

VI Seminarium PLOUG
Warszawa
Styczeń 2003

Czego mądrzy ludzie z Oracle'm nie robią

Piotr Kołodziej

e-mail: Piotr.Kolodziej@oracle.com

Oracle Polska

*Z systemu zakazów można wywnioskować co ludzie zwykle czynią.
Umberto Eco, Wahadło Faucaulta
Kto się nie uczy na błędach ten je popełnia
Zasłyszane*

Mając głowę w chmurach warto się pochylać by dojrzeć kamyki tudzież skórki od bananów. Choć przyznam, że trochę głupio w dobie Javy i XML mówić niczym zrzęda o rzeczach zdawałoby się, przebrzmiałych.

Nie warto długo pisać o dobieraniu parametrów składowania tabel i indeksów – wszyscy doskonale dobieramy INITRANS i PCTFREE dzięki czemu unikamy wielkiej szansy sprowadzenia Oracle do poziomu RDBMS blokującego na poziomie bloków zamiast wierszy, a wydajność masowego ładowania danych nie wpada w pułapkę złożoności kwadratowej. Zresztą w bazie Oracle 9i wprowadzono automatyczne zarządzanie przestrzenią wewnątrz segmentu (*segment space management*), dzięki czemu za jakiś czas takie dywagacje z całym spokojem będą mogły przejść na zasłużoną emeryturę.

Korzystania z typu VARCHAR2 zamiast CHAR i umieszczania kolumn NOT NULL jako pierwszych w kolejności a następnie kolumn dopuszczających wartości NULL nauczyliśmy się jeszcze w czasach, gdy każdy megabajt przestrzeni dyskowej był na wagę złota. Dużej pokory i znajomości tajników związanych z typem CHAR można było ponadto nabyć przy okazji wykonywania instrukcji ALTER TABLE *tabela* MODIFY *kolumna* CHAR(80), rozszerzające kolumnę w tabeli wypełnianej danymi, co poniekąd przypominało lekcje związane ze spóźnionym dodawaniem więzów integralności. I nieomal każdy, kto porównywał zmienne czy pola typu CHAR ze zmiennymi / polami typu VARCHAR2 mógł doświadczyć prawdy, że bywają sytuacje, w których porównywanie łatwo sprowadza na manowce.

Prawidłowo indeksujemy, uwolniliśmy się od natarczywej myśli, by wykorzystać indeks do posortowania kilkudziesięciu wierszy a także by przeglądać duże tabele za pomocą dostępu indeksowanego. Tak jak wcześniej tropiliśmy w planach zapytań dostępu FULL SCAN, tak teraz czujemy, że sekwencja przetwarzania NESTED LOOPS – TABLE ACCESS (BY ROWID) – INDEX (RANGE SCAN) może posiadać odmienne oblicza.

Skoro wstęp zdał się zająć trochę miejsca i uwagi, skoro tyle wiedzy już się zdażyło upowszechnić, warto przejść do zagadnień, którym zostanie poświęcona większa uwaga.

Nadmiar komunikacji międzyprocesowej

Mowa jest srebrem, milczenie jest złotem

Bywają sytuacje, w których kluczem do osiągnięcia rozsądnego czasu przetwarzania jest zminimalizowanie komunikacji międzyprocesowej, zwłaszcza takiej, która wymaga odwołania się do funkcji realizowanych w trybie jądra systemu operacyjnego. W przypadku IPC realizowanego w trybie jądra, opóźnienie (*latency*) związane z komunikacją międzyprocesową trwa z reguły kilka rzędów dłużej, niż narzut na wywołanie funkcji lokalnej procesu, bądź z załadowanej biblioteki dzielonej.

Przypadki, które warto polecić uwadze obejmują m.in.

- Komunikację pomiędzy serwerem aplikacyjnym i serwerem bazy danych (bądź przypadek klient – serwer);
- Komunikację pomiędzy procesem oracle a procedurami zewnętrznymi (za pośrednictwem procesu extproc).

