AUTOMATYKA • 2009 • Tom 13 • Zeszyt 3

Tomasz M. Kowalski*, Kamil Kuliberda*, Cezary Draus**,
Radosław Adamus*, ***, Jacek Wiślicki*, ***

Uogólnione podejście do aktualizacji indeksów w obiektowej bazie danych****

1. Wprowadzenie

Ogólna idea indeksowania w zorientowanych obiektowo bazach danych nie różni się od indeksowania w bazach relacyjnych [5]. Indeksy są strukturami danych, które zwiększają szybkość wyszukiwania obiektów spełniających podane kryteria. Indeksy powinny charakteryzować dwie istotne właściwości: przezroczystość optymalizacji i automatyczną aktualizację. Pierwsza oznacza, że użytkownik bazy danych nie potrzebuje być świadomym istnienia indeksów, ponieważ są one używane automatycznie podczas wykonywania zapytań. W niniejszym opracowaniu autorzy koncentrują się na drugiej właściwości, czyli konserwacji indeksów.

Indeksy, jak wszystkie nadmiarowe struktury, mogą stać się niespójne, kiedy przechowywane dane zostaną zmienione. Z tego powodu, aby zapewnić poprawność indeksów, aktualizacja danych powinna być połączona z przebudową odpowiednich indeksów. Proces przebudowy powinien być przezroczysty tak, aby zwolnić programistę z tego niewygodnego zadania i zmniejszyć przez to ryzyko popełnienia błędów. Dodatkowy czas konieczny do aktualizacji indeksów powinien zostać zminimalizowany. Jest to krytyczne z punktu widzenia efektywności dużych baz danych. Żadna zmiana danych nie może powodować długo trwających weryfikacji istniejących indeksów lub przebudowy całego indeksu od zera. Aby to osiągnąć, systemy bazodanowe powinny efektywnie wyszukiwać indeksy, które się dezaktualizują w skutek modyfikacji danych. Następnie, odpowiednia pozycja w indeksie po-

* Katedra Informatyki Stosowanej, Politechnika Łódzka
** Wydział Elektrotechniki, Elektroniki, Informatyki i Automatyki, Politechnika Łódzka
*** Stypendysta projektu „Innowacyjna dydaktyka bez ograniczeń – zintegrowany rozwój Politechniki Łódzkiej – zarządzanie uczelnią, nowoczesna oferta edukacyjna i wzmacnianie zdolności do zatrudniania, także osób niepełnosprawnych" współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego
**** Praca naukowa finansowana ze środków na naukę w latach 2008/2009 jako projekt badawczy nr N516 383334

winna zostać poprawiona, aby wszystkie wywołania indeksu zapewniały poprawną odpowiedź. Takie działania nie powinny wpływać negatywnie na czas pobierania informacji z bazy danych, a ogólny dodatkowy koszt zapisu danych powinien być jak najmniejszy. Jednakże znalezienie ogólnego i zarazem optymalnego rozwiązania nie jest możliwe z powodu złożoności SZBD (System Zarządzania Bazą Danych). Takie zadanie wymaga analizy wielu rzeczywistych sytuacji po to, aby zminimalizować pogorszenie się wydajności.

Prototyp ODRA jest oparty na architekturze stosowej (*Stack Based Architecture* – SBA) [1, 19]. SBA jest formalną metodologią dotycząca zorientowanych obiektowo języków zapytań i programowania. Głównym celem projektu ODRA jest wytyczanie nowych paradygmatów rozwoju aplikacji bazodanowych. ODRA wprowadza własny język zapytań SBQL (*Stack Based Query Language*). Zaproponowana implementacja indeksowania umożliwia tworzenie, przezroczystą optymalizację oraz automatyczną konserwację indeksów. Wszystkie cechy potrzebne do zapewnienia pożądanego zachowania indeksów są zaimplementowane i funkcjonalne [10].

Reszta dokumentu zorganizowana jest jak wymieniono poniżej. Następny rozdział obejmuje ogólny przegląd mechanizmów indeksowania oraz ich ograniczenia z naciskiem na możliwości wynikające z podejścia do automatycznej aktualizacji indeksów, rozdział 3 prezentuje SBA i SBQL, rozdział 4 opisuje podejście do automatycznej aktualizacji indeksów w prototypie ODRA wraz z wyjaśniającymi przykładami, rozdział 5 zawiera podsumowanie oraz wnioski.

2. Zarys indeksowania w relacyjnych oraz obiektowych bazach danych

Prawie 40 lat badań nad systemami relacyjnymi zaowocowało wieloma rozwiązaniami w dziedzinie indeksowania [5]: *primary index*, *clustered index*, *secondary access paths*, *multi-key index*, *temporary index* [13], *development of diverse index structures*, *itp*.

W systemach SZRBD (System Zarządzania Relacyjną Bazą Danych) klucze definiujące indeks na tabeli są zwykle prostymi wartościami przechowywanymi w kolumnach. Takie indeksy wymagają prostych mechanizmów zapewniających przezroczystość użytkownika. Optymalizatory mogą w łatwy sposób zidentyfikować wyrażenia *where* adresując odwołujące się do poindeksowanych warunków selekcji. Łatwe do wykrycia przez silnik BD są również modyfikacje w indeksowanej tabeli, dlatego szczegóły automatycznej aktualizacji indeksów dla systemów SZRBD są pomijane w specyfikacjach technicznych i traktowane raczej jako kwestie implementacyjne.

