

# Badanie wydajności Oracle 9i w kontekście modelu obiektowego

*Bartłomiej Jabłoński*

Uniwersytet Łódzki

Model obiektowo-relacyjny danych pojawił się już w wersji Oracle 8i. Ze względu na małe możliwości nie był szeroko stosowany w aplikacjach. Wersja Oracle 9i zawiera wiele nowych rozszerzeń i rozwiązań, które pozwalają na poważne rozważenie ich użycia w aplikacjach przemysłowych. Pozostają tylko pytania: Jak wspomniana „nakładka” obiektowości wpływa na wydajność? Czy zamodelowanie wymagań użytkownika z wykorzystaniem obiektowych możliwości serwera bazy danych przyspieszy dostęp do danych, czy wręcz przeciwnie, spowoduje, że dane będą trudniej osiągalne? Jaki jest ostateczny zysk użytkownika końcowego w zastosowaniu nowoczesnej technologii implementacji obiektowej struktury danych?

Autor przeprowadził analizę porównawczą modelu relacyjnego i obiektowego kilku typowych sytuacji w programowaniu bazy danych Oracle 9i. W referacie znajdują się wyniki badań wydajności oraz omówienie kilku technik strojenia modelu obiektowego.

## Informacja o autorze:

Bartłomiej Jabłoński od 1999 roku pracuje na stanowisku asystenta na Wydziale Matematyki Uniwersytetu Łódzkiego, gdzie zajmuje się głównie bazami danych Oracle oraz standardem XML, ze szczególnym uwzględnieniem jego zastosowań w bazach danych. Od 1994 roku wielokrotnie uczestniczył w projektach informatycznych związanych z bazami danych Oracle jako konsultant. Od 1995 prowadzi szkolenia w Centrum Edukacyjnym Oracle Polska. Jest autorem trzech i tłumaczem czterech książek z zakresu informatyki.

## 1. Wstęp

Obiektowość stała się standardem w modelowaniu wymagań użytkownika. Rośnie liczba profesjonalnych narzędzi, w których pojęcia obiektowe są automatycznie konwertowane na kod źródłowy, ułatwiając stworzenie dużych aplikacji w stosunkowo krótkim czasie. Łatwiejsze stało się też utrzymywanie systemu zbudowanego z komponentów-obiektów. Bazy danych długo opierały się tej tendencji. Dopiero od wersji 8i, firma Oracle wyposażyła serwer bazy danych w pierwsze możliwości obiektowe. Niektóre moduły (np. *spatial*) można było wykorzystywać zarówno w wersji relacyjnej jak i relacyjno-obiektowej. Ta ostatnia jednak, mimo prostszego zapisu zapytań SQL, posiadała wiele ograniczeń. Te właśnie ograniczenia powodowały, że w praktyce trudno było znaleźć wdrożoną aplikację, w której zastosowany byłby model obiektowy.

Wersja Oracle 9i została wzbogacona o szereg nowych opcji, np.:

- definiowanie własnych konstruktorów,
- dziedziczenie,
- modyfikowanie typów on-line,
- możliwość przekazywania obiektowych parametrów procedurom zewnętrznym,
- możliwość zagnieżdżania kolekcji,
- możliwość definiowania ciała klasy za pomocą języka Java.

Rosnące możliwości implementowania wymagań użytkownika za pomocą obiektów mogą rodzić pytanie, czy warto zacząć korzystać z tego „wynalazku”. W artykule tym autor zajmuje się aspektem wydajności zapytań SQL realizowanych na tablicach przechowujących obiekty.

W literaturze opisano wiele testów wydajnościowych (benchmarków) porównujących różne bazy danych. Benchmarki te pozwalają podejmować decyzje dotyczące wyboru konfiguracji sprzętowo-programowej. W niniejszym artykule opisano proste testy porównujące wydajność zapytań na strukturach relacyjnych i obiektowych w jednej bazie danych – Oracle 9iR2.

Obiektowość w bazie danych Oracle realizowana jest na poziomie słownika. Dane przechowywane są tradycyjnie w postaci relacyjnej lub w postaci danych upakowanych LOB. Poza fazą parsowania (w niniejszym eksperymencie zapytania były za mało skomplikowane, żeby dawało się zmierzyć różnice w czasie analizy składniowej) nie powinno to wpływać na wydajność, o ile dane będą fizycznie przechowywane w podobnej ilości bloków. Tak generalnie nie jest, o czym można się przekonać z dalszej części tego artykułu.

Oczywiście analiza zysków i kosztów stosowania modelu obiektowego nie kończy się na badaniu wydajności zapytań. Dochodzi do tego: łatwość implementacji wymagań użytkownika, łatwość wprowadzania zmian w późniejszym terminie, zastosowanie bardziej zaawansowanych (i droższych) narzędzi, przeszkolenie zespołu projektowego i administratora, itd...

## 2. Obiektowy model danych w Oracle 9i

### Podstawowe definicje

W Oracle 8i wprowadzono, a w wersji 9i rozszerzono możliwość definiowania własnych typów danych przez użytkownika. Poleceniem `CREATE TYPE` można utworzyć własną klasę, która może składać się z pól i metod. Pola mogą być zdefiniowane w oparciu o typy proste (predefiniowane w serwerze) lub w oparciu o wcześniej przygotowane typy własne. Jeżeli doda się jeszcze do tego możliwość definiowania pól typu wskaźnikowego, to otrzymuje się model danych, który

przy zachowaniu kompatybilności z modelem relacyjnym daje zupełnie nowe narzędzie do opisywania wymagań użytkownika. Metody klas można oprogramowywać przy pomocy wbudowanych języków: PL/SQL, Java lub zewnętrznych języków kompilowanych, np. C.

W jądrze serwera „zaszyto” też obsługę kolekcji obiektów w postaci tablic o zadeklarowanej maksymalnej wielkości (VARRAY) oraz tablic nieograniczonej wielkości z pełnym wsparciem SQL-owym (NESTED TABLE).

Obiekty zadeklarowanych klas można następnie przechowywać w tabelach. Istnieją dwa sposoby implementacji obiektów: pola obiektów mogą być przechowywane jako relacyjne pola rekordów lub, jeśli obiekty przechowywane są wewnątrz kolekcji, cała zawartość obiektów jest przechowywana jako pole typu LOB. W tym ostatnim przypadku, istnieje możliwość wskazania, czy pole to ma być przechowywane w segmencie tabeli razem z resztą pól czy w oddzielnym segmencie.

