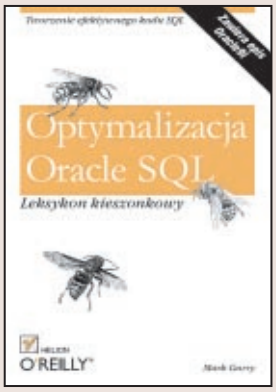
Optymalizacja Oracle SQL. Leksykon kieszonkowy



Autor: Mark Gurry
Tłumaczenie: Bartłomiej Garbacz
ISBN: 83-7197-983-5
Tytuł oryginału: Oracle SQL Tuning. Pocket Reference
Format: B5, stron: 128

Niezoptymalizowane polecenia SQL są jednym z głównych czynników powodujących mało wydajne działanie systemu bazy danych. W niniejszej książce Mark Gurry dzieli się z Czytelnikiem swoimi przemyśleniami dotyczącymi problemu optymalizacji. Autor prezentuje rozwiązania wielu typowych problemów za pomocą wbudowanych w Oracle'a optymalizatorów. Omawia między innymi:

- Problem wyboru optymalizatora
- Działanie optymalizatora regułowego (rule-based)
- Działanie optymalizatora kosztowego (cost-based)
- Problemy wspólne dla obu optymalizatorów

„Optymalizacja Oracle SQL. Leksykon kieszonkowy" zaoszczędzi wiele czasu poświęconego na pisanie wydajnych zapytań. Powinna się znaleźć w biblioteczce każdego administratora i użytkownika Oracle'a.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Problemy i ich rozwiązania w przypadku optymalizatora kosztowego

Optymalizator kosztowy uległ znaczącemu ulepszeniu w porównaniu ze swoją pierwotną wersją. Autor sugeruje, aby w każdym ośrodku, w którym od niedawna używa się systemu Oracle, korzystano właśnie z optymalizatora kosztowego. Ponadto warto pomyśleć także o tym, aby w ośrodkach, w których korzysta się obecnie z optymalizatora regułowego, przygotowano stosowny plan migracji do optymalizatora kosztowego. Istnieją jednak pewne kwestie związane z tym rodzajem optymalizatora, o których trzeba pamiętać. W tabeli 3 wymieniono najczęściej powtarzające się problemy (wraz z częstotliwością ich występowania), jakie Autorowi udało się zaobserwować.

Tabela 3. Często powtarzające się problemy w przypadku optymalizatora kosztowego

Problem	Przypadków
1. Problem asymetrii	30%
2. Analizowanie nieodpowiednich danych	25%
3. Wspólne używanie optymalizatorów przy złączeniach	20%
4. Wybieranie nieodpowiedniego indeksu	20%
5. Złączanie zbyt wielu tabel	< 5%
6. Nieodpowiednie ustawienia parametrów w pliku *INIT.ORA*	< 5%

Problem pierwszy: problem asymetrii

Załóżmy, że problem dotyczy systemu, w którym istnieje tabela `trans` o jednej z kolumn noszącej nazwę `status`. Dopuszczalne są dwie wartości kolumny: `O` dla oznaczenia transakcji otwartych (*open transactions*), które nie zostały jeszcze zaksięgowane, oraz `C` dla ozna-

czenia transakcji zamkniętych (*closed transactions*), które zostały już zaksięgowane i nie wymagają dalszej obsługi. Istnieje ponad milion rekordów, które posiadają status `C` i zawsze tylko 100 wierszy, które mają status `O`.

