Oracle Database 11g. Podręcznik administratora baz danych

Autorzy: Bob Bryla, Kevin Loney
Tłumaczenie: Piotr Pilch
ISBN: 978-83-246-2547-5
Tytuł oryginału: Oracle Database 11g DBA Handbook (Osborne Oracle Press)
Format: 168×237, stron: 776

Poznaj możliwości systemu Oracle Database 11g i profesjonalnie administruj bazami danych

- Jak tworzyć bogate w możliwości aplikacje, zarządzające bazami danych?
- Na czym polega implementowanie solidnych zabezpieczeń z wykorzystaniem uwierzytelnienia i kontroli dostępu?
- W jaki sposób pracować z hurtowniami danych oraz sieciowymi i bardzo dużymi bazami danych?

System Oracle 11g kontynuuje tradycję rozszerzania w kolejnych edycjach możliwości oraz funkcji baz danych Oracle i tym samym dostarcza wymiernych korzyści pracy administratora. Tym razem udoskonalono w nim automatyczne zarządzanie pamięcią, a ponadto zaproponowano nowe narzędzia wspomagające oraz usprawnienia w zakresie dostępności i przejmowania funkcji uszkodzonej bazy. Dzięki takim – często rewolucyjnym – aktualizacjom baza danych Oracle znajduje zastosowanie we wszystkich sytuacjach, w których liczy się bezwzględna stabilność systemu, absolutne bezpieczeństwo danych i szybkość działania. Każdy administrator baz danych czy programista aplikacji, który chce efektywnie wykonywać swoją pracę, powinien poznać nowe funkcje oferowane przez Oracle. Książka „Oracle Database 11g. Podręcznik administratora baz danych" zawiera wszystkie niezbędne, w pełni aktualne informacje, których potrzebujesz, aby sprawnie zarządzać bazą danych Oracle. Dzięki temu fachowemu przewodnikowi dowiesz się, jak skonfigurować sprzęt oraz oprogramowanie pod kątem maksymalnej efektywności i w jaki sposób stosować niezawodne zabezpieczenia. Poznasz prawidłowe strategie monitorowania, kontrolowania i strojenia zarówno samodzielnych, jak i sieciowych baz danych. Korzystając z tego podręcznika, nauczysz się automatyzować proces przywracania i tworzenia kopii zapasowych, zapewniać transparentne możliwości przełączania po awarii oraz dystrybuować bazy danych przedsiębiorstwa z wykorzystaniem środowiska Oracle Net.

- Architektura systemu Oracle
- Uaktualnianie bazy danych do wersji Oracle 11g
- Planowanie przestrzeni tabel i zarządzanie nimi
- Zarządzanie bazą danych
- Projektowanie i implementowanie aplikacji
- Monitorowanie użycia przestrzeni dyskowej
- Bezpieczeństwo baz danych
- Zarządzanie profilami i metody autoryzacji
- Architektura narzędzia Data Guard
- Funkcje zapewniające wysoką dostępność
- Rozproszenie bazy danych

Sprawnie i profesjonalnie zarządzaj wielkimi bazami danych!

Spis treści

Rozdział 8.

Strojenie bazy danych

Patrząc z perspektywy strojenia, każdy system zawiera wydajnościowe wąskie gardło, które w przeciągu dni, a nawet tygodni może wystąpić w różnych składnikach. Celem projektowania pod kątem wydajności jest zapewnienie, że fizyczne ograniczenia aplikacji i powiązanego z nimi sprzętu, takie jak przepustowość operacji wejścia-wyjścia, rozmiary pamięci, wydajność zapytań itd., nie wpłyną na wydajność biznesową systemu. Jeżeli wydajność aplikacji ogranicza procesy biznesowe, które mają być przez tę aplikację obsługiwane, wówczas należy ją dostroić. W trakcie projektowania systemu zawsze trzeba szacować ograniczenia środowiska, w którym aplikacja będzie funkcjonować, biorąc pod uwagę między innymi specyfikę używanego sprzętu oraz sposób komunikowania się aplikacji z bazą danych. Nie istnieje środowisko, w którym można zapewnić nieskończone moce obliczeniowe, dlatego w którymś momencie wydajność każdego systemu się załamie. W trakcie projektowania aplikacji należy dążyć do tego, by wymagana wydajność rozwiązania została zapewniona dzięki odpowiednim parametrom wydajnościowym środowiska.

Strojenie wydajności jest częścią cyklu życia każdej aplikacji bazodanowej, a im wcześniej twórcy aplikacji pochylą się nad wydajnością (dobrze, aby miało to miejsce jeszcze przed wdrożeniem w środowisku produkcyjnym), tym większa jest szansa, że nie będzie ona stanowić problemu w przyszłości. Jak wspomniano już w poprzednich rozdziałach, większość problemów z wydajnością nie przejawia się w postaci niezależnych od siebie symptomów, lecz zwykle wynika ze sposobu zaprojektowania systemu. W trakcie strojenia aplikacji należy się zatem skupić na zidentyfikowaniu i naprawieniu błędów w konstrukcji aplikacji, które stanowią źródło zbyt niskiej wydajności.

Strojenie to ostatni krok w procesie złożonym z czterech etapów. Poprzedzają go planowanie, implementacja i monitorowanie. Jeśli strojenie jest wykonywane jedynie po to, by pozostać w zgodzie ze sztuką, oznacza to przerwanie cyklu życia aplikacji i może się okazać, że usunięcie błędów w jej konstrukcji, będących przyczyną problemów z wydajnością, nie będzie możliwe.

Większość obiektów bazy danych, które można stroić, jest opisana w różnych częściach niniejszej książki; na przykład segmenty wycofania opisano ze szczegółami w rozdziale 7. W tym rozdziale skupimy się jedynie na strojeniu tego rodzaju obiektów, natomiast czynności związane z planowaniem ich i monitorowaniem są opisane w oddzielnych rozdziałach.

Począwszy od systemu Oracle 10*g*, a w systemie Oracle 11*g* w znacząco rozszerzonym zakresie, można korzystać z nowych narzędzi i funkcji strojenia, do których należy między innymi Automated Workload Repository. Oracle zaleca regularne używanie OEM Database Control ze względu na łatwość obsługi i zastosowanie kilku zautomatyzowanych narzędzi monitorujących i diagnozujących. Jednak zanim zajmiemy się narzędziami OEM, zostanie przedstawionych kilka wstępnych wymogów i zasad związanych ze skutecznymi, aktywnymi i reaktywnymi metodami strojenia.

W kolejnych punktach opisane zostaną sposoby strojenia bazy danych w takich obszarach, jak:

- konstrukcja aplikacji,
- polecenia SQL,
- użycie pamięci,
- przechowywanie danych,
- manipulowanie danymi,
- fizyczne przechowywanie danych,
- logiczne przechowywanie danych,
- ruch w sieci.

Strojenie konstrukcji aplikacji

Dlaczego każdy przeznaczony dla administratora baz danych przewodnik po strojeniu bazy powinien zawierać rozdział poświęcony strojeniu konstrukcji aplikacji? I dlaczego powinien to być pierwszy rozdział? Ponieważ żadne inne działanie wykonywane przez administratora bazy danych nie ma takiego wpływu na wydajność systemu, jak projektowanie aplikacji. Sposoby angażowania administratora bazy danych w proces projektowania aplikacji opisano w rozdziale 5. W trakcie projektowania aplikacji można wykonać szereg czynności przyczyniających się do efektywnego i prawidłowego wykorzystania dostępnych technologii. Czynności te opisano w następnych punktach.

Efektywne struktury tabel

Bez względu na to, w jaki sposób zaprojektuje się bazę danych, złe zaprojektowanie tabel będzie zawsze przyczyną niskiej wydajności. Co więcej, przyczyną niezadowalającej wydajności może stać się również bezkrytyczne stosowanie zasad projektowania tabel relacyjnych. Choć ścisłe stosowanie zasad relacyjnych w trakcie projektowania tabel (czyli doprowadzanie do **trzeciej**, a nawet **czwartej postaci normalnej**) jest pożądane pod względem logicznym, to z punktu widzenia fizycznej konstrukcji tabel zwykle jest to rozwiązanie niepożądane (wyjątkiem są środowiska OLTP).

Problemem w konstrukcjach relacyjnych jest to, że choć bardzo dobrze odzwierciedlają one relacje wiążące dane, to nie odzwierciedlają ścieżek dostępu, przy których użyciu użytkownicy będą te dane odczytywać. Gdy zidentyfikowane zostaną wymagania użytkownika, zwykle okazuje się, że w pełni relacyjny model tabel jest zupełnie bezużyteczny dla wielu rozbudowanych zapytań. Pierwsze kłopoty zazwyczaj dotyczą zapytań, które zwracają znaczną liczbę kolumn. Kolumny te są zwykle rozrzucone w wielu tabelach, a więc w zapytaniu konieczne jest łączenie tabel ze sobą. Jeśli jedna z łączonych tabel jest duża, wówczas ucierpieć może na tym wydajność całego zapytania.

W trakcie projektowania struktury tabel dla aplikacji projektanci powinni najpierw opracować model w trzeciej postaci normalnej, a następnie denormalizować dane w taki sposób, by spełnić konkretne wymagania — na przykład przez tworzenie małych tabel podsumowujących (lub widoków materializowanych) dla dużych tabel statycznych. Czy dane podsumowujące można dynamicznie pozyskiwać z dużych tabel statycznych na żądanie? Oczywiście, że można. Jeśli jednak użytkownik będzie pytał o nie bardzo często, a dane źródłowe zmieniają się rzadko, sensowniejszym rozwiązaniem będzie regularne zapisywanie wymaganych danych **w formacie, jakiego oczekuje użytkownik**.

Niektóre aplikacje przechowują na przykład w tej samej tabeli dane bieżące oraz dane historyczne. Każdy rekord może posiadać kolumnę ze znacznikiem czasu, a więc bieżącym rekordem w zbiorze będzie rekord z najmłodszym znacznikiem czasu. Za każdym razem, gdy użytkownik wykonuje na tabeli zapytanie o rekord bieżący, konieczne jest wykonanie podzapytania na przykład o następującej postaci:

```
where timestamp_col =
   (select max(timestamp_col)
      from table
     where emp_no=196811)
```

Jeśli konieczne będzie połączenie takich dwóch tabel, do wykonania będą dwa podzapytania. W małej bazie danych fakt ten może nie wpływać na wydajność, lecz w miarę wzrostu liczby tabel i wierszy zwiększa się prawdopodobieństwo wystąpienia problemów wydajnościowych. Oddzielenie danych historycznych od danych bieżących albo przechowywanie danych historycznych w odrębnej tabeli zwiększy nieco zakres pracy administratorom bazy danych i projektantom aplikacji, lecz w dłuższej perspektywie powinno poprawić wydajność rozwiązania.

Projektowanie struktury tabel z myślą o wymaganiach użytkownika, a nie z myślą o zgodności z teorią relacyjną spowoduje, że opracowywany system lepiej będzie spełniał wymagania użytkowników. Nie oznacza to, że nie powinno się **projektować** bazy danych z wykorzystaniem metodologii 3NF (trzecia postać normalna) i 4NF (czwarta postać normalna). Jest to dobry punkt wyjściowy w przypadku określania wymagań biznesowych i warunek wstępny projektu fizycznej bazy danych. Wśród dostępnych opcji związanych z projektem fizycznej bazy danych znajduje się między innymi możliwość podzielenia pojedynczej tabeli na kilka mniejszych tabel i odwrotnie: możliwość łączenia wielu tabel w jedną. Główny nacisk powinno się kłaść na udostępnienie użytkownikowi najbardziej bezpośredniej możliwej ścieżki dostępu do potrzebnych danych w wymaganym formacie.

Rozkładanie wymagań względem procesorów

Właściwie zaprojektowana i działająca na odpowiednim sprzęcie aplikacja z bazą danych Oracle będzie przetwarzać żądania wejścia-wyjścia bez nadmiernych przestojów, obszary pamięci będą używane bez wymiany i stronicowania pamięci na dysku, a średnie obciążenie procesora nie będzie zbyt wysokie. Dane wczytane do pamięci przez jeden proces pozostaną w niej i będą dostępne również dla wielu innych procesów, dopóki ich ważność nie wygaśnie. Polecenia SQL będą wielokrotnie wykorzystywane dzięki wspólnemu obszarowi SQL, co jeszcze bardziej zmniejszy obciążenie systemu.

Ograniczenie dozwolonego obciążenia systemu operacjami wejścia-wyjścia może spowodować wzrost obciążenia procesora. Zasobami procesora można zarządzać na kilka sposobów:

- Obciążenie procesora należy zaplanować w czasie. Czasochłonne zapytania wsadowe powinny być wykonywane po godzinach najintensywniejszej pracy systemu. Zamiast wykonywać tego typu zapytania z niższym priorytetem systemu operacyjnego i w trakcie, gdy aplikacja jest używana przez użytkowników, lepiej jest wykonywać je z normalnym priorytetem systemu operacyjnego o odpowiedniej porze. Utrzymanie normalnego priorytetu systemu operacyjnego i odpowiednie rozplanowanie zadań w czasie pozwoli zminimalizować potencjalne konflikty w zakresie nakładania blokad, wycofywania operacji i użycia zasobów procesora.
- Należy korzystać z możliwości fizycznego przenoszenia wymagań względem procesora z jednego serwera na inny. Zawsze, gdy jest to możliwe, należy oddzielać serwer bazy danych od wymagań aplikacji względem procesora. Techniki dystrybucji danych, które zostaną przedstawione w rozdziałach opisujących zagadnienia sieciowe, pozwolą na przechowywanie danych w najodpowiedniejszych dla nich miejscach, a wymagania aplikacji względem procesora będzie można odseparować od wymagań względem operacji wejścia-wyjścia na bazie danych.
- Należy rozważyć możliwość zastosowania technologii Real Application Clusters (RAC), aby rozproszyć wymagania dotyczące dostępu do bazy danych na wiele instancji. W rozdziale 10. dokonano obszernego przeglądu funkcji RAC, a także krok po kroku wyjaśniono, jak utworzyć bazę danych RAC.
- Warto korzystać z funkcji zarządzania zasobami bazy danych. Za pomocą narzędzia Database Resource Manager można opracować plany alokacji zasobów oraz grupy użytkowników zasobów. Można korzystać również z dostępnych w bazie Oracle możliwości zmian alokacji zasobów do grup użytkowników. Więcej szczegółowych informacji na temat tworzenia i implementowania grup użytkowników zasobów oraz planów zasobów za pomocą narzędzia Database Resource Manager znajduje się w rozdziale 5.
- Za pomocą narzędzia Parallel Query można rozkładać wymagania co do przetwarzania instrukcji języka SQL na wiele procesorów. Możliwe jest zrównoleglenie niemal każdego polecenia SQL, w tym poleceń `select`, `create table as select`, `create index` i `recover`, a także opcji ładowania narzędzia SQL*Loader Direct Path.

Zakres zrównoleglenia transakcji zależy od zdefiniowanego stopnia zrównoleglenia dla transakcji. Dla każdej tabeli istnieje zdefiniowany stopień zrównoleglenia, natomiast zapytanie może tak zdefiniowany domyślny stopień zrównoleglenia nadpisać własnym stopniem

zdefiniowanym przy użyciu wskazówki PARALLEL. Oracle sprawdza liczbę procesorów dostępnych na serwerze oraz liczbę dysków, na których przechowywane są dane tabeli, i na tej podstawie ustala domyślny stopień zrównoleglenia.

Maksymalny dostępny stopień zrównoleglenia jest definiowany na poziomie instancji. Parametr inicjalizacyjny PARALLEL_MAX_SERVERS wskazuje maksymalną liczbę równoległych procesów zapytań serwera, używanych w tym samym momencie przez wszystkie procesy w bazie danych. Jeśli na przykład parametrowi PARALLEL_MAX_SERVERS dla instancji przypisana zostanie liczba 32 i uruchomione zostanie zapytanie, które używa 30 równoległych procesów zapytań serwera dla celów wykonania zapytania i sortowania danych, wówczas dla wszystkich pozostałych użytkowników bazy danych dostępne pozostaną jedynie dwa równoległe procesy zapytań serwera. Należy zatem ostrożnie udostępniać możliwości zrównoleglenia dla zapytań i operacji wsadowych. Gdy wartością parametru PARALLEL_ADAPTIVE_MULTI_USER jest TRUE, włączony jest algorytm adaptacyjny, którego celem jest zwiększenie wydajności w środowiskach dla wielu użytkowników poprzez równoległe wykonywanie zadań. Algorytm automatycznie redukuje żądany stopień zrównoleglenia, bazując na obciążeniu systemu w momencie uruchomienia zapytania. Rzeczywisty stopień zrównoleglenia jest ustalany na podstawie domyślnego stopnia zrównoleglenia lub stopnia zrównoleglenia dla tabeli albo stopnia zdefiniowanego przez wskazówkę PARALLEL, podzielonego przez współczynnik redukcji.

Dla każdej tabeli można zdefiniować domyślny stopień zrównoleglenia. Służy do tego klauzula parallel poleceń create table oraz alter table. Stopień zrównoleglenia wskazuje serwerowi Oracle liczbę równoległych procesów zapytań serwera, jakich można użyć dla celów poszczególnych etapów operacji. Jeśli na przykład zapytaniu, które wykonuje skanowanie tabeli oraz sortowanie danych, przypisano stopień zrównoleglenia równy 5, oznacza to, że użytych może zostać do dziesięciu równoległych procesów zapytań serwera: pięć na skanowanie i pięć na sortowanie. Możliwe jest również wskazanie stopnia zrównoleglenia dla indeksu w momencie jego tworzenia. Do tego celu wykorzystuje się klauzulę parallel polecenia create index.