Każdy obiekt przechowywany w bazie danych zaopatrywany jest w wewnętrzny identyfikator OID, który służy do rozróżniania obiektów oraz jako swoistego rodzaju „klucz obcy” we wzajemnych odwołaniach między obiektami. W zależności od sytuacji wielkość identyfikatora OID może wynosić 8 lub 16 bajtów, może też być oparty o kolumnę klucza głównego definiowanego przez użytkownika.

Informację o obiektach warto jeszcze uzupełnić o fakt, że dane obiektowe można indeksować i nakładać na nie więzy, w tym również specjalne, nie występujące w modelu relacyjnym. Definicję klasy można oprzeć o poprzednią definicję (dziedziczenie) deklarując w ten sposób bardziej specjalizowaną klasę.

## Implementacja obiektów

Szczegóły implementacyjne deklarowanych typów obiektowych mają znaczący wpływ na wydajność realizowanych zapytań. Do wyjaśnienia tych szczegółów posłużą przykładowe dwie klasy:

```
create type T1 is object (
  a number(10)
);
/
create type T2 is object
  a number(10),
  b T1
);
/
```

Do przechowywania obiektów tych klas można użyć tabel obiektowych (*TB11*, *TB21*) lub tabel, w których pola są typu obiektowego (*TB12*, *TB22*).

```
create table TB11 of T1;
create table TB12 (c T1);

create table TB21 of T2;
create table TB22 (c T2);
```

Dla porównania zdefiniowano jeszcze dwie tabele typowo relacyjne (*TB01*, *TB02*):

```
create table TB01 (
  a number(10)
);

create table TB02 (
  a number(10),
```

```

b number(10)
);

```

W poniższym zestawieniu przedstawiono mapowanie struktur obiektowych na relacyjne. Wiele z tych kolumn jest dodawana automatycznie przez serwer i nie pojawia się użytkownikom jako rezultat polecenia `DESC`. Niektóre z nich można jednak umieścić jawnie na liście wyrażeń zapytania `SELECT`. Pełna ich lista jest dostępna w systemowej tabeli słownikowej `COL$`.

Tabela 1. Mapowanie kolumn dla tabel ze strukturami obiektowymi

Nazwa tabeli	Nazwa kolumny	Wewn. typ danych	Wielkość (w bajtach)	Wielkość rekordu (w bajtach)
TB01	A	number	22	22
TB02	A	number	22	44
	B	number	22	
TB11	SYS_NC_ROWINFO\$	object	1	39
	SYS_NC_OID\$	raw	16	
	A	number	22	
TB12	C	object	1	17
	SYS_NC00002	raw	16	
TB21	SYS_NC_ROWINFO\$	object	1	62
	SYS_NC_OID\$	raw	16	
	A	number	22	
	B	object	1	
	SYS_NC00005\$	number	22	
TB22	C	object	1	45
	SYS_NC00002\$	number	22	
	SYS_NC00003\$	number	22	

W tabelach obiektowych występuje automatycznie dodawana, ukryta kolumna `SYS_NC_ROWINFO$`. Jest to symboliczna kolumna reprezentująca konstruktor przechowywanego obiektu.

Kolumna `SYS_NC_OID$` jest kolumną, w której przechowywany jest unikalny w ramach całej bazy (na ogół) identyfikator. To właśnie wartość z tej kolumny jest później przechowywana w kolumnach typu `REF`. Zostaje ona dokładana do każdej tabeli obiektowej. Należy zauważyć, że dla tabel, w których obiekty przechowywane są w polach taki identyfikator nie jest dodawany. Identyfikator ten w podanym wyżej przykładzie zajmuje 16 bajtów, ale już deklaracja:

```

create table TB11a of T1 (a primary key)
object identifier is primary key;

```

powoduje, że jego wielkość rośnie do 70 bajtów!

Kolumny `SYS_NCxxxx$` to wirtualne kolumny przechowujące zawartości pól rekordów. W dokumentacji technicznej Oracle można znaleźć następujące zdanie: „Serwer automatycznie mapuje złożoną strukturę typów obiektowych na prostą, płaską strukturę tabel”. Czyli zapytanie:

```
select value(p).b.a from tb21 p where a=13;
```

jest, na podstawie danych znajdujących się w słowniku zamieniane de facto na:

```
select sys_nc00005$ from tb21 where a=13;
```

Mogłoby się zatem wydawać, że jedyny „obiektywny” narzut czasowy, to czas związany z przemapowaniem nazw kolumn, czas analizy składniowej zapytania. Niestety tak nie jest. Warto zwrócić szczególną uwagę na rozmiary rekordów w poszczególnych tablicach. Widać, że rozszerzone, w stosunku do modelu relacyjnego, możliwości obiektowe okupione zostały powiększeniem wielkości rekordu. W przypadku pełnych odczytów tabeli (full scan) czas odczytu jest proporcjonalny do wielkości rekordu.

## Implementacja kolekcji

Kolekcje to zbiory obiektów tego samego typu (lub obiektów posiadających wspólnego przodka) dające się przechowywać w pojedynczych polach tabel lub innych obiektów. Istnieją dwie klasy implementujące cechy kolekcji: tablice ograniczone (VARRAY) i tablice zagnieżdżone (NESTED TABLE). Poniżej zdefiniowano klasy kolekcji dla obiektów zdefiniowanych wcześniej oraz tabele, w których te klasy będą składowane:

```
create type t1va is varray(20) of t1;
/
create type t1nt is table of t1;
/
create table tbk_va (a number(10), b t1va);
create table tbk_nt(a number(10), b t1nt) nested table b store as tbk_ntab;
```

Jak widać definiowanie tabeli, w której polem jest tabela zagnieżdżona wymaga dodatkowej klauzuli definiującej segment, gdzie fizycznie mają się znajdować zagnieżdżone dane. Dla kolekcji VARRAY dodatkowa klauzula jest opcjonalna i pozwala umieścić ją fizycznie w oddzielnym segmencie. Domyślnie w segmencie tabeli przechowywane jest maksymalnie 4KB danych. Jeśli wielkość kolekcji przekracza 4KB lub inną zdefiniowaną przez użytkownika wielkość, dane są przenoszone do oddzielnego segmentu automatycznie.