Utworzono następujące polecenie SQL, które jest wykonywane codziennie kilkaset razy, jednak czas odpowiedzi nie jest zadowalający:

```
SELECT acct_no, customer, product, trans_date, amt
  FROM trans
 WHERE status = 'O';
```

`Czas odpowiedzi: 16,308 sekund`

W przykładzie tym — wziętym z życia — optymalizator kosztowy zdecydował, że system Oracle powinien przeprowadzić przegląd całej tabeli (*full table scan*). Stało się tak dlatego, że optymalizator posiadał informację o liczbie różnych wartości, jakie przyjmować mogły pola w kolumnie `STATUS`, ale nie posiadał informacji o liczbie rekordów posiadających każdą z tych wartości. W konsekwencji optymalizator założył równomierny rozkład danych (50/50) dla każdej z dwóch wartości `O` oraz `C`. Przy takim założeniu system Oracle przeprowadza przegląd całej tabeli w celu pobrania danych o otwartych transakcjach.

System Oracle będzie posiadał informację o asymetrii rozkładu danych, czyli liczbie wierszy posiadających określoną wartość w zaindeksowanych kolumnach, jeśli podczas wykonywania polecenia `ANALYZE` lub w momencie wywoływania pakietu `DBMS_STATS` poda się opcję `FOR ALL INDEXED COLUMNS`. Załóżmy teraz, że kolumna `status` posiada indeks. W celu zanalizowania tabeli użyć należy następującego polecenia:

```
ANALYZE TABLE TRANS COMPUTE STATISTICS
        FOR ALL INDEXED COLUMNS
```

Po przeprowadzeniu analizy tabeli i obliczeniu statystyk dla wszystkich zaindeksowanych kolumn, optymalizator kosztowy będzie posiadał informację o tym, że tylko w około 100 wierszach występuje wartość `O`, co sprawi, że w przypadku tej kolumny użyje indeksu. W rezultacie otrzymany zostanie dużo krótszy czas odpowiedzi:

Czas odpowiedzi: 0,259 sekund

Zazwyczaj, optymalizator kosztowy przeprowadza przegląd całej tabeli, jeśli dana wartość kolumny występuje w ponad 12% wierszy tabeli, a korzysta z indeksu, gdy wartość występuje w mniej niż 12% wierszy. Wybór dokonywany przez optymalizator kosztowy nie opiera się na tak prostej regule, jednak praktyka wskazuje, że jest to typowe jego zachowanie.

Przed wprowadzeniem systemu Oracle9*i* — jeśli polecenie wykorzystywało zmienne dowiązane — problem asymetrii wciąż mógł występować nawet wtedy, gdy użyto opcji `FOR ALL INDEXED COLUMNS`. Warto przyjrzeć się następującemu poleceniu:

```
local_status := 'O';

SELECT acct_no, customer, product, trans_date, amt
  FROM trans
 WHERE status = local_status;
```

Czas odpowiedzi: 16,608 sekund

Czas odpowiedzi jest zbliżony do tego, który występował w przypadku nieużywania opcji `FOR ALL INDEXED COLUMNS`. Problem występuje dlatego, że optymalizator kosztowy nie zna wartości zmiennej dowiązanej w momencie określania planu wykonania. Ogólnie rzecz biorąc — w celu uniknięcia problemu asymetrii należy:

- wartości literałów zapisywać w kodzie bezpośrednio (na przykład można użyć `WHERE STATUS = 'O'` zamiast `WHERE STATUS = local_status`);
- zawsze wykonywać analizę z opcją `FOR ALL INDEXED COLUMNS`.

Jeśli mimo to wciąż występują problemy wydajnościowe związane z nieużywaniem przez optymalizator kosztowy indeksu z powodu zmiennych dowiązanych, a nie ma możliwości zmiany kodu źródłowego, pozostaje próba usunięcia statystyk indeksu za pomocą polecenia:

```
ANALYZE INDEX
```

```
TRANS_STATUS_NDX
DELETE STATISTICS;
```

Usunięcie statystyk indeksu poprawia sytuację, gdyż wymusza zachowanie stosowane przez optymalizator regułowy, który zawsze korzysta z istniejących indeksów (zamiast przeglądu całej tabeli).

UWAGA

W systemie Oracle9*i* wartość zmiennych dowiązanych jest określana przed podjęciem decyzji o planie wykonania, co eliminuje konieczność bezpośredniego zapisywania w kodzie wartości literałów.

