Projektowanie baz danych XML. Vademecum profesjonalisty

Autor: Mark Graves
Tłumaczenie: Tomasz Żmijewski
ISBN: 83-7197-667-4
Tytuł oryginału: Designing XML Databases
Format: B5, stron: 498

„Projektowanie baz danych XML. Vademecum profesjonalisty” — to obszerny podręcznik do nauki baz danych XML, wykorzystywanych w Internecie oraz baz stanowiących część większych systemów.

Jeśli dysponujesz gotową bazą danych obsługującą XML, to dzięki tej książce poznasz szczegółowe techniki, w pełni wykorzystujące tę bazę. Jeśli natomiast korzystasz z klasycznych relacyjnych baz danych, nauczysz się tworzyć aplikacje z wykorzystaniem XML. Zainteresowani tworzeniem baz danych XML „od zera”, dowiedzą się jak w pełni wykorzystać dostępne narzędzia.

Dodatkowo autor omawia:

- Najważniejsze techniki projektowe baz danych, systemów obsługujących te bazy oraz aplikacji XML
- Przechowywanie danych XML w bazach obiektowych, relacyjnych i opartych na plikach płaskich
- Zaawansowane techniki modelowania danych XML
- Zapytania kierowane do baz danych XML (uwagi praktyczne, techniki stosowania JDBC oraz podstawy teoretyczne)
- Sposób korzystania z sieciowych baz danych XML za pomocą języka XSL i języka Java
- Architekturę baz danych XML i specjalizowane indeksy
- Włączanie baz danych XML do większych systemów
- Bazy danych XML i ich zastosowanie w nauce

„Projektowanie baz danych XML. Vademecum profesjonalisty” to podstawowe źródło informacji dla projektantów i programistów baz danych, twórców aplikacji XML, projektantów systemów oraz kierowników projektów — szczególnie w środowiskach o specyficznych wymaganiach.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 5.

Architektura systemu baz danych

Język XML świetnie nadaje się do wymiany danych, dlatego też często wykorzystujemy go w komunikacji między systemami. Pojęcia „architektura systemu baz danych" używamy w odniesieniu do sposobu, w jaki aplikacje i użytkownicy korzystają z danych oraz zarządzają nimi. W przypadku baz danych XML użytkownicy i aplikacje muszą załadować dane do bazy, przekształcić je na dokumenty XML, pobrać XML z bazy i powiązać dane relacyjne z kodem XML. Architektura systemu baz danych może być typu klient-serwer — wtedy aplikacja klienta współpracuje bezpośrednio z bazą danych albo może to być struktura trzywarstwowa, w której między klientem a bazą danych pojawia się dodatkowo serwer.

5.1. Architektura systemu

Architektura systemu to sposób funkcjonowania systemu i jego poszczególnych modułów. Każdy *moduł* jest komponentem systemu i realizuje powiązane ze sobą funkcje. Dobrą architekturę systemu tworzy się przez odpowiednie pogrupowanie wymagań funkcjonalnych w moduły i łączenie modułów w system, który spełnia wszystkie założone zadania. Definiując architekturę systemu, należy odpowiedzieć na następujące pytania:

- Ile jest modułów?
- Jak są ze sobą powiązane? (Czy liniowo, czy w drzewo albo graf?)
- Na czym polega ich działanie? Czyli jakie są funkcje poszczególnych modułów?

Z systemami spotykamy się wszędzie — poznawanie większości dziedzin naukowych wiąże się ze studiowaniem konkretnych systemów. Fizyka bryły sztywnej opisuje, jak należące do systemu obiekty oddziałują na siebie, a w przypadku biologii dokładnie analizuje się systemy o „organicznej" naturze. Inżynieria polega na tworzeniu złożonych systemów o założonych funkcjach. W budownictwie przez termin „architektura" rozumie się projekt budynku, jego wygląd i sposób funkcjonowania.

Architektura systemu to obszerne zagadnienie związane z projektowaniem komponentów, które współpracując ze sobą, realizują całościowe zadania. W tej książce będziemy mówić o architekcie systemu, którego zadanie polega na zaprojektowaniu deterministycznych komponentów programowych baz danych XML.

Duże systemy zwykle mają hierarchiczną budowę: większe podsystemy zawierają mniejsze, a te z kolei tworzone są z jeszcze mniejszych podsystemów, i tak dalej, aż do pakietów, które zawierają podstawowe składniki programów, czyli klasy obiektów, funkcje i procedury. W tym rozdziale zajmiemy się abstrakcyjnymi aspektami projektowania i podziałem pracy na różne maszyny dostępne w Sieci.

Pierwsze systemy baz danych były *monolityczne*. Obsługiwała je jedna stacja robocza. Użytkownicy korzystali z systemu za pomocą prostych terminali lub kart perforowanych. Z czasem, kiedy stacje robocze zaczęły mieć coraz większą moc obliczeniową, część obliczeń i funkcji została wydzielona z monolitycznego systemu i przeniesiona na stacje robocze (klientów), w architekturze *klient-serwer*. Podsystem klienta przeprowadzał formatowanie i proste przetwarzanie, zaś serwer zarządzał większością danych. W miarę jak powiększały się bazy danych i włączano je do systemów korporacyjnych, konieczne stało się zastosowanie architektury *trzywarstwowej*. Dzięki zastosowaniu odrębnego serwera uproszczono dostęp klientów do wielu baz danych. Warstwa pośrednia umożliwiła aplikacjom klienta łączenie się z wieloma bazami danych przez jednolity interfejs. Jednocześnie projektanci starali się znaleźć odpowiednie miejsce dla *reguł biznesowych*, które opisują zasady użycia danych. Przetwarzanie reguł mogło przecież „zatkać" serwer lub wymagało powtórzenia tych operacji na stacjach klientów, szczególnie w dużych systemach, w których wiele aplikacji przetwarza reguły. Jeśli reguły biznesowe znajdowały się w warstwie pośredniej, to różne aplikacje, nie troszcząc się o wydajność czy utrzymanie serwera baz danych, mogły z nich korzystać. Wzrost liczby baz danych i aplikacji przyczynia się do tego, że konieczne staje się stosowanie dodatkowych warstw pośrednich, a co z tym związane — zmiana architektury trzywarstwowej na *wielowarstwową*.

Sieciowy system baz danych zawiera — jako jeden ze swoich modułów — serwer sieciowy. Do komunikacji między poszczególnymi warstwami używa się protokołu nośnego sieci, na przykład `http`. Sieciowy system baz danych może mieć architekturę klient-serwer, architekturę trzywarstwową lub wielowarstwową. Najczęściej stosuje się rozwiązanie trzywarstwowe — w takim wypadku baza danych wysyła dane do serwera sieciowego. Jeśli bierzemy pod uwagę architekturę wielowarstwową, to wiele baz danych i serwerów aplikacji wysyła dane do jednego lub wielu serwerów sieciowych, które następnie przekazują dane klientom. Języka XML można użyć do komunikacji pomiędzy komponentami struktury wielowarstwowej. Można również zastosować układ klient-serwer — zwykle serwer sieciowy jest wtedy wbudowany w bazę danych.

Jedną z zalet sieciowych baz danych jest uproszczenie aplikacji przez zastosowanie przeglądarek sieciowych obsługujących aplety Javy. W przypadku apletów kod języka Java jest przenoszony przez Sieć w chwili kiedy jest wywoływany, zaś przeglądarka zapewnia podstawowe formatowanie danych, obsługę formularzy i prostych języków skryptowych; jej możliwości mogą być rozszerzone przez zastosowanie apletów. Chęć poprawienia przepustowości i uruchamiania coraz bardziej złożonych aplikacji w coraz mniejszych systemach powoduje, że funkcje, które były dostępne w aplikacjach klienta, są wykonywane na serwerach lub w warstwach pośrednich (mówimy wtedy o aplikacjach „cienkiego klienta"). I tak na przykład aplety Javy mogą sięgać do bazy danych za pośrednictwem połączenia JDBC, zamiast stosować natywne sterowniki. Aplety mogą też korzystać z udostępnianych w pośredniej warstwie programów analitycznych, zamiast lokalnie wykonywać wszystkie obliczenia.

Dobrze opracowaną architekturę systemu rozpoznaje się po tym, że odpowiednie funkcje umieszczono w odpowiednich modułach. Wyboru można dokonać na podstawie różnych kryteriów, na przykład takich jak unikanie zależności od stosowanych technologii lub określenie zależności pojęć w opisywanej dziedzinie. Takie niefunkcjonalne wymagania to podstawowe kryteria stosowane w początkowych etapach projektowania. Zdefiniowanie odpowiednich kryteriów warunkuje dobrą architekturę systemu, jest też podstawą udanego projektu. W niektórych sytuacjach w doborze kryteriów mogą pomóc następujące pytania:

- Czy funkcje baz danych powinny być wydzielone jako odrębny moduł, by ułatwić uruchamianie systemu w różnych środowiskach?
- Czy w całym systemie będzie można zastosować technologię jednej firmy?
- Czy system powinien być niezależny od dostawcy oprogramowania warstwy pośredniej?
- Czy interfejs użytkownika ma przyjmować schemat bazy danych (minimalizowanie wpływu częstych zmian opisywanej dziedziny)?
- Czy, z uwagi na duże ilości danych, należy ograniczać ilość przesyłanych informacji z bazy danych?
- Czy — by spełnić wymogi bezpieczeństwa — kontrola dostępu do danych powinna być wydzielona jako odrębny moduł?

Najważniejsza jest równowaga między podstawowymi wymaganiami, tak aby zanadto nie ograniczać systemu. Elastyczność i spełnianie precyzyjnych wymagań to sprzeczne oczekiwania wobec systemu. Proces rozszerzania możliwości systemu często jest hamowany przez jego realne uwarunkowania. W oprogramowaniu ważnym dla działania przedsiębiorstwa dużo pracy wkłada się w tworzenie architektury dopasowanej do konkretnych wymagań i pozwalającej wykorzystywać dostępne zasoby. W przypadku szybko tworzonych systemów architekturze poświęca się niewiele uwagi. Wielu autorów skryptów Perl czy CGI łączy instrukcje języków SQL z generacją HTML i specyficznymi dla danej dziedziny regułami biznesowymi, w efekcie w szybkim tempie tworzą sprawnie działający system. Niestety, trudności wiążą się z utrzymaniem takiego systemu, a wzrastają przy każdej zmianie schematu bazy danych, przeglądarki i rozszerzaniu reguł biznesowych.

5.1.1. Klient-serwer

Przetwarzanie XML może się odbywać na serwerze baz danych. Baza danych XML lub program generujący XML mogą być włączone do innego serwera baz danych. (Rozwiązania, które oferują komercyjne relacyjne bazy danych opiszemy w punkcie 6.2.2). Poza tym można tak modyfikować kod serwera, aby zwracał kod HTML. By uzyskać taki efekt, należy skorzystać z darmowego serwera relacyjnej bazy danych, na przykład *mysql*, który potrafi generować XML. Inny sposób polega na dodaniu do obiektowej bazy danych obiektów obsługujących XML (tę metodę wykorzystano w systemie *eXcelon*; opis można znaleźć w punkcie 4.1.2). W tym punkcie zajmiemy się kilkoma prostymi metodami generowania XML z relacyjnego serwera baz danych. W końcowej części tego rozdziału opiszemy serwery baz danych, które charakteryzuje trzywarstwowa architektura i które można włączać do serwerów baz danych, korzystających z języków programowania takich jak Java.

XML można generować bezpośrednio w instrukcji SQL. Jeśli mamy tabelę DEPT z kolumnami: DEPTNO, DNAME i LOC (jak w bazie danych Oracle scott/tiger), to kod z wydruku 5.1 wygeneruje dokument XML. Aby stworzyć dokument XML, należy otoczyć dane znacznikiem <collection>. Istnieje jeszcze jeden warunek, który powinien zostać spełniony, byśmy mogli osiągnąć cel — serwer lub aplikacja klienta powinny obsługiwać funkcje agregujące, tak jak to pokazuje wydruk 5.2. Wprawdzie nie jest to rozwiązanie zbyt ogólne, ale stosując SQL, można w prosty sposób generować kod XML z relacyjnej bazy danych. Co ważne, w przypadku prostych zastosowań może to być wygodne.

Wydruk 5.1. *Kod SQL generujący fragment tabeli wydziałów przykładowej bazy danych Scott/Tiger*

```
select '<dept deptno="'||deptno||'" dname="'||dname||'" loc="'||loc||'"/>"'
  from dept
```

Wydruk 5.2. *Skrypt generujący dokument XML dla tabeli wydziałów przykładowej bazy danych Scott/Tiger*

```
#!/bin/sh
echo '<?xml version=\"1.0\"?>'
echo '<collection>'
sqlplus scott/tiger <<'EOF'
select '<dept deptno="'||deptno||'" dname="'||dname||'" loc="'||loc||'"/>"'
  from dept
'EOF'
echo '/<collection>'
```

Jeśli uniezależnimy się od konkretnych kolumn, to otrzymamy bardziej uniwersalne rozwiązanie. Na serwerze można stworzyć funkcję, która wartości kolumny otoczy znacznikami, których nazwy są nazwą kolumny, wiersze — nazwą tabeli, a dokument — elementem collection.

Rekord relacyjnej tabeli można przekształcić w pojedynczy element, a wartości zapisywać w atrybutach tego elementu, można też stworzyć element z podelementami. Zapis rekordów jako atrybutów jest prostszy, ale użycie podelementów — bardziej uniwersalne. Kiedy korzystamy z podelementów, możemy zastosować sekcje CDATA do zapisywania znaków zastrzeżonych. Atrybuty mogą przechowywać informacje o formatowaniu, można je zastąpić kompletniejszymi danymi ze wskaźnikami kluczy obcych. Różnice między zastosowaniem atrybutów i elementów opisano w punkcie 2.4.4.

Pseudokod, za pomocą którego można przekształcić rekordy tabeli w atrybuty, podano w wydruku 5.3, zaś ten, za pomocą którego przekształca się rekordy na elementy z podelementami, przedstawia wydruk 5.4. Wielu sprzedawców baz danych proponuje takie rozwiązania, w których wyniki zapytania przekształcane są na dokumenty XML. Propozycje firm IBM, Oracle i Microsoft opisano w rozdziale 6.

Wydruk 5.3. *Procedura zapisu danych z relacyjnej tabeli jako atrybutów XML*

```
procedure writeXMLAttributes(table)
   print "<?xml version=\"1.0\"?>"
   print "<collection>"
   for each row in table
      print "<" tablename ">"
      for each column in row
         print columnname "=" """ value """
```

```
      end
      print "/>"
   end
   print "/<collection>"
end
```

Wydruk 5.4. *Procedura zapisu danych z relacyjnej tabeli jako podelementów*

```
procedure writeXMLEmbedElements(table)
   print "<?xml version=\"1.0\"?>"
   print "<collection>"
   for each row in table
      print "<record>"
      for each column in row
         print "<" columnname ">"
         print value
         print "</" columnname ">"
      end
      print "</record>"
   end
   print "</collection>"
end
```

Niektóre przykłady kodu Java zaprezentowane w dalszej części tego rozdziału uruchomią się w systemie o architekturze dwuwarstwowej, która pozwala rozwijać klucze obce, pobierać pojedyncze rekordy oraz narzucać warunki na zapytania. Przetwarzanie XML bezpośrednio na serwerze baz danych, z właściwą mu architekturą klient-serwer, ma kluczowe znaczenie dla wydajności w dużych zbiorach danych. Jednak oprogramowanie jest opisywane jedynie w architekturze trzywarstwowej, gdyż wtedy wymogi związane ze znajomością różnych procesów i ich działaniem są mniejsze niż w przypadku architektury dwuwarstwowej. Jeśli trzeba skorzystać z systemu o architekturze klient-serwer, to funkcjonalność warstwy pośredniej i serwera z systemu o architekturze trzywarstwowej można ze sobą połączyć, tworząc serwer odpowiedni dla modelu opartego na strukturze dwuwarstwowej. Zdarza się, że nawet jeśli przetwarzanie odbywa się na serwerze, to i tak mogą wystąpić problemy z komunikacją między jądrem bazy danych a rozszerzeniami obsługującymi XML — w przypadku struktury trzywarstwowej łatwiej można poprawić wydajność.

5.1.2. Architektura trzywarstwowa

Prostą trzywarstwową architekturę sieciowego systemu baz danych pokazano na rysunku 5.1. System ten składa się z trzech modułów:

- *DBMS* — realizuje funkcje bazy danych i umożliwia dostęp do bazy. Można zastosować relacyjną, obiektową bazę danych, bazę danych XML lub system plików.
- *Warstwa pośrednia* — zawiera serwer sieciowy, który może korzystać z bazy danych i udostępniać te dane klientowi. Można zastosować zwykły serwer sieciowy, serwer sieciowy ze skryptami CGI, serwer z serwletami, serwer z JSP, komercyjne oprogramowanie warstwy pośredniej z dostępem do baz danych lub serwer aplikacji.

Rysunek 5.1. *Trzywarstwowa architektura sieciowego systemu baz danych*

- *Klient* — zawiera interfejs użytkownika, który pozwala korzystać z funkcji warstwy pośredniej. Klient może być też inną aplikacją. Zwykle rolę tę pełni przeglądarka, aplet Javy lub aplikacja napisana w języku Java.

W różnorodny sposób można określać zawartość poszczególnych modułw i sposób ich łączenia. Protokołem komunikacyjnym z klientem może być TCP/IP, HTML lub XML, przenoszone przez HTTP lub CORBA. Komunikacja między bazą danych lub warstwą pośrednią może się odbywać za pomocą JDBC lub CORBA. Zwykle wybór jednej techniki komunikacji wpływa na inne decyzje.

Oczywiście, zawsze można zastosować inne typy architektury i inne techniki. Niezależnie od dokonanego wyboru, projektując system, warto i należy zadać kilka pytań, między innymi:

- W jaki sposób dane są ładowane do bazy danych?
- Za pomocą jakich zapytań dane będą pobierane z bazy?
- W którym module jest generowny XML?
- W którym module odbywa się przetwarzanie XML?

5.2. Serwer sieciowy XML

Serwer XML umożliwia sieciowy dostęp do bazy danych. Obsługuje takie operacje baz danych, jak: przechowywanie, pobieranie, aktualizowanie dokumentów. Baza danych może opierać się na dowolnym modelu danych, ale dane pobierane zawsze mają postać XML. Kiedy serwer danych ładuje dane do bazy innej niż XML, może przyjmować dane w formie XML lub w formacie odpowiednim dla konkretnej bazy.

W następnym punkcie opiszemy niektóre możliwości implementacji, zaś w punkcie 5.2.2 sposób, w jaki można sięgać do danych na serwerze. W punktach 5.2.3 i 5.3 przedstawimy kolejno zapisywanie danych z bazy relacyjnej jako XML i odczyt tych danych. W punkcie 5.4 — pobieranie i odczyt danych XML z serwera bazy danych XML, zaś w punkcie 5.5 — pobieranie danych relacyjnych i XML ze wspólnego serwera.

5.2.1. Możliwości implementacji

Serwer danych XML można stworzyć „od zera”, na przykład modyfikując serwer sieciowy albo korzystając z serwera sieciowego z obsługą CGI, z serwera sieciowego z obsługą serwletów, z obsługą JSP, można także wykorzystać serwer aplikacji lub po prostu kupić gotowy system. Jeśli tworzymy serwer danych XML „od zera”, to należy pamietać, że serwer sieciowy jest podstawą, na której tworzy się funkcje dostępu do danych, funkcje

formatowania XML i zwracania odpowiedzi zgodniez podanymi adresami URL. Gotowy serwer sieciowy (na przykład niewielki, darmowy serwer) można zmodyfikować lub rozszerzyć o funkcje dostępu do bazy danych, formatowanie XML i odpowiadanie na żądania w formie adresów URL. W przypadku używania serwera z obsługą CGI, serwletów lub serwera aplikacji trzeba zrobić to samo. Korzystanie z serwera aplikacji ma jedną zaletę, serwery te posiadają wbudowaną funkcję połączeń z bazą danych. Dobry system komercyjny zazwyczaj od razu dysponuje wszystkimi potrzebnymi funkcjami.

Do serwera danych można dodać funkcje, które wykonują określone zadania przed zwróceniem, wprowadzeniem lub zmodyfikowaniem danych XML w bazie. Na przykład niektóre zapytania mogą wymagać agregacji lub wyliczeń statystycznych. W przypadku wprowadzania danych i ich modyfikowania konieczne może być sprawdzenie poprawności danych lub reguł biznesowych. Warto o tym pamiętać, zanim wybierzemy serwer danych.

Najlepszym rozwiązaniem jest, oczywiście, komercyjny serwer danych XML, gdyż spełnia wszystkie wymagania. Warto także rozważyć możliwość zakupu serwera aplikacji, który można łatwo połączyć ze środowiskiem projektowym (jeśli spełnia nasze wymagania) i który posiada odpowiednie funkcje baz danych. Wadą niektórych serwerów aplikacji jest to, że często projektanci są ograniczani przez dostępne interfejsy użytkownika lub niestandardową metodę łączenia się z bazą danych — wymaga to od nich dodatkowego wysiłku. Jednak tworzenie od początku pakietu łączności z bazą danych (na przykład opartego na JDBC) jest trudnym zadaniem i w przypadku dużych aplikacji komercyjnych należy to uznać za ostateczność (chyba że postawiliśmy sobie za cel stworzenie serwera danych XML — wtedy, oczywiście, rzecz jest warta zachodu). Pakiety łączności z bazą danych w przypadku małych aplikacji mogą być proste. Wystarczy otwarcie połączenia z bazą danych, wykonanie transakcji i zamknięcie połączenia. Taki prosty pakiet opisano w dodatku A. Jednak wykonanie pakietów łączności z bazą danych staje się dużo bardziej odpowiedzialnym zsadaniem, kiedy trzeba utrzymywać połączenia dla wielu transakcji przeprowadzanych z różnych kont, dbać o wydajność (wymaga to zarządzania połączeniami, czasem rozłączania się) i bezpieczeństwo połączeń.

Ważną decyzją związaną z implementacją serwera danych jest wybór języka programowania.

