ZESZYTY NAUKOWE POLITECHNIKI ŁÓDZKIEJ
Nr 1078 ELEKTRYKA, z. 121 2010

TOMASZ MAREK KOWALSKI
RADOSŁAW ADAMUS
KAMIL KULIBERDA
JACEK WIŚLICKI
Wydział Elektrotechniki Elektroniki Informatyki i Automatyki
Politechniki Łódzkiej

KONCEPCJA PRZEZROCZYSTEGO INDEKSOWANIA DLA ROZPROSZONEJ OBIEKTOWEJ BAZY DANYCH OPARTEJ NA ARCHITEKTURZE STOSOWEJ SBA

Recenzent: **prof. Dominik Sankowski**

Maszynopis dostarczono: 1. 10. 2010

Artykuł zawiera opis zaprojektowanego przez autorów w pełni przezroczystego systemu indeksowania dla obiektowej bazy danych opartej na stosowej architekturze (SBA, Stack-Based Architecture). Bazując na solidnym fundamencie SBA autorzy zaproponowali rozwiązania unikalne na polu indeksowania w rozproszonych obiektowych bazach danych. U podstaw zaimplementowanego systemu indeksowania leży moduł zarządzania indeksami oparty na autorskim mechanizmie automatycznych wyzwalaczy aktualizacji indeksów, który jest odpowiedzialny za utrzymanie zbieżności indeksów z danymi. Elastyczność opracowanego rozwiązania umożliwia m.in. optymalizacje przetwarzania predykatów selekcji opartych o dowolne deterministyczne wyrażenia kluczowe.

1. WPROWADZENIE

Indeksy to struktury danych powszechnie używane do organizacji i przyspieszenia dostępu do gromadzonych i przetwarzanych w rozmaitych systemach danych. Od lat 70. są one wykorzystywane do przezroczystej optymalizacji wydajności baz danych i są uznawane za najważniejszą metody poprawy ich szybkości. Dynamiczny rozwój informatyki w zakresie systemów zarządzania bazami danych (SZBD) i wszechstronność ich zastosowań zaowocował wieloma rozwiązaniami dotyczącymi struktur indeksów oraz metod ich wykorzystywania w procesie przetwarzania informacji.

W latach 90. popularnym tematem prac badawczych stały się obiektowe bazy danych. Nie mniej jednak, tendencja do przestawiania się na technologie obiektowe, które z reguły ułatwiają zarządzanie rozległymi systemami o dużym rozmiarze i złożoności, nie dotknęła SZBD. Przemysł skłaniał się ku rozwiązaniom relacyjnym, gdy kwestią kluczową była wydajność. Jest to aspekt wciąż zaniedbany w opartych o obiektowe paradygmaty bazach danych, z uwagi na niedostatek zaawansowanych procedur optymalizacyjnych. Dla przykładu na rynku istnieją tylko pojedyncze rozwiązania implementujące zaawansowane indeksy wspierające przetwarzanie danych o charakterze obiektowym (np. indeks ścieżkowy).

Podstawowe zasady indeksowania w obiektowych bazach danych nie różni się od indeksowania w systemach relacyjnych [4], [6], [8]. Z koncepcyjnego punktu widzenia najistotniejszą własnością indeksu w bazie danych jest przezroczystość. Oznacza ona, że programista aplikacji z bazą danych nie musi być świadomy istnienia indeksów. Najczęściej optymalizator zapytań jest odpowiedzialny za automatyczne wykorzystanie indeksów. Drugi ważny aspekt przezroczystości jest związany z utrzymywaniem spójności między indeksami a indeksowanymi danymi. Jest to problem tzw. automatycznej aktualizacji indeksu. Modyfikacje w bazie powinny być automatycznie wykrywane i odzwierciedlane w odpowiednich indeksach.

Zaproponowane przez autorów rozwiązania są przedstawione w kontekście stosowej architektury (SBA, Stack-Based Architecture) [1], [13] i wynikającego z niej języka zapytań (SBQL, Stack-Based Query Language). Architektura stosowa jest to formalna metodologia dotycząca obiektowych języków zapytań i programowania w bazach danych. Oparty na tej architekturze prototypowy system zarządzania bazą danych ODRA (Object Database for Rapid Application development) [2] to platforma przeznaczona do weryfikacji i testowania wyników prac badawczych.

Dalsza część artykułu zorganizowana jest następująco: sekcja druga opisuje krótko stan wiedzy dotyczący indeksowania w bazach danych, a sekcja trzecia wprowadza kontekst architektury stosowej (SBA). Sekcja czwarta opisuje opracowaną architekturę systemu indeksowania. Sekcja piąta dotyczy techniki

ulotnego indeksowania w przetwarzaniu danych heterogenicznych. Sekcja szósta podsumowuje artykuł.

2. INDEKSOWANIE W BAZACH DANYCH

Wiele metod indeksowania może być przeniesiona z systemów relacyjnych do obiektowych, a nawet zakres ich zastosowania często może być znacząco rozszerzony. Z drugiej strony istnieją sytuacje, kiedy metody znane z relacyjnych baz danych okazują się nieadekwatne w kontekście obiektowego modelu danych. W szczególności, operacja złączenia nie musi być optymalizowana przez indeksowanie, gdyż potrzeba złączeń jest znacznie mniejsza, jako skutek wprowadzenia identyfikatorów obiektów i jawnych linków (wskaźników) w bazie.