Problem drugi: analizowanie nieodpowiednich danych

Autor miał styczność z wieloma systemami, w których problemy z wydajnością wynikały z tego, że tabele i indeksy nie były analizowane w czasie, gdy zawierały typowe ilości danych. Optymalizator kosztowy musi posiadać dokładne informacje (a w tym informacje o objętości danych), aby mógł określić efektywny plan wykonania.

Sytuacje, w których statystyki mogą zostać utracone lub stać się nieaktualne, stanowić może ponowne tworzenie tabeli lub jej przenoszenie, dodawanie indeksu lub tworzenie nowego środowiska. Na przykład administrator może zapomnieć o ponownym utworzeniu statystyk po przeniesieniu schematu bazy danych do środowiska produkcyjnego. Problemy pojawiają się także wtedy, gdy administrator nie posiada wystarczających informacji o bazie danych, którą zarządza i analizuje tabelę w momencie, gdy jest pusta, a nie wtedy, gdy po krótkim okresie czasu ma ona setki lub tysiące wierszy.

Sposób sprawdzenia daty ostatniej analizy

W celu sprawdzenia tego, które tabele, indeksy i partycje zostały przeanalizowane i kiedy zostało to zrobione po raz ostatni, można wykonać

zapytanie pobierające wartość kolumny LAST_ANALYZED z różnych perspektyw USER_XXX. Na przykład w celu określenia daty ostatniej analizy wszystkich tabel należy wykonać:

```
SELECT table_name, num_rows, last_analyzed
  FROM user_tables;
```

Oprócz USER_TABLES istnieje wiele innych perspektyw, dzięki którym można sprawdzić datę analizy różnych obiektów. W celu otrzymania pełnej listy perspektyw zawierających kolumnę LAST_ANALYZED należy wykonać następujące zapytanie:

```
SELECT table_name
  FROM all_tab_columns
 WHERE column_name = 'LAST_ANALYZED';
```

Oczywiście nie chodzi o to, aby analizy z opcją COMPUTE przeprowadzać jak najczęściej. Takie postępowanie może spowodować, że dostrojone polecenie SQL ulegnie rozstrojeniu.

Decyzja o czasie analizy

Ponowne analizowanie tabel i indeksów może być równie niebezpieczne, jak dostosowywanie indeksów i w idealnej sytuacji powinno być przeprowadzane na kopii bazy produkcyjnej przed ostatecznym wprowadzeniem zmian w faktycznej bazie produkcyjnej.

Oprogramowanie firmy Peoplesoft jest przykładem aplikacji, która korzysta z tymczasowych tabel do przechowywania danych, których nazwy kończą się wyrażeniem _TMP. Kiedy rozpoczyna się wykonywanie procesu wsadowego, każda z tych tabel jest zazwyczaj pusta. W czasie wykonaniu każdego etapu procesu wsadowego na tabelach wykonywane są operacje wstawiania i uaktualniania danych.

Ostatnia faza procesu polega na wstawieniu danych do głównych tabel obsługi transakcji aplikacji Poplesoft poprzez ekstrakcję danych z tabel tymczasowych. Po zakończeniu procesu wsadowego zwykle wszystkie wiersze są z tabel tymczasowych usuwane. Transakcje związane z tymi

tabelami nie są zatwierdzone aż do zakończenia procesu, kiedy nie ma już w nich żadnych danych.

Kiedy wydaje się polecenie `ANALYZE` względem tabel tymczasowych, zazwyczaj są one puste. Kiedy optymalizator kosztowy otrzymuje informację o zerowej liczbie wierszy, automatycznie podejmuje decyzję o przeglądzie całej tabeli oraz zastosowaniu złączenia kartezjańskiego. W celu obejścia tego problemu Autor sugeruje zapełnienie tabel tymczasowych danymi w celu przeprowadzenia analizy. Potem można tabele opróżnić z danych i rozpocząć normalne przetwarzanie. Opróżnienie tabel (polecenie `TRUNCATE`) nie powoduje usunięcia statystyk.