Jeśli do stworzenia serwera danych używamy Javy, serwer sieciowy powinien obsługiwać serwlety. Komunikacja bazy danych z serwerem sieciowym może mieć formę „cienkiego klienta" lub „grubego klienta". Wybór jednego z tych rozwiązań zależy od możliwości bazy danych. W przypadku „grubego klienta" cała aplikacja ma bezpośredni dostęp do bazy. W przypadku „cienkiego klienta" serwer sieciowy może sięgać do bazy danych za pomocą odpowiedniego protokołu, który zwykle nie obsługuje specjalnych funkcji bazy. Połączenie z bazą „cienkiego klienta" można zrealizować jako bezpośrednie, zapisane w języku C z obudową w języku Java. Rozwiązanie „cienkiego klienta" zwykle realizuje się za pomocą JDBC (ODBC zrealizowane w Javie), czyli standardowego mechanizmu dostępu do danych w bazach relacyjnych, typowego dla większości (a może nawet dla wszystkich) baz danych. Niektóre bazy danych oferują także bezpośrednie połączenie Javy oparte na JDBC lub innym protokole. Porównanie tych dwóch rozwiązań: „cienkiego klienta" i „grubego klienta" można znaleźć w każdej dobrej książce o JDBC. „Gruby klient" może być wydajniejszy. W tym rozdziale skorzystamy z JDBC („cienki klient"), gdyż jest to rozwiązanie ogólne, dostępne w większości baz danych — zwykle wystarczające.

Alternatywa dla Javy to użycie języka skryptowego Tcl/Tk. Ma on kilka zalet: łatwo można go rozszerzać, bez trudu można go łączyć z językiem C i różnymi pakietami oprogramowania. Niestety, nie jest tak popularny jak Java czy JavaScript, ale warto się zastanowić nad tym wyborem, szczególnie jeśli Tcl/Tk jest dostępny w używanym systemie. Język Tcl/Tk świetnie nadaje się do tworzenia prototypów, gdyż nie są uwzględniane typy danych, a struktury danych są elastyczne, poza tym język ten dysponuje wieloma wbudowanymi możliwościami. W Sieci są dostępne rozszerzenia łączności z bazami Oracle, Sybase i innymi, istnieją darmowe serwery HTTPD dostępne wraz z kodem źródłowym, w tym jeden autorstwa Scriptics. Znane są jeszcze lepsze serwery sieciowe i aplikacje napisane w Tcl/Tk, w tym serwer działający na stronie AOL, którego zaletą jest wbudowana łączność z bazami danych. Wadą jest mała popularność Tcl/Tk, co wpływa na utrudnienia w dostępie do pomocy i mniejszą liczbę darmowych pakietów, skromniejsze możliwości i wolniejszą reakcję na pojawiające się nowinki techniczne. Jednak ci, którzy znają Tcl/Tk, mogą w kilka dni stworzyć dobrze działający serwer danych XML, korzystając z darmowych komponentów. Jest to świetna metoda tworzenia prototypu takiego serwera.

Inne języki skryptowe to Perl i Python. Perl zawiera mnóstwo modułów obsługujących CGI i umożliwia łączność z bazami danych. Coraz większa grupa programistów korzysta z tego języka, gdyż umożliwia programowanie obiektowe oraz posiada sprawne algorytmy.

Osoby, które chciałyby zbudować serwer danych XML oparty na relacyjnej bazie danych, a nie mają dostępu do bazy komercyjnej, mogą skorzystać z darmowej bazy *mysql*, która jest obsługiwana w wielu językach skryptowych.

5.2.2. Dostęp klienta

Kiedy używamy sieciowego serwera danych, aplikacje klienckie uzyskują do niego dostęp za pomocą adresu URL. Adres ten może zawierać większość informacji, a nawet wszystkie, które są potrzebne, by mieć dostęp do danych, zaś dane zapisywane w bazie mogą być wysyłane metodą POST. Jeśli chcielibyśmy mieć dostęp do danych z serwera XML, trzeba zastanowić się nad takimi zagadnieniami:

- W jaki sposób zyskamy dostęp do serwera sieciowego?
- Jak należy podawać adres URL, aby umożliwić dostęp do bazy danych?
- Jakie są najważniejsze wyzwania projektowe?
- Należy użyć jednego adresu URL czy wielu?
- Jakie warunki decydują o konstrukcji adresu URL?

Oto sposoby, możliwości tworzenia adresu URL:

- Jeden adres URL, wszystkie informacje przekazywane są metodą POST pojedynczego dokumentu.
- Jeden adres URL, większość informacji przekazywana jest metodą POST i jako parametry GET.
- Różne adresy URL, które spełniają różne funkcje.

Wszystkie dane i polecenia mogą być umieszczane w dokumencie, a następnie przesyłane pod jednym adresem URL. Zaletą takiego rozwiązania jest przenoszenie wszystkich danych jako XML (dzięki czemu rozwiązanie jest zgodne z innymi metodami). Wada to konieczność parsowania dokumentów (przynajmniej wstępnego), zanim zostanie podjęta decyzja, które dane mają być wysłane. Jeśli na przykład można wprowadzić nowe dane lub zaktualizować istniejące, to wybór odpowiedniej operacji może wymagać częściowego przynajmniej parsowania dokumentu. Przekazywanie częściowo sparsowanego strumienia danych innemu parserowi jest trudne, o ile w ogóle możliwe. Jednak takie rozwiązanie jest przydatne, kiedy można od razu przeanalizować cały dokument. Użycie pojedynczego dokumentu to optymalne rozwiązanie, kiedy z serwerem danych XML współpracuje wiele aplikacji.

Kiedy użytkownicy kontaktują się z serwerem danych XML za pośrednictwem stron sieciowych i formularzy HTML, łatwiejsze jest przekazywanie danych metodami POST i GET. Jest to najlepsze rozwiązanie, kiedy mamy do czynienia z niewielkim zbiorem poleceń i niewielką ilością strukturalnych danych. Jeśli na przykład należy przeglądać dane i zadawać zapytania za pośrednictwem prostych formularzy, parametry można przekazywać w wierszu adresu. W tej książce postąpimy najlepiej jak potrafimy — z pedagogicznego punktu widzenia — użyjemy metod POST/GET, dane strukturalne przekażemy jako XML, podając kod XML jako część adresu URL.

Zastosowanie odrębnych adresów URL i przypisanie ich do innej funkcji lub jednego adresu i wskazywanie funkcji w parametrach to kwestia stylu programowania, choć w zależności od stosowanych technik programistycznych jedno z tych rozwiązań może być prostsze w implementacji.

Adresy URL można podawać w przeglądarce sieciowej, w formularzu HTML, z apletu, aplikacji, ze skryptu CGI lub w jakikolwiek inny sposób zapisany w HTTP.

5.2.3. Ładowanie danych

Załadowanie danych XML do istniejącej bazy relacyjnej (lub obiektowej) stanowi podwójne wyzwanie: semantyczne i techniczne. Pierwsze polega na odwzorowaniu semantyki XML, czyli przekształceniu danych XML w relacje — na przykład elementu `person` w tabelę `personnel`. Kiedy dokument XML jest projektowany z myślą o istniejącej bazie relacyjnej, odwzorowania powinny być proste. W dokumencie XML mogą wystąpić takie znaczniki, którym nie odpowiadają żadne obiekty bazy; dane tego rodzaju można pominąć lub należy tak zmodyfikować bazę, aby je uwzględnić.

Trudność odwzorowywania dokumentów XML w relacje pojawia się, kiedy semantyka XML i relacyjnej bazy danych częściowo się pokrywa. Na przykład dane XML, które pochodzą z jednego systemu, mogą zawierać informacje o pracownikach zatrudnionych etatowo, tymczasowo i na umowę zlecenia, zaś w bazie relacyjnej wszystkim tym grupom pracowników mogą odpowiadać odrębne relacje. W zasadzie problem jest podobny do tego, który wiąże się z łączeniem wielu baz danych. W takiej sytuacji przydają się doświadczenia związane z dużymi bazami danych oraz scalaniem różnych baz, szczególnie jeśli bazy oparte były na różnych modelach danych.

Drugi rodzaj problemu, techniczny — to sposób odwzorowania hierarchicznych danych XML, przekształcenia ich na płaskie relacje. Chodzi o to, żekiedy dojdzie do odwzorowania, zapis danych w formie relacji wymaga „spłaszczenia” danych.

5.2.3.1. Zmiana struktury danych XML

W przypadku przechowywania danych XML w gotowej bazie relacyjnej lub obiektowej mogą się przydać cztery wymienione rozwiązania.

Specjalny skrypt — to najprostszy, choć najmniej ogólny sposób. Polega na stworzeniu specjalnego programu, który odczyta i sparsuje dokument XML, a następnie wstawi dane do odpowiednich tabel.

Ograniczenie struktury — dokument XML można przekształcić (na przykład za pomocą XSLT) w strukturę podobną do relacji bazy danych. Elementy zagnieżdżone można zastąpić wartościami odpowiednich identyfikatorów, zaś ich treść umieścić w innej części dokumentu, która zostanie utworzona później. Połączenie elementu „rodzica" z elementem w nim zagnieżdżonym dokonuje się przez klucz obcy lub tabelę łączącą w relacyjnej bazie danych. W wydruku 5.5 mikromatryca podłoży z zagnieżdżonymi informacjami o genach może być zastąpiona dwoma zbiorami płaskich rekordów, gdzie podłoża będą odwoływały się do genów, korzystając z identyfikatora. Przykłady doświadczeń z mikromatrycą są podobne do opisanych w punkcie 2.3.3 eksperymentów hybrydyzacji na filtrze, ale przy okazji mierzona jest także liczba genów, które połączyły się z DNA matrycy.

Wydruk 5.5. *Zagnieżdżone rekordy przeznaczone do załadowania*

```
<?xml version="1.0"?>
<root>
<spot>
  <grid>1</grid>
  <version/>
  <position>1A1</position>
  <gene>
    <name>TYR1</name>
    <description>DEHYDROGENEZA PREFENATU</description>
    <pathway>BIOSYNTEZA TYROZYNY</pathway>
    <organism>drożdże</organism>
  </gene>
</spot>
<spot>
  <grid>1</grid>
  <version/>
  <position>1A2</position>
  <gene>
    <name>GRD19</name>
    <description>RETENCJA GOLGI PROTEINY</description>
    <pathway>WYDZIELANIE</pathway>
    <organism>drożdże</organism>
  </gene>
</spot>
</root>
```

Tworzenie połączeń — w czasie ładowania można sprawdzać na przykład format danych poprzez proste zapytania. Ułatwia to ładowanie tabel mających klucze obce.

Przekształcenia — jeśli używamy bazy danych opartej na innym modelu danych niż XML, dobrym pomysłem może być przekształcenie XML. Kiedy rolę klienta pełni użytkownik, a nie inna aplikacja, właściwe będzie sformatowanie danych jako XML, zanim zostaną załadowane do bazy, szczególnie w przypadku interaktywnego procesu wprowadzania lub edycji danych, gdy drobne poprawki pojawiają się na zmianę z zapytaniami. Dużym atutem stosowanego w takich systemach XML jest realizacja zapytań. Istnieją różne techniki, za pomocą których można tworzyć dane i korzystać z formularzy do edycji.

W niektórych sytuacjach, aby załadować dane XML do bazy relacyjnej, można użyć ogólnego narzędzia. Warunkiem jest, żeby preprocesor mógł przekształcać hierarchiczne dane XML w płaskie relacje. Jedna z możliwości to zastosowanie arkuszy stylów XSL do przekształcenia specyficznej postaci XML na ogólną postać rekordów, które można załadować za pomocą aplikacji języka Java. Jeśli jednak danych jest dużo, konieczne może być zastosowanie mechanizmu, który nie będzie przetwarzał danych w pamięci — na przykład użycie parsera SAX (opisanego w dodatku B).

Aplikacja ładująca dane może być samodzielnym programem lub znajduje się na serwerze, co pozwala przesłać dane XML do serwera HTTP i ładować do bazy. Poprzez Sieć dane można ładować na dwa sposoby: przez dokument XML lub metodami HTTP — POST/GET. Informacje o bazie danych i transakcjach można podać w parametrach POST/GET lub w dokumencie XML. Dokument ten może zawierać informacje potrzebne do załadowania danych w formie elementów lub atrybutów, zaś ładowane dane mogą być jednym z elementów dokumentu. Użycie dokumentu XML jest ogólniejszą metodą ładowania danych generowanych przez aplikację, ale użycie metod POST/GET pozwala wprowadzać dane ręcznie w formularzach HTML.

5.2.3.2. Ładowanie danych jako parametrów URL

Parametrów POST można użyć do przekazania informacji o ładowaniu danych, elementów oraz atrybutów. W przypadku elementu płaskiego parametry są wyliczane według nazwy. Struktury hierarchiczne są bardziej skomplikowane, ale można je tworzyć, wykorzystując hierarchię nazw. W takiej hierarchii podelementy elementu głównego są wyliczane według nazw, z kolei ich podelementy są nazywane przez połączenie ich nazw z nazwami „rodziców” (z użyciem separatora).

Innym problemem, związanym ze stosowaniem formularzy HTML do przekazywania informacji w parametrach, jest to, że niektóre parametry mogą być potrzebne aplikacji. Na przykład formularz może ustawiać zmienną `Submit`, konto bazy danych i punkt wejścia (nazwę tabeli lub klasę obiektu). Można postępować na cztery sposoby: jednoznacznie identyfikować zmienne aplikacji, jednoznacznie identyfikować zmienne użytkownika, jednoznacznie identyfikować jedne i drugie lub w ogóle się tym nie przejmować. Nazwy zmiennych identyfikuje się, stosując przedrostki, na przykład _, `var_`, `xml_`, `app_` lub `user_`.

By uprościć tworzenie formularza, można umieszczać polecenia XML w wartościach parametrów. Jeśli na przykład konieczne jest utworzenie niepowtarzalnego identyfikatora, można użyć polecenia nakazującego utworzyć taki identyfikator w trakcie ładowania, na przykład `<sequence_generate sequence="transaction_seq">`.

5.2.3.3. Ładowanie danych z dokumentów XML

Dokumenty XML, które chcemy załadować, mogą pochodzić z innej aplikacji, źródła zewnętrznego (bazy danych, zwykłego pliku) lub z formularza. Jednym ze sposobów utworzenia dokumentu XML na podstawie danych z formularza jest wykorzystanie skryptu lub programu formatującego dane z formularza HTML jako XML. Można też użyć apletu języka Java. Dokładniej temat ten omówimy w rozdziale 7.

Autor napisał w języku Java program ładujący dane do bazy relacyjnej, `rLoad`. Wydruk 5.6 pokazuje kod źródłowy `rLoad`. Aplikacja ładuje dane XML w dwóch etapach:

1. Pierwszy polega na przekształceniu danych XML za pomocą arkuszy XSL na dokument XML, składający się ze znaczników `record` i `field`, które odpowiadają ładowanym tabelom oraz metadanych opisujących ładowanie danych. Na przykład dokument z wydruku 5.7 można przekształcić za pomocą arkusza stylów na dokument XML `rLoad` (zobacz wydruk 5.8). Arkusze stylów zostaną dokładniej omówione w rozdziale 7., ale prosty arkusz, który w przykładzie z mikromatrycą przekształca geny, przedstawia wydruk 5.9.
2. Drugi etap to ładowanie danych `record/field` do bazy za pomocą parsera SAX. Dokument XML `record/field`, nazywany dokumentem `rload`, składa się z elementów `record` i `field`, a także metadanych, które spełniają następujące zadania:
   - Wskazują, w której tabeli powinien być umieszczony dany rekord. Rekordy przeznaczone do różnych tabel mogą się przeplatać w dokumencie. Poza tym rekord przeznaczony do jednej tabeli może być zagnieżdżony w rekordzie ładowanym do innej tabeli. Upraszcza to generację dokumentu XML `rload`.
   - Informują o tym, czy rekord zagnieżdżony ma być załadowany przed, czy za rekordem, w którym jest zagnieżdżony. Wpływa to na większą elastyczność definiowania, zwłaszcza w przypadku rekordów, które można zdefiniować w arkuszu XSL bez względu na klucz obcy. Na przykład w dokumencie XML pracownik może zawierać wydział, w którym dana osoba pracuje, tak samo wydział może zawierać listę zatrudnionych w nim osób. W trakcie ładowania rekord niezależny może zostać utworzony jako pierwszy, potem mogą być tworzone rekordy zależne, bez względu na to który rekord jest zagnieżdżonym, a który elementem otaczającym.
   - Wskazują, że wartość pola powinna być użyta do generacji wartości przez generator sekwencji w bazie danych.
   - Wskazują, że wartość pola ma być pobrana z generatora sekwencji bazy danych. Tak więc rekord zależny może odwoływać się do niepowtarzalnego identyfikatora wygenerowanego dla rekordu nadrzędnego.
   - Wskazują, że białe znaki powinny być z wartości odrzucone bądź znormalizowane.
   - Wskazują, że wartość powinna być identyfikatorem rekordu w innej tabeli, której dana kolumna ma wartość równą polu elementu. Pozwala to tworzyć klucze obce na podstawie identyfikatorów generowanych przez bazę danych, zaś klucze alternatywne lub kolumny o niepowtarzalnych wartościach mogą być używane przy wprowadzaniu i ładowaniu danych.