Derived key index (iSystem DB2/400 by IBM) [9], *function-based index* (Oracle by Oracle Corporation)[18], *functional indexes* (Informix by IBM) [9] oraz inne podobne rozwiązania umożliwiające definiowanie kluczy używając wyrażeń odnoszących się do więcej niż jednej kolumny tabeli oraz funkcji wewnętrznych lub użytkownika zwykle nie wprowadzają trudności koncepcyjnych. Funkcje wspierające mogą być napisane w języ-

kach natywnych (np. PL/SQL) lub zewnętrznym języku programowania (np. C++, Java, itp.). Muszą być one deterministyczne (tj. zależne tylko od stanu składu bazy danych) i nie mogą produkować efektów ubocznych (tj. nie wprowadzać żadnych zmian danych). Dla indeksów funkcyjnych (*function-based*) automatyczna aktualizacja indeksów wymaga po prostu reakcji na zmiany jakiejkolwiek wartości w którejkolwiek kolumnie używanej w definicji klucza. Niemniej jednak, ten aspekt indeksowania staje się złożony, kiedy rozważamy model zorientowany obiektowo i rozszerzenia językowe. W drastycznych przypadkach może to nawet prowadzić do poważnych błędów (zobacz rozdz. 2.1).

Rozwiązanie podobne do zaprezentowanego powyżej – *computed-column indices* – zostało zaimplementowane w MSSQL Server [13]. Opiera się ono na dodatkowej kolumnie w tabeli (computed-column), która może definiować indeksowane wyrażenia używając atrybutów pochodnych oraz funkcji użytkownika. W rezultacie auto-aktualizacja indeksu zależy od aktualizacji dodatkowej kolumny.

Tematy organizacji indeksów i ich optymalizacji w zorientowanych obiektowo systemach zarządzania bazami danych zostały głęboko zbadane [2, 7, 22]. Systemy SZOBD (System Zarządzania Obiektową Bazą Danych) bazują na hierarchicznym, obiektowo zorientowanych modelu danych. Jednym z istotnych pojęć modelu obiektowego jest referencja, tj. wskaźnikowe połączenie z obiektem. Referencje wyrażają związki (asocjacje) między obiektami. W rezultacie prób standaryzacji systemów zarządzania zorientowanymi obiektowo bazami danych ODMG zaproponowało OQL, który do pewnego stopnia wpłynął na rozwój zorientowanych obiektowo języków zapytań [3] OQL wprowadza wyrażenia ścieżkowe budowane za pomocą nazw obiektów oddzielonych kropkami w celu uzyskania łatwej nawigacji przy użyciu wskaźników do obiektów. Nawigacja do wskazywanego obiektu w SZOBD może być szybka, gdyż jest zwykle rozwiązana na niskim poziomie korzystając z bezpośredniego połączenia. W relacyjnym modelu takie związki (tj. zależności podstawowego klucza obcego) wymagają tworzenia złączeń, a dla efektywnego wykonania zapytań niezbędne są indeksy. Niemniej, niektóre zorientowane obiektowo systemy mogą bezpośrednio opierać się na płaskim, relacyjno-podobnym modelu danych. W takim wypadku, nawigacja wzdłuż wskaźników ciągle wymaga tworzenia bezpośrednich połączeń wśród obiektów. Założenie, które ogranicza wyrażenia adresujące OQL to fakt, że operand przed operatorem kropki nie powinien dostarczać kolekcji.

W SZOBD przeprowadzono wiele badań w celu zwiększenia efektywności przetwarzania zagnieżdżonych predykatów w zapytaniach (tj. dostępnych używając wyrażeń ścieżkowych) w systemach SZOBD. Badania te dodatkowo są rozszerzone o rozważanie problemów związanych z dziedziczeniem. Rozwiązania o największym znaczeniu to Multi-Index, Inherited Multi-Index, Nested-Inherited Index, Path Index, Access Support Relations [2]. Jednakże, aspekt przezroczystości związany z automatyczną aktualizacją indeksów nie jest w tych pracach zawsze precyzyjnie wyjaśniony. Ogólnie rzecz biorąc, zakłada się, że każda modyfikacja atrybutu instancji klasy oraz tworzenie i kasowanie egzemplarza powinny skutkować odpowiednią akcją aktualizującą indeks. Jednakże, instancje jednej z klas dostępne poprzez ścieżkowe wyrażenie indeksujące mogą się znajdować w różnych

kolekcjach. Co więcej, te kolekcje mogą przechowywać dowolną liczbę obiektów nie powiązanych z obiektami indeksowanymi obciążając tym samym proces autoaktualizacji indeksów. Ten brak konsekwencji jest jednakże usprawiedliwiony. W podejściu do automatycznej aktualizacji indeksów zaprezentowanym w [2] wszystkie egzemplarze klasy powiązane z indeksem opartym o wyrażenie ścieżkowe powinny zostać rozważone dla zapewnienia poprawności indeksu po modyfikacjach danych.

Rozproszony obiektowy system zarządzania H-PCTE stworzony na Uniwersytecie Siegen zaproponował inne rozwiązanie problemu automatycznej aktualizacji indeksów. Jest ono niezależne od rodzaju struktury indeksu i jego zawartości [7]. Podejście bazuje na tak zwanych *definicjach aktualizujących indeks*, które składają się z opisu zdarzenia dla zdarzenia wymuszającego potrzebę aktualizacji indeksu, referencję do struktury indeksu, zapytanie określające elementy, dla których odpowiednie wpisy indeksu mają być zaktualizowane oraz odpowiadającą operacje aktualizacji. Takie *definicje aktualizujące indeks* mogą być utworzone podczas tworzenia indeksu. Rozwiązanie radzi sobie ze złożonymi atrybutami, np. regularnymi wyrażeniami ścieżkowymi oraz funkcjami agregującymi. Z drugiej strony, jego autorzy wspominają o pewnych jego ograniczeniach dotyczących efektywności i uwzględnienia metod użytkownika dając jedynie ogólne sugestie jak zaradzić tym problemom.

Inne podejście do autoaktualizacji indeksów zostało rozważone dla *function-based indexing* rozwijanego w kontekście rozproszonego, zorientowanego obiektowo systemu bazodanowego Thor przez Massachusetts Institute of Technology używając tak zwanych schematów rejestrujących obiekty [8]. Rejestracja dotyczy jedynie obiektów, których modyfikacja może wpłynąć na indeks. Aktualizacja indeksu jest wyzwalana przez mechanizm, który sprawdza dane rejestracji podczas modyfikacji obiektów. Mimo teoretycznej ogólności tego rozwiązania nie został on w pełni zaimplementowany.

