Ćwiczenie 3.
Statystyki, wskazówki

W niniejszym ćwiczeniu zapoznamy się z problematyką zarządzania statystykami oraz stosowaniem wskazówek dla optymalizatora.

Podstawowe statystyki

W zadaniach z poprzednich warsztatów pojawiało się polecenie zbierania statystyk. W tej części warsztatu przyjrzymy się bliżej problematyce zarządzania statystykami.

Uwaga! Przed rozpoczęciem ćwiczenia zapoznaj się z materiałami dotyczącymi statystyk dla optymalizatora (prezentacja pt. „Optymalizacja poleceń SQL. Statystyki").

1. Jeśli masz w narzędziu włączoną dyrektywę `AUTOTRACE`, wyłącz ją. Na początek usuniemy statystyki, które były wykorzystywane przy realizacji zapytań do relacji `OPT_PRACOWNICY` w poprzednich warsztatach. Do tego celu użyjemy funkcji `DELETE_TABLE_STATS` z pakietu `DBMS_STATS` (jako wartość parametru `OWNNAME` podaj nazwę swojego schematu bazodanowego).

```
BEGIN
  DBMS_STATS.DELETE_TABLE_STATS(ownname => '<nazwa_schematu>',
      tabname => 'OPT_PRACOWNICY');
END;
```

2. Sprawdź informacje o statystykach dla relacji `OPT_PRACOWNICY`. Znajdź informacje o znaczeniu poszczególnych kolumn.

```
SELECT num_rows, last_analyzed, avg_row_len, blocks
FROM user_tables
WHERE table_name = 'OPT_PRACOWNICY';

SELECT column_name, num_distinct, low_value, high_value
FROM user_tab_col_statistics
WHERE table_name = 'OPT_PRACOWNICY';
```

   Zapytanie 1. (statystyki dla całej relacji `OPT_PRACOWNICY`) odczytuje rekord z pustymi wartościami. Zapytanie 2. (statystyki dla kolumn relacji `OPT_PRACOWNICY`) zwraca pusty zbiór wynikowy. Oznacza to, że rzeczywiście nasza relacja `OPT_PRACOWNICY` nie posiada statystyk.

   Teraz sprawdź statystyki dla indeksów, jakie utworzyliśmy dla relacji `OPT_PRACOWNICY`.

```
SELECT index_name, num_rows, leaf_blocks, last_analyzed
FROM user_indexes
WHERE table_name = 'OPT_PRACOWNICY';
```

   Tu też otrzymujemy zbiór rekordów z pustymi wartościami kolumn dla poszczególnych indeksów. Usuwając statystyki dla relacji w p. 1., automatycznie usunęliśmy również statystyki dla indeksów relacji `OPT_PRACOWNICY`.

3. Spróbujemy teraz zgromadzić statystyki dla relacji OPT_PRACOWNICY. Do tego celu użyjemy funkcji GATHER_TABLE_STATS z pakietu DBMS_STATS (jako wartość parametru OWNNAME podaj nazwę swojego schematu bazodanowego).

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS(
     ownname=>'<nazwa_schematu>', tabname => 'OPT_PRACOWNICY');
END;
```

   Następnie ponownie wyświetl informacje o statystykach relacji OPT_PRACOWNICY, jej kolumn oraz indeksów.

4. Polecenie, jakie wykonaliśmy w poprzednim punkcie, zgromadziło statystyki przeglądając wszystkie dane relacji OPT_PRACOWNICY. Jeśli relacja, dla której chcemy zgromadzić statystyki jest duża, proces pełnego jej przeglądu w celu zebrania statystyk może być długi. W takiej sytuacji można zgromadzić statystyki dla relacji, przeglądając jedynie próbkę jej danych. Poniższe polecenie zgromadzi statystyki dla relacji OPT_PRACOWNICY na podstawie próbki o rozmiarze 40% liczby rekordów relacji.

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS(
     ownname=>'<nazwa_schematu>', tabname => 'OPT_PRACOWNICY',
     estimate_percent => 40);
END;
```

