AUTOMATYKA • 2005 • Tom 9 • Zeszyt 3

Jacek Wiślicki*, Kamil Kuliberda*

Metabaza dla optymalizacji kosztowej zapytań w obiektowych bazach danych opartych o SBA

1. Wprowadzenie

Techniki optymalizacji zapytań są jednym z głównych aspektów stanowiących o możliwości wykorzystania bazy danych w zastosowaniach praktycznych. Czas ewaluacji zapytania („szybkość") jest podstawową cechą pozwalającą ocenić przeciętnemu użytkownikowi przydatność konkretnego silnika bazy danych do danego zadania, ponieważ faktycznie decyduje o efektywności całego systemu informatycznego opartego na bazie danych.

Niniejszy artykuł koncentruje się na optymalizacji zapytań obiektowych baz danych wywodzących się z koncepcji podejścia stosowego, a w szczególności dotyczy prowadzonych przez autorów prac związanych z technikami optymalizacji kosztowej. Główny nacisk został położony na architekturę metabazy (metamodelu) wykorzystywanej przez projektowany optymalizator dynamiczny.

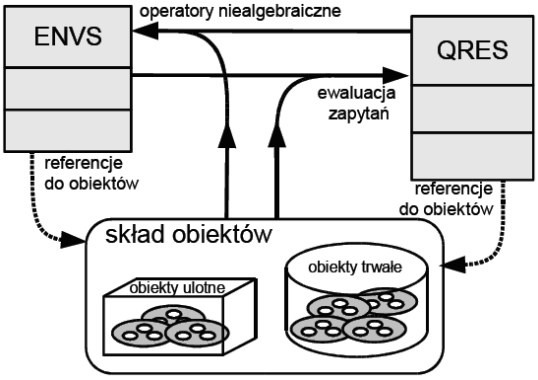
Kolejne podrozdziały zawierają wprowadzenie do podejścia stosowego i podsumowanie technik i rozwiązań optymalizacyjnych mających w tym przypadku zastosowanie. Rozdział 2 opisuje metabazę stworzoną na potrzeby modelu kosztów wchodzącego w skład optymalizatora, natomiast rozdział 3 zawiera krótkie podsumowanie oraz zarys kolejnych etapów prowadzonych prac.

1.1. Podejście stosowe

Podejście stosowe jest względnie nową koncepcją wprowadzającą do obiektowych języków zapytań i mechanizmów realizacji samych zapytań znane z języków programowania rozwiązanie stosu środowisk [1, 2]. Podczas przetwarzania zapytań SBA (*Stack-Based Approach*) stosuje stos środowisk ENVS (*Environment Stack*) oraz stos rezultatów QRES (*Query Result*). Stos ENVS służy do właściwej ewaluacji zapytań, podczas gdy QRES przechowuje ich wyniki (w tym wyniki tymczasowe, wynikające z podzapytań) oraz zmienne pomocnicze, np. iteratory pętli wykorzystanych w zapytaniach. Obydwa stosy zawierają odwołania (poprzez bindery) zarówno do obiektów trwałych („zwykłe" obiekty przecho-

* Katedra Informatyki Stosowanej, Politechnika Łódzka

wywane w bazie danych), jak i obiektów ulotnych, których czas życia ogranicza się do określonego etapu procesu wykonywania zapytania lub będących rezultatem innych operacji związanych z bazą danych (rys. 1).



Rys. 1. Ewaluacja zapytań w obiektowych bazach danych opartych o podejście stosowe z wykorzystaniem stosów ENVS i QRES [1]

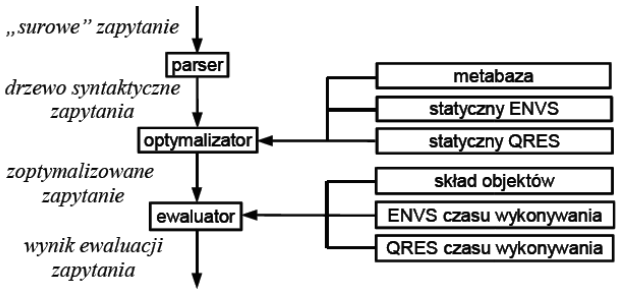
1.2. Optymalizacja zapytań dla SBQL (*Stack-Based Query Language*)

Prowadzone obecnie prace skupiają się na opracowaniu efektywnego dynamicznego optymalizatora zapytań w oparciu o istniejący i zaimplementowany optymalizator statyczny i sprawdzone w nim reguły statycznej optymalizacji (schematycznie pokazany na rys. 2) [3–8].