Badania nad indeksowaniem w zorientowanych obiektowo bazach danych były głównie skoncentrowane nad przetwarzaniem wyrażeń ścieżkowych oraz hierarchii dziedziczenia pomiędzy indeksowanymi kolekcjami [3], [11]. Nieliczne prace dotyczyły problemu generycznego podejścia zapewniającego przezroczystość automatycznej aktualizacji indeksu [7]. Jednakże, brakuje jakichkolwiek informacji potwierdzających, że proponowane rozwiązania zostały zaimplementowane w produktach komercyjnych lub open-source.

W rozproszonych bazach danych najbardziej zaawansowane rozwiązania opierają się o statyczne partycjonowanie indeksu. Są one zaimplementowane w czołowych produktach obiektowo-relacyjnych. Pozwalają one jedynie na definiowanie kluczy indeksu używając prostych wyrażeń korzystających z danych znajdujących się w jednej tabeli. W kwestii globalnej optymalizacji zapytań odnoszących się do heterogonicznych zasobów, autorzy nie znaleźli w literaturze naukowej żadnych sformalizowanych metod opartych o indeksowanie. Analiza stanu wiedzy jednoznacznie wskazuje na potrzebę rozwijania metod i architektury indeksowania dla rozproszonych obiektowych baz danych.

Badania nad optymalizacją zapytań w architekturze stosowej, obejmujące teoretyczne podstawy optymalizacji przez indeksowanie zostały opisane w pracy [12].

3. PROBLEMATYKA INDEKSOWANIA W KONTEKŚCIE SBA

Ortogonalność języka SBQL pozwala na wyjątkowo łatwe definiowanie złożonych predykatów selekcji odnoszących się do dowolnych danych. Głównym celem pracy jest opracowanie architektury indeksowania, która wspomagałaby przetwarzanie możliwie szerokiej rodziny predykatów. Wymaga to generycznego i kompletnego podejścia do problemu przezroczystości. Ponieważ praca dotyczy rozproszonych obiektowych baz danych, kolejnym

ważnym celem jest opracowanie metod optymalizacyjnych, które będą umożliwiały zrównoleglenie obliczeń, w szczególności poprzez wykorzystanie rozproszonych skalowalnych struktur indeksu.

Szczególnie trudną dziedziną w kontekście optymalizacji jest przetwarzanie zapytań odnoszących się do rozproszonych heterogenicznych zasobów. Z tego powodu jako kolejny cel autorzy postawili sobie opracowanie przezroczystej i wydajnej strategii indeksowania, którą można by stosować na poziomie globalnego schematu bazy. Owocem tych prac jest technika ulotnego indeksowania opisana w piątej sekcji artykułu.

Kluczowym aspektem pracy nad wszystkimi metodami optymalizacyjnymi jest zachowanie oryginalnej semantyki zapytań. W tym celu zostały określone reguły, które odnoszą się do opracowanych metod w kontekście przyjętego obiektowego modelu danych i języka zapytań SBQL. Znajomość tych reguł może być przydatna programistom w budowaniu zapytań, których postać umożliwia automatyczną optymalizację. Dodatkowo, przeprowadzone badania mogą być również pomocne projektantom baz danych. Między innymi, określono potencjalny wpływ innych metod optymalizacyjnych na pracę optymalizatora wykorzystującego indeksy.

Opracowane algorytmy i rozwiązania zostały zweryfikowane i potwierdzone na prototypowej implementacji obiektowej bazy danych ODRA [9], [10].

4. ORGANIZACJA INDEKSOWANIA W OBIEKTOWEJ BAZIE DANYCH

W tym rozdziale są omówione podstawowe elementy składające się na system indeksowania, czyli moduł optymalizatora odpowiedzialny za przezroczysty wybór odpowiednich indeksów dla zadanego zapytania (jeśli to możliwe) oraz automatyczna aktualizacja indeksów w odpowiedzi na zmiany dotyczące powiązanych danych. Powyższe rozwiązania zostały przez autorów opracowane i zaimplementowane w prototypie obiektowej bazy danych ODRA.

4.1. Architektura automatycznej aktualizacji indeksu

Każda modyfikacja wykonana na obiektach (dodanie, aktualizacja i skasowanie) jest realizowana przez interfejs CRUD (Create, Retrieve, Update, Delete) obiektowego składu ODRA. Zaproponowane podejście do aktualizacji indeksów koncentruje się na tym elemencie systemu jako najłatwiejszym i pewnym sposobie śledzenia modyfikacji. Możliwe modyfikacje, które mogą być wykonane na obiektach to:

- aktualizacja wartości obiektów typu *integer*, *double*, *string*, *Boolean*, *date* lub referencji,
- kasowanie,

- dodawanie obiektu potomnego (w wypadku obiektów złożonych),
- inne zależne od implementacji modyfikacje bazy danych.

