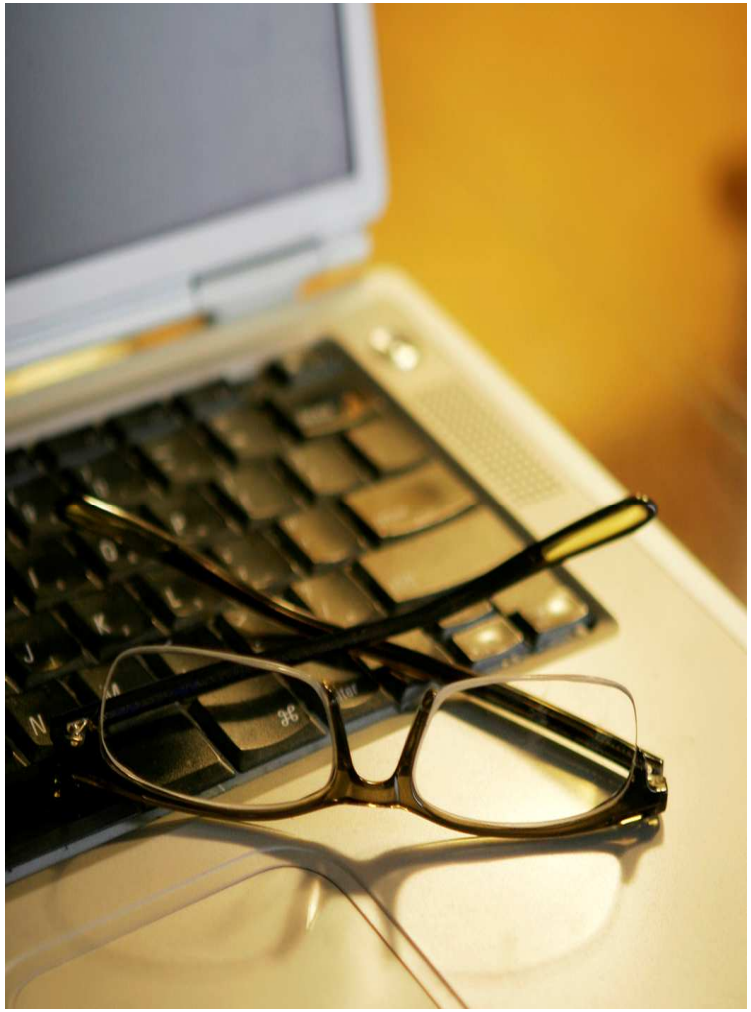
XVI Forum Teleinformatyki

Optymalizacja kodu źródłem redukcji kosztów dotyczących aplikacji bazodanowych wykorzystujących język SQL

Autor: Katarzyna Witan

Analiza wydajności i optymalizacja zapytań SQL



Instrukcje SQL są kluczowymi elementami typowych aplikacji bazodanowych, a więc efektywność ich wykonywania w decydujący sposób wpływa na wydajność samych aplikacji. Twórcy aplikacji bazodanowych i administratorzy baz danych często poświęcają wiele czasu na weryfikację, czy dostęp do danych odbywa się po najszybszych ścieżkach, czyli że **plany wykonywania** zapytań SQL są optymalne. Podstawą weryfikacji powyższego założenie, jest rozważanie wzajemnego związku między strukturą wyrażeń SQL, a planami ich wykonywania.

Charakterystyka optymalizatorów

Istnieją dwa podstawowe typy systemów do optymalizacji zapytań:

- Optymalizatory **regułowe** (korzystają z ustalonych reguł postępowania za pomocą których określają najlepszy plan wykonania np. reguła wymuszająca korzystanie z indeksu, jeżeli został zdefiniowany, niezależnie od rozmiaru tabeli, rodzaju indeksu lub struktury zapytania).
- Optymalizatory **kosztowe** - bardziej korzystne rozwiązanie niż regułowe – (przeprowadzają analizę czasu wykonania, każdego możliwego planu zapytania, w celu uzyskania podstaw do wyliczenia kosztu realizacji wszystkich dopuszczalnych scenariuszy. Optymalnym rozwiązaniem (planem wykonania zapytania) jest plan z najmniejszym kosztem. Podstawą działania jest zastosowanie wskazówek w oparciu o zgromadzone statystyki tabel i indeksów.