Dynamiczna optymalizacja zapytań oparta o model kosztów [9–11] wymaga uwzględnienia większej liczby czynników mogących mieć wpływ na efektywność wykonania zapytania (związanych bezpośrednio z bieżącym stanem bazy danych) niż w przypadku procesu statycznego, polegającego głównie na syntaktycznych transformacjach zapytania do semantycznie równoważnej postaci w oparciu o informacje o istnieniu obiektów i ich zależnościach.

Kolejną niezwykle istotną różnicę pomiędzy optymalizacją statyczną a dynamiczną stanowi fakt, że w przypadku pierwszej z nich zmodyfikowane zapytanie jest zawsze efektywniejsze, a na pewno nie mniej efektywne, niż zapytanie „surowe" – wprowadzone przez klienta bazy danych (aplikację czy użytkownika), przy oczywistym założeniu, że optymalizator wykonuje spójne i konsekwentne przekształcenia, tj. zastosowane w nim algorytmy nie zawierają błędów. W przypadku optymalizacji dynamicznej (kosztowej), przekształcone zapytanie może okazać się mniej efektywne od zapytania „surowego", a szczególnie względem zapytania zoptymalizowanego czysto statycznie. Zjawisko to jest spowodowane podstawowymi zasadami działania optymalizatora kosztowego, który wybiera **najbardziej korzystny** (najmniej zużywający czas i zasoby) plan wykonania zapytania (w praktyce, ze względu na konieczność maksymalnego skrócenia i uproszczenia procesu, aby plan nie wpływał negatywnie na wykonanie zapytania, a jedynie w sposób

oczywisty, najgorsze plany wykonania zostają odrzucone i wybierany jest jeden z pozostałych). Niestety, istnieje możliwość, że „najlepszy" z planów nie okaże się najprostszym i najbardziej efektywnym sposobem ewaluacji zapytania, jeżeli stan bazy nie zostanie w prawidłowy sposób zinterpretowany.



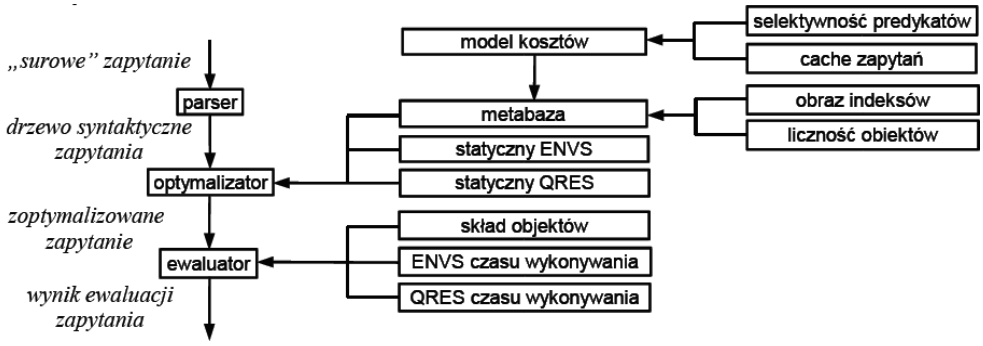
Rys. 2. Statyczny optymalizator zapytań dla SBQL [3]