Minimalna liczba uruchomionych równoległych procesów zapytań serwera jest definiowana przez parametr inicjalizacyjny PARALLEL_MIN_SERVERS. Generalnie wartość parametru powinna być bardzo niska i nie przekraczać liczby 5, chyba że system jest intensywnie używany przez całą dobę. Przypisanie parametrowi niskiej wartości spowoduje, że Oracle będzie musiał uruchamiać coraz to nowe procesy zapytań serwera, ale też doprowadzi do istotnego zmniejszenia ilości pamięci zajmowanej przez jałowe procesy zapytań serwera w trakcie okresów obniżonej intensywności pracy w systemie. Jeśli parametrowi PARALLEL_MIN_SERVERS przypisana zostanie wysoka wartość, wówczas na serwerze może często funkcjonować znaczna liczba jałowych procesów zapytań serwera, które będą zajmować pozyskaną wcześniej pamięć, a nie będą wykonywać jakiejkolwiek operacji.

Zrównoleglenie operacji rozkłada wymagania wynikające z ich przetwarzania na wiele procesorów. Funkcji tych trzeba jednak używać z rozwagą. Jeśli dla dużego zapytania użyty zostanie stopień zrównoleglenia o wartości 5, wówczas dane będą odczytywane przez pięć oddzielnych procesów. A w przypadku tak dużej liczby procesów odczytujących dane może dochodzić do konfliktów w dostępie do danych na dyskach, co wpłynie na wydajność działania systemu. Poprzez użycie Parallel Query zrównoleglenie powinno się stosować selektywnie, wobec tych tabel, których dane są odpowiednio rozproszone na wielu urządzeniach fizycznych. Należy również unikać korzystania ze zrównoleglenia we wszystkich tabelach,

ponieważ — jak już wcześniej wspomniano — pojedyncze zapytane może użyć wszystkich dostępnych równoległych procesów zapytań serwera i uniemożliwić w ten sposób korzystanie ze zrównoleglenia całej reszcie transakcji wykonywanych w bazie danych.

Efektywne projektowanie aplikacji

W dalszej części tego rozdziału zostaną przedstawione szczegółowe zagadnienia dotyczące projektowania aplikacji. Istnieje jednak również kilka bardziej ogólnych wskazówek dotyczących projektowania aplikacji bazy danych Oracle.

Po pierwsze, należy minimalizować liczbę żądań odczytu danych z bazy danych. W tym celu można korzystać z sekwencji, bloków kodu PL/SQL, a także denormalizować tabele. Można także używać rozproszonych obiektów baz danych, takich jak widoki materializowane, i w ten sposób zmniejszać liczbę odczytów danych z bazy.


Uwaga

> Nawet tylko nieznacznie nieefektywny kod SQL może negatywnie wpływać na wydajność całej bazy danych, jeśli będzie dostatecznie często używany. Kod SQL, który wykonuje niewielką liczbę fizycznych operacji wejścia-wyjścia bądź też nie generuje ich wcale, nadal zużywa zasoby procesora.

Po drugie, różni użytkownicy tej samej aplikacji powinni wykonywać zapytania na bazie danych w sposób jak najbardziej zbliżony. Wykorzystywanie spójnych ścieżek dostępu do danych zwiększa prawdopodobieństwo, że żądania będą przetwarzane przy użyciu danych, które są już dostępne w SGA. Współużytkowanie danych jest możliwe nie tylko dzięki już wcześniej odczytanym wierszom i tabelom, ale również dzięki wcześniej używanym zapytaniom. Jeżeli zapytania okażą się identyczne, wówczas sparsowana wersja zapytania może już znajdować się we wspólnym obszarze SQL, a to z kolei pozwoli na skrócenie czasu przetwarzania zapytania. Dokonane w optymalizatorze rozszerzenia mechanizmów obsługujących współużytkowanie kursorów zwiększają prawdopodobieństwo, że instrukcje znajdujące się we wspólnym obszarze będą mogły zostać ponownie użyte; najpierw jednak aplikację trzeba zaprojektować właśnie z myślą o wielokrotnym używaniu instrukcji.

Po trzecie, należy ograniczać zakres użycia dynamicznego kodu SQL. Z założenia tego typu kod jest niezdefiniowany do chwili wykonania go. Dynamiczny kod SQL aplikacji za pierwszym razem może pobrać kilka wierszy, za drugim przeprowadzić kilka operacji pełnego skanowania tabeli zamówień, a za trzecim przypadkowo zastosować złączenie kartezjańskie (operacja ta może zostać świadomie wykonana przez użycie słów kluczowych `cross join` w poleceniu `select`!). Dodatkowo do momentu wykonania dynamicznego kodu SQL nie można zagwarantować, że jest on poprawny składniowo. Dynamicznie generowany kod SQL jest jak obosieczny miecz. Z jednej strony uzyskuje się możliwość dynamicznego tworzenia kodu SQL na podstawie danych wprowadzonych przez użytkownika, a z drugiej zarówno aplikacje wewnętrzne, jak i zewnętrzne aplikacje WWW narażone są na ataki typu *SQL Injection*.

Po czwarte, należy minimalizować liczbę przypadków, gdy otwierana jest i zamykana sesja w bazie danych. Jeśli aplikacja często otwiera sesję, wykonuje niewielką liczbę poleceń, a następnie tę sesję zamyka, wówczas wydajność kodu SQL może być nieistotnym czynnikiem w analizie całkowitej wydajności rozwiązania. Zarządzanie sesją może być kosztowniejsze niż jakakolwiek inna czynność wykonywana w aplikacji.

Gdy używane są procedury składowane, ten sam kod może być używany wielokrotnie i korzystać dzięki temu z dobrodziejstw obszaru wspólnego. Można także ręcznie kompilować procedury, funkcje i pakiety, aby w ten sposób unikać ich kompilowania w fazie wykonania. Gdy tworzy się procedurę, Oracle automatycznie przeprowadza jej kompilację. Jeśli w późniejszym czasie procedura staje się nieprawidłowa, baza danych musi ją ponownie skompilować przed wykonaniem. Aby unikać konieczności ponoszenia kosztów związanych z procesem kompilacji, można użyć polecenia `alter procedure` w następującej postaci:

```
alter procedure MY_RAISE compile;
```

Treść kodu SQL wszystkich procedur znajdujących się w bazie danych można odczytać z kolumny `Text` widoku `DBA_SOURCE`. Widok `USER_SOURCE` wyświetla wszystkie procedury, których właścicielem jest użytkownik wykonujący zapytanie. Również kod źródłowy pakietów i funkcji jest dostępny w widokach `DBA_SOURCE` i `USER_SOURCE`, które z kolei odwołują się do tabeli o nazwie `SYS.SOURCE$`.

Pierwsze dwie przytoczone wskazówki projektowania aplikacji, czyli ograniczenie liczby operacji odczytu wykonywanych przez użytkownika oraz koordynowanie żądań pochodzących od użytkowników, wymagają, by twórca aplikacji posiadał jak najbardziej szczegółową wiedzę na temat sposobów, w jakie dane będą odczytywane oraz na temat używanych ścieżek dostępu do tych danych. Z tego powodu jest niezwykle istotne, by użytkowników angażować co najmniej w takim samym stopniu w proces projektowania aplikacji, w jaki angażuje się ich w projektowanie struktury tabel. Jeżeli użytkownicy spędzają długie godziny z projektantami modelu danych na rysowaniu modelu tabel, a czas poświęcany przez tych użytkowników na pracę z twórcami aplikacji mającej na celu zaprojektowanie ścieżek dostępu do danych jest nieporównanie krótszy, wówczas z dużym prawdopodobieństwem tworzona aplikacja nie spełni oczekiwań użytkowników. Ścieżki dostępu do danych powinny być analizowane jako jeden z etapów procesu modelowania danych.

Strojenie kodu SQL

Podobnie jak w przypadku projektowania aplikacji może się wydawać, że strojenie instrukcji języka SQL także dalece wykracza poza zakres obowiązków administratora bazy danych. Nic bardziej mylnego. Administratorzy baz danych powinni brać udział w analizowaniu kodu SQL tworzonego jako część aplikacji. W prawidłowo zaprojektowanej aplikacji nadal mogą występować problemy z wydajnością, jeśli używany w niej kod SQL będzie źle dostrojony. Sposób zaprojektowania aplikacji oraz problemy z kodem SQL są najczęstszymi przyczynami słabej wydajności prawidłowo zaprojektowanej bazy danych.

Kluczową sprawą w strojeniu kodu SQL jest zminimalizowanie ścieżek przeszukiwania, których używa baza do odnajdywania danych. W większości tabel bazy danych Oracle każdy wiersz posiada swój własny identyfikator `RowID`. Identyfikator `RowID` zawiera informacje na temat fizycznej lokalizacji wiersza: jego pliku, bloku w tym pliku oraz wiersza w bloku bazy danych.

Gdy wykonywane jest zapytanie bez klauzuli `where`, baza danych zwykle wykonuje pełen skan tabeli i odczytuje wszystkie jej bloki. W trakcie pełnego skanu tabeli baza danych lokalizuje pierwszy blok tabeli, a następnie sekwencyjnie odczytuje wszystkie następne bloki. Pełne skanowanie bardzo dużych tabel może być operacją niezwykle czasochłonną.

Jeśli zapytanie dotyczy konkretnych wierszy, baza danych może wykorzystać indeks i przyspieszyć za jego pomocą odczyt pożądanych wierszy. Indeks odwzorowuje znajdujące się w tabeli wartości logiczne na odpowiadające im identyfikatory `RowID`, które z kolei odwzorowują te wartości na ich fizyczne lokalizacje. Indeksy mogą być unikatowe (wówczas każda wartość występuje tylko jeden raz) lub nie. Indeksy przechowują indeksy wierszy `RowID` jedynie dla wartości `NOT NULL` z indeksowanej kolumny.

Indeksować można jednocześnie więcej niż jedną kolumnę. Tworzy się wówczas tak zwany **indeks połączony** (ang. *concatenated index*) lub **indeks złożony** (ang. *composite index*), który będzie używany, gdy pierwsza kolumna indeksu zostanie wykorzystana jako kryterium zapytania w klauzuli `where`. Optymalizator może również zastosować tak zwane podejście *skip-scan*, w którym indeks połączony jest używany nawet wówczas, gdy jego pierwsza kolumna nie wchodzi w skład klauzuli `where` zapytania.

Indeksy należy dostosowywać do wymaganych ścieżek dostępu do danych. Zastanówmy się nad przypadkiem indeksu połączonego złożonego z trzech kolumn. Jak pokazano na poniższym przykładzie, indeks jest tworzony na kolumnach `City`, `State` i `Zip` tabeli `EMPLOYEE`:

```
create index CITY_ST_ZIP_NDX
on EMPLOYEE(City, State, Zip)
tablespace INDEXES;
```

Weźmy też pod uwagę następujące zapytanie:

```
select * from EMPLOYEE
where State='NJ';
```

Jak widać w powyższym zapytaniu, pierwsza kolumna indeksu (`City`) nie występuje w klauzuli `where`. Oracle korzysta z dwóch metod dostępu do wierszy wykorzystujących indeksy: skanowanie indeksu typu *skip-scan* oraz pełen skan indeksu. Optymalizator wybierze ścieżkę dostępu do danych na podstawie statystyk indeksu: jego rozmiaru, rozmiaru tabeli oraz selektywności indeksu. Jeśli użytkownicy często będą wykonywać tego rodzaju zapytanie, może zajść potrzeba zmiany kolejności kolumn w indeksie tak, by pierwszą kolumną stała się `State` i by w ten sposób został odzwierciedlony rzeczywisty sposób używania indeksu.

Skan zakresu indeksu to kolejna optymalizacja bazująca na indeksach, której system Oracle może użyć do efektywnego pobierania konkretnych danych. System korzysta ze skanu zakresu indeksu, gdy zmienna w klauzuli `where` jest równa określonej stałej, mniejsza lub większa od niej, a ponadto zmienna jest kolumną główną, jeśli indeks jest indeksem złożonym. Klauzula `order by` jest niezbędna to tego, aby wiersze zostały zwrócone w kolejności indeksowania, tak jak w przypadku poniższego przykładowego zapytania, które wyszukuje pracowników zatrudnionych przed 1 sierpnia 2007 r.

```
select * from EMPLOYEE where hire_date < '1-AUG-2007';
```

Niezwykle istotne jest, by zapewnić odpowiedni porządek danych w tabeli. Jeśli użytkownicy często wykonują zapytania o **zakres** danych — to znaczy pytają o wartości, które należą do jakiegoś konkretnego zakresu — wówczas dzięki uporządkowaniu danych liczba bloków danych, których odczytanie jest konieczne w celu wykonania zapytania, może się zmniejszyć, poprawiając wydajność bazy. Poukładane w odpowiedniej kolejności pozycje w indeksie będą wskazywać na zbiór sąsiednich bloków należących do tabeli, zamiast na bloki rozsiane po całym pliku (lub plikach) danych.

Rozważmy poniższe przykładowe zapytanie o zakres danych:

```
select *
from EMPLOYEE
where Empno between 1 and 100;
```

Jeśli fizyczne rekordy tabeli EMPLOYEE będą posortowane względem kolumny EMPNO, powyższe zapytanie o zakres danych będzie wymagało odczytania mniejszej liczby bloków danych. Aby uzyskać gwarancję, że wiersze tabeli będą prawidłowo posortowane, można zrzucić rekordy do pliku jednorodnego (albo do innej tabeli), posortować tam je, a następnie usunąć stare rekordy i na ich miejsce załadować na nowo zbiór posortowanych danych. Dodatkowo powinno się wykonać operację zmniejszania segmentów online, aby dla tabel o dużej aktywności poleceń DML przywrócić ilość pofragmentowanej wolnej przestrzeni poniżej poziomu maksymalnego stanu. W ten sposób poprawi się wykorzystanie bufora, a w przypadku pełnych skanów tabeli będzie musiała być przeszukana mniejsza liczba bloków. W celu scalenia wolnej przestrzeni w tabeli należy użyć instrukcji `alter table ... shrink space`.

Wpływ kolejności danych na proces ładowania danych do bazy

Indeksy wpływają na wydajność zarówno zapytań, jak i operacji ładowania danych. W trakcie wykonywania instrukcji `insert` kolejność wierszy ma znaczny wpływ na wydajność ładowania danych. Nawet w środowiskach, w których indeksy są znacznie obciążone, odpowiednie posortowanie wierszy jeszcze przed wykonaniem polecenia `insert` może zwiększyć wydajność ładowania danych nawet o 50 procent.

W miarę powiększania się indeksu serwer Oracle alokuje nowe bloki. Gdy nowa pozycja indeksu jest wstawiana za pozycją wstawioną ostatnio, dodawana pozycja trafi do ostatniego bloku indeksu. Jeżeli obecność nowej pozycji spowoduje, że przekroczone zostanie miejsce dostępne w bloku, pozycja zostanie przeniesiona do nowego bloku. Alokowanie bloków w taki sposób nie ma większego wpływu na wydajność bazy danych.

Jeśli wstawiane wiersze nie będą posortowane, wówczas nowe pozycje indeksu będą zapisywane w istniejących blokach węzłów indeksu. Jeżeli w bloku, do którego dodawana jest nowa wartość, brakuje już miejsca, a blok ten nie jest ostatnim blokiem indeksu, wówczas znajdujące się w nim pozycje zostaną podzielone na dwie części. Połowa z nich pozostanie w bloku dotychczasowym, zaś druga połowa zostanie przeniesiona do nowego bloku. W efekcie wydajność operacji ładowania danych obniży się (ponieważ konieczne będzie wykonanie dodatkowych czynności związanych z zarządzaniem przestrzenią), tak samo jak obniży się wydajność zapytań (ponieważ w indeksie znajduje się więcej nieużywanego miejsca, przez co w celu odczytania takiej samej liczby pozycji konieczne jest odczytanie większej liczby bloków).


Uwaga

Zwiększenie w indeksie liczby jego poziomów wewnętrznych znacznie obniża wydajność operacji wstawiania danych. Aby sprawdzić aktualną liczbę poziomów, należy przeprowadzić analizę indeksu, a następnie z widoku DBA_INDEXES odczytać zawartość kolumny poziomów B.

Ze względu na sposób, w jaki Oracle wewnętrznie zarządza indeksami, dodawanie nowych indeksów zawsze wpłynie na wydajność ładowania danych (ponieważ jest mało prawdopodobne, by wstawiane wiersze były posortowane odpowiednio dla większej liczby kolumn). Z punktu widzenia wydajności ładowania danych lepiej jest używać niewielu indeksów obejmujących wiele kolumn niż wielu indeksów na pojedynczych kolumnach.

Dodatkowe opcje indeksowania

Jeśli dane nie są zbyt selektywne, można rozważyć użycie **indeksów bitmapowych** (ang. *bitmap indexes*). Zgodnie z opisem z rozdziału 16., indeksy bitmapowe sprawdzają się najlepiej w przypadku zapytań na dużych, statycznych zbiorach danych z niewielką liczbą wartości unikatowych. Na tej samej tabeli można utworzyć zarówno indeksy bitmapowe, jak i indeksy zwykłe (indeksy *B-tree*). Wówczas serwer Oracle sam dynamicznie przeprowadzi odpowiednie konwersje indeksów w trakcie przetwarzania zapytania. Więcej informacji na temat indeksów bitmapowych znajduje się w rozdziale 16.


Uwaga

Należy unikać tworzenia indeksów bitmapowych na tabelach, które są modyfikowane przez transakcje online. Z kolei tabele hurtowni danych znakomicie nadają się do zastosowania indeksów bitmapowych.

Jeżeli dwie tabele są często przedmiotem jednego zapytania, wówczas efektywnym narzędziem podwyższania wydajności mogą okazać się **klastry**. Klastry przechowują wiersze pochodzące z różnych tabel w tych samych fizycznych blokach danych, zależnie od ich wartości logicznych (tak zwanego klucza klastrowego).

Zapytania, w których wartość kolumny jest porównywana z konkretną wartością (a nie z zakresem wartości), to tak zwane zapytania równowartościowe. **Klaster haszowy** (ang. *hash cluster*) przechowuje wiersz w konkretnej lokalizacji, wyznaczanej przez wartość znajdującą się w kolumnie klucza klastrowego. W momencie wstawiania każdego wiersza na podstawie jego klucza klastrowego ustalany jest blok, w którym wiersz należy zapisać. Tę samą logikę można wykorzystać w trakcie przetwarzania zapytań, aby szybko odnajdywać bloki danych, na których należy wykonać operacje odczytu. Klastry haszowe zostały zaprojektowane w taki sposób, by zwiększyć wydajność zapytań równowartościowych. Te same klastry nie będą natomiast już tak znacząco zwiększać wydajności zapytań o wartości z zakresów, o których mówiliśmy wcześniej w tym rozdziale. Wydajność będzie znacznie gorsza w przypadku zapytań o zakres danych, zapytań wymuszających pełny skan tabeli lub klastrów haszowych, które są często aktualizowane.

