg nowych funkcji SQL do ich generowania i przetwarzania. XMLType powstał na bazie typu dużych obiektów znakowych (CLOB). Kolejne odsłony – wersje 9.2 oraz 10.1 rozbudowują możliwości bazy w zakresie generowania, modyfikacji, importu oraz eksportu danych XML.

Obecnie Oracle Database pozwala na przechowywanie dokumentów XML w modelu obiektowo-relacyjnym, w postaci obiektów znakowych oraz w postaci binarnej. Umożliwia ich indeksowanie, przeszukiwanie i modyfikację. Zapewnia również szereg funkcjonalności związanych z transformacją i walidacją danych XML.

Zbiór technologii Oracle Database związanych z XML nosi nazwę *Oracle XML DB*.

6 PL/SQL – proceduralny język zapytań baz danych Oracle stanowiący rozszerzenie języka SQL.

## 1.7. PostgreSQL

Twórcy serwera PostgreSQL nieustannie pracują by utrzymać konkurencyjność swojego produktu w stosunku do komercyjnych rozwiązań takich firm jak IBM, Microsoft czy Oracle. Co kilka miesięcy wydawane są kolejne wersje systemu a jego możliwości stale rosną. Jak wyszczególniono w [7] wciąż jednak istnieje spory wachlarz funkcjonalności, które oferują już bazy DB2 czy SQL Server a których nie posiada PostgreSQL. Podobnie rzecz się ma jeśli chodzi o oferowane przez ten system wsparcie dla danych XML.

Pierwszy, znaczący pakiet funkcji XML pojawił się w wydanej w grudniu 2006 roku wersji oznaczonej numerem 8.2. Wówczas serwer oferował m. in. walidację poprawności dokumentu, proste indeksowanie, przeszukiwanie za pomocą XPath. Dokumenty przechowywane były jednak wciąż jako dane typu tekst. Znaczące zmiany nastąpiły wraz z pojawieniem się w lutym 2008 roku wersji 8.3. Najważniejsze z nich to implementacja wsparcia dla standardu SQL/XML oraz dodanie nowego typu danych – XML. Choć obecnie PostgreSQL zawiera całkiem pokaźny zestaw narzędzi do pracy z danymi XML to twórcy serwera wciąż pracują nad kolejnymi. Wśród najważniejszych celów dla kolejnych wersji wymienia się implementację walidacji zgodności dokumentów z XSD, usprawnienie działania zapytań XPath, rozbudowa mechanizmu indeksowania, dostosowanie niektórych funkcji do standardu SQL/XML:2006[7] oraz implementacja obsługi zapytań XQuery.

## 1.8. Pozostałe

Możliwość operowania danymi XML nie jest wyłącznie domeną omawianych w tej pracy SZBZ. Pewne możliwości w tej dziedzinie posiadają także np. bazy danych Sybase ASE[8], FrontBase[9] czy MySQL[10]. Cieszący się dużą popularnością serwer MySQL oferuje jednak tylko funkcje do odpytywania oraz modyfikacji łańcucha znaków reprezentującego dokument XML. Wykorzystuje przy tym język XPath do nawigacji po dokumencie. Funkcje dostępne są od wersji 5.1.

7 SQL/XML:2006 jest nowszą wersją standardu SQL/XML:2003, formalnie ISO/ IEC 9075-14:2006

8 Sybase ASE – produkt firmy Sybase Inc. Oficjalna witryna firmy: www.sybase.com

9 FrontBase – relacyjny SZBD, oficjalna witryna projektu: www.frontbase.com

10 MySQL – SZBD dostępny na licencji GPL oraz komercyjnej. Oficjalna witryna projektu: www.mysql.com

Przykład:

```
<Pracownik>
    <Imie>Tomasz</Imie>
    <Imie>Krzysztof</Imie>
    <Nazwisko>Nowak</Nazwisko>
</Pracownik>
```

**SELECT** ExtractValue(@xmlDoc, '//Imie[1]')

| Imie |
|---|
| Tomasz |
| Krzysztof |

**SELECT UPDATEXML** (@xmlDoc, '/Pracownik/Nazwisko', '<ID>11093</ID>') **AS** NowyXML

NowyXML: '<Pracownik><Imie>Tomasz</Imie><Imie>Krzysztof</Imie><ID>11093</ID></Pracownik>'

*Przykład 4: Wykorzystanie funkcji ExtractValue oraz UpdateXML, serwer MySQL.*

Znacznie więcej możliwości daje serwer Sybase ASE – w wersji 15.0 zaimplementowano m. in. obsługę zapytań XPath i XQuery, indeksowanie dokumentów XML oraz konwersję danych relacyjnych do XML. Jednak podobnie jak w przypadku MySQL dane XML są reprezentowane przez dane typu tekstowego.

Inne SZBD w większości nie posiadają wsparcia dla danych XML lub wspierają je w bardzo ograniczonym stopniu. Twórcy niektórych z nich – jak choćby serwera Firebird[11] zapowiedzieli dopiero implementację wybranych funkcji w kolejnych wersjach.

11 Firebird – wolnodostępny serwer relacyjnych baz danych. Oficjalna witryna projektu: www.firebirdsql.org

# 2. Architektura danych XML w SZBD

## 2.1. Reprezentacja danych typu XML

Dane XML mogą być reprezentowane na kilka sposobów. W każdym z omawianych SZBD implementacja typu XML jest inna, ale można podzielić je na dwie grupy. Pierwszą z nich stanowią implementacje oparte na danych znakowych: VARCHAR, CLOB itp. W rozwiązaniu tym typ XML jest szczególnym przypadkiem typu znakowego – na typ znakowy nałożona jest dodatkowo kontrola poprawności składni XML. Dane przechowywane w bazie są wówczas dokładną kopią danych wprowadzonych do bazy – zachowane są m. in. białe znaki, kolejność atrybutów, pamiętany jest sposób domknięcia znaczników itp. W niektórych przypadkach może być to zaleta – np. gdy zależy nam na zachowaniu dokumentu (licencji, umowy) w oryginalnej formie. Rozwiązanie takie zastosowano m. in. w systemie PostgreSQL.

Reprezentacja w postaci tekstu niesie jednak za sobą wiele istotnych wad – o ile operacje wstawiania czy usuwania całych dokumentów są dość efektywne i proste o tyle manipulacja częścią dokumentu jest już znacznie utrudniona. W takim modelu przede wszystkim nie ma możliwości efektywnego indeksowania zawartości, co uniemożliwia sprawne przeszukiwanie i modyfikację wybranych fragmentów dokumentu. Z punktu widzenia składu danych typ XML w serwerze PostgreSQL nie różni się od typów znakowych. W serwerze tym typ XML, podobnie jak inne mogące służyć do przechowywania znacznych porcji danych, podlega mechanizmowi o nazwie TOAST (ang. *The Oversized-Arrtibute Storage Technique*). Zgodnie z [6] mechanizm polega na kompresji oraz dzieleniu porcji danych w przypadku, gdy ta przekracza rozmiar strony (zwykle 8 kB). Stąd też, dokumenty XML o znacznych rozmiarach, przed umieszczeniem w bazie danych są kompresowane, a następnie w razie konieczności dzielne na kawałki o wielkości rozmiaru strony. Każdy z takich kawałków posiada 32 bitowy nagłówek, z którego 2 bity zarezerwowane są na potrzeby mechanizmu TOAST, natomiast kolejne 30 bitów stanowi całkowitą długość danych wyrażoną w bajtach. Stąd też ograniczenie na wielkość danych XML w serwerze PostgreSQL wynosi 1 GB ($2^{30}$ – 1 bajtów). Cały proces przebiega niezauważalnie dla użytkownika systemu. Zastosowanie takiego mechanizmu ma swoje wady i zalety. Z jednej strony znacznie redukuje rozmiar danych XML w bazie danych. Wynika to z faktu, że dane XML reprezentowane są w postaci tekstu, dają się zatem dobrze kompresować. Wadą

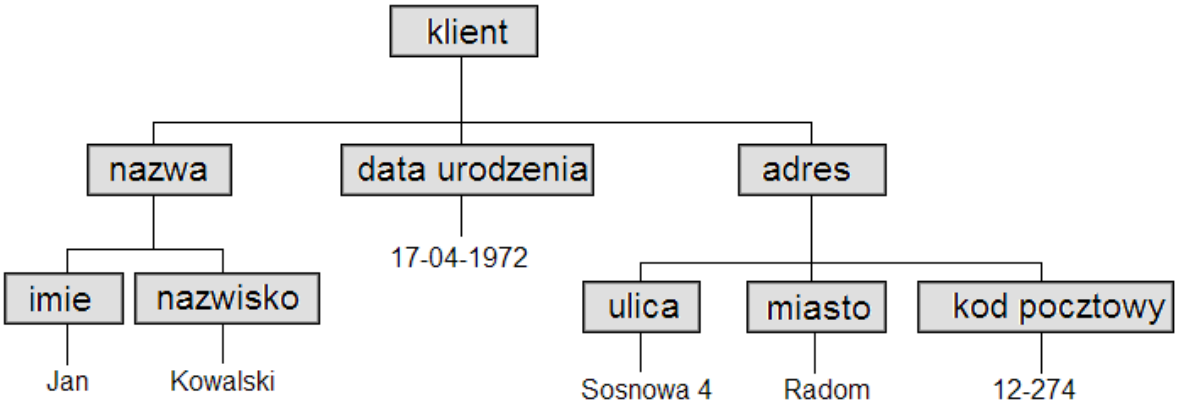
mechanizmu jest wydłużony czas pracy z danymi, które każdorazowo trzeba dekompresować oraz łączyć w całość. Wyniki testów własnych dotyczących szybkości oraz stopnia kompresji danych zaprezentowano w kolejnych podrozdziałach.

Całkiem inne rozwiązanie zastosowano w serwerze SQL Server. system ten jest „świadomy" hierarchicznej struktury dokumentów i dane typu XML przechowuje w hierarchicznej formie. W tym celu wykorzystywany jest standard XML Information Set[12], będący zaleceniem konsorcjum W3C. Standard opisuje abstrakcyjną reprezentację dokumentu i stanowi próbę zdefiniowania, jakie informacje w dokumencie XML są informacjami znaczącymi.

Przykład:

```
<klient>
    <nazwa>
        <imie>Jan</imie>
        <nazwisko>Kowalski</nazwisko>
    </nazwa>
    <data urodzenia>17-04-1972</data urodzenia>
    <adres>
        <ulica>Sosnowa 4</ulica>
        <miasto>Radom</miasto>
        <kod pocztowy>12-274</kod pocztowy>
    </adres>
</klient>
```

12 XML Information Set (XML InfoSet) – standard organizacji W3C opisujący abstrakcyjną reprezentację dokumentu XML. Oficjalna witryna standardu: http://www.w3.org/TR/xml-infoset




*Schemat 1: Dokument XML w hierarchicznej formie.*

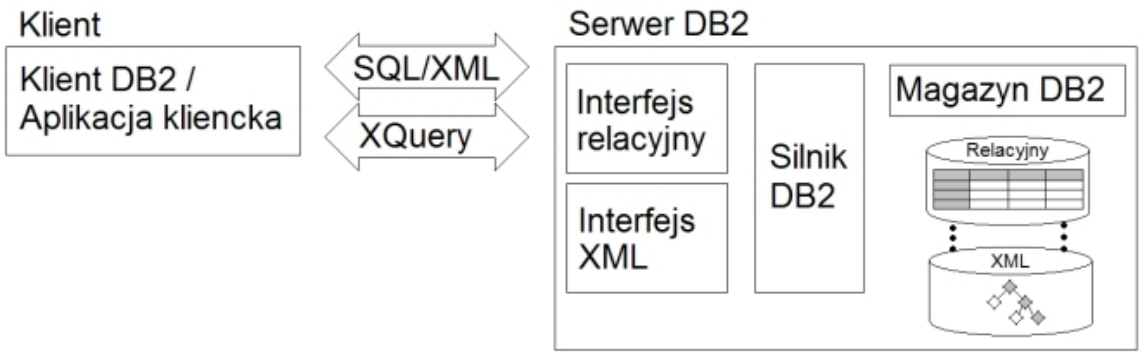
Przed zasileniem bazy danych dokument jest parsowany pod kątem poprawności składni XML a następnie przekształcany do wewnętrznej reprezentacji serwera. W tym przypadku zawartość bazy danych nie jest dokładną kopią danych źródłowych. W bazie danych przechowywana jest struktura drzewiasta, a zatem niektóre, nieistotne informacje o dokumencie zostają utracone. W szczególności nie jest rozróżniany sposób domknięcia pustego elementu (<element></element> jest tym samym co <element/>) oraz pomijane są niektóre białe znaki. Zbiór informacji nieistotnych z punktu widzenia reprezentacji danych XML w postaci hierarchicznej opisany jest przez standard XML InfoSet.

Przykład:

```
INSERT INTO docs VALUES
('<?xml version="1.0"?><magazyn>  <towar id="1"></towar>  <towar id="2"/>  </magazyn>' )

SELECT doc FROM docs
doc
----------------------
<magazyn>
 <towar id="1" />
 <towar id="2" />
</magazyn>
```

*Przykład 5: Zasilenie oraz odtworzenie dokumentu XML na podstawie danych z InfoSet, serwer SQL Server.*

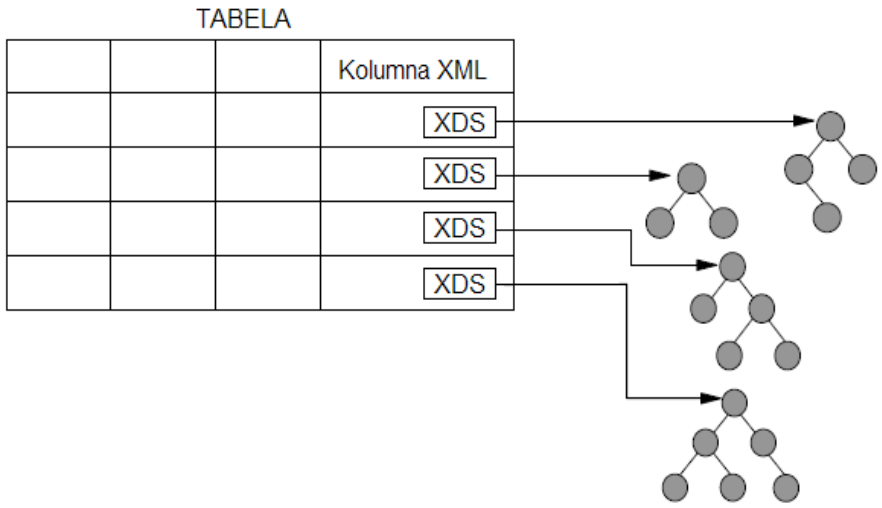
Podobne, choć nieco inne rozwiązania zastosowano w serwerze firmy IBM. DB2 łączy w sobie mechanizmy zarządzania danymi relacyjnymi oraz danymi XML. Dane te składowane i przetwarzane są osobno – stąd też DB2 nazywany jest serwerem hybrydowym. Z punktu widzenia aplikacji klienckich dostęp do danych jest jednakowy – to serwer zarządza i integruje dane XML z danymi relacyjnymi.




*Ilustracja 1: Uproszczona architektura serwera IBM DB2 9.*

Podobnie jak SQL Server także DB2 podczas wprowadzania danych do bazy parsuje

je a następnie przekształca do wewnętrznej, hierarchicznej postaci. Oznacza to, że każdy dokument musi być poprawnie sformatowany zanim zostanie umieszczony w bazie. Każda próba zaimportowania lub zasilenia tabeli danymi, które nie stanowią poprawnego dokumentu XML zakończy się błędem. W przypadku powodzenia każdy wiersz tabeli w kolumnie typu XML zawiera jedynie wskaźnik XDS[13] – właściwe dane składowane są w odrębnych strukturach bazy.




*Ilustracja 2: Relacje pomiędzy kolumną typu XML a danymi XML (model uproszczony), serwer DB2.*

Serwer Oracle oferuje typ XMLType, który daje możliwość składowania danych na dwa sposoby. W zależności od wybranej opcji dokument XML może być reprezentowany w bazie danych jako duży obiekt znakowy (CLOB) lub w postaci binarnej. Wybór pierwszej z opcji upodabnia XMLType do implementacji typu XML serwera PostgreSQL – XMLType jest wówczas typem znakowym z nałożoną kontrolą poprawności dokumentu a zawartość bazy danych jest kopią źródłowego ciągu znaków. Druga opcja pozwala na składowanie danych XML w wewnętrznych, dedykowanych strukturach – podobnie jak ma to miejsce w serwerze DB2. O tym w jaki sposób składować dane XML użytkownik musi zdecydować w momencie tworzenia tabeli z kolumną XMLType (domyślnie wykorzystywany jest typ znakowy).

Przykład:

13 XDS (XML Data Specifier) – wskaźnik zawierający informacje pozwalające na ustalenie lokalizacji danych XML na dysku

```
CREATE TABLE XmlDoc (id INTEGER, doc XMLTYPE)
    XMLTYPE COLUMN doc STORE AS CLOB;

CREATE TABLE XmlDoc (id INTEGER, doc XMLTYPE)
    XMLTYPE COLUMN doc STORE AS BINARY XML;
```

*Przykład 6: Tworzenie tabeli z kolumną typu XMLType w postaci tekstowej oraz binarnej, serwer Oracle.*

Typ binarny jest jedną z nowości systemu Oracle 11g. Jedną z jego głównych zalet jest sposób w jaki dane reprezentowane są w systemie. Dotychczas dane XML wymagały parsowania oraz serializacji za każdym razem, gdy były wymieniane pomiędzy różnymi warstwami aplikacji lub przed zapisem na dysk. Binarny XMLType stanowi reprezentację jednakową dla danych składowanych na dysku, w pamięci systemu czy wykorzystywaną w trakcie transferu danych. Dzięki takiemu ujednoliceniu formatu zredukowano narzut na pracę procesora oraz wymaganą pamięć.

Zgodnie z [5] przekształcanie danych XML do binarnego typu XMLType polega m. in. na rozgraniczeniu znaczników (podział znaczników, grupowanie oraz odwzorowanie na identyfikatory). Proces ten tym skuteczniej redukuje rozmiar dokumentu im bardziej powtarzalna jest jego struktura. Ponadto, tekstowe wartości atrybutów i węzłów przekształcane są do natywnej postaci – np. dane całkowitoliczbowe reprezentowane są jako typ INTEGER. Według zapewnień producenta, operacje te znacznie zmniejszają wielkość dokumentów XML w bazie danych (wyniki testów własnych zaprezentowano w kolejnych podrozdziałach).

## 2.2. Generowanie zawartości XML przy pomocy funkcji SQL/XML

SQL/XML jest standardem ANSI oraz ISO opisującym możliwości i sposoby przetwarzania danych XML w bazach danych SQL. Został stworzony na potrzeby integracji danych relacyjnych z danymi XML. Standard definiuje szereg funkcji pozwalających na tworzenie i modyfikację zawartości typu XML. Za ich pomocą możliwa jest m. in. konwersja łańcuchów znaków na typ XML, tworzenie elementów, atrybutów, komentarzy XML, łączenie małych fragmentów danych w większe itp.

Jedną z podstawowych funkcji do tworzenia zawartości XML jest XMLPARSE. Przekształca on dane z postaci tekstowej do natywnego typu XML, pozwala określić czy łańcuch znaków stanowi kompletny dokument XML czy jedynie fragment oraz w jaki sposób traktować białe znaki. Standardowo funkcja ma postać:

**XMLPARSE ( {DOCUMENT | CONTENT} łańcuch_znaków [ <opcje> ])**,
gdzie **<opcje> := { PRESERVE | STRIP } WHITESPACES**

Ze względu na sposób w jaki serwer przechowuje dane XML poszczególne implementacje różnią się. Np. w serwerze PostgreSQL funkcja nie posiada opcji pozwalających na wybór sposobu w jaki postępować z białymi znakami, z kolei wersja Oracle posiada dodatkową flagę 'WELLFORMED' informującą serwer, że dane są poprawnie sformatowane i nie ma potrzeby sprawdzania poprawności składni XML. W serwerze Oracle podobne rozszerzenia standardu SQL/XML zastosowano także w przypadku wielu innych funkcji. Funkcją odwrotną do XMLPARSE jest XMLSERIALIZE.