   Statystyki, zebrane na podstawie próbki, mogą nie być dokładne. Sprawdź poniższym poleceniem aktualne statystki relacji OPT_PRACOWNICY (kolumna SAMPLE_SIZE pokazuje rozmiar próbki w rekordach).

```
SELECT num_rows, last_analyzed, avg_row_len, blocks, sample_size
FROM user_tables
WHERE table_name = 'OPT_PRACOWNICY';
```

   i zanalizuj wynik:

```
  NUM_ROWS LAST_ANALYZED  AVG_ROW_LEN     BLOCKS       SAMPLE_SIZE
---------- ------------- ------------ ---------- -----------------
     10133 16/07/29                45         65              4053
```

   Jak widzimy, uzyskane statystyki nie są bardzo dokładne (np. liczba rekordów relacji), jednak oddają charakterystykę danych w relacji. Uwaga! Czasem system ignoruje wskazany w poleceniu rozmiar próbki i dobiera własny. Można również powierzyć systemowi dobranie rozmiaru próbki podając jako wartość parametru ESTIMATE_PERCENT stałą DBMS_STATS.AUTO_SAMPLE_SIZE.

5. Jak już wiemy, statystyki obiektu można usunąć. Przykładowa sytuacja: statystyki relacji nie opisują jej aktualnego stanu (np. relacja zwiększyła swój rozmiar), a nie możemy uaktualnić statystyk (np. nie możemy zdegradować wydajności działania systemu na czas zbierania statystyk). W takiej sytuacji lepiej usunąć statystyki i bazować na mechanizmie tzw. dynamicznego próbkowania (przyjrzymy się mu w kolejnym punkcie). Poniższy kod usunie wszystkie statystyki dla relacji `OPT_PRACOWNICY`.

```
BEGIN
  DBMS_STATS.DELETE_TABLE_STATS(
     ownname=>'<nazwa_schematu>', tabname => 'OPT_PRACOWNICY');
END;
```

   Uwaga! Powyższe polecenie, oprócz statystyk dla relacji, usunie również statystyki dla jej kolumn oraz indeksów. Sterują tym parametry `CASCADE_COLUMNS` i `CASCADE_INDEXES` z wartościami domyślnymi równymi `TRUE`. Ustawienie wartości tych parametrów na `FALSE` pozostawi statystyki dla kolumn i indeksów relacji. W pakiecie `DBMS_STATS` istnieją osobne procedury do usuwania statystyk dla indeksów i kolumn relacji, odpowiednio, `DELETE_INDEX_STATS` i `DELETE_COLUMN_STATS`.

   Sprawdź w słowniku bazy danych, czy polecenie rzeczywiście usunęło statystyki relacji, jej kolumn i indeksów.

6. Spróbuj teraz wykonać zapytanie, znajdujące dane pracowników o nazwiskach zgodnych ze wzorcem „Prac155%”. Wyświetl plan wykonania tego polecenia.

```
SELECT * FROM opt_pracownicy WHERE nazwisko LIKE 'Prac155%';
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name            | Rows | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                 |   11 |  1012 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID|OPT_PRACOWNICY   |   11 |  1012 |     3   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           |OP_NAZW_PLACA_IDX|   11 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("NAZWISKO" LIKE 'Prac155%')
       filter("NAZWISKO" LIKE 'Prac155%')

Note
-----
   - dynamic sampling used for this statement (level=2)
```

   Otrzymaliśmy wynik mimo tego, że usunęliśmy statystyki relacji `OPT_PRACOWNICY`. W jaki sposób system mógł dokonać optymalizacji polecenia bez statystyk?

