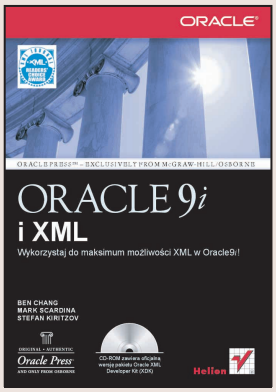
Oracle9*i* i XML

Autorzy: Ben Chang, Mark Scardina, Stefan Kiritzov
Tłumaczenie: Bartłomiej Jabłoński, Cezary Welsyng
ISBN: 83-7361-064-2
Tytuł oryginału: Oracle9i XML Handbook
Format: B5, stron: 432



Wykorzystaj do maksimum możliwości XML w Oracle9i!

Projektuj i uruchamiaj w Oracle9i niezależne od platformy aplikacje oparte na transakcjach wykorzystujące XML – standard opisu danych, umożliwiający integrowanie elektronicznych aplikacji biznesowych i wymianę danych typu business-to-business. Z niniejszej książki, napisanej przez członków zespołu Oracle XML Development Team dowiesz się, jak za pomocą pakietu XML Developer Kit (XDK) można tworzyć, przekształcać i przeglądać dokumenty XML oraz używać ich zgodnie z własnymi potrzebami. Zamieszczone w książce przykłady rzeczywistych zastosowań tego standardu ilustrują sposób, w jaki klienci Oracle mogą efektywnie wykorzystywać wbudowane funkcje obsługi XML.

Z książki dowiesz się jak:

- korzystać z infrastruktury Oracle9i XML oraz pakietu XML Developer Kit (XDK), używać parserów, procesorów, generatorów, przeglądarek i innych narzędzi pakietu XDK,
- tworzyć rozbudowane aplikacje Oracle9i oparte na komponentach XML dla języka Java,
- wykorzystywać wbudowane w Oracle9i nowe funkcje SQL i PL/SQL dla XML oraz mechanizmy przesyłania komunikatów oparte na XML,
- uwzględniać różnice pomiędzy serwerem OAS (Oracle Application Server) i serwerem Oracle9iAS (Oracle9i Application Server),
- projektować i wdrażać aplikacje transakcyjne dla serwerów OAS i Oracle9i,
- umożliwić dostęp do dokumentów w skali całego przedsiębiorstwa poprzez składowanie ich w systemie iFS (Internet File System),
- zarządzać za pomocą narzędzia Oracle Text różnymi rodzajami danych – tekstem, grafiką, dźwiękiem i obrazem wideo – traktując je jak zwykłe typy danych,
- tworzyć e-biznesowe aplikacje internetowe za pomocą komponentu Oracle E-Business XML Services.

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

Spis treści

Rozdział 6.

Wyszukiwanie dokumentów XML za pomocą narzędzia Oracle Text

Bazy danych przechodzą rewolucję. Dane przechowywane w tradycyjnych relacyjnych bazach mają ściśle określoną strukturę a zapytania ich dotyczące muszą być zawsze zgodne ze schematem bazy danych. Takie bazy danych są bardzo przydatne do składowania informacji strukturalnych ale niezbyt nadają się do obsługiwania danych w innej postaci. Klasyczny język SQL wystarcza do wysyłania niestrukturalnych zapytań dotyczących wartości przechowywanych w niewielkich kolumnach ale taki sposób wyszukiwania nie jest efektywny w przypadku długich kolumn, zawierających dane tekstowe. Wyszukiwanie danych składowanych w niestrukturalnych dużych obiektach (ang. *LOB — Large Objects*) tradycyjnie było nieefektywne i przebiegało bardzo wolno. Dlatego właśnie obecne wysiłki wielu producentów oferujących rozwiązania w zakresie baz danych skupiają się na zapewnieniu użytkownikom wydajnego dostępu do informacji złożonego typu. Dotychczasowe, rygorystyczne ograniczenia dostępu do danych zostały złagodzone, zatem wymagania dotyczące struktury danych przechowywanych w bazach danych również stały się nieco mniejsze.

W dzisiejszych nowoczesnych, internetowych bazach danych są przechowywane dane różnego typu. Bardzo często bazy zawierają mnóstwo danych w formie klipów wideo, plików graficznych i dokumentów tekstowych a rozmiar tych danych jest o rzędy wielkości większy od rozmiaru danych strukturalnych. Wraz z upowszechnieniem się Internetu pojawiła się potrzeba uzyskiwania natychmiastowego dostępu do danych, dzięki czemu dokumenty HTML i XML stały się powszechnie stosowanym mechanizmem wymiany danych tekstowych w Internecie. Z tego względu nowoczesne bazy danych, stanowiące zaplecze witryn WWW, muszą być w stanie łatwo wyszukiwać i odczytywać tego rodzaju dokumenty i nadal cechować się skalowalnością i wydajnością.

Oracle Text jest komponentem systemu Oracle9*i* i pozwala użytkownikom na zarządzanie różnymi typami danych za pomocą standardowych mechanizmów dostępu opartych na SQL. W przypadku komponentu Oracle Text, który zastąpił narzędzie *interMedia Text*

(*ConText*), dokumenty tekstowe, obrazy, pliki audio i klipy wideo są traktowane jako dane określonych typów. Komponent ten umożliwia łatwe zarządzanie danymi i zapewnia obsługę Internetu, udostępniając narzędzia do tworzenia stron WWW oraz serwery WWW. Mówiąc bardziej dokładnie, Oracle Text rozszerza funkcjonalność bazy danych o obsługę informacji tekstowych, poprzez zapewnienie płynnego wyszukiwania danych w ramach standardowych zapytań SQL, przetwarzanych całkowicie przez jądro bazy danych i niewymagających użycia dodatkowych komponentów.

W niniejszym rozdziale znajduje się opis mechanizmów indeksowania tekstowego komponentu Oracle Text, ze szczególnym uwzględnieniem funkcji obsługujących standard XML, takich jak `XML_SECTION_GROUP`, `AUTO_SECTION_GROUP`, nowej funkcji `PATH_SECTION_GROUP` oraz operatorów `HASPATH()` i `INPATH()` wprowadzonych w wersji Oracle9*i*. Omówione zostaną również sekcje strefowe, sekcje pól oraz sekcje atrybutów. Przykłady różnych wersji przykładowej aplikacji Bookstore ilustrują praktyczne zastosowanie narzędzia Oracle Text.