Indeksy odwrócone (ang. *reverse indexes*) to kolejne narzędzie do strojenia zapytań równowartościowych. W indeksie odwróconym bajty indeksu są przechowywane w odwrotnej kolejności. W indeksie tradycyjnym dwie następujące po sobie wartości są zapisywane obok siebie. W indeksie odwróconym wartości następujące po sobie nie sąsiadują ze sobą. Na przykład wartości 2004 i 2005 są przechowywane w indeksie odwróconym odpowiednio w postaci 4002 i 5002. Indeksy odwrócone niezbyt nadają się do skanowania zakresów wartości, natomiast mogą doprowadzić do zmniejszenia liczby konfliktów w dostępie do bloków indeksu w sytuacjach, gdy wykonywanych jest znaczna liczba zapytań równowartościowych. Aby dobrze działać, indeksy z kluczem odwróconym mogą wymagać dość częstego odbudowywania.

W celu umożliwienia operacji wstawiania danych indeksy te powinny też dysponować dużą wartością parametru PCTFREE.


Uwaga

Nie można odwrócić indeksu bitmapowego.

Na wyrażeniach, w których używa się kolumn, można tworzyć **indeksy funkcyjne** (ang. *function-based indexes*). Poniższe zapytanie nie mogłoby użyć indeksu *B-tree* na kolumnie Name:

```
select * from EMPLOYEE
where UPPER(Name) = 'JONES';
```

Odwrotnie rzecz ma się z następującym zapytaniem:

```
select * from EMPLOYEE
where Name = 'JONES';
```

Drugie zapytanie może skorzystać z indeksu *B-tree*, ponieważ nie wykonuje na kolumnie Name żadnej funkcji. Zamiast tworzyć indeks na kolumnie Name, można utworzyć indeks na wyrażeniu kolumnowym UPPER(Name), jak w poniższym przykładzie:

```
create index EMP_UPPER_NAME on
EMPLOYEE(UPPER(Name));
```

Indeksy funkcyjne mogą być użyteczne, jednak zawsze, gdy się je tworzy, należy najpierw rozważyć wszystkie poniższe zagadnienia:

- Czy można ograniczyć zakres funkcji używanych względem kolumny? Jeśli tak, to czy w ogóle można wyeliminować wykonywanie funkcji na kolumnie?
- Czy dostępna jest ilość miejsca na dysku odpowiednia do przechowywania dodatkowych indeksów?
- W momencie usuwania tabeli usuwanych będzie więcej indeksów (a więc również więcej obszarów) niż dotychczas. Jak wpłynie to na czas, w jakim tabela powinna zostać usunięta (jest to mniej istotne w przypadku stosowania lokalnie zarządzanych przestrzeni tabel; z funkcji tej powinno się korzystać w systemie Oracle Database 10*g* lub nowszym)?

Indeksy funkcyjne są przydatne, lecz należy je stosować z rozwagą. Im więcej indeksów zostanie utworzonych na tabeli, tym dłużej wykonywane będą operacje insert, update i delete. Oczywiście dotyczy to tworzenia na tabeli dowolnych dodatkowych indeksów, niezależnie od ich typu.

Indeksy tekstowe (ang. *text indexes*) korzystają z funkcji przetwarzania tekstu bazy Oracle (Oracle Text) do tworzenia i zarządzania listami słów oraz ich wystąpieniami. Listy te działają w sposób podobny do indeksów w książkach. Indeksy tekstowe są najczęściej używane do obsługi aplikacji, które wyszukują części słów na podstawie znaków globalnych.

W tabelach partycjonowanych mogą występować indeksy, które rozciągają się na wszystkie partycje (są to tak zwane indeksy globalne), albo indeksy, które są partycjonowane w podobny sposób co tabele (tak zwane indeksy lokalne). Z punktu widzenia strojenia zapytań zwykle przydatniejsze są indeksy lokalne, ponieważ znajduje się w nich mniej pozycji niż w indeksach globalnych.

Generowanie opisów planów wykonania

W jaki sposób można ustalić ścieżkę dostępu, z jakiej skorzysta baza danych w celu wykonania zapytania? Informację taką można uzyskać poleceniem `explain plan`. Polecenie oblicza ścieżkę wykonania dla zapytania i umieszcza wynik obliczeń w tabeli `PLAN_TABLE` bazy danych. Przykładowe zapytanie `explain plan` może mieć następującą postać:

```
explain plan
 for
select *
  from BOOKSHELF
 where Title like 'M%';
```

Pierwszy wiersz powyższego polecenia wskazuje bazie danych, że należy stworzyć plan wykonania zapytania, lecz bez wykonywania samego zapytania. Opcjonalnie w poleceniu można zawrzeć klauzulę `set Statement_ID`, aby w tabeli `PLAN_TABLE` nadać opisowi planu wykonania odpowiednią etykietę. Po słowie kluczowym `for` umieszcza się zapytanie, którego opis planu wykonania ma zostać zwrócony.

W schemacie konta, na którym wykonywane jest przedstawione polecenie, musi znajdować się tabela planu. Oracle udostępnia polecenia `create table` niezbędne dla tej tabeli. Zawierający je plik o nazwie *utlxplan.sql* znajduje się zwykle w podkatalogu *$ORACLE_HOME/rdbms/admin* głównego katalogu oprogramowania Oracle. Użytkownicy mogą skorzystać ze wspomnianego skryptu, aby utworzyć tabelę planu we własnym schemacie.


Uwaga

> Po każdym uaktualnieniu bazy Oracle należy usunąć i ponownie utworzyć tabelę planu, ponieważ skrypty uaktualniające wersje mogą dodawać nowe kolumny.

W celu wykonania zapytania na tabeli planu należy użyć procedury `DBMS_XPLAN`:

```
select * from table(DBMS_XPLAN.DISPLAY);
```

W celu skierowania zapytania do tabeli planu, które dotyczy wykonywania szeregowego lub równoległego, można też użyć skryptów systemu Oracle znajdujących się odpowiednio w plikach *$ORACLE_HOME/rdbms/admin/utlxpls.sql* i *$ORACLE_HOME/rdbms/admin/utlxplp.sql*.

Przedstawione zapytanie zwróci informacje na temat rodzajów operacji, jakie musi wykonać baza danych, aby przetworzyć zapytanie. Zwrócone dane wynikowe będą prezentować kolejne etapy wykonywania zapytania w postaci hierarchicznej i jednocześnie ilustrować relacje między poszczególnymi etapami. Zapytanie może na przykład zwrócić etap, w którym korzysta się z indeksu, którego przodkiem jest polecenie `TABLE ACCESS BY INDEX ROWID`. Na tej podstawie można wywnioskować, że najpierw wykonywany jest etap, w którym wykorzystywany jest indeks, a następnie na podstawie identyfikatorów wierszy `RowID` zwróconych z indeksu odczytywane są odpowiednie wiersze tabeli.

W narzędziu SQL*Plus można używać polecenia `set autotrace on`, aby dla każdego wykonywanego zapytania automatycznie zwracane były wyniki polecenia `explain plan` oraz dane śledzenia. Automatycznie generowane dane śledzenia zostaną zwrócone dopiero po zakończeniu wykonywania zapytania, natomiast wynik polecenia `explain plan` zostanie wygenerowany bez uruchamiania polecenia. Aby włączyć możliwość automatycznego generowania

danych śledzenia w schemacie, w którym narzędzie automatycznego śledzenia będzie używane, musi istnieć tabela planu. Alternatywnie tabela planu może znajdować się w schemacie SYSTEM, który posiada jednocześnie dostęp do schematu używającego narzędzia automatycznego śledzenia. Przed włączeniem opcji `set autotrace on` konieczne jest również wykonanie (jako użytkownik SYS) skryptu *plustrce.sql*, zlokalizowanego w podkatalogu *$ORACLE_HOME/sqlplus/admin* głównego katalogu oprogramowania Oracle. Aby móc wykonać `set autotrace on`, użytkownicy muszą ponadto posiadać rolę PLUSTRACE.


Uwaga

> Aby uzyskać opis planu wykonania bez wykonywania zapytania, należy użyć polecenia `set autotrace traceonly explain`.

Jeżeli używane są opcje zapytań równoległych bądź też zapytania są wykonywane na zdalnych bazach danych, `set autotrace on` zwróci dodatkową sekcję, w której znajdować się będzie treść zapytań wykonywanych przez równoległe procesy zapytań serwera lub treść zapytań wykonywanych w zdalnych bazach danych.

Aby wyłączyć opcję automatycznego śledzenia, należy wykonać polecenie `set autotrace off`.

Poniższy kod źródłowy ilustruje sposób, w jaki włącza się automatyczne śledzenie i generuje opis planu wykonania:

```
set autotrace on trace explain

select *
  from BOOKSHELF
 where Title like 'M%';

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=3 Card=2 Bytes=80)
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'BOOKSHELF' (TABLE) (Cost
          =3 Card=2 Bytes=80)
   2    1     INDEX (RANGE SCAN) OF 'SYS_C004834' (INDEX (UNIQUE)) (Co
          st=1 Card=2)
```

Aby zrozumieć opis planu wykonania, należy zacząć czytać operacje znajdujące się wewnątrz hierarchii od operacji najbardziej wewnętrznych aż do momentu, gdy dojdzie się do zbioru operacji zapisanych z tym samym wcięciem. Następnie należy odczytywać operacje od góry do dołu. W przedstawionym przykładzie nie występują operacje zapisane z tym samym wcięciem, zatem operacje należy czytać od końca. Pierwszą operacją jest skanowanie zakresu indeksu, po czym uzyskiwany jest dostęp do tabeli. Ostatnia operacja SELECT STATEMENT zwraca dane wynikowe do użytkownika. Każda operacja posiada własny identyfikator (pierwsza kolumna) oraz identyfikator przodka (drugi numer, który dla operacji położonej najwyżej jest pusty). W bardziej skomplikowanych opisach planów wykonania konieczne może być użycie identyfikatorów przodków, by ustalić rzeczywistą kolejność wykonywania operacji.

Przedstawiony powyżej plan wskazuje, że dane zwracane użytkownikowi są pozyskiwane w ramach operacji TABLE ACCESS BY INDEX ROWID. Identyfikatory wierszy RowID są wynikiem operacji skanowania zakresu indeksu unikatowego.

Wykonanie każdego etapu wiąże się z poniesieniem określonego kosztu (ang. *cost*). Koszt jest prezentowany w postaci skumulowanej, to znaczy koszt każdej operacji jest równy jej rzeczywistemu kosztowi oraz sumie kosztów wszystkich jej operacji potomnych. Na podstawie wartości kosztów można zidentyfikować te etapy, które stanowią najbardziej znaczące części kosztu wykonania zapytania. Tak zidentyfikowane etapy powinny następnie jako pierwsze podlegać strojeniu.

Przed wykonaniem polecenia `explain plan` należy się upewnić, że w zapytaniu używane są indeksy o najwyższym stopniu selektywności (czyli indeksy najbardziej zbliżone do indeksów unikatowych). Jeżeli użyte zostaną indeksy o niskim stopniu selektywności, baza danych może być zmuszona do wykonania niepotrzebnych operacji odczytu w celu przetworzenia zapytania. Szczegółowa dyskusja na temat strojenia kodu SQL wykracza poza zakres niniejszej książki, należy jednak pamiętać, by dążyć do tego, aby najbardziej „zasobożerne" instrukcje języka SQL korzystały z możliwe najbardziej selektywnych indeksów.

W ogólnym ujęciu wydajność aplikacji zorientowanych transakcyjnie (czyli systemów do wpisywania danych, z których korzysta wielu użytkowników) ocenia się na podstawie ilości czasu, po jakim zapytanie zwraca pierwszy wiersz. W aplikacjach zorientowanych transakcyjnie proces strojenia powinno się ukierunkowywać przede wszystkim na takie użycie indeksów, by zmniejszać czas odpowiedzi bazy danych na zapytanie.

Jeśli aplikacja jest zorientowana wsadowo (to znaczy wykonywane są w niej duże transakcje i raporty), należy skupić się przede wszystkim na skracaniu całkowitego czasu trwania transakcji, a nie na czasie, po jakim transakcja zwraca pierwszy wiersz. Zwiększenie całkowitej wydajności transakcji może wymagać zastosowania pełnego skanowania tabeli zamiast korzystania z indeksów i tym samym doprowadzić do zwiększenia ogólnej wydajności aplikacji.

Jeśli aplikacja korzysta z wielu rozproszonych baz danych, należy dążyć do tego, by liczba przypadków, w których w zapytaniach używa się łączy do baz danych, była jak najmniejsza. Jeśli w danym zapytaniu często odczytuje się dane ze zdalnej bazy danych, koszt uzyskiwania dostępu do takiej zdalnej bazy jest ponoszony za każdym razem, gdy następuje odczyt danych zdalnych. Nawet jeśli koszt dostępu do zdalnych danych jest stosunkowo niewielki, odczytywanie ich tysiące razy w końcu znajdzie swe odzwierciedlenie w obniżeniu wydajności aplikacji. Więcej wskazówek dotyczących strojenia rozproszonych baz danych znajduje się w punkcie „Zmniejszanie ruchu w sieci" w dalszej części tego rozdziału.

Strojenie sposobów użycia pamięci

Począwszy od wersji Oracle 10*g*, można także korzystać z zestawu narzędzi Automatic Workload Repository (AWR) i za ich pomocą gromadzić i zarządzać danymi statystycznymi (więcej informacji na ten temat w dalszej części niniejszego rozdziału). Począwszy od systemu Oracle 11*g*, do dalszego automatyzowania zarządzania ogólną pamięcią używaną przez bazę Oracle można zastosować nowe parametry inicjalizacyjne, takie jak `MEMORY_TARGET`. Gdy nie ma czasu na czytanie raportów narzędzia AWR, parametr ten pomoże w automatycznym strojeniu bazy danych.

Bufor pamięci podręcznej bloków danych oraz obszar wspólny są zarządzane przy użyciu algorytmu „najdawniej używanych" (ang. *least recently used — LRU*). W algorytmie tym wartości są przechowywane w wyznaczonym obszarze pamięci; gdy obszar ten się wypełni, wartość najdawniej używana zostaje z niego usunięta i z powrotem zapisana na dysku. Odpowiednio zwymiarowany obszar pamięci przechowuje najczęściej używane dane, natomiast w celu uzyskania dostępu do danych rzadziej używanych konieczne jest ich odczytanie z dysku.

Zapytania wykonujące logiczne i fizyczne operacje odczytu na bazie danych można przeglądać w widoku V$SQL. Widok V$SQL zawiera skumulowaną liczbę wszystkich operacji logicznego i fizycznego odczytu, wykonanych przez każde z zapytań aktualnie znajdujących się w obszarze wspólnym, a także liczbę wykonań każdego z tych zapytań. Poniższy skrypt jest poleceniem SQL zwracającym zapytania znajdujące się w obszarze wspólnym, przy czym zapytania o największej intensywności operacji wejścia-wyjścia są zwracane jako pierwsze. Zapytanie zwraca ponadto liczbę logicznych operacji odczytu (czyli odczytów z bufora) na jedno wykonanie zapytania:

```
select Buffer_Gets,
       Disk_Reads,
       Executions,
       Buffer_Gets/Executions B_E,
       SQL_Text
 from V$SQL where executions != 0
order by Disk_Reads desc;
```

Jeśli obszar wspólny został opróżniony, zapytania wykonane przed jego opróżnieniem nie będą już dostępne w widoku V$SQL. Nadal jednak można analizować wpływ tych zapytań na wydajność, jeśli tylko użytkownicy je wykonujący wciąż są zalogowani. Widok V$SESS_IO rejestruje skumulowaną liczbę operacji logicznych i fizycznych odczytów w poszczególnych sesjach użytkownika. Z widoku V$SESS_IO można odczytać stosunek operacji odczytów dla każdej sesji, używając na przykład poniższego zapytania:

```
select SESS.Username,
       SESS_IO.Block_Gets,
       SESS_IO.Consistent_Gets,
       SESS_IO.Physical_Reads,
       round(100*(SESS_IO.Consistent_Gets
           +SESS_IO.Block_Gets-SESS_IO.Physical_Reads)/
           (decode(SESS_IO.Consistent_Gets,0,1,
               SESS_IO.Consistent_Gets+SESS_IO.Block_Gets)),2)
                 session_hit_ratio
 from V$SESS_IO sess_io, V$SESSION sess
where SESS.Sid = SESS_IO.Sid
  and SESS.Username is not null
order by Username;
```

Aby uzyskać obiekty, których bloki znajdują się aktualnie w buforze pamięci podręcznej bloków danych, należy wykonać odpowiednie zapytanie na tabeli X$BH w schemacie SYS, jak w poniższym przykładzie (należy zwrócić uwagę, że obiekty SYS i SYSTEM nie wchodzą do zbioru danych wynikowych, aby administrator bazy danych mógł skupić się na tabelach i indeksach aplikacji znajdujących się w SGA):

```
select Object_Name,
       Object_Type ,
       count(*) Num_Buff
  from X$BH a, SYS.DBA_OBJECTS b
 where A.Obj = B.Object_Id
```

```
  and Owner not in ('SYS','SYSTEM')
group by Object_Name, Object_Type;
```


Uwaga

> Analogiczne dane można uzyskać przez wykonanie zapytania na kolumnach `Name` i `Kind` widoku `V$CACHE` (jeśli nie nawiązano połączenia jako użytkownik `SYS`).