Wiązanie relacyjne pomiędzy rekordami tabel *TBK\_NT* i jej tabeli zagnieżdżonej odbywa się za pomocą automatycznie generowanych identyfikatorów przechowywanych w kolumnach *SYS\_NCxxxxyyy\$* i *NESTED\_TABLE\_ID*.

Wielkość rekordu w *TBK\_VA* wskazuje, że pamięć pod deklarowane w typie 20 elementów została już przydzielona - nie jest to sprzyjająca okoliczność dla operacji pełnego odczytu (full scan). W przypadku tabeli zagnieżdżonej pamięć jest alokowana w miarę potrzeb (na takich samych zasadach jak w tabelach relacyjnych alokowana jest pamięć na potrzeby przechowywania rekordów).

Tabela 2. Wykaz wszystkich kolumn dla tabel z kolekcjami

Nazwa tabeli	Nazwa kolumny	Wewn. typ danych	Wielkość (w bajtach)	Wielkość rekordu (w bajtach)
TBK_VA	A	number	22	659
	B	object-varray	637	
TBK_NT	A	number	22	54
	B	object-nested table	16	
	SYS_NC00020003\$	raw	16	

Nazwa tabeli	Nazwa kolumny	Wewn. typ danych	Wielkość (w bajtach)	Wielkość rekordu (w bajtach)
TBK_NTAB	NESTED_TABLE_ID	raw	16	39
	SYS_NC_ROWINFO\$	object	1	
	A	number	22	

### Konwersja modelu relacyjnego do obiektowego

W modelu relacyjnym istnieje tylko jeden rodzaj powiązania pomiędzy danymi: więź klucza obcego zwany powszechnie *relacją* lub związkim jeden-do-wielu. Otrzymuje się ją poprzez jawną specyfikację odpowiednich kolumn oraz poprzez przechowywanie w nich odpowiednich wartości. Łączenie powiązanych relacją danych pochodzących z różnych tabel wymaga od użytkownika umieszczenia w zapytaniu predykatu łączenia, to z kolei powoduje, że użytkownik musi być świadomy tego, jak technicznie realizowana jest ta funkcja bazy danych.

Dla danych obiektowych przewidziano uproszczony sposób definiowania „relacji”. Większość identyfikatorów obiektów generowana jest automatycznie; łączenie danych może zachodzić poprzez odpowiednio zbudowane wyrażenie na liście wyrażen `SELECT`-ta. Innymi słowy, fizycznie dane łączone są również na podstawie równości identyfikatora i wskaźnika, ale szczegóły implementacyjne zostały ukryte przed użytkownikiem. Co więcej, przy zachowaniu unikalności identyfikatora obiektu w ramach całej bazy danych, można definiować wskaźniki na obiekty przechowywane niekoniecznie w jednej tabeli.

W momencie wprowadzenia wskaźników i możliwości deklarowania występowania obiektów wewnątrz innych obiektów, relacyjny związek jeden-do-wielu może być wyrażony obiektowo na wiele sposobów.

**Wiązanie przez wskaźnik** polega na tym, że w tabeli *master* umieszcza się obiekty posiadające własny OID, zaś w tabeli *master* umieszcza się kolumnę typu REF. W kolumnie tej mogą być przechowywane obiekty zadeklarowanej klasy lub wszystkich klas od niej potomnych. Wskaźniki mogą dotyczyć obiektów przechowywanych niekoniecznie w jednej tabeli – jest to zatem znaczne rozszerzenie pojęcia zależności między danymi.

**Wiązanie przez zagnieżdżenie** polega na wskazaniu miejsca, gdzie dane mają być fizycznie składowane. W bazie relacyjnej, wartości były przechowywane w polach rekordu, który miał strukturę liniową. W bazie obiektowej, można korzystać z kolekcji, która pozwala na wyjście w większą ilość wymiarów, a co za tym idzie, powiązane obiekty już nie muszą być przechowywane w oddzielnych strukturach.

## 3. Porównywanie wydajności modelu relacyjnego i obiektowego

Ponieważ implementacja modelu obiektowego i relacyjnego różnią się, należy spodziewać się też różnic w planie wykonania zapytań oraz w czasie ich realizacji. W przeprowadzonym eksperymencie badano szybkość wykonywania się zapytań na strukturach relacyjnych i obiektowych implementujących te same wymagania użytkownika.

Badaną wielkością była liczba logicznych odczytów bloków. Choć na wykresach podane są bezwzględne wyniki, to należy pamiętać, że mogą być one inne na inaczej skonfigurowanych serwerach. Celem badań było poznanie wzajemnych stosunków wydajności poszczególnych rozwiązań implementacyjnych w ramach danego, jednego serwera.

## Testowa baza danych

Eksperyment wykonano na serwerze Oracle 9.2.0.1 Enterprise Edition w środowisku MS Windows 2000. Tworzone tabele umieszczane były w przestrzeni tabel, gdzie rozmiar bloku wynosił 8KB, zaś wielkość bufora danych ok. 50 MB. Model danych, na którym wykonywano eksperyment składał się z dwóch encji: *student* i *ocena* w układzie *master-detail*:



Rys. 1. Model danych bazy testowej

Do bazy wpisano dane 10000 studentów oraz każdemu z nich przypisano dokładnie po 20 ocen, każda z innego przedmiotu. Pola tekstowe były wypełniane napisami o losowej długości. Dane były wpisywane w losowej kolejności tak, aby na przykład oceny tego samego studenta lub studentki o wartościach identyfikatora bliskich sobie, nie były umieszczone w tym samym bloku.

Model ten był następnie implementowany na kilka sposobów. Jako implementacja odniesienia został uznany model relacyjny (*RL*), złożony z dwóch tabel; w tabeli *OCENY* została dołożona dodatkowa kolumna *fk\_student* do reprezentowania relacji pomiędzy danymi.

Badanymi zapytaniami były proste zapytania *SELECT* reprezentatywne dla szerokiej klasy problemów spotykanych w praktyce:

- Test 1. Wybór danych z wszystkich rekordów tabeli *master*,  

```
select * from studenci;
```
- Test 2. Wybór danych z wszystkich rekordów tabeli *detail* dla danego rekordu tabeli *master*,  

```
select * from oceny where fk_student=:b1
```
- Test 3. Wybór danych z jednego rekordu tabeli *detail* dla danego rekordu tabeli *master*,  

```
select o.*
  from oceny o
 where fk_student = :b1
       and kodprzed = :b2;
```
- Test 4. Wybór danych ze wszystkich rekordów tabeli *detail* dla wszystkich rekordów tabeli *master*.  

```
select s.*, o.*
  from studenci s, oceny o
 where s.id = o.fk_student
```

Początkowo pomiary zostały wykonane na tabelach bez indeksów, następnie dodano indeksy, a w końcowej fazie starano się zoptymalizować zapytanie pod kątem zminimalizowania liczby odczytywanych bloków.