Biorąc pod uwagę opóźnienia związane z przełączaniem kontekstu, szeregowaniem zadań przez system operacyjny, opóźnienia sieciowe i stosu TCP/IP można przyjąć następujące ograniczenia komunikacji międzyprocesowej w najbardziej typowych przypadkach:

- Do kilku tysięcy przełączeń na sekundę w przypadku lokalnego IPC
- Do około tysiąca przełączeń na sekundę w przypadku komunikacji sieciowej za pośrednictwem LAN bardzo dobrej jakości
- Do kilku przełączeń na sekundę w przypadku komunikacji sieciowej za pośrednictwem WAN o czasie opóźnienia powyżej 100ms (czyli czas transmisji tam i z powrotem – *roundtrip time* – powyżej 200ms).

Jakie z tego płyną wnioski? Popatrzmy na przykłady.

SELECT i funkcje zewnętrzne

Czyż nie jest znana pokusa, by skorzystać z zastosowania procedury (funkcji) zewnętrznej do w SELECT-cie? I piszący te słowa tej pokusie ulegał. I przekonać się można doświadczalnie, że gdy funkcja zewnętrzna jest przeliczana na wielkiej liczbie rekordów (np. przy niezbyt szczęśliwej filtracji na podstawie innych kryteriów z klauzuli WHERE), maksymalna wydajność przetwarzania zbioru danych jest wówczas ograniczona do najwyższej kilku tysięcy rekordów na sekundę.

Maksymalne przetwarzanie danych po stronie serwera bazy danych

Dawno, dawno temu, kiedy procesory miały kilkunasto-, no może kilkudziesięciomegahercowe zegary, a 9600 było prędkością transmisji w bitach na sekundę nawet nie modemu a terminala podłączonego do koncentratora, na pierwszych stronach *Oracle Concepts Manual* przedstawiana była pewna ilustracja. Ukazywała ona, że zamiast przesyłania do bazy wielu instrukcji SQL, odbierania i przetwarzania wyników można wysłać i wykonać na serwerze blok PL/SQL z zanurzonymi instrukcjami SQL. Warto ponadto zauważyć, że były to już czasy silnego zakorzenienia architektury *two-task*, w której przestrzeń adresowa serwera bazy Oracle była oddzielona od przestrzeni adresowej narzędzi, np. SQL*Forms/Menu.

Doświadczenia wskazują, że w przypadku stosunkowo “lekkich” instrukcji SQL (np. pobranie bądź modyfikacja wiersza po kluczu pierwotnym bądź unikalnym, wstawienie pojedynczego wiersza) grupowanie instrukcji SQL w bloki PL/SQL odpowiadające np. transakcjom pozwala na znaczące zwiększenie prędkości przetwarzania – w niektórych przypadkach nawet kilkukrotne. Warto przy okazji wspomnieć, że w testach TPC-C przeprowadzanych na bazie Oracle transakcje są implementowane w postaci bloków PL/SQL¹.

Przemawiająca do wyobraźni ilustracja o której była wyżej mowa nie stanęła jednak na przeszkodzie innemu zjawisku, które jest bardzo charakterystyczne dla aplikacji napisanych

¹ Dobrym źródłem informacji jest strona <http://www.tpc.org/> na której są opublikowane pełne raporty z przeprowadzonych testów TPC, zawierające m.in. opisy konfiguracji, model danych i kod źródłowym aplikacji TPC.

w kolejnych wcieleniach Oracle Forms. Tym zjawiskiem jest częsty nadmiar wyzwalaczy Oracle Forms odwołujących się do bazy. Prosty przykład? Niech będzie sobie blok formatki, w bloku pola i wyzwalacz POST-QUERY, na przynajmniej kilku polach wyzwalacze POST-CHANGE, każdy z ww. wyzwalaczy wykonuje przynajmniej jedną instrukcję SELECT a zapytanie odświetla dziesiątki rekordów. W konfiguracjach opartych o LAN z bardzo niskimi czasami opóźnień takie konstrukcje mogły uchodzić płazem (zamęczając nieco ponad miarę serwer bazy bądź aplikacji). Jednak w sieciach rozległych oczekiwanie na odświetlenie zawartości bloków mogło się stawać nieakceptowalne. Co gorsza na nic się zda w omawianym przypadku ustawienie array fetch dla bloku i parametrów transmisji SDU/TDU.