Według wiedzy autorów niewielka liczba technik indeksujących zaproponowanych w literaturze naukowej została wdrożona do komercyjnych produktów i ważniejszych prototypów SZOBD. Dokładny przegląd zastosowanych mechanizmów indeksujących jest możliwy poprzez analizę bardziej znaczących obiektowych baz danych.

Przezroczyste indeksowanie w db4o SZOBD firmy db4objects jest zapewnione jedynie dla atrybutów klas definiujących indeksowane kolekcje [4]. Objectivity/DB przez Objectivity dodatkowo wspiera połączone indeksy na większej liczbie atrybutów [14]. Zaawansowane, przezroczyste indeksowanie w Versant zostało osiągnięte poprzez wirtualne atrybuty, co stanowi technikę podobną do indeksowania opartego na obliczonej kolumnie (computed column) w Microsoft SQL Server. Takie podejście umożliwia indeksowanie pochodnych, wirtualnych atrybutów używając jednego lub więcej atrybutów klasy [20]. Jedyną przetestowaną bazą danych, która wspiera przezroczyste indeksowanie przy użyciu krótkich wyrażeń ścieżkowych (atrybuty obiektów powiązanych z obiektami indeksowanymi) jest GemFire [6]. ObjectStore wspiera dodawanie indeksów na kolekcjach używając złożonych wyrażeń ścieżkowych jako klucza (tzw. indeksy *multistep indices* mogą również wykorzystywać metody klas) jednakże umożliwia jedynie częściową przezroczystość automatycznej aktualizacji indeksów. ObjectStore automatycznie aktualizuje indeks w momen-

cie usunięcia lub dodania elementów kolekcji. Jednakże, aktualizacja wpisu indeksu po modyfikacji danych musi być jawnie wyzwolona przez programistę [16].

Wymienione powyżej prototypy i produkty komercyjne reprezentują jedynie fragment istniejącego obrazu systemów zarządzania zorientowanymi obiektowo bazami danych. Niemniej jednak, umożliwiają wystarczający przegląd stanu techniki indeksowania w SZOBD.

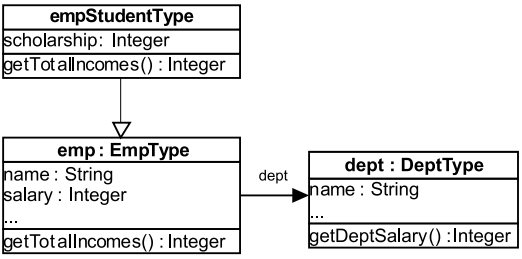
2.1. Ograniczenia Function-Based Index w SZORBD

Proste relacyjne systemy wyznaczają ograniczenia definicji klucza indeksowania:

- klucz indeksu może być jedynie obliczony używając danych w bieżącym rekordzie, ponieważ SQL nie pozwala na definiowanie indeksu używającego danych z innych tabel powiązanych za pomocą relacji klucz podstawowy klucz obcy,
- funkcje agregujące w SQL są zabronione w definicjach indeksów, ponieważ użyte jako warunek selekcji zwracają pojedynczą wartość dla całej tabeli.

Bez zaawansowanych rozszerzeń zorientowanych obiektowo nie ma wsparcia dla metod powiązanych z rekordami, polimorfizmem oraz wyrażeniami ścieżkowymi. Takie ograniczenia dotyczą również większości przezroczystych technik indeksowania w SZORBD (System Zarządzania Obiektowo-Relacyjną Bazą Danych). Autorzy nie znaleźli żadnych obiektowo-relacyjnych baz danych wspierających indeksowanie używające funkcji agregujących lub wyrażeń ścieżkowych. Relatywnie złożone indeksy wiążące się z wywołaniem metod oraz polimorfizmem w środowisku obiektowo-relacyjnym mogą być utworzone przy użyciu rozwiązania Oracle *function-based indices* [18]. Dokumentacja Oracle nie daje jednak wystarczającej informacji o rozważanym problemie automatycznej aktualizacji takich indeksów.

W celu poznania właściwości podejścia Oracle autorzy przeprowadzili testy wprowadzając schemat widoczny na rysunku 1.



Rys. 1. Przykład schematu obiektowo-relacyjnego

Metoda *getTotalIncomes* klasy *EmpType* zwraca wartość atrybutu *salary*. Jest ona przeładowana w klasie *empStudentType* w celu uwzględnienia pola *scholarship*. Tabela *emp* składa się z rekordów obu typów. Tworzenie indeksu na metodzie *getTotalIncomes* powiązanej z tabelą wygląda następująco:

CREATE INDEX emp_gettotalincomes_idx **ON** emp e (e.getTotalIncomes());

Taki indeks jest automatycznie używany przez optymalizator zapytań. Efektywność procesu selekcji jest zwiększona nie tylko poprzez redukcję liczby przejrzanych rekordów, ale również poprzez uniknięcie wywoływania metod, ponieważ obliczony rezultat może być zwrócony poprzez indeks. Indeks został przetestowany na serii prostych zapytań. Modyfikacje atrybutu pensji lub stypendium, np.:

UPDATE emp e **SET** e.salary = 1500 **WHERE** e.name = «KUC»;

wyzwalają odpowiednie zmiany w indeksie. Zgodnie z przewidywaniami czas takiej modyfikacji po dodaniu indeksu *emp_gettotalincomes_idx* pogorszył się. Przetwarzanie aktualizacji jest ponad trzy razy dłuższe, ponieważ automatyczna aktualizacja indeksu wymaga zmiany odpowiadających wpisów w indeksie. Na tyle ile było to możliwe, testy pokazały, że stworzony indeks działa prawidłowo.

