Kosztowy optymalizator zapytań

Marek Wojciechowski, Maciej Zakrzewicz

marek.wojciechowski@cs.put.poznan.pl
maciej.zakrzewicz@cs.put.poznan.pl

Politechnika Poznańska, Instytut Informatyki

Streszczenie

Praktycznie każde zapytanie SQL skierowane do systemu zarządzania bazą danych może być zrealizowane na wiele różnych sposobów. Automatycznym wyborem najbardziej efektywnej metody wykonania zapytania (planu wykonania zapytania) zajmuje się moduł optymalizatora kosztowego. Znajomość zasad funkcjonowania optymalizatora pozwala programistom na łatwiejsze strojenie budowanych aplikacji. Artykuł prezentuje wewnętrzne mechanizmy działania optymalizatora kosztowego w systemie Oracle8i/9i, ze zwróceniem szczególnej uwagi na formułę funkcji kosztu, metody dostępu do danych, metody realizacji operacji połączenia, modele statystyczne tabel i kolumn.

Wprowadzenie do optymalizacji zapytań

Jednym z podstawowych zadań funkcjonalnych stawianych systemom baz danych jest obsługa zapytań użytkowników. Zapytania są zwykle formułowane w języku SQL, którego główną cechą jest deklaratywność, co oznacza, że użytkownik określa „co ma być znalezione", a nie „w jaki sposób ma być znalezione". W związku z tym, dla każdego otrzymanego zapytania, system zarządzania bazą danych musi wygenerować odpowiedni mikroprogram, który zrealizuje zlecone zadanie. W nomenklaturze bazodanowej, mikroprogram taki nazywany jest *planem wykonania zapytania* (query execution plan).
W ogólności, przekład deklaratywnej treści zapytania na stosowny plan wykonania nie jest jednoznaczny – często, w zależności od stopnia skomplikowania samego zapytania, istnieje kilka, kilkanaście, a nawet kilka tysięcy alternatywnych rozwiązań. Ponieważ każde z tych rozwiązań może się cechować bardzo różną wydajnością (czasem wykonania zapytania), to wybór właściwego (optymalizacja) powinien być dokonany bardzo skrupulatnie. Przyjęta metodyka wybierania planów wykonania zapytań będzie miała znaczący wpływ na wydajność całego systemu bazy danych.
Moduł systemu zarządzania bazą danych, który zajmuje się dobieraniem planów wykonania zapytań nazywa się *optymalizatorem zapytań* (query optimizer). W ramach ewolucji systemów zarządzania bazami danych, pojawiły się dwa typy optymalizatorów zapytań: regułowe (rule-based) i kosztowe (cost-based). *Optymalizatory regułowe* dysponują zbiorem reguł postępowania (odnoszonych do konkretnych rodzajów zapytań), przy pomocy których wskazują najlepszy plan wykonania zapytania. Przykładowo, gdy użytkownik dokonuje selekcji rekordów tabeli, a dla tabeli tej dostępny jest indeks, to zostanie on użyty (niezależnie od rozmiaru tabeli i indeksu czy od selektywności warunku WHERE), gdyż tak mówi jedna z reguł. Zaletami optymalizatorów regułowych są: prostota budowy i duża szybkość działania.
W przeciwieństwie do optymalizatorów regułowych, *optymalizatory kosztowe* dokonują wyboru posługując się oszacowaniem czasu realizacji każdego z możliwych planów wykonania zapytania. Oznacza to, że dla danego zapytania najpierw muszą zostać wygenerowane wszystkie alternatywne plany, a następnie dla każdego z nich zostaje analitycznie wyliczony przybliżony czas jego wykonania. W ostatnim kroku, optymalizator kosztowy wybiera ten plan, dla którego wyliczony oczekiwany czas wykonania jest najmniejszy. Aby uniezależnić się od czasu rzeczywistego, na który wpływa wiele czynników niezależnych, zamiast pojęcia czasu wykonania zapytania, posługujemy się zwykle pojęciem *kosztu wykonania zapytania* (query cost). W systemie Oracle jednostka kosztu odpowiada w przybliżeniu czasowi wykonania jednej operacji I/O w systemie operacyjnym.
Warto zwrócić uwagę na to, że sama optymalizacja musi być wykonana w bardzo krótkim czasie tak, aby nie stanowiła istotnego narzutu na całość realizacji zapytania. W środowiskach OLTP oznacza to, że optymalizacja nie powinna zabierać więcej czasu niż ułamek sekundy.