Przykład:

```
INSERT INTO docs VALUES( Xmlparse( Document
'<?xml version="1.0"?><magazyn>    <towar id="1"></towar>       <towar id="2"/> </magazyn>' ));

SELECT Xmlserialize(Document (SELECT doc FROM docs) AS VARCHAR(100));

                                        xmlserialize
-----------------------------------------------------------------------------------------------------------
<?xml version="1.0"?><magazyn>    <towar id="1"></towar>       <towar id="2"/> </magazyn>
```

*Przykład 7: Wykorzystanie funkcji XMLSERIALIZE do pobrania z dokumentu XML w oryginalnej formie, serwer PostgreSQL.*

Pozostałe funkcje SQL/XML pozwalające na tworzenie zawartości XML to m. in.:

XMLELEMENT - tworzy element XML z nazwą, atrybutami oraz zawartością

XMLATTRIBUTES - tworzy atrybut elementu XML

XMLFOREST - tworzy sekwencję elementów XML

XMLCONCAT - łączy listę pojedynczych wartości XML

XMLCOMMENT - tworzy komentarz XML

XMLPI - tworzy instrukcję przetwarzania XML

XMLAGG - funkcja agregująca, agreguje wartości XML

XMLROOT - tworzy element ROOT w dokumencie XML

Przykład:

```
SELECT Xmlelement(name zamowienie, Xmlattributes(146 as id), Xmlelement(name data, current_date));

                xmlelement
------------------------------------------------------------
<zamowienie id="146"><data>2009-08-11</data></zamowienie>
```

*Przykład 8: Wykorzystanie funkcji XMLELEMENT oraz XMLATTRIBUTES do stworzenia fragmentu dokumentu XML, serwer PostgreSQL.*

Przykład:

```
VALUES( Xmlelement( name "Pracownik", Xmlforest( 'Jan' as "imie", 'Nowicki' as "nazwisko", 'M' as
"plec")));

wynik:
<Pracownik>
    <imie>Jan</imie>
    <nazwisko>Nowicki</nazwisko>
    <plec>M</plec>
</Pracownik>
```

*Przykład 9: Wykorzystanie funkcji XMLFOREST do stworzenia sekwencji elementów, serwer DB2.*

Opisane powyżej funkcje to tylko niektóre spośród zdefiniowanych przez standard ISO/IEC 9075:14. Wyszczególniono głównie te, które przeznaczone są do tworzenia

zawartości typu XML. Są one przydatne w procesie automatycznego generowania dokumentów XML i zwalniają programistę bazy danych z obowiązku wprowadzania danych XML do bazy w postaci tekstu. Ponadto, umożliwiają generowanie XML na podstawie danych relacyjnych z tabel i widoków. Zaimplementowane zostały w serwerach PostgreSQL, Oracle oraz DB2, pominięte natomiast w SQL Server. W przypadku serwera firmy Microsoft większość z opisanej funkcjonalności uzyskać można przy pomocy klauzuli FOR XML (opisanej dokładniej w kolejnych podrozdziałach). Ponadto, zadania realizowane przez niektóre z funkcji SQL/XML są w SQL Server wykonywane niejawnie – tak jest w przypadku operacji parsowania tekstu oraz serializacji danych XML (XMLPARSE oraz XMLSERIALIZE).

Poniżej zaprezentowano zestawienie obrazujące stopień w jakim omawiane SZBD obsługują podstawowe funkcje standardu SQL/XML.

| | **PostgreSQL** | **DB2** | **SQL Server** | **Oracle** |
|---|---|---|---|---|
| XMLPARSE | + | + | - | +* |
| XMLSERIALIZE | ++ | ++ | - | ++ |
| XMLELEMENT | + | ++ | - | +* |
| XMLATTRIBUTES | ++ | ++ | - | ++* |
| XMLFOREST | + | + | - | ++ |
| XMLCONCAT | + | + | - | + |
| XMLCOMMENT | ++ | ++ | - | ++ |
| XMLPI | + | + | - | +* |
| XMLAGG | + | ++ | - | ++ |
| XMLROOT | ++ | - | - | ++ |

++ - funkcja zaimplementowana, zgodna ze standardem SQL/XML

\+ - funkcja zaimplementowana, nieznacznie odbiega od standardu

\- - funkcja niezaimplementowana

* - funkcja zawiera rozszerzoną składnię i dodatkowe opcje

*Zestawienie 1: Wybrane funkcje standardu SQL/XML na tle omawianych SZBD.*

## 2.3. Tworzenie XML ze struktur relacyjnych

Jednym z podstawowych zadań jakim sprostać powinien system zarządzania bazami

danych z natywnym typem XML jest zdolność do integracji i swobodnego przekształcania danych relacyjnych na dane hierarchiczne i na odwrót. Konwersja danych pomiędzy modelem relacyjnym i hierarchicznym rodzi potrzebę zdefiniowania schematu opisu struktury danych oraz typu i zakresu prezentowanych informacji, zgodnych ze schematem modelu relacyjnego. Proces wymiany danych pomiędzy strukturą XML a relacyjną powinien zostać poprzedzony wcześniejszym zdefiniowaniem schematu opisującego charakter przechowywanych wartości. Można to uczynić dokonując odwzorowania struktury bazy relacyjnej do standardu XML Schema Definition (XSD). Specyfikacja języka XSD dostarcza zestawu narzędzi dla poprawnego odwzorowania elementów modelu relacyjnego (tabele, kolumny) wraz z zachowaniem występujących zależności oraz integralności istniejących danych. Realizowane jest to z jednej strony poprzez możliwość zastosowania określonego zestawu elementów oraz atrybutów języka XSD, umożliwiających wymuszenie np. unikalności wartości elementu (klucz główny w modelu relacyjnym) lub liczebności występowania elementów (klauzula NOT NULL itp. w modelu relacyjnym), z drugiej zaś strony poprzez określenie relacji zależności pomiędzy elementami schematu hierarchicznego, odzwierciedlającymi związki występujące w modelu relacyjnym.

Każdy z omawianych SZBD posiada własny, specyficzny mechanizm pozwalający generować schematy XML na podstawie modelu relacyjnego. Mechanizmy te w różnym stopniu odzwierciedlają strukturę danych relacyjnych. Jakość generowanych schematów pozostawia często wiele do życzenia. Z przeprowadzonej analizy wynika, że zdarzają się przypadki np. niepoprawnie rozpoznawanych typów danych, wymaganej liczebności czy unikalności elementów. Często definicja typu elementu ma charakter ogólny i konieczne jest jej doprecyzowanie.

Przykład:

| Kolumna | Typ | Modyfikatory |
|---|---|---|
| id | integer | **not null** |
| imie | character varying(20) | |
| nazwisko | character varying(20) | |

**PRIMARY KEY, btree (id)**

```
SELECT Table_To_Xmlschema('table_1', false, false, 'testns');
<xsd:schema  xmlns:xsd="http://www.w3.org/2001/XMLSchema"  targetNamespace="testns" … >
   <xsd:simpleType name="INTEGER"> ... </xsd:simpleType>
   <xsd:simpleType name="VARCHAR"> ... </xsd:simpleType>
  <xsd:complexType name="RowType.testdb.public.table_1">
     <xsd:sequence>
        <xsd:element name="id" type="INTEGER" minOccurs="0"></xsd:element>
        <xsd:element name="imie" type="VARCHAR" minOccurs="0"></xsd:element>
        <xsd:element name="nazwisko" type="VARCHAR" minOccurs="0"></xsd:element>
     </xsd:sequence>
  </xsd:complexType>
  ...
</xsd:schema>
```

*Przykład 10: Generowanie schematu XML przy pomocy funkcji TABLE_TO_XMLSCHEMA, wyróżniono błędnie zdefiniowany element „id”, serwer PostgreSQL.*

Generowanie zawartości XML na podstawie danych składowanych w modelu relacyjnym odbywać się może na kilka sposobów. Wykorzystać do tego celu można funkcje generujące standardu SQL/XML jak również szereg systemowych funkcji, właściwych dla danego SZBD. Poszczególne funkcje różnią się zarówno sposobem jak i treścią generowanych danych. Np. odwzorowanie elementów modelu relacyjnego realizowane może być zarówno poprzez zastosowanie deklaracji elementów, jak i zbioru atrybutów w dokumencie XML.

Przykład:

| Id | Imię | Nazwisko |
|---|---|---|
| 1 | Jan | Kowalski |

**Zbiór atrybutów:**

```
<Pracownik Id=”1” Imię=”Jan” Nazwisko=”Kowalski” />
```

**Zbiór elementów:**

```
<Pracownik>
    <Id>1</Id>
    <Imię>Jan</Imię>
    <Nazwisko>Kowalski</Nazwisko>
</Pracownik>
```

*Przykład 11: Prezentacja elementów w postaci zbioru atrybutów oraz zbioru elementów.*

Funkcje standardu SQL/XML dostępne są w systemach Oracle, DB2 oraz PostgreSQL (zgodnie z zestawieniem 1). Ponadto, zgodnie z [4] (rozdział 17, *Generating XML Data from the Database)* serwer Oracle posiada dodatkowe funkcje języka PL/SQL pozwalające na generowanie danych XML – m. in. SYS_XMLGEN i SYSXMLAGG oraz pakiet DBMS_XMLGEN zawierający szereg funkcji pozwalających na przekształcanie wyników zapytań SQL na format XML (CLOB lub XMLType).

Przykład:

```
DECLARE
    pracKontekst DBMS_XMLGEN.ctxHandle;
    wynik CLOB;
BEGIN
    pracKontekst := DBMS_XMLGEN.newContext('SELECT Id, Imię, Nazwisko FROM Pracownicy');
    DBMS_XMLGEN.setRowTag(pracKontekst, 'Pracownik');
    wynik := DBMS_XMLGEN.getXML(pracKontekst);
    DBMS_XMLGEN.closeContext(pracKontekst);
END;
```

*Przykład 12: Wykorzystanie funkcji pakietu DBMS_XMLGEN do wygenerowania dokumentu XML oraz zdefiniowania nazwy elementów reprezentujących pojedyncze wiersze zapytania SQL, serwer Oracle (wynik stanowi dokument XML ze zbiorem elementów jak w przykładzie 11).*

Przy pomocy funkcji pakietu DBMS_XMLGEN można m. in. odzwierciedlać nazwy kolumn wyników zapytań SQL na dowolne nazwy elementów XML, manipulować liczbą zwracanych lub pomijanych elementów oraz definiować reguły konwersji znaków specjalnych.

Podobne możliwości system SQL Server oferuje za pomocą klauzuli FOR XML. Tak jak przedstawiono w [1] (rozdział 2, *FOR XML and Legacy XML Support*) dzięki niej można w łatwy sposób przekształcać wyniki zapytań SQL do postaci XML. Klauzula posiada wiele opcji i modyfikatorów pozwalających m. in. definiować nazwy elementów (w tym elementu ROOT), dodawać definicje przestrzeni nazw, definiować sposób w jaki poszczególne wartości zwracane przez zapytanie będą reprezentowane w dokumencie XML (jako atrybuty lub elementy o dowolnej nazwie) oraz tworzyć schematy XSD dla wyników zapytania.

Przykład:

```
SELECT
        p.Id AS '@identyfikator',
        p.Imie AS 'imie',
        p.Nazwisko AS 'nazwisko',
        d.Nazwa AS 'dzial/nazwa'
FROM Hr.Pracownicy p LEFT JOIN Dzialy d ON p.id_dzial = d.id
FOR XML PATH ('prac'), ROOT ('lista_prac');
-----------------------------------------------------
<lista_prac>
   <prac identyfikator="12">
      <imie>Jan</imie>
      <nazwisko>Nowak</nazwisko>
      <dzial>
         <nazwa>Narzędziownia</nazwa>
      </dzial>
   </prac>
   ……
```

*Przykład 13: Wykorzystanie klauzuli FOR XML PATH wraz z opcją ROOT do nazwania elementu głównego dokumentu XML generowanego na podstawie wyniku zapytania SQL, serwer SQL Server.*

## 2.4. Import danych XML z zewnętrznych plików

Niewielkie dokumenty XML mogą być wprowadzane do bazy danych w postaci jawnie podawanego tekstu za pośrednictwem np. funkcji SQL/XML. W przypadku większych dokumentów jest to bardzo kłopotliwe lub niemożliwe. Niektóre aplikacje klienckie baz danych posiadają ograniczenia na długość ciągu wejściowego stąd też, konieczne jest stosowanie innych rozwiązań. W przypadku dokumentów XML składowanych na dysku w postaci plików wykorzystać można systemowe funkcje importu oferowane przez każdy z omawianych w tej pracy serwerów. Poniżej zaprezentowano przykłady użycia funkcji importu, specyficznych dla każdego z SZBD.

Przykład:

```
INSERT INTO tab(id,doc) VALUES(1, CAST(PG_READ_FILE('f.xml', 0, 10000000) AS XML))
```

*Przykład 14: Wykorzystanie funkcji PG_READFILE do importu danych XML z zewnętrznego pliku, serwer PostgreSQL.*

```
IMPORT FROM 'xmldir\files.del' OF DEL XML FROM 'xmldir' INSERT INTO tab
```

*Przykład 15: Wykorzystanie funkcji IMPORT do importu danych XML z zewnętrznych plików (wyszczególnionych w pliku importu .del), serwer DB2.*

```
INSERT INTO tab(id,doc) SELECT 1, x.* FROM OPENROWSET( BULK 'f.xml', SINGLE_BLOB) AS x
```

*Przykład 16: Wykorzystanie funkcji OPENROWSET do importu danych XML z zewnętrznego pliku (konwersja do XML odbywa się niejawnie), serwer SQL Server.*

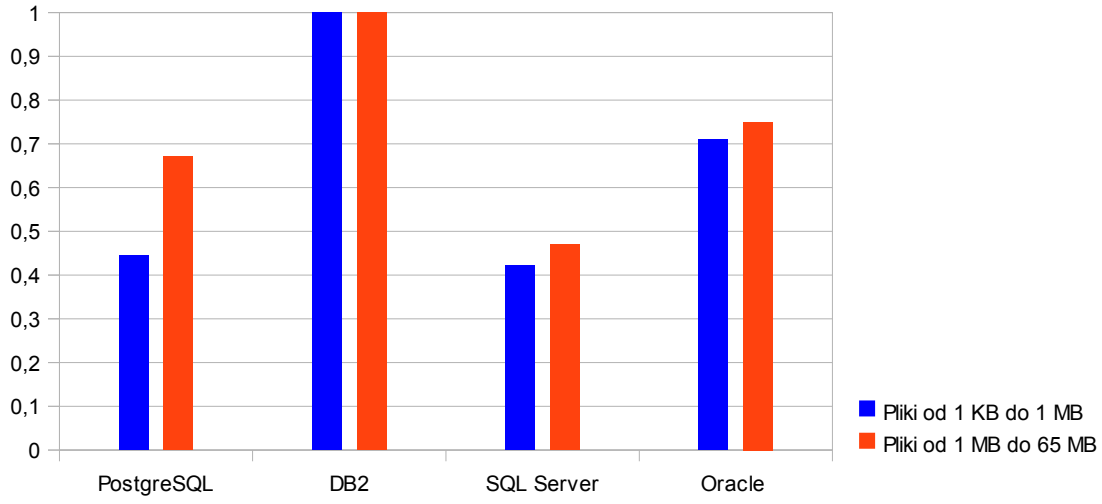
```
INSERT INTO tab VALUES (1, XMLType(BFILENAME('DIR', 'f.xml'), nls_charset_id('UTF8')))
```

*Przykład 17: Wykorzystanie funkcji BFILENAME do importu danych XML z zewnętrznego pliku, serwer Oracle.*

Poniżej zaprezentowano wyniki testów mających na celu sprawdzenie szybkości z jaką serwery baz danych konwertują dokumenty XML do wewnętrznej, natywnej postaci. Wyniki uzyskano na podstawie eksperymentów przeprowadzonych przez autora niniejszej

pracy. Na potrzeby eksperymentów wykorzystano narzędzia diagnostyczne (monitory aktywności, analizatory zapytań SQL) dostępne wraz z aplikacjami klienckim baz danych. Ponadto, czas konwersji na dane typu XML był mierzony przy pomocy aplikacji stworzonej przez autora pracy na potrzeby testów wydajności baz danych. Zadaniem aplikacji było zautomatyzowanie procesu testowania poprzez uzyskanie połączenia z każdym z opisanych SZBD (za pomocą sterownika ODBC[14]) a następnie uruchomienie i pomiar czasu wykonania sekwencji zapytań SQL zasilających bazy danych danymi typu XML. Testy przeprowadzono na danych o różnych rozmiarach (od 1 KB do 65 MB) oraz różnym stopniu zagnieżdżenia elementów. Końcowe wyniki uzyskano metodą wielokrotnego powtarzania testów oraz uśredniania wyników jednostkowych (po odrzuceniu skrajnych rezultatów). Na wykresie przedstawiono wyniki w podziale na dane o rozmiarze od 1 KB do 1 MB oraz od 1MB do 65MB.

Szczegóły dotyczące przebiegu testu oraz wykorzystanych danych opisane zostały w dodatku A.*Dodatek Specyfikacja testu A*.



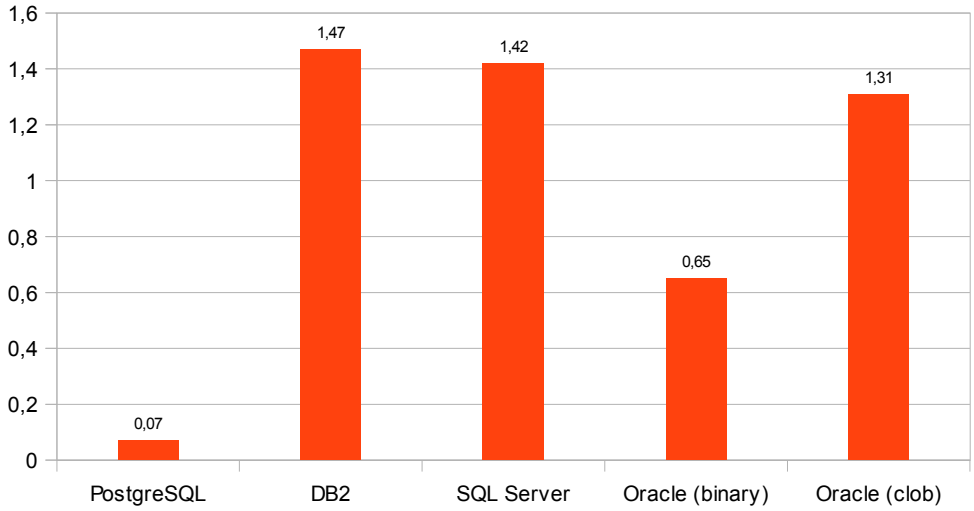

*Zestawienie 2: Porównanie szybkości konwersji danych tekstowych do natywnego typu XML. Na skali czasu 1 oznacza maksymalny czas działania wśród testowanych systemów. Pozostałe czasy zostały proporcjonalnie sprowadzone do tej skali.*

14 ODBC (*Open DataBase Connectivity* - otwarte łącze baz danych) - interfejs pozwalający aplikacjom łączyć się z systemami zarządzania bazami danych

Zestawienie 2 pokazuje, że czas konwersji na typ XML jest najkrótszy w przypadku serwera SQL Server i nieco dłuższy dla PostgreSQL oraz Oracle. Najdłużej trwa konwersja w serwerze DB2 – zarówno dla małych jak i większych dokumentów. Wynika to z konieczności przekonwertowania danych XML do wewnętrznej reprezentacji serwera oraz umieszczenia ich w osobnym, przeznaczonym dla danych XML magazynie. Podobną operację przed umieszczeniem w bazie danych przeprowadza serwer Oracle, jednak w tym przypadku proces trawa o ok 20-30 % krócej. W przypadku serwera PostgreSQL dłuższy czas pracy z większymi dokumentami związany jest z działaniem mechanizmu dzielenia i kompresji danych (TOAST).

Poniższe zestawienie zawiera porównanie przestrzeni dyskowej zajmowanej przez dokumenty XML składowane w poszczególnych bazach danych. Zestawienie jest wynikiem eksperymentów przeprowadzonych przez autora pracy. Polegały one na pomiarze różnicy pomiędzy rozmiarem dokumentów XML w systemie plików Windows (NTFS) a rozmiarem dokumentów składowanych w bazie danych w postacie danych typu XML. Do pomiaru wymienionych wielkości wykorzystano funkcje systemu Microsoft Windows XP oraz funkcje systemowe dostępne w każdym z opisywanych SZBD. Na potrzeby testów w bazach danych składowano dokumenty XML o znacznych rozmiarach (od 1 do 60 MB). Dla każdego z plików testy powtarzane były wielokrotnie, a rezultat końcowy uzyskano metodą uśredniania wyników poszczególnych prób (po uprzednim odrzuceniu wyników skrajnych). Wyniki dla serwera Oracle zaprezentowano w podziale na typ binarny oraz znakowy.