Wydruk 5.6. *Kod Java programu ładującego dane XML, rLoad*

```
/************************ LoadXML.java ***********************/
package com.xweave.xmldb.rload;
/**
 * Główna klasa ładująca dane XML do relacyjnej bazy danych.
 */
import org.xml.sax.*;
import org.xml.sax.helpers.ParserFactory;
import com.ibm.xml.parsers.DOMParser;
import org.w3c.dom.Document;
import java.io.IOException;
import com.xweave.xmldb.util.rdb.*;

public class LoadXML {
    public static OracleDB oracleDB = null;
public LoadXML() {
    super();
}
public static OracleDB getOracleDB() {
    if (oracleDB == null) {
        try {
            oracleDB = new OracleDB(new JDBCAcct("mgraves/mgraves@127.0.0.1"));
            oracleDB.connect();
        } catch (java.sql.SQLException ex) {
            ex.printStackTrace();
        }
    }
    return oracleDB;
}
public static void main(java.lang.String[] args) {
    // Tutaj wstawić kod inicjujący aplikację.
    if (args.length < 1) {
        System.err.println("XMLLoad: wymagany jest <plik> jako argument.");
    }
    (new LoadXML()).parse(args[0]);
}
public void parse(String xmlFile) {
    String parserClass = "com.ibm.xml.parsers.SAXParser";
    try {
        Parser parser = ParserFactory.makeParser(parserClass);
        HandlerBase handler = new JDBCLoadHandler();
            parser.setDocumentHandler(handler);
            parser.setErrorHandler(handler);
        try {
            parser.parse(xmlFile);
        } catch (SAXException se) {
            se.printStackTrace();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    } catch (ClassNotFoundException ex) {
        ex.printStackTrace();
    } catch (IllegalAccessException ex) {
        ex.printStackTrace();
    } catch (InstantiationException ex) {
        ex.printStackTrace();
    }
}
```

```
public static void setOracleDB(OracleDB newValue) {
    LoadXML.oracleDB = newValue;
}
}
/************************ Record.java ***********************/
package com.xweave.xmldb.rload;

/**
 * Zawiera informacje o rekordzie.
 */
import java.lang.*;
import com.xweave.xmldb.util.rdb.RDB;
public class Record {
    public String table;
    protected StringBuffer fieldNameBuffer = null;
    protected StringBuffer fieldValueBuffer = null;
    protected StringBuffer subqWhereBuffer = null;
    protected StringBuffer subqFromBuffer = null;
    private int numSubq = 0;
    private boolean inSubq = false;
    private boolean delay = false;
public Record() {
    super();
}
public void addFieldExpr(String value) {
    if (getFieldValueBuffer().length() > 0) {
        getFieldValueBuffer().append(",");
    }
    inSubq = false;
    getFieldValueBuffer().append(value);
}
public void addFieldName(String name) {
    if (getFieldNameBuffer().length() > 0) {
        getFieldNameBuffer().append(",");
    }
    getFieldNameBuffer().append(name);
}
public void addFieldSubq(String value) {
    if (getSubqWhereBuffer().length() > 0) {
        getSubqWhereBuffer().append(" AND ");
        getSubqFromBuffer().append(", ");
    }
    // podział słowa kluczowego na '.'
    int index = value.indexOf((int) '.');
    if (index == -1) {
        getSubqWhereBuffer().append(" BŁĄD "+value+" - nieznany");
        return;
    }
    String table = value.substring(0,index);
    String column = value.substring(index+1);
    // dodanie do buforów
    numSubq++;
    addFieldExpr("sqt"+Integer.toString(numSubq)+".id");
    inSubq = true;
    getSubqWhereBuffer().append("sqt"+Integer.toString(numSubq)+"."+column+" = ");
    getSubqFromBuffer().append(table+" sqt"+Integer.toString(numSubq));
}
```

```
public void addFieldValue(String value) {
    if (inSubq == true) {
        // note interaction with addFieldSubq
        getSubqWhereBuffer().append("'" + value + "'");
        inSubq = false;
        return;
    }
    if (getFieldValueBuffer().length() > 0) {
        getFieldValueBuffer().append(",");
    }
    // W razie potrzeby Oracle poprawnie zinterpretuje łańcuch jako liczbę.
    getFieldValueBuffer().append("'" + value + "'");
}
protected boolean getDelay() {
    return delay;
}
protected StringBuffer getFieldNameBuffer() {
    if (fieldNameBuffer == null) {
        fieldNameBuffer = new StringBuffer();
    }
    return fieldNameBuffer;
}
protected StringBuffer getFieldValueBuffer() {
    if (fieldValueBuffer == null) {
        fieldValueBuffer = new StringBuffer();
    }
    return fieldValueBuffer;
}
public static Record getFreeRecord() {
    return new Record();
}
public String getSQLInsertString() {
    StringBuffer buf = new StringBuffer();
    buf.append("insert into");
    buf.append(" " + getTable() + " ");
    buf.append("(" + getFieldNameBuffer() + ")");
    if (numSubq > 0) {
        buf.append(" select ");
        buf.append(getFieldValueBuffer());
        buf.append(" from " + getSubqFromBuffer());
        buf.append(" where " + getSubqWhereBuffer());
    } else {
        buf.append(" values ");
        buf.append("(" + getFieldValueBuffer() + ")");
    }
    return buf.toString();
}
protected StringBuffer getSubqFromBuffer() {
    if (subqFromBuffer == null) {
        subqFromBuffer = new StringBuffer();
    }
    return subqFromBuffer;
}
protected StringBuffer getSubqWhereBuffer() {
    if (subqWhereBuffer == null) {
        subqWhereBuffer = new StringBuffer();
    }
```

```
    return subqWhereBuffer;
}
public String getTable() {
    return table;
}
public void print() {
    System.out.println(getSQLInsertString());
    try {
        if (LoadXML.getOracleDB().executeUpdate(getSQLInsertString()) > 0) {
            System.out.println("...nie zadziałało");
        } else {
            System.out.println("...zadziałało");
        }
    } catch (java.sql.SQLException ex) {
        ex.printStackTrace();
        System.out.println("...nie zadziałało");
    }
}
protected void setDelay(boolean newValue) {
    this.delay = newValue;
}
protected void setFieldNameBuffer(StringBuffer newValue) {
    this.fieldNameBuffer = newValue;
}
protected void setFieldValueBuffer(StringBuffer newValue) {
    this.fieldValueBuffer = newValue;
}
protected void setSubqFromBuffer(StringBuffer newValue) {
    this.subqFromBuffer = newValue;
}
protected void setSubqWhereBuffer(StringBuffer newValue) {
    this.subqWhereBuffer = newValue;
}
public void setTable(String newValue) {
    this.table = newValue;
}
}
/************************ JDBCLoadHandler.java ***********************/
package com.xweave.xmldb.rload;

/**
 * Ładowanie danych XML do relacyjnej bazy danych.
 */
import org.xml.sax.*;
import java.util.*;
public class JDBCLoadHandler extends org.xml.sax.HandlerBase {
    private Record currentRecord = null;
    private Stack recordStack = null;
    private Vector printVector = null;
    private boolean inField = false;
    private boolean fieldEmpty = true;
    private short whitespaceOp = 0;
    private final static int WHITESPACE_OP_NONE = 0;
    private final static int WHITESPACE_OP_TRIM = 1;
    private final static int WHITESPACE_OP_REMOVE = 2;
public JDBCLoadHandler() {
    super();
}
```

```
public void characters(char[] chars, int start, int length) {
    fieldEmpty = false;
    if (inField) {
        switch (whitespaceOp) {
            case WHITESPACE_OP_TRIM :
                currentRecord.addFieldValue(String.valueOf(chars, start,
length).trim());
                break;
            case WHITESPACE_OP_REMOVE :
                StringBuffer sb = new StringBuffer();
                for (int i = start; i < start + length; i++) {
                    if (!Character.isWhitespace(chars[i])) {
                        sb.append(chars[i]);
                    }
                }
                currentRecord.addFieldValue(sb.toString());
                break;
            default :
                // default lub NONE: rzeczywiście usuwanie znaków
                currentRecord.addFieldValue(String.valueOf(chars, start,
length).trim());
        }
    }
}
public void endElement(String name) {
    inField = false;
    if (name.equalsIgnoreCase("record")) {
        endRecord();
        return;
    }
    if (name.equalsIgnoreCase("field")) {
        endField();
        return;
    }
}
protected void endField() {
    if (inField && fieldEmpty) {
        currentRecord.addFieldValue("");
    }
}
protected void endRecord() {
    if (getCurrentRecord().getDelay()) {
        getPrintVector().addElement(getCurrentRecord());
        setCurrentRecord(null);
        return;
    }
    getCurrentRecord().print();
    setCurrentRecord(null);
    if (! getPrintVector().isEmpty()) {
        // najpierw nieopóźniony rekord końcowy, więc drukujemy
        // rekordy opóźnione
        Enumeration e = getPrintVector().elements();
        while (e.hasMoreElements()) {
            ( (Record) e.nextElement() ).print();
        }
        getPrintVector().removeAllElements();
    }
}
```

```
protected Record getCurrentRecord() {
    return currentRecord;
}
protected java.util.Vector getPrintVector() {
    if (printVector == null) {
        printVector = new java.util.Vector();
    }
    return printVector;
}
protected java.util.Stack getRecordStack() {
    if (recordStack == null) {
        recordStack = new java.util.Stack();
    }
    return recordStack;
}
protected void setCurrentRecord(Record newValue) {
    if (newValue == null) {
        //clear currentRecord
        if (getRecordStack().empty()) {
            this.currentRecord = null;
        } else {
            this.currentRecord = (Record) getRecordStack().pop();
        }
        return;
    }
    if (getCurrentRecord() != null) {
        //embedded record
        getRecordStack().push(getCurrentRecord());
    }
    this.currentRecord = newValue;
}
protected void setPrintVector(java.util.Vector newValue) {
    this.printVector = newValue;
}
protected void setRecordStack(java.util.Stack newValue) {
    this.recordStack = newValue;
}
public void startElement(String name, AttributeList attrList) {
    //System.out.println(name);
    inField = false;
    whitespaceOp = WHITESPACE_OP_NONE;
    if (name.equalsIgnoreCase("record")) {
        startRecord(attrList);
        return;
    }
    if (name.equalsIgnoreCase("field")) {
        startField(attrList);
        return;
    }
}
protected void startField(AttributeList attrList) {
    if (currentRecord == null) {
        System.out.println("W rekordzie nie ma pola " + attrList.toString());
    }
    inField = true;
    fieldEmpty = true;
    for (int i = 0; i < attrList.getLength(); i++) {
```

```
        if (attrList.getName(i).equalsIgnoreCase("name")) {
            currentRecord.addFieldName(attrList.getValue(i));
        }
        if (attrList.getName(i).equalsIgnoreCase("sequence-generated")) {
            currentRecord.addFieldExpr(attrList.getValue(i) + ".NextVal");
            inField = false;
        }
        if (attrList.getName(i).equalsIgnoreCase("sequence-value")) {
            currentRecord.addFieldExpr(attrList.getValue(i) + ".CurrVal");
            inField = false;
        }
        if (attrList.getName(i).equalsIgnoreCase("keysearch")) {
            currentRecord.addFieldSubq(attrList.getValue(i));
        }
        if (attrList.getName(i).equalsIgnoreCase("whitespace")) {
            if (attrList.getValue(i).equalsIgnoreCase("none")) {
                whitespaceOp = WHITESPACE_OP_NONE;
            } else
                if (attrList.getValue(i).equalsIgnoreCase("trim")) {
                    whitespaceOp = WHITESPACE_OP_TRIM;
                } else
                    if (attrList.getValue(i).equalsIgnoreCase("remove")) {
                        whitespaceOp = WHITESPACE_OP_REMOVE;
                    }
        }
    }
}
protected void startRecord(AttributeList attrList) {
    setCurrentRecord(Record.getFreeRecord());
    for (int i = 0; i < attrList.getLength(); i++) {
        if (attrList.getName(i).equalsIgnoreCase("table")) {
            currentRecord.setTable(attrList.getValue(i));
        }
        if (attrList.getName(i).equalsIgnoreCase("delay")) {
            if (! attrList.getValue(i).equalsIgnoreCase("false")) {
                currentRecord.setDelay(true);
            }
        }
    }
}
}
```

Wydruk 5.7. *Przykład wprowadzanych danych*

```
<?xml version="1.0"?>
<root>
<exper_result>
  <spot>
    <grid>1000</grid>
    <version/>
    <position>1A1</position>
    <gene name="p53" organism="człowiek"/>
  </spot>
  <experiment>
     <exper_cond>ts1</exper_cond>
  </experiment>
  <variant>0</variant>
```

```
  <value>-0.56</value>
</exper_result>
<exper_result>
  <spot>
    <grid>1000</grid>
    <version/>
    <position>1A2</position>
    <gene name="BRCA1" organism="człowiek"/>
  </spot>
  <experiment>
     <exper_cond>ts1</exper_cond>
  </experiment>
  <variant>0</variant>
  <value>3.25</value>
</exper_result>
<exper_result>
  <spot>
    <grid>1000</grid>
    <version/>
    <position>1A3</position>
    <gene name="Pak1" organism="człowiek"/>
  </spot>
  <experiment>
     <exper_cond>ts1</exper_cond>
  </experiment>
  <variant>0</variant>
  <value>2.14</value>
</exper_result>
</root>
```

Wydruk 5.8. *Rekordy zagnieżdżone do ładowania z wartościami zapytań do bazy danych*

```
<?xml version="1.0"?>
<ROOT>
<record table="exper_result">
  <field name="id" sequence-generated="exper_result_s"/>
  <record table="spot">
    <field name="id" sequence-generated="spot_s"/>
    <field name="grid>1000</field>
    <field name="version"/>
    <field name="position>1A1</field>
    <field name="gene" keysearch="gene.name">p53<field/>
  </record>
  <record table="experiment">
     <field name="id" sequence-generated="experiment_s"/>
     <field name="exper_cond">ts1</field>
  </record>
  <field name="variant">0</field>
  <field name="value">-0.56</field>
</record>
<record table="exper_result">
  <field name="id" sequence-generated="exper_result_s"/>
  <record table="spot">
    <field name="id" sequence-generated="spot_s"/>
    <field name="grid>1000</grid>
    <field name="version/>
    <field name="position>1A2</field>
```

```
    <field name="gene" keysearch="gene.name">BRCA1<field/>
  </record>
  <field name="experiment" sequence-value="experiment_s"/>
  <field name="variant>0</field>
  <field name="value>3.25</field>
</record>
</ROOT>
```

Wydruk 5.9. *Arkusz XSL przekształcający dane XML z przykładu na XML rLoad*

```
<?xml version="1.0"?>
<!-- Arkusz stylów ładujący dane o genach -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml"/>
  <xsl:template match="/">
    <xsl:apply-templates select="*" />
  </xsl:template>
  <xsl:template match="root">
    <ROOT>
    <xsl:apply-templates select="*" />
    </ROOT>
  </xsl:template>
  <xsl:template match="gene">
    <record table="GENE">
    <field name="id" sequence-generated="gene_s" />
    <xsl:apply-templates select="@*">
    </xsl:apply-templates>
    </record>
  </xsl:template>
    <xsl:template match="@name">
      <field name="name">
        <xsl:value-of select="."/>
      </field>
    </xsl:template>
    <xsl:template match="@description">
      <field name="description">
        <xsl:value-of select="."/>
      </field>
    </xsl:template>
      <xsl:template match="@pathway">
        <field name="pathway">
        <xsl:value-of select="."/>
        </field>
      </xsl:template>
      <xsl:template match="@organism">
        <field name="organism">
        <xsl:value-of select="."/>
        </field>
      </xsl:template>
</xsl:stylesheet>
```

Implementacja w języku Java składa się z trzech klas: głównej klasy `LoadXML`, klasy `JDBCLoadHandler` używanej z parserem SAX oraz klasy `Record`, zawierającej informacje dotyczące wszystkich ładowanych rekordów. Metoda `main` klasy `LoadXML` pobiera plik XML `rload` jako dane wejściowe i wywołuje metodę `parse`, tworzącą obiekt parsera SAX

z procedurą obsługi `JDBCLoadHandler`. `JDBCLoadHandler` tworzy obiekt `Record` i wypełnia w miarę parsowania jego pola. Jeśli element XML `record` wymaga więcej niż jednego rekordu relacyjnego, tworzonych jest odpowiednio wiele obiektów `Record`.

5.2.4. Generacja XML

Istnieje kilka sposobów generowania dokumentów XML na podstawie danych z relacyjnej bazy danych. Przedstawimy pięć możliwości:

a) Dokument XML można wygenerować na podstawie danych z tabeli relacyjnej bazy danych. Tabela jest formatowana jako dokument XML, zaś kluczy obcych używa się do określenia hierarchii elementów dokumentu. Nazwy typów elementów to nazwy tabeli i kolumn. Takie rozwiązanie przydaje się do przeglądania danych z bazy.

b) To rozwiązanie jest podobne do poprzedniego: o strukturze dokumentu decyduje struktura bazy danych, ale możliwe jest też użycie widoków (perspektyw). Każdy widok jest zamieniany na dokument XML. Dzięki temu tworzenie dokumentów jest bardziej elastyczne, gdyż struktura dokumentu nie zależy bezpośrednio od postaci tabel. Hierarchię elementów określają klucze obce i (lub) relacje jawnie określające, które wartości i jak mają być rozwijane.

c) Dokumenty można tworzyć na podstawie formułowanych *ad hoc* zapytań. Hierarchia elementów nadal zależy od struktury bazy danych (i ewentualnie widoków).

d) Szablon dokumentu można zdefiniować, podając jego części w formie zapytań. Zapytania są wykonywane i przekształcane na XML jak poprzednio, ale wyniki wielu zapytań są zbierane w jednym dokumencie.

e) Można podać zapytanie z podzapytaniami, wtedy strukturę dokumentu określa struktura zapytań. Takie rozwiązanie sprawdza się tylko wtedy, gdy generacja XML odbywa się bezpośrednio w maszynie bazy danych.

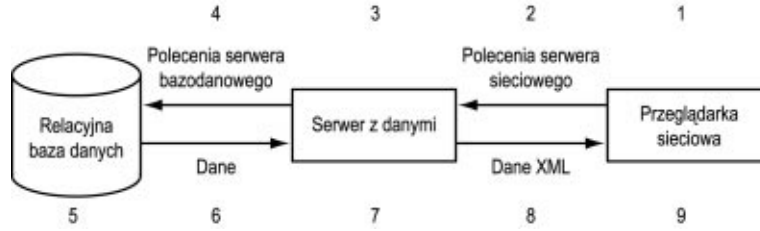
Rozwiązania wymienione w punktach od a) do c) są wykorzystywane w oprogramowaniu, o którym mówi punkt 5.3. Rozwiązanie d) może być przydatne, kiedy generacja XML jest powiązana z serwerem sieciowym obsługującym generację dokumentów po stronie serwera, jak Java Server Pages (JSP).

5.3. Relacyjny serwer danych

Sterownik relacyjnej bazy danych udostępnia dane z relacyjnego DBMS w XML. Do bazy danych kierowane jest zapytanie, po czym wyniki z bazy są formatowane jako XML. W punkcie 6.2 opisano zastosowanie rozwiązań komercyjnych. *Relacyjny serwer danych* jest sterownikiem bazy danych wraz z serwerem sieciowym.

Podstawowy schemat pracy użytkownika, żądającego raportu XML z relacyjnej bazy danych przez przeglądarkę sieciową, wygląda następująco (rysunek 5.2):

1. Użytkownik podaje w przeglądarce sieciowej w formie adresu URL zapytanie do relacyjnej bazy danych.
2. Przeglądarka przesyła żądanie adresu URL do serwera danych.



Rysunek 5.2. *Przetwarzanie żądania użytkownika w trzywarstwowej strukturze relacyjnego serwera danych*

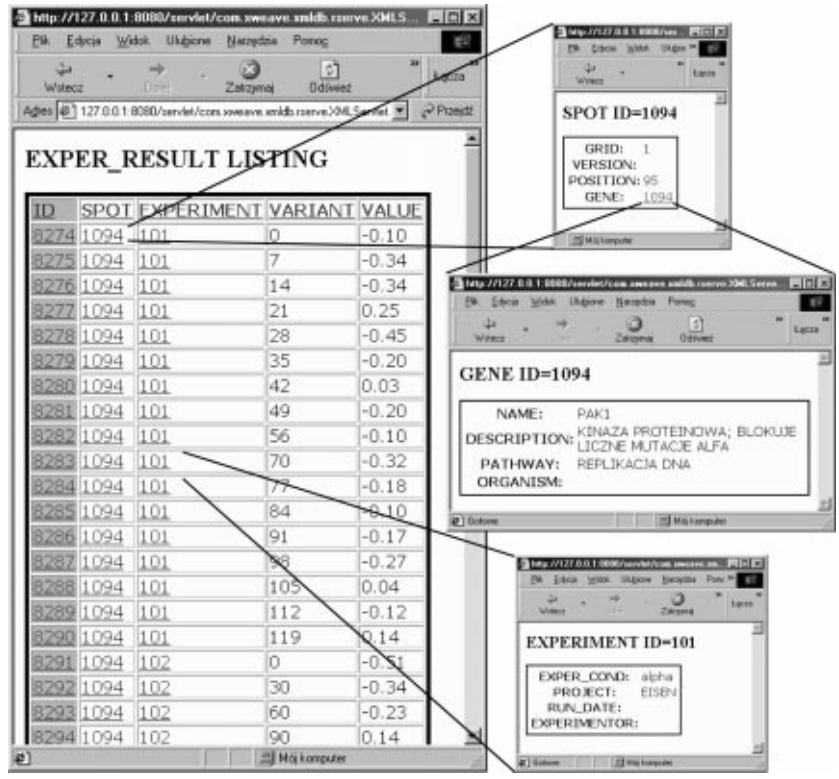
3. Serwer danych analizuje żądanie URL i tworzy zapytanie SQL.
4. Serwer danych przekazuje zapytanie SQL do serwera baz danych.
5. Serwer baz danych wykonuje zapytanie.
6. Serwer baz danych zwraca do serwera danych wyniki zapytania w formie tabelarycznej.
7. Serwer danych formatuje wyniki jako XML.
8. Serwer danych zwraca dane w postaci XML do przeglądarki.
9. Przeglądarka sieciowa parsuje dane XML i wyświetla je.

Jeśli arkusz stylów został użyty, konieczne jest pobranie i przeanalizowanie w przeglądarce także arkusza.

Jeśli chcemy stworzyć relacyjny serwer danych, powinniśmy odpowiedzieć na wiele pytań, między innymi:

- Czy oprócz tabel przewidujemy wyświetlanie danych z widoków?
- Czy dane będą aktualizowane, czy są przeznaczone tylko do odczytu?
- Czy użytkownik może wybierać jednocześnie dane z wielu tabel?
- Jak obsługiwane są złączenia?
- Jak złożone mogą być odwzorowania relacji na strukturę XML?
- Czy mają być obsługiwane (może specjalnie) tabele pomocnicze lub tabele złączające?
- Czy wiązania będą analizowane przez klucze obce? Do ilu poziomów?
- Jak są obsługiwane cykliczne powiązania kluczy obcych?
- Jak są obsługiwane powiązania między tabelami?
- Co pozwalają zrealizować zapytania?
- Jak dużo danych ma być zwracanych? Duże porcje czy niewielkie fragmenty?

Na te pytania trzeba było odpowiedzieć, opracowując relacyjny serwer danych w języku Java, *rServe* (opisano go szczegółowo w punkcie 5.3.5). Przykładowe dane dotyczące bazy danych mikromatrycy pokazano na rysunku 5.3. Przykład sformatowano, korzystając z arkusza stylów opisanego w rozdziale 7. Warto zauważyć, że niektóre kolumny to hiperłącza, za którymi kryją się kolejne dane generowane przez *rServe*.



Rysunek 5.3. *Przykład danych z rServe z bazy danych mikromatryc*

5.3.1. Żądania adresu URL

API do relacyjnego serwera danych można zrealizować jako żądania URL. Zanim zajmiemy się szczegółami *rServe*, opiszemy takie API.

rServe umożliwia korzystanie z danych tylko do odczytu za pośrednictwem adresu URL. Serwlet jest wywoływany z parametrami takimi, jak: nazwa tabeli, ograniczenia na kolumny, konto w relacyjnej bazie danych. Alternatywna implementacja umożliwiałaby podawanie instrukcji SQL w adresie URL. Oto przykłady adresów URL dla *rServe*:

- *http://127.0.0.1/servlets/com.xweave.xmldb.rserve.XMLServlet?tablename=company* — pobiera wszystkie rekordy z tabeli `company`.
- *http://127.0.0.1/servlets/com.xweave.xmldb.rserve.XMLServlet?tablename=company &stylesheet=/ss/generic1.xsl* — pobiera wszystkie rekordy z tabeli `company`, następnie formatuje dane według arkusza stylów */ss/generic1.xsl*.
- *http://127.0.0.1/servlets/com.xweave.xmldb.rserve.XMLServlet?tablename=company &id=12* — pobiera z tabeli `company` rekord o identyfikatorze 12.
- *http://127.0.0.1/servlets/com.xweave.xmldb.rserve.XMLServlet?tablename=company &stylesheet=/ss/generic1.xsl&name=Acme* — pobiera z tabeli `company` rekord, który w kolumnie `name` ma wartość „`Acme`”, następnie otrzymane dane zwraca, stosując arkusz stylów */ss/generic1.xsl*.

Adres URL składa się z trzech zasadniczych części: adresu bazowego, arkusza stylów i zapytania. Adres wskazuje serwlet (lub skrypt CGI), obsługujący dane. W naszym wypadku jest to adres serwletu *rServe* zainstalowany na dowolnym komputerze wskazywanym przez adres (w tym wypadku skorzystaliśmy z adresu IP 127.0.0.1 zarezerwowanego dla serwera lokalnego, *localhost*). Opcjonalny wskaźnik arkusza stylów informuje, gdzie znajduje się arkusz XSL używany przez przeglądarkę do przekształcenia XML na HTML. Arkusz i zapytanie mogą występować w dowolnej kolejności.

Baza adresu może wyglądać następująco:

- *http://127.0.0.1/servlets/com.xweave.xmldb.rserve.XMLServlet?*
- *http://mojkomputer/servlets/com.xweave.xmldb.rserve.XMLServlet?*

Na końcu pojawia się znak zapytania.

Adres arkusza stylów przybiera postać `stylesheet=/ss/generic1.xsl`. Arkusz może zostać pominięty, wtedy zostanie użyty domyślny arkusz przeglądarki. Zwykle powoduje to pokazanie tekstu i znaczników dokumentu XML.

Zapytanie składa się z nazwy tabeli i nazw kolumn, poszczególne pary wartości rozdzielane są znakiem `&`.

Do pobrania wszystkich rekordów z tabeli używa się zapytania `tablename=<tabela>`, gdzie `<tabela>` to nazwa pokazywanej tabeli. Do pobrania konkretnego rekordu używa się zapisu `tablename=<tabela>&id=<id>`. Częścią zapytania może być nazwa dowolnej kolumny.