Niestety definiowanie indeksu na metodzie w Oracle pokazuje pewne niespodziewane wady. Operacje aktualizacji indeksu są również wyzwalane podczas modyfikacji atrybutu *name* kolekcji *emp*. Podobnie, zmiana innych atrybutów obiektów *emp* jest wolniejsza (również bardziej niż 3 razy). Zostało to spowodowane niepotrzebnymi czynnościami aktualizacyjnymi indeksu. Podejście Oracle do aktualizacji indeksów w wypadku indeksów opartych o metody polega na wyzwalaniu mechanizmów aktualizacji indeksu podczas każdej modyfikacji dowolnego atrybutu krotki powiązanej z wpisami indeksu.

Wada wymieniona powyżej rośnie do poważnego problemu w wypadku, kiedy metoda wykorzystana do zdefiniowania klucza dostaje się do danych poza indeksowanym obiektem. Dla przykładu metoda *getDeptSalary DeptType* ma poniższą definicję:

```
CREATE TYPE BODY dept_type IS
 MEMBER FUNCTION getdeptsalary RETURN NUMBER DETERMINISTIC IS
 BEGIN DECLARE aux NUMBER;
  BEGIN
   SELECT sum(salary) INTO aux FROM emp e WHERE e.dept.name =
   self.name;
 RETURN aux;
END; END; END;
```

Dostaje się ona nie tylko do danych krotki typu *DeptType* ale również dosięga kolekcji *emp*. Metoda oblicza sumę zarobków pracowników departamentu i może być użyta jako warunek selekcji. Oracle umożliwia również indeksowanie kolekcji *dept* za pomocą metody *getDeptSalary*:

```
CREATE INDEX dept_getdeptsalary_idx ON
  dept d (d.getDeptSalary());
```

Podobnie jak w przypadku *emp_gettotalincomes_idx*, polecenie aktualizujące wpisy *dept* wyzwala aktualizację indeksu. Jednakże jakiekolwiek modyfikacje w obiektach *emp*, np.:

```
INSERT INTO EMP
 SELECT emp_type («Smith»,350,REF(d))
 FROM DEPT d WHERE d.name = «HR»;
```

nie są brane pod uwagę i indeks *dept_getdeptsalary_idx* traci spójność z danymi. Niestety zapytania, które używają tego indeksu, np.:

```
  SELECT d.name, d.getDeptSalary() FROM DEPT d
WHERE d.getDeptSalary() < 24500;
```

mogą zwracać nieprawidłowe odpowiedzi, ponieważ proces selekcji oraz ostateczne rezultaty zależą od zawartości indeksu. Stąd wykorzystane rozwiązanie aktualizacji indeksu nie jest poprawne dla obsługiwania indeksów z kluczami opartymi o "zbyt skomplikowane" metody. W praktyce indeksy *function-based* w Oracle mogą prowadzić do błędnego działania zapytań i aplikacji w bazie danych.

3. SBA i SBQL

Architektura stosowa (SBA) [17, 19] to teoretyczna i metodologiczna podstawa dla zaprojektowanego rozwiązania zaprezentowanego w tym artykule. SBA rekonstruuje koncepcje języków zapytań z punktu widzenia języków programowania (PL).

Zakłada on zasadę relatywizmu obiektów, czyli braku koncepcyjnej różnicy pomiędzy obiektami różnych typów lub przechowywanymi w składzie na różnych poziomach hierarchii. Wszystko (np. obiekt osoby, atrybut pensji, procedura zwracająca wiek osoby, widok zwracający dobrze opłacanych pracowników) jest obiektem.

W SBA język zapytań (nazwany SBQL – *Stack Based Query Language*) jest specjalnym rodzajem języka programowania. Dlatego też semantyka zapytań bazuje na mechanizmach dobrze znanych z języków programowania, takich jak stos środowiskowy. SBA rozszerza tą ideę dla przypadków operatorów zapytań takich jak selekcja, projekcja/nawigacja, łączenie, kwantyfikatory oraz inne. Używając SBA można precyzyjnie określić semantykę operacyjną (abstrakcyjna implementacja) języków zapytań, włączając relacje z koncepcjami obiektowo relacyjnymi, wbudowanymi zapytaniami do konstrukcji imperatywnych oraz zagnieżdżanie zapytań w abstrakcjach programistycznych: procedurach, perspektywach, metodach, modułach itp.

3.1. Wiązanie nazw

Naturalną właściwością języków opartych na SBA jest zasada zakresu wiązania nazw. Każda nazwa pojawiająca się w zapytaniu jest ograniczona do odpowiedniej nietrwałej

jednostki (obiekt, atrybut, metoda, parametr, itp.) zależnie od jej zakresu. Zasada jest obsługiwana za pośrednictwem stosu środowiskowego (ENVS) zawierającego *bindery*. Mechanizm ten jest dobrze znany z popularnych implementacji języków programowania.

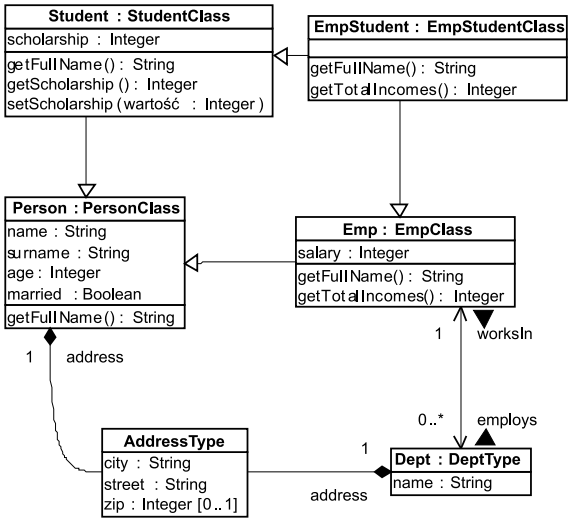
Idea ENVS jest rozszerzona w SBA na kolekcje bazy danych oraz na wszystkie typowe operatory zapytań znane z SQL oraz OQL (np. projekcja, nawigacja, łączenie, kwantyfikatory). Dzięki semantyce opartej o stos (stack-based) osiągnięto pełną ortogonalność oraz złożoność operatorów zapytań. ENVS wspiera również rekursję oraz parametry: wszystkie funkcje, procedury, metody oraz perspektywy zdefiniowane przez SBA mogą być rekurencyjne z definicji.