W buforze pamięci podręcznej bloków danych występuje kilka różnych obszarów pamięci podręcznej:

- **Pamięć podręczna DEFAULT** — jest to standardowa pamięć podręczna przeznaczona dla obiektów, które używają domyślnych rozmiarów bloków bazy danych.
- **Pamięć podręczna KEEP** — jest to pamięć podręczna przeznaczona dla obiektów, które mają być przechowywane w pamięci przez cały czas. Generalnie obszar ten jest używany z myślą o małych tabelach, w których wykonywanych jest niewielka liczba transakcji. Ta pamięć podręczna przydaje się w przypadku wyszukiwania w tabelach takich rzeczy, jak kody województw, kody pocztowe i dane dotyczące sprzedawców.
- **Pamięć podręczna RECYCLE** — jest to pamięć podręczna przeznaczona dla obiektów, które mają być niezwłocznie usuwane z pamięci. Podobnie jak obszar KEEP, pamięć podręczna RECYCLE izoluje obiekty w pamięci, tak by nie kolidowały one z normalnie funkcjonującą pamięcią podręczną DEFAULT.
- **Pamięci podręczne dla bloków o określonych rozmiarach** — Oracle może obsługiwać w jednej bazie danych kilka różnych rozmiarów bloków bazy danych. Dla każdego rozmiaru bloku bazy danych innego niż domyślny należy utworzyć oddzielny obszar pamięci podręcznej.

W przypadku wszystkich obszarów SGA — buforów bloków danych, pamięci podręcznej słowników oraz obszaru wspólnego — należy zawsze kłaść nacisk na współużytkowanie danych przez wielu użytkowników. Każdy z wymienionych obszarów powinien być na tyle duży, by móc przechowywać w nim dane, które są najczęściej odczytywane z bazy danych. Jeśli chodzi o obszar wspólny, powinien on być na tyle duży, by móc w nim przechowywać sparsowane wersje najczęściej używanych zapytań. Gdy obszary pamięci należące do SGA mają odpowiednie wymiary, mogą one zdecydowanie zwiększyć wydajność pojedynczych zapytań, a także bazy danych jako całości.

Rozmiar przydzielony buforom KEEP i RECYCLE nie zmniejsza ilości miejsca dostępnego w buforze pamięci podręcznej bloku danych. Aby nakazać tabeli używanie jednego z nowych obszarów bufora, należy wskazać nazwę obszaru bufora parametrem `buffer_pool` w klauzuli `storage` dla tabeli. Jeśli tabela ma być na przykład szybko usuwana z pamięci, należy przypisać ją do obszaru RECYCLE. Obszar domyślny nosi nazwę DEFAULT, zatem w późniejszym czasie można wykonać polecenie `alter table` i z powrotem skierować tabelę do obszaru DEFAULT. Oto przykład przypisania tabeli do obszaru bufora KEEP:

```
create table state_cd_lookup
 (state_cd   char(2),
  state_nm   varchar2(50)
 )
storage (buffer_pool keep);
```

Za pomocą parametru inicjalizacyjnego LARGE_POOL_SIZE można zdefiniować wyrażany w bajtach rozmiar sterty alokacji obszaru dużych struktur pamięciowych. Sterta alokacji obszaru dużych struktur pamięciowych jest używana przez systemy współużytkujące serwery jako pamięć sesji, w trakcie równoległego wykonywania operacji jako bufory komunikatów oraz przez procesy tworzenia kopii zapasowych jako bufory operacji wejścia-wyjścia. Domyślnie obszar dużych struktur pamięciowych nie jest tworzony.

Począwszy od wersji Oracle 10*g*, można używać narzędzia Automatic Shared Memory Management (ASMM). Aby włączyć narzędzie ASMM, należy przypisać niezerową wartość parametrowi inicjalizacyjnemu SGA_TARGET bazy danych. Gdy parametrowi SGA_TARGET przypisany zostanie pożądany rozmiar obszaru SGA (czyli łączny rozmiar wszystkich obszarów pamięci podręcznej), można następnie przypisać pozostałym parametrom związanym z pamięcią podręczną (DB_CACHE_SIZE, SHARED_POOL_SIZE, JAVA_POOL_SIZE i LARGE_POOL_SIZE) wartość równą zero. Jeśli wymienionym parametrom przypisane zostaną wartości różne od zera, będą one traktowane przez algorytm automatycznego strojenia jako minimalny rozmiar każdego obszaru. Aby wprowadzone zmiany zostały uwzględnione, należy wyłączyć i zrestartować bazę danych. Od tego momentu baza danych zacznie aktywnie zarządzać rozmiarami poszczególnych pamięci podręcznych. Przez cały czas można monitorować rozmiary pamięci podręcznych, korzystając z dynamicznego widoku wydajności V$GASTAT. System Oracle Database 11*g* jeszcze bardziej zwiększa poziom automatyzacji. W przypadku parametru MEMORY_TARGET można ustawić całkowitą ilość pamięci dostępnej dla systemu Oracle. Ilość pamięci określana przez ten parametr jest automatycznie alokowana między obszarami SGA i PGA. Po skonfigurowaniu parametru MEMORY_TARGET dla parametrów SGA_TARGET i PGA_AGGREGATE_TARGET ustawiana jest wartość 0 i jest ona ignorowana.

Ze względu na zmieniające się obciążenie bazy danych baza będzie zmieniać rozmiary pamięci podręcznych, aby odzwierciedlić potrzeby aplikacji. Na przykład jeśli w godzinach nocnych wykonywane są wsadowe operacje ładowania danych mocno obciążające bazę, a w trakcie dnia intensywnie wykonuje się transakcje online, baza danych może zmieniać rozmiary pamięci podręcznych zgodnie z bieżącym obciążeniem. Zmiany te są wprowadzane automatycznie, bez konieczności angażowania administratora bazy danych. Jeżeli dla danego obszaru w pliku parametrów inicjalizacyjnych wskazana zostanie konkretna wartość, Oracle potraktuje tę wartość jako minimalny rozmiar obszaru.


Uwaga

Administratorzy baz danych mogą tworzyć obszary KEEP i RECYCLE w buforze pamięci podręcznej. Dynamiczne zmiany rozmiaru pamięci podręcznej nie mają wpływu na obszary KEEP i RECYCLE, ponieważ nie stanowią one części obszaru bufora DEFAULT.

W narzędziu OEM można sprawdzić, czy włączone jest dynamiczne zarządzanie pamięcią. W tym celu trzeba kliknąć myszą opcję *Memory Parameters*, po czym przycisk *Automatic Shared Memory Management* ustawić na *Enabled* (włączone) lub *Disabled* (wyłączone).

Wybrane pakiety można „przypinać" do obszaru wspólnego. Przypięcie pakietów w pamięci natychmiast po uruchomieniu bazy danych zwiększy prawdopodobieństwo, że dostępny stanie się odpowiednio duży ciągły obszar wolnej przestrzeni w pamięci. W poniższym poleceniu procedura KEEP pakietu DBMS_SHARED_POOL nakazuje przypięcie pakietów w obszarze wspólnym:

```
execute DBMS_SHARED_POOL.KEEP('APPOWNER.ADD_CLIENT','P');
```

Przypinanie pakietów bardziej wiąże się z zarządzaniem aplikacją niż z jej strojeniem, lecz może również wpływać na wydajność aplikacji. Jeśli uda się uniknąć dynamicznego zarządzania pofragmentowanymi obszarami pamięci, zminimalizuje się w ten sposób zakres działań, jakie musi wykonywać serwer Oracle w ramach zarządzania obszarem wspólnym.

Definiowanie rozmiaru SGA

Aby włączyć automatyczne zarządzanie obszarami pamięci podręcznej, należy przypisać parametrowi inicjalizacyjnemu SGA_TARGET rozmiar obszaru SGA.

Jeżeli zapadnie decyzja o ręcznym zarządzaniu obszarami pamięci podręcznej, można parametrowi SGA_MAX_SIZE przypisać rozmiar obszaru SGA. Następnie można zdefiniować rozmiary poszczególnych pamięci podręcznych; rozmiary te mogą być dynamicznie zmieniane w późniejszym czasie, w trakcie działania bazy danych, za pomocą polecenia alter system. W poniższej tabeli opisano parametry wyznaczające rozmiary pamięci podręcznych.

Tabela 8.1. *Parametry rozmiarów pamięci podręcznych*

Parametr	Opis
SGA_MAX_SIZE	Maksymalny rozmiar, jaki może przybrać obszar SGA.
SHARED_POOL_SIZE	Rozmiar obszaru wspólnego.
DB_BLOCK_SIZE	Domyślny rozmiar bloku bazy danych.
DB_CACHE_SIZE	Rozmiar pamięci podręcznej wyrażony w bajtach.
DB_*n*K_CACHE_SIZE	Jeśli w jednej bazie danych używane będą bloki bazy o różnych rozmiarach, należy zdefiniować wartość parametru DB_CACHE_SIZE oraz wartość co najmniej jednego parametru DB_*n*K_CACHE_SIZE. Na przykład jeżeli standardowy rozmiar bloku bazy danych wynosi 4 KB, można także utworzyć pamięć podręczną dla przestrzeni tabel z blokami o rozmiarze 8 KB — służy do tego parametr DB_8K_CACHE_SIZE.

Możliwe jest również ustawienie dla parametru SGA_TARGET rozmiaru mniejszego niż w przypadku parametru SGA_MAX_SIZE. System Oracle użyje parametru SGA_TARGET do początkowej konfiguracji poszczególnych buforów. Z czasem system może przydzielać buforom więcej pamięci, aż do osiągnięcia maksymalnej wartości parametru SGA_MAX_SIZE. Jest to dobra metoda określania całkowitej wymaganej pamięci, jaka powinna być dostępna przed wdrożeniem bazy danych w środowisku produkcyjnym.

Można na przykład zdefiniować następujące wartości parametrów:

```
SGA_MAX_SIZE=1024M
SHARED_POOL_SIZE=220M
DB_BLOCK_SIZE=8192
DB_CACHE_SIZE=320M
DB_4K_BLOCK_SIZE=4M
```

Dzięki tak zdefiniowanym parametrom dla danych odczytywanych przez zapytania, pochodzących z obiektów z przestrzeni tabel o rozmiarze bloku 4 KB, dostępne będą 4 MB pamięci. Obiekty używające standardowego rozmiaru bloku (8 KB) będą mogły używać do 160 MB pamięci podręcznej. Gdy baza danych jest włączona, wartości parametrów SHARED_POOL_SIZE i DB_CACHE_SIZE można zmieniać przy użyciu polecenia alter system.

SGA_TARGET to parametr dynamiczny, którego wartość można zmieniać przy użyciu narzędzia Database Control lub poleceniem alter system.

Wartość SGA_TARGET może rosnąć nawet do wartości posiadanej przez parametr SGA_MAX_SIZE. Wartość tę można zmniejszać aż do osiągnięcia przez któryś z komponentów dostrajanych automatycznie jego rozmiaru minimalnego, zdefiniowanego przez użytkownika albo ustalonego wewnętrznie przez bazę danych. Za pomocą obydwóch wspomnianych parametrów można dostrajać obszar SGA.

Wykorzystanie optymalizatora kosztowego

W każdej kolejnej wersji bazy danych Oracle dodawał nowe funkcje optymalizatora oraz rozwijał funkcje już wcześniej w nim obecne. Aby móc efektywnie korzystać z optymalizatora kosztowego, należy zapewnić regularne wykonywanie analizy tabel i indeksów używanych przez aplikację. Częstotliwość, z jaką powinno się analizować obiekty, zależy od częstości zachodzących w nich zmian. W przypadku aplikacji realizujących transakcje wsadowe obiekty powinny być na nowo analizowane po wykonaniu każdego dużego zbioru transakcji. W przypadku aplikacji OLTP obiekty powinno się analizować cyklicznie (na przykład co tydzień albo co noc).


Uwaga

Począwszy od systemu Oracle Database 10*g* Release 1, nie jest obsługiwany optymalizator bazujący na regułach.

Statystyki dotyczące obiektów są gromadzone przy użyciu procedur pakietu DBMS_STATS. W trakcie analizowania tabeli analizie podlegają również powiązane z nią indeksy. Analizować można schemat (służy do tego procedura GATHER_SCHEMA_STATS) albo konkretną tabelę (przy użyciu procedury GATHER_TABLE_STATS). Można także wykonywać analizy wyłącznie indeksowanych kolumn, co przyspieszy cały proces analiz. Ogólna zasada głosi, że indeksy tabel należy analizować za każdym razem, gdy analizuje się tabelę. Poniższe polecenie wykonuje analizę schematu PRACTICE:

```
execute DBMS_STATS.GATHER_SCHEMA_STATS('PRACTICE', 'COMPUTE');
```

Do przeglądania statystyk dotyczących tabel i indeksów służą tabele DBA_TABLES, DBA_TAB_COL_
↪STATISTICS oraz DBA_INDEXES. Niektóre statystyki dotyczące kolumn są również dostępne w tabeli DBA_TAB_COLUMNS, lecz ich obecność tam wynika jedynie z konieczności zapewnienia zgodności wstecz. Statystyki dotyczące kolumn tabel partycjonowanych znajdują się w tabeli DBA_PART_COL_STATISTICS.


Uwaga

Począwszy od systemu Oracle Database 10*g*, w domyślnej instalacji statystyki są automatycznie gromadzone w okresach konserwacji z wykorzystaniem infrastruktury zautomatyzowanych zadań konserwacyjnych (AutoTask).

W wyniku uruchomienia przedstawionego przed chwilą polecenia przeprowadzona zostanie analiza wszystkich obiektów należących do schematu PRACTICE, do czego użyta zostanie opcja compute statistics. Można także zdecydować o oszacowaniu statystyk na podstawie jedynie wskazanego procentu wierszy w tabeli.

Skutki działania opcji compute statistics

W przykładach zaprezentowanych w poprzednim punkcie do gromadzenia statystyk na temat obiektów używano opcji `compute statistics`. Oracle udostępnia jednak również opcję `estimate statistics`, dzięki której statystyki obiektu są szacowane na podstawie tylko części danych. Jeśli wybrana zostanie opcja `estimate statistics`, najlepiej jest przeprowadzać analizę na jak największej możliwej części danych. Opcja ta pozwala na wskazanie procentowej liczby wierszy, jakie mają być analizowane; zwykle wystarczającą wartością jest 20 procent.

> Polecenie `analyze table . . . compute statistics` lub `analyze table . . . estimate statistics` może przestać być dostępne poza pakietem `DBMS_STATS` w przyszłych wersjach systemu Oracle. Polecenia `analyze` należy używać w przypadku zadań niemających związku ze statystykami, takich jak realizowane za pomocą poleceń `validate structure` lub `list chained rows`, bądź do gromadzenia informacji dotyczących listy wolnych bloków.

W trakcie analizowania danych konieczne może być zapewnienie znacznej ilości miejsca dla operacji sortowania. Ponieważ w trakcie analiz wykonywane mogą być również pełne skany tabel, tuż przed rozpoczęciem analiz powinno się odpowiednio zmienić ustawienia sesji. Z kolei po zakończeniu analizy należy zakończyć sesję lub zmienić jej ustawienia na takie, jakie obowiązywały przed analizą. Ustawieniami sesji, które powinny się zmienić, są parametry `SORT_AREA_SIZE` oraz `DB_FILE_MULTIBLOCK_READ_COUNT`. Począwszy od systemu Oracle Database 10*g*, firma Oracle szczególnie namawia do używania parametru `PGA_AGGREGATE_TARGET` do automatycznego zarządzania wartością parametru `SORT_AREA_SIZE`. Im większy jest obszar sortowania, tym mniejsze jest prawdopodobieństwo, że dla segmentów sortowania konieczne będzie użycie tymczasowej przestrzeni tabel. Im wyższa jest liczba operacji odczytów wielu bloków, tym więcej bloków można odczytywać w trakcie pojedynczej operacji odczytu fizycznego (liczba ta będzie ograniczona przez system operacyjny). Wspomniane ustawienia sesji można zmienić przy użyciu polecenia `alter session`.

Strojenie dostępu do danych

Nawet gdy tabele są odpowiednio skonfigurowane i poindeksowane, wydajność nadal może być niesatysfakcjonująca, jeśli żądania dostępu do plików powodują przestoje. W następnych punktach przedstawione zostaną zalecenia dotyczące konfigurowania plików i przestrzeni tabel.

Ogólnie mówiąc, należy unikać umieszczania plików bazy Oracle w systemach RAID z rozdzielanym zapisem parzystości, takich jak RAID 5. Zapisywanie danych w tego rodzaju systemach plików staje się wraz ze wzrostem liczby użytkowników coraz uciążliwszym wąskim gardłem — dotyczy to zwłaszcza plików zapisywanych sekwencyjnie, takich jak choćby pliki dziennika zdarzeń online. Do tworzenia kopii lustrzanych i paskowania danych lepiej jest wykorzystywać rozwiązania RAID0+1 i zapobiec tym samym powstawaniu wąskich gardeł.

Przestrzenie tabel zarządzane lokalnie

Dzięki użyciu przestrzeni tabel zarządzanych lokalnie proces zarządzania obszarami jest realizowany wewnątrz przestrzeni tabel. Przestrzenie tabel zarządzane lokalnie zarządzają należącą do nich przestrzenią dyskową za pomocą mapy bitowej utrzymywanej w każdym z plików danych. Mapa bitowa zawiera informacje o wolnych i używanych blokach lub zbiorach bloków znajdujących się w pliku danych. Za każdym razem, gdy obszar zostaje zaalokowany lub zwolniony w celu ponownego użycia, baza danych uaktualnia mapę bitową, aby odzwierciedlić nową sytuację.


Uwaga

Począwszy od systemu Oracle Database 10*g*, wszystkie przestrzenie tabel domyślnej instalacji są zarządzane lokalnie. Wielkoplikowe przestrzenie tabel **muszą** być zarządzane lokalnie. Z przestrzeni tabel zarządzanych przez słownik danych należy korzystać tylko w celu zachowania zgodności z poprzednimi wersjami systemu Oracle.