Dodatkowo serwer danych umożliwia podanie konta DBMS w postaci pary nazwa-wartość, `acct=<konto>/<hasło>@<instancja>`. Korzystając z adresu URL, można sięgnąć do dowolnej bazy danych dostępnej na serwerze DBMS, na przykład

```
http://127.0.0.1/servlets/com.xweave.xmldb.rserve.XMLServlet?tablename=dept&stylesheet=
/ss/generic1.xsl&acct=scott/tiger@ORCL
```

Tworząc adres URL, należy zwrócić uwagę na kilka rzeczy. Jeśli używamy formularzy HTML, to nazwy takie jak `Submit` wypełniane są automatycznie. Parametr `Submit` jest pomijany przez serwer danych. W adresie URL znaki specjalne są kodowane w ASCII jako kody szesnastkowe lub zastępowane innymi znakami, na przykład / zapisuje się jako `%2F`, a spację jako +. Ostatecznie adres URL może tak wyglądać:

```
http://127.0.0.1/servlets/com.xweave.xmldb.rserve.XMLServlet?tablename=dept&stylesheet
=%2Fss%2Fgeneric1.xsl&name=Acme&Submit=Find+company
```

A oto rozwinięcie takiego adresu:

```
http://127.0.0.1/servlets/com.xweave.xmldb.rserve.XMLServlet?tablename=dept&stylesheet
=/ss/generic1.xsl&name=Acme
```

5.3.2. Tworzenie zapytań SQL

Relacyjny serwer używa parametrów adresu URL do tworzenia instrukcji SQL. Na przykład żądanie pobrania zawartości tabeli tworzy się, używając parametru `nazwatabeli` w szablonie:

```
select * from <nazwatabeli>
```

Bardziej złożone zapytania można tworzyć stopniowo, dodając ograniczenia na wartości kolumn we frazie `where`, żądania uporządkowania we frazie `order by` oraz podając kolumny do wyświetlenia we frazie `select`. W efekcie uzyskuje się następujące zapytania SQL:

```
select <kolumny> from <nazwatabeli>
    where <warunki>
    order by <kolejność>
```

Dużą część instrukcji SQL można zakodować w adresie URL, choć warunki podawane we frazie `where` wymagają nieco więcej pracy. Nazwa musi być oddzielona od wartości znakiem równości, dlatego inne operacje można zakodować przez podanie wartości i operatora, na przykład:

```
...&partno=1234&quantity=20&...   (quantity równe 20)
...&partno=1234&quantity=<20&...   (quantity mniejsze od 20)
...&partno=1234&quantity=<=20&...   (quantity mniejsze bądź równe 20)
```

5.3.3. Formatowanie wyników jako XML

Dane relacyjne można formatować jako XML, używając nazw tabel i kolumn tabel do sterowania tworzeniem nazw elementów (i ewentualnie nazw atrybutów). Informacje zawarte w kluczach głównych i obcych służą do definiowania struktury dokumentu. Trudności przysparza fakt, że w nazwach kolumn można używać znaków specjalnych, których nie można stosować w nazwach elementów XML. Drugi problem wiąże się z tym, że klucze obce mogą tworzyć cykle (w ten sposób może dojść do utworzenia pętli nieskończonej). Cykl powstaje, kiedy klucz obcy odwołuje się do innej tabeli, do innego klucza obcego, który znowuż odwołuje się do klucza obcego z pierwszej tabeli. W takim cyklu może istnieć więcej kluczy i tabel pośrednich. Rozwiązaniem jest zapamiętywanie kolejnych tabel lub ograniczanie liczby kluczy obcych.

Formatowanie wyników zapytania przebiega następująco:

1. Wypisanie nagłówka dokumentu.
2. Wypisanie znacznika początkowego elementu głównego.
3. Jeśli nazwy kolumn używane są jako nazwy typów elementów, to pobranie ich z metadanych.
4. Przeglądanie kolejnych rekordów relacyjnych.

4.1. Wypisanie znacznika początkowego relacji (na przykład nazwy tabeli, pojęcia ogólniejszego jako „rekord” lub innego tekstu).

4.2. Iteracja realizowana po kolejnych kolumnach rekordu.

4.2.1. Jeśli wartością danej kolumny jest napis, umieszcza się go między znacznikiem początkowym i końcowym.

4.2.2. Jeśli kolumna zawiera klucz obcy, można ewentualnie wypisać rekursywnie kolumny rekordu, który wskazuje dany klucz.

4.3. Wpisanie znacznika końcowego relacji.

5. Wypisanie znacznika końcowego elementu głównego.

Podany algorytm może nie znaleźć zastosowania w dużych bazach danych, gdyż wymaga wykonywania wielu zapytań. Poza tym klucze obce pobierane są w kolejnych krokach (powstaje w ten sposób wiele poziomów drzewa dokumentu). Kiedy pobierane klucze obce są takie same jak wcześniejsze, powtarzane są te same zapytania. Algorytm nie pozwala skorzystać z informacji strukturalnych relacyjnej bazy danych, na przykład informacji o kluczach obcych. Można byłoby ich użyć do tworzenia zapytań pobierających w jednym zapytaniu informacje z wielu wierszy (lub wielu tabel). Dzięki temu w jednym zapytaniu można pobierać informacje do wielu elementów.

5.3.4. Pobieranie danych słownikowych

Komercyjne relacyjne systemy baz danych zawierają informacje o tabelach systemowych, które mogą się przydać przy tworzeniu serwera danych. Na przykład klucz główny stanowi źródło jednoznacznego indeksu rekordów. Można go używać do identyfikowania poszczególnych rekordów. Klucze obce wskazują klucze główne innych relacji. Można ich użyć do łączenia elementu z jego podelementami. Jeśli na przykład klucz obcy tabeli `zakup` wskazuje klucz główny tabeli `klient`, to dokument XML, który pokazuje dane z tabeli zamówień, może uwzględniać dane klienta jako podelement. W większości relacyjnych baz danych informacje o kluczach głównych i obcych są zapisywane w tabelach systemowych. Zapytania SQL mogą pobrać te informacje na potrzeby serwera danych.

Klucze obce można wykorzystać do wypisywania danych relacyjnych jako drzewa XML. Wartość kolumny lub wartości kolumn mogą zawierać wartości klucza głównego innej relacji. Klucz główny jednoznacznie identyfikuje rekord danej relacji, zaś klucz obcy można zastąpić treścią wskazywanego rekordu.

Dane z relacyjnej bazy danych można przekształcić na XML w formie drzewa. Korzystając z relacji „rodzic”-„dziecko” w bazie danych, tworzy się analogiczną relację w drzewie XML. Relacje takie w relacyjnej bazie danych są zwykle kodowane za pomocą kluczy obcych.

Ostrzeżenie dla projektantów baz danych: Terminy „rodzic”-„dziecko” wywołują pewne nieporozumienie. W projekcie relacyjnym zależność „rodzic”-„dziecko” podlega analizie, tak więc „dziecko” zależy od informacji „rodzica”. W drzewach „dzieci” składają się na „rodzica”, więc to „rodzic” zależy od „dzieci”. Wobec tego relacja „rodzic”-„dziecko” w teorii relacji ma znaczenie dokładnie odwrotne niż relacja „rodzic”-„dziecko” w drzewie XML. Na przykład jeśli w relacyjnej bazie danych rejestruje się przynależność pracowników do wydziałów, wydziały będą relacją „rodzicem”, zaś relacja pracowników będzie „dzieckiem”. W przypadku formatowania danych jako XML bezpośrednio na podstawie relacji pracowników, element opisujący pracownika będzie „rodzicem” i będzie miał element wydziału jako swoje „dziecko” (rysunek 5.4.).

Rysunek 5.4. *Porównanie relacji „rodzic”-„dziecko” w teorii relacyjnej i drzewach XML*

Schemat relacyjny doświadczenia z mikromatrycą (zobacz rysunek 5.5) ma klucze obce opisane w tabeli 5.1. Za pomocą tych kluczy obcych można wygenerować kod XML, który prezentuje wydruk 5.10. Klucze obce z tabeli 5.1 są analizowane w celu stworzenia elementów zagnieżdżonych.

Rysunek 5.5. *Relacyjny schemat bazy danych mikromatryc*

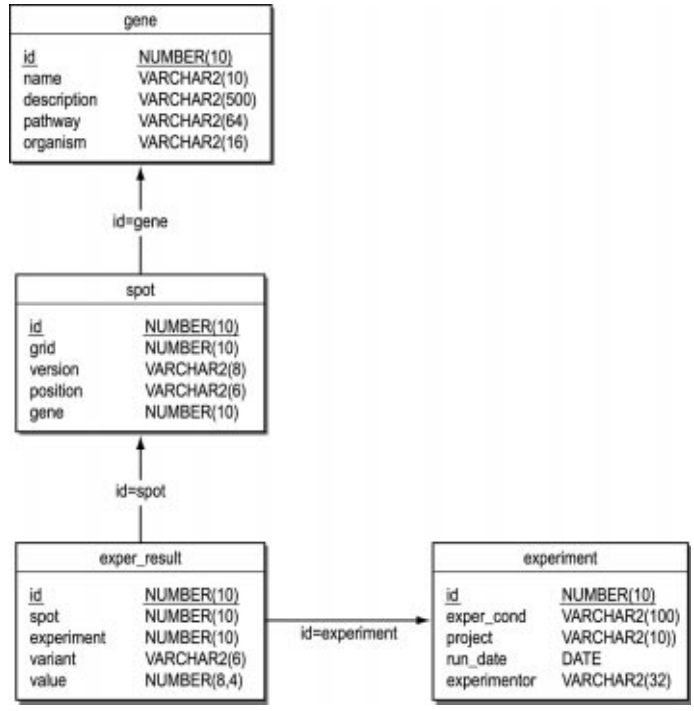


Tabela 5.1. *Klucze obce bazy danych mikromatryc*

Tabela „dziecko”	Kolumna „dziecko”	Tabela „rodzic”	Kolumna „rodzic”
SPOT	GENE	GENE	ID
EXPER_RESULT	SPOT	SPOT	ID
EXPER_RESULT	EXPERIMENT	EXPERIMENT	ID

Wydruk 5.10. *Przykład — wynik działania rServe uzyskany z bazy danych mikromatryc*

```
<?xml version="1.0"?>
<collection>
<EXPER_RESULT id="8274">
  <SPOT>
   <SPOT id="1094">
    <GRID>1</GRID>
    <VERSION/>
    <POSITION>95</POSITION>
    <GENE>
      <GENE id="1094">
        <NAME>PAK1</NAME>
        <DESCRIPTION>KINAZA BIAŁKOWA; POZWALA UNIKNĄĆ LICZNYCH MUTACJI ALFA</DESCRIPTION>
```

```
        <PATHWAY>REPLIKACJA DNA</PATHWAY>
        <ORGANISM/>
      </GENE>
    </GENE>
   </SPOT>
  </SPOT>
  <EXPERIMENT>
    <EXPERIMENT id="101">
      <EXPER_COND>alfa</EXPER_COND>
      <PROJECT>EISEN</PROJECT>
      <RUN_DATE/>
      <EXPERIMENTOR/>
    </EXPERIMENT>
  </EXPERIMENT>
  <VARIANT>0</VARIANT>
  <VALUE>-0.10</VALUE>
</EXPER_RESULT>
</collection>
```

W bazie danych Oracle można użyć kodu z wydruku 5.11, aby pobierać z tabel systemowych informacje o kluczach obcych. Na przykład w tabeli 5.2 pokazano klucze obce relacyjnego systemu przechowywania danych XML o strukturze drobnoziarnistej (rozdział 4.). Kod SQL z wydruku 5.12 wykonuje analogiczne zadania w bazie DB2.

Wydruk 5.11. *Kod SQL pozwala pobrać informacje o kluczach obcych z bazy danych Oracle*

```
select c.table_name child_table,
       c.column_name child_column,
       p.table_name parent_table,
       p.column_name parent_column
  from sys.all_constraints l,
       sys.all_cons_columns c,
       sys.all_cons_columns p
 where l.constraint_type = 'R'
   and l.constraint_name = c.constraint_name
   and l.r_constraint_name = p.constraint_name
   and l.owner = c.owner
   and l.r_owner = p.owner
   and c.position = p.position
```

Wydruk 5.12. *Kod SQL pozwala pobrać informacje o kluczach obcych z bazy danych DB2*

```
select c.tabname child_table,
       c.colname child_column,
       p.tabname parent_table,
       p.colname parent_column
  from syscat.references l,
       syscat.keycoluse c,
       syscat.keycoluse p
 where l.constname = c.constname
   and l.refkeyname = p.constname
   and l.tabschema = c.tabschema
   and l.reftabschema = p.tabschema
   and c.colseq = p.colseq
```

Tabela 5.2. *Klucze obce relacyjnego systemu przechowywania danych XML*

Tabela „dziecko”	Kolumna „dziecko”	Tabela „rodzic”	Kolumna „rodzic”
XDB_ATTR	DOC_ID	XDB_DOC	DOC_ID
XDB_CHILD	DOC_ID	XDB_DOC	DOC_ID
XDB_ELE	DOC_ID	XDB_DOC	DOC_ID
XDB_STR	DOC_ID	XDB_DOC	DOC_ID
XDB_TEXT	DOC_ID	XDB_DOC	DOC_ID
XDB_ATTR	DOC_ID	XDB_ELE	DOC_ID
XDB_CHILD	DOC_ID	XDB_ELE	DOC_ID
XDB_DOC	DOC_ID	XDB_ELE	DOC_ID
XDB_STR	DOC_ID	XDB_ELE	DOC_ID
XDB_TEXT	DOC_ID	XDB_ELE	DOC_ID
XDB_ATTR	ELE_ID	XDB_ELE	ELE_ID
XDB_CHILD	ELE_ID	XDB_ELE	ELE_ID
XDB_DOC	ROOT	XDB_ELE	ELE_ID
XDB_STR	ELE_ID	XDB_ELE	ELE_ID
XDB_TEXT	ELE_ID	XDB_ELE	ELE_ID

5.3.5. Implementacja

Kod języka Java, stanowiący implementację relacyjnego serwera danych *rServe*, pokazuje wydruk 5.13. Diagram klas przedstawiono na rysunku 5.6.

Wydruk 5.13. *Kod Java relacyjnego serwera danych rServe*

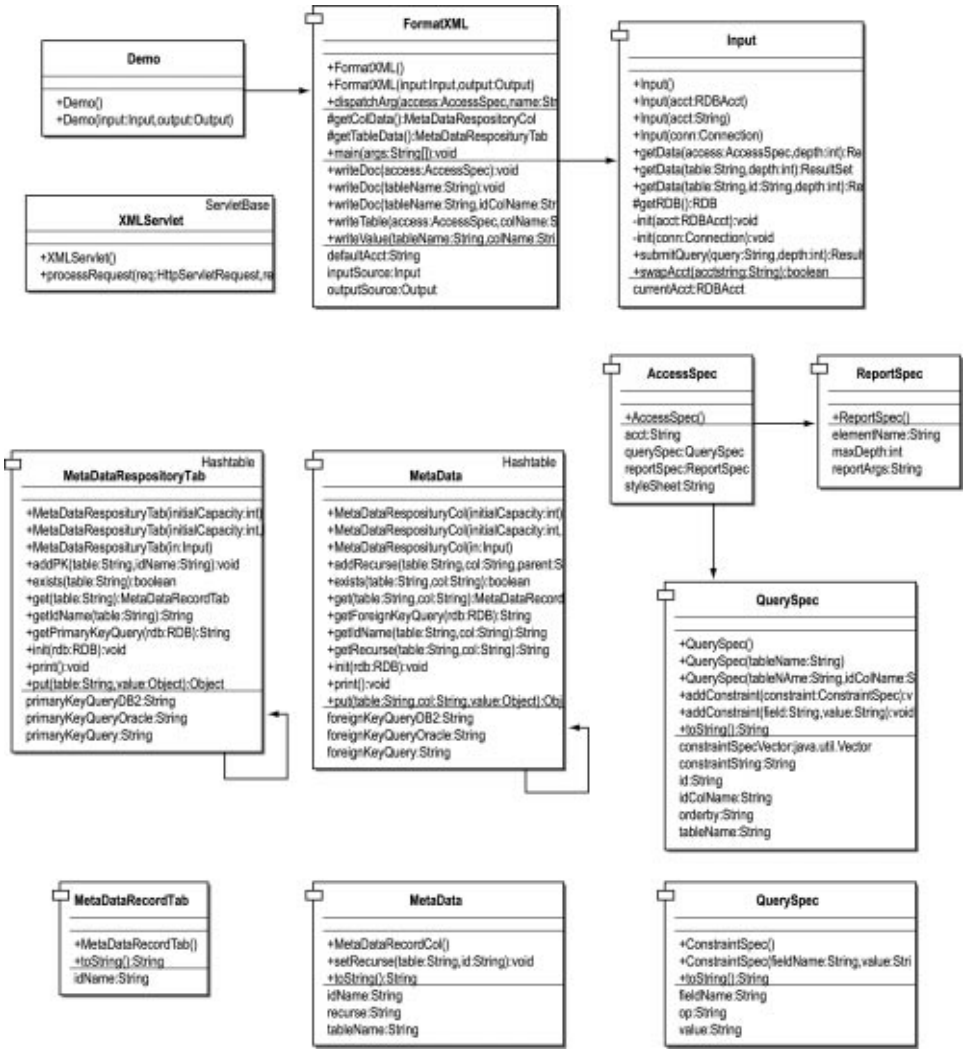
```
/************************ Demo.java ***********************/
package com.xweave.xmldb.rserve;

import com.xweave.xmldb.util.io.Output;
public class Demo extends FormatXML {
/**
 * Konstruktor Demo.
 */
public Demo() {
    super();
}
public Demo(Input input, Output output) {
    super(input, output);
}
}

/************************ FormatXML.java ***********************/
package com.xweave.xmldb.rserve;

import java.sql.*;
import java.util.*;
```