Zaproponowana implementacja indeksowania wprowadza grupę struktur pomocniczych, nazwanych wyzwalaczami aktualizacji indeksów (*Index Update Triggers* w skrócie IUT), łącznie z definicjami wyzwalaczy (*Trigger Definition* w skrócie TD). Te elementy są niezbędne do wykonania aktualizacji indeksów. Każdy IUT wiąże jeden obiekt bazy danych z odpowiednim indeksem poprzez TD. Istniejące IUT-y automatycznie inicjalizują mechanizm aktualizujący, kiedy ma dojść do modyfikacji obiektu. Więcej niż jeden IUT może być połączony z pojedynczym obiektem. Definicje TD zapewniają metodę znalezienia obiektów, które powinny być wyposażone w IUT-y. Dodatkowo TD określa typ IUT-u.
Obiekt jest powiązany z IUT-ami kiedy uczestniczy w dostępie do obiektów niekluczowych lub w obliczaniu wartości kluczowych indeksów. Dlatego modyfikacje obiektów niezwiązanych z żadnym indeksem nie wyzwalają niepotrzebnie aktualizacji indeksu. Zmiana obiektów wyposażonych w IUT-y może wpływać na aktualność indeksów oraz na same IUT-y. Wprowadzono cztery podstawowe typy IUT-ów (każdy IUT odnosi się do innego typu TD):

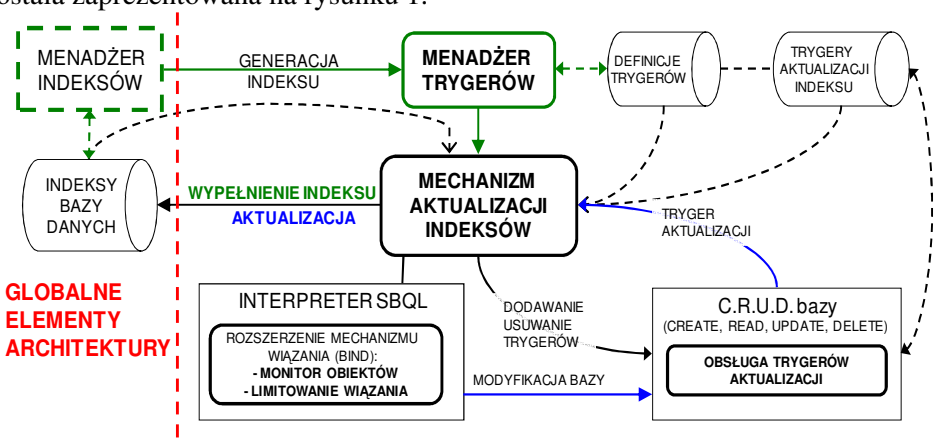
- Root Index Update Trigger (R-IUT) – jest domyślnie powiązany z korzeniem bazy danych. Kiedy nowy obiekt jest tworzony w korzeniu bazy danych wyzwalacz może powodować tworzenie NonkeyPath- lub Nonkey- Update Trigger (opisane poniżej) dla nowego obiektu potomnego. Wyzwalacz może być również użyty w celu inicjalizacji oraz zamknięcia wszystkich wyzwalaczy powiązanych z określonym indeksem.
- Key Index Update Trigger (K-IUT) – powiązany z obiektami używanymi do obliczania wartości klucza dla określonego obiektu niekluczowego (identyfikator przekazany razem z TD jako dodatkowy parametr). Każda modyfikacja takiego obiektu może potencjalnie zmienić proces obliczania klucza i jego wartość. Z tego powodu K-IUT jest odpowiedzialny za aktualizację odpowiednich wpisów indeksu oraz utrzymania odpowiednich K-IUT-ów powiązanych z danym obiektem niekluczowym.
- NonkeyPath Index Update Trigger (NP-IUT) – typ wyzwalacza generowany, gdy niekluczowy obiekt indeksu jest zdefiniowany za pomocą wyrażenia ścieżkowego (np. indeks idxAddrStreet), tj. kiedy obiekt niekluczowy nie jest bezpośrednim potomkiem korzenia bazy danych. Podobnie do R-IUT, ten wyzwalacz może powodować generację Index NonkeyPath lub Nonkey Update Trigger dla nowych obiektów potomnych.
- NonKey Index Update Trigger (NK-IUT) – wyzwalacz, który jest przypisany do obiektów indeksowanych (niekluczowych). Jest generowany przez wyzwalacze aktualizacji obiektu macierzystego (R-IUT albo NP-IUT).

Proces tworzenia NK-IUT składa się z następujących kroków:

1. Wstępnie NK-IUT jest przypisywany do danego obiektu niekluczowego.
2. Obliczana jest wartość klucza.
3. Odpowiadający wpis indeksu jest tworzony (jeśli prawidłowa wartość klucza została znaleziona).
4. K-IUT-y są generowane i parametryzowane identyfikatorami obiektów niekluczowych.

Tworzenie obiektów potomnych dla obiektów niekluczowych inicjalizuje procedury identyczne do opisanych w K-IUT.

Architektura zaproponowanego rozwiązania procesu aktualizacji indeksów została zaprezentowana na rysunku 1.



Rys. 1. Ogólna architektura automatycznej aktualizacji indeksu

Kiedy administrator dodaje indeks, przed wyzwalaczami IUT tworzone są definicje TD (zielone strzałki):

- menadżer indeksów inicjalizuje nowy indeks oraz wysyła do menadżera wyzwalaczy żądanie zbudowania definicji TD,
- następnie, TD aktywuje mechanizm aktualizacji indeksu, który bazując na wiedzy o indeksach oraz definicjach TD przystępuje do dodawania wyzwalaczy IUT.