Gdy używane są przestrzenie tabel zarządzane lokalnie, wówczas w trakcie tworzenia obszarów nie są uaktualniane dane słownikowe ani nie dochodzi do operacji wycofania. Przestrzenie tabel zarządzane lokalnie automatycznie śledzą przylegające do siebie fragmenty wolnej przestrzeni, dzięki czemu nie trzeba wykonywać dodatkowych operacji scalania obszarów. W przestrzeni tabel zarządzanej lokalnie wszystkie obszary mogą mieć taki sam rozmiar lub system może samodzielnie, automatycznie ustalić rozmiar obszarów.

Aby skorzystać z dobrodziejstw lokalnego zarządzania przestrzenią, w klauzuli `extent management` polecenia `create tablespace` można użyć opcji `local`. Poniżej zaprezentowano przykładowe polecenie `create tablespace`, które deklaruje przestrzeń tabel zarządzaną lokalnie:

```
create tablespace CODES_TABLES
datafile '/u01/oracle/VLDB/codes_tables.dbf'
size 500M
extent management local uniform size 256K;
```

Jeśli założymy, że bloki bazy danych, w której przestrzeń tabel utworzono, mają rozmiar 8 KB, wówczas w utworzonej przestrzeni tabel zarządzanie obszarami będzie wykonywanie lokalnie i każdy obszar będzie mieć taki sam rozmiar 256 KB. Każdy bit mapy bitowej opisuje 32 bloki (256 podzielone przez 8). Jeśli klauzula uniform `size` zostanie pominięta, użyta zostanie domyślna klauzula `autoallocate`. Domyślnym rozmiarem w przypadku zastosowania klauzuli uniform jest 1 MB.


Uwaga

Jeśli w poleceniu `create tablespace` użyta zostanie klauzula `local`, nie będzie można w nim użyć klauzuli `default storage`, `minextents` ani `temporary`. Jeżeli do utworzenia przestrzeni tabel użyte zostanie polecenie `create temporary tablespace`, można użyć klauzuli `extent management local`.

Począwszy od wersji Oracle9*i*, przestrzenie tabel są domyślnie zarządzane lokalnie, przez co klauzula `extent management local` ma charakter opcjonalny (podczas tworzenia nowej przestrzeni tabel).


Uwaga

Jeżeli przestrzeń tabel SYSTEM zostanie skonfigurowana jako przestrzeń zarządzana lokalnie, w bazie danych będzie można tworzyć wyłącznie przestrzenie tabel zarządzane lokalnie. Wszelkie przestrzenie tabel zarządzane przez słownik danych, które zaimportowano za pomocą funkcji przenośnych przestrzeni tabel, mogą być otwarte tylko w trybie do odczytu.

Identyfikowanie łańcuchów wierszy

W momencie tworzenia segmentu danych wskazywana jest wartość pctfree. Parametr pctfree wskazuje bazie danych ilość miejsca, jaka w każdym bloku danych powinna pozostać wolna. Wolna przestrzeń zostanie użyta, gdy długość wierszy, które znajdują się już w bloku danych, przekroczy granice bloku w wyniku wykonania operacji update.

Jeśli wykonanie operacji update na wierszu spowoduje, że nie będzie się on już wpasowywał w pojedynczy blok danych, wiersz ten może zostać przeniesiony do innego bloku danych lub może dojść do utworzenia łańcucha rozciągającego się na następny blok. Jeśli zapisywane są wiersze, których długość przekracza rozmiar bloku bazy danych Oracle, automatycznie tworzone będą łańcuchy wierszy.

Istnienie łańcuchów wpływa na wydajność bazy, ponieważ Oracle musi wówczas szukać danych pochodzących z tego samego wiersza logicznego w wielu fizycznych lokalizacjach. Wyeliminowanie niepotrzebnego powstawania łańcuchów wierszy zmniejszy liczbę fizycznych operacji odczytu, jakie trzeba wykonać, aby zwrócone zostały dane z pliku danych.

Tworzenia łańcuchów wierszy można uniknąć przez zdefiniowanie właściwej wartości parametru pctfree w momencie tworzenia segmentów danych. Domyślna wartość 10 powinna zostać zwiększona, jeśli tworzona aplikacja będzie często zamieniać wartości NULL na wartości różne od NULL albo gdy często będzie zmieniać długie wartości tekstowe.

Za pomocą polecenia analyze można zbierać statystyki na temat obiektów bazy danych. Na podstawie tych statystyk optymalizator kosztowy może ustalać najlepszą ścieżkę wykonania, jakiej należy użyć. Dla polecenia analyze dostępna jest opcja, która powoduje wykrycie i zapisanie w tabeli łańcuchów wierszy. Składnia polecenia prezentuje się następująco:

```
analyze table NAZWA_TABELI list chained rows into CHAINED_ROWS;
```

Polecenie analyze umieści dane wynikowe w tabeli o nazwie CHAINED_ROWS znajdującej się w schemacie lokalnym. Kod języka SQL odpowiedzialny za utworzenie tabeli CHAINED_ROWS znajduje się w pliku o nazwie *utlchain.sql*, w podkatalogu *$ORACLE_HOME/rdbms/admin*. Poniższe zapytanie odczytuje zawartość najważniejszych kolumn tabeli CHAINED_ROWS:

```
select
      Owner_Name,       /*Właściciel segmentu danych*/
      Table_Name,       /*Nazwa tabeli z łańcuchami wierszy*/
      Cluster_Name,     /*Nazwa klastra, jeśli występują klastry*/
      Head_RowID        /*Rowid pierwszej części wiersza*/
from CHAINED_ROWS;
```

Dane wynikowe zapytania będą zawierać identyfikatory RowID wszystkich wierszy powiązanych łańcuchowo, dzięki czemu od razu będzie widać, ile wierszy w tabeli ma postać łańcuchów. Jeżeli w tabeli bardzo często powstają łańcuchy wierszy, tabelę taką należy utworzyć na nowo, wskazując dla niej wyższą wartość parametru pctfree.

Efekt powstawania łańcuchów wierszy można zaobserwować, wykonując odpowiednie zapytanie na widoku V$SYSSTAT. Za każdym razem, gdy Oracle odczytuje dane z wiersza mającego postać łańcucha, zwiększa się wartość statystyki table fetch continued row w widoku V$SYSSTAT. Wartość statystyki będzie się zwiększać również wtedy, gdy Oracle będzie odczytywać dane z **wiersza rozciągniętego** (ang. *spanned row*), czyli z wiersza, który przekształcił się w łańcuch, ponieważ jego długość przekroczyła rozmiar bloku. Wiersze rozciągnięte najczęściej występują w tabelach z typami danych LONG, BLOB, CLOB i NCLOB. Statystyki table fetch continued row są też dostępne w raportach AWR (lub w raportach STATSPACK w przypadku systemu Oracle Database 10*g* i jego poprzedników).

Oprócz tworzenia łańcuchów wierszy Oracle od czasu do czasu przenosi wiersze. Gdy rozmiar wiersza przekracza przestrzeń dostępną w bloku, wiersz może zostać wstawiony do innego bloku. Proces przenoszenia wiersza z jednego bloku do drugiego to tak zwana **migracja wiersza** (ang. *row migration*), a przeniesiony wiersz to tak zwany **wiersz zmigrowany** (ang. *migrated row*). W trakcie migracji wierszy Oracle musi dynamicznie zarządzać przestrzenią w wielu blokach i odczytywać listę wolnych bloków (listę bloków dostępnych dla operacji insert). Wiersz zmigrowany nie przybiera postaci łańcucha, lecz wpływa na wydajność transakcji. W rozdziale 6. zamieszczono przykład zastosowania parametru DBMS_ADVISOR do wyszukiwania i reorganizowania tabel z łańcuchami wierszy.

Uzyskanie dostępu do przeniesionego wiersza powoduje zwiększenie wartości licznika w statystykach table fetch continued row.

Zwiększanie rozmiaru bloków bazy Oracle

Efekt zwiększenia rozmiaru bloku bazy danych może być bardzo duży. Podwojenie rozmiaru bloku bazy danych może zwiększyć wydajność operacji wykonujących zapytania mocno obciążające bazę nawet o 50 procent.

Korzyść wynikająca ze zwiększenia wydajności jest osiągana stosunkowo niewielkim kosztem. Ponieważ wzrośnie liczba wierszy w pojedynczym bloku bazy danych, zwiększy się również prawdopodobieństwo wystąpienia konfliktów na poziomie dostępu do bloku w trakcie poleceń wykonujących operacje na danych. Aby uniknąć problemów wiążących się z występowaniem konfliktów, należy zwiększyć wartości parametrów freelists i initrans na poziomie tabeli i indeksów. Generalnie przypisanie parametrowi freelists wartości większej niż 4 nie przyniesie jakiejś znaczącej dodatkowej korzyści. Wartość parametru initrans powinna odzwierciedlać oczekiwaną liczbę transakcji współbieżnych w bloku.

Wartość 4 jest odpowiednia dla parametru initrans w przypadku aplikacji OLTP cechujących się dużą aktywnością poleceń DML. Zwiększenie wartości tego parametru dla aplikacji wykorzystujących hurtownie danych nie spowoduje wzrostu wydajności. Warto też zauważyć, że listy wolnych bloków stosowane są tylko dla obiektów przestrzeni tabel bez używanego mechanizmu ASSM.

Aktualnie Oracle automatycznie pozwala na wykonywanie w każdym bloku danych do 255 współbieżnych transakcji, zależnie od ilości miejsca dostępnego w danym bloku.

Gdy tworzona jest przestrzeń tabel, można w tym samym momencie wskazać rozmiar bloku bazy danych obowiązujący w przestrzeni tabel. Domyślnie w przestrzeni tabel jako rozmiar bloku bazy danych używany jest rozmiar przypisany parametrowi inicjalizacyjnemu `DB_BLOCK_SIZE`. Jeśli w przestrzeni tabel używany będzie niestandardowy rozmiar bloku bazy danych, konieczne będzie utworzenie pamięci podręcznej specjalnie dla bloków o tym rozmiarze. Jeśli bloki bazy danych mają 8 KB i konieczne jest utworzenie przestrzeni tabel, w której obowiązywać będą bloki bazy danych o rozmiarze 4 KB, wówczas najpierw należy przypisać odpowiednią wartość parametrowi `DB_4K_CACHE_SIZE`.

Aby zwiększyć rozmiar bloków w całej bazie danych, trzeba najpierw na nowo utworzyć całą bazę i usunąć jej wszystkie dotychczasowe pliki. Nowe pliki można utworzyć w tej samej lokalizacji co pliki dotychczasowe i mogą one mieć taki sam rozmiar, jak dotychczas, lecz od teraz baza danych będzie nimi zarządzać bardziej wydajnie. Zwiększenie wydajności wynika ze sposobu, w jaki Oracle zarządza informacjami w nagłówkach bloków. Więcej dostępnej przestrzeni jest przeznaczane na przechowywanie danych, dzięki czemu zwiększa się stopień dostępności tych samych bloków danych przechowywanych w pamięci dla większej liczby użytkowników. Podwojenie rozmiaru bloków bazy Oracle nie ma większego wpływu na nagłówki bloków, dzięki czemu dane w nagłówkach bloków zajmują relatywnie mniej miejsca.

Aby zdefiniować rozmiar bloków, przed utworzeniem nowej bazy danych należy zmodyfikować wartość parametru inicjalizacyjnego `DB_BLOCK_SIZE`.

Używanie tabel o strukturze indeksu

Tabela o strukturze indeksu (ang. *index-organized table — IOT*) to indeks, w którym przechowywany jest cały wiersz, a nie tylko wartości kluczowe wiersza. Zamiast identyfikatora wiersza `RowID` kluczem głównym dla wiersza jest jego logiczny identyfikator. Wiersze znajdujące się w tabeli o strukturze indeksu nie posiadają identyfikatorów `RowID`.

W tabeli o strukturze indeksu przechowywane wiersze są posortowane względem wartości ich kluczy głównych. Zatem wydajność dowolnego zapytania o zakres wartości opartego na kluczu głównym może wzrosnąć, ponieważ wiersze są przechowywane jeden obok drugiego (więcej informacji na temat układania danych w zwykłej tabeli przedstawiono w punkcie „Strojenie kodu SQL", we wcześniejszej części rozdziału). Dodatkowo zwiększyć się może również wydajność zapytań równowartościowych opartych na kluczu głównym, ponieważ cały zbiór danych tabeli jest przechowywany w indeksie. W tradycyjnym układzie tabeli i indeksu uzyskanie dostępu z wykorzystaniem indeksu wymaga najpierw uzyskania indeksu, a dopiero potem uzyskania dostępu do tabeli. Natomiast w przypadku tabeli o strukturze indeksu odczyt jest wykonywany tylko w tej tabeli i nie towarzyszy jej żaden dodatkowy indeks.

Jednak trzeba też pamiętać, że poprawa wydajności wynikająca z konieczności odczytywania danych z jednego indeksu zamiast ze standardowego układu tabela-indeks może być niemal niezauważalna, ponieważ każda operacja odczytu przeprowadzana na podstawie indeksu powinna przebiegać szybko. Aby jeszcze bardziej zwiększyć wydajność, w tabelach o strukturze indeksu zawarto następujące dodatkowe funkcje:

- **Obszar przepełnienia** — w momencie tworzenia tabeli o strukturze indeksu można zdefiniować wartość parametru `pctthreshold`, dzięki czemu dane klucza głównego będą mogły być przechowywane oddzielnie od danych wiersza. Jeśli rozmiar danych wiersza przekroczy wartość parametru wyznaczającego ilość miejsca dostępnego

w bloku, wiersz ten zostanie dynamicznie przeniesiony do obszaru przepełnienia. Obszar przepełnienia można utworzyć w oddzielnej przestrzeni tabel, dzięki czemu zwiększą się możliwości rozpraszania operacji wejścia-wyjścia na tabeli.

- **Indeksy drugorzędne** — w tabeli o strukturze indeksu można tworzyć indeksy drugorzędne. Wartości kluczy głównych zostaną zastosowane przez bazę Oracle jako identyfikatory `ROWID` wierszy.
- **Obniżone wymagania względem ilości przestrzeni dyskowej** — w tradycyjnym układzie tabela-indeks te same wartości kluczy muszą być przechowywane w dwóch miejscach. W tabeli o strukturze indeksu wartości są przechowywane w jednym miejscu, co zmniejsza wymagania dotyczące wymaganej przestrzeni.


Wskazówka

Podczas określania obszaru przepełnienia można za pomocą klauzuli `including column` wybrać kolumnę (oraz wszystkie kolejne kolumny w definicji tabeli), która będzie przechowywana w tym obszarze.

```
create table ord_iot
   (order_id number,
    order_date date,
    order_notes varchar2(1000), primary key(order_id,order_date))
     organization index including order_date
     overflow tablespace over_ord_tab
     PARTITION BY RANGE (order_date)
      (PARTITION p1 VALUES LESS THAN ('01-JAN-2005')
       TABLESPACE data01,
       PARTITION p2 VALUES LESS THAN (MAXVALUE)
      TABLESPACE data02);
```

Zarówno kolumna `order_date`, jak i `order_notes` będą przechowywane w obszarze przepełnienia.

Aby utworzyć tabelę o strukturze indeksu, należy użyć klauzuli `organization index` polecenia `create table`. W momencie tworzenia tabeli o strukturze indeksu trzeba wskazać klucz główny. Z tabeli o strukturze indeksu można usuwać kolumny lub oznaczać je jako nieaktywne przy użyciu klauzuli `set unused` polecenia `alter table`.

Strojenie tabel o strukturze indeksu

Podobnie jak indeksy, tabele o strukturze indeksu mogą z biegiem czasu i wraz z kolejnymi operacjami `insert`, `update` i `delete` ulec wewnętrznej fragmentacji. Aby odbudować tabelę o strukturze indeksu, należy użyć klauzuli `move` polecenia `alter table`. W poniższym przykładzie wykonana zostanie przebudowa tabeli `EMPLOYEE_IOT` wraz z jej obszarem przepełnienia:

```
alter table EMPLOYEE_IOT
move tablespace DATA
overflow tablespace DATA_OVERFLOW;
```

Należy unikać przechowywania długich wierszy w tabeli o strukturze indeksu. Generalnie powinno się unikać używania tabeli o strukturze indeksu w przypadku, gdy rozmiar danych przekracza 75 procent rozmiar bloku danych. Jeśli blok bazy danych ma rozmiar 4 KB, a długość wierszy będzie przekraczać 3 KB, należy rozważyć użycie standardowych tabel

i indeksów zamiast tabel o strukturze indeksu. Im wiersze są dłuższe i im większa jest liczba transakcji wykonywanych na tabeli o strukturze indeksu, tym częściej trzeba będzie taką tabelę odbudowywać.


Uwaga

> W tabelach o strukturze indeksu nie można używać typów danych LONG, można natomiast używać typów LOB.

Jak wspomniano już we wcześniejszej części rozdziału, obecność indeksów wpływa na wydajność ładowania danych. Aby osiągnąć najlepsze wyniki, indeks klucza głównego tabeli o strukturze indeksu powinien być ładowany wartościami sekwencyjnymi, aby zminimalizować w ten sposób koszty zarządzania indeksem.

Strojenie operacji manipulowania danymi

Niektóre operacje manipulowania danymi — zwłaszcza operacje polegające na manipulowaniu dużymi ilościami danych — mogą wymagać zaangażowania administratora bazy danych. Dostępnych jest kilka opcji ładowania i usuwania znacznych ilości danych. Więcej informacji na temat tych opcji znajduje się w następnych punktach.

Operacje zbiorczego ładowania danych — użycie opcji Direct Path narzędzia SQL*Loader

Gdy SQL*Loader jest używany w trybie Conventional Path, narzędzie odczytuje rekordy z pliku, generuje polecenia insert i przekazuje je do jądra bazy Oracle. Następnie Oracle znajduje miejsca dla tych rekordów w wolnych blokach tabeli i uaktualnia wszystkie powiązane z nią indeksy.

W trybie Direct Path SQL*Loader tworzy sformatowane bloki danych i zapisuje dane bezpośrednio do plików danych. Rozwiązanie to wymaga odwołania się co jakiś czas do bazy danych, aby odczytać nowe lokalizacje bloków danych. Jednak oprócz tego żadne inne operacje wejścia-wyjścia na jądrze bazy danych nie są wymagane. W efekcie otrzymujemy proces ładowania danych, który jest zdecydowanie szybszy niż w trybie Conventional Path.