Rysunek 5.6.
Diagram klas UML relacyjnego serwera danych rServe



```
import com.xweave.xmldb.util.io.Output;
/**
 * Główny generator formatujący dane relacyjne jako XML.
 */
public class FormatXML {
    public Input inputSource = null;
    public Output outputSource = null;
    protected MetaDataRepositoryCol colData = null;
    protected MetaDataRepositoryTab tableData = null;
    public String defaultAcct = com.xweave.xmldb.Default.getReldbAcct();
public FormatXML() {
    super();
}
public FormatXML(Input input, Output output) {
    super();
    setInputSource(input);
    setOutputSource(output);
}
public static void dispatchArg(AccessSpec access, String name, String value) {
    ReportSpec rep = access.getReportSpec();
    QuerySpec query = access.getQuerySpec();
    if (name.equalsIgnoreCase("TABLENAME")) {
        query.setTableName(value);
        return;
    }
    if (name.equalsIgnoreCase("ID")) {
        query.setId(value);
        return;
    }
    if (name.equalsIgnoreCase("ACCT")) {
        access.setAcct(value);
        return;
    }
    if (name.equalsIgnoreCase("DEPTH")) {
        rep.setMaxDepth(Integer.parseInt(value));
        return;
    }
    if (name.equalsIgnoreCase("REPORTARGS")) {
        rep.setReportArgs(value);
        return;
    }
    if (name.equalsIgnoreCase("QUERYSTR")) {
        query.setConstraintString(value);
        return;
    }
    if (name.equalsIgnoreCase("STYLESHEET")) {
        access.setStyleSheet(value);
        return;
    }
    if (name.equalsIgnoreCase("SUBMIT")) {
        return;
    }
    if (name.equalsIgnoreCase("ORDERBY")) {
        query.setOrderby(value);
        return;
    }
    query.addConstraint(name, value);
}
```

```
protected MetaDataRepositoryCol getColData() {
    if (colData == null) {
        colData = new MetaDataRepositoryCol(getInputSource());
    }
    return colData;
}
/**
 * Metoda stworzona w VisualAge.
 * @return java.lang.String
 */
public String getDefaultAcct() {
    return defaultAcct;
}
public Input getInputSource() {
    if (inputSource == null) {
        setInputSource(new Input(getDefaultAcct()));
    }
    return inputSource;
}
public Output getOutputSource() {
    if (outputSource == null) {
        setOutputSource(new Output());
    }
    return outputSource;
}
protected MetaDataRepositoryTab getTableData() {
    if (tableData == null) {
        tableData = new MetaDataRepositoryTab(getInputSource());
    }
    return tableData;
}
public static void main(String args[]) {
    if (args.length < 1) {
        System.err.println("FormatXML: jako parametr należy podać <tabelę>.");
    }
    (new FormatXML()).writeDoc(args[0]);
}
public void setInputSource(Input newValue) {
    this.inputSource = newValue;
}
public void setOutputSource(Output newValue) {
    this.outputSource = newValue;
}
public void writeDoc(AccessSpec access) {
//    try {
        getOutputSource().writeln("<?xml version=\"1.0\"?>");
        if (access.getStyleSheet() != null) {
            getOutputSource().writeln("<?xml:stylesheet type=\"text/xsl\" href=\""+
access.getStyleSheet() +"\"?>");
        }
        getOutputSource().writeln("<collection>");
        writeTable(access, 1);
        getOutputSource().writeln("</collection>");
/*    } catch (Exception Ex) {
        System.out.println("Exception: " + Ex.getMessage());
    }
    */
}
```

```
public void writeDoc(String tableName) {
    writeDoc(tableName, null, null);
}
public void writeDoc(String tableName, String idColName, String id) {
    AccessSpec access= new AccessSpec();
    QuerySpec query = new QuerySpec(tableName, idColName, id);
    ReportSpec rep = new ReportSpec();
    access.setQuerySpec(query);
    access.setReportSpec(rep);
    writeDoc(access);
}
public void writeTable(AccessSpec access, int depth) {
    try {
        QuerySpec query = access.getQuerySpec();
        ReportSpec rep = access.getReportSpec();
        String name;
        String tableName = query.getTableName().toUpperCase();
        if (rep.getElementName() == null) {
            name = tableName;
        } else {
            name = rep.getElementName();
        }
        //String id = query.getId();
        String idColName = getTableData().getIdName(tableName);
        ResultSet resultSet = getInputSource().getData(access, depth);
        if (resultSet == null) {
            getOutputSource().writeln("<error msg=\"Brak danych\"
acct=\""+getInputSource().getCurrentAcct()+"\" query=\""+query+"\"/>");
            return;
        }
        ResultSetMetaData metaData = resultSet.getMetaData();
        int numCols = metaData.getColumnCount();
        String[] columnLabel = new String[numCols + 1];
        int idColNum = -1;
        for (int i = 1; i <= numCols; i++) {
            columnLabel[i] = metaData.getColumnLabel(i);
            if (columnLabel[i].equalsIgnoreCase(idColName))
                                // Sprawdza się też, czy idColName nie jest puste.
                idColNum = i;
        }
        Vector rowCache[] = new Vector[numCols + 1];
        for (int i = 1; i <= numCols; i++)
            rowCache[i] = new Vector();
        String nextId = null;
        boolean moreRows = resultSet.next();
        boolean repeatRowWrite = false;
        String currRow[] = new String[numCols + 1];
        while (moreRows) {
                        // dla każdego wiersza ustawienie currRow na resultSet
            for (int i = 1; i <= numCols; i++) {
                currRow[i] = resultSet.getString(i);
            }
            // currRow zawiera aktualny wiersz, resultSet zawiera wiersz następny
            moreRows = resultSet.next();
            if (idColNum == -1) {
                getOutputSource().writeln("<" + name + ">");
            } else {
```

```
        if (moreRows) {
            nextId = resultSet.getString(idColNum);
        } else {
            nextId = null;
        }
        // Uwaga: konieczne może być użycie małych znaków
        // w identyfikatorze, aby uniknąć konfliktu z "ID" XML.
        // Ostrzeżenie: Wartość identyfikatora nie jest podawana
        // w cudzysłowie, więc mogą to być także liczby.
        getOutputSource().writeln("<" + name + " " + idColName + "=" + '"' +
          currRow[idColNum] + '"' + ">");
    }
    if (idColNum != -1 && currRow[idColNum].equals(nextId)) {
        // join rows with shared id
        repeatRowWrite = true;
        for (int i = 1; i <= numCols; i++) {
            // inicjalizacja wiersza w cache
            rowCache[i].addElement(currRow[i]);
            // wypisanie wartości
            writeValue(name, columnLabel[i], currRow[i], access, depth + 1);
        }
        while (repeatRowWrite) {
            // aktualizacja currRow na podstawie resultSet
            // wiemy już, że taki sam jest id i że należy wiersz
            // wypisać
            for (int i = 1; i <= numCols; i++) {
                currRow[i] = resultSet.getString(i);
            }
            // wypisanie wartości i aktualizacja cache
            for (int i = 1; i <= numCols; i++) {
                if (currRow[i] != null && !rowCache[i].contains(currRow[i])) {
                    rowCache[i].addElement(currRow[i]);
                    writeValue(name, columnLabel[i], currRow[i], access, depth
                      + 1);
                }
            }
            // aktualizacja resultSet, aby zawierał następny wiersz
            moreRows = resultSet.next();
            if (moreRows) {
                nextId = resultSet.getString(idColNum);
                repeatRowWrite = currRow[idColNum].equals(nextId);
            } else {
                // jeśli bieżący wiersz był ostatnim, wypisanie go
                // i zakończenie pracy
                nextId = null;
                repeatRowWrite = false;
            }
        }
        // zwolnienie rowCache
        for (int i = 1; i <= numCols; i++)
            rowCache[i].removeAllElements();
    } else {
        // wypisanie pojedynczego wiersza
        for (int i = 1; i <= numCols; i++) {
            if (i == idColNum)
                continue;
            writeValue(name, columnLabel[i], currRow[i], access, depth + 1);
        }
```

```
            }
            getOutputSource().writeln("</" + name + ">");
        }
    } catch (SQLException Ex) {
        System.out.println("Wyjątek: " + Ex.getMessage());
    }

}
public void writeValue(String tableName, String colName, String value, AccessSpec access,
int depth) {
    if (value == null) {
        getOutputSource().write("  <" + colName + "/>");
        return;
    }
    getOutputSource().write("  <" + colName + ">");
    String recurse = getColData().getRecurse(tableName, colName);
    String idName = getColData().getIdName(tableName, colName);
    if (recurse == null || idName == null) {
        getOutputSource().write(value);
    } else {
        if (depth > access.getReportSpec().getMaxDepth()) {
            getOutputSource().write("<proxy tablename=\"" + recurse + "\"");
            if (idName != null) {
                getOutputSource().write(" idname=\"" + idName.toLowerCase() + "\"
idval=\"" + value + "\"");
            }
            getOutputSource().write("/>");
        } else {
            AccessSpec newaccess = new AccessSpec();
            newaccess.setQuerySpec(new QuerySpec(recurse, idName, value));
            newaccess.setReportSpec(access.getReportSpec());
            newaccess.setAcct(access.getAcct());
            writeTable(newaccess, depth);
        }
    }
    getOutputSource().write("</" + colName + ">\n");
}
}

/************************ Input.java ***********************/
package com.xweave.xmldb.rserve;

/**
 * Połączenie z relacyjną bazą danych.
 */
import com.xweave.xmldb.util.rdb.*;
import java.sql.*;
public class Input {
    protected RDB rdb;
    public String defaultAcct = "scott/tiger@127.0.0.1:1521:ORCL";
    public RDBAcct currentAcct = null;
public Input() {
    super();
    init(new JDBCAcct(defaultAcct));
}
public Input(RDBAcct acct) {
    super();
    init(acct);
}
```

```
public Input(String acct) {
    super();
    init(new JDBCAcct(acct));
}
public Input(Connection conn) {
    super();
    init(conn);
}
public RDBAcct getCurrentAcct() {
    return currentAcct;
}
public ResultSet getData(AccessSpec access, int depth) {
    if (currentAcct == null ||
!currentAcct.getAcct().equalsIgnoreCase(access.getAcct()))
        if (access.getAcct() != null)
            swapAcct(access.getAcct());
    QuerySpec query = access.getQuerySpec();
    String table = query.getTableName();
    String id = query.getId();
    String idColName = query.getIdColName();
    String constraint = query.getConstraintString();
    String reportargs = access.getReportSpec().getReportArgs();
    String where = null;
    if (constraint != null)
        where = constraint;
    if (id != null) {
        if (idColName == null) {
            // błąd wewnętrzny
            System.err.println("getData: no idColName for id " + id);
            // idColName = "id";
        }
        where = idColName + " = " + id;
        }
    String orderby = "";
    if (query.getOrderby() != null) {
        orderby = " ORDER BY " + query.getOrderby();
    }
    if (where == null) {
        return submitQuery("select " + reportargs + " from " + table + orderby,
depth);
    } else {
        return submitQuery("select " + reportargs + " from " + table + " where " +
where + orderby, depth);
    }
}
public ResultSet getData(String table, int depth) {
    return submitQuery("select * from "+table, depth);
}
public ResultSet getData(String table, String id, int depth) {
    if (id == null) return getData(table, depth);
    return submitQuery("select * from "+table+" where id = "+id, depth);
}
protected RDB getRDB() {
    return rdb;
}
private void init(RDBAcct acct) {
    try {
        rdb = new RDBConnector();
        rdb.connect(acct);
```

```
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
private void init(Connection conn) {
    try {
        rdb = new RDBConnector();
        rdb.connect(conn);
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
public ResultSet submitQuery(String query, int depth) {
    return ((RDBConnector) getRDB()).getData(query, depth);
}
public boolean swapAcct(String acctstring) {
    if (acctstring == null)
        return false;
    // Powinno poprawić się wielokrotne użycie połączeń do bazy danych.
    getRDB().close();
    RDB newdb;
    try {
        newdb = new RDBConnector();
        RDBAcct acct = new JDBCAcct(acctstring);
        currentAcct = acct;
        newdb.connect(acct);
        rdb = newdb;
        return true;
    } catch (SQLException ex) {
        ex.printStackTrace();
        return false;
    }
}
}

/************************ XMLServlet.java ***********************/
package com.xweave.xmldb.rserve;

import com.xweave.xmldb.util.io.*;
import com.xweave.xmldb.util.ServletBase;
import javax.servlet.http.*;
import javax.servlet.*;
import java.util.*;

public class XMLServlet extends ServletBase {
public XMLServlet() {
    super();
}
public void processRequest(HttpServletRequest req, HttpServletResponse res) throws
ServletException, java.io.IOException {
    ServletOutputStream out = startOutput(res);
    Enumeration enum = req.getParameterNames();
    String pname = null;
    AccessSpec access = new AccessSpec();
    while (enum.hasMoreElements()) {
        pname = (String) enum.nextElement();
        FormatXML.dispatchArg(access, pname, req.getParameter(pname));
    }
```

```
    if (access.getQuerySpec().getTableName() == null) {
        log("Błąd w processRequest: nie zdefiniowano tabeli dla:\n  " +
          req.getPathTranslated());
        return;
    }
    log("Zapytanie " + access.getQuerySpec().toString());
    FormatXML proc = new FormatXML();
    proc.setOutputSource(new ServletOutput());
    ((ServletOutput) proc.getOutputSource()).setOut(startOutput(res));
    //((ServletOutput) proc.getOutputSource()).getOut().println("Hello World 2!!!");
    proc.writeDoc(access);
    endOutput(res);
}
}
/********************** SPECIFICATIONS *************************/

/************************ AccessSpec.java **********************/
package com.xweave.xmldb.rserve;

/**
 * Klasa obsługuje dostęp do bazy danych, zapytania i raportowanie.
 */
public class AccessSpec {
    public String acct = null;
    public QuerySpec querySpec = null;
    public ReportSpec reportSpec = null;
    public String styleSheet = null;
public AccessSpec() {
    super();
}
public String getAcct() {
    return acct;
}
public QuerySpec getQuerySpec() {
    if (querySpec == null)
        querySpec = new QuerySpec();
    return querySpec;
}
public ReportSpec getReportSpec() {
    if (reportSpec == null)
        reportSpec = new ReportSpec();
    return reportSpec;
}
public String getStyleSheet() {
    return styleSheet;
}
public void setAcct(String newValue) {
    this.acct = newValue;
}
public void setQuerySpec(QuerySpec newValue) {
    this.querySpec = newValue;
}
public void setReportSpec(ReportSpec newValue) {
    this.reportSpec = newValue;
}
public void setStyleSheet(String newValue) {
    this.styleSheet = newValue;
}
}
```

```
/************************ QuerySpec.java ***********************/
package com.xweave.xmldb.rserve;

/**
 * Zawiera parametry zapytania.
 */
import java.util.*;

public class QuerySpec {
    public String tableName = null;
    public String id = null;
    public String constraintString = null;
    private java.util.Vector constraintSpecVector = null;
    protected String orderby = null;
    public String idColName = null;
public QuerySpec() {
    super();
}
public QuerySpec(String tableName) {
    this();
    setTableName(tableName);
}
public QuerySpec(String tableName, String idColName, String id) {
    this();
    setTableName(tableName);
    setId(id);
    setIdColName(idColName);
}
public void addConstraint(ConstraintSpec constraint) {
    getConstraintSpecVector().addElement(constraint);
}
public void addConstraint(String field, String value) {
    ConstraintSpec constraint = new ConstraintSpec(field, value);
    addConstraint(constraint);
}
public java.util.Vector getConstraintSpecVector() {
    if (constraintSpecVector == null)
        constraintSpecVector = new Vector(1);
    return constraintSpecVector;
}
public String getConstraintString() {
    StringBuffer buf = new StringBuffer();
    if (constraintString != null) {
        buf.append(constraintString);
    }
    Enumeration enum = getConstraintSpecVector().elements();
    ConstraintSpec constraint = null;
    while (enum.hasMoreElements()) {
        if (buf.length() > 0)
            buf.append(" AND ");
        constraint = (ConstraintSpec) enum.nextElement();
        buf.append(constraint.toString());
    }
    if (buf.length() > 0)
        return buf.toString();
    return null;
}
```

```
public String getId() {
    return id;
}
public String getIdColName() {
    return idColName;
}
public String getOrderby() {
    return orderby;
}
public String getTableName() {
    return tableName;
}
public void setConstraintString(String newValue) {
    this.constraintString = newValue;
}
public void setId(String newValue) {
    this.id = newValue;
}
public void setIdColName(String newValue) {
    this.idColName = newValue;
}
public void setOrderby(String newValue) {
    if (newValue.charAt(0) == '-') {
        newValue = newValue.substring(1) + " DESC";
    }
    this.orderby = newValue;
}
public void setTableName(String newValue) {
    this.tableName = newValue;
}
public String toString() {
    return getTableName() + ":" + getIdColName() + "=" + getId() + ":" +
getConstraintString();
}
}

/************************ ConstraintSpec.java ***********************/
package com.xweave.xmldb.rserve;

/**
 * Zawiera warunki ograniczające do frazy where.
 */
public class ConstraintSpec {
    public String fieldName = null;
    public String value = null;
    public String op = "=";
    protected boolean quoteValue = true;
public ConstraintSpec() {
    super();
}
public ConstraintSpec(String fieldName, String value) {
    this();
    setFieldName(fieldName);
    setValue(value);
}
public String getFieldName() {
    return fieldName;
}
```

```
public String getOp() {
    return op;
}
public String getValue() {
    return value;
}
public void setFieldName(String newValue) {
    this.fieldName = newValue;
}
public void setOp(String newValue) {
    this.op = newValue;
}
public void setValue(String newValue) {
    char char1 = newValue. charAt(0);
    // odrzucenie <,>,<=,>=,!= i przygotowanie
    if (char1 == '<' || char1 == '>' || char1== '!') {
        char char2 = newValue.charAt(1);
        if (char2 == '=') {
            setOp(String.valueOf(char1)+"=");
            newValue = newValue.substring(2);
        } else {
            setOp(String.valueOf(char1));
            newValue = newValue.substring(1);
        }
        // kontrola rozwinięcia '@'
        if (newValue.charAt(0) == '@') {
            this.quoteValue = false;
            newValue = newValue.substring(1);
        }
    } else if (char1 == '%' || newValue.charAt(newValue.length()-1) == '%') {
        // jeśli napis zaczyna się i/lub kończy '%', ustawienie op na LIKE
        setOp("LIKE");
    } else if (char1 == '*' || newValue.charAt(newValue.length()-1) == '*') {
        // jeśli napis zaczyna się i/lub kończy '*', ustawienie op na LIKE
        // Dzięki temu prostsze jest ręczne zakodowanie URL i zastąpienie
        // znaku '*' znakiem '%'.
        setOp("LIKE");
        if (char1 == '*') {
            newValue = "%" + newValue.substring(1);
        }
        if (newValue.charAt(newValue.length()-1) == '*') {
            newValue = newValue.substring(0,newValue.length()-1) + "%";
        }
    } else if (char1 == '@') {
        this.quoteValue = false;
        newValue = newValue.substring(1);
    }
    this.value = newValue;
}
public String toString() {
    if (this.quoteValue) {
        try {
            Integer.parseInt(getValue());
            // wartość jako Integer, bez cudzysłowów
            return getFieldName() + " " + getOp() + " " + getValue();
        } catch (NumberFormatException ex) {
            // wartość inna niż Integer, z cudzysłowami
        }
        return getFieldName() + " " + getOp() + " " + "'" + getValue() + "'";
```

```
    } else {
        return getFieldName() + " " + getOp() + " " + getValue();
    }
}
}

/************************ ReportSpec.java ***********************/
package com.xweave.xmldb.rserve;

/**
 * Podanie parametrów raportu.
 */
public class ReportSpec {
    public int currentDepth = 0;
    public int maxDepth = 1;
    protected String reportArgs = "*";
    protected String elementName = null;
public ReportSpec() {
    super();
}
public String getElementName() {
    return elementName;
}
public int getMaxDepth() {
    return maxDepth;
}
public String getReportArgs() {
    return reportArgs;
}
public void setElementName(String newValue) {
    this.elementName = newValue;
}
public void setMaxDepth(int newValue) {
    this.maxDepth = newValue;
}
public void setReportArgs(String newValue) {
    this.reportArgs = newValue;
}
}

/********************** METADANE *************************/

/************************ MetaDataRepositoryTab.java ***********************/
package com.xweave.xmldb.rserve;

import java.util.*;
import java.sql.*;
import com.xweave.xmldb.util.rdb.*;
/**
 * Zawiera metadane o tabelach relacyjnych.
 */
public class MetaDataRepositoryTab extends Hashtable {
    public static MetaDataRepositoryTab defaultInstance = null;
    public String primaryKeyQuery = null;
    public final static String TABLE_INFO_TABLE = "XDB_RS_TAB_INFO";
public MetaDataRepositoryTab(int initialCapacity) {
    super(initialCapacity);
}
```

```
public MetaDataRepositoryTab(int initialCapacity, float loadFactor) {
    super(initialCapacity, loadFactor);
}
public MetaDataRepositoryTab(Input in) {
    super();
    init(((RDBConnector) in.getRDB()));
}
public void addPK(String table, String idName) {
    get(table).setIdName(idName);
}
public boolean exists(String table) {
    return super.get(table.toUpperCase()) != null;
}
public MetaDataRecordTab get(String table) {
    MetaDataRecordTab rec = (MetaDataRecordTab) super.get(table.toUpperCase());
    if (rec == null) {
        rec = new MetaDataRecordTab();
        put(table,rec);
    }
    return rec;
}
public String getIdName(String table) {
    String val = get(table).getIdName();
    if (val != null) {
        return val;
    }
    int index = table.indexOf('.');
    String owner = null;
    if (index != -1) {
        //strip owner, and try again
        owner = table.substring(0, index);
        table = table.substring(index + 1);
        val = get(table).getIdName();
        if (val != null) {
            return val;
        }
    }
    if (table.endsWith("_V")) {
        val = get(table.substring(table.length() - 1)).getIdName();
        if (val != null) {
            return val;
        }
    }
    index = table.indexOf("_V_");
    if (index != -1) {
        val = get(table.substring(0, index)).getIdName();
        if (val != null) {
            return val;
        }
    }
    return null;
}
public String getPrimaryKeyQuery(RDB rdb) {
    RDBConnector rdbconn = ((RDBConnector) rdb);
    // powinno być otwarte, ale lepiej sprawdzić
    if (rdb.isClosed()) {
        // jeśli nie otwarto, tabela pozostaje pusta
```

```
            System.err.println("Nie otwarto repozytorium dla " + rdbconn.getAcct());
            return null;
        }
        // Sprawdzenie połączenia dla podanego typu bazy danych.
        String dbclass = rdbconn.getConnection().getClass().getName();
        if (dbclass.indexOf("oracle") != -1) {
            return getPrimaryKeyQueryOracle();
        }
        if (dbclass.indexOf("db2") != -1) {
            return getPrimaryKeyQueryDB2();
        }
        return null;
    }
    public String getPrimaryKeyQueryDB2() {
        // błąd
        StringBuffer buf = new StringBuffer();
        buf.append("select c.tabname child_table,");
        buf.append("       c.colname child_column,");
        buf.append("       p.tabname parent_table,");
        buf.append("       p.colname parent_column");
        buf.append("  from syscat.references l,");
        buf.append("       syscat.keycoluse c,");
        buf.append("       syscat.keycoluse p");
        buf.append(" where l.constname = c.constname");
        buf.append("   and l.refkeyname = p.constname");
        buf.append("   and l.tabschema = c.tabschema");
        buf.append("   and l.reftabschema = p.tabschema");
        buf.append("   and c.colseq = p.colseq");
        return buf.toString();
    }
    public String getPrimaryKeyQueryOracle() {
        // błąd
        StringBuffer buf = new StringBuffer();
        // Uwaga: jeśli all_constraints zawiera nazwy tabel i kolumn dzielone
        // między różnych użytkowników z kluczami obcymi odwołującymi się
        // do różnych tabel, uzyskane wyniki mogą być fragmentaryczne. Można
        // ten problem rozwiązać, stosując user_constraints i user_cons_constraints.
        buf.append("select c.table_name,                    ");
        buf.append("       c.column_name                    ");
        buf.append("  from sys.all_constraints l,           ");
        buf.append("       sys.all_cons_columns c           ");
        buf.append(" where l.constraint_type = 'P'          ");
        buf.append("   and l.constraint_name = c.constraint_name");
        buf.append("   and l.owner = c.owner                ");
        return buf.toString();
    }
    public void init(RDB rdb) {
        ResultSet res;
        ResultSetMetaData metaData;
        String tab;
        // Pobranie kluczy obcych z tabel systemowych.
        try {
            res = rdb.getData(getPrimaryKeyQuery(rdb));
            metaData = res.getMetaData();
            int numCols = metaData.getColumnCount();
            while (res.next()) {
                tab = res.getString(1);
```

```
            if (exists(tab)) {
                // najprawdopodobniej wielokolumnowy klucz główny
                // wielokolumnowe klucze główne nie są obsługiwane, usuwane
                // jest odniesienie do klucza głównego
                addPK(tab, null);
            } else {
                addPK(tab, res.getString(2));
            }
        }
        // this.print();
    } catch (Exception Ex) {
        System.out.println("Wyjątek: " + Ex.getMessage());
    }
    // Jeśli dostępne są dodatkowe wskazówki co do analizy danych,
    // są teraz pobierane.
    try {
        String queryStr = "select * from " + TABLE_INFO_TABLE;
        res = rdb.executeQuery(queryStr);
        metaData = res.getMetaData();
        int numCols = metaData.getColumnCount();
        while (res.next()) {
            addPK(res.getString(1), res.getString(2));
        }
    } catch (Throwable Ex) {
        // jeśli nie ma wskazówek, nie ma błędu
    }

}
public void print() {
    Enumeration e = this.keys();
    String key;
    MetaDataRecordTab rec;
    while (e.hasMoreElements()) {
        key = (String) e.nextElement();
        rec = (MetaDataRecordTab) get(key);
        System.out.println(key +"\t"+ rec.getIdName());
    }
}
public Object put(String table, Object value) {
    return super.put(table.toUpperCase(),value);
}
public void setPrimaryKeyQuery(String newValue) {
    this.primaryKeyQuery = newValue;
}
}

/************************ MetaDataRecordTab.java ***********************/
package com.xweave.xmldb.rserve;

/**
 * Rekord metadanych z tabel relacyjnych.
 */
public class MetaDataRecordTab {
    public String idName;
public MetaDataRecordTab() {
    super();
}
```

```
public String getIdName() {
    return idName;
}
public void setIdName(String newValue) {
    this.idName = newValue;
}
public String toString() {
    return "("+getIdName()+")";
}
}

/************************ MetaDataRepositoryCol.java **********************/
package com.xweave.xmldb.rserve;

import java.util.*;
import java.sql.*;
import com.xweave.xmldb.util.rdb.*;
/**
 * Zawiera metadane o relacyjnych kolumnach.
 */
public class MetaDataRepositoryCol extends Hashtable {
    public static MetaDataRepositoryCol defaultInstance = null;
    public String foreignKeyQuery = null;
    public final static String COLUMN_INFO_TABLE = "XDB_RS_COL_INFO";
public MetaDataRepositoryCol(int initialCapacity) {
    super(initialCapacity);
}
public MetaDataRepositoryCol(int initialCapacity, float loadFactor) {
    super(initialCapacity, loadFactor);
}
public MetaDataRepositoryCol(Input in) {
    super();
    init(((RDBConnector) in.getRDB()));
}
public void addRecurse(String table, String col, String parent, String id) {
    get(table, col).setRecurse(parent, id);
}
public boolean exists(String table, String col) {
    return get(table.toUpperCase()+"."+col.toUpperCase()) != null;
}
public MetaDataRecordCol get(String table, String col) {
    MetaDataRecordCol rec = (MetaDataRecordCol)
get(table.toUpperCase()+"."+col.toUpperCase());
    if (rec == null) {
        rec = new MetaDataRecordCol();
        put(table,col,rec);
    }
    return rec;
}
public String getForeignKeyQuery(RDB rdb) {
    RDBConnector rdbconn = ((RDBConnector) rdb);
    // powinno być aktywne, ale lepiej sprawdzić
    if (rdb.isClosed()) {
        //if not open, table remains empty
        System.err.println("Repository is not open for " + rdbconn.getAcct());
        return null;
    }
```

```
        // Sprawdzenie połączenia z konkretnym typem bazy danych.
        String dbclass = rdbconn.getConnection().getClass().getName();
        if (dbclass.indexOf("oracle") != -1) {
            return getForeignKeyQueryOracle();
        }
        if (dbclass.indexOf("db2") != -1) {
            return getForeignKeyQueryDB2();
        }
        return null;
    }
    public String getForeignKeyQueryDB2() {
        StringBuffer buf = new StringBuffer();
        buf.append("select c.tabname child_table,");
        buf.append("       c.colname child_column,");
        buf.append("       p.tabname parent_table,");
        buf.append("       p.colname parent_column");
        buf.append("  from syscat.references l,");
        buf.append("       syscat.keycoluse c,");
        buf.append("       syscat.keycoluse p");
        buf.append(" where l.constname = c.constname");
        buf.append("   and l.refkeyname = p.constname");
        buf.append("   and l.tabschema = c.tabschema");
        buf.append("   and l.reftabschema = p.tabschema");
        buf.append("   and c.colseq = p.colseq");
        return buf.toString();
    }
    public String getForeignKeyQueryOracle() {
        StringBuffer buf = new StringBuffer();
        // Uwaga: jeśli all_constraints zawiera nazwy tabel i kolumn dzielone
        // między różnych użytkowników z kluczami obcymi odwołującymi się
        // do różnych tabel, uzyskane wyniki mogą być fragmentaryczne. Można
        // ten problem rozwiązać, stosując user_constraints i user_cons_constraints.
        buf.append("select c.table_name child_table,        ");
        buf.append("       c.column_name child_column,      ");
        buf.append("       p.table_name parent_table,       ");
        buf.append("       p.column_name parent_column      ");
        buf.append("  from sys.all_constraints l,           ");
        buf.append("       sys.all_cons_columns c,          ");
        buf.append("       sys.all_cons_columns p           ");
        buf.append(" where l.constraint_type = 'R'          ");
        buf.append("   and l.constraint_name = c.constraint_name");
        buf.append("   and l.r_constraint_name = p.constraint_name");
        buf.append("   and l.owner = c.owner                ");
        buf.append("   and l.r_owner = p.owner              ");
        buf.append("   and c.position = p.position          ");
        //    buf.append("   and p.column_name = 'ID'             ");
        return buf.toString();
    }
    public String getIdName(String table, String col) {
        String val = get(table, col).getIdName();
        if (val != null) {
            return val;
        }
        return null;
    }
    public String getRecurse(String table, String col) {
        String val = get(table, col).getRecurse();
        if (val != null) {
```

```
            return val;
        }
        int index = table.indexOf('.');
        String owner = null;
        if (index != -1) {
            // odrzucenie użytkownika, ponowna próba
            owner = table.substring(0, index);
            table = table.substring(index + 1);
            val = get(table, col).getRecurse();
            if (val != null) {
                if (owner != null) {
                    return owner + "." + val;
                }
                return val;
            }
        }
        if (table.endsWith("_V")) {
            val = get(table.substring(table.length() - 1), col).getRecurse();
            if (val != null) {
                if (owner != null) {
                    return owner + "." + val;
                }
                return val;
            }
        }
        index = table.indexOf("_V_");
        if (index != -1) {
            val = get(table.substring(0, index), col).getRecurse();
            if (val != null) {
                if (owner != null) {
                    return owner + "." + val;
                }
                return val;
            }
        }
        return null;
    }
    public void init(RDB rdb) {
        ResultSet res;
        ResultSetMetaData metaData;
        String tab, col;
        // Pobranie z tabel systemowych kluczy obcych.
        try {
            res = rdb.getData(getForeignKeyQuery(rdb));
            metaData = res.getMetaData();
            int numCols = metaData.getColumnCount();
            while (res.next()) {
                tab = res.getString(1);
                col = res.getString(2);
                if (exists(tab, col)) {
                    // najprawdopodobniej wielokolumnowy klucz obcy
                    // wielokolumnowe klucze obce nie są obsługiwane, usunięcie
                    // odwołania do tego klucza
                    addRecurse(tab, col, null, null);
                } else {
                    addRecurse(tab, col, res.getString(3), res.getString(4));
                }
            }
```

```
        // this.print();
    } catch (Exception Ex) {
        System.out.println("Wyjątek: " + Ex.getMessage());
    }
    // Jeśli dostępne są dodatkowe wskazówki dotyczące analizy danych, pobranie ich
    try {
        String queryStr = "select * from " + COLUMN_INFO_TABLE;
        res = rdb.executeQuery(queryStr);
        metaData = res.getMetaData();
        int numCols = metaData.getColumnCount();
        while (res.next()) {
            addRecurse(res.getString(1), res.getString(2), res.getString(3),
res.getString(4));
        }
    } catch (Throwable Ex) {
        // nie ma wskazówek, nie ma błędu
    }

}
public void print() {
    Enumeration e = this.keys();
    String key;
    MetaDataRecordCol rec;
    while (e.hasMoreElements()) {
        key = (String) e.nextElement();
        rec = (MetaDataRecordCol) get(key);
        System.out.println(key +"\t"+ rec.getRecurse() +"\t"+ rec.getIdName());
    }
}
public Object put(String table, String col, Object value) {
    return put(table.toUpperCase()+"."+col.toUpperCase(),value);
}
public void setForeignKeyQuery(String newValue) {
    this.foreignKeyQuery = newValue;
}
}

/************************ MetaDataRecordCol.java **********************/
package com.xweave.xmldb.rserve;

/**
 * Rekord metadanych o kolumnach relacyjnych.
 */
public class MetaDataRecordCol {
    public String idName;
    public String tableName;
public MetaDataRecordCol() {
    super();
}
public String getIdName() {
    return idName;
}
public String getRecurse() {
    return tableName;
}
public String getTableName() {
    return tableName;
}
```

```
public void setIdName(String newValue) {
    this.idName = newValue;
}
public void setRecurse(String table, String id) {
    this.tableName = table;
    this.idName = id;
}
public void setTableName(String newValue) {
    this.tableName = newValue;
}
public String toString() {
    return getRecurse()+"("+getIdName()+")";
}
}
```

