STUDIA INFORMATICA 2001
Volume 22 Number 3 (45)

Piotr CZEKALSKI
Politechnika Śląska, Instytut Informatyki

MODELE HIERARCHICZNYCH STRUKTUR W RELACYJNYCH BAZACH DANYCH O ARCHITEKTURZE KLIENT-SERWER

Streszczenie. W artykule przedstawiono wybrane zagadnienia związane z wykorzystaniem struktur słownikowych w relacyjnych bazach danych oraz aplikacjach zarządzających bazami. Omówiono wykorzystanie struktur hierarchicznych jako prostego narzędzia pozwalającego użytkownikom na poruszanie się po strukturze oraz porównano techniki implementacji słowników w zakresie funkcjonalności i prostoty implementacji aplikacji klienta bazy. Omówienie oparto na elementach systemu rejestru usług medycznych i systemu informatycznego kas chorych.

HIERARCHICAL DATABASE STRUCTURES IN CLIENT-SERVER ARCHITECTURE

Summary. This article covers selected problems involved with hierarchical dictionaries implementation. It contains functionality and client implementation comparison, particularly simplicity and usability. The comparison is based on parts of Medical Services Registering System and software solutions for National Health Services.

1. Rola słowników w strukturach relacyjnych baz danych

Słowniki stanowią znaczącą grupę tabel w większości systemów opartych na serwerach relacyjnych baz danych. Znaczna część operacji wprowadzania i modyfikacji danych w aplikacjach klienta związana jest z podawaniem (lub wybieraniem z innej struktury danych) jednej z wielu możliwych pozycji. Przykładem takiej operacji może być podanie kodu terytorialnego miejsca zamieszkania osoby podczas wpisywania danych do bazy osobowej lub wypełnienie arkusza choroby kodami chorób wg międzynarodowej klasyfikacji

ICD-10 w systemie rejestracji usług medycznych. Należy zaznaczyć, że słownik liniowy jest traktowany jako szczególny przypadek słownika hierarchicznego, jednopoziomowego.

2. Przegląd rozwiązań

2.1. Struktura liniowa z wykorzystaniem pól opisowych

Naturalne dążenie operatorów wszystkich systemów informatycznych skłania się w kierunku wykorzystywania rozbudowanych nazw pozycji, pozwalających w prostszy sposób na zapamiętanie i późniejsze odnalezienie danych, niż posługiwanie się bezimiennymi i niejednokrotnie skomplikowanymi kodami. Z drugiej strony takie podejście powoduje znaczny wzrost objętości tabel w bazie danych i wiąże się z późniejszym brakiem możliwości wygenerowania wiarygodnych statystyk w związku z częstymi błędami popełnianymi przez operatora (literówki, zamiana małych i dużych liter, różna liczba białych znaków w odstępach między wyrazami itp.) i brakiem zdefiniowanego zakresu dopuszczalnych wartości. Przykład tabeli głównej został zamieszczony na rys. 1.

Wartości wpisywane do pola CHOROBA tworzą słownik chorób w formie opisów.

KARTA_CHOROBY		
ID	NUMBER	<pk>
STATUS	CHAR(1)	
CHOROBA	VARCHAR2(128)	
DATA	DATE	

Rys. 1. Przykładowa tabela z polem opisowym CHOROBA
Fig. 1. A sample table with the descriptive field CHOROBA (disease)

Głównymi wadami takiego rozwiązania są:

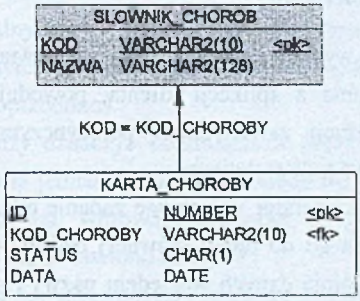
- brak możliwości jawnej hierarchizacji grup danych (w opisywanym przypadku – grup chorobowych),
- brak możliwości generowania statystyk o satysfakcjonującej wiarygodności ze względu na małą powtarzalność wprowadzanych opisów,
- brak kontroli nad wprowadzanymi opisami z poziomu narzędzi operujących bezpośrednio na bazie danych,
- utrudnione lub wręcz uniemożliwione przeprowadzenie zastąpienia jednych danych innymi w sposób zautomatyzowany (problem taki pojawia się np. podczas zmian w słowniku chorób wg międzynarodowej klasyfikacji ICD-10),
- znaczny nadmiar gromadzonych informacji, związany z koniecznością wielokrotnego wpisywania tych samych danych dla poszczególnych wierszy.