Oracle Text — wyszukiwarka tekstowa następnej generacji

We współczesnym relacyjnym systemie baz danych są uwzględnione wszystkie aspekty związane z indeksowaniem prostych typów danych, takich jak liczby całkowite (ang. *integer*) i niewielkie ciągi tekstowe (ang. *strings*). Złożone typy danych, takie jak tekst strukturalny, dane przestrzenne, pliki graficzne i dźwiękowe oraz klipy wideo wymagają wyszukiwania informacji opartego na analizie zawartości. Każdy ze złożonych typów danych jest charakteryzowany przez format specyficzny dla aplikacji, określone wymagania dotyczące indeksowania oraz predykaty selekcji. Na przykład dokumenty ze znacznikami, takie jak HTML i XML, mogą wymagać wyszukiwania opartego na znacznikach lub wartościach atrybutów. To wszystko powoduje potrzebę zapewnienia mechanizmów indeksowania złożonych typów danych i wypracowanie wyspecjalizowanych technik indeksowania.

Rozwiązanie firmy Oracle polegało na wprowadzeniu koncepcji typu indeksu (ang. *indextype*). Odpowiednie zaplecze oferowane przez Oracle pozwala na definiowanie typów indeksów w zależności od potrzeb użytkowników. Typ indeksu jest odpowiada posortowanym lub odwzorowanym bitowo typom indeksowym wbudowanym do serwera Oracle, z jedną istotną różnicą. Typy indeksów nie są wbudowane lecz implementowane przez programistę aplikacji. Model ten jest oparty na koncepcji indeksowania zespołowego (ang. *cooperative indexing*). Zgodnie z tym modelem komponent interMedia i system Oracle9*i* współpracują ze sobą w celu zbudowania i utrzymywania indeksów typów danych, takich jak dane tekstowe, przestrzenne czy *OLAP* (ang. *online analytical processing*). Komponent interMedia odpowiada za zdefiniowanie struktury indeksu, utrzymywanie zawartości podczas wykonywania operacji ładowania i uaktualniania oraz za przeszukiwanie indeksu podczas obsługi żądania. Sama struktura indeksu może być zapamiętana w bazie danych Oracle w postaci *tabeli indeksowej* (ang. *IOT — Index-Organized Table*) lub na zewnątrz, jako plik systemu operacyjnego. Programista określa szczegóły implementacji typu indeksu, natomiast jądro bazy danych implementuje wbudowane indeksy (posortowane lub odwzorowane bitowo).

Indeksy Oracle Text mogą być tworzone niemal w każdej kolumnie bazy danych. Gdy dokument jest wczytywany do jednej lub wielu takich kolumn, zmiany są wykrywane automatycznie i dokument ten zostaje oznaczony jako gotowy do indeksowania. Podczas indeksowania, które może być wykonane natychmiast lub według harmonogramu określonego przez użytkownika, wszystkie słowa i części dokumentu są zapisywane w tabelach indeksowych, wykorzystywanych później podczas operacji wyszukiwania. Zapytania tekstowe są realizowane na podstawie indeksu. Po przetworzeniu takiego zapytania jest zwracana lista trafień opisujących dopasowane dokumenty. Indeksy są utrzymywane w bazie danych, a wszystkie żądania są przetwarzane w bazie danych za pomocą pojedynczego interfejsu aplikacji.

Optymalizator kosztów Oracle (ang. *CBO — Cost Based Optimizer*) szacuje i wybiera najszybszy plan wykonania, przy określonych właściwościach uruchomieniowych wyszukiwanych danych. Optymalizator CBO może potokowo przetwarzać identyfikatory wierszy (ang. *rowids*) zgodne z predykatem lub sprawdzać na początku, czy z takim predykatem zgodny jest wiersz o podanym identyfikatorze. W tym drugim przypadku wykorzystuje się specjalny algorytm, wprowadzony w systemie Oracle8*i*, za pomocą którego można uzyskać dostęp do „odwrotnego" indeksu tekstowego dla podanego wiersza.

Baza Oracle8*i* była pierwszą relacyjną bazą danych, w której obsługę zapytań tekstowych zawarto w jej jądrze. Ten w pełni zintegrowany system do przetwarzania całego żądania wykorzystuje jądro RDBMS i to z tego powodu narzędzie Oracle Text zasłużyło na miano „wyszukiwarki tekstowej następnej generacji" (ang. *Next-Generation Text Engine*).

Model indeksowania w Oracle Text

Przetwarzanie tekstu za pomocą narzędzia Oracle Text może być rozumiane jako przetwarzanie potokowe pomiędzy różnymi modułami, którego przebieg na każdym etapie można modyfikować za pomocą dostępnych opcji. W wyniku przetwarzania potokowego otrzymuje się *indeks odwrócony* (ang. *inverted index*), będący listą słów pobranych z dokumentu. Przy każdym słowie znajduje się lista dokumentów, w których się ono pojawia. Indeks ten jest nazywa się „odwróconym", ponieważ zazwyczaj jest to lista dokumentów, z których każdy zawiera listę słów. Każdy etap przetwarzania potokowego można dopasować do własnych potrzeb za pomocą *systemu preferencji* (ang. *Preference System*) lub *grup sekcji* (ang. *Section Groups*). Są one tworzone przy użyciu specjalnych interfejsów aplikacji:

```
ctx_ddl.create_preference(...);
ctx_ddl.create_section_group(...);
```

a następnie wstawiane do potoku indeksującego za pomocą klauzuli `parameters`:

```
create index xml_index on Bookstore (xml_text)
          indextype is ctxsys.index
          parameters ('...');
```

W kolejnych punktach znajduje się szczegółowe omówienie poszczególnych modułów potoku, przepływu informacji w potoku a także opcji i poleceń, które można zastosować na każdym z etapów procesu indeksowania. Na rysunku 6.1 znajduje się ogólny schemat przepływu informacji w potoku.

Rysunek 6.1. *Przepływ informacji w potoku indeksującym*

Skład danych

Na początku potoku indeksującego znajduje się *skład danych* (ang. *datastore*), w którym po przeszukaniu wierszy tabel są umieszczane dane odczytane z kolumny. Wiersze nie są przeszukiwane w jakiejś określonej kolejności. Tekst indeksowany przez narzędzie Oracle Text może zostać zapamiętany w bazie danych lub w zewnętrznym systemie plików (wówczas będzie on zarządzany przez tę bazę). Skład danych z adresami URL pozwala na zarządzanie w ramach bazy danych dokumentami odległymi, znajdującymi się na innych serwerach, do których dostęp jest uzyskiwany za pośrednictwem protokołu HTTP lub FTP. Są to zazwyczaj dane kolumn ale w niektórych składach danych są one tylko wskaźnikiem do danych dokumentu. Przykładowo, skład `URL_DATASTORE` korzysta z danych kolumn jak z adresów URL i za pomocą operacji `GET` odczytuje dane i zwraca je do klienta, który wysłał żądanie. Składy danych są tworzone i włączane do potoku za pomocą następującego skryptu:

```
ctx_ddl.create_preference ('xml_urls', 'URL_DATASTORE');
ctx_ddl.set_attribute('xml_urls', 'HTTP_PROXY', 'www.proxy.us.oracle.com');
...
create_index xml_index on bookstore (xml_text)
   ...
   parameters ('datastore xml_urls ...');
```

Filtr

Filtr pobiera dane dokumentu ze składu danych i przekształca je na postać tekstową (XML, HTML lub zwykły tekst). Jest to potrzebne w przypadku dokumentów binarnych, takich jak pliki Microsoft Word czy Adobe Acrobat. Oracle Text posiada filtry dla ponad 100 formatów plików, które można przemieszać w pojedynczej kolumnie. Filtry w poniższym kodzie generują wyniki wyłącznie w formacie HTML.