Główna klasa *rServe*, `Demo`, przekształca dane relacyjne ze źródła wskazanego przez klasę `Input` i zapisuje dane do strumienia wskazanego przez klasę `Output`. Informacje o zapytaniu, koncie i formatowaniu danych pobierane są przez klasę kontener `AccessSpec` i jej klasy pomocnicze, `QuerySpec` oraz `ReportSpec`. Klasa `QuerySpec` zawiera warunki umieszczane we frazie `where` zapytania SQL, klasa pomocnicza `ConstraintSpec` jest używana do zapisania poszczególnych warunków. Klasa `Input` formatuje zapytanie SQL, przy czym do formatowania frazy `where` używa się klasy `QuerySpec`. Poza tym klasa `Input` komunikuje się z relacyjną bazą danych (za pomocą kolejnej klasy pomocniczej z *com.xweave.xmldb.util.rdb*, opisanej w dodatku A).

Zapytanie wykonywane jest przez utworzenie obiektu `AccessSpec` i przekazanie go metodzie `writeDoc` klasy `FormatXML`. Metoda `writeDoc` zapisuje nagłówek dokumentu i znacznik początkowy oraz końcowy elementu głównego, wywołuje metodę `writeTable`. Ta metoda z kolei realizuje algorytm opisany w punkcie 5.3.3. Do pobrania danych z bazy używany jest obiekt `Input` (przechowywany w polu `inputSource`), zaś obiekt `Output` (z pola `outputSource`) generuje sformatowany kod XML. Metoda `writeTable` używa metody `writeValue` do wypisania kolejnych danych, ta ostatnia może rekursywnie wywoływać metodę `writeTable` w przypadku odwoływania się do innych tabel przez klucze obce.

Każda instancja klasy `MetaDataRecordTab` zawiera opis kluczy głównych jednej tabeli. Wszystkie instancje składają się na repozytorium metadanych tabel, które jest inicjalizowane przy pierwszej próbie dostępu do danych. `MetaDataRepositoryTab` jest tablicą asocjacyjną, zawierającą jako wartości obiekty `MetaDataRecordTab`, a jako klucze nazwy tabel i kolumn kluczy głównych. Każdy obiekt `MetaDataRecordTab` zawiera nazwę tabeli i nazwę pola klucza głównego, wskazywanych w kluczu. Istnieje też metoda JDBC, której używa się, by sięgać do systemowej tabeli kluczy głównych w celu inicjalizacji. My nie używaliśmy jej, tak by przedstawiane rozwiązanie można było łatwiej zastosować w połączeniu z innymi protokołami.

Informacje o kluczach obcych zapisywane są w obiektach klasy `MetaDataRecordCol`. `MetaDataRepositoryCol` to tablica asocjacyjna zawierająca jako wartości obiekty `MetaDataRecordCol`, a jako klucze nazwy kluczy obcych używanych do przechodzenia do pozostałych danych. Każdy obiekt `MetaDataRecordCol` zawiera nazwę tabeli i nazwę kolumny klucza głównego wskazywanej przez klucz obcy.

5.4. Serwer danych XML

Serwer danych XML łączy funkcje bazy danych XML i serwera sieciowego. Użytkownicy i aplikacje, sięgając do danych za pośrednictwem serwera danych XML, korzystają z interfejsu sieciowego.

W tym punkcie przedstawimy jako przykład serwer danych XML *xServe*, który pozwala zapisywać, edytować i wyszukiwać dane z bazy danych XML za pomocą mechanizmów relacyjnych (zgodnie z zasadami opisanymi w rozdziale 4.). Serwer ten realizuje model danych scharakteryzowany w rozdziale 3. i udostępnia API przez adres URL, podobnie jak relacyjny serwer danych opisany w poprzednim punkcie.

Oprócz operacji na dokumentach (opisane w rozdziale 3.) są dostępne analogiczne operacje dotyczące fragmentów dokumentów (odpowiadają poszczególnym elementom z dokumentu). Możliwe więc jest pobieranie, usuwanie i aktualizowanie elementów.

Na przykładzie tego konkretnego serwera danych XML zaprezentowano niektóre funkcje oferowane przez system baz danych XML dostępny w Sieci. Wprawdzie rozwiązanie to należy traktować jako prosty prototyp, jednak można go używać do zapisywania i pobierania dokumentów XML lub zapisywania danych zorientowanych hierarchicznie. Przykład ten posłużył także do pokazania sposobu tworzenia systemu baz danych XML i realizacji niektórych funkcji w praktyce.

Serwer *xServe* „rozumie" następujące polecenia:

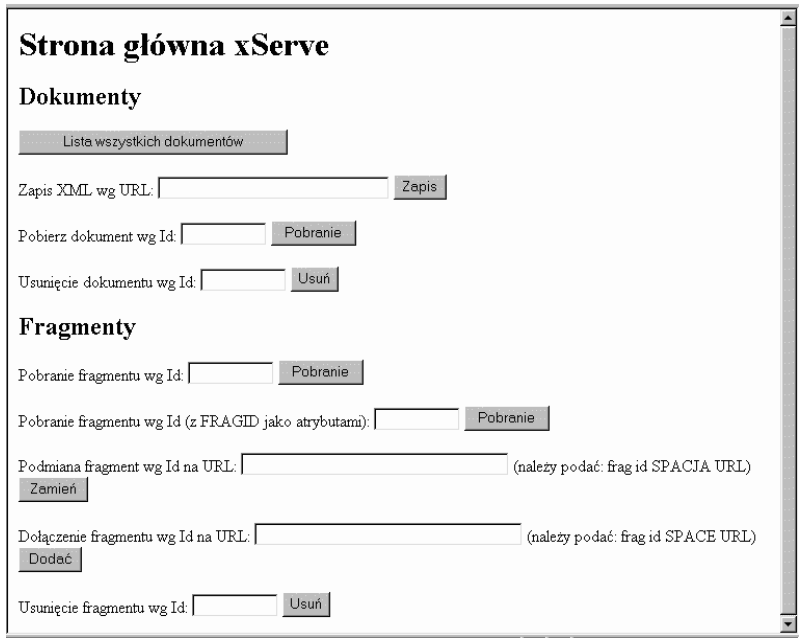
- Zapisanie dokumentu spod podanego adresu URL.
- Pobranie dokumentu lub jego fragmentu z bazy danych.
- Pobranie dokumentu jako fragmentu lub dowolnego fragmentu jako całego dokumentu. W ten sposób na podstawie innych dokumentów i ich fragmentów można tworzyć nowe dokumenty.
- Aktualizacja dokumentu przez zastąpienie jego fragmentu innym dokumentem lub jego fragmentem spod podanego adresu URL (można użyć także adresu *xServe*).
- Dołączenie fragmentu XML spod wskazanego adresu URL do danego fragmentu.
- Usunięcie dokumentu lub jego fragmentu z bazy danych.

Przydatne mogłyby być inne polecenia, na przykład zmiana kolejności elementów w dokumencie, wyszukiwanie dokumentów według podanych warunków lub realizacja innych operacji opisanych w rozdziale 3.

Stronę główną *xServe* pokazano na rysunku 5.7, jej kod źródłowy HTML zawiera wydruk 5.14. Za pomocą formularza HTML tworzy się adresy URL, sięgające do bazy danych XML. Oto przykłady adresów URL:

- *http://127.0.0.1/servlets/com.xweave.xmldb.xserve.XMLServlet?list=1*
- *http://127.0.0.1/servlets/com.xweave.xmldb.xserve.XMLServlet?store=http://127.0.0.1/temp1.xml*

Rysunek 5.7. *Strona główna xServe*



- *http://127.0.0.1/servlets/com.xweave.xmldb.xserve.XMLServlet?store=D:\Temp\cancer.xml*
- *http://127.0.0.1/servlets/com.xweave.xmldb.xserve.XMLServlet?getfragdoc=7.1*
- *http://127.0.0.1/servlets/com.xweave.xmldb.xserve.XMLServlet?appendfrag=7.1 + http://127.0.0.1/servlets/com.xweave.xmldb.xserve.XMLServlet?retrieve=4*
- *http://127.0.0.1/servlets/com.xweave.xmldb.xserve.XMLServlet?deletefrag=1.2*

Wydruk 5.14. *Strona główna xServe*

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html">
<title>Strona główna xServe</title>
</head>

<body>
<h1>Strona główna xServe</h1>
<h2>Dokumenty</h2>

  <form action="http://127.0.0.1:8080/servlet/com.xweave.xmldb.xserve.XMLServlet"
        method="GET">
    <input type="hidden" name="list" value="1">
    <p><input type="submit" name="ignore" value="Lista wszystkich dokumentów">
  </form>

  <form action="http://127.0.0.1:8080/servlet/com.xweave.xmldb.xserve.XMLServlet"
        method="GET">
  <p> Zapis XML wg URL:
  <input type="input" name="store" size=30>
```

```
  <input type="submit" name="ignore" value="Zapis">
  </form>

  <form action="http://127.0.0.1:8080/servlet/com.xweave.xmldb.xserve.XMLServlet"
        method="GET">
  <p> Pobierz dokument wg Id:
  <input type="input" name="retrievedoc" size=10>
  <input type="submit" name="ignore" value="Pobranie">
  </form>

  <form action="http://127.0.0.1:8080/servlet/com.xweave.xmldb.xserve.XMLServlet"
        method="GET">
  <p> Usunięcie dokumentu wg Id:
  <input type="input" name="delete" size=10>
  <input type="submit" name="ignore" value="Usuń">
  </form>

<h2>Fragmenty</h2>

  <form action="http://127.0.0.1:8080/servlet/com.xweave.xmldb.xserve.XMLServlet"
        method="GET">
  <p> Pobranie fragmentu wg Id:
  <input type="input" name="getfragdoc" size=10>
  <input type="submit" name="ignore" value="Pobranie">
  </form>

  <form action="http://127.0.0.1:8080/servlet/com.xweave.xmldb.xserve.XMLServlet"
        method="GET">
  <p> Pobranie fragmentu wg Id (z FRAGID jako atrybutami):
  <input type="input" name="getfragiddoc" size=10>
  <input type="submit" name="ignore" value="Pobranie">
  </form>

  <form action="http://127.0.0.1:8080/servlet/com.xweave.xmldb.xserve.XMLServlet"
        method="GET">
  <p> Podmiana fragmentu wg Id na URL:
  <input type="input" name="setfrag" size=35>
  (należy podać: frag id SPACJA URL)
  <input type="submit" name="ignore" value="Zamień">
  </form>

  <form action="http://127.0.0.1:8080/servlet/com.xweave.xmldb.xserve.XMLServlet"
        method="GET">
  <p> Dołączenie fragmentu wg Id na URL:
  <input type="input" name="appendfrag" size=35>
  (należy podać: frag id SPACE URL)
  <input type="submit" name="ignore" value="Dodać">
  </form>

  <form action="http://127.0.0.1:8080/servlet/com.xweave.xmldb.xserve.XMLServlet"
        method="GET">
  <p> Usunięcie fragmentu wg Id:
  <input type="input" name="deletefrag" size=10>
  <input type="submit" name="ignore" value="Usuń">
  </form>

</body>
</html>
```

5.4.1. Implementacja

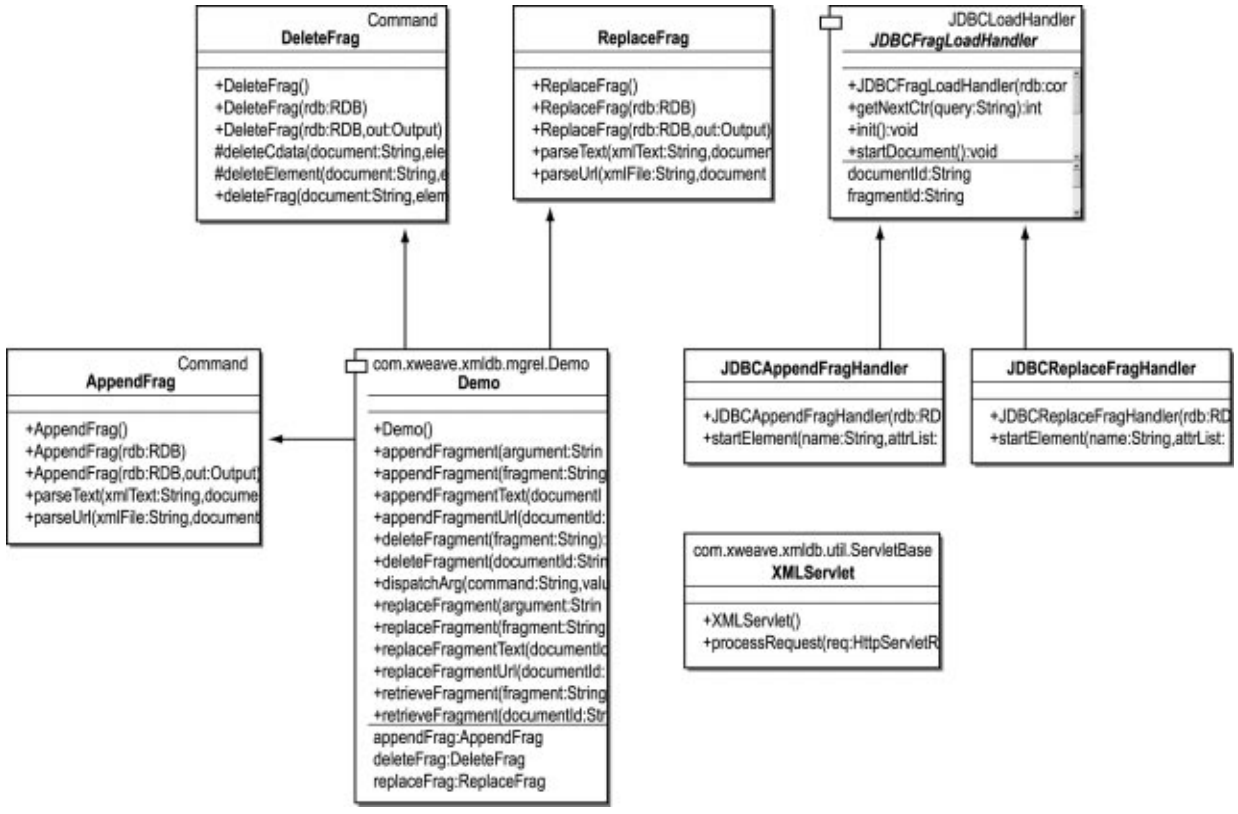
Kod języka Java, stanowiący implementację relacyjnego serwera danych *xServe*, pokazuje wydruk 5.15. Diagram klas pokazano na rysunku 5.8.

Wydruk 5.15. *Kod Java serwera danych XML xServe*