Statystyczne modele obiektów bazy danych

Analityczne wyznaczenie kosztu dla wybranego planu wykonania zapytania wymaga znajomości wielu parametrów bazy danych, np. liczby rekordów, jakie zostaną znalezione, rozmiarów łączonych tabel, liczby poziomów indeksu B*-drzewo, itd. Pozyskiwanie tych parametrów w czasie pracy optymalizatora wymagałoby realizacji wielu dodatkowych zapytań, które negatywnie wpływałyby na czas samej optymalizacji. W związku z tym, optymalizator kosztowy posługuje się statystycznymi modelami tych obiektów bazy danych,

których dotyczy zapytanie. Modele te przechowywane są m.in. w następujących tabelach słownika danych (w nawiasach nazwy odpowiednich perspektyw słownika danych): TAB$ (DBA_TABLES), IND$ (DBA_INDEXES), HIST_HEAD$ (DBA_TAB_COLUMNS), HISTGRM$ (DBA_TAB_HISTOGRAMS).
Na model statystyczny tabeli, wykorzystywany przez optymalizator kosztowy, składają się następujące parametry:

- liczba rekordów tabeli (DBA_TABLES.NUM_ROWS)
- liczba bloków bazy danych, jakie są lub były wykorzystywane przez tabelę (DBA_TABLES.BLOCKS)
- średni rozmiar każdej kolumny tabeli (DBA_TAB_COLUMNS.AVG_COL_LEN)
- minimalna i maksymalna wartość znajdująca się w każdej kolumnie (DBA_TAB_COLUMNS.LOW_VALUE, DBA_TAB_COLUMNS.HIGH_VALUE)
- liczba różnych wartości każdej kolumny (DBA_TAB_COLUMNS.NUM_DISTINCT)
- rozkład wartości kolumny – *histogram* – przedstawiający punkty podziału wartości atrybutu na równoliczne przedziały (DBA_TAB_HISTOGRAMS.ENDPOINT_ACTUAL_VALUE, DBA_TAB_HISTOGRAMS.ENDPOINT_VALUE)

Model statystyczny indeksu B*-drzewo zawiera następujące parametry:

- liczba poziomów drzewa (DBA_INDEXES.BLEVEL)
- liczba bloków liści (DBA_INDEXES.LEAF_BLOCKS)
- liczba różnych wartości klucza (DBA_INDEXES.DISTINCT_KEYS)
- współczynnik uporządkowania, o wartości bliskiej liczbie bloków tabeli, jeżeli jej rekordy są fizycznie uporządkowane zgodnie z indeksem, lub o wartości bliskiej liczbie rekordów, jeżeli rekordy nie wykazują żadnego uporządkowania fizycznego w stosunku do indeksu (DBA_INDEXES.CLUSTERING_FACTOR)

Modele statystyczne obiektów bazy danych nie są automatycznie budowanie ani uaktualniane – jest to zadaniem administratora. W celu skonstruowania pełnego modelu statystycznego tabeli należy wykonać następujące polecenie ANALYZE:

```
ANALYZE TABLE <NAZWA_TABELI>
ESTIMATE STATISTICS
FOR ALL COLUMNS
```

lub wykorzystać procedurę biblioteczną DBMS_STATS.GATHER_TABLE_STATS().

Dostępne narzędzia monitorowania pracy optymalizatora zapytań

Optymalizator pracuje w sposób niezauważalny dla zwykłego użytkownika. Jest automatycznie wywoływany zawsze wtedy, gdy w systemie pojawia się nowe zapytanie. Istnieją jednak sposoby monitorowania jego działania, pozwalające programistom i administratorom obserwować metodę i wynik podejmowania decyzji optymalizacyjnych.
Na żądanie użytkownika, system zarządzania bazą danych może udostępnić wybrany przez optymalizator plan wykonania zapytania. Do tego celu służy polecenie EXPLAIN PLAN, lecz wygodniej jest skorzystać z bazującej na nim funkcji AUTOTRACE programu