Poprzez wybór wszystkich danych z rekordu (`select *`) wyeliminowano przypadek, gdy optymalizator pominie dostęp do tabeli, wykorzystując tylko indeks. Zapytania musiały być modyfikowane w zależności od struktury danych, na których były stosowane.

W pierwszym i ostatnim zapytaniu optymalizator nie będzie miał innego wyboru, jak wybrać opcję *full scan*. Zatem w szybkości realizacji zapytania, będzie miała największy wpływ zwartość danych. Im więcej danych da się upakować w jednym bloku, tym mniej bloków trzeba będzie odczytać, tym krótszy czas odpowiedzi. Test drugi i trzeci stawiają duże pole do popisu dla osób, które zajmują się strojeniem SQL-a. W wyborze konkretnego rekordu najczęściej najlepsze rezultaty uzyskuje się poprzez umiejętne zastosowanie indeksów.

Zapytania były wykonywane dla struktur danych opisanych w tabeli 3.

Tabela 3. Lista zastosowanych modeli

Skrót	Opis
<b>RL</b>	Model odniesienia. Obie tabele są relacyjne. W tabeli <i>detail</i> umieszczono dodatkową kolumnę <i>fk student</i> realizującą warunek łączenia.
<b>RO</b>	Tabela <i>detail</i> jest relacyjna, tabela <i>master</i> jest tabelą obiektową. W tabeli <i>detail</i> znajduje się kolumna <i>fk student</i> typu REF wskazująca na adres <i>OID</i> rekordu <i>master</i> .
<b>ROR</b>	Tabela <i>detail</i> jest relacyjna, tabela <i>master</i> zawiera jedną kolumnę, w której mieszczą się dane w postaci obiektów.
<b>OAI</b>	Tabela <i>master</i> zawiera kolumnę typu VARRAY mieszczącą rekordy <i>detail</i> . Tablica VARRAY została ograniczona do 20 elementów.
<b>ON</b>	Tabela <i>master</i> zawiera kolumnę typu NESTED TABLE mieszczącą rekordy <i>detail</i> .

W tabeli 4. znajduje się porównanie średnich rozmiarów struktur danych.

Tabela 4. Wielkości struktur danych testowych (w bajtach)

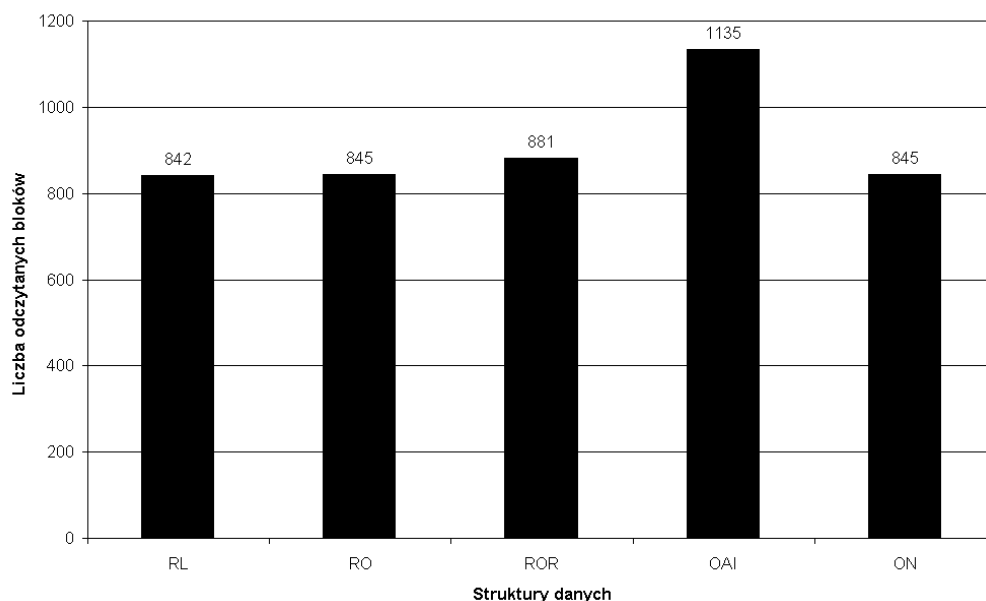
	<i>STUDENCI</i>		<i>OCENY</i>	
	śr. dług. rek. (w bajtach)	liczba bloków	śr. dług. rek. (w bajtach)	liczba blo- ków
<b>RL</b>	100	180	13	432
<b>RO</b>	117	180	46	1378
<b>ROR</b>	102	180	13	432
<b>OAI</b> <sup>1</sup>	312	496	-	-
<b>ON</b>	117	180	26	810

<sup>1</sup> – dane dla ocen mieszczą się w tabeli *studenci*.

## Test 1.: Pełny przegląd tabeli *master*

Zapytanie wybierające wszystkie rekordy z tabeli *master* to jedno z częstszych zapytań prezentujących listę kategorii, podsumowań lub, jak w tym przypadku, listę wszystkich studentów. Ponieważ dane pobierane są z jednej tabeli i zapytanie wybiera wszystkie rekordy, nie ma potrzeby

zakładania indeksów. Wyniki testu dla zapytania: `select * from studenci;` zaprezentowane są na wykresie 1.



