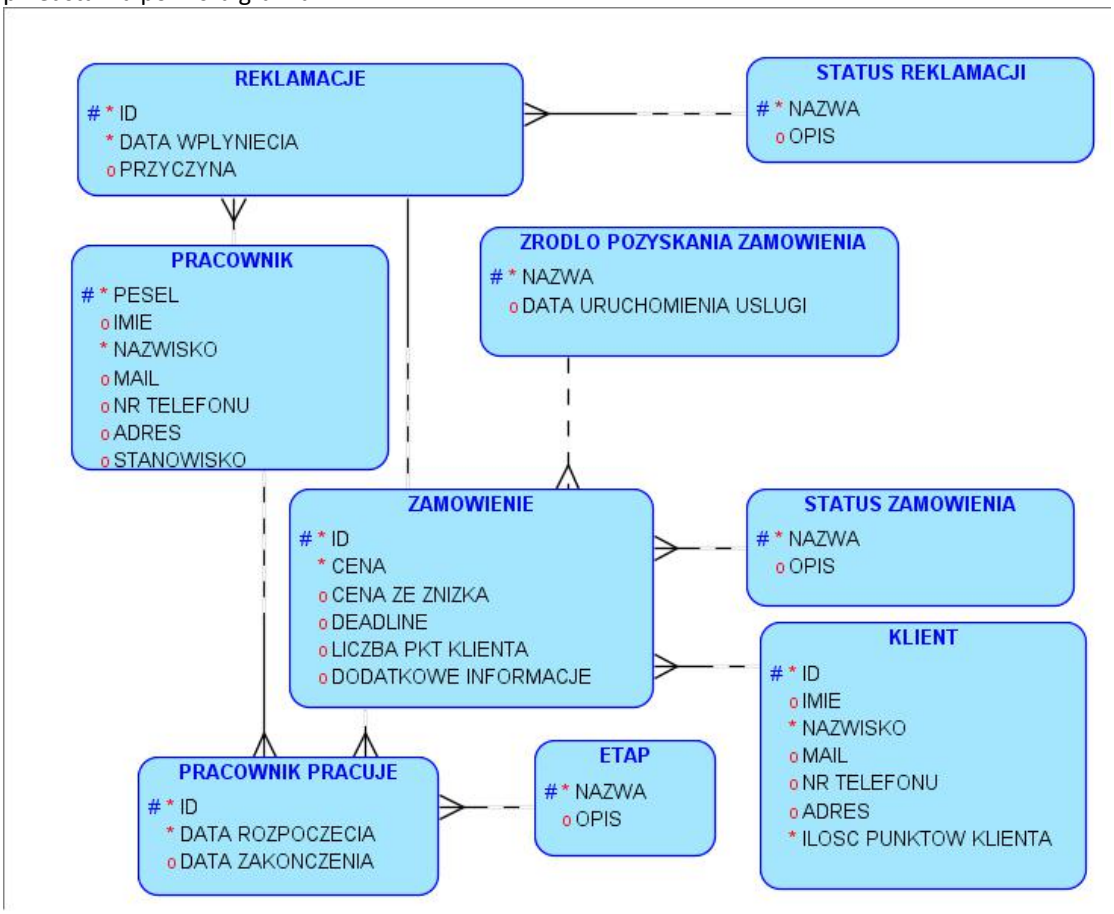
DOKUMENTACJA PROJEKTU
Z PRZEDMIOTU BAZY DANYCH

Autor: **Hubert Ziarek**
Uczelnia: **Wydział EiTI PW**
Temat: **Baza danych małej firmy świadczącej usługi w zakresie projektów graficznych**

Przygotowana przeze mnie baza jest symulacją bazy danych małej firmy przygotowującej projekty graficzne na zamówienie. Baza danych składa się z 9 tabel, których nazwy i relacje logiczne przedstawia poniższa grafika:



Baza dba przede wszystkim o poprawność danych wprowadzanych do tabeli PRACOWNIK PRACUJE, która zawiera informacje o tym który pracownik pracował nad danym zamówieniem na danym etapie. Ograniczenia egzekwowane przez bazę:

- Etapów związanych bezpośrednio z realizacją graficzną nie może realizować pracownik nie pracujący na stanowisku grafik,
- Data zakończenia prac nad etapem nie może być wcześniejsza niż data rozpoczęcia,
- Etapy „przyjecie” i „oddanie” dotyczące danego zamówienia mogą pojawić się w tabeli tylko raz,
- Nie ma możliwości wprowadzania prac nad kolejnymi etapami bez wprowadzenia etapu „przyjecie”,
- Nie ma możliwości wprowadzania prac nad etapami po wprowadzeniu etapu „oddanie”.

W trakcie wprowadzania danych dynamicznie uaktualniana jest suma punktów przypisanych klientowi oznaczających zniżkę oraz jest liczona nowa cena z uwzględnieniem zniżki posiadanej przez klienta, który składa zamówienie.

Dodatkowo wystąpienie etapu „oddanie” w tabeli PRACOWNIK PRACUJE zmienia status zamówienia (którego to wystąpienie dotyczy) na „zrealizowane”.

Warty wspomnienia jest również fakt, że reklamacja powiązana z danym zamówieniem może wystąpić tylko raz. Ewentualna realizacja reklamacji występuje jako nowe zamówienie, a zatem ewentualna kolejna reklamacja będzie już dotyczyła nowego zamówienia, co oznacza, że dwie reklamacje dotyczące jednego zamówienia są błędem.

DOKUMENTACJA INDEKSÓW

Zastosowanie indeksów testowałem na tablicy transakcyjnej ZAMOWIENIA o 2500 rekordach.

Najpierw indeks nałożony na pojedynczą kolumnę - klient:

create index zamowienie_tylko_klient on zamowienie (klient);

Zapytanie wykorzystane w testach:

```
explain plan for
select klient from zamowienie where klient < 267;
select *
from table (dbms_xplan.display);
```

Dla wartości zapytania mniejszej lub równej 267 uzyskiwałem następujący log:

```
-------------------------------------------------------------------------------------------
| Id  | Operation         | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                         |   832 |  3328 |     3   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN | ZAMOWIENIE_TYLKO_KLIENT |   832 |  3328 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Dla większej zaś:

```
| Id  | Operation         | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                         |   835 |  3340 |     3   (0)| 00:00:01 |
|*  1 |  INDEX FAST FULL SCAN | ZAMOWIENIE_TYLKO_KLIENT |   835 |  3340 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Widać stąd, że wartością graniczną jest 267, która jako argument zapytania pozwala wyświetlić około **32%** rekordów.

Warto odnotować też zysk wynikający z użycia indeksu. Log z przeszukiwanie tabeli bez indeksu

```
---------------------------------------------------------------------------
| Id  | Operation         | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |            |   936 |  3744 |     8   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| ZAMOWIENIE |   936 |  3744 |     8   (0)| 00:00:01 |
---------------------------------------------------------------------------
```

Wykorzystanie procesora spadło z 8% (bez indeksu) do 3% z indeksem.

Indeksy nałożone na kilka kolumn testowałem w oparciu o 2 następujące indeksy:

```
create index zamowienie_idx1 on zamowienie(klient, cena, id);
create index zamowienie_idx2 on zamowienie(cena, klient, id);
```

I zapytanie:

```
explain plan for
select id from zamowienie where klient < 250 and cena < 50 ;
select *
from table (dbms_xplan.display);
```

Wielokrotnie zmieniałem wartości ograniczeń zapytania. Stopień skomplikowania jaki wynika z nałożenia indeksu na 3 kolumny oraz wykorzystania 2 indeksów uniemożliwił mi znalezienie konkretnych wartości, ale oto kilka wyników różnych zapytań wraz z wnioskami:

CENA<40 AND KLIENT<600

```
-------------------------------------------------------------------------------
| Id  | Operation            | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                     |   145 |  1885 |     2   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN    | ZAMOWIENIE_IDX2     |   145 |  1885 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

KLIENT<250 AND CENA<450

```
-------------------------------------------------------------------------------
| Id  | Operation            | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                     |   701 |  9113 |     4   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN    | ZAMOWIENIE_IDX1     |   701 |  9113 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

KLIENT<600 AND CENA<480

```
-------------------------------------------------------------------------------
| Id  | Operation            | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                     |  1799 | 23387 |     4   (0)| 00:00:01 |
|*  1 |  INDEX FAST FULL SCAN| ZAMOWIENIE_IDX1     |  1799 | 23387 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Wnioski:

Jak widać Oracle wybiera indeks, który ma jako pierwszą kolumnę, na którą zostały nałożone bardziej zawężone wymagania poszukiwań. Co istotne, nie udało mi się uzyskać pełnego skanowania dla indeksu ZAMOWIENIE_IDX2, co może oznaczać, że w sytuacji w której Oracle nie ma istotnego zysku z użycia indeksu (nie może dokonać skanowania częściowego) wybiera pierwszy dostępny indeks porządkujący kolumny, w ramach których ma dokonać poszukiwań.

HINTY

Hinty testowałem w oparciu o algorytmy wykorzystywane przez Oracle przy wykonywaniu zapytania JOIN, przy wyłączonych indeksach. Oto wyniki:

Zapytanie

```
explain plan for

select klient.nazwisko, zamowienie.id
from
klient join zamowienie on klient.id = zamowienie.klient;
select *
from table (dbms_xplan.display);
```

pozwoliło uzyskać wykorzystanie algorytmu HASH_JOIN.

Zapytanie

```
explain plan for

select klient.nazwisko, zamowienie.id
from
klient join zamowienie on klient.id = zamowienie.klient
where klient.id = 100;
select *
from table (dbms_xplan.display);
```

pozwoliło uzyskać wykorzystanie algorytmu NESTED LOOP.

Niestety uzyskanie algorytmu MERGE JOIN okazało się bardzo trudne; żaden ze znalezionych sposobów nie pozwolił go uzyskać, z wyjątkiem niezbyt przydatnego warunku złączenia „>=”:

```
explain plan for

select klient.nazwisko, zamowienie.id
from
klient join zamowienie on klient.id >= zamowienie.klient;
select *
from table (dbms_xplan.display);
```

WYMUSZANIE ALGORYTMÓW PRZY POMOCY HINTÓW

Zapytanie wymuszające MERGE JOIN w sytuacji, gdzie Oracle domyślnie korzystał z HASH JOIN:

```
explain plan for
select /*+ USE_MERGE(zamowienie klient) */ klient.nazwisko, zamowienie.id
from
klient join zamowienie on klient.id = zamowienie.klient;
select *
from table (dbms_xplan.display);
```

Zapytanie wymuszające NESTED LOOP w sytuacji, gdzie Oracle domyślnie korzystał z HASH JOIN:

```
explain plan for
select /*+ USE_NL(zamowienie klient) */ klient.nazwisko, zamowienie.id
from
klient join zamowienie on klient.id = zamowienie.klient;
select *
from table (dbms_xplan.display);
```

Na koniec warto dodać, że udało mi się przekonać o tym, że hinty są jedynie wskazówką i nie wszystko pozwalają wymusić. Widać to na przykładzie zapytania

```
explain plan for

select klient.nazwisko, zamowienie.id
from
klient join zamowienie on klient.id = zamowienie.klient
where klient.id = 100;
select *
from table (dbms_xplan.display);
```

które wykorzystuje algorytmu NESTED LOOP. Żadna z prób wymuszenia innego algorytmu nie zakończyła się sukcesem, co jest logiczne wziąwszy pod uwagę fakt, że wybieramy tu jeden konkretny rekord i z próba haszowania czy sortowania nie ma większego sensu.