Do potoku można ponadto dołączyć własne filtry lub filtry oferowane przez innych producentów. *Filtr własny* (ang. *custom-built filter*) jest po prostu programem wykonywalnym lub skryptem, który pobiera dwa argumenty: nazwę pliku zawierającego sformatowany tekst źródłowy oraz nazwę pliku, do którego ma zostać zapisany wynik filtrowania. Podobnie jak składy danych, filtry są tworzone i włączane do potoku za pomocą systemu preferencji, tak jak pokazano poniżej:

```
ctx_ddl.create_preference ('xml_filter', 'USER_FILTER');
ctx_ddl.set_attribute('xml_filter', 'EXECUTABLE', 'xmlfilter.exe');
...
create_index xml_index on bookstore (xml_text)
   ...
   parameters ('datastore xml_urls  filter xml_filter');
```

Moduł podziału na sekcje

Moduł podziału na sekcje (ang. *sectioner*) pobiera wynik filtrowania i przekształca go na zwykły tekst. Proces konwersji jest sterowany przez klasę reprezentującą grupę sekcji, która przekazuje do modułu informację o sposobie parsowania danych wejściowych. Dane te mogłyby zostać potraktowane jako kod XML, kod HTML lub jako zwykły tekst. Grupa sekcji zwykle zawiera jedną lub więcej sekcji. Definiując proste odwzorowanie, programiści aplikacji mogą każdej sekcji nadać wybraną nazwę. Moduł rozpoznaje poszczególne sekcje w każdej jednostce tekstu i automatycznie indeksuje go według tych sekcji. Mogą to być sekcje XML, HTML lub też sekcje zawarte pomiędzy znacznikami *<ZnacznikOtwierający> ... <ZnacznikZamykający>* w ramach grupy AUTO_SECTION_GROUP. Korzystając z operatora CONTAINS można za pomocą poniższej instrukcji przeprowadzić wyszukiwanie indeksowanych sekcji:

```
SELECT pk FROM bookstore
   WHERE CONTAINS (xml_text, '(The Spy) WITHIN title') > 0;
```

Po wysłaniu tego zapytania zostaje wyszukany dokument zawierający znaczniki XML, podobne do poniższych:

```
<book>
     <id> 4 </id>
     <author> John LeCarre </author>
     <title> The Spy Who Came in from the Cold </title>
</book>
```

Jeżeli grupa sekcji jest pusta, moduł przekształca tekst sformatowany na zwykły ale nie indeksuje granic sekcji. Wynik działania modułu zawiera granice sekcji oraz treść dokumentu, reprezentowane jako zwykły tekst. Treść ta jest następnie przekazywana do modułu podziału na leksemy. Konwersja na zwykły tekst wiąże się również z wykrywaniem znaczników najważniejszych sekcji, usuwaniem „niewidocznych" informacji oraz ponownym formatowaniem tekstu. Granice sekcji są następnie przekazywane do wyszukiwarki. Kolejnym krokiem jest tworzenie grup sekcji, które są wstawiane do potoku za pomocą odpowiednich interfejsów aplikacji:

```
ctx_ddl.create_section_group ('group_name', 'group_type');
...
create index
   ...
   parameters (section group xml_group ...');
```

Moduł podziału na sekcje uwzględnia typ grupy wtedy, gdy przekształca dane wejściowe pobrane z filtra na zwykły tekst i gdy oblicza indeksy dla granic sekcji. W przypadku dokumentów XML najważniejsze grupy sekcji są następujące:

- BASIC_SECTION_GROUP — rozpoznawane są tylko znaczniki otwierające i zamykające. Nie są obsługiwane elementy DOCTYPE, encje i atrybuty znaczników. Przetwarzanie polega na usunięciu znaczników (ang. *markup tags*) z wynikowego tekstu zwykłego.
- HTML_SECTION_GROUP — ta grupa sekcji jest oczywiście przeznaczona dla dokumentów HTML. Obsługiwane są nie tylko znaczniki HTML 4.0 ale i znaczniki nieznane. Usuwana jest zawartość elementów SCRIPT i STYLE oraz komentarze. Zachowywana jest jednak treść sekcji TITLE.

- `XML_SECTION_GROUP` — moduł podziału na sekcje traktuje tekst wejściowy jako dokument XML. Obsługiwane są wewnętrzne encje, wewnętrzne elementy `DOCTYPE` oraz atrybuty znaczników. Znaczniki są usuwane, natomiast wewnętrzne encje w odpowiedni sposób przetwarzane na wyjściową postać tekstu zwykłego. Indeksowaniu podlegają również sekcje użytkownika zdefiniowane w grupie sekcji.
- `AUTO_SECTION_GROUP` — bardzo podobna do grupy `XML_SECTION_GROUP`. Tekst wejściowy jest traktowany jako dokument XML ale nie są obsługiwane sekcje zdefiniowane przez użytkownika. Zamiast tego wszystkie niepuste znaczniki są automatycznie indeksowane jako sekcje strefowe, o nazwach sekcji takich samych jak nazwy znaczników.
- `PATH_SECTION_GROUP` — podobna do grupy `AUTO_SECTION_GROUP` w zakresie przetwarzania znaczników i atrybutów występujących w sekcji. W tym przypadku moduł podziału na sekcje dopuszcza stosowanie nowych operatorów `HASPATH()` i `INPATH()` do przeprowadzania wyszukiwań typu XPATH. Ponadto, podczas wyszukiwania przeprowadzanego w ramach sekcji w nazwach atrybutów i znaczników są rozróżniane małe i wielkie litery.