SQL*Plus. W celu uaktywnienia funkcji AUTOTRACE, dzięki której po wykonaniu każdego zapytania program SQL*Plus wyświetli jego plan, należy wykonać polecenie:

```
SET AUTOTRACE ON EXPLAIN
```

Poniżej przedstawiono przykładowe zapytanie wraz z opisem najlepszego planu jego wykonania, wyświetlonym przez funkcję AUTOTRACE programu SQL*Plus:

```
SQL> select max(pensja)
  2  from pracownicy
  3  where id between 15 and 45;

MAX(PENSJA)
-----------
       5045


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=4 Card=1 Bytes=7)
   1    0   SORT (AGGREGATE)
   2    1     TABLE ACCESS (BY INDEX ROWID) OF 'PRACOWNICY' (Cost=4 Card=31 Bytes=217)
   3    2       INDEX (RANGE SCAN) OF 'I4' (NON-UNIQUE) (Cost=3 Card=31)
```

Z powyższego planu wykonania zapytania możemy odczytać m.in., że optymalizator kosztowy postanowił wykorzystać indeks I4, dzięki któremu trafił do odpowiednich rekordów tabeli PRACOWNICY; dostęp do indeksu kosztował trzy jednostki kosztu, dostęp do tabeli powiększył ten koszt o jedną jednostkę; funkcja grupowa MAX() otrzymała 31 rekordów, a zwróciła jeden; łączny rozmiar rekordów odczytanych z tabeli wynosił 217 bajtów.
Innym, o wiele bardziej skomplikowanym narzędziem monitorowania pracy optymalizatora jest zapis raportu z jego pracy. W celu uaktywnienia tej funkcji należy wykonać następujące polecenie ALTER SESSION:

```
ALTER SESSION SET EVENTS ‘10053 TRACE NAME CONTEXT FOREVER’
```

Od tej chwili, dla każdego nowego zapytania, które będzie przetwarzać optymalizator kosztowy, w pliku śladu użytkownika (katalog `user_dump_dest`) zostanie zapisany przebieg procesu podejmowania decyzji o wyborze najlepszego planu. Poniżej przedstawiono niewielki fragment pliku śladu użytkownika, w którym odnotowane zostały działania optymalizatora.

```
SINGLE TABLE ACCESS PATH
Column:         ID  Col#: 1      Table: PRACOWNICY   Alias: PRACOWNICY
    NDV: 50000     NULLS: 0         DENS: 2.0000e-005
  TABLE: PRACOWNICY     ORIG CDN: 50000  CMPTD CDN: 31
  Access path: tsc  Resc:  148  Resp:  148
  Access path: index (scan)
      INDEX#: 2875  TABLE: PRACOWNICY
      CST: 4  IXSEL:  6.1444e-004  TBSEL:  6.1444e-004
  BEST_CST: 4.00  PATH: 4  Degree:  1
```

Z powyższego raportu wynika m.in., że optymalizator kosztowy rozważał dwa plany wykonania zapytania: pełen odczyt tabeli (access path: tsc) oraz dostęp poprzez indeks (access path: index (scan)); oszacowany koszt pełnego odczytu tabeli to 148 jednostek, natomiast koszt dostępu poprzez indeks to cztery jednostki; selektywność predykatu selekcji to 0.0006144 (tzn. 31 rekordów z 50000).

Estymacja kosztu planu wykonania zapytania

Skuteczność działania optymalizatora kosztowego jest uzależniona od precyzji dokonywanych przez niego estymacji kosztów planów wykonania zapytań. W dalszej części artykułu przedstawimy podstawowe metody szacowania kosztów dla różnych przykładów planów.

Pełen odczyt tabeli

Pełen odczyt tabeli (full table scan) polega na odczytaniu wszystkich bloków tabeli, począwszy od bloku nagłówka segmentu tabeli, a skończywszy na ostatnim bloku kiedykolwiek wykorzystywanym przez tabelę do zapisu rekordów (wskaźnik wysokiej wody). Ponieważ podczas wykonywania pełnego odczytu tabeli, z dysku pobierane są sąsiednie bloki, dlatego możliwe jest wykorzystanie dyskowego odczytu sekwencyjnego (wieloblokowego), w ramach którego jedno żądanie we/wy powoduje jednoczesne pobranie wielu kolejnych bloków. Czas wykonania jednego wieloblokowego odczytu sekwencyjnego jest w przybliżeniu równy czasowi jednego swobodnego odczytu jednoblokowego, ponieważ odczyt sekwencyjny nie wymaga dodatkowego przesuwania głowicy dyskowej (najbardziej czasochłonny element kosztu dostępu do dysku). W systemie Oracle, administrator może definiować rozmiar odczytów sekwencyjnych przy pomocy parametru inicjalizacyjnego `db_file_multiblock_read_count`. Jego wartość mierzona jest w blokach bazy danych (rozmiar definiowany przez `db_block_size`). Domyślnie odczyty sekwencyjne są ośmioblokowe.