Usunięcie indeksu skutkuje skasowaniem wyzwalaczy IUT (razem z NK-IUT skasowane zostają odpowiadające wpisy indeksu) oraz definicji TD. Mediator zarządzający dodawaniem i usuwaniem wyzwalaczy IUT jest specjalnym rozszerzeniem interfejsu CRUD.

Drugim przypadkiem, w którym uruchomiony zostaje mechanizm aktualizacji indeksu jest moment otrzymania przez interfejs CRUD sygnału modyfikacji obiektu połączonego z co najmniej jednym wyzwalaczem IUT (niebieska strzałka). CRUD powiadamia mechanizm aktualizacji indeksu o nadchodzących

modyfikacjach i przeprowadzone zostają wszystkie konieczne przygotowania przed aktualizacją bazy danych. Ten krok jest szczególnie istotny w wypadku zmian, które mogą wpłynąć na wartość klucza danego obiektu niekluczowego. Składa się on z:

1. lokalizowania wpisu indeksu, który odpowiada obiektowi niekluczowemu (potrzebna jest wartość klucza),
2. identyfikacji obiektów, które zostały użyte w procesie obliczania klucza (są one wyposażone w ten sam K-IUT).

Po zebraniu wymaganych informacji, CRUD wykonuje żądane modyfikacje, a mechanizm aktualizacji indeksu wykonuje następujące operacje:

1. aktualizuje wpisy indeksu dla danego obiektu niekluczowego,
2. aktualizuje istniejące IUT-y poprzez dodanie nowych lub usunięcie nieaktualnych.

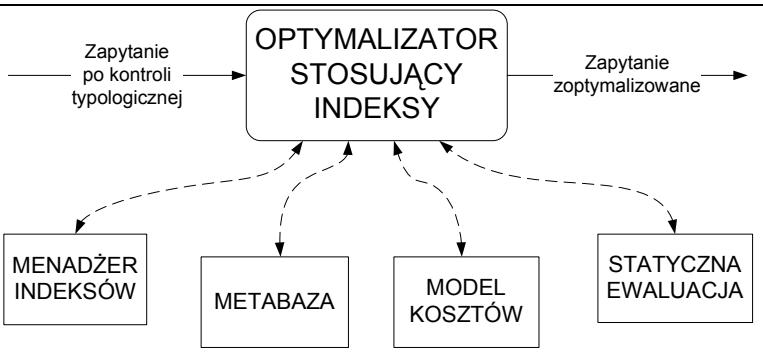
Znaczącym modułem, z którego korzysta mechanizm aktualizacji indeksów jest interpreter języka SBQL (pokazany również na rysunku 1) rozszerzony m.in. o możliwość rejestrowania obiektów bazy danych w czasie wykonywania wyrażenia wyznaczającego wartość kluczową. Ta funkcjonalność jest używana w celu lokalizacji wszystkich obiektów, które powinny być wyposażone w K-IUT-y.

Dzięki zaproponowanej architekturze automatycznie zapewniającej spójność indeksów i danych jest możliwa optymalizacja przez scentralizowane lub rozproszone przezroczyste indeksowanie przetwarzania predykatów selekcji opartych o dowolne wyrażenia kluczowe korzystające z danych w rozproszonej obiektowej bazie danych.

W przeciwieństwie do wszystkich zaimplementowanych rozwiązań problemu automatycznej aktualizacji indeksów zaprezentowanych w literaturze naukowej lub dołączonej do produktów komercyjnych, podejście autorów zaimplementowane w SZOBD ODRA, oparte o wyzwalacze aktualizacji indeksów (*Index Update Triggers*), zapewnia przezroczyste, kompletne oraz uogólnione wsparcie różnych definicji indeksów. Co więcej, dodatkowe koszty modyfikacji danych powiązane z konserwacją indeksów dotyczą wyłącznie obiektów wykorzystanych przy dostępie do poindeksowanych obiektów lub użytych przy obliczaniu wartości klucza.

4.2. Optymalizacja zapytań przez indeksowanie

Za optymalizację odpowiedzialny jest optymalizator stosujący indeksy, którego schemat działania przedstawiono na rysunku 2. Niezbędnymi elementami do jego pracy są menadżer indeksów, metabaza, model kosztów oraz statyczny ewaluator zapytań. Optymalizator indeksów automatycznie stosuje indeksy w trakcie procesu kompilacji zapytań.



Rys. 2. Schemat optymalizatora stosującego indeksy

Podstawowa procedura optymalizacji przez indeksowanie działa na selektywnych zapytaniach, gdzie lewą stronę operatora *where* stanowi kolekcja zaindeksowana przez jeden lub więcej indeksów. Algorytm analizuje wszystkie predykaty selekcji połączone z operatorem *and* i próbuje znaleźć indeks, którego klucz pasuje do predykatów. Jeżeli więcej niż jeden indeks zostaje znaleziony, optymalizator wybiera jeden z najlepszą selektywnością (na podstawie istniejącego modelu kosztów).