Sekcje posiadają trzy ważne atrybuty: `tag`, `name` i `type`. Atrybut `tag` informuje moduł o sposobie, w jaki sekcja ma zostać rozpoznana. Gdy moduł znajduje sekwencję *<znacznik> ... </znacznik>*, wówczas automatycznie ją indeksuje. Znaczniki są niepowtarzalne w ramach w sekcji należących do danej grupy. Odwołania do nazw sekcji znajdują się w zapytaniach. Wiele znaczników może zostać odwzorowanych na tę samą nazwę sekcji. Są wówczas traktowane jako instancje tej samej sekcji. Oto najważniejsze typy sekcji w przypadku dokumentów XML:

- **sekcje strefowe** (ang. *zone sections*) — mogą się powtarzać a każda instancja jest w semantyce zapytań traktowana osobno. Sekcje strefowe mogą zawierać inne sekcje (w tym również sekcje strefowe) oraz same stanowić części innych sekcji. W przypadku grup `AUTO_SECTION_GROUP` i `PATH_SECTION_GROUP` moduł podziału na sekcje automatycznie indeksuje niepuste znaczniki jako sekcje strefowe, o nazwach takich samych jak nazwy znaczników.
- **sekcje pól** (ang. *field sections*) — moduł podziału na sekcje wydziela zawartość takiej sekcji i osobno indeksuje jej elementy. Dzięki temu zapytania odwołujące się do sekcji pól są obsługiwane do trzech razy szybciej niż zapytania skierowane do sekcji strefowych, zwłaszcza gdy znaczniki tej sekcji występują w każdym dokumencie. Większa szybkość oznacza zmniejszenie elastyczności. Sekcje pól są z założenia stosowane w przypadku sekcji niepowtarzających się i nienakładających się na siebie. Jeżeli sekcja pola powtarza się w tym samym dokumencie, wówczas jest traktowana jako swoja kontynuacja a nie jako odrębna instancja. Jeżeli nakłada się z sobą samą lub z inną sekcją pola, wtedy jest niejawnie zamykana w tym miejscu, gdzie rozpoczyna się tamta druga sekcja. W jednej grupie sekcji może znajdować się do 64 sekcji pól.
- **sekcje atrybutów** (ang. *attribute sections*) — sekcje atrybutów mogą być dodawane wyłącznie do grup `XML_SECTION_GROUP`, `AUTO_SECTION_GROUP` i `PATH_SECTION_GROUP`. Moduł podziału na sekcje przeprowadza indeksowanie zawartości atrybutu sekcji i udostępnia ją dla zapytań. W przypadku grupy `AUTO_SECTION_GROUP` lub `PATH_SECTION_GROUP` moduł ten automatycznie indeksuje wartości atrybutów jako sekcje atrybutów o nazwach *<znacznik>*@*<atrybut>*.

- **sekcje stop** (ang. *stop sections*) — sekcji tych można używać tylko w odniesieniu do grup AUTO_SECTION_GROUP i PATH_SECTION_GROUP. Sekcja stop oznacza, że odpowiedni <*znacznik*> ma zostać zignorowany i nie powinien być indeksowany jako sekcja. Liczba sekcji stop w grupach AUTO_SECTION_GROUP i PATH_SECTION_GROUP jest nieograniczona.

Moduł podziału na leksemy

Moduł podziału na leksemy (ang. *lexer*) pobiera zwykły tekst od modułu podziału na sekcje i dzieli go na dyskretne elementy leksykalne lub słowa. Oracle Text zawiera moduły obsługi języków, w których słowa są rozgraniczane odstępami (ang. *whitespace-delimited languages*) a także wyspecjalizowane moduły dla języków azjatyckich, w przypadku których segmentacja jest bardziej złożona. Domyślny moduł podziału na leksemy (basic_lexer) posiada również funkcje obsługi *tematów* (ang. *themes*), co pozwala na tworzenie ujednoliconych indeksów tekstowych i tematycznych. Podobnie jak składy danych, moduły podziału na leksemy są tworzone i wstawiane do potoku za pomocą poniższego systemu preferencji:

```
ctx_ddl.create_preference ('my_basic_lexer', 'basic_lexer');
...
create_index xml_index on bookstore (xml_text)
   ...
   parameters ('lexer my_basic_lexer ...');
```

Pod koniec procesu system Oracle9*i* z modułu podziału na leksemy pobiera elementy leksykalne, natomiast z modułu podziału na sekcje odczytuje *pozycje sekcji* (ang. *section offsets*). Na ich podstawie zbudowany zostaje indeks odwrócony, w którym są przechowywane informacje o elementach leksykalnych oraz dokumentach, w których te elementy występują.

Praca z Oracle Text

Poniższy kod źródłowy zawiera typową sekwencję instrukcji SQL, której można użyć w Oracle Text pracując z dokumentami strukturalnymi:

Listing 6.1. *Typowa sekwencja instrukcji SQL w Oracle Text na przykładzie tabeli bookstore*

```
create table  bookstore (
     pk        NUMBER PRIMARY KEY ,
     xml_text  CLOB
) ;
insert into bookstore values (111,
  '<Book>
      <Id>4<\Id>
      <Author>John LeCarre</Author>
      <Title>The Night Manager</Title>
   </Book>');
insert into bookstore values (112,
   '<Book>
      <Id>5<\Id>
```

```
      <Author>John Grisham</Author>
      <Title>The Client</Title>
   </Book>');

/* ... dodawanie pozostałych książek do księgarni */

commit;

begin
   ctx_ddl.create_section_group('xml_sections', 'XML_SECTION_GROUP');
   ctx_ddl.add_zone_section    ('xml_sections', 'titlesec', 'title');
   ctx_ddl.add_zone_section    ('xml_sections', 'authorsec', 'author');
   ctx_ddl.create_preference   ('my_basic_lexer', 'basic_lexer');
   ctx_ddl.set_attribute       ('my_basic_lexer', 'index_text','true');
   ctx_ddl.set_attribute    ('my_basic_lexer', 'index_themes','false');
end;

create index  xml_index  on  bookstore (xml_text)
   indextype is ctxsys.context
   parameters ( 'lexer my_basic_lexer  SECTION GROUP xml_sections' ) ;

select pk  from bookstore
   where contains (xml_text, 'LeCarre WITHIN authorsec') > 0 ;
```

Powyższy kod, w którym wykorzystano odwołania do aplikacji Bookstore, jest prostym przykładem zastosowania SQL w Oracle Text. Ilustruje on proces tworzenia i wypełniania tabeli oraz tworzenia grupy sekcji i indeksu tekstowego. Zawiera ponadto jedno proste zapytanie.

Najpierw jest tworzona tabela. Do identyfikacji każdego dokumentu użyto klucza głównego. Dokument XML został zdefiniowany jako obiekt *CLOB* (ang. *Character Large Object*) systemu Oracle. Składa się on ze znaków należących do zestawu znaków bazy danych Oracle9*i*. Tabela jest wypełniana dokumentami zawierającymi dane o książkach, których liczebność może sięgać dziesiątków tysięcy egzemplarzy.