Problematyka:

- Optymalizatory poszczególnych środowisk bazodanowych wykazują zróżnicowaną wrażliwość na reguły optymalizacji, powszechnie postrzegane przez programistów jako uniwersalne
- Optymalizatory nie zawsze są w stanie wskazać najlepsze rozwiązanie np. ze względu na niewłaściwe ujęcie danych mających istotny wpływ na realizację zapytania, lub braki w danych uniemożliwiające prawidłową analizę.

Ogólne wskazówki – czego unikać w trakcie pisania kodu? /1

- **Priorytetowe traktowanie najbardziej restrykcyjnych warunków w klauzuli WHERE** – w przypadku warunków połączonych operatorem np. AND testujących stałe wartości, jako pierwszy w kolejności powinien zostać umieszczony najbardziej restrykcyjny warunek, w największym stopniu redukujący liczbę przeszukiwanych wierszy
- **Wymuszenie użycia indeksu poprzez odpowiednie ustawienie kolejności tabel, oraz warunków w klauzuli WHERE** – wymuszenie operacji bazującej na indeksie umożliwia umieszczenie tabeli z najmniejszą liczbą wierszy jako ostatniej w klauzuli FROM, przy jednoczesnym użyciu danej tabeli jako pierwszej w ramach klauzuli WHERE.
- **Poprzedzanie nazw tabeli w części FROM zapytania nazwą właściciela danej tabeli** – takie rozwiązanie pozwala na unikniecie problemów z identyfikacją wykorzystywanych przez zapytanie obiektów.
- **Nie korzystanie z operatorów nierówności: ‘<>’, ‘=!’** – Porównanie tego typu jest niekorzystne, gdyż prowadzi do sekwencyjnego przeglądania tabeli z pominięciem indeksu na kolumnie której dotyczy porównanie.

Ogólne wskazówki – czego unikać w trakcie pisania kodu? /2

- **Nie używanie klauzul IS (NOT) NULL** - Wartości NULL często powodują problemy w obsłudze wektora wyników (np w aplikacjach klienckich), wymagają użycia dodatkowych funkcji do obsługi itp. Najlepiej jest zamieniać puste wartości na konkretny znak lub liczbę i w tej postaci przechowywać je w bazie danych.
- **Zastąpienie domyślnej wartości funkcji COUNT** – wydajniejszym rozwiązaniem w przypadku dążenia do uzyskania licznika jest zastąpienie standardowo stosowanego znaku gwiazdki COUNT(*) wartością liczbową COUNT(1), lub nazwą kolumny posiadającej indeks, gdyż sam indeks może już mieć obliczoną liczbę wierszy.
- **Unikanie symboli wieloznacznych na początku szukanego słowa klauzuli WHERE w wyrażeniu LIKE** - takie użycie znaku wieloznacznego uniemożliwia poprawne wykorzystanie indeksu, nawet jeżeli został on założony na przeszukiwanej kolumnie. Ewentualna zamiana w razie możliwości klauzuli LIKE na BETWEEN.
- **Nie używanie operatora NOT IN** – zdecydowanie wydajniejsza jest klauzula NOT EXISTS.
- **Unikanie zapytań z użyciem DISTINCT** – o wiele wydajniej jest użyć GROUP BY.

Ogólne wskazówki – czego unikać w trakcie pisania kodu? /3

- **Nie zakładanie indeksów na często aktualizowanych kolumnach, oraz na kolumnach które są używane okazjonalnie** – częsta aktualizacja danych wymusza również aktualizację indeksów, co spowalnia proces funkcjonowania bazy danych, natomiast stosowanie indeksów na rzadko używanych kolumnach może powodować nadmierną utratę czasu wynikającego z konieczności obsługi indeksów, które w rzeczywistości nie są potrzebne
- **Unikanie operatora IN** - często zapytanie zawierające IN wykonywane jest za pomocą podzapytania. Zastosowanie takiego rozwiązania poprzedzone jest budową zbioru wynikowego podzapytania w formie tymczasowej tabeli roboczej, która jest następnie przeglądana od lewej do prawej strony. Jest to kosztowne działanie.
- **Unikanie instrukcji UNION** – jest ona implementowana przez konstruowanie dwóch zbiorów wynikowych, a następnie ich łączne sortowanie, w celu ułatwienia usunięcia duplikatów. Jeżeli wystąpienie duplikatów nie ma istotnego wpływu na docelowe wyniki, które powinny zostać otrzymane w efekcie realizacji zapytania, znacznie lepszym rozwiązaniem jest użycie UNION ALL.
- **Zawsze przed wykonaniem zapytania SQL należy przyjrzeć się jego planowi wykonania (EXPLAIN PLAN).**

