Piotr Wanat, Adam Pelikant
Wyższa Szkoła Informatyki, Katedra Inżynierskich
Zastosowań Informatyki, 93-008 Łódź, ul Rzgowska 17a
email: Piotr.Wanat@kir.com.pl,
apelikan@wsinf.edu.pl

OBSŁUGA ACYKLICZNYCH GRAFÓW SKIEROWANYCH (DRZEW) Z POZIOMU JĘZYKA ZAPYTAŃ SQL

Streszczenie – Artykuł porusza niektóre aspekty obsługi grafów z poziomu języka zapytań SQL. Przedstawione zostały metody ich reprezentacji w modelu relacyjnym oraz najbardziej reprezentatywne metody obsługi. Istotą artykułu jest polemika z dwoma algorytmami zaproponowanymi przez J. Celko [1].

1 Wstęp

O potrzebie gromadzenia i obróbki danych we współczesnym świecie nikogo nie trzeba przekonywać. Od początku lat 70 ubiegłego wieku dominują relacyjne bazy danych [2]. Jednak rewolucja wprowadzona przez Codd'a przyniosła nie tylko postęp ale również problemy z reprezentacją pewnych typów danych. Wiele problemów dostarcza przechowywanie, a zwłaszcza obróbka danych powiązanych ze sobą w sposób hierarchiczny czy też w postaci sieci. W praktyce, takie struktury danych są bardzo często spotykane np.: opis struktury podległości czy zarządzania w firmie, opis przepływu informacji czy też połączeń komunikacyjnych różnych typów. Drugą przesłanką podjęcia tematyki było przekonanie o wyższości stosowania rozwiązań z tzw. „cienkim klientem" (thin client), które z powrotem zyskuje wielu zwolenników. Sprowadza się to do próby sprowadzenia przetwarzania na stronę serwera czyli programowania na możliwie podstawowym poziomie. W przypadku baz danych stosowania SQL lub jego rozszerzeń strukturalnych.

Strukturalny Język Zapytań SQL (Structured Query Language) jest językiem obsługi i komunikacji z bazami danych. Ma za zadanie umożliwić użytkownikowi prostą i skuteczną metodę odczytywania i zapisywania danych z/do baz danych. SQL. Prawie wszystkie popularne

systemy są w dużej mierze zgodne przynajmniej z podstawowym zakresem standardu SQL'92.

Funkcjonowanie współczesnych systemów relacyjnych baz danych oparte jest głównie na tym języku. SQL został pierwotnie zaprojektowany jako język do formułowania zapytań, oparty na rachunku relacyjnym. Obecnie jest on jednak uniwersalnym interfejsem do większości systemów zarządzania bazami danych, tj. wszelkie operacje dotyczące definicji danych, dostępu do danych i ich modyfikacji, jak również czynności administracyjne, odbywają się poprzez komendy i programy zapisane w SQL. Niestety prawie wszyscy producenci systemów bazodanowych stosują własne rozszerzenia standardu ANSI, co oczywiście ułatwia pracę na danej platformie ale w znacznej mierze utrudnia tworzenie rozwiązań uniwersalnych, międzyplatformowych. Dodatkowo korzystanie z rozszerzeń proceduralnych wiąże się ze stosowaniem rozwiązań opartych o kursory, co w znacznym stopniu pogarsza wydajność przetwarzania przez wzrost obciążenia zasobów i spowolnienie przetwarzania. Dlatego tam gdzie tylko to jest możliwe należy dążyć do stosowania rozwiązań opartych o podstawowy standard ANSI. Niestety w przypadku grafów cyklicznych stosowanie kursorów jest konieczne. Stąd artykuł dotyczy tylko i wyłącznie grafów acyklicznych (drzew), w przypadku których, do rozwiązywania w zasadzie wszystkich problemów, możliwe jest zastosowanie tylko poleceń podstawowego zakresu SQL.