Rozwiązanie to cechuje się również pewnymi zaletami. Są nimi:

- duża elastyczność rozwiązania, związana z brakiem konieczności edycji słownika w przypadku wpisywania wartości wcześniej nie występującej w tabeli,
- łatwość implementacji aplikacji klienta (zarządzającej bazą).

2.2. Struktura liniowa z wykorzystaniem zewnętrznego słownika

Metoda przechowywania w zewnętrznym słowniku danych opisowych w powiązaniu z kodem stanowi rozwiązanie większości problemów zasygnalizowanych w punkcie 2.1. Jest to najczęściej występujący przypadek, w którym kod słownika jest kluczem obcym w tabeli głównej przechowującej dane, tak jak to pokazano na rys. 2 .



Rys. 2. Struktura z zewnętrznym słownikiem chorób
Fig. 2. A structure containing an external diseases dictionary

Niewątpliwą wadą takiego rozwiązania jest konieczność stosowania czasochłonnych złączeń, co może mieć niebagatelny wpływ na jakość aplikacji, w szczególności tych, które muszą zapewnić obsługę wielu danych i klientów jednocześnie, oraz krytycznie niski czas dostępu do informacji.

Prezentowane rozwiązanie jest stosowane praktycznie we wszystkich systemach informatycznych wykorzystujących relacyjne bazy danych.

2.2.1. Słowniki wewnętrzne i buforowane

Alternatywnym rozwiązaniem dla słowników przechowywanych w tabelach są słowniki wewnętrzne, „zaszyte" w kodzie programu. Jest to rozwiązanie funkcjonalnie porównywalne do opisanego w punkcie 2.2, lecz do takiego wykorzystania nadają się jedynie słowniki, które nie będą modyfikowane (wiąże się to z koniecznością ponownej kompilacji programu) i nie są zbyt duże (kilka, kilkanaście pozycji). Widocznym rezultatem stosowania słowników wewnętrznych jest zmniejszenie obciążenia sieci, gdyż nie następuje tu transmitowanie zawartości słowników z bazy danych.

Pośrednim rozwiązaniem pomiędzy słownikiem zewnętrznym i słownikiem „zaszytym" jest metoda buforowania słownika w pamięci programu klienta. Buforowanie polega na jednorazowym wczytywaniu danych ze słownika zewnętrznego, znajdującego się w bazie danych, do przygotowanych wcześniej struktur w programie. Wczytywanie to odbywa się najczęściej podczas startu aplikacji klienta.

Takie rozwiązanie gwarantuje elastyczność w przypadku konieczności zmiany zawartości słownika, a z drugiej strony zmniejsza ruch w sieci i obciążenie serwera bazy danych, znacznie przyspieszając wykonanie zapytań.

Obydwa powyższe rozwiązania doskonale nadają się w przypadku słowników o hierarchii jednopoziomowej (słowników liniowych).

2.3. Uniwersalny słownik liniowy

Dążenie do zapewnienia wygody i szybkości pracy zarówno w zakresie opracowywania, jak i późniejszego korzystania z aplikacji klienta, powoduje konieczność wzbogacenia standardowego zestawu pozycji zawartych w pojedynczym rekordzie reprezentującym pozycję słownika o dodatkowe pola techniczne.

Mając na uwadze fakt, iż operator wykonując zadanie odnalezienia i wybrania kodu ze słownika (w celu przypisania go do tabeli głównej) będzie wymagał od oprogramowania klienta możliwości wyszukiwania danych względem nazwy (np. nazwy miasta lub gminy), celowe wydaje się dołożenie do słownika liniowego pozycji przechowującej nazwę zapisaną wielkimi literami, w celu ułatwienia opisywanej operacji. Wiąże się to z faktem, iż większość serwerów baz danych rozróżnia wielkie i małe litery, w związku z czym odnalezienie w słowniku kodów terytorialnych, np. pozycji „krynica" w sytuacji, w której w bazie danych znajduje się pozycja „Krynica", zakończy się porażką. Odpowiadające opisywanemu przykładowi zapytanie w języku SQL (zakładając strukturę tabeli analogiczną do znajdującej się na rys. 2) znajduje się poniżej:

```
SELECT KOD FROM SL_TERYT WHERE NAZWA="krynica";
```

W zaistniałej sytuacji wygodnie jest zmodyfikować warunek tak, aby porównanie następowało względem wielkich lub małych liter. W tym celu można skorzystać z oferowanych przez serwer funkcji konwersji łańcuchów. W przypadku serwera ORACLE są to odpowiednio funkcje `UPPER(char)` i `LOWER(char)`. Konwersja łańcucha, który jest poszukiwany po stronie aplikacji klienta, nie nastręcza większych kłopotów, tak więc powyższe zapytanie powinno zostać zmodyfikowane do następującej postaci:

```
SELECT KOD FROM SL_TERYT WHERE UPPER(NAZWA)="KRYNICA";
```