Co więcej użycie ENVS może być również rozszerzone ponad wykonywanie zapytań do innych zadań jak wsparcie czasu wykonania dla aktualizacji indeksów. Rozszerzenie to zostanie zaprezentowane w następnym rozdziale, jako część zaproponowanego mechanizmu aktualizacji indeksów.

4. Rozwiązanie aktualizacji indeksu

4.1. Przykładowa baza danych

Schemat na rysunku 2 jest wprowadzony w celu zilustrowania opisu indeksowania w SZOBD ODRA.



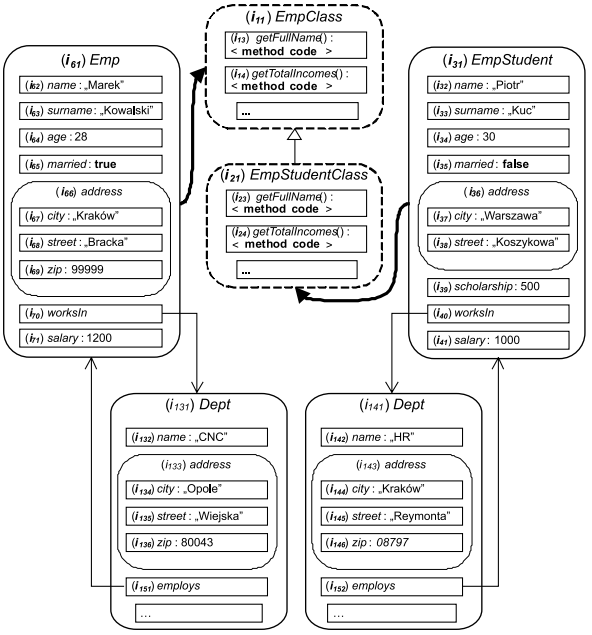
Rys. 2. Przykładowy schemat zorientowany obiektowo

Przykładowy schemat ilustruje rekordy personelu pewnej firmy. Wprowadzono kilka klas PersonClass, StudentClass, EmpClass, EmpStudentClass oraz dwa typy strukturalne DeptType i AddressType. Trwałe instancje klas wymienionych powyżej mogą być dostępne

za pośrednictwem nazw klas ich instancji Person, Student, Emp i w końcu EmpStudent. Obiekty reprezentujące departamenty firmy, nazwane *Dept* mają strukturę *DeptType* z atrybutem głównym *name*. Instancje klasy *EmpClass* reprezentują obecnych pracowników firmy oraz rozszerzają obiekty *Person* o atrybut *salary*. Obiekty *Emp* oraz *Dept* są powiązane obiektami wskazującymi nazwanymi odpowiednio *worksIn* oraz *employs*. Kolejna klasa, która rozszerza klasę *PersonClass* to *StudentClass*. Wprowadza ona atrybut *scholarship*. Ostatnia klasa pokazana na schemacie, nazwana *EmpStudentClass,* dziedziczy po *EmpClass* oraz *StudentClass*. Została wprowadzona, aby reprezentować studentów, którzy są jednocześnie pracownikami firmy. Użycie nazwy *Person* w zapytaniu SBQL daje w rezultacie wszystkie instancje klasy *PersonClass* i jej podklas. Podobnie poprzez nazwę *Emp* programista odnosi się do instancji obu klas *EmpClass* i *EmpStudentClass*.

Klasy mogą zawierać metody korzystające z polimorfizmu (nadpisywanych w podklasach). Na przykład metoda *getTotalIncomes*() klasy *EmpClass* zwraca wartość atrybutu *salary*, ale dla instancji klasy *EmpStudentClass* zwraca sumę atrybutów *salary* i *scholarship*.

Odnosząc się do danych schematu z rysunku 2, możemy wprowadzić przykład składu na rysunku 3 prezentujący klasy i obiekty, ich wartości, identyfikatory oraz najistotniejsze relacje między nimi.



Rys. 3. Przykładowy skład zawierający klasy i obiekty

4.2. Indeksowanie w ODRA SZOBD

Podstawy teorii optymalizacji zapytań używając indeksów w SBA zostały zaprezentowane w [22].

Implementacja indeksowania wspiera indeksy oparte o struktury liniowego hashingu. [11], które mogą być łatwo rozszerzone do rozproszonej wersji SDDS [12], aby optymalnie wykorzystać zasoby obliczeniowe gridu.

Ponadto, implementacja zapewnia szerokie wsparcie dla optymalizacji zapytań poprzez:

- wsparcie dla optymalizacji gęstych i zakresowych zapytań na kluczach typów *integer*, *real*, *string, date*,
- gęste indeksowanie dla wartości kluczy typu *reference*,
- indeksowanie przy użyciu wielu kluczy,

Schemat na rysunku 2 daje możliwość zaprezentowania szerokiej różnorodności indeksów obsługiwanych przez silnik indeksów ODRA. Podejście autorów do aktualizacji indeksów zostanie wyjaśnione przy użyciu następujących przykładowych indeksów:

- idxPerAge – który zwraca obiekty Person wg wartości atrybutu *age*.
- *idxEmpWorkCity* – indeks, który wykorzystuje pochodny atrybut worksIn.Dept.address.city do zwracania obiektów *Emp*.
- *idxAddrStreet* – to indeks, który zwraca podobiekty *address* obiektów *Person* wg atrybutu *street*. Inaczej niż w wypadku innych indeksów obiekty niekluczowe są zdefiniowane za pomocą wyrażenia ścieżkowego, tj. *Person.address*.
- *idxEmpTotalIncomes* – indeks, który używa metody *getTotalIncomes*() z klasy *Emp* jako klucza do selekcji obiektów *Emp*. Ta metoda jest nadpisana dla instancji klasy *EmpStudent*.