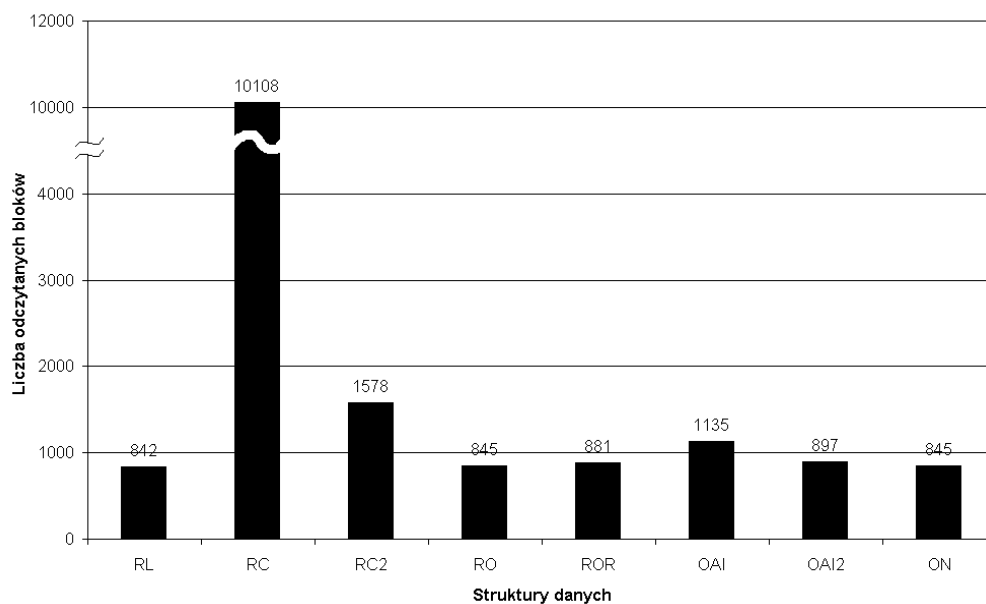
Wykres 1. Liczby logicznych odczytów dla pełnego przeglądu tabeli *master*.

Z powyższego doświadczenia wynika, że różnice występujące pomiędzy strukturą typowo relacyjną, a różnymi strukturami obiektowymi nie różnią się znacząco. Jedynym wynikiem odbiegającym od średniej jest przeszukiwanie tabeli *OAI*, w której dane są przechowywane jako kolekcja *VARRAY*. Jest to jednocześnie tabela, która przechowywana jest w największej liczbie bloków (patrz tabela 4).

Istnieje możliwość przechowywania danych *LOB* (a więc i *VARRAY*) w oddzielnym segmencie. W segmencie tabeli przechowywany jest wtedy tylko specjalny wskaźnik, a więc przy odczycie *full scan* oszczędza się na ilości odczytanych bloków. Dla takiej struktury (*OAI2*) stwierdzono 897 odczytanych bloków, czyli ciągle najwyższy, ale już tak nie odbiegający od reszty, wynik.

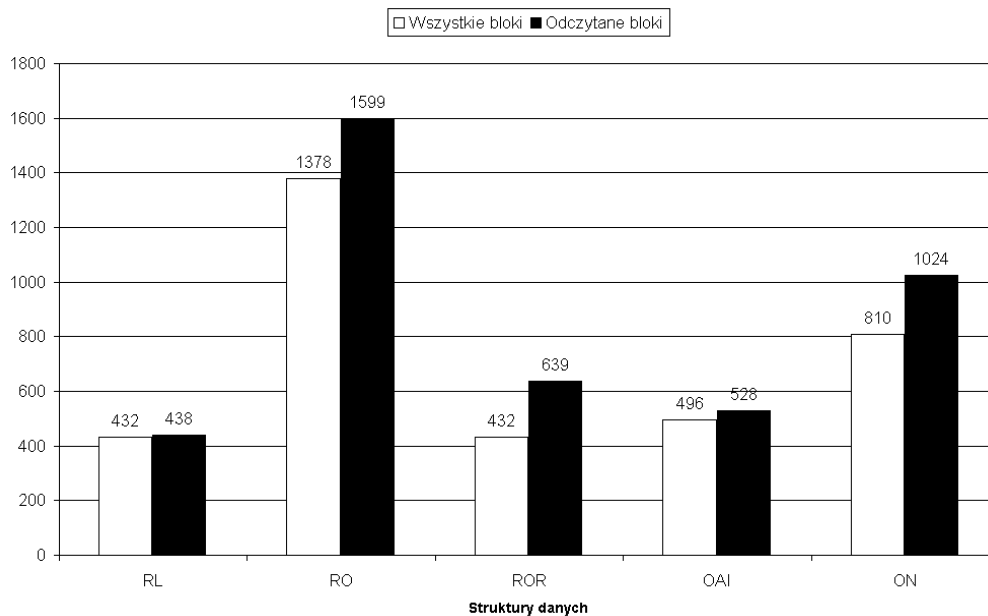
Struktura *OAI* przypomina nieco strukturę klastra (cluster). W niej również tuż obok danych rekordu *master* przechowywane są dane rekordów *detail*. Dla klastra (*RC*), gdzie każdy student jest pozycją w indeksie klastrowym uzyskano wynik 10108. Dla klastra hashowanego (*RC2*) zoptymalizowanego tak, aby 500 studentów przypadało na jedną pozycję indeksu (uzyskuje się wtedy lepszy współczynnik wypełnienia bloków), wynik był zdecydowanie lepszy – 1578.

Na wykresie 3. powtórzono zestawienie wyników z wykresu 2. uzupełnionego o 3 dodatkowe struktury *RC*, *RC2* i *OAI2*.

Wykres 2. Liczby logicznych odczytów dla pełnego przeglądu tabeli *master*

## Test 2. Wszystkie rekordy tabeli *detail* dla danego rekordu tabeli *master*.

Zapytanie wybierające wszystkie podrzędne rekordy przydaje się, gdy istnieje potrzeba poznania wszystkich cech konkretnej osoby, kategorii... Dla zapytania: `select * from oceny where fk_student=:b1`; uzyskane wyniki zaprezentowano razem ze średnimi wielkościami rekordów tabeli *detail*.