W przypadku słowników o znacznej liczbie rekordów (np. słownik kodów terytorialnych wg GUS – ok. 3,5 tys. wierszy) spowoduje to znaczne spowolnienie operacji wyszukiwania, gdyż większość serwerów baz danych nie potrafi skorzystać z indeksów, jeżeli w warunku

poszukiwania wykorzystano funkcję konwertującą łańcuch do dużych lub małych liter. Ponieważ w szczególnych przypadkach użyteczne jest wyszukiwanie wg kodu pozycji w słowniku, celowe staje się dołożenie kolumn, w których wartości kodów i nazw są zapisane dużymi literami, co pozwoli na uniknięcie stosowania funkcji konwersji znaków i znacznie przyspieszy operację wyszukiwania.

Poza opisanymi kolumnami należy zapewnić możliwość „wyłączenia" niektórych pozycji słownika (np. w celu pominięcia ich podczas wyświetlania dostępnych opcji do wyboru).

Rozpatrzmy następującą, przykładową sytuację:

- w słowniku kontrahentów znajdują się pozycje odpowiadające firmie, która w bieżącym roku zmieniła osobowość prawną i została dodana do słownika ponownie, figurując pod odmiennym kodem i z odmiennymi danymi,
- w związku z tym należy uniemożliwić wybranie starego kodu operatorowi, który jest odpowiedzialny za wprowadzanie do systemu faktur, aby zapobiec potencjalnej pomyłce (nazwa kontrahenta w bieżącym roku jest identyczna z nazwą w roku poprzednim).

W takim przypadku należy oznaczyć kontrahentów nieaktywnych i uniemożliwić ich wybór. Pozycji tych nie można jednak usunąć ze względu na powiązania z fakturami z lat poprzednich. Rozwiązaniem jest wprowadzenie pola statusowego (flagi). Minimalny proponowany zestaw kolumn przedstawiono na rys. 3 .

Tak zdefiniowany zestaw kolumn można rozszerzać o dodatkowe, zwiększając zakres gromadzonych danych w słowniku.

SLOWNIK_LINIOWY		
KOD	VARCHAR2(10)	<pk>
UPPER_KOD	VARCHAR2(10)	
NAZWA	VARCHAR2(128)	
UPPER_NAZWA	VARCHAR2(128)	
STATUS	CHAR(1)	

Rys. 3. Słownik liniowy
Fig. 3. A linear dictionary

2.4. Struktury hierarchiczne

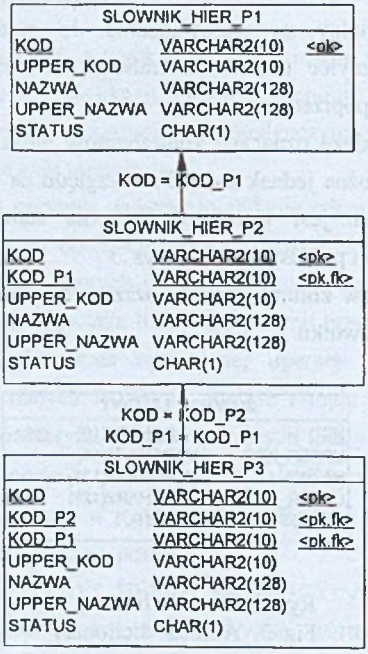
W przypadku konieczności przechowywania w bazie danych struktury o organizacji hierarchicznej, pojawia się problem odwzorowania naturalnych zależności występujących pomiędzy elementami tworzącymi słownik. Założeniem jest, że każdy element (z wyjątkiem elementów znajdujących się na pierwszym poziomie hierarchii) posiada dokładnie jeden element nadrzędny, do którego jest przypisany.

Opracowując taką strukturę należy mieć na uwadze elastyczność rozwiązania, optymalność z punktu widzenia wielkości i czasu wykonania zapytań oraz możliwość łatwego opracowania oprogramowania klienta, pozwalającego na poruszanie się w hierarchii zarówno w głąb, jak i w poziomie.

Istnieje wiele rozwiązań tak postawionego problemu. W następnych punktach przedstawiono niektóre z możliwych rozwiązań.

2.4.1. Realizacja struktury hierarchicznej za pomocą wielu tabel

Jednym z możliwych rozwiązań realizacji struktury danych jest zastosowanie połączonych tabel, z których każda tabela odpowiada jednemu, z góry przyporządkowanemu, poziomowi w hierarchii. Przykład takiej struktury znajduje się na rys. 4 .