Jeśli na tabeli utworzono indeksy, wówczas w trakcie ładowania danych indeksy te zostaną przełączone w stan DIRECT PATH. Po zakończeniu operacji ładowania nowe klucze (wartości kolumny indeksowanej) zostaną posortowane i złączone z kluczami już istniejącymi w indeksie. Aby utrzymywać tymczasowy zestaw kluczy, dla celów operacji ładowania zostanie utworzony tymczasowy segment indeksu, którego rozmiar będzie co najmniej równy rozmiarowi największego indeksu w tabeli. Ilość przestrzeni wymaganej dla indeksu można zminimalizować przez uprzednie wstępne posortowanie danych i użycie klauzuli SORTED INDEXES w pliku kontrolnym narzędzia SQL*Loader.

Aby zminimalizować obciążenie wynikające z dynamicznego alokowania przestrzeni niezbędnego w trakcie operacji ładowania, należy wcześniej utworzyć segment danych, do którego dane będą ładowane, oraz zaalokować dla niego całą wymaganą przestrzeń. Powinno się

również wstępnie posortować dane w kolumnach największego indeksu w tabeli. Posortowanie danych i pozostawienie indeksów na tabeli na czas ładowania danych w trybie Direct Load zwykle powoduje, że wydajność operacji jest wyższa niż w przypadku, gdyby przed ładowaniem danych usunięto indeksy, a następnie odtworzono je po zakończeniu ładowania.

Aby móc skorzystać z opcji Direct Path, tabela nie może być tabelą klastrową, a ponadto nie mogą w niej występować żadne aktywne transakcje. W trakcie ładowania danych przestrzegane będą jedynie ograniczenia `NOT NULL`, `UNIQUE` oraz `PRIMARY KEY`. Natomiast po zakończeniu operacji ładowania można automatycznie przywrócić ograniczenia `CHECK` i `FOREIGN KEY`. Aby zapewnić automatyczne przywrócenie obydwóch ograniczeń, należy użyć w pliku kontrolnym narzędzia SQL*Loader następującej klauzuli:

```
REENABLE DISABLED_CONSTRAINTS
```

W przypadku ponownego włączenia ograniczeń jedynym wyjątkiem od spodziewanego efektu jest to, że ponowne włączenie procedur wyzwalanych w trakcie operacji wstawienia danych nie spowoduje wykonania tych procedur na nowych wierszach w tabeli. Aby wykonać wszystkie polecenia, które powinny były zostać wykonane w ramach procedur wyzwalanych, należy je uruchomić ręcznie.

Opcja Direct Path ładowania danych narzędziem SQL*Loader znacznie zwiększa wydajność ładowania danych do tabel bazy Oracle w porównaniu z działaniem opcji Conventional Path, ponieważ dzięki niej omija się konieczność przetwarzania kodu SQL, działania związane z zarządzaniem buforem pamięci podręcznej oraz niepotrzebne operacje odczytu bloków danych. Opcja Parallel Data Loading narzędzia SQL*Loader pozwala na ładowanie danych do tej samej tabeli jednocześnie przez wiele procesów. W ten sposób używane są dostępne zasoby systemowe oraz następuje redukcja spodziewanego całkowitego czasu ładowania. Jeśli tylko dostępna jest wystarczająca ilość zasobów procesora i wejścia-wyjścia, zastosowanie opcji Parallel Data Loading może znacząco zmniejszyć całkowite czasy ładowania danych.

Aby przeprowadzić ładowanie w trybie Parallel Data Loading, należy uruchomić kilka sesji narzędzia SQL*Loader przy użyciu słowa kluczowego `parallel` (w przeciwnym razie SQL*Loader założy na tabeli blokadę na wyłączność). Każda sesja funkcjonuje niezależnie od pozostałych i wymaga własnego pliku kontrolnego. Poniższy kod wywołuje trzy oddzielne operacje ładowania w trybie Direct Path i w każdej z nich w wierszu polecenia używany jest parametr `PARALLEL=TRUE`:

```
sqlload USERID=ME/PASS CONTROL=PART1.CTL DIRECT=TRUE PARALLEL=TRUE
sqlload USERID=ME/PASS CONTROL=PART2.CTL DIRECT=TRUE PARALLEL=TRUE
sqlload USERID=ME/PASS CONTROL=PART3.CTL DIRECT=TRUE PARALLEL=TRUE
```

Każda sesja domyślnie tworzy swoje własne pliki dziennika, błędów i danych odrzuconych (*part1.log*, *part2.log*, *part3.log*, *part1.bad*, *part2.bad* itd.). Ponieważ do tej samej tabeli dane ładuje wiele sesji jednocześnie, w operacji ładowania danych Parallel Data Loading dozwolone jest jedynie używanie opcji `APPEND`. W trybie Parallel Data Loading nie są natomiast dostępne opcje narzędzia SQL*Loader `REPLACE`, `TRUNCATE` i `INSERT`. Jeżeli przed rozpoczęciem operacji ładowania danych z tabeli trzeba usunąć wszystkie dane, trzeba je usunąć ręcznie (poleceniami `delete` lub `truncate`). Jeśli używany jest tryb Parallel Data Loading, nie można użyć narzędzia SQL*Loader do automatycznego usunięcia rekordów.


Uwaga

> Gdy używany jest tryb Parallel Data Loading, sesja narzędzia SQL*Loader nie utrzymuje indeksów. Przed rozpoczęciem procesu ładowania trzeba usunąć wszystkie indeksy tabeli i wyłączyć wszystkie ograniczenia `PRIMARY KEY` oraz `UNIQUE`. Po zakończeniu ładowania indeksy tabeli można na nowo utworzyć.

W przypadku ładowania danych w trybie Direct Path Loading wykonywanym seryjnie (`PARALLEL=FALSE`), SQL*Loader ładuje dane do obszarów w tabeli. Jeśli proces ładowania zostanie przerwany przed załadowaniem wszystkich danych, do tego czasu niektóre dane mogą zostać już zatwierdzone. W trybie Parallel Data Loading każdy proces ładowania tworzy tymczasowe segmenty dla celów ładowania danych. Później segmenty tymczasowe są łączone z tabelą. Jeśli proces ładowania w trybie Parallel Data Loading zostanie przerwany przed załadowaniem wszystkich danych, segmenty tymczasowe nie zostaną dołączone do tabeli. A jeśli segmenty tymczasowe nie zostaną dołączone do tabeli, do której dane są ładowane, żadne dane nie zostaną w tej tabeli zatwierdzone.

Aby skierować poszczególne sesje ładowania danych do oddzielnych plików danych, należy użyć parametru `FILE` narzędzia SQL*Loader. Dzięki skierowaniu każdej sesji ładowania do jej własnego pliku danych można równoważyć obciążenie operacjami wejścia-wyjścia wykonywanymi w trakcie ładowania. Ładowanie danych wiąże się z intensywnym wykonywaniem operacji wejścia-wyjścia i w przypadku ładowania równoległego operacje te trzeba rozproszyć na większej liczbie dysków, aby uzyskać zauważalny wzrost wydajności w porównaniu z ładowaniem seryjnym.

Po zakończeniu ładowania w trybie Parallel Data Load każda sesja może podjąć próbę ponownego włączenia ograniczeń w tabeli. Dopóki trwa przynajmniej jedna sesja ładowania danych, próba przywrócenia ograniczeń nie powiedzie się. Sesja ładowania, która zakończy się jako ostatnia, powinna na nowo włączać ograniczenia i próba ta powinna zakończyć się powodzeniem. Po zakończeniu operacji ładowania danych zawsze powinno się sprawdzać status ograniczeń. Jeśli w tabeli, do której ładowano dane, znajdują się ograniczenia `PRIMARY KEY` i `UNIQUE`, wówczas przed ich ponownym włączeniem można utworzyć powiązane z nimi indeksy.

Zbiorcze przenoszenie danych — korzystanie z tabel zewnętrznych

Oracle pozwala na wykonywanie zapytań na plikach znajdujących się poza bazą danych. Służy do tego obiekt o nazwie **tabela zewnętrzna** (ang. *external table*). Strukturę tabeli zewnętrznej definiuje klauzula `organization external` polecenia `create table`, której składnia bardzo przypomina składnię używaną w plikach kontrolnych narzędzia SQL*Loader.

Wierszami tabeli zewnętrznej nie można manipulować, nie można też tworzyć w tabeli zewnętrznej indeksów, ponieważ każde żądanie dostępu do tabeli powoduje wykonanie jej pełnego skanowania (to znaczy pełnego skanowania pliku na poziomie systemu operacyjnego). W efekcie wydajność zapytań na tabelach zewnętrznych zwykle jest niższa niż na tabelach przechowywanych w bazie danych. Jednak w przypadku systemów, w których ładuje się duże zbiory danych, zastosowanie tabel zewnętrznych może się wiązać z uzyskaniem kilku dodatkowych korzyści:

- Ponieważ tabela nie jest przechowywana w bazie danych, dane są zapisywane tylko jeden raz (tylko poza bazą danych, a nie poza bazą i wewnątrz niej), co pozwala zaoszczędzić miejsce na dysku.
- Ponieważ dane nie są w ogóle ładowane do bazy danych, oszczędza się czas potrzebny na wykonanie tej operacji.

Jako że tabel zewnętrznych nie można indeksować, obiekty te są najbardziej użyteczne w trakcie operacji, w których programy wsadowe odczytują duże zbiory danych. Na przykład w wielu środowiskach hurtowni danych znajdują się obszary etapowe, w których dane są ładowane do tabel tymczasowych, a dopiero potem następuje wstawienie wierszy do tabel, na których użytkownik będzie wykonywał zapytania. Zamiast ładować dane do tabel tymczasowych, można bezpośrednio odczytywać pliki systemu operacyjnego za pośrednictwem tabel zewnętrznych, oszczędzając w ten sposób czas i miejsce na dysku.

Z punktu widzenia architektury dzięki tabelom zewnętrznym zawartość bazy danych można zorientować na obiekty, z których użytkownik będzie korzystał najczęściej — niewielkie tabele kodów, tabele agregujące i tabele transakcyjne — a bardzo duże zbiory danych przechowywać na zewnątrz bazy. Pliki odczytywane przez tabele zewnętrzne można zamieniać w dowolnym momencie, nie narażając przy tym transakcji wykonywanych w bazie danych na żadne dodatkowe koszty.

Zbiorcze wstawianie danych — najczęściej spotykane pułapki i najskuteczniejsze rozwiązania

Jeśli dane nie są wstawiane z pliku jednorodnego, program SQL*Loader nie będzie zbyt użytecznym narzędziem. Jeśli trzeba na przykład przenieść duży zbiór danych z jednej tabeli do drugiej, zapewne będziemy chcieli uniknąć konieczności zapisywania danych w pliku jednorodnym i wczytywania ich z powrotem do bazy. Najszybszym sposobem przenoszenia danych wewnątrz bazy jest przeniesienie ich z jednej tabeli do drugiej bez wychodzenia na poziom systemu operacyjnego.

Gdy dane przenosi się z jednej tabeli do drugiej, można skorzystać z kilku następujących metod zwiększenia wydajności operacji migrowania danych:

- strojenia struktur (usuwania indeksów i procedur wyzwalanych),
- wyłączania ograniczeń na czas migracji danych,
- użycia wskazówek i opcji w celu zwiększenia wydajności transakcji.

Pierwsze z przytoczonych rozwiązań, czyli strojenie struktur, polega na wyłączeniu wszystkich procedur wyzwalanych i indeksów znajdujących się w tabeli, do której dane są ładowane. Jeśli w tabeli docelowej istnieje na przykład procedura wyzwalana, funkcjonująca na poziomie wierszy, procedura ta będzie wykonywana po załadowaniu do tabeli każdego kolejnego wiersza. W miarę możliwości przed rozpoczęciem ładowania danych należy wyłączyć procedury wyzwalane. Jeżeli natomiast procedura wyzwalana powinna być wykonywana na każdym wstawionym wierszu, można wykonać operację zbiorczą po wstawieniu wszystkich wierszy zamiast wywoływania procedury po każdej operacji `insert`. Jeżeli struktury zostaną odpowiednio dostrojone, wówczas operacja zbiorcza powinna trwać krócej niż łączny czas

wykonania procedury wyzwalanej po wstawieniu każdego wiersza. Trzeba jedynie się upewnić, że operacje zbiorcze zostaną wykonane na wszystkich wierszach, które nie zostały wcześniej przetworzone przez procedury wyzwalane.

Oprócz wyłączania procedur wyzwalanych przed rozpoczęciem ładowania danych powinno się również zdjąć indeksy z tabeli docelowej. Jeżeli indeksy pozostaną w tabeli, Oracle będzie nimi dynamicznie zarządzał w trakcie wstawiania każdego kolejnego wiersza. Zamiast więc stale wykonywać operacje na indeksach, lepiej jest je usunąć przed rozpoczęciem ładowania danych, a następnie ponownie utworzyć, gdy ładowanie dobiegnie końca.


Uwaga

Wyłączanie indeksów i procedur wyzwalanych rozwiązuje większość problemów dotyczących wydajności, pojawiających się w trakcie operacji migrowania dużych zbiorów danych z jednej tabeli do drugiej.

Oprócz wyłączania indeksów warto również rozważyć wyłączenie ograniczeń założonych w tabeli. Jeżeli dane źródłowe znajdują się już w tabeli bazy danych, można sprawdzić ich zgodność z ograniczeniami (na przykład występowanie kluczy obcych albo zgodność z ograniczeniem `CHECK`) jeszcze przed rozpoczęciem ładowania ich do tabeli docelowej. Natomiast po zakończeniu ładowania danych ograniczenia można ponownie założyć.

Jeżeli po zastosowaniu opisanych metod wydajność nadal nie jest satysfakcjonująca, należy rozważyć użycie opcji udostępnianych przez bazę Oracle, służących do strojenia operacji migrowania danych. Dostępne są następujące opcje:

- **Wskazówka `append` dla poleceń `insert`.** Podobnie jak tryb ładowania Direct Path, wskazówka `APPEND` powoduje, że bloki danych są ładowane do tabeli, począwszy od poziomu maksymalnego stanu (ang. *high water mark*). Użycie wskazówki `APPEND` może zwiększyć ilość używanego miejsca na dysku.
- **Opcja `nologging`.** Jeżeli wykonywane jest polecenie `create table as select`, można użyć opcji `nologging`, aby uniknąć zapisywania danych do dzienników powtórzeń w trakcie operacji ładowania.
- **Opcje „Parallel".** Mechanizm Parallel Query używa wielu procesów do wykonania pojedynczego zadania. W przypadku polecenia `create table as select` można zrównoleglić część `create table` oraz samo zapytanie. W przypadku użycia opcji zrównoleglenia powinno się również użyć opcji `nologging`, ponieważ w przeciwnym razie operacje wykonywane równolegle często trzeba będzie wstrzymywać ze względu na zserializowane operacje zapisu do plików dziennika powtórzeń online.

Zanim jednak użyta zostanie któraś z powyższych zaawansowanych opcji, trzeba najpierw przeanalizować strukturę tabeli docelowej i upewnić się, że usunięte zostały wszystkie pułapki wspomniane wcześniej w tym punkcie.

Jednym z rozwiązań jest zaimplementowanie odpowiedniej logiki, dzięki której polecenia `insert` będą przetwarzane w tablicach, a nie jako jeden zbiór. Na przykład języki COBOL i C obsługują tablice poleceń `insert`, dzięki czemu można zmniejszyć rozmiar transakcji niezbędnych do przetworzenia dużego zbioru danych.

Zbiorcze usuwanie danych — polecenie truncate

Od czasu do czasu użytkownicy muszą usunąć z tabeli wszystkie rekordy naraz. Gdy w trakcie tej operacji napotkane zostaną błędy, użytkownicy często skarżą się, że segmenty wycofania są zbyt małe, podczas gdy tak naprawdę to rozmiar transakcji jest zbyt duży.

Drugi problem pojawia się po usunięciu wszystkich wierszy. Mimo tego, że w segmencie nie ma już żadnych wierszy, segment nadal zajmuje całą zaalokowaną do niego przestrzeń. Zatem usunięcie wszystkich wierszy nie prowadzi do zmniejszenia się zaalokowanego miejsca na dysku nawet o jeden bajt.

Obydwa problemy rozwiązuje polecenie `truncate`. Jest to polecenie DDL, a nie DML, zatem **nie można go wycofać**. Po wywołaniu polecenia `truncate` na tabeli wszystkie znajdujące się w niej wiersze zostają usunięte, a żadna procedura wyzwalana `delete` nie jest w trakcie tego procesu wykonywana. Jednak tabela utrzymuje wszystkie obiekty od niej zależne, takie jak przyznane uprawnienia, indeksy i ograniczenia.

Użycie polecenia `truncate` to najszybsza metoda usuwania dużych zbiorów danych. Ponieważ polecenie usuwa wszystkie rekordy znajdujące się w tabeli, być może konieczne będzie wprowadzenie zmian w aplikacji, tak aby żaden z rekordów chronionych nie znajdował się w tej samej tabeli, co rekordy, które mają zostać usunięte. Jeżeli używane są partycje, można wyciąć jedną partycję tabeli bez wpływu na pozostałe jej partycje (więcej informacji na ten temat w rozdziale 16.).

Przykładowe polecenie `truncate` na tabeli przedstawia się następująco:

```
truncate table EMPLOYEE drop storage;
```

Przedstawiony przykład, w którym usuwane są wszystkie rekordy tabeli `EMPLOYEE`, ilustruje szerokie możliwości polecenia `truncate`. Klauzula `drop storage` służy do dealokacji z tabeli przestrzeni innej niż `initial` (jest to opcja domyślna). Można więc usunąć z tabeli wszystkie znajdujące się w niej wiersze i odzyskać całą zajmowaną przez nie przestrzeń z wyjątkiem przestrzeni zaalokowanej dla obszaru inicjalnego, bez usuwania samej tabeli.