Opracowany optymalizator zapytań obsługuje szereg operatorów definiujących predykaty oraz ich koniunkcję i alternatywę. W celu zachowania semantycznej równoważności zapytań oryginalnych i przetworzonych przez optymalizator, określono reguły w kontekście przyjętego obiektowego modelu danych i języka zapytań SBQL. W szczególności uwzględniono m.in. następujące zagadnienia mające wpływ na ten aspekt optymalizacji:

- ustalenie przemienności predykatów selekcji,
- zakaz wykonywania optymalizacji, które mogą spowodować uniknięcie błędów wykonania lub efektów ubocznych pewnych wyrażeń,
- określenie reguł umożliwiających poprawną optymalizację zapytań przy użyciu indeksów, w których klucz jest zdefiniowany na opcjonalnym atrybucie.

Wszystkie opracowane metody są uniwersalne i mogą być stosowane do zapytań o zasięgu lokalnym lub globalnym.

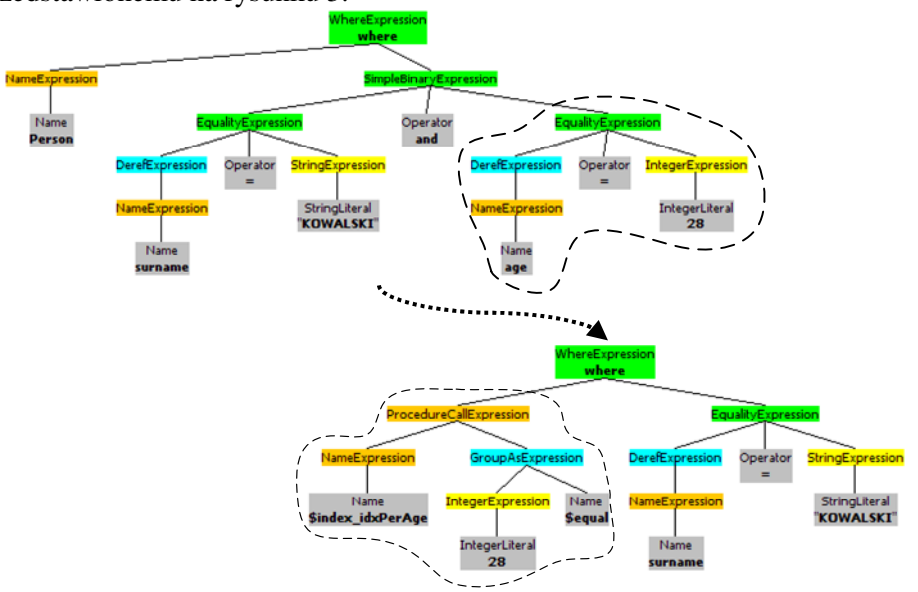
Implementacja indeksowania w ODRA wspiera indeksy z wieloma kluczami, typu gęstego lub zakresowego. Dodatkowo wprowadzono typ *enum*, m.in. do zwiększenia elastyczności indeksowania na wielu kluczach (np. możliwość pomijania klucza w wywołaniu indeksu). Dzięki własnościom języka SBQL, tj. ortogonalności i kompozycyjności, zaimplementowane rozwiązania dostarczają generycznego wsparcia dla dowolnych definicji indeksów np. złożonych z wyrażeń zawierających polimorficzne metody i operatory agregujące.

4.3. Przykład koncepcyjny

Przykładowo, weźmy zapytanie wyszukujące osoby o nazwisku Kowalski w wieku 28 lat:

```
Person where ((surname = "KOWALSKI") and (age = 28))
```

Jest ono w postaci pośredniej poddane procesowi optymalizacji przedstawionemu na rysunku 3.



Rys. 3. Przykład działania optymalizatora stosującego indeksy

Optymalizator zastępuje fragment zapytania wywołaniem indeksu zwracającym kolekcję osób w wieku 28 lat:

```
$index_idxPerAge(28 groupas $equal) where surname = "KOWALSKI"
```

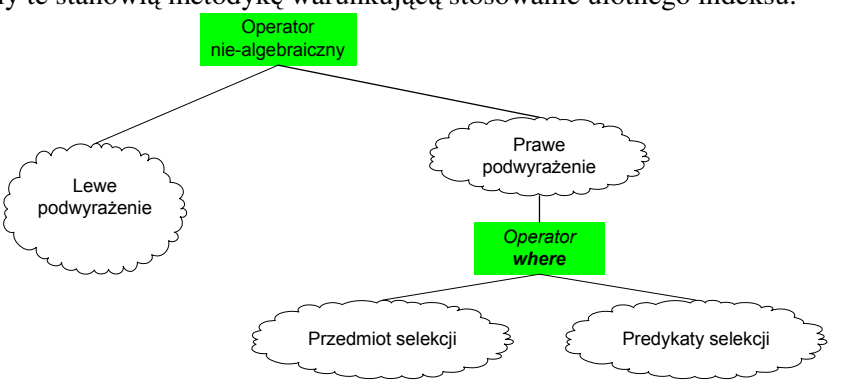
przez co zredukowany jest rozmiar przetwarzanej przez operator **where** kolekcji. Ta transformacja zachowuje semantykę zapytania jednocześnie powodując, że dla dużego zestawu danych wydajność jego przetwarzania wzrasta. Dla danych testowych, w których co 20-ta osoba (w przybliżeniu) była w wieku 28 lat, otrzymano ponad 15-krotne przyspieszenie.