Shared pool – zarządzanie to wyzwanie

Z czym jak z czym ale z obszarem *shared pool* należy się obchodzić bardzo delikatnie. Po pierwsze, *shared pool* bardzo nie lubi zbyt częstych i zbyt daleko idących zmian w jego zawartości. Alokacja i dealokacja jego obszarów jest zabezpieczana przez jedną jedyną ścieżkę krytyczną (*latch*), zaś jego zawartości korzystają wszystkie sesje niemal na każdym kroku, gdyż stanowi pamięć podręczną wykorzystywaną na wiele sposobów: *data dictionary*, przetwarzanie SQL i PL/SQL, autoryzacje, tworzenie i usuwanie sesji itd. Jeśli chcemy doprowadzić serwer bazy danych do krótszego czy dłuższego zawieszenia, najłatwiej tego dokonać przez wprowadzenie zamieszania w ww. obszarze.

Słów parę o parametryzacji zapytań przez zmienne wiązane

Obszar *shared pool* najłatwiej zapełnić przy pomocy rodziny zapytań, która ma niemal identyczną postać, lecz różni się wartościami parametrów zapytań, wpisanych bezpośrednio jako literały, np.

```
select ename, job from scott.emp where empno=10
select ename, job from scott.emp where empno=11
...
select ename, job from scott.emp where empno=101
```

Programując w C, Javie i innych językach, czy też korzystając z dynamicznego SQL w PL/SQL łatwiej jest konkatenować wartości parametrów z tekstem zapytania niż wstawiać do tekstu SQL znaczniki zmiennych wiązanych (*placeholders*) i następnie je wiązać ze zmiennymi programu. Przy odpowiednich warunkach – tj. częstym wykonywaniu zapytań, dużej zmienności parametrów (zwłaszcza systemy o charakterystyce OLTP) – może to doprowadzić do znacznego obciążenia obszaru *shared pool* przez ciągle dealokowanie i alokowanie pamięci dla coraz to nowych instrukcji SQL. Zwiększenie obszaru *shared pool* w takich przypadkach może jedynie opóźnić moment wystąpienia spowolnień w pracy, może za to spowodować zwiększenie ich dotkliwości.

Dlaczego ten temat został poruszony, gdy możliwe jest automatyczne wymuszenie współdzielenia kursorów przez proste ustawienie parametru instancji CURSOR_SHARING na SIMILAR bądź FORCE? Zastosowanie literałów bywa niezbędne do prawidłowej optymalizacji zapytań. I nie powinno się z niego rezygnować, gdy literały nie doprowadzają do wyraźnego różnicowania tekstów zapytań. Wymuszenie współdzielenia kursorów warto potraktować jako naprawdę ostatnią deskę ratunku.

O mnożeniu bytów

Obok jednolitego tekstu SQL drugim warunkiem współdzielenia tzw. kursora w obszarze *shared pool* jest odwoływanie się do identycznych obiektów. Oznacza to, że zapytanie `SELECT * FROM EMP` może być współdzielone, gdy `EMP` jest tym samym obiektem, do którego odwołują się użytkownicy realizujący wspomiane zapytanie. Dla ustalenia uwagi – synonim `A.EMP` wskazujący na tabelę `SCOTT.EMP` jest innym obiektem, niż `SCOTT.EMP`. Tak samo synonimy `A.EMP` i `B.EMP` są innymi obiektami pomimo, że wskazują na tabelę `SCOTT.EMP`. Wniosek? Użytkownicy A i B, choć wydają takie samo zapytanie `SELECT * FROM EMP`, choć faktycznie czytają tę samą tabelę `SCOTT.EMP` nie mogą współdzielić kursora. Takie zjawisko w skali masowej prowadzi do zwiększenia obciążenia obszaru *shared pool* proporcjonalnego do liczby użytkowników i prawdziwych kłopotów z prawdziwym skalowaniem.

Warto zatem unikać stosowania prywatnych synonimów w celu ominięcia kwalifikowania nazwy obiektu nazwą schematu. Jeśli nie można z jakichś względów dokonać takiego kwalifikowania, warto rozważyć przełączenie przestrzeni nazw użytkownika na inny schemat przez `ALTER SESSION SET CURRENT_SCHEMA` przy wejściu do aplikacji. Inną, bardziej kłopotliwą i wprowadzającą pewien narzut możliwością jest zastosowanie publicznych synonimów. Wspomniany narzut jest związany z utrzymywaniem w *library cache* negatywnych zależności dla każdego obiektu, do którego odwołują się użytkownicy. Dla każdego obiektu liczba kopii opisu negatywnej zależności jest równa liczbie użytkowników, którzy do niego się odwołują za pośrednictwem ww. synonimu.