Powyższa sytuacja związana jest z faktem, że procedury optymalizacji statycznej są zdecydowanie deterministyczne (szczególnie w przypadku semantycznej i syntaktycznej jednorodności i spójności SBQL), oparte na jednoznacznych i jasno określonych regułach, natomiast analiza i transformacje wchodzące w skład optymalizacji dynamicznej są w dużej mierze heurystyczne, związane z empirycznym doborem parametrów optymalizatora. Z tego powodu główną zasadą optymalizatora kosztowego jest, aby **nie pogarszał** wykonania statycznie zoptymalizowanych zapytań. W konsekwencji optymalizator dynamiczny (w szczególności jego rdzeń – **model kosztów**) powinien być angażowany w proces optymalizacji opcjonalnie, nie jako jego nieodłączny element, i powinien być traktowany jako uzupełnienie optymalizacji statycznej stosowane w sytuacjach, kiedy z dużym prawdopodobieństwem wpłynie korzystnie na całość procesu optymalizacji, np. w oczywistym przypadku syntaktycznie równoważnych podzapytań, gdzie kolejność wykonywania może wpłynąć na wykonanie całego zapytania (zamiana kolejności złączeń), lub kiedy istnieje możliwość wyboru jednego spośród dostępnych indeksów.

Ponieważ liczba parametrów stanu bazy danych wpływających na czas i konsumpcję zasobów potrzebnych do ewaluacji zapytania jest trudna do oszacowania, a sam proces optymalizacji powinien być maksymalnie uproszczony ze względu na czas i obciążenie systemu, na obecnym etapie rozwoju optymalizatora dynamicznego uwzględnia się następujące (rys. 3) parametry:

- **liczność obiektów** (encji) w bazie danych (składzie obiektów), brany pod uwagę jest głównie rząd wielkości;
- **selektywność predykatów**, (związana z rozkładem obiektów) zgodnie z koncepcją wprowadzoną w [12] i kontynuowaną w [3] dla uproszczenia obliczeń zredukowana do informacji o „maksymalnej" i „minimalnej" wartości w danym rozkładzie i liczności obiektów, jakkolwiek w dalszych pracach rozważane jest również zastosowanie rzeczywistych rozkładów i bardziej skomplikowanej analizy związanej z zastosowaniem rachunku prawdopodobieństwa;

– **obraz indeksów**, informacja o istnieniu indeksów i ich wydajności (szybkość działania i selektywność);
– **cache zapytań**, istnienie zapamiętanych wyników zapytań (ewaluowanych uprzednio), które mogą zostać zastosowane podczas wykonania bieżącego zapytania



Rys. 3. Dynamiczny optymalizator zapytań dla SBQL jako rozbudowany optymalizator statyczny [9]

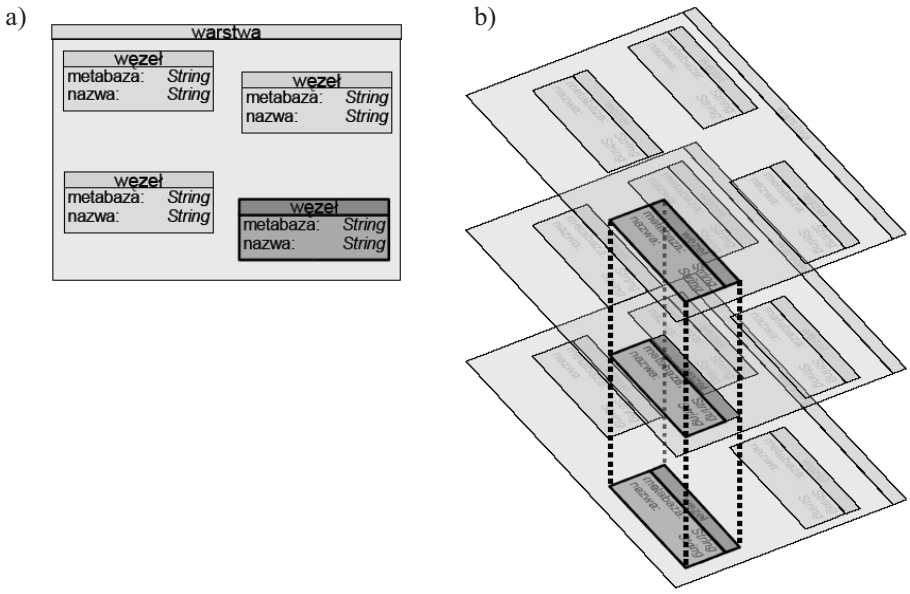
Wykorzystanie ww. parametrów w optymalizacji dynamicznej i ich interpretacja zostały opisane w [11].