Następnie jest tworzona grupa `XML_SECTION_GROUP`, zawierająca dwie sekcje: dla znacznika `<author>` oraz dla znacznika `<title>`:

```
ctx_ddl.create_section_group('xml_sections', 'XML_SECTION_GROUP');
ctx_ddl.add_zone_section    ('xml_sections', 'titlesec', 'title');
ctx_ddl.add_zone_section    ('xml_sections', 'authorsec', 'author');
```

W przypadku tej grupy sekcji następuje odwzorowanie na kolumnę `xml_text`. Oznacza to, że tekst jest traktowany przez motor Oracle Text jako strukturalny dokument XML. Znaczniki otwierające, znaczniki zamykające oraz wewnętrzna definicja typu dokumentu (DTD) są rozpoznawane jako osobne elementy. Elementy `<author>` i `<title>` zostają ponadto przygotowane do procedury indeksowania dla celów późniejszego wyszukiwania. Potem jest budowany indeks tekstowy a do potoku zostają włączone podstawowy moduł podziału na leksemy (`basic_lexer`) oraz grupa sekcji:

```
create index  xml_index  on  bookstore (xml_text)
   indextype is ctxsys.context
   parameters ( 'lexer my_basic_lexer  SECTION GROUP xml_sections' ) ;
```

Klauzula `indextype` oznacza, że ma zostać zbudowany indeks tekstowy a nie zwykły indeks w postaci drzewa zbalansowanego (ang. *B-tree*). Po jego utworzeniu, w bazie danych zawierającej przykładowe dwa dokumenty można już przeprowadzać wyszukiwanie kontekstowe, używając w tym celu funkcji `contains`:

```
select pk  from bookstore
   where contains (xml_text, 'LeCarre WITHIN authorsec') > 0 ;
```

W wyniku zapytania zostają zwrócone wszystkie wiersze tabeli `bookstore`, w których kolumna tekstowa zawiera ciąg `LeCarre` w elemencie `<author>`. Język SQL w wersji Oracle nie posiada (na razie) zaimplementowanego typu logicznego, dlatego należało użyć operatora `> 0`. W powyższym przykładzie wynik zapytania byłby następujący:

```
   PK
--------
   111
```

W kolejnych punktach, na przykładzie aplikacji Bookstore, zostaną omówione poszczególne etapy przetwarzania potokowego. Pozwoli to na pokazanie zastosowania różnych technik użycia narzędzia Oracle Text.

Skład danych

Skład danych w Oracle Text zarządza procesem odczytu danych z kolumn bazy danych.

Bezpośredni skład danych (ang. *direct datastore*) jest najprostszy. Operuje on na danych składowanych w kolumnie indeksowanej. Nie wymaga on przystosowywania atrybutów do potrzeb użytkownika. Bezpośredni skład danych jest składem domyślnym (`DEFAULT_DATASTORE`), dlatego nie trzeba go jawnie dodawać do potoku Oracle Text. W przykładzie aplikacji Bookstore użyto właśnie bezpośredniego składu danych.

Plikowy skład danych (ang. *file datastore*) odczytuje dane kolumn z pliku. Po otwarciu pliku czyta dane, traktując jego zawartość jako kolumnę indeksowaną. Oto zmodyfikowana tabela `bookstore`, z plikowym składem danych:

Listing 6.2. *Zmodyfikowana tabela bookstore z plikowym składem danych*

```
create table  bookstore (
     id        NUMBER PRIMARY KEY ,
     xml_text  varchar2(2000)
) ;
insert into bookstore values (111, 'book1.xml');
insert into bookstore values (112, 'book2.xml');
...
begin
   ctx_ddl.create_preference ('xml_files', 'FILE_DATASTORE');
   ctx_ddl.set_attribute ('xml_files', 'PATH', '/xml/files');
   ...
end;
...
create index  xml_index  on  bookstore (xml_text)
   indextype is ctxsys.context
   parameters ('datastore xml_files lexer my_basic_lexer  SECTION GROUP
xml_sections');
```

Skład danych URL (ang. *URL datastore*) odczytuje dane kolumn jako adresy URL. Obsługiwane są protokoły HTTP, FTP oraz bezpośredni dostęp do plików. Gdyby dane o książkach miały być przechowywane w miejscach wskazywanych przez adresy URL, wówczas omawiany przykład wyglądałby następująco:

```
...
begin
   ctx_ddl.create_preference ('xml_urls', 'URL_DATASTORE');
   ctx_ddl.set_attribute ('xml_urls', 'HTTP_PROXY', '<serwer-proxy>');
   ctx_ddl.set_attribute ('xml_urls', 'Timeout', '300');
   ...
end;
insert into bookstore values (111, 'file://xml/files/book1.xml');
insert into bookstore values (112, 'file://xml/files/book2.xml');
...
create index  xml_index  on  bookstore (xml_text)
   indextype is ctxsys.context
   parameters ('datastore xml_urls lexer my_basic_lexer  SECTION GROUP xml_sections');
```

Skład danych użytkownika (ang. *user datastore*) jest oparty na procedurze składowanej, która syntetyzuje dokument. Jeżeli informacje o książkach z przykładowej aplikacji Bookstore są przechowywane w postaci zwykłego tekstu, wówczas składu danych użytkownika można użyć do przekształcenia ich na postać dokumentu XML. Oto zmodyfikowany kod przykładowego programu, w którym wykorzystano skład danych użytkownika:

Listing 6.3. *Zmodyfikowana tabela bookstore ze składem danych użytkownika*

```
create table  bookstore (
     pk        NUMBER PRIMARY KEY,
     id        INTEGER,
     author    VARCHAR2(80),
     title     VARCHAR2(80),
     xml_text  CLOB               /* miejsce na skład danych użytkownika */
) ;
insert into bookstore (pk, id, author, title) values (111, 4, 'John LeCarre', 'The Spy
➥Who Came in from the Cold');
insert into bookstore (pk, id, author, title) values (112, 5, 'John Grisham', 'The
➥Client');
...

/* procedura składowana */
create procedure toXML (rid in rowid, tlob in out tlob)
   offset number := 1;
begin
   for book in (select id,author,title,xml_text from bookstore where rowid = rid)
   loop
      synthesize_XML (tlob, book.id, book.author, book.title, offset);
      dbms_lob.append(tlob, book.xml_text);
   end loop;
end toXML;

grant execute on toXML to public;
connect xml/bookstore

begin
   ctx_ddl.create_preference ('xml_proc', 'user_datastore');
```

```
   ctx_ddl.set_attribute     ('xml_proc', 'procedure', 'toXML');
   ...
end;
...
create index  xml_index  on  bookstore (xml_text)
   indextype is ctxsys.context
   parameters ('datastore xml_proc lexer my_basic_lexer SECTION GROUP xml_sections');
```

Procedura pomocnicza `synthesize_XML` nie została tutaj pokazana. Procedura ta pobiera dane o książkach i składa tymczasowy dokument XML (w postaci obiektu *TLOB*).

```
'<Book><Id>' || book.id || '<\Id><Author>' || book.author || '</Author><Title>' ||
➥book.title '</Title> </Book>'
```

Następnie obiekt `TLOB` (ang. *Temporary Large Object*) jest kopiowany do kolumny `xml_text`. Dzięki temu reszta kodu SQL jest niemal taka sama jak w poprzednich przykładach. Najważniejszą różnicą jest utworzenie składu danych użytkownika a nie składu plikowego czy składu URL.