Rys. 4. Przykładowa struktura hierarchiczna składająca się z trzech tabel
Fig. 4. A sample three-table hierarchical structure

Jak widać, struktura taka składa się z wielu słowników liniowych, połączonych więzami integralności. Schemat przedstawiony na rys. 4 odpowiada trójpoziomowemu drzewu o dowolnej, ograniczonej jedynie pojemnością pola KOD liczbie potomków w każdej gałęzi. Struktury drzewiaste i algorytmy podstawowych operacji wykonywanych na tych strukturach można znaleźć m.in. w [1].

Implementacja tego typu pozwala na szybkie przeszukiwanie struktury w ramach wybranego poziomu, ale zawiera znaczne ograniczenie związane ze z góry ustaloną maksymalną liczbą poziomów hierarchii. Należy również zaznaczyć fakt, iż każdy poziom cechuje się odmienną listą kolumn (ze względu na import kluczy obcych z tabel nadrzędnych), co znacznie utrudnia napisanie uniwersalnego narzędzia do przeglądania struktur. Dodatkowo, podczas typowej operacji przeszukiwania od ogółu do szczegółu, konieczne staje się wykonywanie zapytań na różnych tablicach w miarę poruszania się w głąb hierarchii.

2.4.2. Realizacja struktury hierarchicznej z wykorzystaniem jednej tabeli – rozwiązanie 1

W celu ułatwienia operacji wyszukiwania i wprowadzania danych oraz uproszczenia aplikacji klienta, korzystne jest takie zmodyfikowanie struktury, aby w całości mieściła się ona w jednej tabeli bazy danych. Przykładem takiej struktury jest tabela ze specjalnymi polami przechowującymi informację o sposobie pionowego powiązania elementów hierarchii oraz informująca o poziomie w zajmowanej hierarchii. Na potrzeby niniejszego opracowania przyjęto, że korzeń znajduje się zawsze na najniższym poziomie i jest oznaczony najmniejszą wartością. Na rys. 5 znajduje się schemat słownika kodów terytorialnych w proponowanej formie.

DX_TERYT		
KOD	VARCHAR2(7)	<pk>
UPPER_KOD	VARCHAR2(7)	
NAZWA	VARCHAR2(64)	
UPPER_NAZWA	VARCHAR2(64)	
D1	NUMBER(4)	
D2	NUMBER(4)	
D3	NUMBER(4)	
D4	NUMBER(4)	
STATUS	CHAR(1)	

Rys. 5. Słownik kodów terytorialnych - rozwiązanie 1
Fig. 5. A hierarchical dictionary of area codes, solution 1

Pozycje D_1 .. D_4 oznaczają położenie w hierarchii elementu, zgodnie z następującymi regułami (*i* oznacza i-ty poziom, *n* głębokość słownika, tutaj równą 4):

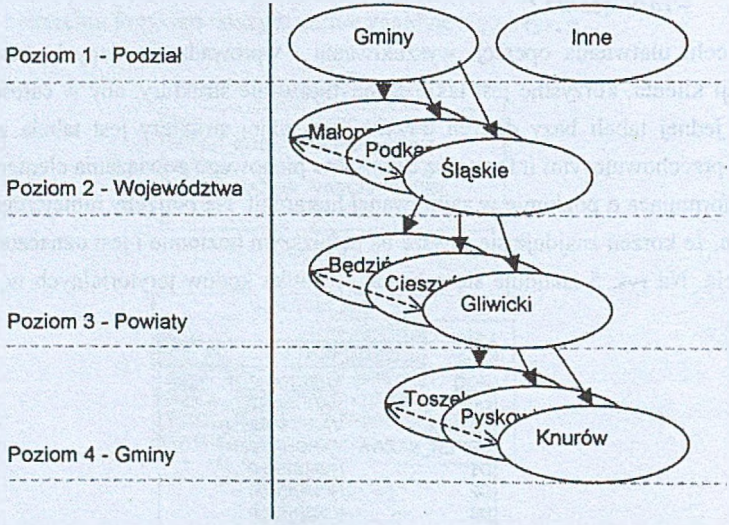
- każda pozycja ma unikalny kod,
- pozycja znajdująca się na i-tym poziomie ma D_{i+1} .. D_n równe 0,
  - pozycja będąca liściem ma wszystkie wartości D_i różne od 0,
  - korzeń ma wszystkie wartości D_i z wyjątkiem D_1 równe 0,
- każdy potomek na i-tym poziomie ma D_1 do D_{i-1} równe z wartościami, które ma jego ojciec,

- jeśli wymagane jest uporządkowanie, porządek na i-tym poziomie, wśród potomków jednego ojca (z i-1 poziomu) jest wyznaczany wg kolumny D_i.