Niniejszy artykuł, nie jest jednak kolejną instrukcją stosowania języka i podstawowych poleceń składni, lecz traktuje o zastosowaniu języka SQL w implementacji do **drzew czyli skierowanych grafów acyklicznych**. Jak więc z tego wynika artykuł ten jest specjalizacją zaawansowanego programowania SQL. Nie omawia on także wszystkich zagadnień związanych z w/w tematem z uwagi na jego wielkość, a skupia się jedynie na najciekawszych rozwiązaniach będących polemiką z rozwiązaniami przedstawionymi prze J. Celko [1].

2 Definicje podstawowe

Grafy są to struktury danych dostarczające ogólnej metody reprezentacji wielu typów danych i ich zależności. Grafy utworzone są z wierzchołków połączonych krawędziami. Wierzchołki zwyczajowo przedstawiane są jako prostokąty, a krawędzie jako strzałki. Każda krawędź reprezentuje jednokierunkową zależność, jaka występuje pomiędzy wierzchołkami połączonymi tą krawędzią. Zbiór wierzchołków i krawędzi nazywa się drzewem, będącym specjalnym rodzajem grafu skierowanego.

Krawędzie w grafie mogą być skierowane, czyli umożliwiać przemieszczanie się tylko w jedną stronę lub nieskierowane - umożliwiające

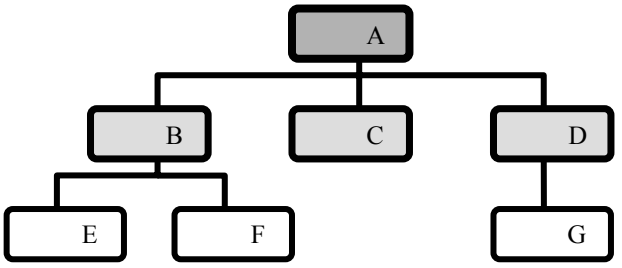
podróż w obie strony. Grafy klasyfikuje się jako rekonwergentne i nierekorwergentne. Grafy rekorwergentne w odróżnieniu od tych drugich dostarczają wielu dróg pomiędzy parą wierzchołków.

Drzewa są grafami skierowanymi i nierekorwergentnymi, czyli umożliwiają podróż tylko w jedną stronę oraz pomiędzy parą wierzchołków istnieje tylko jedna droga.

2.1 Metody reprezentacji Drzew

2.1.1 Schemat

Korzeń drzewa uwidoczniony jest na górze schematu, a wierzchołki liści na dole. Wierzchołki reprezentowane są przez prostokąty, a krawędzie poprzez linie je łączące. Droga przebiega z góry na dół, czyli od korzenia do liści. Metoda ta stosowana jest przy posługiwaniu się modelem krawędziowej reprezentacji drzew.



A

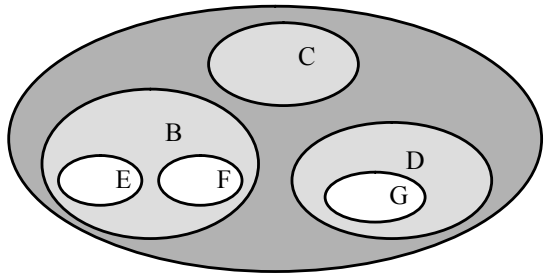
Klucz	Wartość	Nadrzędne
1	A	<NULL>
2	B	1
3	C	1
4	E	2
5	F	2
6	G	4

B

Rys. 1. Krawędziowa metoda reprezantacji drzew – A graf, B reprezentacja relacyjna

2.1.2 Zbiory zagnieżdżone

Wierzchołki w tej metodzie przedstawione są jako zbiory w formie owali. Korzeń drzewa to owal, który zawiera w sobie wszystkie inne wierzchołki drzewa. Liść to owal, który nie zawiera w sobie żadnego innego zbioru. Krawędzie w metodzie zbiorów zagnieżdżonych nie są reprezentowane w formie graficznej, lecz są ilustrowane przez zawieranie się zbiorów. Metoda ta stosowana jest wtedy, gdy chcemy posłużyć się modelem odwiedzinowym.