   Zwróć uwagę na sekcję *Note* w powyższym planie. Pojawił się tam komunikat „*dynamic sampling used for this statement (level=2)*”. Oznacza on, że potrzebne do optymalizacji zapytania statystyki zostały zebrane w procesie tzw. dynamicznego próbkowania. Proces ten, bezpośrednio przed rozpoczęciem optymalizacji polecenia, dokonuje analizy części danych obiektu bazodanowego („próbkuje dane”) i na tej podstawie generuje statystyki opisujące obiekt. Stopień zaawansowania procesu został ustawiony na poziom 2 (poziom domyślny; im poziom wyższy, tym więcej czasu przed wykonaniem polecenia system przeznacza na dynamiczne próbkowanie i tym dokładniejsze statystyki, zakres od 0 do 10). Statystyki generowane w trakcie dynamicznego

próbkowania nie są składowane w słowniku bazy danych i muszą być generowane ponownie przed każdym wykonaniem zapytania. Dynamiczne próbkowanie jest włączane automatycznie w sytuacji, gdy optymalizator nie ma statystyk dotyczących obiektu, który jest adresowany w poleceniu SQL.

7. Na poziom z jakim realizowane jest dynamiczne próbkowanie w sytuacji braku statystyk, można wpłynąć, dodając do polecenia tzw. wskazówkę o nazwie `DYNAMIC_SAMPLING`. Jej parametrem jest poziom, na jakim statystyki będą zbierane. Poziom 0 to wyłączenie dynamicznego próbkowania, poziom 10 jest poziomem najwyższym – dynamiczne próbkowanie jest wtedy realizowane w najszerszym zakresie. Spróbuj wykonać zapytanie z poprzedniego punktu, całkowicie wyłączając dynamiczne próbkowanie.

```
SELECT /*+ DYNAMIC_SAMPLING(0) */ *
FROM opt_pracownicy WHERE nazwisko LIKE 'Prac155%';
```

Przypomnijmy, że system nie ma statystyk dla relacji `OPT_PRACOWNICY`, zakazaliśmy również wykonania dynamicznego próbkowania. Mimo to zapytanie zostało wykonane.

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Rows | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |  266 | 24472 |    10   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| OPT_PRACOWNICY  |  266 | 24472 |    10   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           |OP_NAZW_PLACA_IDX|   48 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("NAZWISKO" LIKE 'Prac155%')
       filter("NAZWISKO" LIKE 'Prac155%')
```

Zwróćmy uwagę na szacunki optymalizatora (kolumna `Rows` w planie). Są one znacząco inne niż te z planu z dynamicznym próbkowaniem. Otóż system w tej sytuacji wziął domyślne wartości dla poszczególnych statystyk. Oczywiście nie opisują one w żaden sposób profilu danych, do których kierowane jest zapytanie i plan, wygenerowany na ich podstawie, najczęściej nie będzie planem optymalnym.

8. Zbierz ponownie pełne statystyki dla relacji `OPT_PRACOWNICY` i jej kolumn. Od razu zgromadź również statystyki dla indeksów tej relacji: można to zrobić za pomocą osobnej procedury `GATHER_INDEX_STATS` lub dodając do procedury `GATHER_TABLE_STATS` parametr `CASCADE` o wartości `TRUE`. Uwaga: pominięcie parametru `CASCADE` oznacza, że zostanie użyta stała `DBMS_STATS.AUTO_CASCADE` wskazująca, czy SZBD ma zebrać statystyki dla indeksów.

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS(
     ownname=>'<nazwa_schematu>', tabname => 'OPT_PRACOWNICY',
     cascade => TRUE);
END;
```

Sprawdź, czy statystyki dla relacji `OPT_PRACOWNICY`, jej kolumn i indeksów zostały zebrane.