Ponadto, nawet bardziej złożone indeksy mogą ułatwiać przetwarzanie zapytań, np. indeks *idxDeptYearCost* dla obiektów *Dept*, którego klucz jest oparty na wyrażeniu sum(*employs.Emp.salary*)*12, które zwraca przybliżony koszt całkowity rocznych płac dla wybranego departamentu.

4.3. Wyzwalacze aktualizacji indeksów

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 da-

nych 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.

Nasza implementacja wprowadza cztery podstawowe typy IUT-ów (każdy IUT odnosi się do innego typu TD):

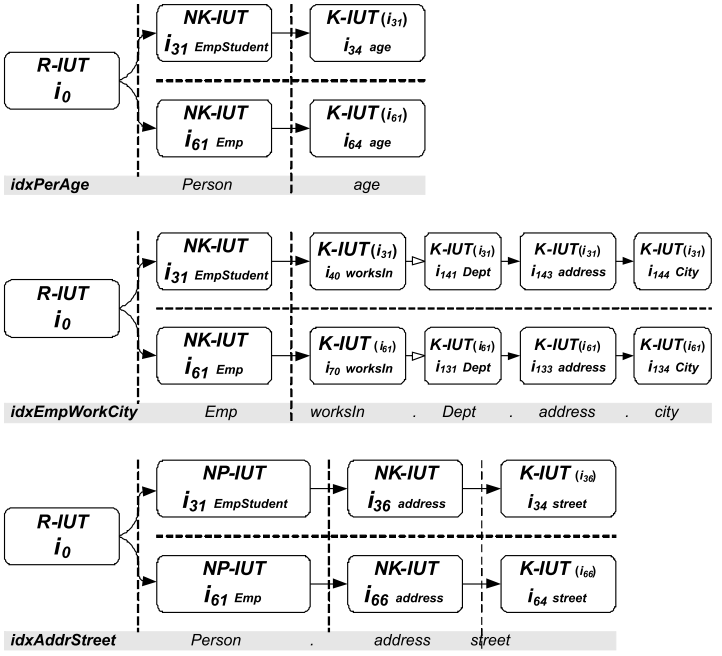
- Root Index Update Trigger (R-IUT) – jest domyślnie powiązany z korzeniem bazy danych, który jest bezpośrednim lub pośrednim obiektem nadrzędnym dla wszystkich indeksowanych obiektów bazy. Kiedy nowy obiekt jest tworzony w korzeniu bazy danych wyzwalacz może powodować tworzenie NonkeyPath- lub Nonkey- 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 powiązanego z obiektami, które są potencjalnie obiektami nadrzędnymi dla nowych obiektów indeksowanych. Ten typ wyzwalacza jest 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.

Bazując na przykładowym składzie przedstawionym na rysunku 3, dla indeksów *idx PerAge*, *idxEmpWorkCity* oraz *idxAttrStreet* wprowadzonych w poprzedniej sekcji IUT-y pokazane na rysunku 4 zostałyby wygenerowane. Załóżmy, że i_0 to identyfikator korzenia bazy. Identyfikatory obiektów niekluczowych powiązanych z K-IUT-ami znajdują się w nawiasach.



Rys. 4. Przykładowe wyzwalacze aktualizacji indeksów

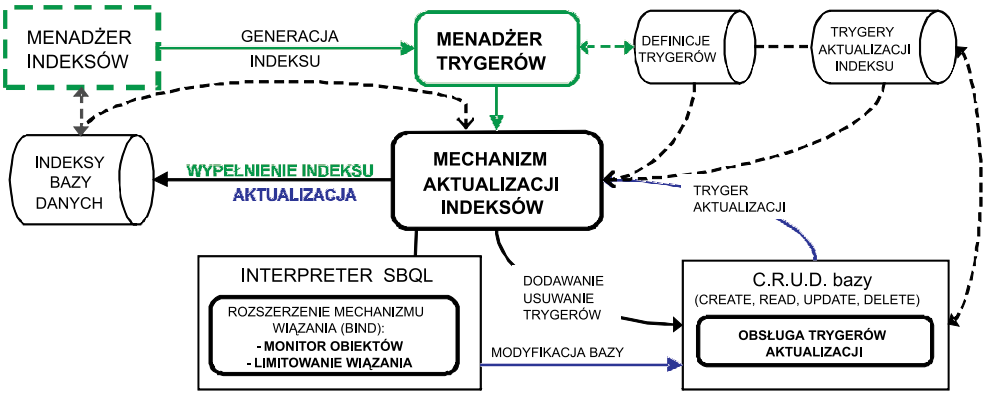
4.4. Proces aktualizacji indeksów

Architektura zaproponowanego rozwiązania procesu aktualizacji indeksów została zaprezentowana na rysunku 5.

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.



Rys. 5. Architektura silnika aktualizacji indeksów

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.

4.5. Interpreter SBQL oraz rozszerzenie wiązania

Znaczącym modułem, z którego korzysta mechanizm aktualizacji indeksów jest interpreter języka SBQL (pokazany również na rysunku 5) rozszerzony o możliwości:

– Rejestrowania obiektów bazy danych w czasie wykonywania wyrażenia wyznaczającego wartość kluczową. Rejestrowanie jest możliwe podczas wiązania nazw obiektów na ENVS (inne jednostki bazy danych jak procedury, widoki itp. oraz literały są pomijane) – ta funkcjonalność jest używana w celu lokalizacji wszystkich obiektów, które są albo powinny być wyposażone w K-IUT-y.

– Ograniczenia pierwszego przeprowadzonego wiązania jedynie do jednego wymienionego obiektu – ta właściwość znacząco przyspiesza i ułatwia weryfikację czy nowy potomek podobiektu dodany do obiektu z R-IUT lub NP-IUT powinien być wyposażony w NP-IUT czy NK-IUT, tj. czy nowy potomek jest obiektem niekluczowym czy potencjalnym pośrednim lub bezpośrednim rodzicem obiektu niekluczowego.