Polecenia `INSERT` i `UPDATE` języka SQL używane przez aplikację w celu wstawienia danych do tabel tymczasowych można sprawdzić stosując procedurę śledzenia (*tracing*) względem procesu wsadowego, który wstawia i uaktualnia dane. Tych samych poleceń SQL można użyć do własnoręcznego zapełnienia tabel danymi.

Przy zastosowaniu takiego ujęcia problemu w jednym z dużych ośrodków w Australii, który korzystał z oprogramowania Peoplesoft, czas wykonania procesu wsadowego spadł z 36 godzin do mniej niż 30 minut.

Jeśli analizowanie tabel przechowujących dane tymczasowe zawierające produkcyjne ilości danych nie rozwiązuje problemów wydajnościowych, warto rozważyć usunięcie statystyk odpowiednich dla tych tabel. Wymusza to zastosowanie względem poleceń SQL, które odwołują się do tych tabel, zasad działania optymalizatora regułowego. Statystyki można usunąć korzystając z polecenia `ANALYZE` *`nazwatab`* `DELETE STATISTICS`. Po ich usunięciu ważną sprawą jest zapewnienie tego, aby tabele te nie były używane w złączeniach z tabelami, które posiadają statystyki. Należy także zapewnić to, aby względem niezanalizowanych tabel nie były używane indeksy posiadające statystyki. Jeśli tabele tymczasowe są wykorzystywane oddzielnie i złączenia występują tylko pomiędzy nimi samymi, wtedy preferowanym podejściem jest często wykorzystanie zasad działania optymalizatora regułowego.

Problem trzeci: wspólne używanie optymalizatorów przy złączeniach

Jak wspomniano wcześniej, w sytuacji, gdy tabele podlegają złączeniu i jedna z nich zostanie zanalizowana, zaś pozostałe tabele nie, optymalizator kosztowy działa najmniej korzystnie.

Analizując tabele oraz indeksy przy użyciu procedury `DBMS_STATS.GATHER_SCHEMA_STATS` oraz procedury `GATHER_TABLE_STATS` należy pamiętać o podaniu opcji `CASCADE=>TRUE`. Domyślnie pakiet `DBMS_STATS` zbiera statystyki jedynie dla tabel. Posiadanie statystyk dla tabel, ale nie dla ich indeksów, także może spowodować obieranie przez optymalizator kosztowy niewydajnych planów wykonania.

Jeden z przypadków wystąpienia takiego problemu, z jakim zetknął się Autor, miał miejsce w systemie posiadającym niezanalizowaną tabelę `trans` oraz zanalizowaną tabelę `acct`. Administrator w celu usunięcia danych ponownie utworzył tabelę `trans`, ale zapomniał wykonać analizę. Poniższy przykład ilustruje wydajność wykonania złączenia obu tabel:

```
SELECT a.account_name, SUM(b.amount)
  FROM trans b, acct a
 WHERE b.trans_date > sysdate - 7
   AND a.act_id = b.acct_id
   AND a.acct_status = 'A'
 GROUP BY account_name;

SORT GROUP BY
     NESTED LOOPS
          TABLE ACCESS BY ROWID ACCT
              INDEX UNIQUE SCAN ACCT_PK
          TABLE ACCESS FULL TRANS
```

`Czas odpowiedzi: 410 sekund`

Czas odpowiedzi uległ znacznemu skróceniu po zanalizowaniu tabeli `trans` za pomocą poniższego polecenia:

```
ANALYZE TABLE trans ESTIMATE STATISTICS
     SAMPLE 3 PERCENT
     FOR ALL INDEXED COLUMNS
```