Polecenie `truncate` działa również w klastrach. W poniższym przykładzie używana jest opcja `reuse storage`, aby pozostawić zaalokowaną całą pustą przestrzeń zajętą przez segment:

```
truncate cluster EMP_DEPT reuse storage;
```

Po wykonaniu polecenia z przykładu wszystkie rekordy znajdujące się w klastrze `EMP_DEPT` zostaną usunięte.

Aby wyciąć partycję, trzeba znać jej nazwę. W poniższym przykładzie partycja `PART3` tabeli `EMPLOYEE` zostanie wycięta przy użyciu polecenia `alter table`:

```
alter table EMPLOYEE
truncate partition PART3
drop storage;
```

Wycięcie partycji `PART3` nie wpłynie w żaden sposób na pozostałe partycje tabeli `EMPLOYEE`. Więcej szczegółowych informacji na temat tworzenia i zarządzania partycjami znajduje się w rozdziale 16.

Alternatywnym rozwiązaniem jest utworzenie programu PL/SQL, który będzie używał dynamicznego kodu SQL do podzielenia dużej operacji `delete` na kilka mniejszych transakcji.

Używanie partycji

Partycji można używać do fizycznego izolowania danych. Można na przykład przechowywać dane na temat transakcji z poszczególnych miesięcy w oddzielnej partycji tabeli `ORDERS`. Jeżeli na tabeli będzie wykonywane zbiorcze ładowanie albo usuwanie danych, można odpowiednio dostosować partycje, aby dostroić operację wykonywaną na danych. Na przykład:

- Można wyciąć partycję i jej indeksy bez wpływania na resztę tabeli.
- Można usunąć partycję przy użyciu klauzuli `drop partition` polecenia `alter table`.
- Można usunąć lokalny indeks partycji.
- Można włączyć tryb `nologging` dla partycji, aby zmniejszyć negatywne oddziaływanie dużych transakcji.

Pod względem wydajności największa korzyść związana z używaniem partycji wypływa z możliwości zarządzania partycjami w oddzieleniu od reszty tabeli. Na przykład dzięki możliwości wycinania partycji można usuwać z tabeli duże zbiory danych (lecz nie wszystkie dane znajdujące się w tabeli), bez generowania danych dla dziennika powtórzeń. W krótkim okresie beneficjentem osiągniętej w ten sposób wyższej wydajności jest administrator bazy danych, zaś w długim okresie całe przedsiębiorstwo odczuwa korzyści związane z większą dostępnością danych. W rozdziale 16. znajduje się więcej informacji na temat implementowania partycji i subpartycji.

Dzięki użyciu opcji `exchange partition` można istotnie zmniejszyć wpływ procesów ładowania danych na dostępność systemu. Najpierw trzeba w tym celu utworzyć pustą tabelę o takiej samej strukturze kolumn jak tabela partycjonowana. Następnie dane trzeba załadować do nowej tabeli i wykonać jej analizę. Na nowej tabeli trzeba utworzyć indeksy, które będą odpowiadać indeksom w tabeli partycjonowanej. Po wykonaniu opisanych czynności należy zmienić partycjonowaną tabelę przy użyciu klauzuli `exchange partition` w taki sposób, by wymienić pustą partycję na nową tabelę już wypełnioną danymi. Od tego momentu wszystkie załadowane dane będą już dostępne w tabeli partycjonowanej. Cała operacja nie ma większego wpływu na dostępność systemu, ponieważ jest to operacja DDL.

Strojenie fizycznych mechanizmów przechowywania danych

Operacje wejścia-wyjścia powinny być równomiernie rozproszone na jak największej liczbie urządzeń. Standardowe rozwiązanie nosi nazwę **SAME** (ang. *stripe and mirror everything*, czyli: paskowanie i tworzenie kopii lustrzanych wszystkich danych). Kluczowymi ograniczeniami, które trzeba wziąć pod uwagę, są limity przepustowości operacji wejścia-wyjścia dysków, a więc rozproszenie operacji wejścia-wyjścia na wielu dyskach pozwala osiągnąć

przepustowość równą sumie przepustowości poszczególnych urządzeń. Paskowanie zwiększa przepustowość, a ta z kolei poprawia wydajność. Z kolei kopie lustrzane stanowią zabezpieczenie na wypadek, gdy dysk ulegnie awarii.

Oprócz strojenia mechanizmów przechowywania danych na poziomie fizycznym istnieje kilka dodatkowych czynników, które należy wziąć pod uwagę. W kolejnych punktach opisano czynniki o charakterze zewnętrznym wobec bazy danych, które mogą mieć istotny wpływ na możliwość szybkiego uzyskania dostępu do danych.

Używanie urządzeń o dostępie bezpośrednim

Urządzenia o dostępie bezpośrednim (ang. *raw devices*) są dostępne w większości systemów operacyjnych z rodziny Unix. Gdy używa się tych urządzeń, Oracle pomija bufor pamięci podręcznej Uniksa i eliminuje obciążenie generowane przez system plików. W przypadku aplikacji wykonującej intensywne operacje wejścia-wyjścia urządzenia o dostępie bezpośrednim mogą doprowadzić do wzrostu wydajności o około 20 procent w porównaniu z tradycyjnymi systemami plików (a także wywołać trochę mniejszy wzrost w porównaniu z mechanizmem Automatic Storage Management). Jednak poczynione ostatnio usprawnienia w systemach plików znacznie zmniejszyły tę różnicę w wydajności.

Urządzeniami o dostępie bezpośrednim nie można zarządzać przy użyciu tych samych poleceń, co w przypadku systemów plików. Nie można na przykład tworzyć zapasowej kopii pojedynczych plików przy użyciu polecenia `tar` — zamiast niego trzeba użyć polecenia `dd`. Polecenie `dd` jest znacznie mniej elastyczne i ogranicza możliwości przywracania bazy.


Uwaga

> Pliki bazy danych Oracle nie powinny znajdować się na tych samych urządzeniach fizycznych, co inne pliki, zwłaszcza jeśli używane są urządzenia o dostępie bezpośrednim. Przechowywanie na tym samym urządzeniu aktywnego systemu plików Uniksa i aktywnego urządzenia o dostępie bezpośrednim bazy Oracle spowoduje problemy z wydajnością operacji wejścia-wyjścia.

Większość systemów operacyjnych, które obsługują urządzenia o dostępie bezpośrednim, udostępnia również warstwę zarządzania woluminami logicznymi pozwalającą administratorom na wykonywanie poleceń systemu plików w odniesieniu do urządzeń o dostępie bezpośrednim. Dzięki takiemu podejściu można połączyć korzyści wynikające z zarządzania systemem plików ze zwiększeniem wydajności osiąganym dzięki urządzeniom o dostępie bezpośrednim. Jeżeli planuje się używanie urządzeń o dostępie bezpośrednim, należy użyć również narzędzie do zarządzania woluminami logicznymi, aby uprościć w ten sposób zarządzanie systemem.

Używanie mechanizmu Automatic Storage Management

Począwszy od systemu Oracle 10*g*, do zarządzania obszarem magazynowania bazy danych można użyć mechanizmu Automatic Storage Management (ASM). W rozdziale 4. zamieszczono kilka przykładów i szczegółowo przeanalizowano to, w jaki sposób ASM zapewnia większość zalet urządzeń o dostępie bezpośrednim związanych z wydajnością, a jednocześnie oferuje łatwość obsługi charakterystyczną dla tradycyjnego systemu plików systemu operacyjnego.

Podczas tworzenia w środowisku ASM nowej przestrzeni tabel lub innej struktury bazy danych, takiej jak plik kontrolny lub plik dziennika powtórzeń, zamiast pliku systemu operacyjnego można określić grupę dysków jako obszar przechowywania struktury bazy danych. ASM oferuje łatwość obsługi charakterystyczną dla mechanizmu Oracle-Managed Files (OMF) i łączy ją z funkcjami kopii lustrzanych i paskowania, aby zapewnić solidnego menedżera systemu plików i woluminów logicznych, który może nawet obsługiwać wiele węzłów klastra Oracle Real Application Cluster (RAC). ASM eliminuje konieczność nabywania zewnętrznego menedżera woluminów logicznych.

ASM nie tylko poprawia wydajność przez automatyczne rozmieszczanie obiektów bazy danych na wielu urządzeniach, ale też zwiększa dostępność, umożliwiając dodawanie do bazy danych nowych urządzeń dyskowych bez konieczności zamykania bazy. ASM automatycznie przywraca równomierną dystrybucję plików przy minimalnym wymogu interwencji administratora.

Zmniejszanie ruchu w sieci

Wraz ze wzrostem stopnia rozproszenia baz danych i aplikacji coraz realniejsze staje się zagrożenie, że to sieć obsługująca serwery okaże się „wąskim gardłem" procesu udostępniania danych użytkownikom. Ponieważ administratorzy baz danych zwykle nie mają większego wpływu na zarządzanie siecią, istotne jest wykorzystanie możliwości bazy danych w zakresie redukcji liczby pakietów sieciowych wymaganych do udostępnienia danych. Zmniejszenie wielkości ruchu w sieci zmniejszy także stopień uzależnienia od samej sieci, a więc pozwoli wyeliminować źródło potencjalnych problemów z wydajnością.

Replikacja danych z wykorzystaniem widoków materializowanych

Oracle pozwala na wykonywanie zapytań i manipulowanie danymi z baz zdalnych. Jednak niepożądane jest ciągłe przesyłanie znacznych ilości danych z jednej bazy danych do drugiej. Aby zmniejszyć ilość danych przesyłanych siecią, powinno się rozważyć zastosowanie innych mechanizmów replikacji.

W klasycznym środowisku rozproszonym każdy element danych istnieje w jednej bazie. Gdy konieczne jest odczytanie danych, pozyskuje się je z baz zdalnych przy użyciu łączy bazodanowych. Takie minimalistyczne podejście można porównać do implementowania aplikacji w klasycznej trzeciej postaci normalnej — czyli podejścia, w którym trudno jest stworzyć jakąkolwiek poważniejszą aplikację. Zmiany w tabelach aplikacji mających zwiększyć wydajność odczytu danych polegają między innymi na denormalizacji danych. Proces denormalizacji z założenia prowadzi do przechowywania danych nadmiarowych, aby w ten sposób skrócić ścieżkę dostępu do danych przez użytkownika.

W środowisku rozproszonym cel ten wypełnia replikowanie danych. Zamiast wykonywać zapytania, które przemierzają sieć w celu spełnienia żądania użytkownika, wykonuje się replikację danych z serwerów zdalnych na serwerze lokalnym. Replikację można wykonywać różnymi metodami, o których więcej powiemy w następnych punktach.

Dane replikowane stają się nieaktualne od razu w momencie ich zreplikowania. Replikowanie danych w celu zapewnienia odpowiedniej wydajności jest więc najefektywniejsze, gdy dane źródłowe są zmieniane stosunkowo rzadko albo gdy do obsługi procesów biznesowych można użyć starych danych.

Dostępne w bazie Oracle funkcje obsługi rozwiązań rozproszonych pozwalają na zarządzanie replikacją danych wewnątrz bazy. **Widoki materializowane** (ang. *materialized views*) replikują dane z głównego źródła do wielu lokalizacji docelowych. Oracle posiada narzędzia służące do odświeżania danych i uaktualniania obiektów docelowych w określonych interwałach czasowych.

Widoki materializowane mogą działać w trybie tylko do odczytu albo mogą pozwalać na uaktualnianie danych. Zagadnienia związane z zarządzaniem widokami materializowanymi zostaną opisane w rozdziale 17., natomiast w niniejszym punkcie zajmiemy się aspektami związanymi ze strojeniem tego rodzaju widoków.

Przed utworzeniem widoku materializowanego dla celów replikacji trzeba najpierw utworzyć w bazie danych łącze z bazą źródłową. Poniższe przykładowe polecenie tworzy prywatne łącze bazodanowe o nazwie `HR_LINK`, z nazwą usługi `LOC`:

```
create database link HR_LINK
connect to HR identified by ESNOTHR1968
using 'loc';
```

Jak można zauważyć w powyższym przykładzie, polecenie `create database link` posiada kilka parametrów:

- nazwę łącza (w przykładzie jest to `HR_LINK`);
- konto, do którego należy się połączyć;
- nazwę usługi zdalnej bazy danych (można ją znaleźć w pliku *tnsnames.ora* serwera); w naszym przykładzie usługa nosi nazwę `LOC`.

Widoki materializowane automatyzują procesy replikacji i odświeżania danych. W momencie tworzenia widoków materializowanych ustala się interwał odświeżania, aby zaplanować odświeżenia zreplikowanych danych. Można zapobiec lokalnym uaktualnieniom i korzystać z odświeżeń bazujących na transakcjach. W ramach operacji odświeżania bazującego na transakcjach, dostępnych dla wielu rodzajów widoków materializowanych, z głównej bazy danych wysyłane są jedynie te wiersze, które w widoku materializowanym uległy zmianie. Mechanizm ten, opisany szerzej w dalszej części tego rozdziału, może znacząco zwiększyć wydajność odświeżania.

W poniższym przykładzie przedstawiono składnię używaną do stworzenia widoku materializowanego na serwerze lokalnym. W poleceniu wskazywana jest nazwa widoku materializowanego (`LOCAL_EMP`) oraz podawane są parametry przechowywania. Podano także zapytanie bazowe oraz interwał odświeżania. W przykładzie nakazano widokowi materializowanemu, by od razu pozyskał dane źródłowe, a kolejne odświeżenie wykonał po siedmiu dniach (`SYSDATE+7`).

```
create materialized view LOCAL_EMP
pctfree 5
tablespace data_2
storage (initial 100K next 100K pctincrease 0)
```

```
refresh fast
    start with SysDate
    next SysDate+7
as select * from EMPLOYEE@HR_LINK;
```

Klauzula `refresh fast` wskazuje bazie danych, by do odświeżania lokalnego widoku materializowanego używać dziennika widoku materializowanego. Możliwość użycia dzienników widoku materializowanego w trakcie odświeżania danych jest dostępna jedynie wówczas, gdy zapytanie bazowe widoku materializowanego jest na tyle proste, by serwer Oracle mógł ustalić wiersz widoku materializowanego, który ulegnie zmianie po wprowadzeniu zmian w wierszu w tabelach źródłowych.

Gdy używany jest dziennik widoku materializowanego, do obiektów docelowych wysyłane są jedynie zmiany, jakie zaszły w tabeli głównej. Jeśli używany jest złożony widok materializowany, wówczas zamiast klauzuli `refresh complete` powinno się użyć klauzuli `refresh fast`. Klauzula `refresh complete` powoduje zastąpienie wszystkich danych w bazowej tabeli widoku materializowanego.

Dzienniki widoku materializowanego trzeba utworzyć w głównej bazie danych przy użyciu polecenia `create materialized view log`. Poniżej znajduje się przykładowe polecenie `create materialized view log`:

```
create materialized view log on EMPLOYEE
tablespace DATA
storage (initial 500K next 100K pctincrease 0);
```

Dziennik widoku materializowanego jest zawsze tworzony w tym samym schemacie, co tabela główna.

Dzięki prostym widokom materializowanym oraz dziennikom widoków materializowanych można zmniejszyć ilość ruchu w sieci generowanego w trakcie utrzymywania zreplikowanych danych. Ponieważ za pośrednictwem dziennika widoku materializowanego przesyłane są jedynie dane zmodyfikowane, utrzymywanie prostych widoków materializowanych powinno wymagać mniejszej ilości zasobów sieciowych niż w przypadku złożonych widoków materializowanych, zwłaszcza jeśli tabele główne są tabelami dużymi i dość statycznymi. Jeśli tabele główne nie są statyczne, wolumen transakcji przesyłanych za pośrednictwem dziennika widoku materializowanego może w ogóle nie różnić się od wolumenu transakcji przesyłanego w celu wykonania pełnego odświeżenia. Więcej informacji na temat odświeżania danych przy użyciu widoków materializowanych znajduje się w rozdziale 17.

Bez względu na to, która metoda odświeżania zostanie wybrana, w bazowej tabeli widoku materializowanego należy utworzyć indeksy, aby zoptymalizować wykonanie zapytań na widoku. W zakresie wydajności celem jest udostępnianie użytkownikom potrzebnych im danych w wymaganym przez nim formacie i w jak najkrótszym czasie. Dzięki utworzeniu materializowanych widoków danych zdalnych można zapobiec używaniu łączy bazodanowych w trakcie wykonywania zapytań. Z kolei utworzenie widoków materializowanych na danych lokalnych pozwala uniknąć powtarzających się operacji agregowania dużych ilości danych i w zamian prezentować użytkownikom dane wstępnie agregowane, które odpowiadają na większość ich pytań.

Używanie wywołań zdalnych procedur

Jeśli w rozproszonym środowisku bazodanowym trzeba używać procedur, można skorzystać z jednej z dwóch opcji: utworzyć procedurę odwołującą się do tabel zdalnych albo utworzyć procedurę zdalną, która będzie wywoływana przez aplikację lokalną.

Odpowiednia lokalizacja procedury zależy od rozproszenia danych i sposobu, w jaki dane mają być używane. Główny nacisk powinno się kłaść na minimalizowanie ilości danych, które trzeba przesyłać siecią w celu wypełnienia żądania użytkownika. Procedura powinna się znajdować w bazie danych, która zawiera większość danych używanych w trakcie wykonywania procedury.

Rozważmy na przykład procedurę w następującej postaci:

```
create procedure MY_RAISE (My_Emp_No IN NUMBER, Raise IN NUMBER)
as begin
      update EMPLOYEE@HR_LINK
      set Salary = Salary+Raise
      where Empno = My_Emp_No;
end;
```

Powyższa procedura korzysta tylko z jednej tabeli (`EMPLOYEE`) znajdującej się na węźle zdalnym (wskazywanym przez łącze `HR_LINK`). Aby zmniejszyć ilość danych przesyłanych w sieci, procedurę należy przenieść do zdalnej bazy danych wskazywanej przez łącze bazodanowe `HR_LINK`, a z klauzuli `from` procedury usunąć odwołanie do tej bazy danych. Następnie można wywołać procedurę z lokalnej bazy danych przy użyciu łącza bazodanowego, jak w poniższym poleceniu:

```
execute MY_RAISE@HR_LINK(1234,2000);
```

Powyższe polecenie przekazuje do procedury dwa parametry: `My_Emp_No` o wartości 1234 oraz `Raise` o wartości 2000. Procedura jest wywoływana przy użyciu łącza bazodanowego, aby wskazać bazie danych, gdzie procedura ta się znajduje.