Szczegóły dotyczące przebiegu testu oraz wykorzystanych danych opisane zostały w dodatku B.*Dodatek Specyfikacja testu B.*




*Zestawienie 3: Porównanie wymaganej przestrzeni dyskowej dla dokumentów XML, w osi pionowej stosunek rozmiaru dokumentu XML w bazie danych do rozmiaru pliku na dysku w systemie plików Windows (NTFS).*

Serwer PostgreSQL jako jedyny w wymienionych stosuje klasyczną kompresję danych – stąd tak znaczące różnice w wielkości danych w bazie. Zastosowany algorytm charakteryzuje się dużą szybkością działania przy zachowaniu stosunkowo wysokiego stopnia kompresji danych (testy wykazały, że średni stopień kompresji dla dokumentów XML o rozmiarach 5-10 MB wynosi ok 17, przy czym stopień kompresji dla jednakowych dokumentów poddanych kompresji przy pomocy popularnych algorytmów (rar, zip, bz2) wyniósł ok 25-40). Znaczną redukcję wymaganej przestrzeni dyskowej zapewnia również typ binarny XMLType serwera Oracle. Testy wykazały, że zastosowany proces rozgraniczenia znaczników oraz konwersji wartości atrybutów i węzłów na inny niż tekstowy typ danych daje blisko dwukrotną redukcję wielkości danych w stosunku to typu XMLType CLOB (według twórców serwera Oracle może być to nawet pięciokrotny zysk). W pozostałych przypadkach dane w bazie przechowywane są w nieskompresowanej formie, we właściwej dla danego serwera wewnętrznej postaci stąd też ich rozmiar przekracza rozmiar importowanego dokumentu XML.

## 2.5. Ograniczenia dotyczące danych typu XML

### 2.5.1. PostgreSQL

Maksymalny rozmiar danych typu XML, podobnie jak pozostałych łańcuchów znaków wynosi 1 GB. Pamiętać jednak należy, że duże dokumenty XML przechowywane są w bazie w skompresowanej formie. Oznacza to, że pierwotny rozmiar pliku XML importowanego do bazy danych może być znacznie większy niż 1GB.

### 2.5.2. SQL Server

Zgodnie z [1] (rozdział 3. *The xml Data Type*) w SQL Server ograniczenie na wielkość pola typu XML wynosi 2 GB. Na rozmiar ten składa się właściwy dokument XML oraz jego indeksy: podstawowy oraz dodatkowe (opisane szczegółowo w rozdziale *Indeksy XML w SQL Server*).

### 2.5.3. DB2

W przeciwieństwie do typu VARCHAR czy CLOB typ XML nie posiada skojarzonej ze sobą długości. Według [2] (3.1 *The XML Data Type*) oraz [3] (rozdział 3. *XML database design*) magazyn XML oraz architektura przetwarzania w DB2 nie nakłada limitu na rozmiar dokumentu XML. Obecnie jedynie protokół komunikacyjny klient-serwer ogranicza wielkość dokumentu do 2 GB.

Magazyn XML przystosowany jest jedynie do składowania poprawnie sformatowanych dokumentów XML. Oznacza to, że nie ma możliwości umieszczania fragmentów dokumentów (bez elementu ROOT).

### 2.5.4. Oracle

Zgodnie z [4] (dodatek B. *Oracle XML DB Restrictions*) maksymalny rozmiar danych typu XMLType wynosi 4 GB. Wielkość ta ma jednak inne znaczenie w przypadku wersji binarnej a inne w wersji znakowej. Dla typu XMLType CLOB oznacza, że dokument może zawierać co najwyżej $2^{31}$ -1 znaków (2 bajty na znak). Dokumenty składowane jako XMLType binary podlegają konwersji na typ binarny i mają w bazie danych rozmiar znacznie mniejszy niż pierwotnie, w postaci tekstu. Stąd też, możliwe jest umieszczanie w systemie dokumentów znacznie przekraczających rozmiar 4 GB.

### 2.5.5. Podsumowanie

| | **PostgerSQL** | **DB2** | **Oracle** | **SQL Server** |
|---|---|---|---|---|
| **Maks. rozmiar** | 1 GB | dowolny * | 4 GB ** | 2 GB |

* protokół komunikacyjny klient-serwer nakłada ograniczenie do 2 GB na dokument

** zarówno dla dokumentów przechowywanych jako CLOB jak i binarnych po kompresji

*Zestawienie 4: Ograniczenie na wielkość dokumentu XML w bazie danych*

# Rozdział III
# XQuery/XPath w relacyjnych bazach danych

# 1. Język ścieżek XML: XPath

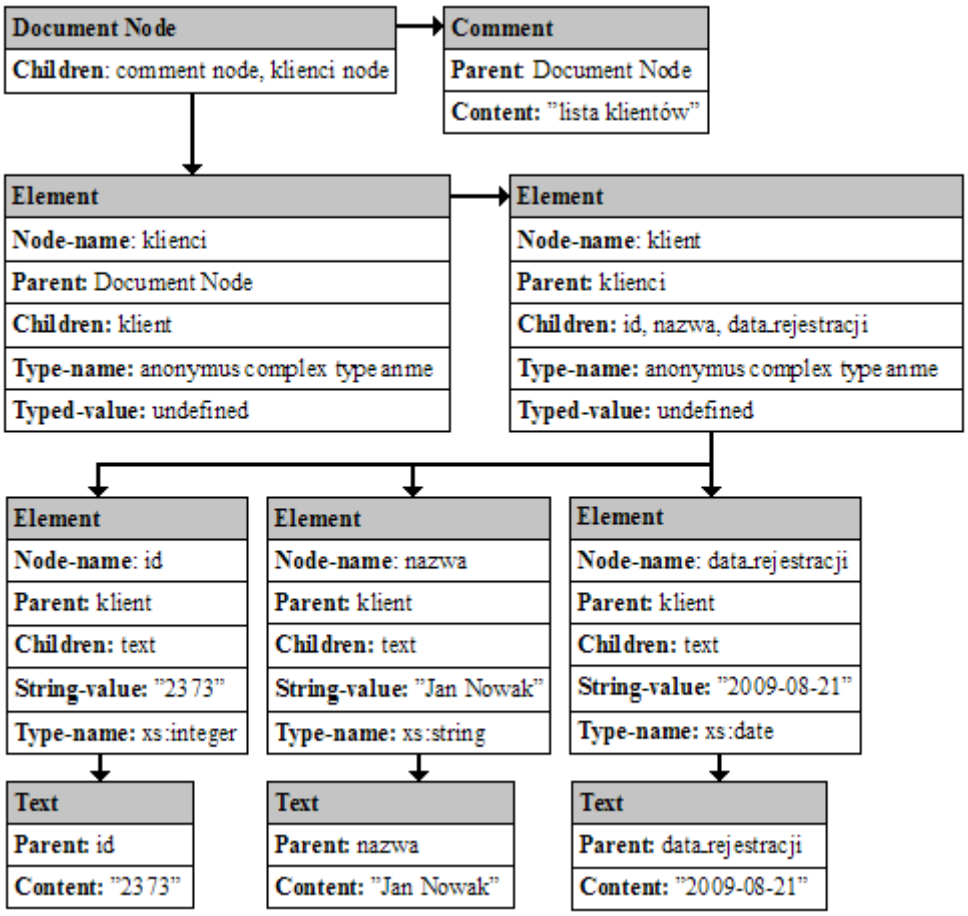
XPath (ang. XML Path Language) jest językiem wyrażeń służącym do adresowania składowych dokumentu XML. Znajduje zastosowanie w wielu technologiach związanych z XML - m. in. XQuery, XPointer, XSLT. Jest rekomendowany przez konsorcjum World Wide Web. Od 23 stycznia 2007 oficjalnie rekomendowanym standardem jest XPath 2.0. Nowa wersja standardu zawiera wiele znaczących zmian w stosunku do XPath 1.0. Jedną najistotniejszych zmian jest wprowadzenie modelu danych XDM (XQuery/XPath Data Model)[15] oraz związanego z nim nowego zestawu dostępnych typów danych. Zgodnie ze specyfikacją [9] XPath 1.0 zawiera jedynie podstawowe typy danych: liczby, łańcuchy znaków oraz wartości logiczne. Wynikiem wyrażenia XPath 1.0 jest zbiór węzłów. Specyfikacja standardu XPath 2.0 [10] definiuje blisko 50 różnych typów danych w tym typy złożone oraz typy definiowane przez użytkownika. Służy do przetwarzania wartości zgodnych z modelem danych XDM. Wynikiem wyrażenia XPath 2.0 jest również wartości w modelu XDM. Stanowi ona sekwencję zawierająca zero lub więcej elementów, które mogą być zarówno atomowymi wartościami (liczby, daty, łańcuchy znaków, wartości logiczne i wiele innych) jak również węzłami XML. Standard XDM definiuje siedem rodzajów dostępnych węzłów: dokument, element, atrybut, instrukcja przetwarzania, przestrzeń nazw, komentarz oraz tekst. W przeciwieństwie do zbioru węzłów (zwracanych przez XPath 1.0) elementy sekwencji mogą się powtarzać oraz posiadają kolejność. Więcej informacji na temat modelu XML Data Model zawiera specyfikacja [11].

Poniżej przedstawiono przykładową reprezentację danych XML w postaci drzewa w modelu danych XDM.

Przykład:

15 XDM (XQuery 1.0 and XPath 2.0 Data Model) – model danych języków XPath 2.0 oraz XQuery 1.0 rekomendowany przez konsorcjum World Wide Web Consortium. Oficjalna witryna projektu: http://www.w3.org/TR/xpath-datamodel

```
<!-- lista klientów-->
<klienci>
    <klient>
        <id>2373</id>
        <nazwa>Jan Nowak</nazwa>
        <data_rejestracji>2009-08-21</data_rejestracji>
    </klient>
</klienci>
```




*Schemat 2: Uproszczona reprezentacja dokumentu XML w postaci drzewa w modelu danych XML Data Model (XDM).*

Adresowanie za pomocą XPath stanowi podstawowy sposób nawigacji dla dokumentów XML. Systemy zarządzania bazami danych zawierające natywny typ danych XML w większości posiadają również wsparcie dla technologii XPath 1.0. Część z nich posiada również implementację nowszej wersji standardu.

Serwery DB2, Oracle oraz SQL Server obsługują zapytania XQuery 1.0 stąd też mają mechanizmy pozwalające przetwarzać wyrażenia w standardzie XPath 2.0 (jest on integralną częścią standardu XQuery 1.0). Serwer PostgreSQL jako jedyny z opisywanych nie posiada wsparcia dla wyżej wymienionych standardów (wsparcie to jest obecnie przedmiotem prac twórców systemu). Zawiera natomiast wbudowane funkcje pozwalające na przetwarzanie wyrażeń XPath 1.0. Zadanie to realizowane jest z wykorzystaniem zewnętrznego komponentu – *libxml2*[16]. Jest to popularna, stosowana w wielu projektach typu Open Source biblioteka zawierająca implementację wielu standardów związanych z językiem XML.

Jedną z funkcji serwera PostgreSQL pozwalających przetwarzać wyrażenia języka ścieżek w kontekście zapytań SQL jest funkcja XPATH.

Przykład:

```
SELECT  XPATH ('//klient/nazwa[1]/text()', doc) AS nazwa FROM table1;

nazwa
------------------
{"Jan Nowak"}
```

*Przykład 18: Wykorzystanie funkcji XPATH do przetwarzania wyrażeń języka XPath 1.0, serwer PostgreSQL.*

Wyrażenia w języku XPath występują przeważnie w kontekście innych konstrukcji językowych np. jako argumenty zapytań XQuery czy funkcji SQL/XML. Wykorzystywane są również w procesie indeksowania danych XML. Stąd też, implementacja XPath oraz przykłady wykorzystania w serwerach DB2, Oracle oraz SQL Server omówione zostały szerzej  w kolejnych podrozdziałach.

16 Libxml2 – darmowa biblioteka narzędziowa XML (zawiera m. in. parsery XML, implementację XPath 1.0, Xointer 1.0 i wiele innych) napisana w języku C, dostępna na licencji *MIT Licence*. Oficjalna witryna projektu: http://xmlsoft.org

## 2. Język zapytań XML: XQuery

XML Query jest językiem zapytań służącym do przeszukiwania dokumentów XML. 23 stycznia 2007 roku jako XQuery 1.0 roku zyskał rekomendację organizacji W3C. Formalnie stanowi rozszerzenie języka XPath 2.0 i również operuje na danych w modelu XDM. Rozszerzenie to polega m. in. na dodaniu zmiennych pomocniczych i klauzul służących do przeszukiwania, przekształcania, filtrowania oraz sortowania wyników zwróconych przez XPath. W rezultacie za pomocą XQuery tworzyć można konstrukcje językowe zwane w skrócie FLWOR (od angielskich słów *For-Let-Where-Order by-Return*), które operują na danych hierarchicznych w podobny sposób jak konstrukcje języka SQL: *Select-From-Where* na danych relacyjnych. Specyfikacja XQuery zawiera również instrukcje warunkowe oraz bogaty wachlarz funkcji i operatorów. Ponadto pozwala na definiowanie własnych funkcji – funkcji użytkownika. Mogą być one proste lub rekurencyjne. Przy ich konstruowaniu korzystać można zarówno ze standardowych funkcji XQuery jak również z własnych, uprzednio zdefiniowanych. Pełna lista funkcji i operatorów języka XQuery 1.0 dostępna jest w specyfikacji [12].

Serwery DB2 i Oracle umożliwiają uruchamianie zapytań XQuery na dwa sposoby: za pomocą funkcji bazodanowych specyficznych dla każdego z SZBD oraz za pomocą standardowych funkcji SQL/XML. SQL Server nie posiada wsparcia dla tych ostatnich - oferuje za to zestaw własnych, analogicznych funkcji o podobnych możliwościach.

Podstawową funkcją SQL/XML służącą do wykonywania zapytań XQuery jest XMLQUERY. Jej składnia pozwala przekazywać zapytania oraz dodatkowe parametry i argumenty. Wynikiem działania (podobnie jak w przypadku wyrażeń XPath 2.0) jest sekwencja elementów zgodna z modelem danych XDM. Wśród pozostałych funkcji standardu SQL/XML znaleźć można m. in. funkcję XMLCAST oraz XMLEXISTS. Pierwsza z nich pozwala rzutować zwracane przez XMLQUERY wyniki na standardowe typy SQL. Druga to predykat, przy pomocy którego określić można, czy zapytanie XQuery zwraca niepustą sekwencję.

Przykład:

```
SELECT XMLCAST (
   XMLQUERY ('$d/klienci/klient/nazwa' PASSING doc AS "d")
AS VARCHAR (20) ) nazwa FROM table1;

NAZWA
----------------
Jan Nowak
```

*Przykład 19: Wykorzystanie funkcji XMLQUERY i XMLCAST do wykonania zapytania XQuery i rzutowania wyniku na typ VARCHAR, serwer DB2.*

```
SELECT  XMLQUERY(  'for  $k  in /klienci
          where $k/punkty > 1000
          return
          <Szczegoly>
             <Klient  imie="{$k/imie}"  nazwisko="{$k/nazwisko}"/>
             <Nagroda>{if ($k/plec = "K") then "kosmetyki" else "sprzęt elektroniczny"} </Nagroda>
          </Szczegoly>'
       PASSING doc RETURNING CONTENT)  Nagrody_za_punkty
   FROM table1;
```

*Przykład 20: Wykorzystanie konstrukcji FLWOR do wygenerowania zawartości typu XML, serwer Oracle.*

Serwery Oracle oraz DB2 wyposażone są w silnik zapytań XQuery odpowiedzialny za obsługę wyrażeń w tym języku. Według [3] oraz [4] odbywa się ona w sposób natywny. Oznacza to, że zadania takie jak parsowanie, przetwarzanie czy optymalizacja zapytania wykonywane są bez przekształcania do języka SQL. Przetwarzane dane natomiast nie podlegają konwersji na model relacyjny - pozostają prze cały proces w postaci hierarchicznej.

SQL Server nie posiada implementacji funkcji SQL/XML związanych z XQuery. Wyżej opisane zadania realizowane są przez metody udostępnione przez typ danych XML. Podstawową z nich jest ***query***(), która przyjmuje zapytanie XQuery jako parametr i zwraca sekwencję węzłów w postaci danych typu XML. Funkcjonalność tej metody jest bardzo zbliżona do XMLQUERY. Ponadto do dyspozycji pozostają m. in. metody ***nodes***(), ***exist***(), ***value***(), za pomocą których można przekształcić dane XML do modelu relacyjnego, sprawdzić istnienie zadanego elementu, wyznaczyć skalarną wartość zadanego węzła.

Przykład:

```
SELECT doc.query('/klienci/klient/nazwa[1]/text()') AS nazwa FROM table1
WHERE  doc.exist('/klienci/klient[data_rejestracji="2009-08-21"]') = 1

nazwa
----------------
Jan Nowak
```

*Przykład 21: Wykorzystanie metod query() oraz exist() do wyznaczenia wartości w dokumencie spełniającym zadany warunek, serwer SQL Server.*

Funkcjonalność wymienionych metod nie odbiega znacząco od możliwości jakie dają funkcje standardu SQL/XML. Jednak sposób implementacji wsparcia dla zapytań XQuery w serwerze SQL Server znacznie różni się od tych zastosowanych w serwerach Oracle czy DB2. Twórcy SQL Server zdecydowali się na wykorzystanie istniejącego silnika SQL w procesie przetwarzania zapytań XQuery. Aby było to możliwe, wejściowe dane XML są rozdzielane (ang. *shredding*) na dane w modelu relacyjnym i w tej postaci podlegają działaniom procesora SQL. O tym, że operacja rozdzielania miała miejsce informuje dodatkowa pozycja w planie zapytania - **Table Valued Function {XML Reader}**. Wynikiem operacji jest tabela podobna do tej zamieszczonej poniżej.

```
<klienci>
    <klient>
        <id>2373</id>
        <nazwa>Jan Nowak</nazwa>
        <data_rejestracji>2009-08-21</data_rejestracji>
    </klient>
</klienci>
```

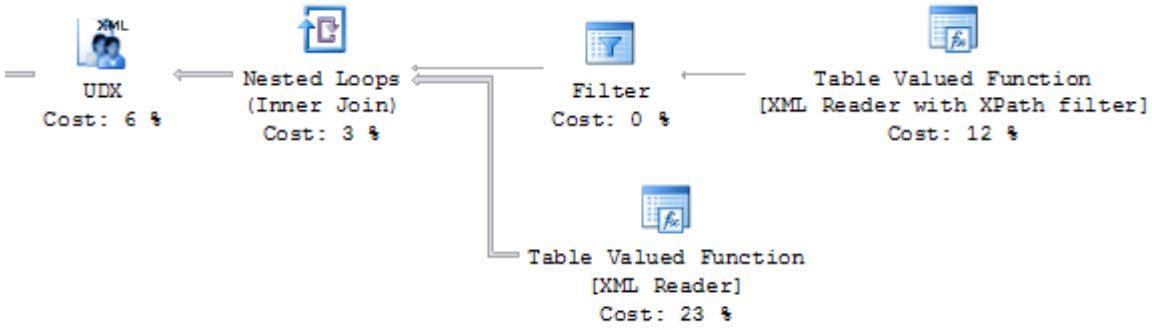
| ORDPATH | TAG | NODE | NODETYPE | VALUE | PATH_ID |
|---|---|---|---|---|---|
| 1 | 1 (klienci) | Element | klienciT | null | #klienci |
| 1.1 | 2 (klient) | Element | klientT | null | #klient#klienci |
| 1.1.1 | 3 (id) | Element | xs:int | 2373 | #id#klient#klienci |
| 1.1.3 | 4 (nazwa) | Element | xs:string | Jan Nowak | #nazwa#klient#klienci |
| 1.1.5 | 5 (data_rejestracji) | Element | xs:date | 2009-08-21 | #data_rejestracji#klient#klienci |

*Tabela 1: Uproszczona tabela wygenerowana na podstawie dokumentu XML - wynik operacji Table Valued Function {XML Reader}, serwer SQL Server.*

Kolejnym krokiem jest tłumaczenie zapytania XQuery na język SQL i wykonanie na danych uzyskanych z operacji rozdrabniania. Dane wynikowe są następnie poddawane przekształceniu z powrotem na typ XML. Odpowiedzialny za to jest operator **UDX**[17] **XML SERIALIZER**.