A

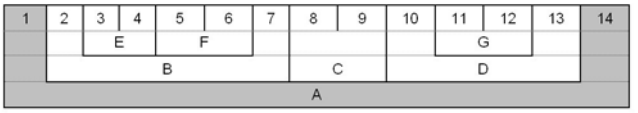
Klucz	Wartość	Prawo	Lewo
1	A	1	14
2	B	2	7
3	E	3	4
4	F	5	6
5	C	8	9
6	D	10	13
7	G	11	12

B

Rys. 2. Odwiedzinowa metoda reprezentacji drzew – A zbiory zagnieżdżone, B reprezentacja relacyjna

Niestety SQL posiada bardzo ubogie narzędzia obsługi danych hierarchicznych. Nie odwzorowuje bezpośrednio tych danych na tabele, ponieważ podstawą tabel są zbiory, a nie grafy. SQL nie obsługuje bezpośrednio ani wyszukiwania surowych danych w sensowny, rekurencyjny lub hierarchiczny sposób, ani też nie wykonuje funkcji zdefiniowanych rekurencyjnie, które występują powszechnie w aplikacjach tego typu. Wielu programistów używa nadrzędnych języków programowania poziomu 3GL bądź programów generujących raporty, które potrafią lepiej obsługiwać struktury drzewiaste. Tylko nieliczne

rozszerzenia np. SQL plus (ORACLE) dostarczają poleceń do bezpośredniego wyświetlania struktur hierarchicznych (CONNECT BY)



Rys. 3. Schemat obliczania wartości kolumn graniczeń PRAWO i LEWO

Nasuwa się w tym miejscu pytanie, dlaczego używać do tego celu właśnie SQL, kiedy istnieją inne, wygodne narzędzia. Odpowiedzią jest uniwersalność i przenośność języka SQL, niezależnie od platformy systemowej lub bazodanowej.

Za pomocą SQL można dokonać między innymi następujących działań:

znajdowanie wierzchołków liści,
znajdowanie wierzchołka korzenia,
wyszczególnienie dróg wewnętrznych,
znajdowanie poziomów drzewa,
znajdowanie wysokości drzewa,
znajdowanie poddrzew,
wstawianie i usuwanie poddrzewa.

Metody zaprezentowane w niniejszym artykule pokazują różnice pomiędzy zaproponowanymi przez autorów rozwiązaniami, a postulowanymi przez J. Celko [1]. Przedstawmy dwie z metod, gdzie różnice te są najbardziej widoczne.

3 Operacja wstawiania poddrzewa w modelu odwiedzinowym

Operacja ta nie jest zadaniem prostym, tak jak w przypadku reprezentacji krawędziowej. Operacja ta zostanie zaprezentowana na przykładzie dodania jednego wierzchołka do istniejącego drzewa, przy czym wierzchołek ten nie musi być liściem.

W pierwszym kroku określamy miejsce w drzewie gdzie chcemy wstawić nowy wierzchołek i wstawiamy nowy wierzchołek:

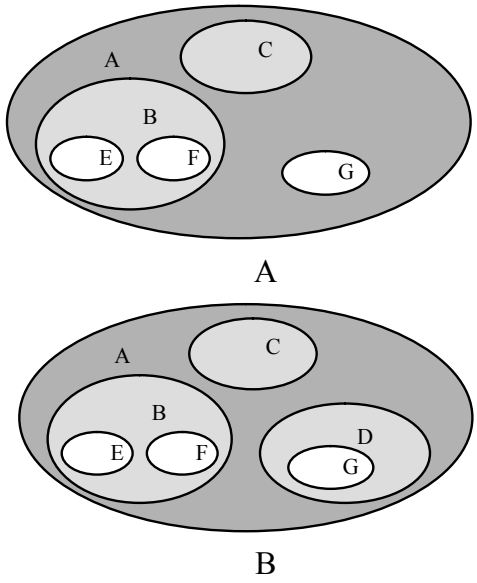
```
INSERT INTO KadryB (Pracownik, Przełożony, Płaca, Lewo, Prawo)
Values( :nowy_wierzchołek , :wierzchołek_nadrzędny , 15, 0, 0);
```