Nowy plan wykonania oraz czas odpowiedzi przedstawiały się następująco:

```
SORT GROUP BY
     NESTED LOOPS
          TABLE ACCESS BY ROWID ACCT
              INDEX UNIQUE SCAN ACCT_PK
          TABLE ACCESS BY ROWID TRANS
              INDEX RANGE SCAN TRANS_NDX1
```

Czas odpowiedzi: 3,1 sekund

W innym systemie, który Autor także dostrajał, twórca oprogramowania zarządzającego informacjami kadrowymi zalecił analizowanie tylko indeksów, a tabel — nie. Dostawca oprogramowania opracował aplikację dla systemu baz danych firmy Microsoft SQL Server i przeniósł ją do systemu Oracle. Rezultat analizowania samych indeksów miał daleko sięgający — negatywny — wpływ na wydajność. Na przykład:

```
SELECT COUNT(*)
  FROM trans
 WHERE acct_id     = 9
   AND cost_center = 'VIC';

TRANS_IDX2 jest na ACCT_ID
TRANS_NDX3 jest na COST_CENTER
```

Czas odpowiedzi: 77,3 sekund

Ironią losu było to, że twórca oprogramowania obarczał winą system Oracle. Twierdził bowiem, że jego wydajność jest niższa od systemu SQL Server. Po zanalizowaniu tabel oraz indeksów czas odpowiedzi polecenia SQL został drastycznie zmniejszony do 0,415 sekundy. Czas odpowiedzi wielu innych poleceń SQL także znacznie się zmniejszył.

Morał płynący z tej historii mógłby brzmieć: strojenie systemu Oracle powinno być domeną ekspertów systemu Oracle, zaś eksperci systemu

SQL Server powinni przy tym systemie pozostać. Jednakże specjaliści z sektora IT — coraz bardziej mobilni i pracujący z wieloma systemami baz danych — powinni być może po prostu z większą uwagą czytać podręczniki, kiedy przyjmują na siebie obowiązek strojenia nowej bazy danych.

Problem czwarty: wybieranie nieodpowiedniego indeksu

Optymalizator kosztowy wybiera czasem indeks podrzędny, nawet jeśli wydaje się sprawą oczywistą, że użyty być powinien inny indeks. Warto przyjrzeć się następującemu wyrażeniu `WHERE` występującemu w oprogramowaniu Peoplesoft:

```
where business_unit     = :5
  and ledger            = :6
  and fiscal_year       = :7
  and accounting_period = :8
  and affiliate         = :9
  and statisctics_code  = :10
  and project_id        = :11
  and account           = :12
  and currency_cd       = :13
  and deptid            = :14
  and product           = :15
```

System Peoplesoft, z którego pochodzi powyższy przykład, posiadał indeks zawierający wszystkie kolumny wyszczególnione w wyrażeniu `WHERE`. Wydawać by się mogło, że system Oracle do wykonania zapytania użyje właśnie tego indeksu. Jednak optymalizator kosztowy zdecydował o użyciu indeksu w kolumnach (`business_unit`, `ledger`, `fiscal_year`, `account`). Po odtworzeniu polecenia SQL i porównaniu czasu wykonania z przypadkiem użycia wskazówki nakazującej wykorzystanie większego indeksu okazało się, że jest on ponad czterokrotnie krótszy od czasu wykonania przy użyciu indeksu wybranego przez optymalizator.

Dalsze badania wykazały, że indeks ten powinien być utworzony jako unikatowy (`UNIQUE`), lecz w procesie usuwania danych i odtwarzania

tabeli omyłkowo utworzono go jako nieunikatowy. Oczywiście czterokrotny zysk czasu bardzo ucieszył użytkownika systemu.