Poniżej zaprezentowano fragment planu zapytania z *Przykład 21* zawierający m. in. operatory **Table Valued Function: {XML Reader}** i {**XML Reader with XPath filter}** (szczególny przypadek {XML Reader} – operator generuje jedynie rekordy zgodne z zadanym wyrażeniem XPath) oraz **UDX XML SERIALIZER**:

17 Extended Operators (UDX) – rodzina operatorów SQL Server odpowiedzialna za przetwarzanie zapytań w języku XQuery



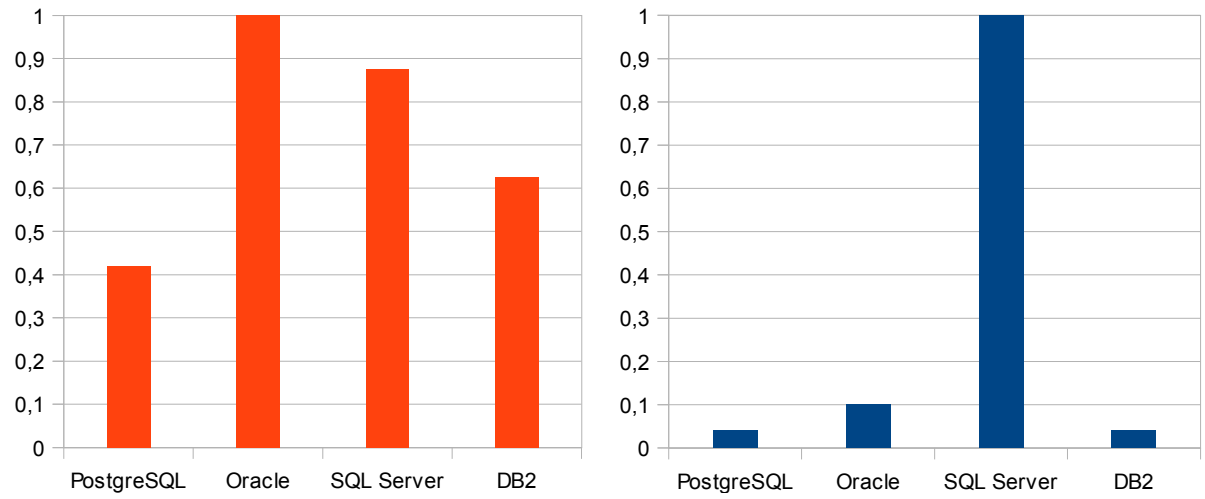

*Ilustracja 3: Fragment planu zapytania XQuery, serwer SQL Server.*

Zastosowane rozwiązanie niesie ze sobą ryzyko wielokrotnego, dynamicznego wykonywania operacji rozdzielania danych. W przypadku skomplikowanych zapytań XPath/XQuery obniżają one efektywność przetwarzania zapytań. Dlatego też, SQL Server w procesie indeksowania XML dokonuje wstępnego podziału danych (ang. *pre-shredding*) dzięki czemu w czasie wykonywania zapytania część danych jest dostępna w postaci relacyjnej. Sposób indeksowania danych w SQL Server został omówiony szerzej w podrozdziale *Indeksowanie danych XML*.

Poniżej zaprezentowano zestawienie zawierające porównanie szybkości wyszukiwania elementów XML przy pomocy zapytań XPath. Zestawienie przygotowano na podstawie eksperymentów przeprowadzonych przez autora pracy. W testach wykorzystano dokumenty o różnych rozmiarach (10KB – 10MB), bez indeksów XML. Na potrzeby testów dla każdego z dokumentów przygotowano szereg zapytań XPath o różnym stopniu skomplikowania, wykorzystujące różne operatory oraz predykaty dostępne w składni XPath. Wyniki zaprezentowano w podziale na 2 typy zapytań. Pierwszą grupę (kolor czerwony) stanowią zapytania o elementy XML, gdy struktura dokumentu jest dobrze znana. Drugą (kolor niebieski) – zapytania o elementy dokumentu, którego struktura jest tylko częściowo znana. W szczególności, zapytania drugiego typu zawierają operatory wieloznaczne takie jak operator dowolnego węzła o zadanej nazwie ('*//element*'), dowolnego dziecka elementu ('*'), dowolnego atrybutu ('@*'), rodzica węzła ('..') itp. Wyniki końcowe uzyskano metodą wielokrotnego powtarzania i uśredniania wyników pojedynczych prób.

Szczegóły dotyczące przebiegu testu oraz wykorzystanych danych opisane zostały w dodatku

C.*Dodatek Specyfikacja testu C.*




*Zestawienie 5: Porównanie szybkości wyszukiwania elementów XML przy pomocy zapytań XPath. Na skali czasu 1 oznacza maksymalny czas działania wśród testowanych systemów. Pozostałe czasy zostały proporcjonalnie sprowadzone do tej skali.*

Zestawienie 5 pokazuje, że uzyskane wyniki są silnie uzależnione od rodzaju przetwarzanych zapytań. W pierwszym przypadku (kolor czerwony) wszystkie SZBD uzyskały porównywalne wyniki. Co ciekawe, serwer PostgreSQL uzyskał lepsze rezultaty od serwerów DB2 oraz Oracle, które przetwarzają zapytania w sposób natywny. Ten dobry wynik to zasługa biblioteki *libxml2* odpowiedzialnej za obsługę języka XPath. Parsery zawarte w bibliotece uchodzą za bardzo efektywne, co znajduje potwierdzenie w wynikach prowadzonych testów. W drugiej grupie zapytań (kolor niebieski) proporcje wyników między serwerami PostrgeSQL, Oracle i DB2 nie zmieniły się. W przypadku SQL Server nastąpiło natomiast bardzo znaczące pogorszenie efektywności przetwarzania. W większości przeprowadzonych prób uzyskiwany czas był kilkukrotnie gorszy niż w pozostałych SZBD. W pojedynczych przypadkach zapytania przetwarzane były nawet dwustu-krotnie wolniej. Szczególnie kosztowne w użyciu okazały się operatory dowolnego potomka w drzewie XML ('*') oraz operator rodzica ('..'). Tak słaba wydajność związana jest ze sposobem w jaki SQL Server przetwarza zapytania XPath - dla zapytań zawierających wymienione operatory serwer zmuszony jest do wielokrotnego uruchamiania operacji rozdziału danych XML na model relacyjny, co w głównej mierze przyczynia się do obniżenia wydajności procesu.

## 3. Modyfikacja danych XML

Jedną z podstawowych cech, którą powinien posiadać SZBD wspierający format XML jest zdolność do swobodnego modyfikowania danych tego typu. Większość dostępnych systemów wychodzi naprzeciw temu zadaniu i dostarcza narzędzi umożliwiających operacje takie jak dodawanie, usuwanie oraz zmianę zawartości węzłów XML. Spośród opisywanych serwerów jedynie PostgreSQL nie posiada jeszcze funkcji pozwalających na wygodne modyfikowanie zawartości typu XML. W chwili obecnej sposobem na realizację tego zadania może być pobranie całego dokumentu, modyfikacja za pomocą zewnętrznych narzędzi i ponowne wprowadzenie zmodyfikowanego dokumentu do bazy danych. Pozostałe serwery: DB2, Oracle oraz SQL Server pozwalają na zmianę zawartości bez konieczności odczytu i zapisu całego dokumentu. Ze względu na znaczne różnice w sposobie reprezentacji danych XML implementacja powyższych funkcji różni się w poszczególnych SZBD.

Język XQuery 1.0 pozwala na selekcjonowanie fragmentów dokumentu XML oraz przetwarzanie zwracanych wyników. Standard ten nie definiuje jednak funkcji umożliwiających modyfikację samego dokumentu. Funkcje takie zdefiniowane zostały w oddzielnym dokumencie o nazwie XQuery Update Facility 1.0[18] stanowiącym rozszerzenie dla standardów XQuery 1.0 oraz XPath 2.0. Część z nich została zaimplementowana w opisywanych SZBD.

SQL Server umożliwia modyfikację danych XML za pomocą funkcji *modify()* wraz z jednym z trzech słów kluczowych: *insert*, *delete* i *replace value of*. Pozwalają one odpowiednio: wstawiać, usuwać oraz modyfikować węzły w dokumencie XML. Implementacja, podobnie jak w przypadku XQuery bazuje na silniku zapytań SQL i wymaga wcześniejszego rozdzielenia danych XML (tak jak przedstawiono w *Tabela 1*). Następnie konstrukcja *modify()* tłumaczona jest na język SQL, a powstałe zapytanie modyfikuje wygenerowaną tabelę. Ostatecznie zmodyfikowane dane są ponownie konwertowane na typ XML. W przypadku, gdy dane XML nie są zindeksowane wszystkie operacje wykonywane są „w locie” co stanowi duże obciążenie dla serwera.

18 XQuery Update Facility 1.0 – rozszerzenie standardów W3C: XQuery 1.0 oraz XPath 2.0. Oficjalna witryna projektu: http://www.w3.org/TR/xqupdate

Przykład:

```
UPDATE table1 SET doc.modify( 'insert ( <email>jnowak@abc.pl</email> )
        before
        (//klient[nazwa = "Jan Nowak"]/data_rejestracji)[1]')
WHERE id = 1
```

doc:

```
----------------------
<klienci>
    <klient>
        <id>2373</id>
        <nazwa>Jan Nowak</nazwa>
        <email>j.nowak@abc.pl</email>
        <data_rejestracji>2009-08-21</data_rejestracji>
    </klient>
</klienci>
```

*Przykład 22: Wstawianie nowego węzła w zadanym miejscu dokumentu XML, serwer SQL Server*

Serwer DB2 począwszy od wersji 9.5 również wspiera niektóre funkcje z rozszerzonego standardu XQuery Update Facility. W poprzednich wersjach modyfikacja danych XML odbywać się mogła jedynie „na zewnątrz" bazy danych. Obecnie operacje takie jak wstawianie, usuwanie czy modyfikowanie poszczególnych węzłów XML odbywają się bez konieczności kosztownego konwertowania danych na typ tekstowy czy model relacyjny. Zgodnie z [14] dane XML przez cały czas przetwarzania pozostają w wewnętrznej, hierarchicznej postaci. W serwerze DB2 zawartość typu XML modyfikować można przy pomocy konstrukcji *copy-modify-return* zdefiniowanej w [13].

Przykład:

```
UPDATE table1 SET doc = xmlquery ( 'copy  $new := $DOC  modify
    do delete  $new/klienci/klient/data_rejestracji  return  $new')
WHERE id = 1
```

doc:

```
----------------------
<klienci>
    <klient>
        <id>2373</id>
        <nazwa>Jan Nowak</nazwa>
        <email>j.nowak@abc.pl</email>
    </klient>
</klienci>
```

*Przykład 23: Usuwanie zadanych węzłów z dokumentu XML, serwer DB2 v 9.5.*

Proces modyfikacji danych za pomocą konstrukcji *copy-modify-return* przebiega według następującego scenariusza:

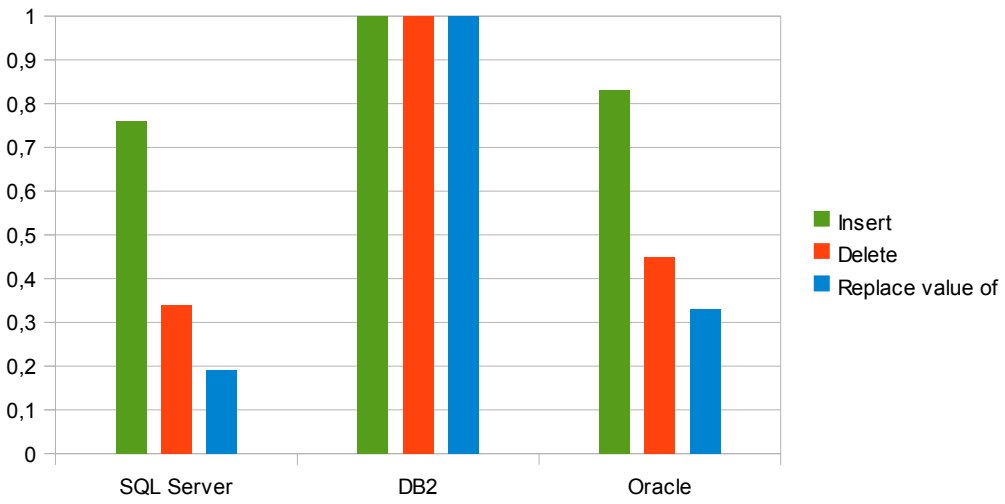
1) tworzona jest kopia oryginalnego dokumentu
2) na kopii dokumentu wykonywane są operacje wstawiania, usuwania lub zamiany zawartości węzła XML (*insert, delete* lub *replace value of*)
3) zmodyfikowana kopia zwracana jest jako wynik funkcji *xmlquery* i przypisywana do pola typu XML

Podobna sytuacja ma miejsce podczas modyfikacji danych XML w serwerze Oracle. Funkcje SQL odpowiedzialne za dodawanie, usuwanie lub zmianę węzłów XML również operują na kopii danych – oryginalne dane mogą być zmodyfikowane jedynie poprzez przypisanie otrzymanych wyników do pola XML. Nie oznacza to jednak, że podczas modyfikacji zawsze nadpisywany jest cały dokument. Zarówno w serwerze DB2, Oracle jak i SQL Server zaimplementowano mechanizmy odpowiedzialne za wyszukiwanie różnic pomiędzy oryginalną i zmodyfikowaną wersją danych. Dzięki temu na dysk zapisywane są jedynie te fragmenty, które uległy zmianie.

Poniżej zaprezentowano wyniki testów przeprowadzonych przez autora pracy. Mają

one na celu porównanie efektywności serwerów w zakresie modyfikacji fragmentów danych XML. Podobnie jak w poprzednich testach wyniki końcowe są uśrednieniem wyników uzyskanych z pojedynczych prób. Na wykresie wyróżniono operacje wstawiania (*insert*), usuwania (*delete*) oraz zmiany zawartości pojedynczego węzła XML (*replace value of*). Na potrzeby eksperyment modyfikowano dokumenty XML o rozmiarach 20kB – 1MB.

Szczegóły dotyczące przebiegu testu oraz wykorzystanych danych opisane zostały w dodatku D.*Dodatek Specyfikacja testu D*.




*Zestawienie 6: Porównanie szybkości modyfikacji i zapisu zmodyfikowanych danych w bazach danych SQL Server, DB2 oraz Oracle. Na skali czasu 1 oznacza maksymalny czas działania wśród testowanych systemów. Pozostałe czasy zostały proporcjonalnie sprowadzone do tej skali.*

Wyniki przeprowadzonych testów odbiegają od oczekiwań autora pracy - jednoznacznie wskazują, że operacje modyfikacji danych XML (szczególnie operacja zmiany wartości węzła) są najkosztowniejsze w serwerze DB2. Dzieje się tak, mimo iż modyfikacje wykonywane są w sposób natywny a do bazy danych zapisywane są tylko te węzły, których zawartość uległa zmianie. Zaskoczeniem jest dobry wynik uzyskany przez SQL Server. Sposób w jaki modyfikowane są dane XML wskazywałby na możliwość występowania

kosztownych zapytań i w rezultacie negatywnie wpłynął na końcowe wyniki. Nie znalazło to jednak potwierdzenia w wynikach przeprowadzonych testów.

# 4. XQuery kontra SQL

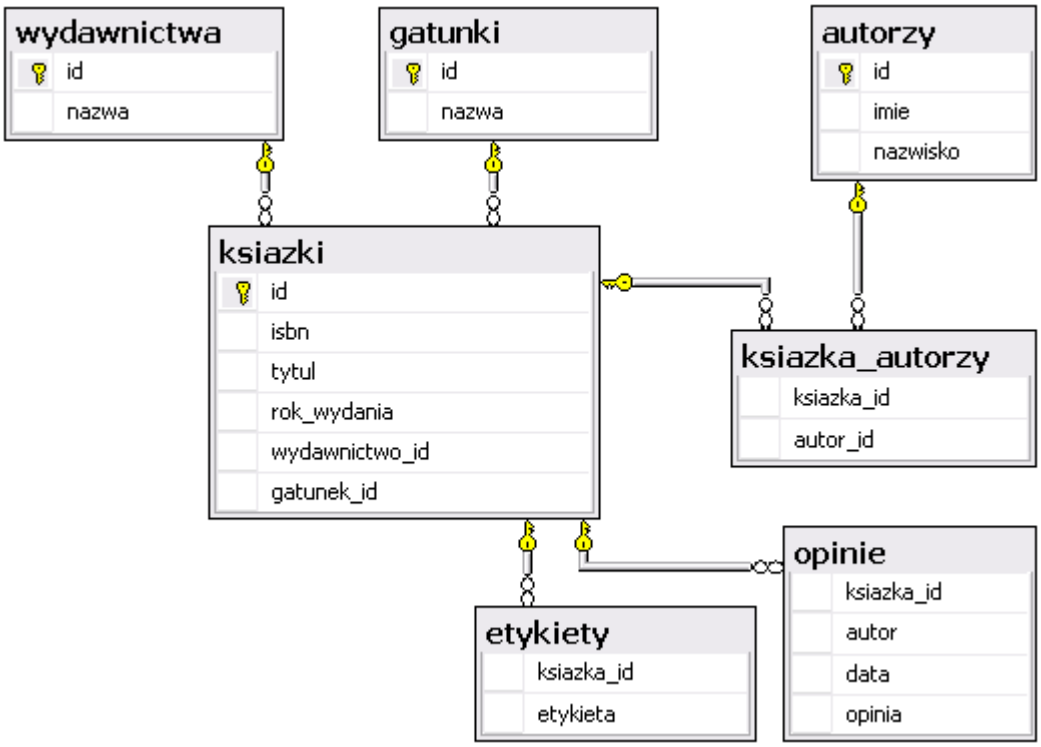
XML jest bardzo elastycznym i uniwersalnym formatem danych pozwalającym reprezentować dane hierarchiczne, strukturalne, półstrukturalne jak również dane o nieokreślonej strukturze. Twórcy systemów zarządzania bazami danych z natywnym typem danych XML zalecają jednak dużą ostrożność przy wyborze XML do składowania dużych ilości danych. Wybór taki ma bowiem uzasadnienie tylko w niektórych przypadkach. Za typem XML przemawiają sytuacje, w których przynajmniej jeden z poniższych warunków jest spełniony:

- dane mają swoją naturalną kolejność, kolejność elementu w kolekcji dostarcza dodatkowy kontekst
- istnieje potrzeba zastosowania niezależnego od systemu modelu, który wymaga przenośnego formatu danych
- przewidywane są częste zmiany struktury danych
- składowane dane są rzadkie
- składowane dane są półstrukturalne bądź nie mają zdefiniowanej struktury
- składowane dane są hierarchiczne

W pozostałych sytuacjach, w szczególności gdy dane są silnie strukturalne, sugerowanym przez twórców SZBD rozwiązaniem jest zastosowanie modelu relacyjnego.

W dalszej części zaprezentowano wyniki dwóch testów przeprowadzonych przez autora pracy, mających na celu sprawdzenie wydajności zapytań SQL oraz XPath/XQuery w zależności od struktury i wybranego modelu danych. W pierwszym z nich porównano wydajność zapytań do danych o dobrze zdefiniowanej, płaskiej strukturze. W drugim teście wykorzystano dane rzadkie, o budowie hierarchicznej, z zaznaczeniem kolejności elementów. W obydwu testach do reprezentacji danych wykorzystano zarówno model relacyjny jak i typ danych XML. Dla każdego z nich przygotowano zestaw zapytań SQL oraz XPath/XQuery o analogicznym działaniu.

Poniżej przedstawiono strukturę tabel oraz schemat XSD danych wykorzystanych w teście pierwszym.