Sekcje pól i sekcje strefowe

Po niewielkiej modyfikacji omawianego przykładu można pokazać zastosowanie sekcji pól i sekcji strefowych.

Pokazane zostaną tylko te instrukcje SQL, które są tematycznie związane z bieżącym zagadnieniem.

```
...
insert into bookstore values  (
   111,
   '<Book>
      <Id>4<\Id>
      <Authors>
          <Author1>John Kay</Author1>
          <Author2>Mary Powell</Author2>
      </Authors>
      <Title>Don Juan </Title>
   </Book>'
);
insert into bookstore values  (
   112,
   '<Book>
      <Id>5<\Id>
      <Authors>
          <Author1>Juan Smith</Author1>
      </Authors>
      <Title>One Fine Day </Title>
   </Book>'
);
...
begin
   ...
   ctx_ddl.add_zone_section    ('xml_sections', 'authors', 'authors');
```

```
    ctx_ddl.add_zone_section    ('xml_sections', 'authorsec', 'author1');
    ctx_ddl.add_zone_section    ('xml_sections', 'authorsec2', 'author2');
    ...
end;

create index  xml_index  on  bookstore (xml_text)
    indextype is ctxsys.context
    parameters ('SECTION GROUP xml_sections ...');
```

Po wysłaniu zapytania:

```
select pk  from bookstore where
    contains (xml_text, 'Mary WITHIN authorsec') > 0 ;
```

dokument nie zostanie znaleziony. Mimo iż `Mary` jest imieniem autorki, nie znajduje się ono w sekcji `authorsec`. Oryginalne rozwiązanie można zmodyfikować poprzez odwzorowanie nazw `<author1>` i `<author2>` na jedną nazwę sekcji:

```
begin
    ...
    ctx_ddl.add_zone_section    ('xml_sections', 'authorsec', 'author1');
    ctx_ddl.add_zone_section    ('xml_sections', 'authorsec', 'author2');
    ...
end;
```

Po wysłaniu takiego zapytania dokument zostanie znaleziony, ponieważ znaczniki `<author1>` i `<author2>` są traktowane jako pojedyncza sekcja `authorsec`.

Jak już wspominano w tym rozdziale, każda instancja sekcji strefowej jest traktowana osobno. Innymi słowy, po wysłaniu zapytania:

```
select pk  from bookstore where
    contains (xml_text, '(Mary and Powell) WITHIN authorsec') > 0 ;
```

dokument zostanie znaleziony, natomiast po wysłaniu zapytania takiego, jak poniżej:

```
select pk  from bookstore where
    contains (xml_text, '(Mary and John) WITHIN authorsec') > 0 ;
```

wynik tego zapytania będzie pusty.

Mimo iż książka odpowiadająca pierwszemu dokumentowi posiada autorów o imionach Mary i John, należących do sekcji `authorsec`, to jednak nie należą one do tej samej sekcji `authorsec`. Gdyby jednak znacznik nadrzędny `<authors>` również został odwzorowany jako sekcja `authorsec`:

```
    ctx_ddl.add_zone_section    ('xml_sections', 'authorsec', 'authors');
```

wówczas wynik ostatniego zapytania nie byłby pusty, ponieważ oba imiona znalazłyby się w tej samej sekcji.

Jeżeli programista jest zainteresowany wyłącznie znajdywaniem imion i nie ma znaczenia, w jakiej sekcji one się pojawiają, wówczas powinien on skonfigurować poszczególne sekcje w następujący sposób:

```
begin
    ...
    ctx_ddl.add_zone_section    ('xml_sections', 'authors', 'authors');
    ctx_ddl.add_zone_section    ('xml_sections', 'authorsec', 'author1');
```

```
    ctx_ddl.add_zone_section    ('xml_sections', 'authorsec', 'author2');
    ctx_ddl.add_zone_section    ('xml_sections', 'authorsec', 'title');
    ...
end;
```

Teraz zapytanie w postaci:

```
select pk  from bookstore where
   contains (xml_text, 'Juan within names') > 0 ;
```

wyszuka oba dokumenty, ponieważ w pierwszym dokumencie imię Juan jest zawarte w części `<title>`, natomiast w drugim — w części `<author1>`. Warto zauważyć, że takie skonfigurowanie sekcji pozwala na rozróżnianie pomiędzy nazwami w części `<title>` i nazwami w częściach `<author1>` i `<author2>`, na przykład za pomocą następującego zapytania zagnieżdżonego (czyli wykorzystującego klauzulę `WITHIN`):

```
select pk  from bookstore where
   contains (xml_text, '(Juan within names) WITHIN authors') > 0 ;
```

Zapytanie zagnieżdżone pozwala na znalezienie drugiego dokumentu, ponieważ tylko ten dokument zawiera imię Juan w części `<authors>`. Takie zapytania mogą być używane wyłącznie w przypadku sekcji strefowych. Oracle Text indeksuje sekcje strefowe, zatem sekcje równoważne nie mogą być rozróżniane. Na przykład zapytanie:

```
select pk  from bookstore where
   contains (xml_text, '(Juan within authors) WITHIN names') > 0 ;
```

również jest realizowane pomyślnie. Dzieje się tak, ponieważ sekcje obejmują dokładnie ten sam zakres. Zapytanie zagnieżdżone nie odzwierciedla dokładnie relacji między elementem nadrzędnym i elementem podrzędnym. Tak samo będzie traktowana relacja danego elementu zarówno z elementem podrzędnym o dwa poziomy (ang. *parent-grandchild*), jak i o trzy poziomy (ang. *parent-great-grandchild*). Jak już wyjaśniono wcześniej w rozdziale, każda instancja sekcji pola w ramach dokumentu jest traktowana jako kontynuacja takiej sekcji. Jeżeli w przykładzie utworzone zostaną sekcje pól:

```
begin
   ...
   ctx_ddl.add_field_section    ('xml_sections', 'authors', 'authors');
   ctx_ddl.add_field_section    ('xml_sections', 'authorsec', 'author1');
   ctx_ddl.add_field_section    ('xml_sections', 'authorsec', 'author2');
   ...
end;
```

wówczas poniższe zapytanie:

```
select pk  from bookstore where
   contains (xml_text, '(Mary and John) WITHIN authorsec') > 0 ;
```

zostanie zrealizowane pomyślnie, mimo iż w przypadku sekcji strefowych `authorsec` dokument nie mógł zostać znaleziony za jego pomocą. Moduł podziału na sekcje konkatenuje poszczególne instancje sekcji pól dla każdego dokumentu. Sekcje pól są lepsze w przypadku sekcji niepowtarzających się, takich jak `<title>`. Są one wyodrębniane z dokumentu i indeksowane osobno.