Plany wykonania dla różnych środowisk bazodanowych – sposób realizacji i interpretacja - Oracle

- Sprawdzenie czy w schemacie bazy danych istniej tabela PLAN_TABLE, przeznaczona do przechowywania danych wynikowych planu wykonania – jeżeli nie utworzenie jej na podstawie skryptu (utlxplan.sql – katalog rdbms/admin) udostępnianego przez Oracle
- Nadanie użytkownikowi uprawnień administracyjnych pozwalających na wykonywanie poleceń związanych z optymalizacją
- Wygenerowanie i wyświetlenie planu wykonania jest czteroetapowym procesem:

  a). usunięcie wszystkich wierszy z tabeli planu wykonania (PLAN_TABLE)

  b). wygenerowanie wyników dla zapytania SQL weryfikowanego pod względem wydajności, za pomocą instrukcji przeznaczonej do tego celu: *EXPLAIN PLAN FOR <optymalizowane zapytanie>;*

  c). wyświetlenie planu wykonania zapytania poprzez realizację następującej instrukcji:

  *SELECT * FROM PLAN_TABLE*

  START WITH ID=0

  CONNECT BY PRIOR ID=PARENT_ID

  ORDER BY ID;

Plany wykonania dla różnych środowisk bazodanowych – sposób realizacji i interpretacja – DB2

- Sprawdzenie czy w schemacie bazy danych istnieje siedem tabel: EXPLAIN_INSTANCE; EXPLAIN_STREAM; EXPLAIN_OBJECT; EXPLAIN_ARGUMENT; EXPLAIN_OPERATOR; EXPLAIN_PREDICATOR; EXPLAIN_STATEMENT, przeznaczonych do przechowywania danych związanych z planem wykonania – jeżeli nie utworzenie ich na podstawie skryptu (explain.ddl– katalog sqlib/misc) udostępnianego przez producenta. Skrypt należy uruchomić po zalogowaniu do schematu w ramach którego tabele będą wykorzystywane.
- Zestawienie połączenia z bazą danych z poziomu katalogu misc
- Zamiana aktualnie używanego schematu na schemat użytkownika, którego konto zostanie użyte w trakcie generowania planu.
- Wygenerowanie i wyświetlenie planu wykonania składa się z czterech etapów:
- a). usunięcie wszystkich wierszy z nadrzędnej w schemacie tabeli EXPLAIN_INSTANCE, a tym samym automatyczne usunięcie powiązanych danych w pozostałych sześciu tabelach przechowujących dane na temat planów wykonania.

Plany wykonania dla różnych środowisk bazodanowych – sposób realizacji i interpretacja – DB2

b). wygenerowanie wyników dla zapytania SQL weryfikowanego pod względem wydajności, za pomocą instrukcji:

EXPLAIN PLAN FOR <optymalizowane zapytanie>;

c). wyświetlenie planu wykonania zapytania przy użyciu jednego z narzędzi:

- Visual Explain – narzędzie graficznej analizy – wymaga zainstalowania oprogramowania klienckiego na stacji roboczej i nie jest dostępne dla wszystkich platform systemowych;
- Narzędzie db2exfm – program uruchamiany z wiersza poleceń w dowolnym środowisku systemowym – generuje dużą ilość nadmiarowych danych, zbędnych przy optymalizacji, tym samym utrudnia szybką analizę danych wynikowych.
- Manualne wprowadzenie zapytania i analiza tabeli danych planu – dostarcza niezbędne do optymalizacji dane w zwięzłej formie, umożliwiającej ustalenie kolejności złączeń, oraz metod złączeń, a także metod dostępu do tabel

Efektywny plan wykonania zapytania

Celem optymalizacji jest osiągnięcie efektywnego planu wykonania.

Jako **efektywny plan wykonania** postrzegany jest prosty plan wykonania zapytania składający się ze złączeń pętli zagnieżdżonych wykonywanych w poprawnej kolejności.