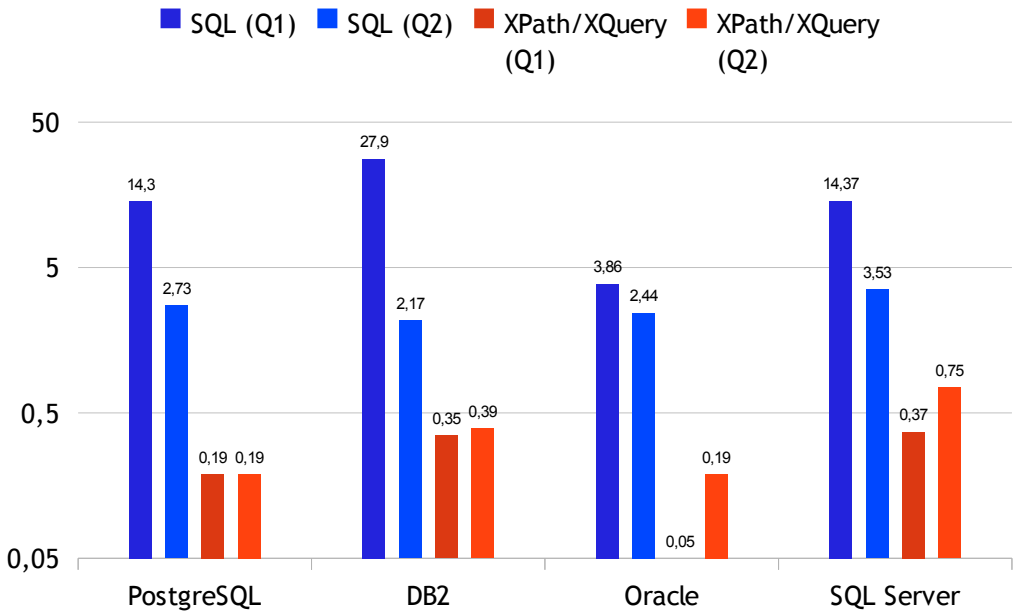

*Schemat 3: Uproszczona struktura tabel wykorzystanych w teście wydajności zapytań dla danych strukturalnych*

```
<xs:element name="ksiazka">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="tytul" minOccurs="1" maxOccurs="1"/>
            <xs:element ref="autor" maxOccurs="unbounded"/>
            <xs:element ref="wydawnictwo" maxOccurs="1"/>
            <xs:element ref="rok_wydania" maxOccurs="1"/>
            <xs:element ref="gatunek" maxOccurs="1"/>
            <xs:element ref="etykiety" minOccurs="0"/>
            <xs:element ref="opinie" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="isbn" type="xs:long" use="required"/>
    </xs:complexType>
</xs:element>
```

*Schemat 4: Fragment schematu XSD opisującego dane wykorzystane w teście wydajności zapytań dla danych strukturalnych*

Dane testowe reprezentują katalog książek zawierający ponad 60.000 pozycji. Do jego stworzenia wykorzystano informacje dostępne w sieci Internet oraz wygenerowane specjalnie na potrzeby eksperymentu. Jednakowe dane umieszczono w modelu relacyjnym (zgodnym ze schematem 3) oraz w postaci XML (dane XML stanowią 26 pojedynczych dokumentów, z których zawiera od 50 do 8.000 pozycji). Na dane XML nałożono schemat XSD oraz indeksy (za wyjątkiem serwera PostgreSQL). Zaprezentowane poniżej rezultaty uzyskano uruchamiając wielokrotnie zapytania i uśredniając wyniki pojedynczych prób.

Szczegóły dotyczące przebiegu testu, wykorzystanych danych oraz zapytań dla poszczególnych grup wyników (Q1, Q2) zawiera dodatek E.*Dodatek Specyfikacja testu E*.



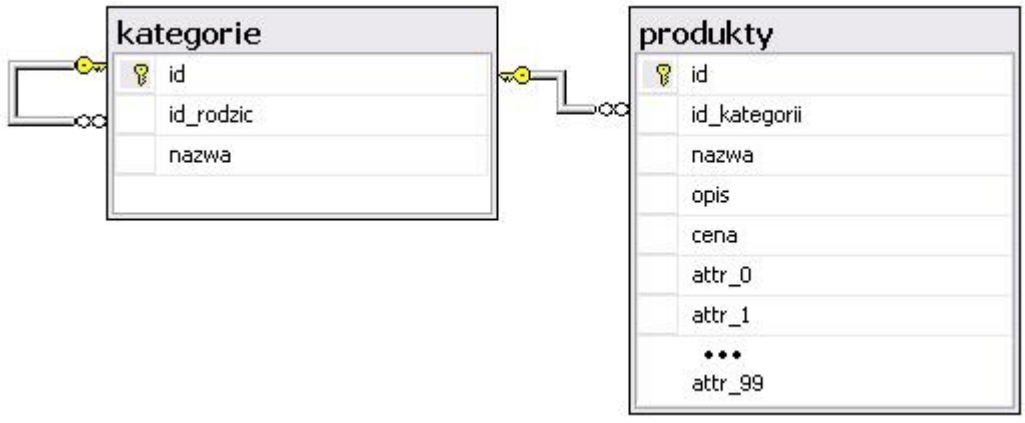

*Zestawienie 7: Porównanie wydajności zapytań SQL oraz XPath/XQuery dla danych strukturalnych w podziale na dwie grupy zapytań. W osi pionowej wydajność wyrażona w liczbie przetworzonych zapytań w czasie 1 sek.*

Zgodnie z oczekiwaniami autora testów wydajność zapytań SQL okazała się znacznie wyższa. W serwerach PostgreSQL, DB2 oraz Oracle maksymalne średnie relacje czasów zapytań XQuery do zapytań SQL okazały się bardzo zbliżone i wyniosły ok 70:1. W systemie

SQL Server relacja ta była o połowę mniejsza i wyniosła ok 38:1. W skrajnych przypadkach zapytania XQuery działały 1000-krotnie wolniej niż analogiczne SQL. Minimalne średnie relacje czasów wyniosły 14:1 (PostgreSQL, Oracle) oraz 5:1 (DB2, SQL Server). Tak duże różnice w czasach zapytań wynikają m. in. z faktu, iż SZBD optymalizowane są pod kątem wydajności dla danych strukturalnych a język SQL, w przeciwieństwie do XPath/XQuery jest naturalnym językiem relacyjnych baz danych. Dodatkowo, dane strukturalne mają zdefiniowany typ, nie wymagają parsowania, posiadają stały rozmiar i przeważnie ułożone są na dysku w sposób umożliwiający szybkie odczyt dużej liczby krotek. Przetwarzanie danych XML wiąże się natomiast z koniecznością ich parsowania oraz wykonywania wielu kosztownych operacji na danych znakowych.

W drugim teście wykorzystano dane rzadkie o budowie hierarchicznej – katalog 150.000 produktów podzielonych na ok 5.000 kategorii. Dane testowe posiadają strukturę drzewa, w którym węzłami wewnętrznymi są kategorie, natomiast węzłami zewnętrznymi (liśćmi) są produkty. Dodatkowo, z każdym produktem skojarzonych jest szereg atrybutów, właściwych dla jego kategorii. Drzewo danych ma wysokość 7, przy czym poziom liści drzewa wynosi od 5 do 7. Tak przygotowane dane zaimportowane zostały do baz danych na trzy sposoby: w postaci tabel relacyjnych, danych typu XML oraz z wykorzystaniem połączenia obydwu technik.

W reprezentacji tabelarycznej hierarchię kategorii uzyskano poprzez zastosowanie tabeli z parą identyfikatorów: własnym oraz kategorii nadrzędnej(rodzica), zgodnie z poniższym schematem.




*Schemat 5: Uproszczona struktura tabel wykorzystanych w teście wydajności zapytań SQL dla danych hierarchicznych*

Podczas testów prowadzonych na serwerach PostgreSQL, SQL Server oraz DB2 hierarchię kategorii odtwarzano z wykorzystaniem rekursywnych zapytań CTE (ang. *Common Table Expression*) reprezentujących tymczasowy zestaw rekordów definiowany w zasięgu jednego polecenia SQL. W testach serwera Oracle w tym celu wykorzystano klauzulę CONNET BY.

Przykład:

```
SELECT k.nazwa, p.nazwa, LEVEL
FROM kategorie k
INNER JOIN produkty p ON p.id_kategorii = k.id
START WITH k.nazwa = 'kategoria_0.2.7'
CONNECT BY PRIOR k.id = k.id_rodzic
ORDER BY LEVEL;
```

*Przykład 24: Wykorzystanie klauzuli CONNECT BY w celu pobrania wszystkich produktów należących pośrednio lub bezpośrednio do kategorii o zadanej nazwie, serwer Oracle*

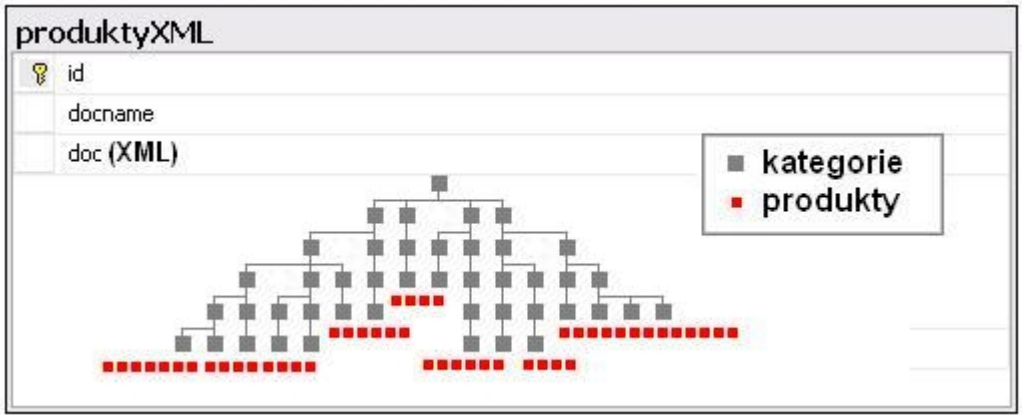
Przykład:

```
WITH RECURSIVE CTE_kategorie AS
(
    SELECT id, id_rodzic, nazwa, 0 AS POZIOM
    FROM kategorie WHERE nazwa = 'kategoria_0.2.7'
    UNION
    SELECT kt.id, kt.id_rodzic, kt.nazwa, POZIOM+1
    FROM kategorie kt INNER JOIN CTE_kategorie C ON kt.id_rodzic = C.id
)
SELECT k.nazwa, p.nazwa, POZIOM
FROM CTE_kategorie k
    INNER JOIN produkty p ON p.id_kategorii = k.id
ORDER BY POZIOM;
```

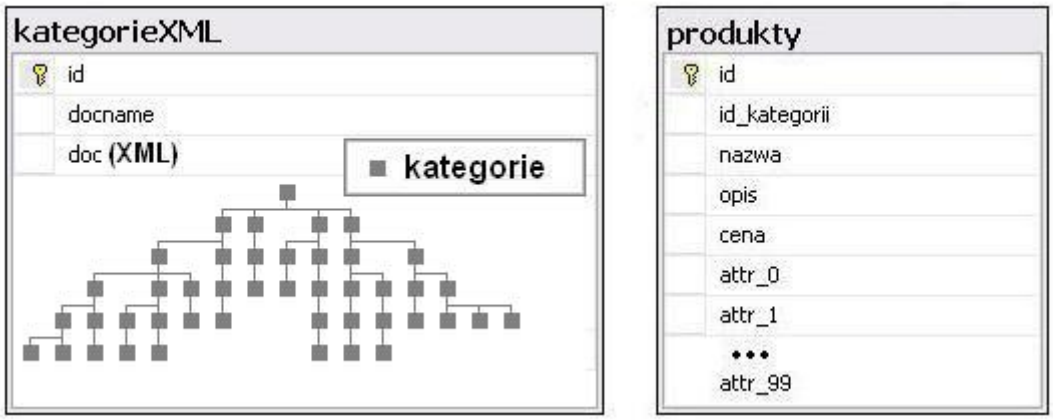
*Przykład 25: Wykorzystanie rekursywnego zapytania CTE w celu pobrania wszystkich produktów należących pośrednio lub bezpośrednio do kategorii o zadanej nazwie, serwer PostgreSQL*

Identyczny komplet danych umieszczony został w bazie danych z wykorzystaniem kolumny

typu XML na dwa sposoby: jako pojedynczy dokument XML zawierający kategorie wraz z produktami (tak jak przedstawiono na schemacie 6) oraz jako dokument XML zawierający jedynie kategorie połączony nieformalnie z tabelą relacyjną zawierającą produkty (tak jak przedstawiono na schemacie 7).




*Schemat 6: Tabela z kolumną typu XML wykorzystana w teście wydajności zapytań XQuery/XPath dla danych hierarchicznych*




*Schemat 7: Struktura tabel wykorzystana w teście wydajności zapytań SQL/XML dla danych hierarchicznych*

W modelu przedstawionym na schemacie 7 nieformalne powiązanie obydwu tabel otrzymano poprzez dodanie atrybutu „*id*” do każdego z węzłów XML reprezentującego kategorię produktów i wykorzystano jak w poniższy przykładach.

Przykład:

```
SELECT p.nazwa FROM
(
    SELECT  kat.id
    FROM  kategorieXML,  XMLTABLE(
        'for  $k  in  //kategoria_0.2.7//*  return  $k' PASSING doc COLUMNS  id  NUMBER  PATH  '@id')
kat
) k
INNER JOIN produkty p ON p.id_kategorii = k.id
ORDER BY p.id;
```

*Przykład 26: Wykorzystanie funkcji XMLTABLE do konwersji danych XML do formatu relacyjnego, serwer Oracle*

Przykład:

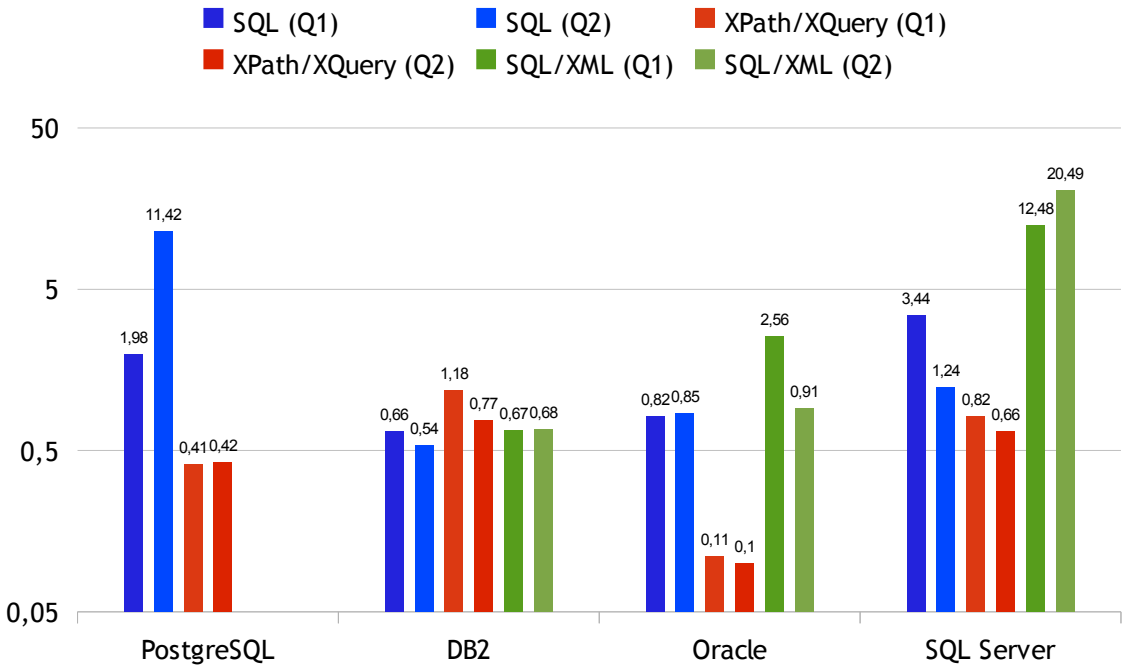
```
DECLARE @x XML;
SET @x = (SELECT doc FROM kategorieXML WHERE id = 0)

SELECT p.nazwa
FROM
(
    SELECT k.kategoria.value( N'@id',  N'integer' )  AS  id
    FROM @x.nodes( N'//kategoria_0.2.2.1//*' )  AS k ( kategoria )
) kat
INNER JOIN  produkty  p  ON  kat.id  =  p.id_kategorii
ORDER BY  p.id;
```

*Przykład 27: Wykorzystanie metod nodes()  oraz value() w celu pobrania węzłów XML i konwersji wartości danego atrybutu na typ SQL:INTEGER, serwer SQL Server*

Dla każdego z opisanych modeli danych przygotowano szereg zapytań o jednakowym

działaniu. Na zestawieniu 8 kolorem niebieskim oznaczono wyniki dla zapytań do danych o budowie relacyjnej, kolorem czerwonym wyniki dla danych składowanych wyłącznie jako XML natomiast kolorem zielonym wyniki dla danych o budowie jak na schemacie 7. Dla serwera PostgreSQL, ze względu na brak implementacji funkcji SQL/XML pozwalających na łatwą konwersję danych XML do formatu relacyjnego, testy przeprowadzono jedynie dla zapytań SQL i XPath. Wyniki zaprezentowano w rozbiciu na dwie grupy zapytań (Q1 oraz Q2).

Szczegóły dotyczące przebiegu testu, wykorzystanych danych oraz zapytań dla poszczególnych grup wyników zawiera dodatek F.*Dodatek Specyfikacja testu F.*




*Zestawienie 8: Porównanie wydajności zapytań SQL, XPath/XQuery oraz SQL/XML dla danych hierarchicznych w podziale na dwie grupy zapytań. W osi pionowej wydajność wyrażona w liczbie przetworzonych zapytań w czasie 1 sek.*

Zgodnie z otrzymanymi wynikami we wszystkich testowanych systemach za wyjątkiem DB2 zastosowanie modelu relacyjnego i rekursywnych zapytań SQL okazało się lepszym rozwiązaniem niż składowania wszystkich danych w postaci pojedynczego dokumentu XML i ekstrahowanie węzłów za pomocą XQuery. Różnice otrzymanych czasów

wyniosły od 30% do nawet 1200% na korzyść zapytań SQL. Przeciętnie polecenia rekursywne działały od 3 (SQL Server) do 8 (Oracle) razy szybciej, mimo że końcowy zbiór krotek poddawany był sortowaniu (w celu zasymulowania kolejności elementów). Całkiem odmienne rezultaty otrzymano dla modelu łączącego zalety danych XML oraz tabel relacyjnych. W przypadku systemów Oracle oraz SQL Server, gdy dokument XML zawierał jedynie hierarchię kategorii a listę produktów umieszczono w osobnej tabeli uzyskano wyniki kilkukrotnie lepsze w stosunku do SQL i kilkunastokrotnie lepsze w stosunku do XQuery. Rezultat wynika z faktu, iż w testowanych zapytaniach zastąpiono kosztowne polecenia rekursywne na instrukcje XQuery, które w wymienionych serwerach działają bardzo szybko dla stosunkowo niedużych dokumentów XML. Jedynie w systemie DB2 odnotowano nieznaczną przewagę w wydajności poleceń XQuery nad zapytaniami SQL i SQL/XML – różnica wyniosła przeciętnie ok 50%.

# Rozdział IV
# Indeksy XML w relacyjnych bazach danych

# 1. Indeksowanie danych XML

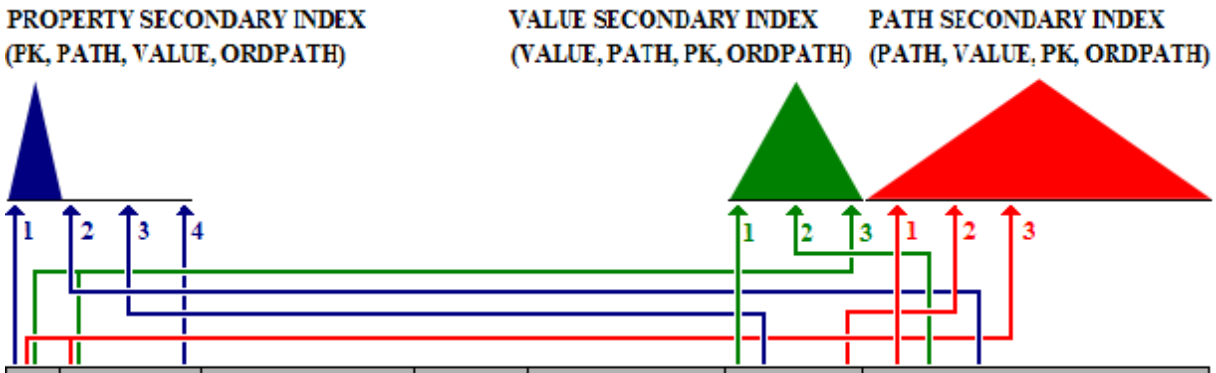
Indeksy to uporządkowane struktury, których zadaniem jest przyśpieszenie dostępu do danych w bazie. W relacyjnych bazach danych są powszechnie stosowane i często znacząco podnoszą efektywność zapytań typu DQL (ang. *Data Query Language*). Problem szybkiego wyszukiwania elementów dotyczy także danych w postaci XML. Ze względu na swoją odmienną budowę dokumenty XML wymagają jednak innych mechanizmów pozwalających indeksować dane. Stąd też, obok danych typu XML, relacyjne bazy danych oferują specjalny typ indeksu – indeks XML. Podobnie jak w przypadku danych relacyjnych indeksy XML mają za zadanie podnieść wydajność operacji związanych z wyszukiwaniem danych. Sposób w jaki zostały one zaimplementowane w poszczególnych SZBD jest zróżnicowany i ma związek ze sposobem implementacji samego typu danych XML. Poniżej omówione zostały rozwiązania zastosowane w serwerach Oracle, DB2 oraz SQL Server. Przedstawiono także zestawienia zawierające porównanie najistotniejszych cech poszczególnych implementacji indeksów XML. Wszystkie wyniki zaprezentowane w niniejszym rozdziale otrzymano w drodze testów przeprowadzonych przez autora pracy. W rozdziale pominięto omówienie dotyczące serwera PostgreSQL ze względu na brak implementacji indeksu typu XML. Indeks ten jest obecnie przedmiotem prac twórców systemu.