Oznacza to, że za pomocą zapytania niezagnieżdżonego w postaci:

```
select pk  from bookstore where
   contains (xml_text, '4') > 0 ;
```

nie można znaleźć właściwego dokumentu, mimo iż umieszczono w tym zapytaniu cyfrę 4. Gdyby indeksowanie było oparte na sekcjach strefowych, to samo zapytanie zostałoby zrealizowane pomyślnie. Sytuację tę można zmienić, sprawiając, by odpowiednia sekcja pola stała się widoczna. W tym celu wystarczy określić wartość opcjonalnego, czwartego argumentu `add_field` (typu logicznego) jako `TRUE`:

```
ctx_ddl.add_field_section ('xml_sections', 'authorsec', 'author1', TRUE);
```

Teraz zawartość sekcji pola jest widoczna dla zapytań niezagnieżdżonych. Jest to możliwe dzięki podwójnemu indeksowaniu słowa, będącego zarówno częścią wyodrębnionej sekcji, jak i częścią dokumentu. Takie rozwiązanie zabiera oczywiście odpowiednio większą część przestrzeni indeksowej.

Wewnętrzne elementy `DOCTYPE` są analizowane w grupach `XML_SECTION_GROUP` oraz `PATH_SECTION_GROUP`. W tym przypadku należy tworzyć sekcje pól ograniczone nazwami definicji typu (DTD), dzięki czemu znaczniki należące do różnych definicji `DOCTYPE` będą mogły być rozróżniane. Składnia znacznika jest następująca: `(<nazwa_definicji_doctype>) <nazwa_znacznika>`:

```
begin
   ctx_dll.create_section_group('xml_sections', 'XML_SECTION_GROUP');
   ctx_ddl.add_field_section ('xml_section', '(Bookstore)authorsec',
                                                        'author');
   ctx_ddl.add_field_section ('xml_section', '(MyBookstore)names',
                                                        'author');
   ctx_ddl.add_field_section ('xml_section', 'authorsec', 'author');
   ...
end;
```

Teraz, gdy moduł podziału na sekcje rozpoznaje znacznik `<author>`, indeksuje go jako sekcję `authorsec` (jeżeli elementem `DOCTYPE` jest `Bookstore`) lub jako sekcję `names` (jeżeli elementem `DOCTYPE` jest `MyBookstore`). W sekcji grupy mogą znajdować się zarówno sekcje ograniczone nazwami `DOCTYPE`, jak i nieograniczone tymi nazwami. Sekcje ograniczone nazwami elementów `DOCTYPE` dotyczą tylko tych elementów `DOCTYPE`. Dla wszystkich pozostałych elementów `DOCTYPE` mają wówczas zastosowanie sekcje nieograniczone nazwami `DOCTYPE`. Nie ma to wpływu na wyszukiwanie, ponieważ jest ono dokonywane na podstawie nazw sekcji.

Sekcje stop

W przypadku grup `AUTO_SECTION_GROUP` i `PATH_SECTION_GROUP` moduł podziału na sekcje automatycznie indeksuje niepuste znaczniki jako sekcje strefowe, o nazwach takich samych jak nazwy znaczników. Niektóre znaczniki mogą nie być istotne dla aplikacji, dlatego ich indeksowanie byłoby stratą czasu, gdyż nigdy nie byłyby one wykorzystywane do wyszukiwania informacji. Do obsługi takich znaczników służą sekcje `stop`, których można używać w grupach `AUTO_SECTION_GROUP` i `PATH_SECTION_GROUP`. Sekcje te wskazują znaczniki, które podczas indeksowania powinny zostać pominięte. Jeżeli na przykład grupa `PATH_SECTION_GROUP` zostanie utworzona w następujący sposób:

```
begin
   ctx_ddl.create_section_group('xml_sections', 'PATH_SECTION_GROUP');
   ctx_ddl.add_stop_section    ('xml_section', 'Id');
   ...
end;
```

wówczas znacznik <Id> zostanie zignorowany i będzie indeksowany jako sekcja. Liczba sekcji stop w grupie jest nieograniczona. Ograniczniki DOCTYPE są obsługiwane. Na przykład instrukcja:

```
ctx_ddl.add_stop_section  ('xml_section', '(MyBookstore)author');
```

oznacza, że znacznik <author> będzie ignorowany wyłącznie w ramach elementu DOCTYPE o nazwie MyBookstore.

Sekcje atrybutów

Sekcje atrybutów w grupach XML_SECTION_GROUP i PATH_SECTION_GROUP pozwalają na indeksowanie wartości atrybutów sekcji i wyszukiwanie ich za pomocą zapytań. Podczas wyszukiwania w grupie PATH_SECTION_GROUP w nazwach atrybutów i znaczników są rozróżniane małe i wielkie litery. Oto zmodyfikowany przykład Bookstore, w którym wykorzystano sekcje atrybutów:

```
insert into bookstore values (
111,
  '<Book id = "4"
         author = "John LeCarre"
         title = "The Night Manager" />'
);
insert into bookstore values (112,
   '<Book id = "5"
          author = "Juan Smith"
          title = "The Night Club" />'
);
...
begin
   ctx_ddl.create_section_group('xml_sections', 'XML_SECTION_GROUP');
   ctx_ddl.add_attr_section    ('xml_sections', 'authorsec', 'book@author');
   ctx_ddl.add_attr_section    ('xml_sections', 'titlesec', 'book@title');
   ...
end;

create index  xml_index  on  bookstore (xml_text)
   indextype is ctxsys.context
   parameters ( 'SECTION GROUP xml_sections ...' ) ;
```

Nazwa atrybutu jest używana w połączeniu z nazwą znacznika, do którego ten atrybut należy:

<nazwa_znacznika>@<nazwa_atrybutu>

Oto przykładowe zapytania do bazy danych Bookstore:

```
select pk  from bookstore where
   contains (xml_text, 'Juan WITHIN author') > 0 ;

select pk  from bookstore where
   contains (xml_text, '(The Night Manager) WITHIN title') > 0 ;

select pk  from bookstore where
   contains (xml_text, '(Juan and John) WITHIN author') > 0 ;
```