5. TECHNIKA ULOTNEGO INDEKSOWANIA

W tym rozdziale omówiona jest opracowana przez autorów *technika ulotnego indeksowania*. Polega ona na omówionej wcześniej architekturze z wyłączeniem automatycznej aktualizacji indeksu, której osiągniecie w heterogenicznym, rozproszonym środowisku jest bardzo trudnym zadaniem, a nawet w praktyce często niemożliwym. Pominięcie mechanizmów automatycznej aktualizacji indeksu jest możliwe, gdyż w odróżnieniu od normalnych indeksów

ulotny indeks jest materializowany podczas wykonywania zapytania. Dzięki takiemu podejściu można tę technikę stosować do danych wirtualnie udostępnionych przez perspektywy SBQL, także do danych odnoszących się do zasobów heterogenicznych.

Opracowana technika jest skuteczna w przetwarzaniu złożonych zapytań, w których indeks jest wywoływany więcej niż jeden raz. Taka sytuacja jest przedstawiona na rysunku 4. Optymalizowane wyrażenie **where** powinno znajdować się po prawej stronie operatora nie-algebraicznego, którego lewe podwyrażenie zwraca kolekcję. Przedmiot selekcji powinien być niezależnym zapytaniem, a wartości predykatów selekcji zależne od wspomnianego operatora. Reguły te stanowią metodykę warunkującą stosowanie ulotnego indeksu.



Rys. 4. Drzewo zapytania podatnego na zastosowanie ulotnego indeksu

Najważniejsza cecha odróżniająca ulotny indeks od normalnego indeksu dotyczy ograniczeń na definicję wartości niekluczowych. Regularne indeksy mogą być zakładane tylko na kolekcjach obiektów określonych przez proste wyrażenia ścieżkowe. Takie ograniczenie jest spowodowane możliwościami mechanizmu automatycznej aktualizacji indeksu, więc nie dotyczy ono techniki ulotnego indeksowania. Definicja wartości niekluczowej ulotnego indeksu może być dowolnym wyrażeniem zwracającym:

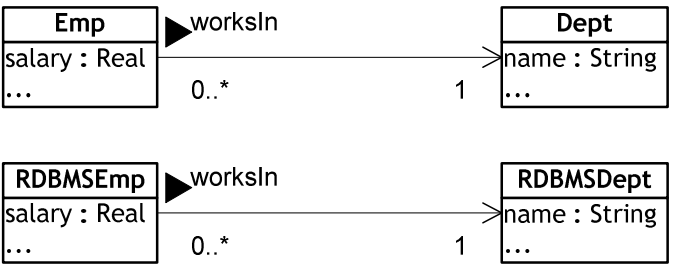
- referencje zdalnych obiektów,
- wirtualne obiekty (określone przez perspektywy),
- bindery i literały.

Podstawowym założeniem dotyczącym definicji klucza i wartości niekluczowych jest determinizm, tj. że wartości niekluczowe i kluczowe pozostają niezmienne dopóki dane użyte do ich wyznaczenia się nie zmienią.

Znaczącą zaletą techniki ulotnego indeksowania jest jej praktyczność z punktu widzenia integracji, rozproszonych i heterogenicznych zasobów. Zostało to potwierdzone testem przytoczonym w kolejnej sekcji.

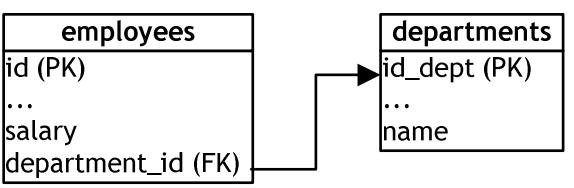
5.1. Przykład koncepcyjny

Test techniki ulotnego indeksowania przeprowadzono na testowym, heterogenicznym schemacie danych, przedstawionym na rysunku 5.



Rys. 5. Globalny obiektowy schemat

Testowy schemat składa się z dwóch kolekcji Emp i Dept reprezentujących rzeczywiste obiektowe dane opisujące firmę O oraz perspektyw SBQL odnoszących się do firmy R, przykrywających przezroczyście zintegrowane zasoby relacyjnej bazy danych (schemat przedstawiony na rysunku 6). Integracja relacyjnego schematu została zrealizowana za pomocą zaawansowanej generycznej osłony [14] opracowanej w ramach projektu e-Gov Bus - Advanced e-Government Information Service Bus [5].



Rys. 6. Rzeczywisty schemat relacyjnego zasobu

Przykładowe zapytanie:

```
Emp as empaux.(empaux.name + " " + empaux.surname,
(empaux.worksIn.Dept.name as deptnameaux).
((empaux.salary as empsalaryaux).
count(RDBMSEmp where worksIn.RDBMSDept.name = deptnameaux and salary >
empsalaryaux))
```

zwraca imię i nazwisko każdego pracownika firmy O wraz z liczbą pracowników firmy R, którzy pracują w takim samym dziale, co dany pracownik, i więcej od niego zarabiają. Jest ono zgodne z wzorem drzewa przedstawionym na rysunku 4. W podkreślonym podzapytaniu, które odwołuje się do zasobów relacyjnych, można więc zastosować optymalizację opartą o globalny ulotny

indeks. Alternatywnie może być ono również poddane lokalnej optymalizacji – optymalizatora osłony do relacyjnego zasobu.