## 1.1. Indeksy XML w SQL Server

SQL Server zawiera 4 typy indeksów XML: 1 podstawowy (*Primary XML Index*) oraz 3 dodatkowe (*Secondary XML Index*). Podczas uruchomiania zapytań XQuery na dokumencie XML SQL Server dokonuje rozdziału danych na format relacyjny. Utworzenie indeksu podstawowego na kolumnie typu XML skutkuje wcześniejszym podziałem danych (ang. *preshreeding*) i eliminuje obciążenie związane z wykonywaniem tej operacji każdorazowo podczas uruchamiania zapytań XQuery. Indeks podstawowy stanowi zatem tabelę podobną do zaprezentowanej w *Tabela 1* z tą różnicą, że zawiera dodatkowo kolumnę *PK* (*Primary Key*) przechowującą wartości klucza podstawowego indeksowanej tabeli (jednym z warunków koniecznych do utworzenia indeksu podstawowego jest obecność klucza podstawowego w indeksowanej tabeli). Każdy rekord indeksu podstawowego(tabeli) odpowiada pojedynczemu węzłowi XML. Kolumna *PK* pozwala określić które rekordy należą do danego dokumentu

oraz umożliwia łączenie indeksu z innymi tabelami w bazie danych podczas wykonywania zapytań XQuery.

Podczas, gdy indeks podstawowy eliminuje z planu zapytania operacje rozdzielania danych, efektywność zapytań można jeszcze podnieść poprzez utworzenie indeksów dodatkowych. SQL Server oferuje 3 ich rodzaje: *PATH Secondary Index*, *VALUE Secondary Index* oraz *PROPERTY Secondary Index*. Są one, w przeciwieństwie do indeksu podstawowego tradycyjnymi, relacyjnymi indeksami o strukturze B-drzewa. Założenie indeksu dodatkowego skutkuje w rzeczywistości założeniem zwykłego, nieklastrowego indeksu na utworzonej tabeli (indeksie *Primary XML*). W zależności od jego rodzaju (*PATH, VALUE* bądź *PROPERTY*) pola w tabeli wykorzystywane są w różnej kolejności, zgodnie z poniższą ilustracją.




| PK | ORDPATH | TAG | NODE | NODETYPE | VALUE | PATH |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 (klienci) | Element | klienciT | null | #klienci |
| 1 | 1.1 | 2 (klient) | Element | klientT | null | #klient#klienci |
| 1 | 1.1.1 | 3 (id) | Element | xs:int | 2373 | #id#klient#klienci |
| 1 | 1.1.3 | 4 (nazwa) | Element | xs:string | Jan Nowak | #nazwa#klient#klienci |
| 1 | 1.1.5 | 5 (data_rejestracji) | Element | xs:date | 2009-08-21 | #data_rejestracji#klient#klienci |

*Ilustracja 4: Podstawowy indeks XML (tabela w wersji uproszczonej) oraz struktura indeksów dodatkowych (Secondary XML Index), SQL Server*

*PATH Secondary XML Index* został zaprojektowany z myślą o optymalizacji zapytań, w których:

a) jedynie ścieżka XML została określona

b) jedynie ścieżka XML została określona a wartość węzła użyta została w predykacie

*VALUE Secondary XML Index* podnosi efektywność zapytań, w których wartość węzła jest określona lecz nazwa i/ lub lokalizacja węzła zawierającego daną wartość jest nieznana.

*PROPERTY Secondary XML Index* został stworzony z myślą o sytuacjach, gdy dane typu XML stanowią kontener wartości skalarnych. Służy optymalizacji zapytań pobierających wartość jednego bądź większej liczby węzłów XML.

Poniżej zaprezentowano przykład tworzenia indeksu podstawowego oraz dodatkowych w tabeli *table1* z kolumną *doc* typu XML (utworzenie indeksu dodatkowego możliwe jest tylko wtedy, gdy uprzednio utworzony został indeks podstawowy).

```
CREATE PRIMARY XML INDEX PXML_table1_doc
ON table1 ( doc );

CREATE XML INDEX SXML_table1_doc_value
ON table1 ( doc )
USING XML INDEX PXML_table1_doc  FOR VALUE;
```

*Przykład 28: Tworzenie indeksów XML: podstawowego oraz dodatkowego (VALUE Secondary Index), server SQL Server*

## 1.2. Indeksy XML w DB2

Indeksy XML w DB2 dzielą się na 2 typy: indeksy tworzone automatycznie oraz tworzone przez użytkownika. Jednym z dwóch pierwszego typu jest indeks regionów XML (nazwa DB2: *XML Regions Index,* oznaczenie: *XRGN*) tworzony w przestrzeni danych relacyjnych w trakcie dodawania pierwszego dokumentu do tabeli z kolumną typu XML. Regionem nazywane są węzły i poddrzewa pojedynczego dokumentu XML przechowywane na jednej stronie danych w XDA (ang. *XML Data Area*). Liczba regionów (i jednocześnie wpisów w indeksie) jest bezpośrednio związana z wielkością strony – im większa strona danych tym mniejszy indeks i na odwrót. Indeks ten stanowi łącznik pomiędzy składem

relacyjnym a przestrzenią danych hierarchicznych (XDA). Wpis związany z pojedynczym wierszem tabeli wskazuje zawsze na węzeł *ROOT* dokumentu XML. Serwer tworzy zawsze dokładnie jeden taki indeks dla każdej tabeli zawierającej kolumnę XML.

Drugim indeksem tworzonym bez udziału użytkownika jest indeks ścieżek XML (nazwa DB2: *XML Column Path Index*, oznaczenie: *XPTH*). Powstaje on automatycznie dla każdej z kolumn XML i odwzorowuje ścieżki na identyfikatory ścieżek. Podczas wstawiania nowego dokumentu do bazy danych system ekstrahuje każdą unikatową ścieżkę i umieszcza w indeksie wraz z unikatowym w ramach kolumny identyfikatorem. Dzięki temu możliwe jest łatwe wyselekcjonowanie dokumentów, które zawierają węzły o ścieżce użytej w zapytaniu. Używany jest celu podniesienia wydajności dla ogółu zapytań XQuery oraz SQL/XML.

Oprócz indeksów tworzonych automatycznie DB2 pozwala tworzyć indeksy XML definiowane przez użytkownika (nazwa DB2: *XML Index*). Umożliwiają one zwiększenie szybkości dostępu do grupy węzłów określonej za pomocą wzorca XPath. W indeksie XML umieszczana jest m. in. wartość węzła oraz identyfikator dokumentu, w którym znajduje się węzeł o ścieżce zgodnej ze wskazanym wzorcem. Działanie indeksu polega na ograniczeniu zbioru skanowanych dokumentów jedynie do tych, które zawierają poszukiwane węzły.

Poniżej zaprezentowano przykład tworzenia indeksu XML dla danych typu znakowego (SQL VARCHAR).

Przykład:

```
CREATE INDEX Klient_miasto_Index ON table1(xmlDoc) GENERATE KEY
USING XMLPATTERN '/klienci/klient/adres/miasto' AS SQL VARCHAR
```

*Przykład 29: Tworzenie indeksu XML dla węzłów o wartościach typu SQL VARCHAR, serwer DB2*

W rzeczywistości uruchomienie powyższego zapytania powoduje utworzenie dwóch indeksów: logicznego (oznaczenie DB2: XVIL) o nazwie zdefiniowanej przez użytkownika oraz fizycznego (oznaczenieDB2: XVIP) o nazwie wygenerowanej przez serwer. Użytkownik operuje indeksem XML na poziomie logicznym, choć ten przechowuje jedynie informacje o zastosowanym wzorcu. Rzeczywiste wartości indeksu XML zawarte są w indeksie fizycznym – jego działanie jest jednak niewidoczne dla użytkownika.

Index XVIP ma strukturę B-drzewa i składowany jest w przestrzeni indeksów relacyjnych. W przeciwieństwie do tradycyjnych indeksów może zawierać zero, jeden lub więcej niż jeden wpis dla pojedynczego wiersza w tabeli. Liczba wpisów w indeksie zależy od liczby węzłów XML w dokumencie, które odpowiadają zadanemu wzorcowi.

## 1.3. Indeksy XML w Oracle

Wersja Oracle 11g oprócz nowego typu danych – binarnego XML dostarcza również nowy typ indeksu – XMLIndex przeznaczony dla danych XML. Indeks może być wykorzystywany w trakcie wyliczania funkcji SQL/XML takich jak XMLExists(), XMLTable(), XMLQuery(). Implementacja indeksu oparta jest na szeregu logicznych, relacyjnych indeksów utworzonych na tabeli ścieżek (ang. *Path Table*). Tabela ta zawiera jeden wiersz dla każdego zindeksowanego węzła w dokumencie XML. Przechowywane są w niej następujące informacje:

- identyfikator wiersza, w którym umieszczony jest dokument zawierający dany węzeł (kolumna ROWID)
- wskaźnik pozwalający na szybki dostęp do odpowiedniego fragmentu danych składowanych w przestrzeni danych hierarchicznych (kolumna LOCATOR)
- klucz pozwalający określić dokładną pozycję węzła w hierarchii dokumentu (z zachowaniem kolejności węzłów). Klucz ma postać sekwencji: $n_1.n_2$, … $.n_k$ oznaczającej, że dany węzeł jest $n_k$-tym potomkiem $n_{k-1}$-tego potomka … $n_1$-tego potomka węzła ROOT (kolumna ORDER_KEY)
- identyfikator ścieżki XPath węzła (kolumna PATHID)
- wartość węzła prostego (bez potomków) w postaci tekstowej (kolumna VALUE)

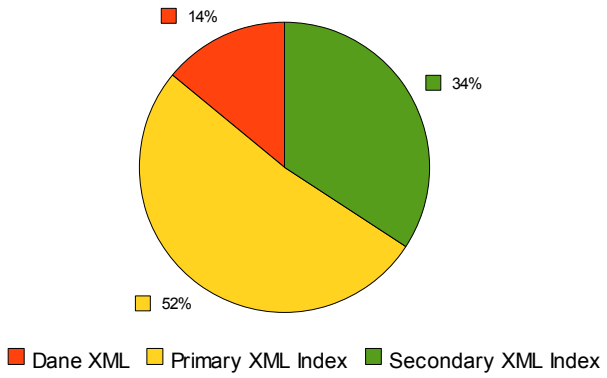
Na tak utworzonej tabeli automatycznie tworzone są tradycyjne indeksy w postaci B-drzew: indeks ścieżek (ang. *path index*), indeks pozycji (ang. *order index*) oraz indeks wartości (ang. *value index*). Rozwiązanie to jest analogiczne do idei indeksu podstawowego i indeksów dodatkowych zastosowanych w systemie SQL Server.

## 2. Wady i zalety stosowania indeksów XML

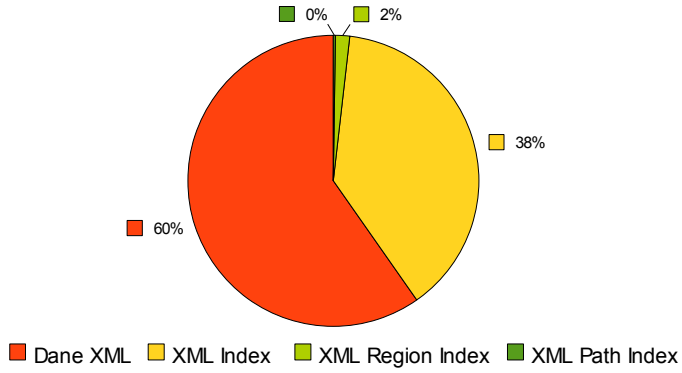
Stosowanie indeksów XML niesie ze sobą główną korzyść w postaci zwiększenia wydajności zapytań XQuery oraz SQL/XML. Zysk może być ogromny w jednej sytuacji a znikomy w innej - zależy to w głównej mierze o konkretnych danych i użytego zapytania. Podobnie jak w przypadku indeksów tworzonych dla standardowych typów SQL potencjalne korzyści mają także swoją cenę. Użytkowanie indeksów niesie ze sobą konieczność ich odbudowy po operacji modyfikacji na danych. Przy dużej liczbie operacji tego typu poniesione koszty mogą skutecznie niweczyć plany podniesienia wydajności bazy danych. Nie bez znaczenia pozostaje również rozmiar samej struktury. O ile w przypadku standardowych indeksów ich wielkość jest przeważnie istotnie mniejsza od wielkości całej tabeli o tyle rozmiar indeksu XML może nawet kilkukrotnie przewyższać rozmiar indeksowanych danych.

Poniżej zaprezentowano szereg zestawień przedstawiających procentowy podział przestrzeni dyskowej wymaganej przez poszczególne SZBD do składowania danych oraz indeksów XML. Zestawienia utworzono badając średni rozmiar indeksów utworzonych na danych XML o łącznym rozmiarze od 250 KB do 250 MB.

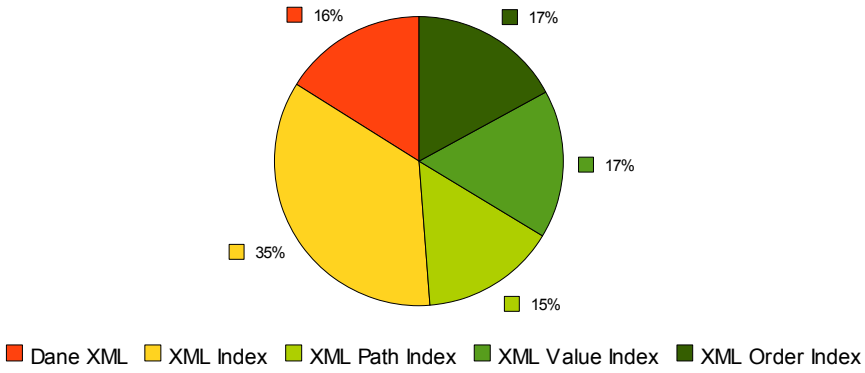
Szczegóły dotyczące przebiegu testu oraz wykorzystanych danych zawiera dodatek G.*Dodatek Specyfikacja testu G*.




*Zestawienie 9: podział przestrzeni dyskowej wymaganej do składowania danych oraz indeksów XML: podstawowego oraz jednego dodatkowego , serwer SQL Server*




*Zestawienie 10: podział przestrzeni dyskowej wymaganej do składowania danych oraz indeksów XML , serwer DB2*




*Zestawienie 11: podział przestrzeni dyskowej wymaganej do składowania danych oraz indeksów XML , serwer Oracle*

Zgodnie z otrzymanymi wynikami na każdy 1MB danych XML składowanych przez SQL Server potrzeba 3,71 MB przestrzeni dla indeksu podstawowego oraz 2,43 MB na każdy z indeksów dodatkowych. Producent SQL Server zaleca jednak tworzenie nie więcej niż jednego indeksu dodatkowego – stąd łączny łączny rozmiar indeksu można szacować na ok 6MB na każdy 1MB danych XML. Podobny rozmiar dla jednakowych danych uzyskano w przypadku serwera Oracle: 2,19 MB indeksu podstawowego oraz po ok 1MB indeksów dodatkowych na każdy 1MB danych XML. Wszystkie trzy indeksy dodatkowe tworzone są automatycznie zatem łączny rozmiar indeksu może nawet pięciokrotnie przewyższać rozmiar danych. Tak duże wartości wynikają z zawartości tabel stanowiących trzon indeksów w wymienionych SZBD. Dla każdego węzła XML tworzony jest bowiem rekord zawierający oprócz wartości także szereg dodatkowych informacji, na podstawie których możliwe jest

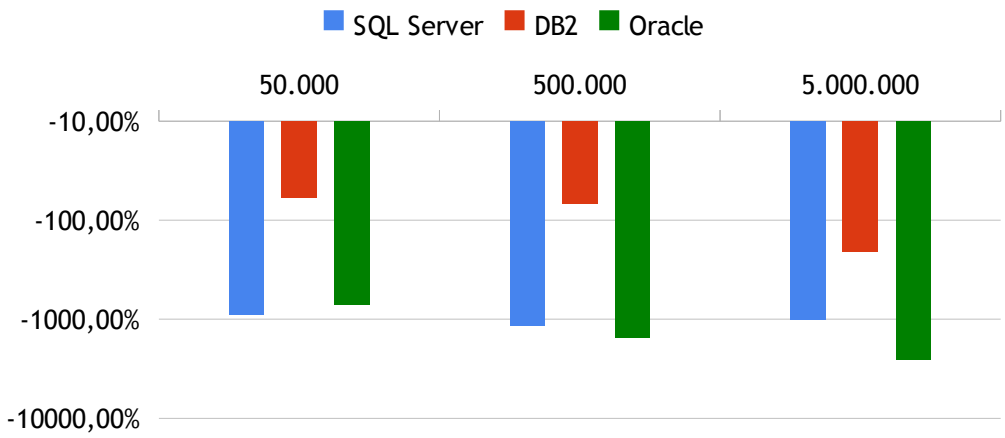
odtworzenie całego dokumentu.

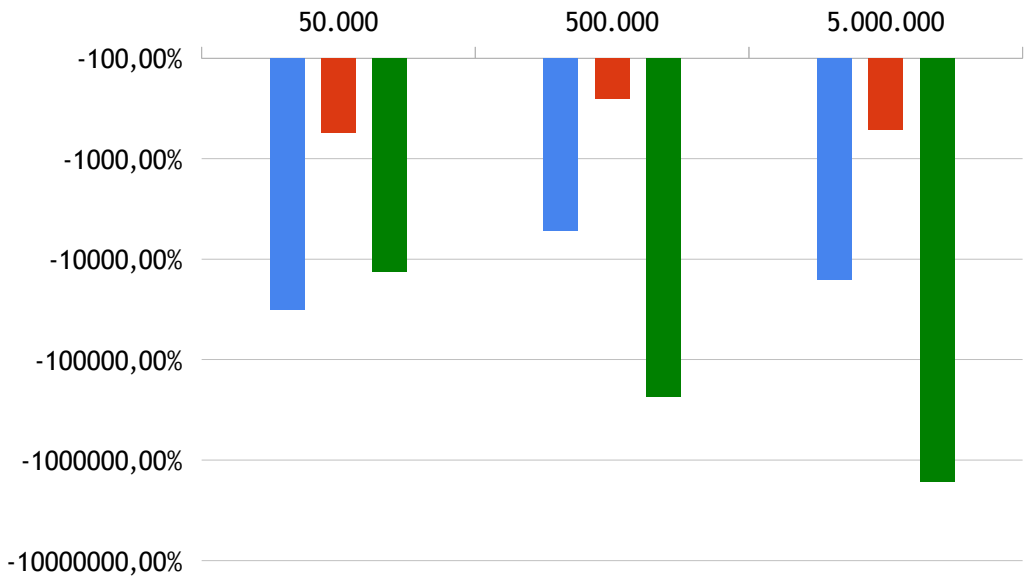
Zupełnie inaczej skonstruowany jest indeks XML w systemie DB2. Zawiera on jedynie informacje o wartości wybranych węzłów i pozwala na szybkie określenie czy dany dokument zawiera szukany węzeł czy nie. Wynik zapytania jest już odczytywany z oryginalnych danych a nie z indeksu (jak w przypadku SQL Server czy Oracle). Dlatego też rozmiar samego indeksu jest znacznie mniejszy i wynosi ok 60% rozmiaru danych XML. Wynik ten uzyskano indeksując wszystkie ścieżki składowanych dokumentów (wzorzec: /* ). W przypadku indeksowania jedynie wybranych ścieżek rozmiar indeksu jest proporcjonalnie mniejszy. Indeksy tworzone automatycznie: XML Region Index oraz XML Path Index są relatywnie małe – ich łączny rozmiar w zależności od liczby unikatowych ścieżek wynosi od jednego do kilkunastu procent rozmiaru indeksowanych danych. W przypadku typowych dokumentów XML nie przekracza kilku procent.