Następnym krokiem jest rozstawienie wierzchołków znajdujących się na prawo od wierzchołka wybranego w pierwszym kroku, a także zmiana reprezentacji prawostronnej tego wierzchołka. Za pomocą następującego zapytania wykonujemy przesunięcie numerów odwiedzin

prawostronnych o 2 pozycje, ponieważ dodajemy tylko jeden wierzchołek:

```
UPDATE KadryB SET prawo=prawo+2
WHERE prawo>=
        (SELECT prawo FROM KadryB
                WHERE pracownik= :wierzchołek_nadrzędny );
```

gdzie operator ***prawo>=*** zapewnia nam zmianę reprezentacji prawostronnej wybranego wierzchołka.



Rys. 4. Dodanie węzła do istniejącego drzewa: A stan przed wstawieniem, B po wstawieniu węzła

Teraz konieczne jest przesunięcie reprezentacji lewostronnej wierzchołków znajdujących się na prawo od wybranego wierzchołka. Numer lewostronnego ograniczenia wierzchołka nie będzie zmieniany, z uwagi na fakt, że jest on rodzicem dodawanego wierzchołka:

```
UPDATE KadryB SET lewo=lewo+2
WHERE lewo>
        (SELECT prawo FROM KadryB
                WHERE pracownik= :wierzchołek_nadrzędny );
```

Ostatnim etapem jest zmiana numerów odwiedzin wstawionego wierzchołka, tak aby pasowałyby one do reszty drzewa:

```
UPDATE KadryB SET lewo=
(SELECT prawo-2 FROM KadryB WHERE pracownik=
:wierzchołek_nadrzędny ) WHERE pracownik= :nowy_wierzchołek ;
```

oraz

```
UPDATE KadryB SET prawo=
(SELECT prawo-1 FROM KadryB WHERE pracownik=
:wierzchołek_nadrzędny )
WHERE pracownik= :nowy_wierzchołek ;
```

Poniżej zostanie zaprezentowana metoda zaprezentowana przez J. Celko [1]:

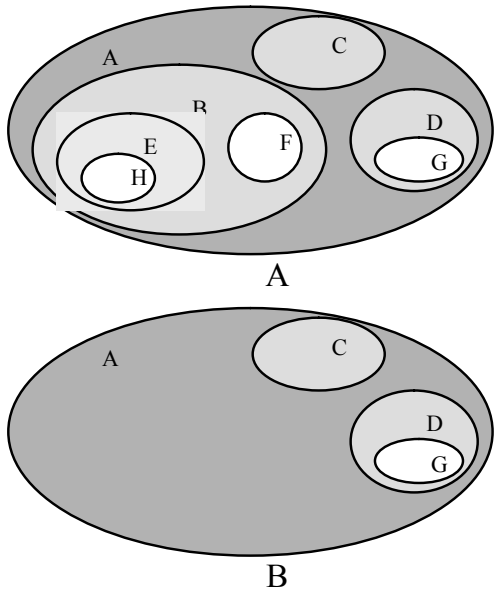
```
UPDATE KadryB SET prawo=prawo+2
WHERE prawo>=
        (SELECT prawo FROM KadryB
                WHERE pracownik= :wierzchołek_nadrzędny );
UPDATE KadryB SET lewo=lewo+2
WHERE lewo>
        (SELECT lewo FROM KadryB
                WHERE pracownik= :wierzchołek_nadrzędny );
UPDATE KadryB SET lewo=
(SELECT lewo+1 FROM KadryB WHERE pracownik=
:wierzchołek_nadrzędny )
WHERE pracownik= :nowy_wierzchołek ;
UPDATE KadryB SET prawo=
(SELECT prawo-1 FROM KadryB WHERE pracownik=
:wierzchołek_nadrzędny)
WHERE pracownik= :nowy_wierzchołek ;
```

Różnice w stosunku do metody zaproponowanej przez autorów oraz we wspomnianym opracowaniu zaznaczone są poprzez podkreślenie. Składnia powyższa [1] działa poprawnie, gdy dodajemy nowy wierzchołek do wierzchołka liścia. Gdy spróbujemy dodać do wierzchołka będącego rodzicem, to błędnie zmienią się numery odwiedzin wierzchołków potomków oraz dodanego wierzchołka. Aby metoda działała poprawnie należy zmienić 2 i 3 zapytanie.