Jednak pojawiły się inne problemy. Ten sam indeks był idealnym kandydatem do wykorzystania w znajdującym się poniżej poleceniu, które było jednym z częściej wykonywanych w przypadku przetwarzania danych na końcu miesiąca lub końcu roku:

```
where business_unit     = :5
  and ledger            = :6
  and fiscal_year       = :7
  and accounting_period between 1 and 12
  and affiliate         = :9
  and statisctics_code  = :10
  and project_id        = :11
  and account           = :12
  and currency_cd       = :13
  and deptid            = :14
  and product           = :15
```

Pomimo poprawnego utworzenia indeksu jako unikatowego, optymalizator kosztowy ponownie go nie wziął pod uwagę. Jedyna różnica pomiędzy poleceniem bieżącym a poprzednim polegała na tym, że dotyczyło ono raczej *zakresu* okresów obrachunkowych (*accounting period*) dla roku fiskalnego (*fiscal year*), a nie po prostu jednego okresu obrachunkowego.

Dla powyższego wyrażenia `WHERE` używany był ten sam, co poprzednio, nieodpowiedni indeks z kolumnami (`business_unit`, `ledger`, `fiscal_year`, `account`). I ponownie — po zmierzeniu czasu wykonania polecenia przy użyciu indeksu wybranego przez optymalizator kosztowy oraz indeksu zawierającego wszystkie kolumny — okazało się, że ten drugi zapewniał co najmniej trzykrotnie szybsze wykonanie.

Problem rozwiązano dzięki przestawieniu kolumny `accounting_period` na ostatnią pozycję w indeksie (oryginalnie znajdowała się na trzeciej). Nowy indeks miał następującą postać:

```
business_unit
ledger
fiscal_year
```

```
affiliate
statisctics_code
project_id
account
currency_cd
deptid
product
accounting_period
```

Innym sposobem zmuszenia optymalizatora kosztowego do użycia danego indeksu jest wykorzystanie jednej ze wskazówek, które pozwalają na jego określenie. Jest to dobre rozwiązanie, jednak wiele ośrodków korzysta z pakietów dostarczanych przez twórców oprogramowania, których nie można modyfikować (a w konsekwencji nie można wykorzystać wskazówek). Jednak możliwe jest utworzenie perspektywy zawierającej wskazówkę oraz nadanie użytkownikom uprawnień dostępu do tej perspektywy. Będzie ona przydatna, jeśli polecenie SQL, którego wydajność wykonania pozostawia wiele do życzenia, stanowi część raportu lub zapytania bezpośredniego, które mogą odczytywać perspektywę.

W ostateczności okazuje się czasem, że można wymusić użycie indeksu, jeśli usunie się jego statystyki. Czasem można także użyć polecenia `ANALYZE ESTIMATE` z jedynie podstawową wartością 1064 analizowanych wierszy. Często zdarza się, że plan wykonania zmieniony zostanie na pożądany, jednak ten rodzaj postępowania ma w sobie coś z „czarowania". Niezmiernie istotną sprawą jest to, aby stosując takie „magiczne" działania dokładnie udokumentować wykonane czynności. Jeszcze inna metoda polega na próbie zmniejszenia parametru `OPTIMIZER_INDEX_COST_ADJ`[*] do wartości z przedziału 10 do 50.

Podsumowując trzeba odpowiedzieć na pytanie o to, dlaczego optymalizator kosztowy podejmuje takie nieodpowiednie decyzje. Po pierwsze — należy podkreślić, że zła decyzja dotycząca planu wykonania to raczej wyjątek niż reguła. Przykłady z niniejszego podrozdziału pokazują, że kolumny są rozpatrywane raczej indywidualnie niż grupowo. Gdyby tak było, w pierwszym z prezentowanych przykładów optyma-