Histogramy

Szczególnym rodzajem statystyki dla kolumny jest histogram. Przeprowadzimy teraz eksperyment, pokazujący znaczenie tej statystyki w procesie optymalizacji zapytań do danych o rozkładzie nierównomiernym

1. Wykonaj zapytanie do słownika bazy danych, które odczyta informacje o statystykach kolumn relacji `OPT_PRACOWNICY`.

   ```
   SELECT column_name, num_distinct, low_value, high_value,
          num_buckets, histogram
   FROM user_tab_col_statistics
   WHERE table_name = 'OPT_PRACOWNICY';
   ```

   Dla niektórych kolumn relacji `OPT_PRACOWNICY`, w kolumnie `HISTOGRAM` wyniku powyższego zapytania, pojawiają się wartości *„FREQUENCY”* albo *„HEIGHT BALANCED”*. Oznacza to, że system zbudował dla tych kolumn histogramy, odpowiednio, częstotliwości lub o zrównoważonej wysokości. Decyzję, dla których kolumn zebrać te statystyki, system podejmuje na podstawie obserwacji rozkładu wartości w kolumnie (histogram jest użyteczny jeśli rozkład jest nierównomierny) oraz stwierdzenia, czy dana kolumna jest używana w warunkach filtrujących zapytań.

2. Przyjrzyjmy się kolumnie `PLACA_DOD` relacji `OPT_PRACOWNICY`. Wartość `NUM_DISTINCT` dla tej kolumny to 2, czyli dziedzina wartości tej kolumny składa się z dwóch elementów. System zdecydował o zbudowaniu histogramu dla tej kolumny, to wskazuje, że rozkład jej wartości jest nierównomierny. Wykonaj zapytanie, które pokaże rozkład wartości kolumny `PLACA_DOD`.

   ```
   SELECT placa_dod, COUNT(*)
   FROM opt_pracownicy
   GROUP BY placa_dod;
   ```

   Widzimy, że w 99,99% rekordów relacji kolumna ta ma wartość 100 a w pozostałym 0,01% wartość 999.

3. Usuń histogram dla kolumny `PLACA_DOD` poniższym poleceniem.

   ```
   BEGIN
     DBMS_STATS.DELETE_COLUMN_STATS(
       ownname => '<nazwa_schematu>', tabname => 'OPT_PRACOWNICY',
       colname => 'PLACA_DOD', col_stat_type => 'HISTOGRAM');
   END;
   ```

   Sprawdź w słowniku bazy danych, czy rzeczywiście histogram został usunięty (wartość *„NONE”* w kolumnie `HISTOGRAM`).

   ```
   SELECT num_distinct, low_value,
          high_value, num_buckets, histogram
   FROM user_tab_col_statistics
   WHERE table_name = 'OPT_PRACOWNICY'
   AND column_name = 'PLACA_DOD';
   ```

   Teraz jedyną statystyką dla kolumny `PLACA_DOD`, na podstawie której system może wnioskować o profilu danych w tej kolumnie, jest liczba różnych wartość (`NUM_DISTINCT`) równa 2.

4. Zbuduj indeks bitmapowy o nazwie `OP_BMP_PLACA_DOD_IDX`, w kluczu którego będą wartości kolumny `PLACA_DOD` relacji `OPT_PRACOWNICY`.

```
CREATE BITMAP INDEX op_bmp_placa_dod_idx ON opt_pracownicy(placa_dod);
```

Sprawdź, czy dla utworzonego indeksu są statystyki (wykonaj odpowiednie zapytanie do perspektywy systemowej `USER_INDEXES`).

5. Wyświetl plany wykonania dla dwóch poniższych zapytań.

```
SELECT * FROM opt_pracownicy WHERE placa_dod = 100;

-------------------------------------------------------------------------------------
| Id  | Operation         | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                |  5000 |   180K|    19   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| OPT_PRACOWNICY |  5000 |   180K|    19   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PLACA_DOD"=100)


SELECT * FROM opt_pracownicy WHERE placa_dod = 999;

-------------------------------------------------------------------------------------
| Id  | Operation         | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                |  5000 |   180K|    19   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| OPT_PRACOWNICY |  5000 |   180K|    19   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PLACA_DOD"=999)
```