Rozważmy następujący przykład pełnego odczytu tabeli. Dana jest tabela PRACOWNICY, zawierająca 972 bloki (poniżej wskaźnika wysokiej wody – BLOCKS). Parametr inicjalizacyjny `db_file_multiblock_read_count` posiada wartość 8. Tabela nie posiada żadnych indeksów. Użytkownik wykonuje następujące zapytanie:

```
select max(pensja)
from pracownicy
where id between 100 and 1100
```

Ponieważ nie dysponujemy żadnymi indeksami, jedyną metodą wykonania powyższego zapytania będzie realizacja pełnego odczytu tabeli. Koszt takiej operacji szacowany jest według poniższego wzoru:

$$\text{cost} = \left\lceil K_m \frac{blocks + 1}{db\_file\_multiblock\_read\_count} \right\rceil$$

gdzie:

- *blocks* to liczba bloków tabeli
- '*+1*' dotyczy jednego bloku nagłówka segmentu tabeli
- *db_file_multiblock_read_count* to ustawiony w systemie rozmiar wieloblokowych odczytów sekwencyjnych
- K_m to współczynnik korygujący, zależny od wartości *db_file_multiblock_read_count* (tabelka 1); reprezentuje on ryzyko wydłużenia odczytu sekwencyjnego w przypadku napotkania końca cylindra dyskowego lub końca ekstentu; ryzyko to zależne jest od rozmiaru odczytu

Tab. 1 Wartości współczynnika korygującego K_m dla odczytów sekwencyjnych

db_file_multiblock_read_count	K_m
1	0,596500
2	0,755967
3	0,868342
4	0,958064
5	1,033995
6	1,100481
7	1,160019
8	1,214190
9	1,264068
10	1,310419
11	1,353812

Zgodnie z powyższym wzorem, koszt pełnego odczytu tabeli PRACOWNICY wyniesie:
1.21419 * (972+1) /8 = 148

Poniżej przedstawiono wynik naszych obliczeń dostarczony przez optymalizator kosztowy:

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=148 Card=1 Bytes=7)
   1    0   SORT (AGGREGATE)
   2    1     TABLE ACCESS (FULL) OF 'PRACOWNICY' (Cost=148 Card=1001 Bytes=7007)
```

Dostęp do tabeli poprzez indeks B*-drzewo

Indeksy B*-drzewo służą do poprawy efektywności dostępu do pojedynczych rekordów tabeli, spełniających równościowe lub nierównościowe warunki selekcji. Wartość kosztu dostępu do tabeli poprzez indeks B*-drzewo wynika z odczytu węzłów drzewa na drodze od korzenia do liści, odczytu liści zawierających adresy ROWID rekordów, oraz odczytu samych rekordów z tabeli.

Rozważmy następujący przykład dostępu do tabeli poprzez indeks B*-drzewo. Dana jest tabela PRACOWNICY, zawierająca 50000 rekordów (NUM_ROWS) w 972 blokach (BLOCKS). Parametr inicjalizacyjny `db_file_multiblock_read_count` posiada wartość 8. Tabela posiada indeks I_ID, zbudowany na kolumnie ID. Model statystyczny tego indeksu to: 2 poziomy (BLEVEL), 480 bloków liści (LEAF_BLOCKS), współczynnik zgrupowania = 970 (CLUSTERING_FACTOR). Histogram wartości atrybutu ID został przedstawiony w tabelce 2. Z histogramu tego możemy odczytać np., że w tabeli PRACOWNICY jest tyle samo rekordów posiadających wartości ID pomiędzy 1 a 667, co rekordów posiadających wartości ID pomiędzy 667 a 1334.

Tab 2. Histogram kolumny ID (razem 75 przedziałów)

ENDPOINT_VALUE
1
667
1334
2001
2668
3335
4002
4669
5336
6003
6670
7337
...
50000

Załóżmy, że użytkownik wykonuje następujące zapytanie:

```
select max(pensja)
from pracownicy
where id between 100 and 1100
```

Do realizacji powyższego zapytania może być użyty jeden z dwóch planów: pełen odczyt tabeli lub dostęp do szukanych rekordów przy pomocy indeksu I_ID. Koszt pełnego odczytu tabeli został wyliczony w poprzednim podrozdziale. Natomiast dostęp do rekordów tabeli poprzez indeks I_ID będzie wymagał następujących kroków:

1. Odczyt bloku nagłówka segmentu indeksu (koszt = 1)
2. Odczyt korzenia indeksu (koszt = 1)
3. Odczyt procentu liści indeksu, równego selektywności warunku selekcji (`id between 100 and 1100`); według histogramu, w każdym przedziale znajduje się 50000/75 rekordów = 667; wartości z przedziału pomiędzy 100 a 1100 mieszczą się w dwóch zakresach histogramu: <1;667) i <667;1334), w pierwszym stanowią (667-100)/667=85%, w drugim (1100-667+1)/667=65,07%; w związku z tym, oczekiwana liczba rekordów spełniających warunek selekcji wynosi 667*(0.85+0.6507) = 1001; zatem w celu odczytania z indeksu tej liczby wskaźników do rekordów wymagane jest pobranie (1001/50000)*LEAF_BLOCKS = 10 (koszt = 10)
4. Odczyt bloków tabeli, wskazywanych przez wskaźniki z liści indeksu. Liczba odczytów wyliczana jest w oparciu o współczynnik zgrupowania indeksu (CLUSTERING_FACTOR) w następujący sposób: (1001/50000)*CLUSTERING_FACTOR = 20 (koszt = 20)

Łączny koszt dostępu do tabeli poprzez indeks B*-drzewo wyniósł w naszym przykładzie 32. Ponieważ jest to mniej, niż koszt pełnego odczytu tabeli, dlatego optymalizator wybiera ten plan do realizacji zapytania:

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=32 Card=1 Bytes=7)
   1    0   SORT (AGGREGATE)
   2    1     TABLE ACCESS (BY INDEX ROWID) OF 'PRACOWNICY' (Cost=32 Card=1001 Bytes=7007)
   3    2       INDEX (RANGE SCAN) OF 'I_ID' (NON-UNIQUE) (Cost=12 Card=1001)
```

Pełen odczyt indeksu bez dostępu do tabeli

Pewne typy zapytań mogą zostać w pełni wykonane bez konieczności dostępu do tabeli bazowej, a jedynie w oparciu o dane zawarte w indeksach. Rozważmy przykład poniższego zapytania do tabeli PRACOWNICY, na której atrybucie ID zbudowany jest indeks B*-drzewo o nazwie I_ID:

```
select avg(id)
from pracownicy
where id between 100 and 20000
```

Zauważmy, że zapytanie to może zostać wykonane przy użyciu samego indeksu I_ID, bez konieczności sięgania do tabeli bazowej PRACOWNICY. Ponadto, indeks I_ID może być użyty na jeden z dwóch sposobów: (1) z zastosowaniem tradycyjnej nawigacji korzeń-liście w celu wyszukania liści zawierających szukane wartości (tzw. odczyt zakresowy) lub (2) z zastosowaniem wieloblokowego odczytu sekwencyjnego całej struktury indeksu, a następnie wyławiania szukanych wartości z liści. Koszt pełnego odczytu indeksu wyliczany jest analogicznie do kosztu pełnego odczytu tabeli:

$$\text{cost} = \left\lceil K_m \frac{blocks + 1}{db\_file\_multiblock\_read\_count} \right\rceil$$

cost = 1.21419 * (480+1+1) /8 = 73

Łączny koszt pełnego odczytu indeksu B*-drzewo wyniósł w naszym przykładzie 73. Ponieważ w tym przypadku jest to mniej, niż koszt zakresowego odczytu indeksu oraz mniej niż koszt pełnego odczytu całej tabeli bazowej, dlatego optymalizator wybiera ten plan do realizacji zapytania:

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=73 Card=1 Bytes=4)
   1    0   SORT (AGGREGATE)
   2    1     INDEX (FAST FULL SCAN) OF 'I_ID' (NON-UNIQUE) (Cost=73 Card=19891 Bytes=79564)
```

Sortowanie rekordów

Operacja sortowania rekordów jest realizowana m.in. na potrzeby klauzul ORDER BY, GROUP BY, DISTINCT, połączenia, itd. Algorytm sortowania wykorzystywany przez Oracle to wielokanałowy Merge Sort, polegający na sortowaniu (sort) fragmentów tabeli w buforze pamięci operacyjnej i zapisywaniu ich w segmencie tymczasowym, a następnie na stopniowym łączeniu (merge) posortowanych fragmentów w większe, aż do osiągnięcia jednego fragmentu wynikowego. Maksymalny rozmiar pamięci operacyjnej przeznaczonej na

sortowanie jest określany przy pomocy parametru inicjalizacyjnego `sort_area_size`. Rozmiar pojedynczego okna wykorzystywanego w fazie Merge jest określany przy pomocy parametru inicjalizacyjnego `sort_multiblock_read_count` (domyślnie 2). System zarządzania bazą danych wykorzystuje technikę bezpośredniego (szybszego) dostępu do segmentu tymczasowego, polegającą na pominięciu bufora danych i procesu zapisującego DBW.

Rozważmy następujący przykład sortowania danych. Użytkownik wykonuje sortowanie kolumn NAZWISKO i STANOWISKO (według kolumny NAZWISKO) z tabeli PRACOWNICY. W modelu statystycznym, tabela PRACOWNICY posiada 50000 rekordów, kolumna NAZWISKO posiada rozmiar 10 bajtów, a kolumna STANOWISKO rozmiar 9 bajtów. Parametr `sort_area_size` posiada wartość 50000, `sort_multiblock_read_count` wartość 2, a `db_block_size` wartość 2048. Poniżej przedstawiono treść zapytania użytkownika.

```
select nazwisko, stanowisko
from pracownicy
order by nazwisko
```

Rozmiar sortowanych rekordów określany jest jako suma rozmiarów wszystkich sortowanych kolumn, plus po jednym bajcie na kolumnę, plus 10 bajtów sygnatury haszowej, która faktycznie wykorzystywana jest do operacji porównań. W omawianym przypadku oznacza to, że sortowaniu podlegać będą rekordy 10+9+1+1+10 = 31 bajtowe. Rekordów tych będzie 50000, co daje łączny ich rozmiar 50000*31=1550000 bajtów, podzielonych na 2048-bajtowe bloki, których będzie 1550000/(2048-24)) = 766 (24 bajty nagłówka bloku).

Znając rozmiar bufora w pamięci operacyjnej, możemy obliczyć, na ile fragmentarycznych sortowań zostaną podzielone dane: 1550000/(50000*0.68) = 46 (0.68 wynika ze sposobu gospodarowania pamięcią buforową przez Oracle – zawsze ok. 32% konsumowane jest na potrzeby systemowe). Następnie wyznaczymy liczbę kanałów łączenia, jakie zmieszczą się w pamięci operacyjnej. Każdy kanał zajmuje `db_block_size*sort_multiblock_read_count` = 4096 bajtów, a Oracle na kanały przeznacza 2/3 obszaru pamięci (reszta wykorzystywana jako bufory do zapisu). To daje 2/3*34428/4096 = 5 kanałów. W związku z tym, algorytm Merge Sort będzie wymagał $\log_5 46 = 3$ faz łączenia. Koszt każdej fazy łączenia to odczyt wszystkich sortowanych bloków, a następnie ich wieloblokowy zapis sekwencyjny (rozmiar zapisu równy liczbie kanałów). To daje koszt pojedynczej fazy: 766+766/5 = 920. Koszt całego sortowania wyniesie ½ * (3*920 + 766) = 1763, gdzie ½ reprezentuje poprawę wydajności wynikającą z dostępów bezpośrednich. Należy jeszcze uwzględnić koszt początkowego pełnego odczytu tabeli (148), aby uzyskać ostateczny koszt planu wykonania zapytania: 1911. Poniżej przedstawiono wynik dostarczony przez optymalizator kosztowy:

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=1911 Card=50000 Bytes=950000)
   1    0   SORT (ORDER BY) (Cost=1911 Card=50000 Bytes=950000)
   2    1     TABLE ACCESS (FULL) OF 'PRACOWNICY' (Cost=148 Card=50000 Bytes=950000)
```