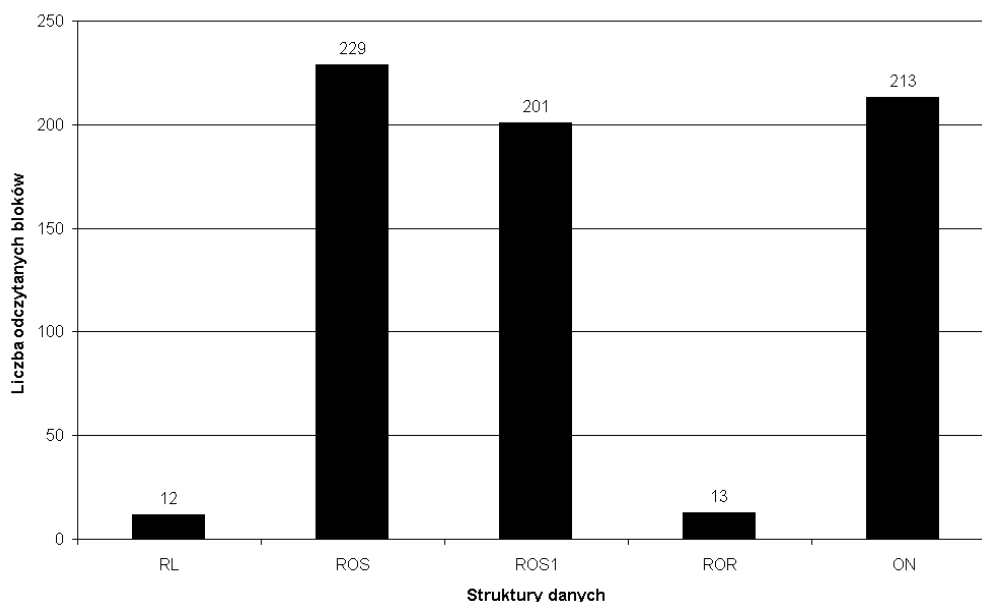
Wykres 3. Średnie wielkości rekordów oraz liczby logicznych odczytów.

Ponieważ nie utworzono indeksów, jedyną możliwością wyszukania odpowiednich rekordów tabeli *detail* jest pełny przegląd tej tabeli. Liczby logicznych odczytów powinny być zatem, podobnie jak w teście 1. skorelowane ze średnią wielkością rekordów.

Na komentarz zasługują dwa wyniki znacznie odbiegające od „relacyjnego optimum”; są to: *RO* i *ON*. Pierwszy z nich wynika najprawdopodobniej z tego, iż połączenie danych pomiędzy rekordem tabeli *master* i *detail* następuje na drodze porównania wewnętrznego identyfikatora (*REF*). W pozostałych strukturach (poza *ON*) identyfikator studenta jest przechowywany w tabeli *detail*, a zatem tabela *master* nie musi być w ogóle czytana. Dodatkowym czynnikiem jest sam rozmiar takiego identyfikatora, który dla *RO* wynosi 36 bajtów.

Oczywiście, przy podobnych wyszukiwaniach wydajność zapytań poprawia się po dodaniu indeksu na kolumnę klucza obcego (lub kolumnę przechowującą wskaźnik). W przypadku modelu *RO* oznacza to nałożenie indeksu na kolumnę typu *REF*. Jest to dozwolone tylko wtedy, gdy na wskaźnik założono więź *SCOPE* ograniczający wskazywany zakres tylko do jednej tabeli. W przypadku modelu z tabelą zagnieżdżoną sprawa jest bardziej skomplikowana, gdyż identyfikator łączący dane został automatycznie dodany przez serwer w momencie tworzenia tablicy. Na szczęście jest on dostępny w postaci kolumny o nazwie *nested\_table\_id*, na którą można nakładać własne indeksy (nie jest on zakładany automatycznie).

Po założeniu wspomnianych indeksów wyniki wyglądają następująco:



Wykres 4. Liczby logicznych odczytów bloków dla wszystkich rekordów zaindeksowanych tabel *detail*

Z powyższego wykresu wynika, największy przyrost wydajności uzyskano dla struktury relacyjnej *RL* oraz *ROR*. Pozostałe struktury do zrealizowania zapytania wymagają sięgnięcia do tabeli *master*, czyli realizowany jest algorytm złączenia danych – stąd tak duże liczby odczytanych bloków.

Dla struktury *ROS* (z więzłem *SCOPE*) uzyskano pewien dodatkowy zysk poprzez dodanie klauzuli *WITH ROWID* (*ROS1*) do kolumny wskaźnika *fk\_student* (wielkość wskaźnika rośnie wtedy do 42 bajtów).

Powyższy wykres pokazuje również, że założenie indeksu na kolumnie *nested\_table\_id* w tabeli zagnieżdżonej prawie pięciokrotnie przyspiesza wyszukiwanie danych. Jednak wykorzystania indeksu można się tylko domyślać, gdyż nie wskazuje na to plan wykonania:

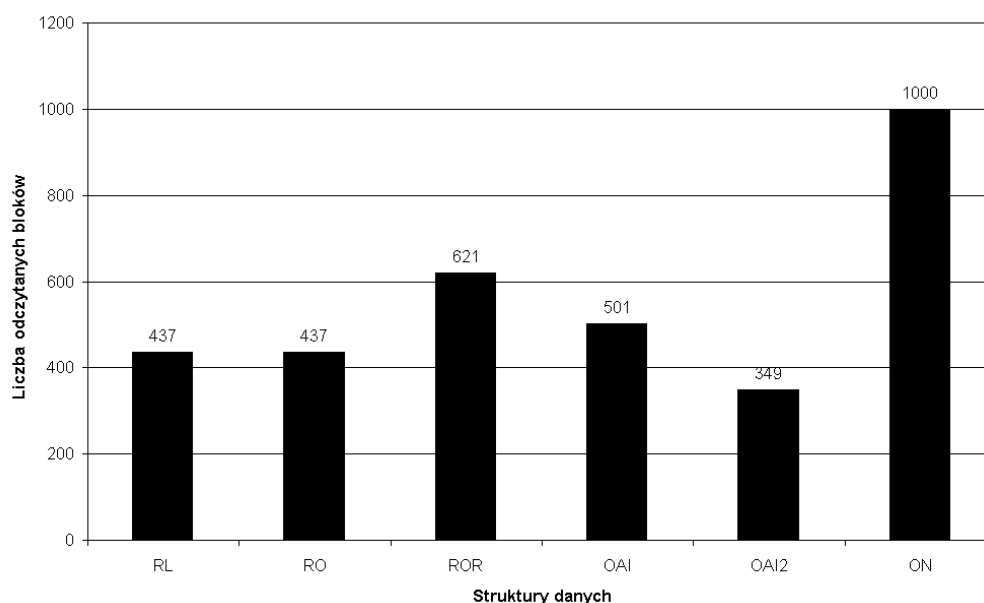
Plan wykonania

```
-----
0      SELECT STATEMENT Optimizer=CHOOSE
1      0      TABLE ACCESS (FULL) OF 'STUDENCI'
```

### Test 3. Jeden rekord tabeli *detail* dla danego rekordu tabeli *master*.

Dla wyszukania konkretnej cechy dla konkretnego obiektu należy użyć poniższego zapytania. Wyniki testu dla tabel bez założonych indeksów są przedstawione na wykresie 5.

```
select o.*
  from oceny o
 where fk_student = :b1
    and kodprzed = :b2;
```



Wykres 5. Liczba logicznych odczytów dla jednego rekordów tabeli *detail*.