2. Budowa metabazy

Na obecnym etapie rozwoju zakłada się oczywiste zastosowanie metabazy podczas semantycznej analizy zapytań (walidacji) oraz podczas procesów optymalizacyjnych, co niejako „ogranicza" jej dostępność, jako elementu systemu bazodanowego, jedynie do wnętrza tego systemu. W dalszej fazie metabaza najprawdopodobniej zostanie wyposażona w funkcjonalność komunikacji z systemami zewnętrznymi, np. IDE (*Integrated Development Environment*) czy SCM (*Software Configuration Management*) [10, 11, 13].

Bez względu na zakres zastosowań (w chwili obecnej pod uwagę brane są jedynie funkcje wspomniane powyżej), uzasadnioną koniecznością wydaje się zachowanie maksymalnej prostoty, przejrzystości i elastyczności metabazy – ma ona ewidentnie utylitarne zastosowania i powinna być możliwie **najłatwiejsza** do analizy, zrozumienia i wykorzystania przez użytkowników (programistów) [13, 14]. Celowe jest zerwanie z konwencją wprowadzoną w standardzie ODMG (*Object Database Management Group*) [15], w opinii autorów czyniącą z metabazy twór skomplikowany, nadmiernie rozbudowany, o znacznie ograniczonych możliwościach praktycznego wykorzystania i utrzymania (szersze odniesienie się do metamodelu specyfikacji ODMG znajduje się w [13, 14]).

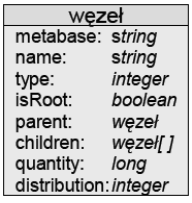
Zgodnie z opisanym powyżej zakresem wykorzystania metabazy i wymaganiami modelu kosztów, proponowana metabaza oparta jest o strukturę warstwową (rys. 4). Pojęcie **warstwy** nie definiuje tutaj oddzielnego tworu fizycznego, lecz logiczne cechy **węzła** (obiektu bazy danych), niezależne od siebie.



Rys. 4. Warstwowa struktura metabazy dla potrzeb modelu kosztów w dynamicznej optymalizacji zapytań dla SBQL: a) pojedyncza warstwa; b) zależność węzłów w poszczególnych warstwach metabazy

Każdy węzeł posiada unikalną w ramach metabazy **nazwę** oraz informację o **typie** obiektu, który reprezentuje, np. obiekt atomowy (typu string, integer, boolean itp.), obiekt złożony czy procedurę. Ponadto dla każdego węzła zdefiniowana jest wartość logiczna, określająca, czy jest to **obiekt korzeniowy** (taki, który jest bezpośrednio dostępny w składzie obiektów). W przypadku węzłów złożonych, zdefiniowana jest także właściwość (realizowana jako tablica) zawierająca nazwy węzłów reprezentujących jego obiekty wewnętrzne oraz węzeł reprezentujący obiekt złożony zawierający w sobie lokalny węzeł (rzeczywisty obiekt, który definiuje w metabazie). W ten sposób uzyskana zostaje bardzo łatwa do zrealizowania obukierunkowa nawigacja rodzic-dziecko (realizująca rzeczywistą strukturę encji istniejących w składzie obiektów), przy zachowaniu wszystkich węzłów (niezależnie od poziomu zagnieżdżenia odpowiadających im obiektów) na semantycznie równoważnym poziomie.

Struktura węzła została schematycznie zaprezentowana na rysunku 5.



Rys. 5. Struktura węzła w ramach metabazy

Cechy węzła definiowane przez poszczególne warstwy uwzględnione podczas dotychczasowych prac nad metabazą stanowią:

- **indeksy** związane z węzłem;
- **pointery** wskazujące na inne obiekty (związane bezpośrednio z nawigacją i zależnościami logicznymi między rzeczywistymi obiektami bazy danych);
- **cache** zawierający informacje o zapamiętanych zapytaniach związanych z węzłem.