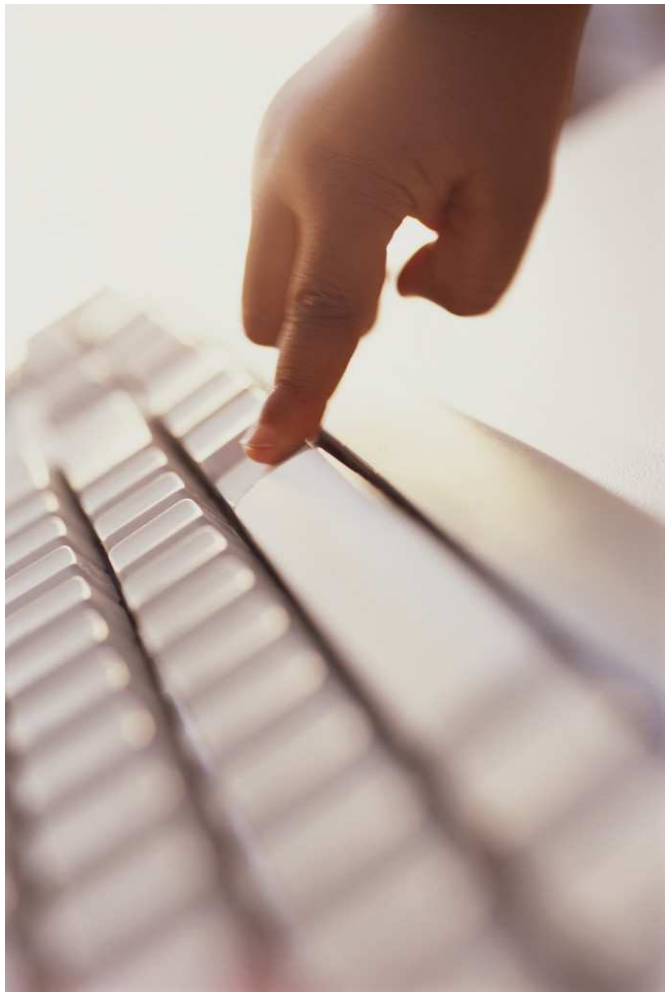
Główne cechy efektywnego planu wykonania są następujące:

- Koszt realizacji planu jest proporcjonalny do liczby zwracanych wierszy
- Nie jest wymagana szczególnie duża przestrzeń pamięci, typowa dla funkcji mieszających i sortujących
- Plan nie musi być modyfikowany ze względu na zwiększające się rozmiary tabel
- Działanie planu jest w dużym stopniu niezależne od rozkładu danych
- Szczególna efektywność planu uwidacznia się w sytuacji, gdy zapytanie zwraca mniej wierszy, niż było zakładane (filtry są bardziej selektywne niż przypuszczano).

Czynniki wpływające na efektywność planu wykonania

Przy optymalizacji zapytań na podstawie analizy przeprowadzonej w oparciu o wyniki planu wykonania, należy wziąć pod uwagę następujące czynniki:

- **Metody dostępu do tabel**
  - użycie indeksu
  - pełne przeglądanie tabel
- **Kolejność złączeń**
- **Metody złączeń**
  - pętle zagnieżdżone
  - złączenia mieszające
  - złączenia sortująco scalające



Podsumowanie

- Najważniejszym czynnikiem wpływającym na wydajność aplikacji jest szybkość wykonywania instrukcji SQL
- Optymalizacja zapytań SQL umożliwia przekształcenie powolnych instrukcji w ich szybko działające odpowiedniki bez konieczności ogólnego modyfikowania sposobu pracy systemu, oraz bez dokonywania zmian w aplikacjach odwołujących się do bazy danych. Wspomniane rozwiązanie jest szczególnie użyteczne ze względu, na fakt, że nie jest bardzo skomplikowane i zazwyczaj nie wiążą się z nim negatywne skutki uboczne.
- Poprawnie skonstruowany kod stanowi dobrą podstawę dla prawidłowego rozwoju aplikacji
- Optymalizacja zapytań SQL jest rozwiązaniem długofalowym i korzystniejszym niż doraźne rozwiązywanie problemów związanych z wydajnością aplikacji, poprzez zakup nowych podzespołów, gdyż nie wymaga żadnych nakładów finansowych

DZIĘKUJĘ ZA UWAGĘ