Duża liczba logicznych odczytów dla struktury *ON* spowodowana jest tym, że do odnalezienia właściwego rekordu tabeli *detail* wymagany jest też odczyt z tabeli *master*.

Wynik dla *OAI2* jest lepszy od „relacyjnego” *RL*. Można to tłumaczyć tym, że rekordy tabeli *detail* w rzeczywistości nie są kolejnymi rekordami, tylko mieszczą się w jednej strukturze LOB-a. Po założeniu indeksu wynik dla *RL* będzie jednak zdecydowanie lepszy.

Po założeniu odpowiednich indeksów otrzymujemy rezultaty zaprezentowane na wykresie 6. (oczywiście struktura *RO* musi być zmieniona na *ROS*, ew. *ROSI*):

Nie da się poindeksować struktury *OAI*, ale można zastąpić *kod\_przed* pozycją w kolekcji VARRAY. Wtedy po dopisaniu odpowiedniej metody dostępu otrzymujemy *OAI3*. Wynik 281 logicznych odczytów jest lepszy niż wynik dla struktur *OAI* lub *OAI2*.

```
create type t_oceny_oai31 as varray(20) of number(2);
/

create or replace type t_oceny_oai32 as object(
  oc t_oceny_oai31,
  member function getvalue(pi number) return number,
  pragma restrict_references
    (getvalue, wnds, rnds, wnps, rnps)
);
/

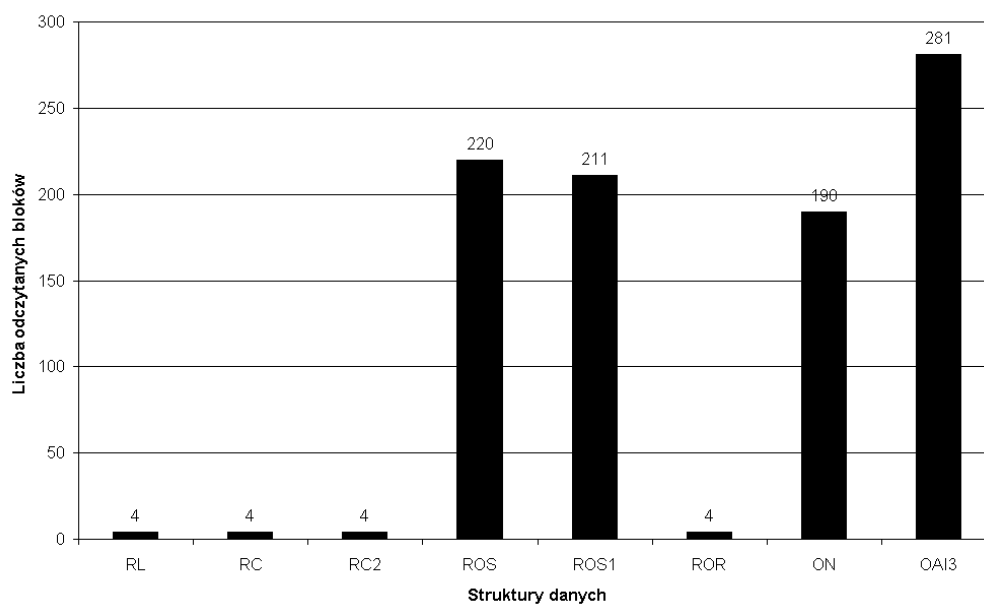
create or replace type body t_oceny_oai32 is
  member function getvalue(pi number) return number is
```

```

begin
  return(self.oc(pi));
end;
end;
/

```

Dla tej struktury (*OAI3*) już niestety nie da się zdefiniować oddzielnego segmentu dla kolekcji VARRAY.



Wykres 6. Liczby logicznych odczytów dla wszystkich rekordów tabeli *detail*.

Dla porównania wyniki uzupełniono o pomiar wydajności klastrów *RC* i *RC2*

Tym razem plan wykonania dla zagnieżdżonej tabeli uwzględnia założony indeks:

Plan wykonania

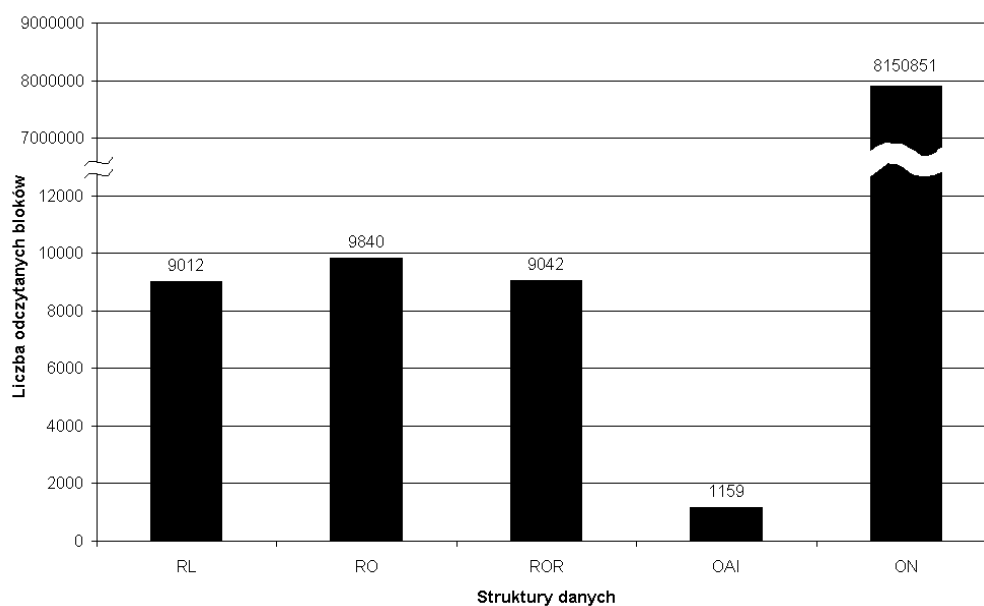
```

-----
0      SELECT STATEMENT
1  0    TABLE ACCESS (BY INDEX ROWID) OF 'OCENY_NTAB'
2  1      NESTED LOOPS
3  2          TABLE ACCESS (FULL) OF 'STUDENCI'
4  2          INDEX (RANGE SCAN) OF 'IDX2' (NON-UNIQUE)