W pierwszym i drugim przypadku zostaną zwrócone znalezione dokumenty. Trzecie zapytanie natomiast nie zostanie zrealizowane, ponieważ wystąpienia sekcji atrybutów podanych w zapytaniach są rozróżniane. Warto zwrócić uwagę na to, że realizacja dwóch pierwszych zapytań w grupie PATH_SECTION_GROUP zakończyłaby się niepowodzeniem, gdyby w nazwach znaczników użyto wielkich liter:

```
select pk  from bookstore where
   contains (xml_text, 'Juan WITHIN AUTHOR') > 0 ;

select pk  from bookstore where
   contains (xml_text, '(The Night Manager) WITHIN TITLE') > 0 ;
```

Tekst atrybutu jest uznawany za niewidoczny, dlatego też za pomocą zapytań niezagnieżdżonych, takich jak to przedstawione poniżej:

```
select pk  from bookstore where
   contains (xml_text, 'Juan') > 0 ;
```

nie można znaleźć szukanego dokumentu. Nazwy sekcji atrybutów nie mogą nakładać się na nazwy sekcji strefowych i sekcji pól. Dopuszczalne jest odwzorowywanie wielu nazw atrybutów na pojedyncze nazwy sekcji. Mimo iż definicje typów dokumentów (DTD) są analizowane, to określone w nich domyślne wartości ich atrybutów nie są obsługiwane. Innymi słowy, w przypadku pominięcia rzeczywistej wartości atrybutu w dokumencie wartość domyślna definicji DTD nie zostaje uwzględniona w procesie indeksowania.

W przypadku grup AUTO_SECTION_GROUP i PATH_SECTION_GROUP moduł podziału na sekcje automatycznie indeksuje wartości atrybutów jako sekcje atrybutów o nazwach *<znacznik>*@*<atrybut>*.

Przykładowo, po wpisaniu zapytania:

```
select pk  from bookstore where
   contains (xml_text, '(The Night Manager) WITHIN book@title') > 0 ;
```

przeprowadzane jest wyszukiwanie wewnątrz wartości atrybutu title, ponieważ atrybut ten był automatycznie indeksowany w postaci sekcji atrybutu.

Wyszukiwanie według składni XPATH w grupie PATH_SECTION_GROUP

Począwszy od wersji Oracle9*i* dostępne są dwa nowe operatory: INPATH() i HASPATH(). Dzięki nim w sekcjach grupy PATH_SECTION_GROUP możliwe jest wyszukiwanie informacji za pomocą zapytań typu XPATH. W poprzednim przykładzie zapytanie było skierowane do sekcji title. Aby znaleźć wyrażenie Night w jednej z podsekcji sekcji book (w tym przypadku chodzi o title) można również użyć następującego zapytania:

```
select pk  from bookstore where
   contains (xml_text, 'Night WITHIN INPATH(//bookstore/book)') > 0 ;
```

Możliwość zastępowania nazw sekcji wyrażeniami XPATH pozwala na rozszerzenie zakresu zastosowań operatora WITHIN w systemie Oracle9*i*. Dopuszczalne jest również sprawdzanie atrybutów, jeżeli więc wyrażenie „Night” pojawiło się w wielu sekcjach title ale sekcje te posiadają atrybuty number, to żądane tytuły można znaleźć dopasowując do siebie odpowiednie wartości tych atrybutów:

```
select pk  from bookstore where
   contains (xml_text, 'Night WITHIN INPATH(//bookstore/book/title[@number="one"])') > 0;
```

Jak już wspominano, podczas wyszukiwania w grupie PATH_SECTION_GROUP, w nazwach atrybutów i znaczników są rozróżniane małe i wielkie litery.

Na koniec kilka słów o operatorze HASPATH(), który — analogicznie do operatora INPATH() zwiększającego funkcjonalność operatora WITHIN — rozszerza zastosowanie operatora CONTAINS. Operator HASPATH() pozwala na wyszukiwanie informacji w dokumentach XML zawierających wyrażenia XPATH. Jeżeli szukana ścieżka znajduje się w danym dokumencie, wówczas po wywołaniu operatora HASPATH() zwrócona zostaje wartość 100, tak jak po uruchomieniu poniższego przykładowego kodu, opartego na przykładzie Bookstore:

```
select pk  from bookstore where
   contains (xml_text, 'HASPATH(//bookstore/book/title)') > 0 ;
```

Operator ten może również wyszukiwać w dokumentach XML elementów zawierających określoną wartość, na przykład tytuł „The Night Manager" i zwracać wartość 100 dla tych dokumentów, które zawierają taki tytuł:

```
select pk  from bookstore where
   contains (xml_text, 'HASPATH(BookTitle="The Night Manager")') > 0 ;
```

Dynamiczne dodawanie sekcji

Dodawanie nowych sekcji do istniejącego indeksu bez konieczności jego przebudowywania jest przydatne wtedy, gdy pojawiają się dokumenty z nowymi znacznikami lub nowymi elementami DOCTYPE. Oto przykładowa sekwencja SQL, za pomocą której można dodać do indeksu nową sekcję strefową o nazwie names, wykorzystując znacznik <title>:

```
alter index xml_index rebuild
parameters ('add zone section names tag title')
```

Z kolei poniższa sekwencja SQL umożliwia dodanie do indeksu nowej sekcji pola o nazwie authors. Tutaj wykorzystano do tego celu znacznik <author>:

```
alter index xml_index rebuild
parameters ('add field section authors tag author')
```

Dodanie nowych sekcji modyfikuje wyłącznie metadane indeksu i nie powoduje jego przebudowy. Oznacza to, że nowe sekcje stają się dostępne dla każdego dokumentu indeksowanego już po ich dodaniu do indeksu i nie mają one wpływu na dokumenty indeksowane wcześniej. Jeżeli indeks zawiera już dokumenty, w których znajdują się nowo dodane sekcje, należy je samodzielnie zaznaczyć do ponownego indeksowania, polegającego zwykle na uaktualnieniu indeksowanej kolumny na siebie samą.

Narzędzie *Oracle Text* w obecnej wersji nie potrafi wyszukiwać i wyodrębniać fragmentów dokumentu XML spełniających kryteria zapytania. Innymi słowy, przed rozpoczęciem operacji wyszukiwania należy podzielić duże dokumenty XML na mniejsze, sensowne części. W przyszłości będą dostępne operacje umożliwiające przeprowadzanie bardziej inteligentnego, kontekstowego i szczegółowego wyszukiwania informacji o określonym znaczeniu w dużych dokumentach XML, zawierających dziesiątki tysięcy elementów <book>.