Kolejnym problemem związanym z użytkowaniem indeksów XML jest konieczność ich odbudowy po operacjach modyfikacji danych takich jak wstawanie czy usuwanie dokumentu XML bądź jego fragmentu. Poniżej zaprezentowano zestawienia obrazujące spadek wydajności operacji wstawiania i usuwania danych XML związany z koniecznością przebudowy indeksu XML. Wyniki uzyskano badając czas wstawiania oraz usuwania danych z tabel z nałożonym indeksem XML oraz bez nałożonego indeksu.

Wyniki zaprezentowano w podziale na średni rozmiar indeksu (oś pozioma) wyrażony liczbą wpisów indeksu (węzłów XML) w chwili jego przebudowy.

Szczegóły dotyczące przebiegu testu, wykorzystanych danych oraz zapytań dla poszczególnych grup wyników zawiera dodatek H.*Dodatek Specyfikacja testu H*.





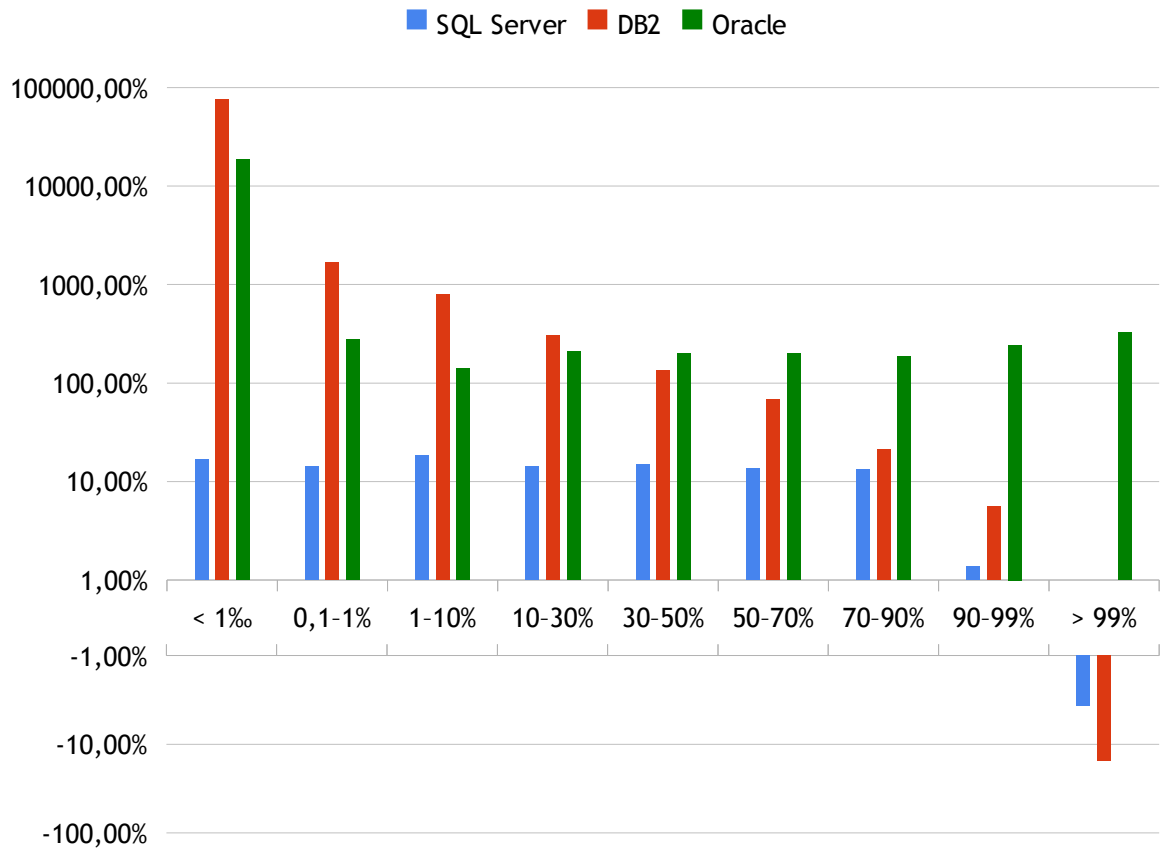

*Zestawienie 12: Spadek wydajności operacji wstawiania danych (górny wykres) oraz usuwania danych (dolny wykres) wynikający z zastosowania indeksu XML.*

Zgodnie z otrzymanymi wynikami najmniejsze koszty przebudowy indeksu charakteryzują serwer DB2. Dla większości testów spadek wydajności wstawiania danych wyniósł ok 60% natomiast usuwania danych ok 500%. Dodatkowo spadek ten jest w małym stopniu zależny od wielkości indeksu – duży rozmiar indeksu nie wpływa znacząco na czas jego odbudowy. Znacznie gorsze wyniki uzyskano dla pozostałych systemów. W SQL Server czas wstawiania danych XML jest przeciętnie 10-krotnie a usuwania 150-krotnie dłuższy gdy na tabelę nałożony jest indeks XML. W serwerze Oracle koszt przebudowy indeksu jest największy. Dodatkowo istnieje silna zależność pomiędzy wielkością indeksu a czasem jego aktualizacji.

Tak duże różnice wynikają ze sposobu implementacji indeksu XML. W przypadku serwera DB2 odbudowa indeksu sprowadza się jedynie do dodania bądź usunięcia wpisów indeksu w postaci B-drzewa. W serwerach Oracle oraz SQL Server przebudowa indeksu polega na aktualizacji tabeli ścieżek oraz przebudowaniu wszystkich indeksów dodatkowych utworzonych na tabeli. Dlatego też, w przypadku serwerów Oracle oraz SQL Server zaleca się wyłączenie indeksu na czas wstawiania bądź usuwania większej ilości danych XML.

O tym czy warto ponieść koszty utrzymania indeksów XML w głównej mierze decyduje stopień w jakim zwiększą one efektywność poszczególnych zapytań. Poniżej zaprezentowano wyniki testów, których celem było zbadanie efektywności indeksów XML poszczególnych SZBD. W testach wykorzystano dokumenty XML o łącznym rozmiarze ok 250MB wygenerowane specjalnie na potrzeby eksperymentów. Rozmiar pojedynczego dokumentu wynosił od 1kB do 2MB. Zapytania XQuery skonstruowano tak, aby z pojedynczego dokumentu XML zwracały nie więcej niż kilka pojedynczych węzłów. Końcowe wyniki uzyskano wywołując wielokrotnie każde z zapytań oraz uśredniając wyniki pojedynczych prób po odrzuceniu skrajnych. Wyniki zaprezentowano w podziale na selektywność (oś pozioma) wyrażoną stosunkiem liczby dokumentów zawierających szukane węzły XML do łącznej liczby dokumentów.

Szczegóły dotyczące przebiegu testu, wykorzystanych danych oraz zapytań dla poszczególnych grup wyników zawiera dodatek I.*Dodatek Specyfikacja testu I*




*Zestawienie 13: Wzrost wydajności zapytań XQuery wynikająca z zastosowania indeksu XML*

Przeprowadzone testy wykazały, że korzyści wynikające ze stosowania indeksów XML w systemie DB2 są największe ale bezpośrednio związane ze selektywnością zapytań. Efektywność zapytań o niskim współczynniku selektywności wzrastała po zastosowaniu indeksu nawet 1000-krotnie. Efektywność malała proporcjonalnie do wzrostu współczynnika. W szczególności, zapytania których wyniki pobierane były z wszystkich dokumentów przeszukiwanej tabeli wykonywane były wolniej na danych z utworzonym indeksem XML. Wynika to stąd, że rola indeksu w tym SZBD polega jedynie na wyborze dokumentów, które należy przeszukać. Na łączny czas złożyły się zatem czas przeskanowania wszystkich dokumentów (tak jak w przypadku danych nieindeksowanych) oraz czas przeskanowania indeksu XML. Wydajność serwera Oracle po zastosowaniu indeksu wzrosła przeciętnie o ok 200%.Wzrost wydajności zapytań do bazy SQL Server okazał się bardzo znikomy i dla większości przypadków wyniósł ok 20%.

# Rozdział V
# Podsumowanie pracy

Jednym z głównych celów niniejszej pracy było zweryfikowanie możliwości poszczególnych systemów zarządzania bazami danych pod kątem szeroko pojętej obsługi formatu danych XML. W części poświęconej temu zadaniu główny nacisk położono na kwestie związane z importowaniem, eksportowaniem oraz generowaniem dokumentów XML, bądź ich fragmentów, a także ze sposobem reprezentacji danych XML w bazach. W licznych przykładach wykazano sposób, w jaki poszczególne zadania realizowane są w każdym z omawianych systemów. Zamieszczone fragmenty kodu obrazują podstawowe różnice w implementacji tych zadań w zależności od dialektu języka SQL, którym posługuje się dany SZBD. Z przeprowadzonych testów jednoznacznie wynika, że omawiane serwery baz danych są przystosowane do przechowywania zarówno dokumentów XML o rozmiarach rzędu kilku kilobajtów jak również tych rzędu kilkuset megabajtów. Operacje wstawiania i pobierania z bazy kompletnych dokumentów działają bardzo sprawnie, podobnie jak operacje związane z kontrolą poprawności składniowej oraz zgodności z zadanym schematem XSD. W środowisku testowym będącym w dyspozycji autora pracy zmierzona wydajność importu danych XML wyniosła nawet 60 MB/sek. co wydaje się być wynikiem satysfakcjonującym, tym bardziej, że XML jako format tekstowy pozwala na reprezentację dużej ilości danych w stosunku do zajmowanej przestrzeni dyskowej. Ograniczenia dotyczące maksymalnej wielkości pojedynczego dokumentu nakładane przez serwery PostgreSQL - 1GB (po kompresji), SQL Server - 2GB i Oracle - 4GB zdają się nie mieć większego znaczenia, gdyż pliki XML o tak dużych rozmiarach są bardzo rzadko wykorzystywane a w razie konieczności mogą być w łatwy sposób podzielone na kilka mniejszych.

W tej części pracy omówione zostały również mechanizmy odpowiedzialne za tworzenie danych XML na podstawie danych składowanych w strukturach relacyjnych. Możliwości jakie oferują SZBD są w tej materii bardzo duże, zwłaszcza w systemach Oracle i DB2, nieco mniejsze w SQL Server. Ten ostatni jako jedyny z opisywanych nie korzysta ze specyfikacji SQL/XML. Wymusza tym samym na użytkownikach znajomość funkcji i konstrukcji językowych właściwych dla dialektu T-SQL, które w istotny sposób różnią się od standardowych. Choć rozwiązania zaproponowane przez twórców SQL Server są dość przejrzyste, intuicyjne i wygodne w użyciu to z pewnością przysporzą wielu kłopotów w przypadku konieczności migracji kodów SQL na inny serwer bazodanowy. Problemu takiego nie będzie natomiast w relacjach między serwerami Oracle, DB2 oraz PostgreSQL.

W kolejnym rozdziale analizie poddano funkcje pozwalające na wyszukiwanie oraz modyfikowanie danych XML za pomocą języków zapytań XPath, XQuery oraz SQL/XML.

Głównym nacisk położono tu na wydajność mechanizmów odpowiedzialnych za przetwarzanie zapytań, stąd też w rozdziale tym zaprezentowane zostały wyniki licznych testów porównawczych przeprowadzonych przez autora pracy. W pierwszym z nich rozpatrywano zapytania selektywne XPath, przy pomocy których wyszukiwano fragmenty danych XML. Wyniki zamieszczone w zestawieniu 5, str. 46 potwierdzają wysoką wydajność biblioteki *libxml2* zastosowanej do przetwarzania zapytań XPath w serwerze PostgreSQL. Okazała się ona średnio o 50 – 100% bardziej efektywna w stosunku do komponentów wykorzystanych w pozostałych badanych SZBD. W tym samym teście wykazano niezwykłą czułość systemu SQL Server na operatory wieloznaczne – w niektórych przypadkach ich stosowanie okazało się katastrofalne w skutkach i przyczyniało się do nawet 100-krotnego pogorszenia wydajności. Dlatego też, o ile to możliwe, podczas konstruowania zapytań XPath należy postępować z zaleceniami producenta – dokumentacja SQL Server dość precyzyjnie wskazuje, których operatorów i konstrukcji należy unikać aby zapobiec opisanym, niekorzystnym skutkom.

W dalszej części omówiono zapytania XQuery, za pomocą których możliwa jest modyfikacja wybranych fragmentów danych XML. Możliwości serwera PostgreSQL mocno odbiegają w tym względzie od pozostałych trzech – nie ma tu bowiem sposobu na łatwą ingerencję w wybrany fragment danych. Dane mogą być co prawda modyfikowane ale wymaga to pobrania całego dokumentu i operowania na nim jak na danych tekstowych. Choć uzyskane wyniki okazały się najlepsze w systemach Oracle i SQL Server, to należy zaznaczyć, że serwer DB2 jako jedyny z badanych okazał się bardzo stabilny i niewrażliwy na zapytania zawierające operatory wieloznaczne.

Na zestawieniu 7, str. 54 zamieszczono wyniki eksperymentów mających na celu zbadanie wydajności zapytań w języku XPath i XQuery na tle konstrukcji SQL odnoszących się do jednakowych danych. Otrzymane wyniki jednoznacznie wskazują, że w przypadku ogólnym, gdy oczekiwana jest wysoka efektywność w dostępie do danych, typ XML nie stanowi rozsądnej alternatywy dla danych przechowywanych w modelu relacyjnym. We wszystkich rozpatrywanych SZBD rozwiązania oparte na tabelach relacyjnych okazały się średnio 20-50 razy bardziej wydajne. Wynik taki jest zgodny z oczekiwaniami autora, tym bardziej, że typ XML z założenia ma poszerzać możliwości baz danych a nie konkurować z dotychczasowymi rozwiązaniami. Jak wynika z testów, których wyniki zamieszczono na zestawieniu 8, str. 59 wykorzystanie typu XML może mieć jednak uzasadnienie w szczególnych przypadkach, zwłaszcza, gdy ma posłużyć do reprezentacji danych o strukturze

hierarchicznej, drzewiastej. Zastosowanie takie zdaje się być trafne nie tylko ze względu na lepszą wydajność – reprezentacja danych hierarchicznych w postaci XML jest bardziej naturalna, przejrzysta, a dodatkowo zapytania w wykorzystaniem XPath/XQuery są bardziej czytelne niż zapytania rekursywne SQL.

W rozdziale IV omówione zostały mechanizmy indeksowania danych XML mające na celu skrócenie czasu wyszukiwania poszczególnych elementów. Z przeprowadzonych testów wynika, że jedynie rozwiązanie zastosowane w serwerze DB2 spełnia pokładane w nim nadzieje – w przypadku, gdy dane zostaną odpowiednio podzielone a szukany jest jedynie nieduży fragment danych indeks daje rzeczywiste korzyści w postaci znacznego polepszenia wydajności zapytań XPath/ XQuery. Dodatkowo, jako jedyny z przedstawionych pozwala precyzyjnie określić obszary, które mają zostać zindeksowane, co w zdecydowany sposób ogranicza wymaganą przez indeks przestrzeń dyskową. Ponadto, czas wymagany na jego przebudowę po operacjach modyfikacji danych jest stosunkowo nieduży – podczas wstawiania nowych danych do bazy z utworzonym indeksem XML zaobserwowano spadek wydajności o 50%. Całkowicie odmienne wyniki uzyskano badając koszty utrzymania indeksów w systemach Oracle i SQL Server - w większości przypadków odnotowano blisko 100-krotny spadek wydajności. W pojedynczych scenariuszach, gdy rozmiar indeksu był dość znaczny jego przebudowa powodowała całkowity paraliż w dostępie do danych XML. Nie bez znaczenia jest także wielkość samej struktury – utworzenie indeksu zwiększa o ponad 600% rozmiar danych XML. Wszystkie te niedogodności w zestawieniu ze znikomym wzrostem wydajności zapytań sprawiają, że ogólna ocena indeksów XML w systemach Oracle i SQL Server wypada zdecydowanie negatywnie. Decydując się zatem na ich zastosowanie w którymś z dwóch wymienionych SZBD należy zwrócić szczególną uwagę na zalecenia producentów oraz pamiętać, aby dezaktywować indeks na czas operacji modyfikacji większej ilości danych.

Bez wątpienia natywny typ XML w systemach zarządzania bazami danych otwiera wiele nowych, ciekawych możliwości. W niniejszej pracy szczegółowo omówiona została jedynie część z nich. Główny nacisk położono na te zagadnienia, które związane były ze wstępnie postawionymi zadaniami. W pracy udało się zrealizować większość z nich, choć z pewnością wiele obszarów nie zostało jeszcze wyczerpanych. Ze względu na rozległość omawianej materii pominięte zostały m. in. możliwość serwerów DB2, Oracle i SQL Server związane z obsługą schematów XSD oraz przekształceń XSLT. Nie uwzględniono także kwestii bezpieczeństwa i sterowania dostępem do wybranych dokumentów XML, bądź ich

fragmentów. Interesujące, zwłaszcza z punktu widzenia programistów wydają się także liczne rozwiązania związane z integracją typów i funkcji XML z interfejsami API popularnych platform programistycznych takich jak JAVA czy .NET.

Tworzenie niniejszej pracy było zadaniem ciekawym, wymagającym, które w sposób zasadniczy poszerzyło wiedzę jej autora – zarówno w zakresie zagadnień szczegółowo omówionych, jak i tych, dla których zabrakło miejsca na papierze. Testowanie i eksperymentowanie z danymi XML, choć często trudne i czasochłonne, przyniosło autorowi pracy dużo zadowolenia i satysfakcji.

# BIBLIOGRAFIA


[1] Michael Coles. *Pro SQL Server 2008 XML*. Apress, 2008.

[2] Matthias Nicola, Bert van der Linden, *Native XML Support in DB2 Universal Database, 2005*

[3] Whei-Jen Chen, Art Sammartino, Dobromir Goutev, Felicity Hendricks, Ippei Komi, Ming-Pang Wei, Rav Ahuja, Matthias Nicola. *DB2 9 pureXML Guide*. IBM Redbooks, 2007

[4] Drew Adams i inni. *Oracle XML DB Developer's Guide, 11g Release 1 (11.1)*. Oracle Corporation, 2008

[5] Mark Drake, *New Features in Oracle XML DB for Oracle Database 11g Release 1,* 2007. Oracle Corporation, 2007

[6] PostgreSQL 8.4.3 Documentation, Chapter 53. Database Physical Storage.

http://www.postgresql.org/docs/8.4/static/storage-toast.html (2010-04-04)

[7] PostgreSQL 8.4.3 Documentation, Appendix D. SQL Conformance
http://www.postgresql.org/docs/8.4/interactive/unsupported-features-sql-standard.html (2010-04-05)

[8] World Wide Web Consortium. *Extensible Markup Language (XML) 1.0.* http://www.w3.org/TR/xml (2009-06-14)

[9] World Wide Web Consortium. *XML Path Language (XPath).* http://www.w3.org/TR/xpath (2009-08-20)

[10] World Wide Web Consortium. *XML Path Language (XPath) 2.0.* http://www.w3.org/TR/xpath20 (2009-08-21)

[11] World Wide Web Consortium. *XQuery 1.0 and XPath 2.0 Data Model (XDM).* http://www.w3.org/TR/xpath-datamodel (2009-08-21)

[12] World Wide Web Consortium. *XQuery 1.0 and XPath 2.0 Functions and Operators.* http://www.w3.org/TR/xquery-operators (2009-08-21)

[13] World Wide Web Consortium. *XQuery Update Facility 1.0.*

http://www.w3.org/TR/xqupdate (2009-09-10)

[14] Matthias Nicola, Uttam Jain. *Update XML in DB2 9.5.* IBM developerWorks, 2007. http://www.ibm.com/developerworks/data/library/techarticle/dm-0710nicola (2009-08-30)

# Zawartość nośnika DVD

- Praca magisterska w formacie PDF
- Zestawy zawierające dane i zapytania testowe wykorzystane podczas prowadzenia testów i eksperymentów (informacje dotyczące sposobu wykorzystania danych zawartych w poszczególnych zestawach zawarte zostały w dodatkach A.- I.)