Jak widzimy, dla obu zapytań optymalizator zaproponował identyczne plany wykonania mimo tego, że zwracane przez zapytania zbiory rekordów są krańcowo odmienne: pierwsze zapytanie (z predykatem `PLACA_DOD = 100`) odczytuje 9 999 rekordów, zapytanie drugie (z predykatem `PLACA_DOD = 999`) tylko 1 rekord. Dlaczego tak się dzieje? Spójrzmy na wartość w kolumnie `ROWS` operacji pełnego przeglądnięcia tabeli, w ramach której zostaje zrealizowany predykat filtrujący zapytania. W obu przypadkach system ocenił, że predykat odfiltruje połowę rekordów relacji (5 000 rekordów). Takie oszacowanie zostało podjęte na podstawie statystyki `NUM_DISTINCT` kolumny `PLACA_DOD`: system „wie”, że dziedzina tej kolumny składa się jedynie z dwóch wartości, zakłada rozkład równomierny wartości w tej kolumnie (usunęliśmy histogram!), czyli ocenia, że predykat odfiltruje około 50% rekordów relacji, czyli 5 000. Dla takiego rozmiaru danych odpowiednią operacją jest pełne przeglądnięcie tabeli – i tę operację optymalizator użył w obu przypadkach. O ile w przypadku zapytania z predykatem `PLACA_DOD = 100` nie jest to plan zły, o tyle dla zapytania z predykatem `PLACA_DOD = 999` jest to plan fatalny, szczególnie w sytuacji obecności indeksu dla kolumny `PLACA_DOD`.

6. Naprawmy sytuację i dodajmy do zbioru statystyk kolumny `PLACA_DOD` histogram. Możemy to wykonać poniższym poleceniem (parametr `METHOD_OPT` pozwala wskazać kolumnę, dla której ma zostać zgromadzony histogram).

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS(
   ownname => '<nazwa_schematu>', tabname => 'OPT_PRACOWNICY',
   method_opt => 'FOR COLUMNS placa_dod SIZE AUTO');
END;
```

Następnie sprawdź w słowniku bazy danych, czy w statystykach kolumny `PLACA_DOD` relacji `OPT_PRACOWNICY` pojawił się histogram.

7. Wykonajmy ponownie oba zapytania i zanalizujmy ich plany (w drugim planie dla poprawienia czytelności pominięto kolumnę *Time*).

```
SELECT * FROM opt_pracownicy WHERE placa_dod = 100;

-------------------------------------------------------------------------------------
| Id  | Operation         | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                |  9999 |   361K|    19   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| OPT_PRACOWNICY |  9999 |   361K|    19   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PLACA_DOD"=100)


SELECT * FROM opt_pracownicy WHERE placa_dod = 999;