```

#### Test 4. Wszystkie rekordy tabeli *detail* i wszystkie rekordy tabeli *master*.

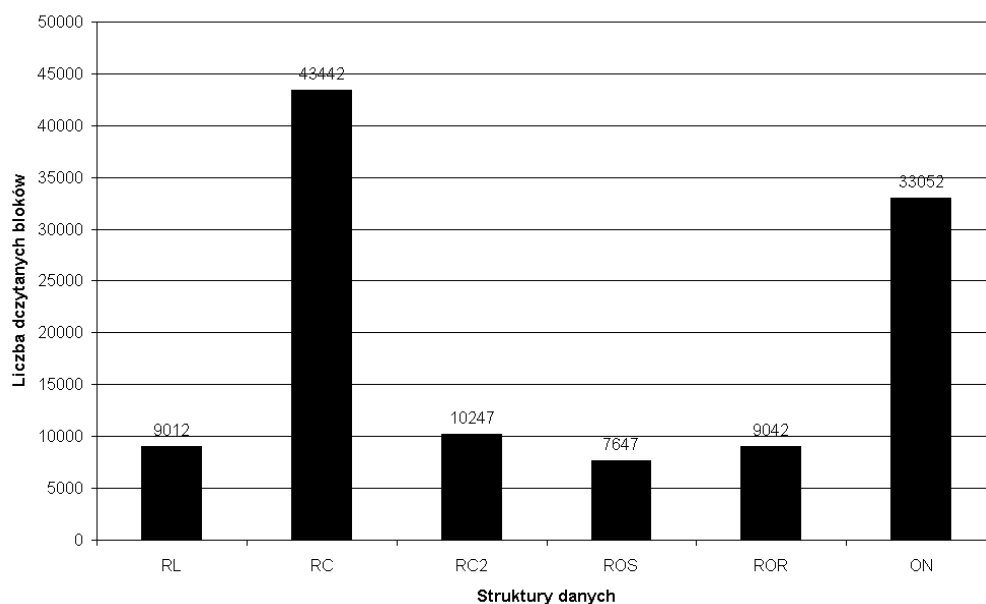
W tym doświadczeniu zostaną wyświetlone wszystkie dane z obu tabel. Wykres 7. przedstawia rezultaty dla struktur, w których nie ma indeksów.



Wykres 7. Liczby logicznych odczytów dla obu tabel

Wszystkie struktury poza *OAI* wymagają złączenia danych z dwóch tabel. Dla struktur *RL*, *RO* i *ROR* optymalizator wybrał algorytm *hash-join*. Struktura *OAI* z definicji posiada dane już złączone i stąd najlepszy wynik uzyskano właśnie dla niej. Wynik ten jest lepszy nawet dla sytuacji, w której założono indeksy (wykres 8.).

Wyniki dla *RL* i *ROR* są identyczne przed i po założeniu indeksów. Dzieje się tak dlatego, że metoda łączenia danych w obu przypadkach jest taka sama: *hash-join*.

Wykres 8. Liczby logicznych odczytów dla tabeli *master* i *detail*.

Dla porównania dodano też wyniki dla opisanych wcześniej struktur klastrowych *RC* i *RC2* oraz struktura *ROS* (czyli *RO* z indeksem).

## 4. Podsumowanie

Dla każdego testu wybrano struktury najlepszą i najgorszą. Wyniki zaprezentowane są w tabeli 5.

Tabela 5. Podsumowanie wyników

	Test 1	Test 2	Test 3	Test 4
<b>RL</b>	+	++	++	+
<b>RC</b>	--		++	-
<b>RC2</b>	-		++	--
<b>RO</b>	+	--	++	+
<b>ROS</b>		+	+	--
<b>ROS1</b>		+	+	
<b>ROR</b>	+	++	++	+
<b>OAI</b>	-	-	-	++
<b>OAI2</b>	+		-	
<b>OAI3</b>			-	
<b>ON</b>	+	+	+	--

Z zestawienia widać, że najbardziej optymalne są dwie struktury *RL* i *ROR*. Pierwsza z nich to tradycyjna struktura relacyjna. *ROR* została zadeklarowana jako:

```
create type t_student_ror is object(
  id          number(10),
  imie        varchar2(70),
  nazwisko    varchar2(70),
  grupa_stud varchar2(10),
  data_ur     date
);
/

create table studenci_ror(
  student t_student_ror
);

create table oceny_ror(
  fk_student number(10),
  kodprzed   number(2),
  ocena      number(2)
);
```

Mimo istnienia obiektów, tabela *STUDENCI\_ROR* jest przechowywana jako zwykła tabela relacyjna. Przemapowanie zachodzi podczas analizy składniowej zapytań. Nie należy zatem się dziwić, że wyniki wydajnościowe będą podobne.

Dobre wyniki otrzymano też dla struktur *RO* (lub, dla zapytań podobnych do tych, które użyto w teście drugim, *ROS*) i tabela zagnieżdżona *ON*.

O pozostałych strukturach (w tym również o relacyjnych klastrach) można powiedzieć, że są dobre tylko dla określonego rodzaju zapytań. Dla innych zapytań wyniki dla nich nie są najlepsze.

**Bibliografia**

- [CaSk92] Cattel R., Skeen J.: Object Operations Benchmark. ACM Transactions on Database Systems, Vol 17, No 1, 1992, str 1-31.
- [DuDa88] Duhi J., Damon C.: A Performance Comparison of Object and Relational Databases Using the Sun Benchmark. Materiały konferencyjne OOPSLA, 1988, str. 153-163
- [Harr01] Harrison G.: Oracle SQL High-Performance Tuning 2ed., Prentice Hall PTR, 2001, ISBN 0-13-012381-1
- [ORACLE] Oracle, Application Developer's Guide - Object-Relational Features, part no A96594-01
- [WoZa02] Wojciechowski M., Zakrzewicz M.: TPC Benchmarking, Materiały konferencyjne PLOUG2002, Zakopane, październik, 2002, ISSN 1641-2117