O roli ról w bazie danych

Czy sposób przyznawania użytkownikom praw dostępu do obiektów bazy danych może mieć wpływ na wydajność? Okazuje się, że może. Od czasów Oracle 7 zalecaną metodą jest przydzielanie przywilejów do ról bazodanowych a następnie nadawanie użytkownikom odpowiednich ról. Oprócz uproszczenia pracy administratora (Próbował kto tylko nadać paru setkom użytkowników prawa do około tysiąca obiektów, o utrzymaniu spójności wstydliwie milcząc? Ile czasu to zajmie, nawet przy automatyzacji przez skrypt?) pozwala to na minimalizację obciążenia obszaru *dictionary cache* i uproszczenie autoryzowania praw dostępu do obiektów bazy danych w fazie parsowania zapytania.

Czy warto się czasem przeprosić z anonimowymi blokami PL/SQL?

Czy hierarchia zależności pomiędzy obiektami PL/SQL składowanymi w bazie jest do ogarnięcia? Czy ktoś nad nią panuje? Przypuszczać można, że takie pytania padały niejednokrotnie i w niejednym miejscu. Bywały one z pewnością na czasie, gdy zmiany strukturalne bądź kompilacje PL/SQL nawzajem się blokowały, czy też blokowały pracę użytkowników.

Jeżeli procedura PL/SQL składowana w bazie istnieje tylko po to, by wywołać ją z jedyne go modułu w aplikacji warto rozważyć rezygnację ze składowania procedury PL/SQL w bazie i zastąpienie jej przez wywoływanie anonimowego bloku PL/SQL. Może to pozwolić na pewne uproszczenie hierarchii zależności między obiektami PL/SQL.

Natywna kompilacja PL/SQL

Wraz z wprowadzeniem do Oracle 9i tzw. natywnej kompilacji PL/SQL mówi się o głównie o szybszym wykonywaniu kodu maszynowego od interpretowania p-kodu. Jest to niewątpliwa prawda, choć nie jest to cała prawda. Drugą zasadniczą korzyścią jest fakt, że kod PL/SQL

skompilowany natywnie nie jest ładowany do obszaru *shared pool*, lecz dołączany jako biblioteka dzielona do przestrzeni adresowej procesu Oracle. W przypadku systemów opartych w dużej mierze o PL/SQL składowany w bazie stanowi to szansę na znaczące zredukowanie obciążenia newralgicznego punktu serwera bazy danych Oracle do kwestii związanych z zarządzaniem *dictionary cache* (głównie weryfikacji praw dostępu).

Skoro była mowa o zależnościach pomiędzy modułami PL/SQL składowanymi w bazie: ze względów wydajnościowych warto unikać wzajemnego wywoływania przez moduły PL/SQL interpretowane i skompilowane natywnie.

W miejsce zakończenia

Na koniec warto zaznaczyć, że z czasem pewne prawdy mogą się stać półprawdami. Temu zjawisku mogą kiedyś podlegać np. kwestie związane z mechanizmami komunikacji międzyprocesowej (IPC). Obecną tendencją jest konstruowanie mechanizmów IPC, które korzystają z bardzo szybkich mechanizmów komunikacji (*memory channel*, interfejsy SCI itd) oraz wykonywane są w trybie użytkownika, dzięki czemu opóźnienie komunikacyjne może być zredukowane nawet do rzędu pojedynczych mikrosekund (maszyny HP Alpha, interfejs *memory channel*, protokół RDG). Bieżącym zastosowaniem ww. mechanizmów IPC jest skalowanie Oracle Real Application Clusters.

A co do nowinek... Umiar, zdrowy rozsądek i wczytanie się w dokumentację nigdy nie zaszkodzą. Nie zmieniamy dynamicznie rozmiaru *shared pool*-a, gdy w bazie pracują setki użytkowników. Prostszy może być restart instancji.