-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name               | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                    |     1 |    37 |     1   (0)|
|   1 |  TABLE ACCESS BY INDEX ROWID |OPT_PRACOWNICY      |     1 |    37 |     1   (0)|
|   2 |   BITMAP CONVERSION TO ROWIDS|                    |       |       |            |
|*  3 |    BITMAP INDEX SINGLE VALUE |OP_BMP_PLACA_DOD_IDX|       |       |            |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("PLACA_DOD"=999)
```

Jak widzimy, plan dla pierwszego zapytania się nie zmienił – jest to nadal plan z operacją pełnego przeglądnięcia tabeli. Zwróćmy jednak uwagę na oszacowanie liczby rekordów: jest ono teraz prawidłowe (9 999 rekordów) – tu widzimy wpływ obecności histogramu.

Plan drugi zmienił się diametralnie. System na podstawie histogramu teraz „wie”, że wartość 999 występuje tylko w jednym rekordzie relacji, a to jest idealna sytuacja do zastosowania w planie metody dostępu, wykorzystującej indeks. I taki plan z indeksem bitmapowym na kolumnie `PLACA_DOD` zostaje wybrany do realizacji polecenia.

Wniosek: histogram jest bardzo użyteczną statystyką dla kolumn, których rozkład nie jest równomierny. Histogram dla takich kolumn pozwala optymalizatorowi w poprawny sposób oszacować selektywność predykatu odwołującego się do danej kolumny. Dla kolumn o równomiernym rozkładzie wartości histogram nie jest potrzebny – tu w zupełności wystarcza statystyka pokazująca rozmiar dziedziny wartości kolumny (`NUM_DISTINCT` w `USER_TAB_COLUMNS`).

Wskazówki

Uwaga! Przed rozpoczęciem ćwiczenia zapoznaj się z materiałami dotyczącymi wskazówek dla optymalizatora (prezentacja pt. „Optymalizacja poleceń SQL. Wskazówki").

Użytkownik SZBD ma możliwość „wspomóc" optymalizator w procesie budowy planu wykonania polecenia SQL jeśli widzi, że działania optymalizatora nie skutkują uzyskaniem dobrego (optymalnego) planu wykonania mimo poprawności polecenia SQL, obecności aktualnych statystyk i dobrze zaprojektowanych indeksów. Mechanizmem umożliwiającym wpływ użytkownika na proces optymalizacji polecenia SQL są wskazówki (ang. *hints*). Wskazówkę umieszczamy za pierwszą klauzulą polecenia (np. zaraz za `SELECT` w przypadku zapytania), oznaczając ją jako komentarz. Nazwę wskazówki poprzedzamy znakiem „+".

Uwaga! Jeśli popełnimy błąd przy definicji wskazówki, zostanie ona zignorowana (żaden komunikat o błędnej wskazówce nie zostanie wyświetlony). System również zignoruje wskazówkę, której nie będzie mógł zastosować (np. użycie indeksu z innej relacji niż ta, która jest adresowana w poleceniu).

Pamiętaj! Plan, wygenerowany na podstawie wskazówek, może oczywiście nie być planem optymalnym. Dając optymalizatorowi wskazówkę niejako bierzesz na siebie odpowiedzialność za jakość planu, a tym samym za efektywność procesu wykonania polecenia SQL

1. Wyświetl plan zapytania wyszukującego dane pracownika o identyfikatorze 900.

```
SELECT * FROM opt_pracownicy WHERE id_prac = 900;
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |     1 |    37 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| OPT_PRACOWNICY |     1 |    37 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | OP_PK          |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

   Jak się zapewne spodziewałaś/eś, optymalizator zaproponował użycie indeksu na kolumnie `ID_PRAC`.

2. Dodamy teraz do zapytania wskazówkę `FULL`, „zmuszającą" optymalizator do użycia pełnego przeglądnięcia tabeli. Możesz to zrobić tak:

```
SELECT /*+ FULL(opt_pracownicy) */ *
FROM opt_pracownicy WHERE id_prac = 900;
```

   albo tak:

```
SELECT --+ FULL(opt_pracownicy)
* FROM opt_pracownicy WHERE id_prac = 900;
```

   We obu przykładach powinniśmy otrzymać plan w poniżej przedstawionej postaci.

```
-------------------------------------------------------------------------------------------
| Id  | Operation         | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                |     1 |    37 |    19   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| OPT_PRACOWNICY |     1 |    37 |    19   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

3. Jeśli w zapytaniu zdefiniowano alias dla relacji, zamiast nazwy relacji we wskazówce musi zostać użyty alias, inaczej wskazówka zostanie zignorowana. Pokazuje to poniższe doświadczenie:

```
SELECT /*+ FULL(pracownicy) */ *
FROM opt_pracownicy p
WHERE id_prac = 900;

SELECT /*+ FULL(p) */ *
FROM opt_pracownicy p
WHERE id_prac = 900;
```

W pierwszym zapytaniu wskazówka została zignorowana – w klauzuli `FROM` zdefiniowano alias `p` dla relacji `OPT_PRACOWNICY`, jednak we wskazówce nadal użyto nazwy relacji. Wskazówka zostanie zastosowana, jeśli w jej definicji zmienimy nazwę relacji na jej alias (drugie zapytanie).

4. Taki sam efekt jak przy zastosowaniu wskazówki `FULL` możemy uzyskać przy pomocy wskazówki `NO_INDEX`, która: (1) zakazuje użycia wszystkich indeksów przy dostępie do danych relacji jeśli parametrem wskazówki jest tylko nazwa (alias) relacji albo (2) zakazuje użycia konkretnego indeksu jeśli parametrami wskazówki są: nazwa (alias) relacji i nazwa indeksu.

W tym zapytaniu żaden indeks, jaki został utworzony dla relacji `OPT_PRACOWNICY`, nie zostanie użyty.