Znaczenie pozostałych pól zostało wyjaśnione w punkcie 2.3 .

Niewątpliwą zaletą w porównaniu do struktury składającej się z wielu tabel (punkt 2.4.1) jest ułatwienie konstrukcji aplikacji klienta, choć np. przygotowanie uniwersalnego zapytania w języku SQL, zwracającego np. elementy znajdujące się w hierarchii na podanym poziomie, wymagałoby dynamicznego konstruowania klauzuli `WHERE` tego zapytania.

Rozważmy wykonanie zapytań na przykładowej grupie danych pochodzących ze słownika kodów terytorialnych. Konstrukcję tego słownika przedstawiono na rys. 6 .



Rys. 6. Wewnętrzna struktura słownika kodów terytorialnych
Fig. 6. A hierarchical dictionary of area codes – internal structure

Poniżej przedstawiono przykładowe zapytania:

- odnalezienie pozycji znajdujących się na pierwszym poziomie hierarchii:

```
select kod, nazwa from dx_teryt where d2=0
```

w wyniku czego otrzymujemy listę pozycji:

```
KOD D1 D2 D3 D4 NAZWA
--- -- -- -- -- ------------
G   1  0  0  0  Gminy
I   2  0  0  0  Inne
```

wykorzystujemy tutaj zależność mówiącą, że jeśli na $D_i = 0$, to również $D_{i+1}=0$.

- odnalezienie wszystkich potomków pozycji „Gmina" w drugim poziomie hierarchii: `select kod, nazwa from dx_teryt t1 where d3 = 0`

```
and d2!=0 and exists (select 1 from dx_teryt t2 where
t2.d1=t1.d1 and t2.kod='G').
```

W wyniku otrzymujemy listę województw, w szczególności województwo śląskie:

```
KOD D1 D2 D3 D4 NAZWA
--- -- -- -- -- ------------
24  1  24 0  0  ŚLĄSKIE
```

- odnalezienie wszystkich potomków pozycji „ŚLĄSKIE", czyli odnalezienie powiatów województwa śląskiego:

```
select kod, d1, d2, d3, d4, nazwa from dx_teryt t1 where d4 = 0 and d3!=0 and exists (select 1 from dx_teryt t2 where t2.d1=t1.d1 and t2.d2=t1.d2 and t2.kod='24')
```

Kłopotliwym zagadnieniem może okazać się wyświetlenie wszystkich elementów ścieżki łączącej element bieżący z korzeniem za pomocą jednego zapytania SQL. Jakkolwiek serwery baz danych oferują w tej chwili możliwość wykonywania zapytań rekurencyjnych (m.in. serwery ORACLE, patrz [3] – „SQL Language Reference"), sposób odwzorowania hierarchii pionowej utrudnia lub wręcz uniemożliwia wykorzystanie tego typu zapytań ze względu na odmienną listę warunków dla każdego poziomu. Różnica jest dobrze widoczna przy porównaniu zapytania zwracającego listę województw i listę powiatów.

Chociaż taka konstrukcja słownika ogranicza z góry maksymalną głębokość przechowywanej hierarchii (w tym przypadku 4), to jej zwiększenie jest stosunkowo proste – należy dołożyć kolejną kolumnę D_{n+1} i wypełnić ją zerami dla dotychczas istniejących pozycji, a pozycje nowego poziomu wprowadzać do słownika zgodnie z regułami podanymi na początku tego punktu.

2.4.3. Realizacja struktury hierarchicznej z wykorzystaniem jednej tabeli – rozwiązanie 2

Rozwiązanie podane w punkcie 2.4.3 ma znaczne ograniczenie – zwiększenie głębokości struktury wymaga przebudowania zarówno tabeli po stronie serwera bazy danych, jak i aplikacji klienta. Powyższego ograniczenia nie ma struktura prezentowana w niniejszym punkcie. Schemat struktury przedstawiono na rys. 7. Struktura ta znacznie różni się od struktury opisanej w punkcie 2.4.2. Przyjęto tutaj organizację pionową hierarchii za pomocą paragrafów. Każdy element posiada swój własny, unikalny kod oraz związaną z nim nazwę. Z przyczyn opisanych w punkcie 2.3 dostępne są również pola `UPPER_KOD` i `UPPER_NAZWA` oraz `STATUS`. W celu ułatwienia wizualnego rozróżnienia pozycji w przeglądarce, wprowadzono pole `SHOW_KOD`. Pole to zawiera wartość równą `UPPER_KOD`, jeśli element jest liściem i jest puste w pozostałych przypadkach.