Podstawowe funkcje interpretera używanego przez mechanizm aktualizacji indeksów to:

– przemieszczanie się z korzenia bazy danych lub obiektów wyposażonych w NP-IUT-y do obiektów niekluczowych,
– generowanie wartości klucza dla obiektów niekluczowych.

Następny podrozdział prezentuje generyczność zaproponowanego podejścia na przykładzie dotyczącym wyzwalaczy K-IUT.

4.6. Przykład koncepcyjny

Poniższe przykłady dotyczą indeksu *idxEmpTotalIncomes* z kluczem opartym na metodzie *getTotalIncomes*(), która jest polimorficzna w zależności od klasy obiektu niekluczowego. Dla instancji klasy *EmpClass getTotalIncomes*() zwraca wartość atrybutu *salary*:

```
return deref(salary);
```

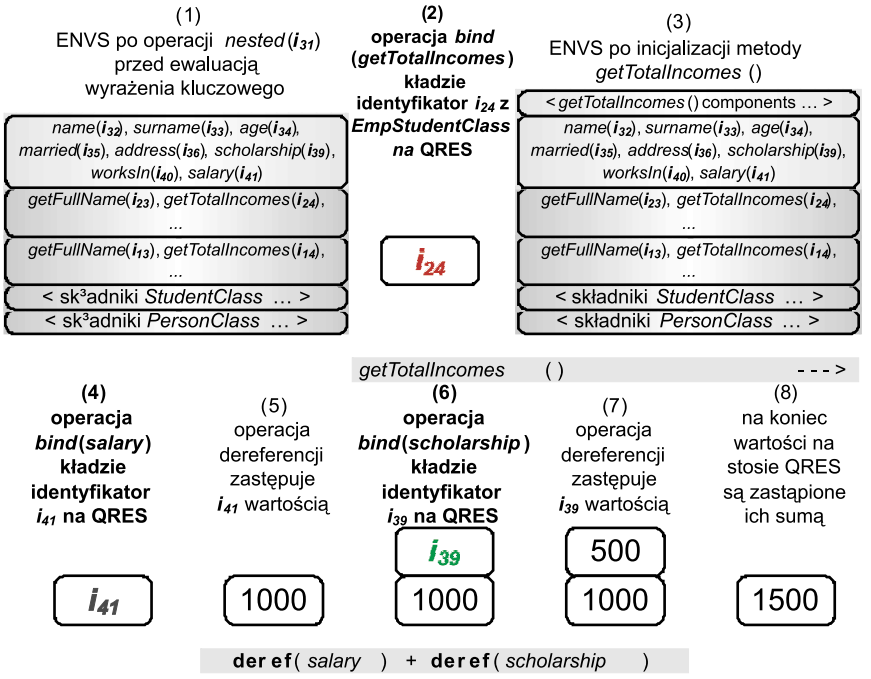
natomiast dla *EmpStudentClass* uwzględnia również atrybut *scholarship*:

```
return deref(salary) + deref(scholarship);
```

Prześledźmy jak moduł aktualizacji indeksów radzi sobie z instancją klasy *EmpStudentClass*. Poniższe wyrażenie aktualizuje atrybut *scholarship* obiektów *EmpStudent,* których wiek jest równy 30:

```
(EmpStudent where age = 30)
.setScholarship(1500);
```

Według przykładowego schematu na rysunku 3 w związku z wywołaniem metody *setScholarship*() zostanie zaktualizowany atrybut i_{39} *scholarship* obiektu i_{31} *EmpStudent*. Przed wykonaniem modyfikacji mechanizm aktualizacji indeksu znajduje K-IUT-a **< index:** *idxEmpTotalIncomes*, **obiekt niekluczowy:** i_{31} **>** powiązanego z atrybutem i_{39} *scholarship*. Obliczana jest wartość klucza dla indeksu *idxEmpTotalIncomes*. Z czynności przeprowadzonych przez interpreter i pokazanych na rysunku 6 wynika, że obiekty i_{24} *getTotalIncomes* (procedura obiektu *EmpStudentClass*), atrybut *salary* i_{41} i atrybut *scholarship* i_{39} miały wpływ na wartość klucza 1500. Identyfikator procedury może być odrzucony podczas rewizji wyzwalaczy aktualizacyjnych.



Rys. 6. Obliczanie wartości klucza indeksu *idxEmpTotalIncomes* dla obiektu i_{31} przed aktualizacją

Po aktualizacji atrybutu *scholarship* ponowne obliczenia wartości klucza są podobne do zaprezentowanych na rysunku 6 (jedynie wartości atrybutów są różne). W celu zakończenia operacji CRUD mechanizm aktualizacji indeksu:

- aktualizuje indeks *idxEmpTotalIncome*: wpis dla **niekluczowego obiektu i_{31}** jest zaktualizowany do nowej wartości 2500,
- aktualizuje *IUT-y* dla indeksu *idxEmpTotalIncomes* i **niekluczowego obiektu i_{31}**: przed jak i po modyfikacją w bazie mechanizm aktualizacji indeksów wykrył te same wyzwalacze *IUT*, dlatego żadne zmiany nie zostały wykonane.

5. Wnioski

Mechanizm aktualizacji indeksów zaimplementowany w prototypie SZOBD ODRA stanowi dowód opracowanej koncepcji. Jego ocena może zostać przeprowadzona na podstawie zasadniczych właściwości i jego wpływu na możliwości indeksowania bazy danych.