---

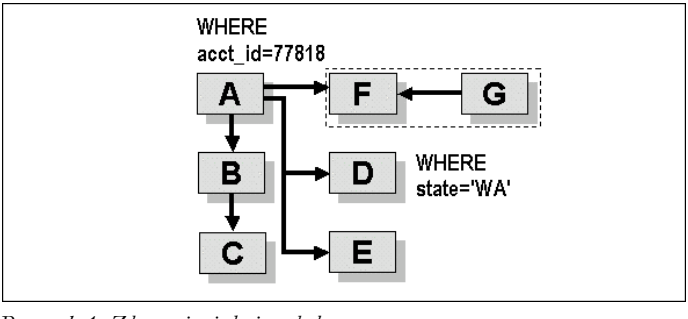
[*] Wartość tego parametru ustawia się w pliku *INIT.ORA — przyp. tłum*

lizator kosztowy stwierdziłby — bez konieczności odtworzenia indeksu przez administratora jako unikatowego — że każdy wiersz posiada unikatowe wartości. Przykład drugi pokazuje, że jeśli kilka kolumn indeksu posiada małą liczbę różnych dopuszczalnych wartości, a polecenie SQL żąda dostępu do większości z nich, to optymalizator kosztowy często pomija taki indeks. Dzieje się tak, mimo że rozpatrywane razem kolumny są ściśle określone i zapytanie zwróci niewiele wierszy.

Nieco usprawiedliwiając działanie optymalizatora należy stwierdzić, że użycie indeksów o mniejszej ilości kolumn często daje znaczny wzrost wydajności wykonywania w porównaniu z użyciem indeksów o wielu kolumnach.

Problem piąty: złączanie zbyt wielu tabel

Pierwsze wersje optymalizatora kosztowego często wykorzystywały metodę „dziel i rządź" w sytuacji, gdy złączaniu podlegało więcej niż pięć tabel. Rozpatrzmy przykład przedstawiony na rysunku 1. Zapytanie wybiera wszystkie dane związane z przedsiębiorstwem o identyfikatorze rachunku (kolumna `acct_id`) równym 777818. Przedsiębiorstwo posiada kilka oddziałów, a zapytanie dotyczy oddziału znajdującego się w stanie Waszyngton (*WA*). Tabela A to tabela `acct`, tabela F to `acct_address`, zaś tabela G to `address`.



Rysunek 1. Złączenie siedmiu tabel

Użytkownik oczekuje, że zapytanie zwróci stosunkowo niedużą liczbę wierszy z różnych tabel, a czas odpowiedzi nie będzie przekraczał 1 sekundy. Najlepiej jest, jeśli system Oracle otrzymuje wiersze z tabeli `acct_address` odpowiadające danemu rachunkowi, a następnie złącza ją z tabelą `address` w celu określenia tego, czy adresy odpowiadają stanowi Waszyngton.

Jednakże ze względu na to, że złączeniu podlega tak wiele tabel, optymalizator kosztowy często decydował będzie o tym, że przetwarzane będą tabele F i G niezależnie od pozostałych i dopiero na końcu dane zostaną scalone. Rezultatem złączenia tabel F i G będzie to, że będą musiały zostać wybrane wszystkie adresy, które dotyczą stanu Waszyngton. Proces ten może zająć nawet kilka minut, co zapewne spowoduje, że ogólny czas wykonania będzie dużo dłuższy od tego, który miałby miejsce, gdyby system Oracle sterował dostępem do wszystkich tabel od tabeli A.

Zakładając, że tabela `acct_address` (F) posiada indeks w kolumnie `acct_id`, można problem ten rozwiązać wykorzystując odpowiednią wskazówkę instruującą optymalizator kosztowy, że użyty powinien zostać ten właśnie indeks. Znacznie zwiększy to wydajność.

Co interesujące — optymalizator regułowy ma często dużo większe problemy z poprawnym określeniem planu wykonania w przypadku złączania wielu tabel niż optymalizator kosztowy. Optymalizator regułowy często w ogóle nie używa tabeli `acct` jako tabeli sterującej. Aby to wymusić, należy w wyrażeniu `FROM` nazwę tabeli A umieścić jako ostatnią.

Jeśli wykorzystywane jest gotowe oprogramowanie, najlepszym sposobem może być utworzenie perspektywy zawierającej wskazówkę (o ile jest to dopuszczalne i możliwe w przypadku używanego pakietu).