SL_TERYT		
KOD	VARCHAR2(8)	<pk>
UPPER_KOD	VARCHAR2(8)	
NAZWA	VARCHAR2(128)	
UPPER_NAZWA	VARCHAR2(128)	
LVL	NUMBER(1)	
CHAPTER	NUMBER(8)	
ORDR	NUMBER(8)	
SELECTABLE	NUMBER(1)	
STATUS	NUMBER(1)	
NEXT_CHAPTER	NUMBER(8)	
PREV_CODE	VARCHAR2(8)	
SHOW_KOD	VARCHAR2(8)	

Rys. 7. Słownik kodów terytorialnych – rozwiązanie 2
Fig. 7. A hierarchical dictionary of area codes, solution 2

Ponieważ słowniki są z reguły wykorzystywane nie tylko do przechowywania, lecz także do wybierania pozycji, celowe staje się rozróżnienie, czy pozycja jest pozycją wybieralną. Najczęściej „wybieralnymi" pozycjami są liście hierarchii znajdujące się na końcu ścieżki prowadzącej od korzenia, choć zdarzają się przypadki, gdy trzeba umożliwić operatorowi wybór zarówno liści, jak i elementów nie będących elementami terminalnymi (tzw. grup wybieralnych). W celu rozróżnienia statusu „wybieralności" elementu wprowadzono pole SELECTABLE.

Organizacja z wykorzystaniem paragrafów wiąże się z koniecznością zastosowania pięciu pól otrzymujących wzajemne powiązania pomiędzy elementami:

- LVL – przechowuje poziom zagłębienia w hierarchii rozpatrywanego elementu, przy czym element o LVL=1 należy do korzenia słownika,
- CHAPTER – przechowuje bieżący paragraf – wszystkie elementy będące bezpośrednimi potomkami jednego ojca mają identyczną wartość tego pola, tak więc np. w słowniku kodów terytorialnych wszystkie powiaty województwa śląskiego mają identyczną wartość tego pola
- ORDR – ustala porządek w ramach jednego paragrafu, dla każdego paragrafu jest numerowany od 1,
- NEXT_CHAPTER – przechowuje numer paragrafu podrzędnego, czyli definiuje jednoznacznie, która elementy stanowią grupę elementów podrzędnych w stosunku do elementu rozpatrywanego, wartość NULL tego pola oznacza, że element jest liściem (nie ma elementów podrzędnych),
- PREV_CODE – przechowuje wartość kodu „ojca" rozpatrywanego elementu, wartość NULL tego pola oznacza, że element jest korzeniem.

Aby wyjaśnić bliżej zasadę działania słownika tego typu, rozpatrzmy realizację następujących zapytań w słowniku kodów terytorialnych (listy wynikowe zostały zawężone do istotnych pozycji w celu ograniczenia objętości opracowania):

- wyszukanie korzeni, tj. pozycji znajdujących się na pierwszym poziomie w hierarchii:

  ```
  select kod, nazwa, next_chapter from sl_teryt where lvl=1
  ```

  w wyniku wykonania zapytania otrzymamy listę województw składającą się z kodu, nazwy oraz paragrafu, w ramach którego pogrupowane są wszystkie powiaty w poszczególnych województwach:

  ```
  KOD NAZWA     NEXT_CHAPTER
  --- -------- ------------
   .....................
  24  ŚLĄSKIE  263
   .....................
  ```

  Jak widać, paragraf mieszczący wszystkie powiaty województwa śląskiego nosi numer 263,

- znając numer paragrafu, w którym znajdują się interesujące nas pozycje (powiaty woj. śląskiego) konstruujemy zapytanie mające na celu uzyskanie listy tychże powiatów w kolejności ustalonej w słowniku:

  ```
  select kod, nazwa, prev_code from sl_teryt where chapter=263 order by ordr
  ```

  w wyniku jego wykonania otrzymujemy listę powiatów:

  ```
  KOD  NAZWA           PREV_CODE
  ---- ------------- ---------
  2401 BĘDZIŃSKI      24
  2402 BIELSKI        24
  2403 CIESZYŃSKI     24
  2404 CZĘSTOCHOWSKI 24
  2405 GLIWICKI       24
   ..................
  ```

  Jak widać, wszystkie powiaty województwa śląskiego mają pole `PREV_CODE` ustalone na 24, wskazując na swojego „ojca". Należy zauważyć, że identyczny rezultat daje nam wykonanie zapytania `select kod, nazwa, prev_code from sl_teryt where prev_code=24 order by ordr`

Dwa powyższe przykłady ilustrują ideę złączeń w słowniku opisywanego typu. Bardzo istotnym ułatwieniem w przypadku konstrukcji narzędzi operujących na słowniku jest fakt,

że raz przygotowane zapytania (w szczególności parametryzowane klauzule WHERE) nadają się do wykorzystania zawsze, niezależnie od poziomu elementów w hierarchii.