```
/************************ Demo.java ***********************/
package com.xweave.xmldb.xserve;

import com.xweave.xmldb.util.io.*;
import com.xweave.xmldb.util.rdb.*;
import com.xweave.xmldb.fgrel.*;
/**
 * Demonstracja serwera danych XML przy użyciu drobnoziarnistej struktury
 * relacyjnej w Oracle.
 */
public class Demo extends com.xweave.xmldb.mgrel.Demo {
    private DeleteFrag deleteFrag = null;
    private AppendFrag appendFrag = null;
    private ReplaceFrag replaceFrag = null;
public Demo() {
    super();
}
/**
 * Dołączenie fragmentu XML w id <documentId>.<elementId><space><xmlFile>
 */
public boolean appendFragment(String argument) {
    String documentId, elementId, xmlFile;
    int dotindex = argument.indexOf('.');
    int spaceindex = argument.indexOf(' ');
    if (spaceindex == -1) {
        return false;
    }
    if (dotindex == -1) {
        documentId = argument.substring(0, spaceindex);
        elementId = "1";
    } else {
        documentId = argument.substring(0, dotindex);
        elementId = argument.substring(dotindex + 1, spaceindex);
    }
    xmlFile = argument.substring(spaceindex + 1);
    return this.appendFragmentUrl(documentId, elementId, xmlFile);
}
/**
 * Dołączenie fragmentu id <documentId>.<elementId>
 */
public boolean appendFragment(String fragment, String xmlFile) {
    String documentId, elementId;
    int dotindex = fragment.indexOf('.');
    if (dotindex == -1) {
        documentId = fragment;
        elementId = "1";
    } else {
        documentId = fragment.substring(0, dotindex);
        elementId = fragment.substring(dotindex + 1);
    }
```

Rysunek 5.8.
Diagram klas UML serwera danych XML xServe



```
return this.appendFragmentUrl(documentId, elementId, xmlFile);
}
/**
 * Dodanie fragmentu w id <documentId> <elementId>
 */
public boolean appendFragmentText(String documentId, String elementId, String xmlFile)
{
    return getAppendFrag().parseText(xmlFile, documentId, elementId);
}
/**
 * Dodanie fragmentu w id <documentId> <elementId>
 */
public boolean appendFragmentUrl(String documentId, String elementId, String xmlFile)
{
    return getAppendFrag().parseUrl(xmlFile, documentId, elementId);
}
/**
 * Usunięcie fragmentu XML z id <documentId>.<elementId>
 */
public boolean deleteFragment(String fragment) {
    String documentId, elementId;
    int dotindex = fragment.indexOf('.');
    if (dotindex == -1) {
        documentId = fragment;
        elementId = "1";
    } else {
        documentId = fragment.substring(0, dotindex);
        elementId = fragment.substring(dotindex + 1);
    }
    return this.deleteFragment(documentId, elementId);
}
/**
 * Usunięcie fragmentu XML z id <documentId>.<elementId>
 */
public boolean deleteFragment(String documentId, String elementId) {
    return getDeleteFrag().deleteFrag(documentId, elementId);
}
public boolean dispatchArg(String command, String value) {
    if (command.equalsIgnoreCase("GETFRAG")) {
        this.retrieveFragment(value);
        return true;
    }
    if (command.equalsIgnoreCase("GETFRAGDOC")) {
        this.getOutput().writeln("<?xml version=\"1.0\"?>");
        this.retrieveFragment(value);
        return true;
    }
    if (command.equalsIgnoreCase("SETFRAG")) {
        this.replaceFragment(value);
        return true;
    }
    if (command.equalsIgnoreCase("APPENDFRAG")) {
        this.appendFragment(value);
        return true;
    }
    if (command.equalsIgnoreCase("DELETEFRAG")) {
        this.deleteFragment(value);
        return true;
    }
```

```
    if (command.equalsIgnoreCase("GETFRAGIDDOC")) {
        this.getOutput().writeln("<?xml version=\"1.0\"?>");
        this.getFormat().setWriteElementId(true);
        this.retrieveFragment(value);
        this.getFormat().setWriteElementId(false);
        return true;
    }
    return super.dispatchArg(command, value);
}
/**
 * Pobranie obiektu polecenia dołączenia fragmentu.
 */
public AppendFrag getAppendFrag() {
    if (appendFrag == null) {
        appendFrag = new AppendFrag(getRdb(), getOutput());
    }
    return appendFrag;
}
/**
 * Pobranie obiektu polecenia usunięcia fragmentu.
 */
public DeleteFrag getDeleteFrag() {
    if (deleteFrag == null) {
        deleteFrag = new DeleteFrag(getRdb(), getOutput());
    }
    return deleteFrag;
}
/**
 * Pobranie obiektu polecenia zastąpienia fragmentu.
 */
public ReplaceFrag getReplaceFrag() {
    if (replaceFrag == null) {
        replaceFrag = new ReplaceFrag(getRdb(), getOutput());
    }
    return replaceFrag;
}
/**
 * Zapisanie fragmentu XML w id <documentId>.<elementId><space><xmlFile>
 */
public boolean replaceFragment(String argument) {
    String documentId, elementId, xmlFile;
    int dotindex = argument.indexOf('.');
    int spaceindex = argument.indexOf(' ');
    if (spaceindex == -1) {
        return false;
    }
    if (dotindex == -1) {
        documentId = argument.substring(0, spaceindex);
        elementId = "1";
    } else {
        documentId = argument.substring(0, dotindex);
        elementId = argument.substring(dotindex + 1, spaceindex);
    }
    xmlFile = argument.substring(spaceindex + 1);
    return this.replaceFragmentUrl(documentId, elementId, xmlFile);
}
```

```
/**
 * Zapisanie fragmentu XML w id <documentId>.<elementId>
 */
public boolean replaceFragment(String fragment, String xmlFile) {
    String documentId, elementId;
    int dotindex = fragment.indexOf('.');
    if (dotindex == -1) {
        documentId = fragment;
        elementId = "1";
    } else {
        documentId = fragment.substring(0, dotindex);
        elementId = fragment.substring(dotindex + 1);
    }
    return this.replaceFragmentUrl(documentId, elementId, xmlFile);
}
/**
 * Zastąpienie fragmentu XML w id <documentId> <elementId>
 */
public boolean replaceFragmentText(String documentId, String elementId, String
xmlText) {
    return getReplaceFrag().parseText(xmlText, documentId, elementId);
}
/**
 * Zastąpienie fragmentu XML w id <documentId> <elementId>
 */
public boolean replaceFragmentUrl(String documentId, String elementId, String xmlFile)
{
    return getReplaceFrag().parseUrl(xmlFile, documentId, elementId);
}
/**
 * Sformatowanie fragmentu XML z id <documentId>.<elementId>
 */
public boolean retrieveFragment(String fragment) {
    String documentId, elementId;
    int dotindex = fragment.indexOf('.');
    if (dotindex == -1) {
        documentId = fragment;
        elementId = "1";
    } else {
        documentId = fragment.substring(0, dotindex);
        elementId = fragment.substring(dotindex + 1);
    }
    return this.retrieveFragment(documentId, elementId);
}
/**
 * Sformatowanie fragmentu XML z id <documentId>.<elementId>
 */
public boolean retrieveFragment(String documentId, String elementId) {
    return getFormat().writeFrag(documentId, elementId);
}
}

/************************ XMLServlet.java ***********************/
package com.xweave.xmldb.xserve;

import javax.servlet.*;
import javax.servlet.http.*;
```

```
import java.io.*;
import java.util.*;
import com.xweave.xmldb.util.io.ServletOutput;
/**
 * Interfejs serwleta do bazy danych.
 */
public class XMLServlet extends com.xweave.xmldb.util.ServletBase {
/**
 * Konstruktor XMLServlet.
 */
public XMLServlet() {
    super();
}
public void processRequest(HttpServletRequest req, HttpServletResponse res) throws
  ServletException, java.io.IOException {
    ServletOutputStream out = startOutput(res);
    Enumeration enum = req.getParameterNames();
    String pname, pvalue = null;
    Demo db = new Demo();
    db.setOutput(new ServletOutput(out));
    db.setRdbAccessString(com.xweave.xmldb.Default.getXmldbAcct());
    db.connect();
    while (enum.hasMoreElements()) {
        pname = (String) enum.nextElement();
        pvalue = req.getParameter(pname);
        log("Executing " + pname + " " + pvalue);
        db.dispatchArg(pname, pvalue);
    }
    if (db.getRdbAccessString() == null) {
        log("Błąd w processRequest: nie zdefiniowano ciągu wejścia dla:\n  " +
          req.getPathTranslated());
    }
    endOutput(res);
}
}

/********************** POLECENIA **************************/

/************************ AppendFrag.java **********************/
package com.xweave.xmldb.xserve;

import com.xweave.xmldb.fgrel.*;
import com.xweave.xmldb.util.rdb.RDB;
import com.xweave.xmldb.util.io.Output;
import java.sql.*;
import org.xml.sax.*;
import org.xml.sax.helpers.ParserFactory;
import com.ibm.xml.parsers.DOMParser;
import org.w3c.dom.Document;
import java.io.IOException;

/**
 * Polecenie dołączania fragmentu w bazie danych.
 */
public class AppendFrag extends Command {
public AppendFrag() {
    super();
}
```

```
public AppendFrag(RDB rdb) {
    super(rdb);
}
public AppendFrag(RDB rdb, Output out) {
    super(rdb, out);
}
/**
 * Parsowanie pliku XML za pomocą parsera SAX i zapisanie go jako fragmentu.
 * Uwaga: parser z <parserClass> MUSI być dostępny w ścieżce CLASSPATH.
 */
public boolean parseText(String xmlText, String documentId, String elementId) {
    String parserClass = com.xweave.xmldb.Default.getParserClass();
    try {
        Parser parser = ParserFactory.makeParser(parserClass);
        HandlerBase handler;
        if (getRdb() == null) {
            return false;
        } else {
            handler = new JDBCAppendFragHandler(getRdb());
        }
        parser.setDocumentHandler(handler);
        parser.setErrorHandler(handler);
        try {
            ((LoadHandler) handler).setDocumentName("Frag"+documentId+"."+"elementId");
            ((JDBCAppendFragHandler) handler).setDocumentId(documentId);
            ((JDBCAppendFragHandler) handler).setFragmentId(elementId);
            parser.parse(new InputSource(new java.io.StringReader(xmlText)));
            return true;
        } catch (SAXException se) {
            getOutput().writeln(se.toString());
            se.printStackTrace();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    } catch (ClassNotFoundException ex) {
        ex.printStackTrace();
    } catch (IllegalAccessException ex) {
        ex.printStackTrace();
    } catch (InstantiationException ex) {
        ex.printStackTrace();
    }
    return false;
}
/**
 * Parsowanie pliku XML za pomocą parsera SAX i zapisanie go jako fragmentu.
 * Uwaga: parser z <parserClass> MUSI być dostępny w ścieżce CLASSPATH.
 */
public boolean parseUrl(String xmlFile, String documentId, String elementId) {
    String parserClass = com.xweave.xmldb.Default.getParserClass();
    try {
        Parser parser = ParserFactory.makeParser(parserClass);
        HandlerBase handler;
        if (getRdb() == null) {
            return false;
        } else {
            handler = new JDBCAppendFragHandler(getRdb());
        }
```

```
            parser.setDocumentHandler(handler);
            parser.setErrorHandler(handler);
            try {
                ((LoadHandler) handler).setDocumentName(xmlFile);
                ((JDBCAppendFragHandler) handler).setDocumentId(documentId);
                ((JDBCAppendFragHandler) handler).setFragmentId(elementId);
                parser.parse(xmlFile);
                return true;
            } catch (SAXException se) {
                getOutput().writeln(se.toString());
                se.printStackTrace();
            } catch (IOException ioe) {
                ioe.printStackTrace();
            }
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        } catch (IllegalAccessException ex) {
            ex.printStackTrace();
        } catch (InstantiationException ex) {
            ex.printStackTrace();
        }
        return false;
    }
    }

    /************************ DeleteFrag.java ***********************/
    package com.xweave.xmldb.xserve;

    import com.xweave.xmldb.fgrel.*;
    import com.xweave.xmldb.util.rdb.RDB;
    import com.xweave.xmldb.util.io.Output;
    import java.sql.*;
    import java.util.*;
    /**
     * Polecenie usunięcia fragmentu.
     */
    public class DeleteFrag extends Command {
    /**
     * Konstruktor DeleteFrag.
     */
    public DeleteFrag() {
        super();
    }
    public DeleteFrag(RDB rdb) {
        super(rdb);
    }
    public DeleteFrag(RDB rdb, Output out) {
        super(rdb, out);
    }
    protected void deleteCdata(String document, String elementId, String cdataId, String
    table) throws SQLException {
        StringBuffer buf = new StringBuffer();
        buf.append("delete from");
        buf.append(" " + table + " ");
        buf.append("where doc_id = " + document);
        buf.append("  and ele_id = " + elementId);
        buf.append("  and cdata_id = " + cdataId);
        rdbExecute(buf.toString());
    }
```

```
/**
 * Wypisanie z bazy danych elementu i jego "dzieci".
 */
protected void deleteElement(String document, String elementId) throws SQLException {
    StringBuffer buf;
    // usuwanie "dzieci"
    // sprawdzenie, czy są jakieś "dzieci"
    buf = new StringBuffer();
    buf.append("select indx, child_class, child_id from");
    buf.append(" " + CHILD_TABLE + " ");
    buf.append("where doc_id = " + document);
    buf.append("  and ele_id = " + elementId);
    ResultSet rset = getRdb().getData(buf.toString());
    boolean moreRows = rset.next();
    // pobranie wszystkich wierszy, następnie rekursja realizowana, aby
    // uniknąć pozostawienia otwartego kursora
    Vector vec = new Vector();
    int index;
    while(moreRows) {
        // wiersze są uporządkowane według indeksu
        index = Integer.parseInt(rset.getString(1));
        if (index > vec.size()) {
            vec.setSize(index);
        }
        vec.insertElementAt(new Child(rset.getString(2),rset.getString(3)), index);
        moreRows = rset.next();
    }
    // usuwanie kolejnych "dzieci"
    Enumeration e = vec.elements();
    Child child;
    String childClass;
    while (e.hasMoreElements()) {
        child = (Child) e.nextElement();
        if (child == null) {continue;}
        childClass = child.getClassName();
        if (childClass.equalsIgnoreCase("ELE")) {
            // rekursywne wywołanie deleteElement
            deleteElement(document, child.getId());
        } else if (childClass.equalsIgnoreCase("STR")) {
            //deleteCdata działa na STR lub TEXT przez JDBC
            deleteCdata(document, elementId, child.getId(), STR_TABLE);
        } else if (childClass.equalsIgnoreCase("TEXT")) {
            //deleteCdata działa na STR lub TEXT przez JDBC
            deleteCdata(document, elementId, child.getId(),TEXT_TABLE);
        } else {
            // konieczna obsługa innych klas dodanych do bazy danych
        }
    }
    // usunięcie z tabeli informacji o "dzieciach"
    buf = new StringBuffer();
    buf.append("delete from");
    buf.append(" " + CHILD_TABLE + " ");
    buf.append("where doc_id = " + document);
    buf.append("  and ele_id = " + elementId);
    rdbExecute(buf.toString());
    // usunięcie atrybutów
    buf = new StringBuffer();
    buf.append("delete from");
```

```
        buf.append(" " + ATTR_TABLE + " ");
        buf.append("where doc_id = " + document);
        buf.append("  and ele_id = " + elementId);
        rdbExecute(buf.toString());
        // usunięcie elementu
        buf = new StringBuffer();
        buf.append("delete from");
        buf.append(" " + ELE_TABLE + " ");
        buf.append("where doc_id = " + document);
        buf.append("  and ele_id = " + elementId);
        rdbExecute(buf.toString());
    }
    public boolean deleteFrag(String document, String element) {
        // Przedstawione rozwiązanie działa tylko w przypadku usuwania
        // kaskadowego, w przeciwnym wypadku konieczne jest jawne usuwanie
        // rekursywne.
        boolean status = true;
        try {
            // usunięcie informacji o "dziecku" z tabeli dla danego fragmentu
            StringBuffer buf = new StringBuffer();
            buf.append("delete from");
            buf.append(" " + CHILD_TABLE + " ");
            buf.append("where doc_id = " + document);
            buf.append("  and child_id = " + element);
            rdbExecute(buf.toString());
            // usunięcie elementu
            deleteElement(document, element);
        } catch (SQLException ex) {
            status = false;
            ex.printStackTrace();
        }
        return status;
    }
    }

    /************************ ReplaceFrag.java ***********************/
    package com.xweave.xmldb.xserve;

    import com.xweave.xmldb.fgrel.*;
    import com.xweave.xmldb.util.io.*;
    import com.xweave.xmldb.util.rdb.*;
    import java.sql.*;
    import org.xml.sax.*;
    import org.xml.sax.helpers.ParserFactory;
    import com.ibm.xml.parsers.DOMParser;
    import org.w3c.dom.Document;
    import java.io.IOException;
    /**
     * Polecenie podstawiające fragment w bazie danych.
     */
    public class ReplaceFrag extends Command {
    /**
     * Konstruktor ReplaceFrag.
     */
    public ReplaceFrag() {
        super();
    }
```

```
/**
 * Konstruktor ReplaceFrag.
 * @param rdb com.xweave.xmldb.util.rdb.RDB
 */
public ReplaceFrag(RDB rdb) {
    super(rdb);
}
/**
 * Konstruktor ReplaceFrag.
 * @param rdb com.xweave.xmldb.util.rdb.RDB
 * @param out com.xweave.xmldb.util.io.Output
 */
public ReplaceFrag(RDB rdb, Output out) {
    super(rdb, out);
}
/**
 * Parsowanie pliku XML za pomocą parsera SAX i zapisanie go jako fragmentu.
 * Uwaga: parser z <parserClass> MUSI być dostępny w ścieżce CLASSPATH.
 */
public boolean parseText(String xmlText, String documentId, String elementId) {
    String parserClass = com.xweave.xmldb.Default.getParserClass();
    try {
        Parser parser = ParserFactory.makeParser(parserClass);
        HandlerBase handler;
        if (getRdb() == null) {
            return false;
        } else {
            handler = new JDBCReplaceFragHandler(getRdb());
        }
        parser.setDocumentHandler(handler);
        parser.setErrorHandler(handler);
        try {
            ((LoadHandler)
handler).setDocumentName("Frag"+documentId+"."+"elementId");
            ((JDBCReplaceFragHandler) handler).setDocumentId(documentId);
            ((JDBCReplaceFragHandler) handler).setFragmentId(elementId);
            parser.parse(new InputSource(new java.io.StringReader(xmlText)));
            return true;
        } catch (SAXException se) {
            getOutput().writeln(se.toString());
            se.printStackTrace();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    } catch (ClassNotFoundException ex) {
        ex.printStackTrace();
    } catch (IllegalAccessException ex) {
        ex.printStackTrace();
    } catch (InstantiationException ex) {
        ex.printStackTrace();
    }
    return false;
}
/**
 * Parsowanie pliku XML za pomocą parsera SAX i zapisanie go jako fragmentu.
 * Uwaga: parser z <parserClass> MUSI być dostępny w ścieżce CLASSPATH.
 */
```

```
public boolean parseUrl(String xmlFile, String documentId, String elementId) {
    String parserClass = com.xweave.xmldb.Default.getParserClass();
    try {
        Parser parser = ParserFactory.makeParser(parserClass);
        HandlerBase handler;
        if (getRdb() == null) {
            return false;
        } else {
            handler = new JDBCReplaceFragHandler(getRdb());
        }
        parser.setDocumentHandler(handler);
        parser.setErrorHandler(handler);
        try {
            ((LoadHandler) handler).setDocumentName(xmlFile);
            ((JDBCReplaceFragHandler) handler).setDocumentId(documentId);
            ((JDBCReplaceFragHandler) handler).setFragmentId(elementId);
            parser.parse(xmlFile);
            return true;
        } catch (SAXException se) {
            getOutput().writeln(se.toString());
            se.printStackTrace();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    } catch (ClassNotFoundException ex) {
        ex.printStackTrace();
    } catch (IllegalAccessException ex) {
        ex.printStackTrace();
    } catch (InstantiationException ex) {
        ex.printStackTrace();
    }
    return false;
}
}

/****************** Procedury obsługi zdarzeń parsera SAX ***************/

/************************ JDBCFragLoadHandler.java ***********************/
package com.xweave.xmldb.xserve;

import com.xweave.xmldb.fgrel.*;
import org.xml.sax.*;
import com.xweave.xmldb.util.rdb.*;
import java.sql.SQLException;
/**
 * Procedura obsługi SAX, która używa JDBC do załadowania fragmentu do bazy danych
 */
public abstract class JDBCFragLoadHandler extends JDBCLoadHandler {
    protected String fragmentId = null;
public JDBCFragLoadHandler(com.xweave.xmldb.util.rdb.RDB rdb) {
    super(rdb);
}
public String getDocumentId() {
    return documentId;
}
public String getFragmentId() {
    return fragmentId;
}
```

```
/*
 * Pobranie następnego identyfikatora z wyników zapytania.
 */
public int getNextCtr(String query) {
    // pobranie następnego identyfikatora z Oracle
    try {
        java.sql.ResultSet resultSet = getRdb().getData(query);
        if (resultSet == null) {
            // błąd w zapytaniu
            // spowoduje dalsze kłopoty, więc generujemy wyjątek
            throw new Error("Niemożliwe pobranie z bazy danych identyfikatora:" +
query);
        }
        resultSet.next();
        String result = resultSet.getString(1);
        if (result == null) {
            // pusta tabela
            return 1;
        } else {
            return Integer.parseInt(result) + 1;
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
        // spowoduje dalsze kłopoty, więc generujemy wyjątek
        throw new Error("Niemożliwe pobranie z bazy danych nowego id:" + query);
    }
}
/**
 * Pobranie identyfikatora dokumentu ze zmiennej lub z bazy danych.
 */
public void init() {
    String documentId = getDocumentId();
    // pobranie następnego eleCtr
    StringBuffer buf = new StringBuffer();
    buf.append("select max(ele_id) from ");
    buf.append(ELE_TABLE);
    buf.append(" where doc_id = " + documentId);
    eleCtr = getNextCtr(buf.toString());
    // pobranie następnego attrCtr
    buf = new StringBuffer();
    buf.append("select max(attr_id) from ");
    buf.append(ATTR_TABLE);
    buf.append(" where doc_id = " + documentId);
    attrCtr = getNextCtr(buf.toString());
    // pobranie następnego cdataCtr
    buf = new StringBuffer();
    buf.append("select max(child_id) from ");
    buf.append(CHILD_TABLE);
    buf.append(" where doc_id = " + documentId);
    buf.append(" and child_class in ('STR', 'TEXT')");
    cdataCtr = getNextCtr(buf.toString());
}
public void setFragmentId(String newValue) {
    this.fragmentId = newValue;
}
/**
 * Inicjalizacja liczników na początku dokumentu XML.
 */
```