Niewątpliwą zaletą idei aktualizacji indeksów w relacyjnych i relacyjno-obiektowych bazach danych jest ekonomiczne użycie składu danych. Informacja niezbędna dla mechanizmów zapewniających spójność między danymi i indeksami jest powiązana z kolumnami

tabeli, ponieważ jest identyczna dla każdego wiersza. Przeciwnie w zaimplementowanym SZOBD ODRA wyzwalacze aktualizujące indeksy są w wielu przypadkach zapisane łącznie ze złożonymi obiektami oraz obiektami zawierającymi pojedyncze wartości. Taka sytuacja jest dopuszczalna biorąc pod uwagę ogromną ilość danych w pamięci operacyjnej (lub na dysku twardym) jaką w dzisiejszych czasach zarządzają bazy danych.

Zwiększanie wydajności również zależy od różnorodności wykorzystanych struktur indeksów oraz elastyczności podczas definiowania samych indeksów. Właściwości języka SBQL umożliwiają łatwe formowanie złożonych warunków selekcji (włączając użycie skomplikowanych wyrażeń z polimorficznymi metodami i operatorami agregującymi). Zaproponowana organizacja indeksowania w SZOBD ODRA zapewnia wszystkie potrzebne mechanizmy do tworzenia indeksów z kluczami opartymi na takich predykatach. Jak pokazano, najbardziej popularne SZBD nie pozwalają korzystać z podobnych możliwości i elastyczności. Co więcej, próby rozszerzenia mechanizmów indeksowania (Oracle *function-based indices*) prowadzą do niepotrzebnych strat efektywności modyfikacji danych, a nawet do błędów w otrzymywanych wynikach zapytań i błędnego działania aplikacji bazodanowych.

W przeciwieństwie do wszystkich zaimplementowanych rozwiązań problemu automatycznej aktualizacji indeksów zaprezentowanych w literaturze naukowej lub produktach 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. Można spierać się o zwiększony koszt przechowywania spowodowany przez wyzwalacze IUT; niemniej jednak, jak pokazano w [2] konserwacja indeksów zdefiniowanych jako złożone wyrażenia wymaga wprowadzenia dużej ilości dodatkowych danych do struktury indeksu. Inną zaletą stosowania autorskich wyzwalaczy IUT ustawionych na obiektach użytych do określenia wartości klucza jest to, że zawierają bezpośrednie referencje do indeksowanych obiektów, natomiast inne rozwiązania są często zmuszone identyfikować je pośrednio (np. przez metody wstecznej nawigacji) [2, 7].

Zaprezentowane rozwiązanie aktualizacji indeksów w SZOBD jest generyczne i uniwersalne, jednakże wymaga optymalizacji w celu uniknięcia niepotrzebnych strat efektywności szczególnie w przypadkach prostych aktualizacji. Takie optymalizacje zostały opracowane i częściowo zaimplementowane, np.:

- unikanie ponownego obliczania całego wyrażenia kluczowego dla wyrażeń ścieżkowych,
- opóźnianie działań konserwacyjnych (leniwa aktualizacja indeksu).

Literatura

[1] 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, 2008, 179–197.

[2] Bertino E. *et al*., *Indexing Techniques for Advanced Database Systems*. Kluwer Academic Publishers, Boston Dordrecht London, 1997.
[3] Cattell R.G.G., Barry D.K. (Eds.), *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000.
[4] db4objects Inc., *db4o Tutorial for Java*. Production Release V6.3, 2008.
[5] Elmasri R., Navathe S.B., *Fundamentals of Database Systems 4th ed.* Pearson Education, Inc., ISBN: 83-7361-716-7, 2004.
[6] GemStone, *GemFire Enterprise Developer's Guide*, Version 5.7, 2008.
[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, ACM 1997, ISBN 0-89791-970-X, 1997, 136–143.
[8] Hwang D.J., *Function-based indexing for object-oriented databases*. Massachusetts Institute of Technology, 1994 (Ph.D. thesis).
[9] IBM®, *Information Center*. version 6, 1st edition: http://publib.boulder.ibm.com/infocenter/, 2008.
[10] Kowalski T.M., Wislicki 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, 2008, 97–117.
[11] Litwin W., *Linear Hashing : a new tool for file and tables addressing*. Reprinted from VLDB-80 in READINGS IN DATABASES. 2-nd ed., Morgan Kaufmann Publishers, Inc., Stonebraker M. (Ed.), 1994.
[12] Litwin W., Nejmat M.A., Schneider D.A., *LH*: Scalable, Distributed Database System.* ACM Trans. Database Syst., 21(4), 1996, 480–525.
[13] Microsoft, *SQL Server 2008 Books Online*.: http://msdn.microsoft.com/en-us/library/ms130214.aspx, 2008.
[14] Objectivity, *Objectivity/SQL++*. Part Number: 93-SQLPP-0, Release 9.3, 2006.
[15] Płodzień J., *Optimization Methods In Object Query Languages*. IPIPAN, Warszawa, 2000 (Ph.D. Thesis).
[16] Progress Software Corporation, *ObjectStore Java API User Guide.* ObjectStore, Release 7.1 for all platforms, 2008.
[17] SBA & SBQL Web pages: http://www.sbql.pl/.
[18] Strohm R. *et al*., *Oracle® Database Concepts*. 11g Release 1 (11.1), Part Number B28318-05.
[19] Subieta K, Adamus R., Habela P., Kaczmarski K., Lentner M., Stencel K., 2008. *Stack-Based Architecture and Stack-Based Query Language*. ICOODB 2008, Berlin, available online at odbms.org portal: http://www.odbms.org/download/030.02%20Subieta%20Stack-Based%20Architecture%20and%20Stack-Based%20Query%20Language%20March%202008.PDF, 2008.
[20] Versant, *VERSANT Database Fundamentals Manual.* (Release 7.0.1.0) 2005.
[21] Objectivity, *Objectivity/SQL++*. Part Number: 93-SQLPP-0, Release 9.3, 2006.
[22] Płodzień J., *Optimization Methods In Object Query Languages*. IPIPAN, Warszawa, 2000 (Ph.D. Thesis).
[23] Progress Software Corporation, *ObjectStore Java API User Guide.* ObjectStore, Release 7.1 for all platforms, 2008.