W drugim zapytaniu zmienione zostało ***Lewo*** na ***Prawo*** w podzapytaniu SELECT aby zmianie nie uległy numery lewostronne wierzchołków będących potomkiem *:wierzchołka nadrzędnego*. W trzecim zapytaniu zmienione zostało ***lewo+1*** na ***prawo-2*** w SELECT aby zmienić numer lewostronny dodanego wierzchołka licząc od numeru prawostronnego *:wierzchołka nadrzędnego* a nie od lewostronnego jak było w błędnym zapytaniu. Metoda zaproponowana przez autorów działa poprawnie bez względu na miejsce wstawiania nowego wierzchołka.

4 Usuwanie poddrzewa w modelu odwiedzinowym

Zadanie to jest operacją jeszcze bardziej skomplikowaną niż dodawanie poddrzewa, ponieważ po jego usunięciu w tabeli powstaje luka w numeracji odwiedzin. Metoda usuwania poddrzew bazuje na zasadzie, że numer odwiedzin prawostronnych wierzchołka korzenia jest większy od numeru wierzchołka potomnego, natomiast numer odwiedzin lewostronnych korzenia jest mniejszy od lewostronnego numeru odwiedzin wierzchołka potomka.



Rys. 5. Usunięcie poddrzewa z istniejącego drzewa: A stan przed usunięciem, B po usunięciu poddrzewa

Dzięki takiemu założeniu można zastosować predykat BETWEEN i jako parametr *:usuwane_poddrzewo* podać wierzchołek korzenia, którego poddrzewo chcemy usunąć.

Pierwszym krokiem będzie usunięcie poddrzewa określonego korzenia:

```
DELETE FROM KadryB
WHERE lewo BETWEEN
            (SELECT lewo FROM KadryB
             WHERE pracownik = :usuwane_poddrzewo)
        AND
            (SELECT prawo FROM KadryB
             WHERE pracownik = :usuwane_poddrzewo);
```

Po usunięciu poddrzewa w drzewie powstała luka w ciągłości numerów lewo- i prawostronnych wierzchołków. Luka ta musi zostać zamknięta. Tworzony jest pomocniczy widok *Spłaszczone_Drzewo* na tabeli *KadryB* z listą numerów zarówno lewo- jak i prawostronnych:

```
CREATE VIEW Spłaszczone_Drzewo(odwiedziny) AS
SELECT lewo FROM KadryB
UNION
SELECT prawo FROM KadryB;
```

Na podstawie widoku *Spłaszczone_Drzewo* tworzone są dwa widoki *Pierwsze_Odwiedziny* i *Ostatnie_Odwiedziny*, znajdujące luki w numerach odwiedzin:

- lewostronnych

```
CREATE VIEW Pierwsze_Odwiedziny (odwiedziny) AS
SELECT (odwiedziny+1) FROM Spłaszczone_Drzewo
WHERE
(odwiedziny+1) NOT IN (SELECT odwiedziny FROM
Spłaszczone_Drzewo)
AND
(odwiedziny+1) < (SELECT MAX(odwiedziny) FROM
Spłaszczone_Drzewo);
```

- prawostronnych

```
CREATE VIEW Ostatnie_Odwiedziny(odwiedziny) AS
SELECT (odwiedziny-1) FROM Spłaszczone_Drzewo
WHERE
(odwiedziny-1) NOT IN (SELECT odwiedziny FROM
Spłaszczone_Drzewo)
 AND (odwiedziny-1) > 0;
```

gdzie zastosowanie ostatnich warunków:

```
(odwiedziny+1) < (SELECT MAX(odwiedziny) FROM
Spłaszczone_Drzewo)
```

i

```
(odwiedziny-1) > 0
```

ma na celu zabezpieczenie przed wyjściem poza skrajne ograniczenia numerów odwiedzin.