3. Zapytania hierarchiczne w języku SQL

Konstruktorzy serwerów baz danych, wychodząc naprzeciw oczekiwaniom programistów, udostępniają mechanizmy zapytań hierarchicznych SQL (stanowią one rozszerzenie standardu). Funkcje takie oferuje m.in. serwer ORACLE w wersji 8, który wykorzystano do zaprezentowania tego typu zapytań. Wykorzystanie zapytań hierarchicznych znacznie ułatwia rozwiązanie zadań o charakterze wyświetlenia ścieżki wielopoziomowej lub znalezienie wszystkich potomków elementu, nie tylko bezpośrednich.

Poniżej omówiono elementy opisywanego rozszerzenia języka SQL i podano przykłady wykorzystania do konstruowania zapytań operujących na strukturze słownika hierarchicznego opisanej w punkcie 2.4.3 .

Podstawowe rozszerzenia języka SQL wykorzystywane do tworzenia zapytań hierarchicznych to:

- klauzula `WHERE` - ogranicza zakres wyświetlanych danych, operując na wynikowym zestawie danych, uzyskanym po wykonaniu zapytania hierarchicznego, nie modyfikując przy tym warunków złączenia,
- klauzula `START WITH` - nakłada warunek początkowy zapytania ustalając, od jakiej wartości ma rozpocząć się przeszukiwanie hierarchiczne,
- klauzula `CONNECT BY` - ustala sposób połączenia pomiędzy pozycją nadrzędną i podrzędną w hierarchii,
- operator `PRIOR` - określa wiersz ojca.

Należy tutaj pamiętać, że powyższe rozszerzenia nie są związane z żadną strukturą, więc możemy je wykorzystać zarówno do poszukiwania coraz głębiej (wychodząc od korzenia), jak również w górę hierarchii, zaczynając od elementu położonego wewnątrz hierarchii lub na jej końcu. Serwer ma także ograniczenie – konstruowane zapytania muszą dotyczyć jednej tablicy.

Przykładowe zadania do rozwiązania z wykorzystaniem elementów hierarchicznego SQL:

- Wyświetlenie wszystkich powiatów i gmin województwa śląskiego (na wszystkich poziomach poniżej pierwszego) – wykorzystujemy tutaj następujące połączenie – kolumna `NEXT_CHAPTER` ojca musi być równa kolumnie `CHAPTER` potomka, a jako warunek ograniczający przyjmiemy `KOD` pierwszego elementu równy 24 (woj. śląskie, patrz punkt 2.4.3). Stosowne zapytanie wygląda następująco:

```
select level, kod, nazwa from sl_teryt start with kod =
```

```
'24' connect by prior next_chapter = chapter order by
level.
```

Pseudokolumna LEVEL określa poziom zagłębienia w zapytaniu i jest słowem zastrzeżonym dla serwera ORACLE. Należy pamiętać, że poziom zagłębienia zapytania jest liczony niezależnie od logiki konstrukcji słownika (LEVEL=1 oznacza zawsze pierwszy element, który jest wybierany na podstawie warunku START WITH, a poziom największy oznacza element, dla którego nie zachodzi warunek CONNECT BY – jest on ostatni w hierarchii). Wynikiem działania zapytania jest lista powiatów, gmin oraz jeden wiersz o kodzie 24 oznaczający województwo śląskie.

- Aby z powyższego zapytania usunąć gminy, należy wprowadzić dodatkowy warunek w klauzuli CONNECT BY : `connect by prior next_chapter = chapter and level<=2.`
- Aby z rozpatrywanego zapytania usunąć pozycję określającą województwo śląskie i wyświetlić jedynie powiaty i gminy, należy wprowadzić klauzulę WHERE do zapytania: `select level, kod, nazwa from sl_teryt where not kod='24' start with kod = '24' connect by prior next_chapter = chapter order by level.`
- Zagadnienie wyświetlenia ścieżki od ojca do elementu bieżącego w przypadku standardowego języka SQL wymaga skonstruowania programowej pętli i wielokrotnego wywoływania instrukcji SQL czytającej poszczególne poziomy. Wykorzystując funkcje zapytań rekurencyjnych można otrzymać wynik za pomocą jednego zapytania. Spróbujmy dowiedzieć się, w jakim powiecie i województwie znajduje się gmina IMIELIN o kodzie 2414021 :

```
select kod, nazwa from sl_teryt start with kod='2414021'
connect by prior prev_code = kod.
```

W wyniku otrzymamy następującą listę:

```
KOD       NAZWA
========= ================
2414021   IMIELIN
2414      TYSKI
24        ŚLĄSKIE
```

Jak łatwo zauważyć, opisane rozszerzenia znakomicie ułatwiają operacje modyfikowania i usuwania danych wewnątrz struktur o charakterze drzewa. Algorytmy operacji na drzewach można znaleźć m.in. w [1]. Więcej informacji o zapytaniach hierarchicznych oraz ich bieżącym rozwoju można znaleźć w dokumentacji serwerów ORACLE, w szczególności w [3].