```
public void startDocument() {
    init();
}
}

/************************ JDBCAppendFragHandler.java **********************/
package com.xweave.xmldb.xserve;

import com.xweave.xmldb.fgrel.*;
import org.xml.sax.*;
import com.xweave.xmldb.util.rdb.*;
import java.sql.SQLException;
/**
 * Procedura obsługi SAX, korzystająca z JDBC do dodania w bazie fragmentu.
 */
public class JDBCAppendFragHandler extends JDBCFragLoadHandler {
public JDBCAppendFragHandler(RDB rdb) {
    super(rdb);
}
/**
 * Obsługa elementu, atrybutów i powiązania elementu z jego "rodzicem".
 */
public void startElement(String name, AttributeList attrList) {
    // DEBUG: System.out.println(name);
    // Utworzenie zapisu o nowym elemencie.
    Element parent = getCurrentElement();
    String parentId;
    String indexNum = null;
    if (parent == null) {
        parentId = "NULL";
    } else {
        parentId = parent.getId();
    }
    StringBuffer buf;
    String currentId;
    if (parent == null) {
        // element główny, pobranie następnego indeksu
        parentId = getFragmentId();
        // pobranie następnego indeksu
        buf = new StringBuffer();
        buf.append("select max(indx) from ");
        buf.append(CHILD_TABLE);
        buf.append(" where doc_id = " + getDocumentId());
        buf.append(" and ele_id = " + parentId);
        try {
        java.sql.ResultSet resultSet = getRdb().getData(buf.toString());
        if (resultSet == null) {
            // błąd w zapytaniu
            // spowoduje dalsze kłopoty, więc generujemy wyjątek
            throw new Error("Niemożliwe pobranie INDX z bazy dla: " +
                getDocumentId() + "." + parentId + "/n" + buf.toString());
        }
        resultSet.next();
        String ctr = resultSet.getString(1);
        if (ctr == null) {
            // pusta tabela
            indexNum = "1";
```

```
            } else {
                indexNum = String.valueOf(Integer.parseInt(ctr) + 1);
            }
            } catch (SQLException ex) {
                ex.printStackTrace();
                // spowoduje dalsze kłopoty, więc generujemy wyjątek
                throw new Error("Niemożliwe pobranie INDX z bazy dla: " +
                    getDocumentId() + "." + parentId + "/n" + buf.toString());
            }
        }
        // dalszy ciąg tworzenia elementu
        setCurrentElement(Integer.toString(nextEleCtrValue()));
        currentId = getCurrentElement().getId();
        buf = new StringBuffer();
        buf.append("insert into");
        buf.append(" " + ELE_TABLE + " ");
        buf.append("(doc_id, ele_id, parent_id, tag)");
        buf.append(" values ");
        buf.append("(" + getDocumentId() + ", " +
                         currentId + ", " +
                         parentId + ", " +
                         "'" + name + "')");
        rdbExecute(buf.toString());
        // tworzenie zapisu "dziecka"
        buf = new StringBuffer();
        buf.append("insert into");
        buf.append(" " + CHILD_TABLE + " ");
        buf.append("(doc_id, ele_id, indx, child_class, child_id)");
        buf.append(" values ");
        buf.append("(" + getDocumentId() + ", " + parentId + ", ");
        if (parent == null) {
            // element główny
            buf.append(indexNum);
        } else {
            // podelement
              buf.append(parent.incrIndexCtr());
        }
          buf.append(", " + "'ELE', " + currentId + ")");
        if (! rdbExecute(buf.toString()) ) {
            throw new Error("Niemożliwa aktualizacja w bazie informacji o \"dziecku\": " +
                getDocumentId() + "." + parentId +
                ", element "+ currentId + "/n" + buf.toString());
        }
        // Tworzenie zapisów dla poszczególnych atrybutów
        for (int i = 0; i < attrList.getLength(); i++) {
            buf = new StringBuffer();
            buf.append("insert into");
            buf.append(" " + ATTR_TABLE + " ");
            buf.append("(doc_id, attr_id, ele_id, name, value)");
            buf.append(" values ");
            buf.append("(" + getDocumentId() + ", " +
                             nextAttrCtrValue() + ", " +
                             currentId + ", " +
                             "'" + attrList.getName(i) + "', " +
                              "'" + attrList.getValue(i) + "')");
            rdbExecute(buf.toString());
        }
    }
}
```

```
/************************ JDBCReplaceFragHandler.java ***********************/
package com.xweave.xmldb.xserve;
import com.xweave.xmldb.fgrel.*;
import org.xml.sax.*;
import com.xweave.xmldb.util.rdb.*;
import java.sql.SQLException;
/**
 * Procedura obsługi SAX, która używa JDBC do zastąpienia w bazie fragmentu.
 */
public class JDBCReplaceFragHandler extends JDBCFragLoadHandler {
/**
 * Konstruktor JDBCReplaceFragHandler.@param rdb com.xweave.xmldb.util.rdb.RDB
 */
public JDBCReplaceFragHandler(RDB rdb) {
    super(rdb);
}
/**
 * Obsługa elementu, atrybutów i powiązania elementu z jego "rodzicem".
 */
public void startElement(String name, AttributeList attrList) {
    // DEBUG: System.out.println(name);
    // Utworzenie zapisu o nowym elemencie
    Element parent = getCurrentElement();
    String parentId;
    String indexNum = null;
    if (parent == null) {
        parentId = "NULL";
    } else {
        parentId = parent.getId();
    }
    StringBuffer buf;
    String currentId;
    if (parent == null) {
        // po raz pierwszy
        setCurrentElement(getFragmentId());
        currentId = getCurrentElement().getId();
        // pobranie starego identyfikatora "rodzica"
        buf = new StringBuffer();
        buf.append("select parent_id from ");
        buf.append(ELE_TABLE);
        buf.append(" where doc_id = " + getDocumentId());
        buf.append(" and ele_id = " + currentId);
        parentId = getRdb().getDataItem(buf.toString());
        if (parentId == null) {
            // spowoduje dalsze kłopoty, więc generujemy wyjątek
            throw new Error("Niemożliwe pobranie id "rodzica" z bazy: " +
                getDocumentId() + "." + currentId + "/n" + buf.toString());
        }
        // pobranie starego indeksu
        buf = new StringBuffer();
        buf.append("select indx from ");
        buf.append(CHILD_TABLE);
        buf.append(" where doc_id = " + getDocumentId());
        buf.append(" and ele_id = " + parentId);
        buf.append(" and child_id = " + currentId);
        indexNum = getRdb().getDataItem(buf.toString());
        if (indexNum == null) {
            // spowoduje dalsze kłopoty, więc generujemy wyjątek
            throw new Error("Niemożliwe pobranie INDX z bazy danych: " +
```

```
                getDocumentId() + "." + currentId + "/n" + buf.toString());
        }
        // usunięcie starych informacji o elemencie i jego "dzieciach"
        //utworzenie nowego polecenia DeleteFrag i użycie go
        DeleteFrag delFragCmd = new DeleteFrag(getRdb());
        if (! delFragCmd.deleteFrag(getDocumentId(), currentId)) {
            // spowoduje dalsze kłopoty, więc generujemy wyjątek
            throw new Error("Niemożliwe usunięcie z bazy danych fragmentu: " +
                getDocumentId() + "." + currentId + "/n" + buf.toString());
        }
    } else {
        // nie element główny, zapis nowego elementu
        setCurrentElement(Integer.toString(nextEleCtrValue()));
        currentId = getCurrentElement().getId();
    }
    // dalszy ciąg tworzenia elementu
    buf = new StringBuffer();
    buf.append("insert into");
    buf.append(" " + ELE_TABLE + " ");
    buf.append("(doc_id, ele_id, parent_id, tag)");
    buf.append(" values ");
    buf.append("(" + getDocumentId() + ", " +
                     currentId + ", " +
                     parentId + ", " +
                     "'" + name + "')");
    rdbExecute(buf.toString());
    // utworzenie informacji o "dzieciach"
    buf = new StringBuffer();
    buf.append("insert into");
    buf.append(" " + CHILD_TABLE + " ");
    buf.append("(doc_id, ele_id, indx, child_class, child_id)");
    buf.append(" values ");
    buf.append("(" + getDocumentId() + ", " + parentId + ", ");
    if (parent == null) {
        // element główny
        buf.append(indexNum);
    } else {
        // podelement
          buf.append(parent.incrIndexCtr());
    }
      buf.append(", " + "'ELE', " + currentId + ")");
    rdbExecute(buf.toString());
    // Tworzenie zapisów dla poszczególnych atrybutów
    for (int i = 0; i < attrList.getLength(); i++) {
        buf = new StringBuffer();
        buf.append("insert into");
        buf.append(" " + ATTR_TABLE + " ");
        buf.append("(doc_id, attr_id, ele_id, name, value)");
        buf.append(" values ");
        buf.append("(" + getDocumentId() + ", " +
                         nextAttrCtrValue() + ", " +
                         currentId + ", " +
                         "'" + attrList.getName(i) + "', " +
                          "'" + attrList.getValue(i) + "')");
        rdbExecute(buf.toString());
    }
}
}
```

Klasa główna *xServe*, `Demo`, jest subklasą klasy `Demo` z drobnoziarnistego systemu przechowywania danych opisanego w punkcie 4.2.4. Klasa `Demo` zawiera pola dla każdej klasy implementującej polecenia serwera danych i wykonuje polecenia, korzystając z metody `dispatchArg`. Każde polecenie jest subklasą. Każde polecenie jest podklasą klasy abstrakcyjnej `Command`, zawierającą obiekt `RDB`, łączący się z bazą danych oraz obiekt `Output`, wypisujący sformatowane dane do odpowiedniego strumienia. `RDB` i `Output` — zostały opisane w dodatku A. Polecenia *xServe* pozwalają pobierać, dodawać i podmieniać zarówno fragmenty dokumentu, jak też pobierać, zapisywać i usuwać dokumenty.

Do dodawania i podmiany fragmentów używa się procedury obsługi parsera SAX. Parser ten został opisany w dodatku B. Klasa abstrakcyjna `JDBCFragLoadHandler` pobiera największy indeks „dziecka", elementu zagnieżdżonego i atrybutów, dzięki czemu podczas wstawiania danych będą tworzone odpowiednie, niepowtarzalne identyfikatory. `JDBCReplaceFragHandler` korzysta z polecenia `DeleteFrag`, by usunąć stary element i wstawić w jego miejsce nowy fragment XML. `JDBCAppendFrag` wstawia fragment XML po ostatnim elemencie zagnieżdżonym. Oba polecenia wstawiają dane podobnie jak polecenie ładowania dokumentu `JDBCLoadHandler` w drobnoziarnistym, relacyjnym systemie przechowywania danych.

5.5. Hybrydowy serwer łączący technologię relacyjną i XML

Opisane w poprzednich punktach dwa serwery danych: relacyjny i XML sprawdzają się w różnych sytuacjach, jednak równie przydatny jest system hybrydowy, obsługujący zapytania kierowane do obydu serwerów. Hybrydowy serwer danych pozwala sięgać zarówno do danych w postaci relacyjnej, jak i do danych XML. Dane mogą być przechowywane w układzie najbardziej dla nich odpowiednim. Taki hybrydowy serwer pozwala, w pewnym stopniu, łączyć dane z różnych modeli danych — tworzone są połączenia między strukturami danych stanowiącymi podstawę obydwu sposobów przechowywania danych. Na przykład dokument XML może udostępniać element odpowiadający zapytaniu relacyjnemu. Kiedy pobierany jest dokument XML, może zostać wykonane zapytanie relacyjne i jego wyniki mogą być włączone do strumienia danych. Analogicznie wartości z kolumn relacyjnej bazy danych mogą wskazywać dokumenty lub ich fragmenty w bazie danych XML, zaś dane XML mogą być łączone z danymi relacyjnymi pobranymi i sformatowanymi jako XML. Dzięki temu można połączyć wydajność modelu relacyjnego z elastycznością modelu XML.

Serwer hybrydowy można stworzyć w architekturze wielowarstwowej z dwóch serwerów: klasycznego serwera relacyjnego i serwera danych XML. Serwer hybrydowy przekazuje żądania adresu URL do obu serwerów danych i łączy otrzymane dokumenty (możliwe jest też wykorzystanie innych protokołów niż HTTP).

Zapewne byłoby łatwiej stworzyć nowy serwer danych w architekturze trzywarstwowej, aby sięgać do serwera relacyjnego i serwera XML. Upraszcza to strukturę systemu, ale wiąże się z tym, że serwer XML i serwer relacyjny powinny być dostępne w ramach tej samej aplikacji. Jako że aplikacje *xServe* i *rServe* mogą współistnieć, system hybrydowy — *xrServe* — może przekazywać polecenia bezpośrednio do odpowiednich klas. Architektura wielowarstwowa byłaby podobna do trzywarstwowej, z tym że istniałyby dodatkowe procesy formatowania, wysyłania i pobierania danych z dwóch odrębnych serwerów.

W tej samej strukturze można umieścić dodatkowe funkcje. Dobrze byłoby na przykład umożliwić w dokumencie XML lub tabeli relacyjnej dostęp do zewnętrznych adresów URL. Inny element dostępowy może oferować informacje pozwalające sięgnąć do zasobu z plików lub URL (jak pliki XML).

5.5.1. Implementacja

Serwer hybrydowy jest rozwinięciem serwera danych XML, który przedstawiliśmy w poprzednim punkcie. Operacje serwera danych XML są dziedziczone, aby uniknąć powtórzeń. Operacje serwera relacyjnego są przekazywane odrębnym instancjom. W trakcie parsowania adresu URL tworzony jest obiekt `AccessSpec` na dane zapytania. Następnie wywoływana jest metoda `writeAccess`, aby przekazać polecenia odpowiedniej instancji: obiektowi `this` *xServe* lub obiektowi z pola `formatRel` *rServe*. Jako że jedyną obsługiwaną operacją jest wypisanie fragmentu XML z dowolnego źródła, kod jest dość prosty.

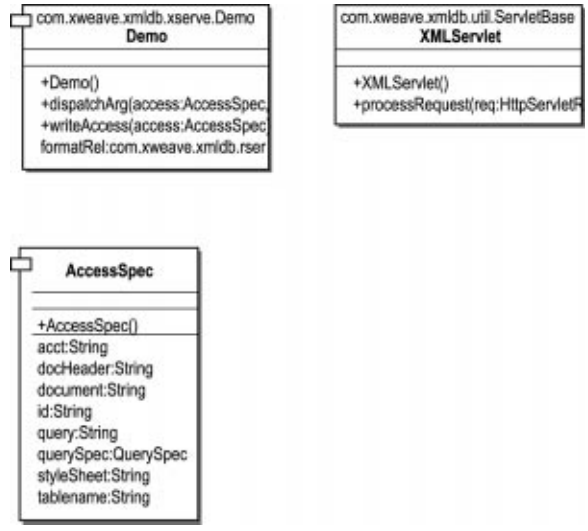
Kod Java dla serwera hybrydowego *xrServe* pokazuje wydruk 5.16, a diagram klas UML zawiera rysunek 5.9.

Wydruk 5.16. *Kod Java hybrydowego serwera danych relacje-XML*

```
/************************ Demo.java ***********************/
package com.xweave.xmldb.xrserve;

import com.xweave.xmldb.util.io.*;
import com.xweave.xmldb.util.rdb.*;
import com.xweave.xmldb.xserve.*;
import com.xweave.xmldb.rserve.*;

/**
 * Demonstracja hybrydowego serwera danych, relacyjno-XML
 */
public class Demo extends com.xweave.xmldb.xserve.Demo {
    protected com.xweave.xmldb.rserve.Demo formatRel = null;
public Demo() {
    super();
}
public void dispatchArg(AccessSpec access, String command, String value) {
    if (command.equalsIgnoreCase("TABLENAME")) {
        access.setTablename(value);
        return;
    }
    if (command.equalsIgnoreCase("ID")) {
        access.setId(value);
        return;
    }
    if (command.equalsIgnoreCase("DOCUMENT")) {
      access.setDocument(value);
        return;
    }
    if (command.equalsIgnoreCase("ACCT")) {
        access.setAcct(value);
        return;
    }
```



Rysunek 5.9. *Diagram klas hybrydowego serwera danych łączącego technologię relacyjną i XML*

```
    if (command.equalsIgnoreCase("STYLESHEET")) {
        access.setStyleSheet(value);
        return;
    }
    if (command.equalsIgnoreCase("DOCHEADER")) {
        access.setDocHeader(value);
        return;
    }
    if (command.equalsIgnoreCase("IGNORE") || command.equalsIgnoreCase("SUBMIT")) {
        // pominięcie argumentów
        return;
    }
        // Niezinterpretowane argumenty należą do zapytania do relacyjnej bazy
    access.getQuerySpec().addConstraint(command, value);
}
public com.xweave.xmldb.rserve.Demo getFormatRel() {
    if (formatRel == null) {
        formatRel = new com.xweave.xmldb.rserve.Demo();
    }
    return formatRel;
}
public void setFormatRel(com.xweave.xmldb.rserve.Demo newValue) {
    this.formatRel = newValue;
}
public boolean writeAccess(AccessSpec access) {
    if (access.getDocument() != null) {
        // zapis dokumentu
        com.xweave.xmldb.fgrel.Format formatxml = getFormat();
        formatxml.setOutput(getOutput());
        if (access.getDocHeader() != null) {
            getOutput().writeln("<?xml version=\"1.0\"?>");
        }
        if (access.getStyleSheet() != null) {
            getOutput().writeln("<?xml:stylesheet type=\"text/xsl\" href=\"" +
access.getStyleSheet() + "\"?>");
        }
```

```
        if (access.getId() != null) {
            // zapis fragmentu
            return formatxml.writeFrag(access.getDocument(), access.getId());
        } else {
            // zapis całego dokumentu
            return formatxml.writeDoc(access.getDocument());
        }
    } else
        if (access.getTablename() != null) {
            // zapis tabeli
            com.xweave.xmldb.rserve.Demo formatrel = getFormatRel();
            formatrel.setOutputSource(getOutput());
            com.xweave.xmldb.rserve.QuerySpec query = access.getQuerySpec();
            com.xweave.xmldb.rserve.AccessSpec relaccess = new
com.xweave.xmldb.rserve.AccessSpec();
            relaccess.setQuerySpec(query);
            relaccess.setAcct(access.getAcct());
            if (access.getStyleSheet() != null) {
                relaccess.setStyleSheet(access.getStyleSheet());
            }
            query.setTableName(access.getTablename());
            // zapis odpowiedniego fragmentu
                        // ZROBIĆ: podać jako nagłówek dokumentu
            if (access.getId() != null) {
                // zapis pojedynczego rekordu
                query.setId(access.getId());
            } else
                if (access.getQuery() != null) {
                    // zapis zapytania
                    query.setConstraintString(access.getQuery());
                    // w przeciwnym wypadku zapis całej tabeli
                    // nie są ustawiane żadne parametry
                }
                // zapis dokumentu, czyli elementu głównego
            if (access.getDocHeader() == null) {
                formatrel.writeTable(relaccess, 0);
            } else {
                formatrel.writeDoc(relaccess);
            }
        } else {
            // brak informacji
            return false;
        }
    return true;
}
}

/************************ AccessSpec.java **********************/
package com.xweave.xmldb.xrserve;

import java.util.*;
import com.xweave.xmldb.rserve.QuerySpec;
/**
 * Specyfikacja dostępu do serwera.
 */
public class AccessSpec {
    public String document = null;
    public String tablename = null;
```

```
    public String id = null;
    public String query = null;
    public String acct = com.xweave.xmldb.Default.getXmldbAcct();
    public String styleSheet = null;
    public String docHeader = null;
    public QuerySpec querySpec = null;
public AccessSpec() {
    super();
}
public String getAcct() {
    return acct;
}
public String getDocHeader() {
    return docHeader;
}
public String getDocument() {
    return document;
}
public String getId() {
    return id;
}
public String getQuery() {
    return query;
}
public QuerySpec getQuerySpec() {
    if (querySpec == null)
        querySpec = new QuerySpec();
    return querySpec;
}
public String getStyleSheet() {
    return styleSheet;
}
public String getTablename() {
    return tablename;
}
public void setAcct(String newValue) {
    this.acct = newValue;
}
public void setDocHeader(String newValue) {
    this.docHeader = newValue;
}
public void setDocument(String newValue) {
    this.document = newValue;
}
public void setId(String newValue) {
    this.id = newValue;
}
public void setQuery(String newValue) {
    this.query = newValue;
}
public void setQuerySpec(QuerySpec newValue) {
    this.querySpec = newValue;
}
public void setStyleSheet(String newValue) {
    this.styleSheet = newValue;
}
public void setTablename(String newValue) {
```

```
    this.tablename = newValue;
}
}

/************************ XMLServlet.java ***********************/
package com.xweave.xmldb.xrserve;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.xweave.xmldb.util.io.ServletOutput;

/**
 * Interfejs serwleta do bazy danych.
 */
public class XMLServlet extends com.xweave.xmldb.util.ServletBase {
public XMLServlet() {
    super();
}
public void processRequest(HttpServletRequest req, HttpServletResponse res) throws
ServletException, java.io.IOException {
    ServletOutputStream out = startOutput(res);
    Enumeration enum = req.getParameterNames();
    String pname, pvalue = null;
    Demo db = new Demo();
    db.setOutput(new ServletOutput(out));
    db.setRdbAccessString("mgraves/mgraves@127.0.0.1:1521:ORCL");
    AccessSpec access = new AccessSpec();
    while (enum.hasMoreElements()) {
        pname = (String) enum.nextElement();
        pvalue = req.getParameter(pname);
        log("Dispatching " + pname + " " + pvalue);
        db.dispatchArg(access, pname, pvalue);
    }
    if (access.getAcct() != null) {
        db.setRdbAccessString(access.getAcct());
    }
    if (db.getRdbAccessString() == null) {
        log("Błąd w processRequest: nie podano łańcucha dostępu dla:\n  " +
          req.getPathTranslated());
    }
    db.writeAccess(access);
    endOutput(res);
}
}
```