Następnie wykorzystywane są perspektywy *Pierwsze-* i *Ostatnie_Odwiedziny* do utworzenia tabeli *Luki,* zawierającej wyszczególnione luki jakie muszą być zamknięte. Widok ten wskaże także liczbę początkową i końcową luk w numeracji odwiedzin oraz ich rozmiary:

```
CREATE TABLE Luki (początek INT, koniec INT, wielkość INT);
INSERT INTO Luki
SELECT t1.odwiedziny, t2.odwiedziny, ((t2.odwiedziny-t1.odwiedziny)+1)
FROM Pierwsze_Odwiedziny t1, Ostatnie_Odwiedziny t2
WHERE t2.odwiedziny =
        (SELECT MIN(t3.odwiedziny) FROM Ostatnie_Odwiedziny t3
        WHERE t1.odwiedziny <=t3.odwiedziny);
```

J. Celko [1] proponuje zamiast tabeli utworzenie widoku. Przy zastosowaniu widoku, dla każdej przerwy w numeracji zarejestrowanej w widoku *Luki* wymaga wykonania zapisanych poniżej zapytań likwidujących te nieciągłości:

```
WHILE EXISTS (SELECT * FROM Luki)
DO BEGIN
    UPDATE KadryB SET prawo = prawo-(SELECT wielkość FROM Luki)
    WHERE prawo > (SELECT MIN(początek) FROM Luki);
    UPDATE KadryB SET lewo = lewo-(SELECT wielkość FROM Luki)
    WHERE lewo > (SELECT MIN(początek) FROM Luki);
END;
```

spowoduje błędne działanie algorytmu, ponieważ widok *Luki* w pewnych okolicznościach automatycznie zostanie zmieniony (zaktualizowany) i polecenie zakończy się błędem. Jeśli jednak zastosowana zostanie tabela pomocnicza (może być tymczasowa), to wartości wpisane do niej będą niezmienne przy obu operacjach UPDATE. Końcową operacją przy usuwaniu poddrzewa będzie wykonanie obu w/w operacji UPDATE na **tabeli** *Luki* dla każdej luki w niej zarejestrowanej.

5 Podsumowanie

Artykuł przedstawia tylko dwa z wielu opracowanych algorytmów. Wybór ich był podyktowany przede wszystkim ich reprezentatywnością oraz dość dużym stopniem złożoności. Drugim aspektem była chęć przedstawienia konkurencyjnych rozwiązań względem zaprezentowanych przez J. Celko. Dodatkowo praca przedstawia dwie metody reprezentacji grafów w modelu relacyjnym.

Popularną metodę krawędziową oraz mniej znaną metodę odwiedzinową. Metoda krawędziowa (przy odpowiedniej modyfikacji) może być stosowana do wszystkich typów grafów, natomiast druga z nich jest dostosowana tylko do reprezentacji drzew. Metoda odwiedzinowa oferuje większą elastyczność reprezentacji drzew jednak wymaga większego wysiłku przy tworzeniu algorytmów przetwarzania. Pokazane algorytmy wskazują również na bardzo duże możliwości języka zapytań SQL, który bez stosowania rozszerzeń proceduralnych pozwala na rozwiązywanie

bardzo szerokiej klasy problemów powyżej zakresu przetwarzania danych.

Literatura

[1] Celko J.: SQL – Zaawansowane Techniki Programowania, MIKOM, Wrszawa 1999 r.

[2] Codde E. F.: The relational model for database management, MA: Addison –Wesley 1990 r.

[3] Date C. J.: Relational database: writings 1989-1991, MA: Addison –Wesley 1992 r.

[4] Wanat P.: Drzewa w SQL w przykładach – narzędzie dydaktyczne, Wyższa Szkoła Informatyki, Łódź 2004.

ACYCLICAL GRAPHS (TREES) HANDLING FROM THE LEVEL OF STRUCTURE QUERY LANGUAGE SQL

Summary – The paper concern some aspects of acyclical graphs handling due to structure query language. Methods of their representations in relational model have been presented as well as the most representative methods of their handling. The polemic with two algorithms presented by Joe Celko is the essence of this paper.