4. Zastosowania

Struktury hierarchiczne o charakterze słowników mają ogromne znaczenie w prawie każdym systemie informatycznym, w szczególności w systemach, w których są gromadzone i przetwarzane duże ilości danych, jak np. wojewódzkie czy ogólnopolskie systemy ewidencji (osób, pojazdów itp.). W zależności od wybranej struktury, słowniki ułatwiają pracę zarówno użytkownikom początkującym, którzy w czasie pracy najczęściej posługują się opisami słownymi, jak i bardziej zaawansowanym, którzy w pracy wykorzystują kody. Nie bez znaczenia jest tu odpowiednia konstrukcja interfejsu użytkownika.

Struktury hierarchiczne wspierają ograniczenie ruchu w sieci pomiędzy aplikacjami klienta oraz serwerami baz danych, przyspieszając pracę systemu.

Przygotowanie wygodnego narzędzia do przeglądania i wybierania pozycji ze słowników stanowi jeden z kluczowych elementów w pracy nad aplikacją. Dobrze przygotowany obiekt pozwoli na stosunkowo proste rozszerzenie standardowej struktury, zarówno o charakterze liniowym (zaprezentowanej w punkcie 2.3), jak i o charakterze hierarchicznym (opisanej w punkcie 2.4.3). Należy tutaj zwrócić uwagę na fakt, iż struktura hierarchiczna z punktu 2.4.3 stanowi rozszerzenie, w sensie listy pól, struktury liniowej z punktu 2.3, tak więc narzędzie przeglądające strukturę liniową bez żadnych modyfikacji pozwoli na przeglądanie struktury hierarchicznej w trybie liniowym.

LITERATURA

1. Wirth N.: Algorytmy + struktury danych = programy, WNT, Warszawa 2001.
2. Harel D.: Rzecz o istocie informatyki Algorytmika, WNT, Warszawa 2000.
3. Oracle Corp.: Documentation Library for Oracle8 Enterprise Edition release 8.0.5, Redwood Shores, USA 1998.
4. Ullman J.D., Widom J.: Podstawowy wykład z systemów baz danych, WNT, Warszawa 2000.
5. Pankowski T.: Podstawy baz danych, PWN, Warszawa 1992.
6. Silberschatz A., Korth H. F., Sudarschan S.: Database System Concepts, McGraw-Hill Companies 1996.

Recenzent: Dr inż. Stanisław Jędruś

Wpłynęło do Redakcji 15 marca 2001 r.

Abstract

The article covers selected problems involved with hierarchical dictionaries implementation.

Chapter 1 contains an introduction to the dictionaries idea as easy and useful software solution for key – name pairs table where these pairs are organized in any way. It also contains basic information about linear vs. hierarchical dictionary.

Chapter 2 covers evolution process starting from simple table with descriptive field, through the simple code – name pairs linear dictionary, to the hierarchical dictionaries, various implemented. This coverage contains structure definition and organization (single- & multi-table) with exact fields definition and relation description between records. All descriptions are given by example, mostly using hierarchical dictionary of area codes (in various implementations, suitable for considered one), with internal structure organized as on fig. 6. Structures described in the article are: basic table with descriptive field – fig. 1., simple key – name external dictionary without additional fields – fig. 2., universal linear dictionary – fig.3., multi-table hierarchical dictionary implementation – fig. 4., single-table hierarchical dictionary implementation in two variants (limited level structure and unlimited level structure) – fig. 5. and 7. This chapter also contains some algorithmic considerations on DQL commands construction in SQL.

Chapter 3 consists of hierarchical queries SQL language extension (brief reference) and takes into consideration this extension as useful tool helping in queries construction for hierarchical browsing. Extension reference is based on ORACLE database server DQL reference. This chapter also contains examples of hierarchical queries that select data from the dictionary of area codes, i.e. query operating on a range of levels, selecting all records that belong to the path coming from the root to the element or query operating on the sub-tree of considered record.

Many of described structures and methods are part of Medical Services Registering System and software solutions for National Health Services.