Pierwsza optymalizacja polega na zastosowaniu ulotnego (globalnego) indeksu *vltlIdxRDBMSEmp*, którego parametry to nazwa działu i zakres płacy pracownika firmy R. Podkreślone podzapytanie zostałoby przekształcone do następującej postaci:

```
count($vltlIdxRDBMSEmp(deptnameaux groupas $equal;
(empsalaryaux, 1.7976931348623157E308, false, true) groupas $range))
```

Tworzenie ulotnego indeksu wymaga realizacji następującego zapytania SBQL:

```
RDBMSEmp join (worksIn.RDBMSDept.name, salary)
```

które jest zamieniane na następujące zapytanie SQL do relacyjnej bazy:

```
exec_immediately("select employees.info, employees.department_id,
employees.surname, employees.salary, employees.id, employees.sex,
employees.name, employees.birth_date, departments.name from employees,
departments where ((departments.id = employees.department_id) AND
(departments.id = departments.id))")
```

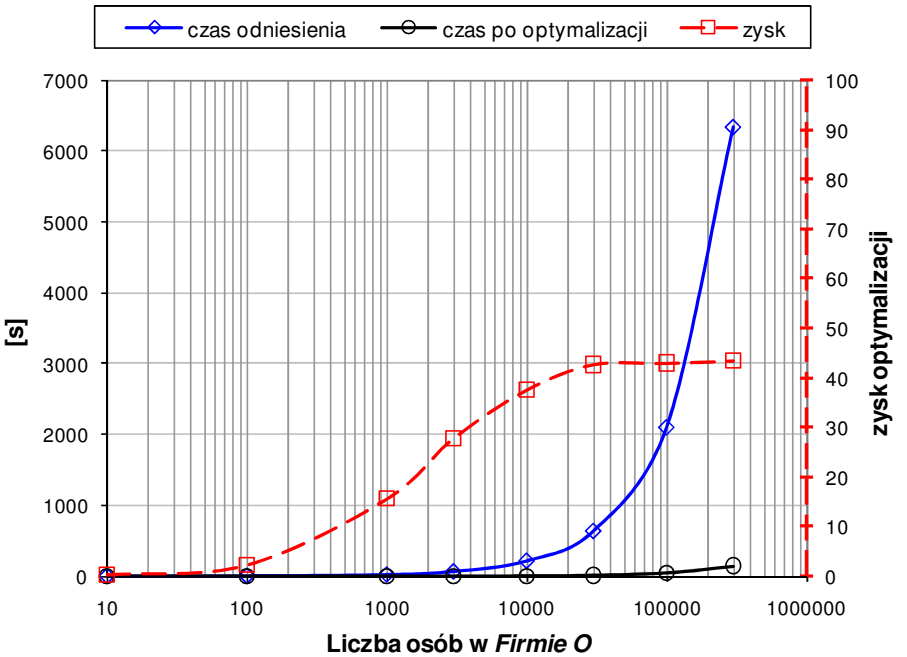
Alternatywny sposób ewaluacji polega na zastąpieniu podkreślonego podzapytania zapytaniem SQL wysłanym do relacyjnej bazy:

```
exec_immediately("select COUNT(*) from employees, departments where
departments.name = '" + deptnameaux + "' AND departments.id =
employees.department_id AND employees.salary > '" + empsalaryaux + "'")
```

Na relacyjnym zasobie jest ono poddane lokalnej natywnej optymalizacji (np. indeksy, projekcje i złączenia) i następnie wykonane. Zapytanie to jest wykonywane osobno dla każdego pracownika firmy O.

Technika ulotnego indeksowania w odróżnieniu od natywnej optymalizacji wymaga wysłania tylko jednego zapytania do zasobu relacyjnego.

Na wykresie na rysunku 7 przedstawiono w zależności od liczby osób w Firmie O czasy wykonania: po lokalnej optymalizacji (kolor niebieski), po zastosowaniu ulotnego indeksu (kolor czarny) oraz zysk optymalizacji. Przeprowadzony test wykazał ponad 40-krotne zmniejszenie czasu ewaluacji dla dużej liczby wywołań ulotnego indeksu.



Rys. 7. Czasy wykonania i zysk autorskiej optymalizacji

Jedyne przeciwwskazanie dotyczące zastosowania techniki ulotnego indeksowania występuje, jeśli materializacja indeksu pochłonęłaby zbyt dużo lokalnych zasobów, co mogłoby obniżyć ogólną wydajność przetwarzania. Z tego powodu, w przypadku braku wystarczającego modelu kosztów, decyzje o założeniu ulotnych indeksów powinien podejmować administrator.

6. PODSUMOWANIE

Przedstawione rozwiązanie zapewnia przezroczyste indeksowanie wykorzystujące indeksy oparte o jeden lub wiele kluczy. Optymalizacja dotyczy przetwarzania predykatów selekcji opartych o dowolne, deterministyczne i pozbawione efektów ubocznych wyrażenia, na które mogą się składać: np. wyrażenia ścieżkowe, funkcje agregujące i wywołania metod klas (z uwzględnieniem dziedziczenia i polimorfizmu). Zaproponowana architektura indeksowania może być zastosowana do rozproszonych homogenicznych źródeł danych. Wybór struktury indeksu, scentralizowanej czy rozproszonej, nie jest w żaden sposób ograniczony.