```
SELECT /*+ NO_INDEX(opt_pracownicy) */ *
FROM opt_pracownicy
WHERE id_prac = 900;
```

Tu z kolei indeks o nazwie `OP_PK` nie będzie mógł zostać użyty przy wykonaniu zapytania.

```
SELECT /*+ NO_INDEX(opt_pracownicy op_pk) */ *
FROM opt_pracownicy
WHERE id_prac = 900;
```

5. Kolejna wskazówka o nazwie `INDEX` „każe” optymalizatorowi użyć indeksu przy realizacji polecenia SQL. Wyświetlmy plan wykonania poniższego zapytania.

```
SELECT nazwisko, etat FROM opt_pracownicy;
```

Oczywiście w tym przypadku odpowiednią metodą dostępu będzie pełne przeglądnięcie tabeli.

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                | 10000 |   175K|    19   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| OPT_PRACOWNICY | 10000 |   175K|    19   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Zmuśmy optymalizator, aby przy realizacji tego zapytania użył jakiegoś indeksu (system powinien wybrać indeks, dla którego koszt planu z indeksem będzie najmniejszy).

```
SELECT /*+ INDEX(opt_pracownicy) */ nazwisko, etat
FROM opt_pracownicy;
```

System nas posłuchał – użył w planie indeksu `OP_PK`, wykonując do niego dostęp za pomocą operacji pełnego przeglądnięcia indeksu (metoda podobna do pełnego szybkiego przeglądnięcia indeksu z tą różnicą, że wykonuje odczyt liści indeksu zachowując ich kolejność, blok po bloku).

Jednak plan jest w tym przypadku absurdalny, co zresztą widać po jego wysokim koszcie (9622 jednostek).

```
-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                | 10000 |   175K|  9622   (1)| 00:01:56 |
|   1 |  TABLE ACCESS BY INDEX ROWID| OPT_PRACOWNICY | 10000 |   175K|  9622   (1)| 00:01:56 |
|   2 |   INDEX FULL SCAN           | OP_PK          | 10000 |       |    21   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Wykonaj samodzielnie kilka eksperymentów ze wskazówką `INDEX`, nakłaniając optymalizator do użycia wskazanych indeksów w planie wykonania zapytania do relacji `OPT_PRACOWNICY`.

6. Puśćmy wodze fantazji i wskażmy optymalizatorowi indeks bitmapowy do realizacji tego polecenia. Służy do tego wskazówka `INDEX_COMBINE`, jej zasady stosowania są takie same jak przy wskazówce `INDEX`.

```
SELECT /*+ INDEX_COMBINE(opt_pracownicy) */ nazwisko, etat
FROM opt_pracownicy;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                | Rows  | Bytes|Cost(%CPU)|Time    |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                     | 10000 |  175K|   106 (0)|00:00:02|
|   1 |  TABLE ACCESS BY INDEX ROWID |OPT_PRACOWNICY       | 10000 |  175K|   106 (0)|00:00:02|
|   2 |   BITMAP CONVERSION TO ROWIDS|                     |       |      |          |        |
|   3 |    BITMAP INDEX FULL SCAN    |OP_BMP_PLACA_DOD_IDX |       |      |          |        |
-----------------------------------------------------------------------------------------------
```

Tym razem optymalizator dokonał pełnego przeglądnięcia indeksu bitmapowego, odczytując bitmapy ze wszystkich jego liści. Następnie wykonał operację konwersji bitmap do adresów rekordów, wreszcie na podstawie adresów odczytał dane z rekordów relacji. Widzimy, że ten plan również nie jest dobry: jego koszt wielokrotnie przewyższa koszt planu z pełnym przeglądnięciem relacji.