Pośrednie wyniki przedstawionej estymacji prowadzonej przez optymalizator są także zapisywane w pliku śladu użytkownika:

```
Join order[1]: PRACOWNICY [PRACOWNICY]
ORDER BY sort
    SORT resource      Sort statistics
      Sort width:             5 Area size:       34428   Degree: 1
      Blocks to Sort:       766 Row size:           31 Rows:      50000
      Initial runs:          46 Merge passes:        3 Cost / pass:        920
      Total sort cost: 1763
Best so far: TABLE#: 0  CST:       1911  CDN:      50000  BYTES:     950000
```

Połączenie tabel

Operacja połączenia tabel w bazie danych Oracle jest realizowana jedną z trzech metod: Nested Loops, Sort Merge lub Hash Join. Wyboru najlepszej z nich dokonuje optymalizator kosztowy, odpowiednio estymując koszt planu wykonania zapytania. Poniżej przedstawiono sposób wyliczania kosztu połączenia metodą Sort Merge dla następującego zapytania użytkownika:

```
select p1.nazwisko, p2.nazwisko
from pracownicy p1, pracownicy p2
where p1.stanowisko=p2.stanowisko
```

Operacja połączenia metodą Sort Merge polega na wstępnym posortowaniu łączonych rekordów w obu tabelach, a następnie na wykonaniu dwukanałowego łączenia odpowiadających sobie rekordów. Koszt takiej operacji połączenia stanowi sumę kosztów odczytu każdej z tabel oraz ich sortowania. Ponieważ rozmiar sortowanego rekordu jest taki sam, jak w poprzednim podrozdziale, dlatego sortowanie będzie kosztować łącznie 1763 * 2 = 3526, a kosz pełnego odczytu obu łączonych tabel to 2 * 148 = 296. Ostateczny koszt realizacji operacji połączenia metodą Sort Merge to 3526 + 296 = 3822 i jest to w tym przypadku najniższy koszt, jaki znalazł optymalizator kosztowy:

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=3822 Card=795468474 Bytes=30227802012)
   1    0   MERGE JOIN (Cost=3822 Card=795468474 Bytes=30227802012)
   2    1     SORT (JOIN) (Cost=1911 Card=50000 Bytes=950000)
   3    2       TABLE ACCESS (FULL) OF 'PRACOWNICY' (Cost=148 Card=50000 Bytes=950000)

   4    1     SORT (JOIN) (Cost=1911 Card=50000 Bytes=950000)
   5    4       TABLE ACCESS (FULL) OF 'PRACOWNICY' (Cost=148 Card=50000 Bytes=950000)
```

Pośrednie wyniki przedstawionej estymacji prowadzonej przez optymalizator są także zapisywane w pliku śladu użytkownika:

```
SM Join
  Outer table:
    resc: 148  cdn: 50000  rcz: 19  deg: 1  resp: 148
  Inner table: PRACOWNICY
    resc: 148  cdn: 50000  rcz: 19  deg:  1  resp: 148
    SORT resource      Sort statistics
      Sort width:             5 Area size:       34428   Degree: 1
      Blocks to Sort:       766 Row size:           31 Rows:      50000
      Initial runs:          46 Merge passes:        3 Cost / pass:        920
      Total sort cost: 1763
    SORT resource      Sort statistics
      Sort width:             5 Area size:       34428   Degree: 1
      Blocks to Sort:       766 Row size:           31 Rows:      50000
      Initial runs:          46 Merge passes:        3 Cost / pass:        920
      Total sort cost: 1763
  Merge join  Cost:  3822  Resp:  3822
```

Podsumowanie

Trafność estymacji kosztu planu wykonania zapytania ma wpływ na wybory dokonywane przez optymalizator. Z kolei trafność tych wyborów determinuje całkowitą wydajność systemu. W artykule przedstawiono ogólne zasady wyznaczania kosztów w systemie Oracle8i/9i. Zwrócono uwagę na znaczenie uwzględniania m.in. rozmiarów danych, rozkładów ich wartości, ilości dostępnej pamięci operacyjnej, charakterystyk indeksów oraz natury dostępów I/O.