# A. Dodatek Specyfikacja testu A

**Cel testu:**

Zbadanie szybkości konwersji danych tekstowych do natywnego typu XML w poszczególnych systemach zarządzania bazami danych.

**Przebieg testu:**

W teście importowano dokumenty XML wykorzystując właściwe dla każdego z badanych SZBD funkcje systemowe pozwalające na import danych z zewnętrznych plików oraz konwersję do natywnego typu XML. Końcowe rezultaty uzyskano metodą wielokrotnego powtarzania testów i uśredniania wyników pojedynczych prób po odrzuceniu skrajnych. Testy prowadzono w podziale na dwie grupy plików – do rozmiaru 1 MB oraz od 1 MB do 65 MB. W testach wykorzystano następujące funkcje systemowe SZBD:

**PostgreSQL**

Funkcja: **`pg_read_file`**

Przykład wykorzystania:

```
INSERT INTO test(id,doc) VALUES(1, CAST(PG_READ_FILE('f.xml', 0, 10000000) AS XML))
```

**SQL Server**

Funkcja: **`openrowset`**

Przykład wykorzystania:

```
INSERT INTO test(id,doc) SELECT 1, x.* FROM OPENROWSET( BULK 'f.xml', SINGLE_BLOB) AS x
```

**DB2**

Funkcja: **`import`**

Przykład wykorzystania:

```
IMPORT FROM 'xmldir\files.del' OF DEL XML FROM 'xmldir' INSERT INTO test
```

Przykładowa zawartość pliku *'xmldir\files.del':*

*0, filename, <XDS FIL='f.xml' />*

### Oracle

Funkcja: **`bfilename`**

Przykład wykorzystania:

**INSERT INTO** test **VALUES** (1, XMLType(**BFILENAME**('***DIR***', '***f.xml***'), nls_charset_id('UTF8')))

### Wykorzystane zapytania:

Do utworzenia tabel z kolumną typu XML wykorzystano skrypt:

ROOT/Zestawy_danych/zestaw_1/Zapytania/SQL/*DBNAME*[19]/01_create_tableXML.sql

### Wykorzystane dane:

dla grupy plików o rozmiarze od 1 KB do 1 MB

ROOT[20]/Zestawy_testowe/Zestaw_1/Dane/XML/*.xml

dla grupy plików o rozmiarze od 1 MB do 65MB

ROOT/Zestawy_testowe/Zestaw_2/Dane/XML/*.xml

19 DBNAME – nazwa SZBD, do którego odnosi się dany plik bądź katalog. Możliwe wartości: Oracle, DB2, PostgreSQL, SQL Server

20 ROOT – katalog główny nośnika DVD dołączonego do pracy

# B. Dodatek Specyfikacja testu B

**Cel testu:**

Zbadanie wymaganej przestrzeni dyskowej dla danych typu XML w poszczególnych systemach zarządzania bazami danych.

**Przebieg testu:**

W teście porównywano rozmiar danych XML przed i po zaimportowaniu do bazy. Rozmiar zaimportowanych danych mierzono przy pomocy funkcji systemowych właściwych dla każdego z badanych systemów zarządzania bazami danych. Dane zaimportowano do bazy za pomocą funkcji opisanych w dodatku *A.Dodatek Specyfikacja testu A*.

**Wykorzystane zapytania:**

Do utworzenia tabel z kolumną typu XML wykorzystano skrypt:

ROOT/Zestawy_danych/zestaw_1/Zapytania/SQL/*DBNAME*/01_create_tableXML.sql

**Wykorzystane dane:**

ROOT/Zestawy_testowe/Zestaw_2/Dane/XML/*.xml

ROOT/Zestawy_testowe/Zestaw_6/Dane/XML/*.xml

# C. Dodatek Specyfikacja testu C

**Cel testu:**

Porównanie szybkości wyszukiwania elementów XML przy pomocy zapytań XPath.

**Przebieg testu:**

W teście porównywano wydajność poszczególnych systemów zarządzania bazami danych w zakresie wyszukiwania elementów XML przy pomocy zapytań XPath. Końcowe rezultaty uzyskano metodą wielokrotnego powtarzania testów i uśredniania wyników pojedynczych prób po odrzuceniu skrajnych. Testy prowadzono w podziale na dwie grupy zapytań – zapytania dla danych o dobrze znanej strukturze (bez znaków wieloznacznych) oraz zapytania dla danych o strukturze częściowo znanej (zawierające znaki wieloznaczne).

**Wykorzystane dane:**

ROOT/Zestawy_testowe/Zestaw_1/Dane/XML/*.xml

ROOT/Zestawy_testowe/Zestaw_2/Dane/XML/*.xml

**Wykorzystane zapytania:**

Do utworzenia tabel z kolumną typu XML wykorzystano skrypt:

ROOT/Zestawy_danych/zestaw_1/Zapytania/SQL/*DBNAME*/01_create_tableXML.sql

Zapytania XPath:

(bez znaków wieloznacznych)

ROOT/Zestawy_testowe/Zestaw_3/Zapytania/XPath/DB2/q1.sql

ROOT/Zestawy_testowe/Zestaw_3/Zapytania/XPath/Oracle/q1.sql

ROOT/Zestawy_testowe/Zestaw_3/Zapytania/XPath/SQL Server/q1.sql

ROOT/Zestawy_testowe/Zestaw_3/Zapytania/XPath/PostgreSQL/q1.sql

(zawierające znaki wieloznaczne)

ROOT/Zestawy_testowe/Zestaw_3/Zapytania/XPath/DB2/q2.sql

ROOT/Zestawy_testowe/Zestaw_3/Zapytania/XPath/Oracle/q2.sql

ROOT/Zestawy_testowe/Zestaw_3/Zapytania/XPath/SQL Server/q2.sql

ROOT/Zestawy_testowe/Zestaw_3/Zapytania/XPath/PostgreSQL/q2.sql

# D. Dodatek Specyfikacja testu D

**Cel testu:**

Porównanie szybkości wstawiania, usuwania oraz modyfikacji fragmentów danych XML przy pomocy zapytań XQuery.

**Przebieg testu:**

W teście porównywano wydajność poszczególnych systemów zarządzania bazami danych w zakresie dodawania, usuwania oraz modyfikowania elementów XML przy pomocy zapytań XQuery. Końcowe rezultaty uzyskano metodą wielokrotnego powtarzania testów i uśredniania wyników pojedynczych prób po odrzuceniu skrajnych. Testy prowadzono w podziale na trzy grupy zapytań:

- zapytania wstawiające dane do dokumentów XML
- zapytania usuwające dane z dokumentów XML
- zapytania modyfikujące dokumenty XML

**Wykorzystane dane:**

ROOT/Zestawy_testowe/Zestaw_1/Dane/XML/*.xml

ROOT/Zestawy_testowe/Zestaw_2/Dane/XML/*.xml

**Wykorzystane zapytania:**

Do utworzenia tabel z kolumną typu XML wykorzystano skrypt:

ROOT/Zestawy_danych/zestaw_1/Zapytania/SQL/*DBNAME*/01_create_tableXML.sql

Zapytania XQuery:

**(**zapytania wstawiające dane do dokumentów XML)

ROOT/Zestawy_testowe/Zestaw_4/Zapytania/XQuery/Oracle/q1.sql

ROOT/Zestawy_testowe/Zestaw_4/Zapytania/XQuery/DB2/q1.sql

ROOT/Zestawy_testowe/Zestaw_4/Zapytania/XQuery/SQL Server/q1.sql

**(**zapytania usuwające dane z dokumentów XML)

ROOT/Zestawy_testowe/Zestaw_4/Zapytania/XQuery/Oracle/q2.sql

ROOT/Zestawy_testowe/Zestaw_4/Zapytania/XQuery/DB2/q2.sql

ROOT/Zestawy_testowe/Zestaw_4/Zapytania/XQuery/SQL Server/q2.sql

**(**zapytania modyfikujące dokumenty XML)

ROOT/Zestawy_testowe/Zestaw_4/Zapytania/XQuery/Oracle/q3.sql

ROOT/Zestawy_testowe/Zestaw_4/Zapytania/XQuery/DB2/q3.sql

ROOT/Zestawy_testowe/Zestaw_4/Zapytania/XQuery/SQL Server/q3.sql

# E. Dodatek Specyfikacja testu E

**Cel testu:**

Zbadanie wydajności zapytań SQL oraz XPath/XQuery do danych strukturalnych reprezentowanych w postaci tabel relacyjnych oraz danych typu XML w serwerach bazodanowych PostgreSQL, SQL Server, DB2 oraz ORACLE.

**Przebieg testu:**

W teście porównywano wydajność zapytań SQL oraz XPath/XQuery w poszczególnych systemach zarządzania bazami danych. Końcowe rezultaty uzyskano metodą wielokrotnego powtarzania testów i uśredniania wyników pojedynczych prób po odrzuceniu skrajnych. Testy prowadzono w podziale na dwie grupy zapytań (Q1 oraz Q2).

**Wykorzystane dane:**

Do utworzenia struktury tabel danych relacyjnych wykorzystano skrypt:

ROOT/Zestawy_danych/zestaw_5/Dane/SQL/01_create_table[_*DBNAME*].sql

Do utworzenia struktury tabel danych XML wykorzystano skrypt:

ROOT/Zestawy_danych/zestaw_5/Dane/SQL/01_create_tableXML[_*DBNAME*].sql

Do zasilenia tabel relacyjnych danymi wykorzystano skrypty:

ROOT/Zestawy_danych/zestaw_5/Dane/SQL/02_insert_wydawnictwa[_*DBNAME*].sql

ROOT/Zestawy_danych/zestaw_5/Dane/SQL/03_insert_gatunki.sql

ROOT/Zestawy_danych/zestaw_5/Dane/SQL/04_insert_autorzy.sql

ROOT/Zestawy_danych/zestaw_5/Dane/SQL/05_insert_ksiazki[_*DBNAME*].sql

ROOT/Zestawy_danych/zestaw_5/Dane/SQL/06_insert_ksiazkiautorzy.sql

ROOT/Zestawy_danych/zestaw_5/Dane/SQL/07_insert_opinions.sql

ROOT/Zestawy_danych/zestaw_5/Dane/SQL/08_insert_etykiety.sql

Do zasilenia tabeli z kolumną typu XML wykorzystano dokumenty XML dostępne w lokalizacji:

ROOT/Zestawy_danych/zestaw_5/Dane/XML/*.xml

Do utworzenia indeksów XML na bazach Oracle, DB2 oraz SQL Server wykorzystano skrypty:

ROOT/Zestawy_danych/zestaw_5/Dane/SQL/11_create_XMLIndex_*DBNAME*.sql

**Wykorzystane zapytania:**

dla danych w postaci relacyjnej – Q1:

ROOT/Zestawy_danych/zestaw_5/Zapytania/SQL/q1.sql

ROOT/Zestawy_danych/zestaw_5/Zapytania/SQL/q2.sql

ROOT/Zestawy_danych/zestaw_5/Zapytania/SQL/q4.sql

ROOT/Zestawy_danych/zestaw_5/Zapytania/SQL/q5.sql

ROOT/Zestawy_danych/zestaw_5/Zapytania/SQL/q6.sql

dla danych w postaci relacyjnej – Q2:

ROOT/Zestawy_danych/zestaw_5/Zapytania/SQL/q3.sql

dla danych w postaci XML – Q1:

ROOT/Zestawy_danych/zestaw_5/Zapytania/XPath_XQuery/*DBNAME*/q1.sql

ROOT/Zestawy_danych/zestaw_5/Zapytania/XPath_XQuery/*DBNAME*/q2.sql

ROOT/Zestawy_danych/zestaw_5/Zapytania/XPath_XQuery/*DBNAME*/q4.sql

ROOT/Zestawy_danych/zestaw_5/Zapytania/XPath_XQuery/*DBNAME*/q5.sql

ROOT/Zestawy_danych/zestaw_5/Zapytania/XPath_XQuery/*DBNAME*/q6.sql

dla danych w postaci XML – Q2:

ROOT/Zestawy_danych/zestaw_5/Zapytania/XPath_XQuery/DBNAME/q3.sql

# F. Dodatek Specyfikacja testu F

**Cel testu:**

Zbadanie wydajności zapytań SQL, XPath/XQuery oraz SQL/XML do danych o strukturze hierarchicznej reprezentowanych postaci tabel relacyjnych, danych typu XML oraz z wykorzystaniem połączenia obydwu technik. Testy dla zapytań SQL, XPath/XQuery przeprowadzono na serwerach bazodanowych PostgreSQL, SQL Server, DB2 oraz ORACLE. Testy dla zapytań SQL/XML przeprowadzono na serwerach bazodanowych SQL Server, DB2 oraz ORACLE.

**Przebieg testu:**

W teście porównywano wydajność zapytań SQL, XPath/XQuery oraz SQL/XML w poszczególnych systemach zarządzania bazami danych. Końcowe rezultaty uzyskano metodą wielokrotnego powtarzania testów i uśredniania wyników pojedynczych prób po odrzuceniu skrajnych. Testy prowadzono w podziale na dwie grupy zapytań (Q1 oraz Q2).

**Wykorzystane dane:**

Do utworzenia struktury tabel wykorzystano skrypt:

ROOT/Zestawy_danych/zestaw_6/Dane/SQL/00_create_table[_*DBNAME*].sql

Do zasilenia tabel z danymi relacyjnymi wykorzystano skrypty:

ROOT/Zestawy_danych/zestaw_6/Dane/SQL/01_insert_kategorie.sql

ROOT/Zestawy_danych/zestaw_6/Dane/SQL/02_insert_produkty.sql

Do zasilenia tabli z kolumną XML zawierającą całość danych w postaci XML wykorzystano dokument:

ROOT/Zestawy_danych/zestaw_6/Dane/XML/kategorie_produkty.xml

Do zasilenia tabli z kolumną XML zawierającą jedynie hierarchię kategorii w postaci XML wykorzystano dokument:

ROOT/Zestawy_danych/zestaw_6/Dane/XML/kategorie.xml

Do utworzenia indeksów XML na bazach Oracle oraz SQL Server wykorzystano skrypty:

ROOT/Zestawy_danych/zestaw_6/Dane/SQL/03_create_XMLIndex_*DBNAME*.sql

**Wykorzystane zapytania:**

dla danych w postaci relacyjnej – Q1:

ROOT/Zestawy_danych/zestaw_6/Zapytania/SQL/*DBNAME*/q1.sql

dla danych w postaci relacyjnej – Q2:

ROOT/Zestawy_danych/zestaw_6/Zapytania/SQL/*DBNAME*/q1.sql

dla danych w postaci hierarchicznej – Q1:

ROOT/Zestawy_danych/zestaw_6/Zapytania/XPath-XQuery/*DBNAME*/q1.sql

dla danych w postaci hierarchicznej – Q2:

ROOT/Zestawy_danych/zestaw_6/Zapytania/XPath-XQuery/*DBNAME*/q1.sql

dla danych reprezentowanych jako połączenie danych relacyjnych i danych XML – Q1:

ROOT/Zestawy_danych/zestaw_6/Zapytania/SQL-XML/*DBNAME*/q1.sql

dla danych reprezentowanych jako połączenie danych relacyjnych i danych XML – Q2:

ROOT/Zestawy_danych/zestaw_6/Zapytania/SQL-XML*DBNAME*/q1.sql

# G. Dodatek Specyfikacja testu G

**Cel testu:**

Zbadanie przestrzeni dyskowej wymaganej przez poszczególne SZBD do składowania danych oraz
indeksów XML.

**Przebieg testu:**

Zestawienia utworzono badając średni rozmiar indeksów utworzonych na danych XML o łącznym rozmiarze od 250 KB do 250 MB. Rozmiar danych oraz indeksów XML mierzono przy pomocy funkcji systemowych właściwych dla każdego z badanych serwerów bazodanowych.

**Wykorzystane zapytania:**

Do utworzenia tabel z kolumną typu XML wykorzystano skrypt:

ROOT/Zestawy_danych/zestaw_1/Zapytania/SQL/*DBNAME*/01_create_tableXML.sql

do utworzenia indeksów XML wykorzystano następujące skrypty:

ROOT/Zestawy_danych/zestaw_7/Zapytania/00_create_XMLIndex_*DBNAME*.sql

**Wykorzystane dane:**

ROOT/Zestawy_danych/zestaw_7/Dane/XML/f1.xml

ROOT/Zestawy_danych/zestaw_7/Dane/XML/f2.xml

ROOT/Zestawy_danych/zestaw_7/Dane/XML/f3.xml

ROOT/Zestawy_danych/zestaw_7/Dane/XML/f4.xml

# H. Dodatek Specyfikacja testu H

**Cel testu:**

Zbadanie spadku wydajności operacji wstawiania i usuwania danych XML związanego z koniecznością przebudowy indeksu XML.

**Przebieg testu:**

Na potrzeby testu utworzono dwie tabele z kolumną typu XML o jednakowej strukturze. Na jednej z tabel utworzono indeks XML. Do tak utworzonych tabel importowano jednakowe porcje danych XML. W teście mierzono różnicę w czasie wykonania operacji na tabeli z utworzonym indeksem XML oraz na tabeli bez indeksu. Następnie z obu tabel usuwano jednakowe porcje danych XML ponownie badając różnice w czasach operacji. Do importu danych XML wykorzystano systemowe funkcje dostępne w każdym z opisywanych SZBD – wymienione zostały w dodatku *A.Dodatek Specyfikacja testu A*. Wyniki zaprezentowano w podziale na średnią liczbę wpisów w indeksie XML w czasie jego przebudowy.

**Wykorzystane zapytania:**

do utworzenia tabel z kolumną typu XML wykorzystano następujące skrypty:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/00_create_tableXML.sql

do utworzenia indeksów XML wykorzystano następujące skrypty:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/01_create_XMLIndex.sql

**Wykorzystane dane:**

dla średniej liczby wpisów indeksu równej 50.000:

ROOT/Zestawy_danych/zestaw_8/Dane/XML/klienci00*.xml

dla średniej liczby wpisów indeksu równej 500.000:

ROOT/Zestawy_danych/zestaw_8/Dane/XML/klienci0*.xml

dla średniej liczby wpisów indeksu równej 5.000.000:

ROOT/Zestawy_danych/zestaw_8/Dane/XML/klienci*.xml

# I. Dodatek Specyfikacja testu I

**Cel testu:**

Zbadanie efektywności indeksów XML w poszczególnych systemach zarządzania bazami danych.

**Przebieg testu:**

Na potrzeby testu utworzono dwie tabele z kolumną typu XML o jednakowej strukturze. Na jednej z tabel utworzono indeks XML. Do tak utworzonych tabel zaimportowano jednakowy komplet dokumentów XML. Następnie w obydwu tabelach wyszukiwano jednakowe dane mierząc różnicę w czasie wykonania operacji.

**Wykorzystane zapytania:**

do utworzenia tabel z kolumną typu XML wykorzystano następujące skrypty:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/00_create_tableXML.sql

do utworzenia indeksów XML wykorzystano następujące skrypty:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/01_create_XMLIndex.sql

zapytania dla grupy zapytań o selektywności < 1‰:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/q1.sql

zapytania dla grupy zapytań o selektywności 0,1 – 1%:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/q2.sql

zapytania dla grupy zapytań o selektywności 1 – 10%:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/q3.sql

zapytania dla grupy zapytań o selektywności 10 – 30%:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/q4.sql

zapytania dla grupy zapytań o selektywności 30 – 50%:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/q5.sql

zapytania dla grupy zapytań o selektywności 50 – 70%:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/q6.sql

zapytania dla grupy zapytań o selektywności 70 – 90%:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/q7.sql

zapytania dla grupy zapytań o selektywności 90 – 99%:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/q8.sql

zapytania dla grupy zapytań o selektywności > 99%:

ROOT/Zestawy_danych/zestaw_8/Zapytania/*DBNAME*/q9.sql

**Wykorzystane dane:**

ROOT/Zestawy_danych/zestaw_8/Dane/XML/*.xml