7. Kolejna wskazówka o nazwie `INDEX_FFS` powoduje, że optymalizator w planie używa metody pełnego szybkiego przeglądnięcia indeksu. Wyświetlmy na początek plan poniższego zapytania:

```
SELECT placa
FROM opt_pracownicy
WHERE plec = 'K';
```

```
-------------------------------------------------------------------------------------
| Id  | Operation        | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |                   |  2000 | 12000 |     6   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN| OP_PLEC_PLACA_IDX |  2000 | 12000 |     6   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("PLEC"='K')
```

Całe zapytanie można obsłużyć zakresowym przeglądnięciem indeksu `OP_PLEC_PRACA_IDX`: warunek filtrujący odwołuje się do kolumny z części wiodącej klucza indeksu (kolumna `PLEC`), natomiast w klauzuli `SELECT` odwołujemy się do kolumny z części niewiodącej klucza (kolumna `PLEC`).

Możemy sobie zażyczyć, aby to polecenie było wykonane za pomocą szybkiego pełnego przeglądnięcia indeksu:

```
SELECT /*+ INDEX_FFS(opt_pracownicy) */ placa
FROM opt_pracownicy
WHERE plec = 'K';
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation            | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                   |  2000 | 12000 |     8   (0)| 00:00:01 |
|*  1 |  INDEX FAST FULL SCAN| OP_PLEC_PLACA_IDX |  2000 | 12000 |     8   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PLEC"='K')
```

System zastosował wskazaną metodę, jednak widzimy, że podpowiedź pogorszyła plan: jego koszt się zwiększył.

8. Kolejna podpowiedź, `INDEX_SS`, wskazuje, aby optymalizator w planie użył metody przeglądnięcia indeksu z pominięciem kolumn. Zastosujmy ją do poprzedniego zapytania:

```
SELECT /*+ INDEX_SS(opt_pracownicy) */ placa
FROM opt_pracownicy
WHERE plec = 'K';
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation        | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |                   |  2000 | 12000 |    25   (0)| 00:00:01 |
|*  1 |  INDEX SKIP SCAN | OP_PLEC_PLACA_IDX |  2000 | 12000 |    25   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("PLEC"='K')
```

Optymalizator uwzględnił naszą podpowiedź, jednak nie poprawiła ona kosztu planu a znacznie go pogorszyła.

9. Wskazówka `INDEX_JOIN` pozwala na użycie w planie operacji połączenia indeksów. Wykonaj poniższe zapytanie:

```
SELECT /*+ INDEX_JOIN(opt_pracownicy op_pk op_nazw_placa_idx) */
       nazwisko
FROM opt_pracownicy
WHERE id_prac < 1000 AND placa = 1500;
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation              | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                   |     1 |    17 |    38   (0)| 00:00:01 |
|*  1 |  VIEW                  | index$_join$_001  |     1 |    17 |    38   (0)| 00:00:01 |
|*  2 |   HASH JOIN            |                   |       |       |            |          |
|*  3 |    INDEX RANGE SCAN    | OP_PK             |     1 |    17 |     3   (0)| 00:00:01 |
|*  4 |    INDEX FAST FULL SCAN| OP_NAZW_PLACA_IDX |     1 |    17 |    44   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("ID_PRAC"<1000)
   2 - access(ROWID=ROWID)
   3 - access("ID_PRAC"<1000)
   4 - filter("PLACA"=1500)
```

Widzimy, że oba indeksy zostały użyte: `OP_PK` metodą zakresowego przeglądnięcia, `OP_NAZW_PLACA_IDX` metodą pełnego szybkiego przeglądnięcia indeksu. Wyniku obu przeglądów indeksów zostały poddane operacji połączenia, która wyprodukowała zbiór wynikowy polecenia bez konieczności realizacji dostępu do relacji adresowanej w zapytaniu.