Potencjalnie, w dalszych pracach nad metabazą, w formie warstw mogą zostać zaprezentowane także perspektywy, dziedziczenie, role oraz pozostałe cechy i współzależności obiektów, ich brak nie wpływa jednak w chwili obecnej na realizację optymalizatora dynamicznego i modelu kosztów opartych jak do tej pory o model składu M0 [1].

3. Wnioski

Struktura metabazy zrealizowana w oparciu o warstwy zapewnia intuicyjną nawigację w ramach obiektów (wspomniane relacje rodzic-dziecko), między obiektami (pointery), dostęp do dowolnej cechy obiektu (nawigacja „prostopadła" do warstw, obiekty nie reprezentujące konkretnej cechy, po prostu nie występują w odpowiadającej jej warstwie) oraz dostęp do obiektów wykazujących daną cechę (nawigacja „równoległa" do warstwy).

Oddzielenie aspektów związanych z niezależnymi od siebie logicznie właściwościami (warstwy) wpływa na elastyczność metabazy oraz daje możliwość wprowadzania do niej nowych funkcjonalności. Zapewnione zostały także przejrzystość i czytelność metamodelu oraz łatwość rozbudowy.

Przeprowadzone testy prototypu metabazy opartego na opisanej powyżej strukturze warstwowej (zaimplementowanego w Java™) pozwoliły na poruszanie się po modelu bazy w sposób opisany powyżej oraz łatwy dostęp do jego wszystkich elementów. W kolejnym etapie prac nastąpi integracja metabazy z systemem ODRA tworzonym przez zespół prof. Kazimierza Subiety w PJWSTK w Warszawie w C#. Modyfikacja systemu ODRA o rozszerzoną metabazę pozwoli na uruchomienie i testy optymalizatora kosztowego w architekturze opisanej w [9–11].

Literatura

[1] Subieta K.: *Teoria i konstrukcja obiektowych języków zapytań*. Warszawa, PJWSTK 2004
[2] Subieta K., Beeri C., Matthes F., Schmidt J.W.: *A Stack-Based Approach to Query Languages*. Warszawa, IPIPAN Raport 738, 1993
[3] Płodzień J.: *Optimization Methods in Object Query Languages*. PhD Thesis. Warszawa, IPIPAN 2000
[4] Płodzień J., Kraken A.: *Object Query Optimization through Detecting Independent Subqueries*. Information Systems, vol. 25, No. 8, 2000, 467–490
[5] Płodzień J., Subieta K.: *Applying Low-Level Query Optimization Techniques by Rewriting*. Monachium, DEXA 2001, 867–876
[6] Płodzień J., Subieta K.: *Static Analysis of Queries as a Tool for Static Optimization*. IDEAS, Grenoble 2001, 117–122
[7] Płodzień J., Subieta K.: *Query Optimization through Removing Dead Subqueries*. Wilno, ADBIS 2001, 27–40

[8] Wiślicki J.: *Optymalizacja zapytań dla języka SBQL*. „Sieci i systemy informatyczne: teoria, projekty, wdrożenia, aplikacje", XI Konferencja, Łódź 2003, 245–252
[9] Wiślicki J.: *Model kosztów w optymalizacji zapytań dla obiektowych baz danych opartych o koncepcję SBA*. „Sieci i systemy informatyczne: teoria, projekty, wdrożenia, aplikacje", XII Konferencja, Łódź, 2004, 387–392
[10] Wiślicki J.: *Dynamiczna optymalizacja zapytań w obiektowych bazach danych*. Półrocznik AGH Automatyka, t. 8, z. 3, 2004, 497–503
[11] Wiślicki J.: *Architektura modelu kosztów w dynamicznej optymalizacji zapytań obiektowych baz danych opartych o koncepcję SBA*. WKŁ, 2005 (w druku)
[12] Ramakrishnan R.: *Database management systems*. McGraw-Hill, 1998
[13] Habela P.: *Metamodel for Object-Oriented Database Management Systems*. PhD Thesis. Warszawa, IPIPAN 2002
[14] Habela P., Roantree M., Subieta K.: *Flattening the Metamodel for Object Databases.* Bratysława, ADBIS 2002, 263–275
[15] Cattell R.G.G., Barry D.K.: *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000