Opracowana *technika ulotnego indeksowania* polega na tej samej architekturze indeksowania, ale dodatkowo może być stosowana do przetwarzania danych heterogenicznych wirtualnie dostępnych poprzez perspektywy SBQL.

Dodatkowe nie opisane w tym artykule badania autorów w zakresie optymalizacji przez indeksowanie, to m.in.:

- optymalizacja ewaluacji kwantyfikatora egzystencjalnego,
- uwzględnienie relacji dziedziczenia pomiędzy klasami,
- określenie potencjalnego wpływu innych metod optymalizacyjnych na pracę optymalizatora stosującego indeksy,
- metoda optymalizacji rankingowych zapytań, która umożliwia wykorzystanie zarówno istniejących lokalnych indeksów, jak i rozproszonego, skalowalnego globalnego indeksu,
- oraz metodyka zwiększania niezależności zapytań od mechanizmów administracji indeksami.

Dalsze prace w tej tematyce powinny być ukierunkowane na uwzględnienie wertykalnej i mieszanej fragmentacji oraz perspektyw SBQL, rozwój metod opartych o technikę ulotnego indeksowania, opracowanie „nie-ulotnego" globalnego indeksowania heterogenicznych zasobów oraz dalszy rozwój implementacji w prototypie ODRA .

LITERATURA

[1] **Adamus R., Habela P., Kaczmarski K., Letner M, Stencel K., Subieta K.:** Stack-Based Architecture and Stack-Based Query Language. ICOODB 2008, Berlin: http://www.odbms.org/download/030.02%20Subieta%20Stack-Based%20Architecture%20and%20Stack-Based%20Query%20Language%20March%202008.PDF

[2] **Adamus R., Kowalski T.M., Subieta K. et al.:** Overview of the Project ODRA. Proceedings of the First International Conference on Object Databases, ICOODB 2008, Berlin, ISBN 078-7399-412-9, pp. 179-197.

[3] **Bertino E., Catania B., Chiesa L.:** Definition and Analysis of Index Organizations for Object-Oriented Database Systems. Information Systems, Vol. 23 No. 2, p. 65-108, April 1, 1998.

[4] **Chaudhuri S.:** An Overview of Query Optimization in Relational Systems. Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, Seattle, Washington, United States , pp. 34-43, 1998.

[5] e-Gov Bus: http://www.rodan.pl/web/guest/projekty_europejskie/egov_bus

[6] **Elmasri R., Navathe S. B.**: Fundamentals of Database Systems. 4th Edition, Pearson Education, Inc, publishing as Addison-Wesley, 2004, ISBN 0-321-12226-7.

[7] **Henrich A.:** The Update of Index Structures in Object-Oriented DBMS. Proceedings of the Sixth International Conference on Information and Knowledge Management (CIKM'97), Las Vegas, Nevada, November 10-14, 1997. ACM 1997, ISBN 0-89791-970-X: pp. 136-143.

[8] **Jarke M., Koch J.:** Query Optimization in Database Systems. ACM Computing Surveys 16(2), 1984, pp. 111-152.

[9] **Kowalski T.M., Kuliberda K, Adamus R., Wislicki J., Murleski J.:** Local and Global Indexing Strategies and Data-Structures in Distributed Object-Oriented Databases. SiS 2006 Proceedings, Łódź, Poland, 2006, pp. 153-156.

[10] **Kowalski T.M., Wiślicki J., Kuliberda K., Adamus R., Subieta K.:** Optimization by Indices in ODRA. Proceedings of the First International Conference on Object Databases, ICOODB 2008, Berlin, ISBN 078-7399-412-9, pp. 97-117.

[11] **Maier D., Stein J.:** Indexing in an object-oriented DBMS. In Proceedings on the 1986 international Workshop on Object-Oriented Database Systems, International Workshop on Object-Oriented Database Systems. IEEE Computer Society Press, pp. 171-182.

[12] **Płodzień J.:** Optimization Methods In Object Query Languages. PhD Thesis. IPIPAN, Warszawa 2000.

[13] **Subieta K.:** Stack-Based Approach (SBA) and Stack-Based Query Language (SBQL). http://www.sbql.pl, 2008.

[14] **Wiślicki J.:** An object-oriented wrapper to relational databases with query optimisation. PhD Thesis, Technical University of Łódź, Łódź 2008.

CONCEPT OF TRANSPARENT INDEXING FOR DISTRIBUTED OBJECT DATABASE BASED ON STACK-BASED ARCHITECTURE

Abstract

The paper describes fully transparent indexing system for an object-oriented database based on SBA (Stack-Based Architecture). Relying on solid fundament (i.e. SBA) authors designed unique solutions in the area of indexing in distributed object databases. The engine of indexing is an indices management module based on automatic index update triggers responsible for maintaining the cohesion between indices and data. The flexibility of the designed solution enables optimisation of processing of selection predicates built using arbitrary deterministic expressions.

Politechnika Łódzka
Katedra Informatyki Stosowanej