Pod względem strojenia struktur bazy danych wywoływanie procedury zdalnej ma tę zaletę, że całość przetwarzania wykonywanego przez procedurę jest realizowana w bazie danych, w której znajdują się dane. Wywoływanie procedury zdalnej minimalizuje ilość ruchu w sieci, jaki należy wygenerować, aby wykonać przetwarzanie przez procedurę.

Aby uzyskać przezroczystość lokalizacji procedury, można użyć lokalny synonim, który wskazuje na procedurę zdalną. W synonimie podaje się nazwę łącza bazodanowego, aby żądania pochodzące od użytkownika automatycznie angażowały zdalną bazę danych:

```
create synonym MY_RAISE for MY_RAISE@HR_LINK;
```

Użytkownik będzie mógł dzięki temu wykonywać polecenia w postaci:

```
execute MY_RAISE(1234,2000);
```

Polecenie to wykona procedurę zdalną definiowaną przez synonim `MY_RAISE`.

Użycie narzędzia Automatic Workload Repository

W przypadku systemu Oracle Database 10*g* i jego poprzedników pakiet STATSPACK gromadzi statystyki dotyczące bazy danych i generuje raporty. Jednak raporty te są wyłącznie w formacie tekstowym! Począwszy od systemu Oracle 10*g*, dostępne jest rozwiązanie Automatic Workload Repository (AWR), które rozszerza funkcje pakietu STATSPACK. AWR generuje wszystkie statystyki, które można uzyskać za pomocą pakietu STATSPACK, a także wiele innych. AWR jest ściśle zintegrowany z narzędziem OEM, dzięki czemu w prosty sposób można analizować i usuwać problemy z wydajnością.

Podobnie jak STATSPACK, AWR gromadzi i utrzymuje statystyki wydajności dla celów identyfikowania problemów oraz strojenia. Na podstawie danych AWR można tworzyć raporty oraz odczytywać je za pośrednictwem widoków i narzędzia OEM. Można na przykład tworzyć raporty na temat przebiegu ostatniej sesji albo statystyki dotyczące całego systemu i użycia kodu SQL.

AWR zbiera statystyki systemowe co godzinę (pobierając „migawki" bazy danych) i przechowuje dane w tabelach repozytorium. Podobnie jak w przypadku pakietu STATSPACK, wymagania repozytorium AWR dotyczące dostępnego miejsca na dysku wzrastają, gdy wydłuża się okres utrzymywania danych historycznych lub zmniejszony zostanie interwał między kolejnymi migawkami. Domyślnie w repozytorium przechowywane są dane sprzed maksymalnie siedmiu dni. Migawki przechowywane w repozytorium AWR można przeglądać za pośrednictwem widoku DBA_HIST_SNAPSHOT.

Aby włączyć AWR, należy przypisać parametrowi inicjalizacyjnemu STATISTICS_LEVEL wartość TYPICAL lub ALL. Jeśli parametrowi STATISTICS_LEVEL przypisana zostanie wartość BASIC, będzie można ręcznie tworzyć migawki danych AWR, lecz nie będą one aż tak szczegółowe, jak migawki wykonywane automatycznie przez AWR. Ustawienie dla parametru STATISTICS_LEVEL wartości ALL powoduje, że oprócz statystyk tworzonych w przypadku wartości TYPICAL generowane są odpowiednie statystyki dotyczące systemu operacyjnego i planów wykonywania.

Zarządzanie migawkami

Aby ręcznie wykonać migawkę, należy użyć procedury CREATE_SNAPSHOT pakietu DBMS_WORKLOAD_
↪REPOSITORY:

```
execute DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT ();
```

Aby zmienić ustawienia migawek, należy użyć procedury MODIFY_SNAPSHOT_SETTINGS. Można zmienić czas utrzymywania migawki (w minutach) oraz interwał (również w minutach). Poniższe polecenie zmienia interwał dla bieżącej bazy danych na 30 minut:

```
execute DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS
( interval => 30);
```

Aby usunąć migawki z danego zakresu, należy wywołać procedurę DROP_SNAPSHOT_RANGE i wskazać początkowy i końcowy identyfikator migawek, które mają zostać usunięte:

```
execute DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE
    (low_snap_id => 1, high_snap_id => 10);
```

Zarządzanie punktami odniesienia

Zestaw wybranych migawek można wskazać jako punkt odniesienia dla wydajności systemu. Dane punktu odniesienia będą utrzymywane w celu porównywania do nich w późniejszym czasie migawek. Procedura CREATE_BASELINE służy do wskazania początkowej i końcowej migawki punktu odniesienia:

```
execute DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE
(start_snap_id => 1, end_snap_id => 10,
baseline_name => 'Punkt poniedziałkowy');
```

W momencie tworzenia punktu odniesienia Oracle nada mu identyfikator. Punkty odniesienia z przeszłości można przeglądać za pomocą widoku DBA_HIST_BASELINE. Migawki wskazane jako początek i koniec punktu odniesienia będą utrzymywane do czasu, gdy punkt odniesienia zostanie usunięty. Aby usunąć punkt odniesienia, należy wykonać procedurę DROP_BASELINE:

```
execute DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE
(baseline_name => 'Punkt poniedziałkowy', cascade => FALSE);
```

Jeżeli parametrowi CASCADE procedury DROP_BASELINE zostanie przypisana wartość TRUE, wówczas w momencie usuwania punktu odniesienia usunięte zostaną również powiązane z nim migawki.

Dane AWR można przeglądać przy użyciu narzędzia OEM lub za pomocą widoków danych słownikowych, wspomnianych wcześniej w tym punkcie. Dodatkowymi widokami obsługującymi AWR są: V$ACTIVE_SESSION_HISTORY (odświeżany co sekundę), DBA_HIST_SQL_PLAN (plany wykonania) oraz DBA_HIST_WR_CONTROL (ustawienia AWR).

Generowanie raportów AWR

Raporty narzędzia AWR można generować przy użyciu narzędzia OEM lub za pomocą dostępnych skryptów raportujących. Skrypt *awrrpt.sql* generuje raport dotyczący różnic w statystykach między migawką początkową i końcową. Natomiast drugi skrypt, o nazwie *awrrpti.sql*, wyświetla raport utworzony na podstawie migawek początkowej i końcowej wskazanej bazy danych i instancji.

Skrypty *awrrpt.sql* i *awrrpti.sql* znajdują się w podkatalogu *$ORACLE_HOME/rdbms/admin* głównego katalogu oprogramowania Oracle. W momencie wykonania raportu (z dowolnego konta administratora bazy danych) trzeba będzie wskazać format raportu (HTML lub tekstowy), liczbę dni, dla których migawki mają zostać przedstawione, identyfikatory migawek początkowej i końcowej oraz nazwę pliku wynikowego.

Uruchamianie raportów narzędzia Automatic Database Diagnostic Monitor

Zamiast ręcznie wykonywać raporty na podstawie tabel narzędzia AWR (podobnie jak to miało miejsce w przypadku pakietu STATSPACK obecnego w poprzednich wersjach systemu Oracle), można wykorzystać narzędzie Automatic Database Diagnostic Monitor (ADDM). Ponieważ narzędzie to bazuje na danych AWR, wymaga zdefiniowania odpowiedniej wartości parametru STATISTICS_LEVEL (TYPICAL lub ALL, zgodnie z wcześniejszym opisem). Dostęp do ADDM można uzyskać w sekcji *Performance Analysis* narzędzia OEM; można również ręcznie wykonać raport ADDM.

Aby uruchomić ADDM na zestawie migawek, należy posłużyć się skryptem *addmrpt.sql*, znajdującym się w podkatalogu *$ORACLE_HOME/rdbms/admin* katalogu głównego oprogramowania Oracle.


Uwaga

Aby wykonać raport, należy posiadać uprawnienie systemowe ADVISOR.

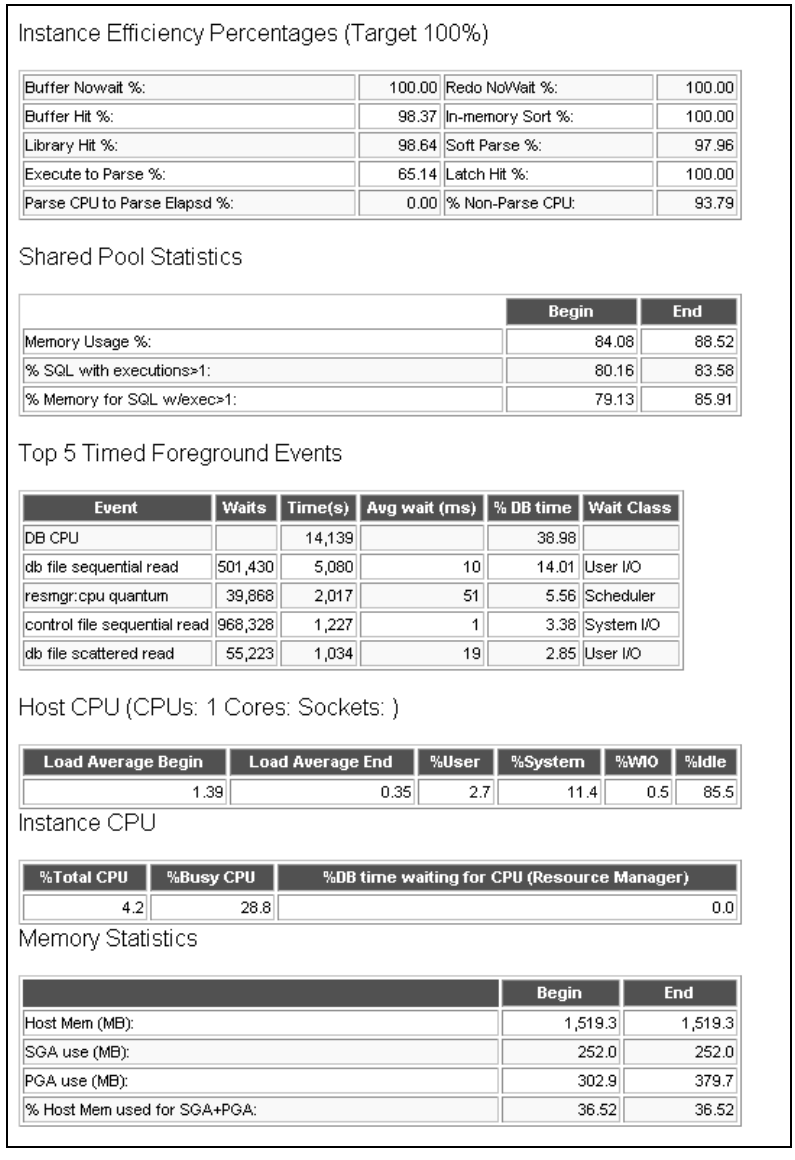
W narzędziu SQL*Plus należy wykonać skrypt *addmrpt.sql*. Konieczne będzie podanie identyfikatora migawki początkowej i końcowej oraz nazwy pliku wynikowego.

Aby dotrzeć do danych ADDM, można skorzystać z narzędzia OEM albo użyć widoków danych słownikowych ADVISOR. Do grupy widoków ADVISOR należą widoki DBA_ADVISOR_TASKS (zadania istniejące), DBA_ADVISOR_LOG (status i postęp zadań), DBA_ADVISOR_RECOMMENDATIONS (zakończone zadania diagnostyczne oraz rekomendacje) oraz DBA_ADVISOR_FINDINGS. Zaimplementowanie zaleceń ADDM pozwoli na wyeliminowanie kwestii zidentyfikowanych przez to narzędzie. Rysunek 8.1 przedstawia typowy raport AWR wygenerowany na podstawie domyślnego punktu odniesienia. W przytoczonym przykładzie migawka zainicjowana została 14 września 2007 r., a zakończona 22 września 2007 r. Wydaje się, że baza danych w niewielkim stopniu wykorzystuje spore zasoby procesora i pamięci. Na przykład nie ma miejsca „rywalizacja" o rejestry, a ponadto jest wystarczająca ilość pamięci, aby wszystkie operacje sortowania wykonywać bez użycia dysku.

Zastosowanie narzędzia Automatic SQL Tuning Advisor

Narzędzie Automatic SQL Tuning Advisor, będące nowością w systemie Oracle Database 11*g*, uruchamiane jest po otwarciu domyślnego okna konserwacji (za pomocą AutoTask) i zajmuje się instrukcjami SQL o największym obciążeniu, które zostały zebrane przez mechanizm AWR. Po rozpoczęciu w oknie konserwacji automatycznego strojenia instrukcji SQL narzędzie Automatic SQL Tuning Advisor wykonuje następujące kroki:

1. Identyfikacja na podstawie statystyk AWR powtarzającej się instrukcji SQL o dużym obciążeniu. Ignorowane są ostatnio strojone i rekurencyjne instrukcje SQL.
2. Strojenie instrukcji SQL o dużym obciążeniu za pomocą wywołań kierowanych do narzędzia SQL Tuning Advisor.



Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	100.00	Redo NoWait %:	100.00
Buffer Hit %:	98.37	In-memory Sort %:	100.00
Library Hit %:	98.64	Soft Parse %:	97.96
Execute to Parse %:	65.14	Latch Hit %:	100.00
Parse CPU to Parse Elapsd %:	0.00	% Non-Parse CPU:	93.79

Shared Pool Statistics

	Begin	End
Memory Usage %:	84.08	88.52
% SQL with executions>1:	80.16	83.58
% Memory for SQL w/exec>1:	79.13	85.91

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		14,139		38.98	
db file sequential read	501,430	5,080	10	14.01	User I/O
resmgr:cpu quantum	39,868	2,017	51	5.56	Scheduler
control file sequential read	968,328	1,227	1	3.38	System I/O
db file scattered read	55,223	1,034	19	2.85	User I/O

Host CPU (CPUs: 1 Cores: Sockets:)

Load Average Begin	Load Average End	%User	%System	%WIO	%Idle
1.39	0.35	2.7	11.4	0.5	85.5

Instance CPU

%Total CPU	%Busy CPU	%DB time waiting for CPU (Resource Manager)
4.2	28.8	0.0

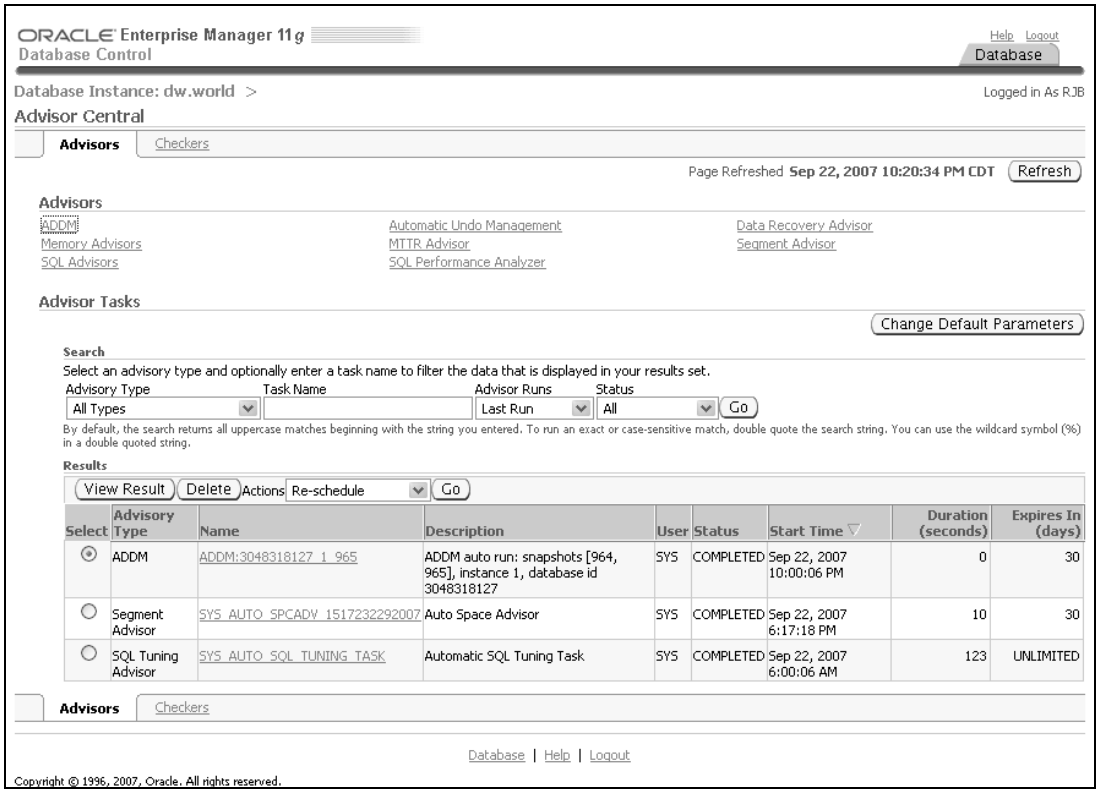
Memory Statistics

	Begin	End
Host Mem (MB):	1,519.3	1,519.3
SGA use (MB):	252.0	252.0
PGA use (MB):	302.9	379.7
% Host Mem used for SGA+PGA:	36.52	36.52

Rysunek 8.1. *Przykładowy raport AWR uzyskany za pośrednictwem OEM*

3. Tworzenie profili SQL dla instrukcji SQL o dużym obciążeniu. Wydajność testowana jest z użyciem profilu i bez niego.
4. Jeśli wydajność poprawiła się co najmniej trzykrotnie, profil jest automatycznie zatrzymywany. W przeciwnym razie informacji o wzroście wydajności należy szukać w raporcie dotyczącym strojenia.

Rysunek 8.2 prezentuje na stronie *Advisor Central* podsumowanie zadań narzędzia Advisor. W przykładzie widać zestawienie wyników dla narzędzi Automatic Database Diagnostic Monitor (ADDM), Segment Advisor i SQL Tuning Advisor.



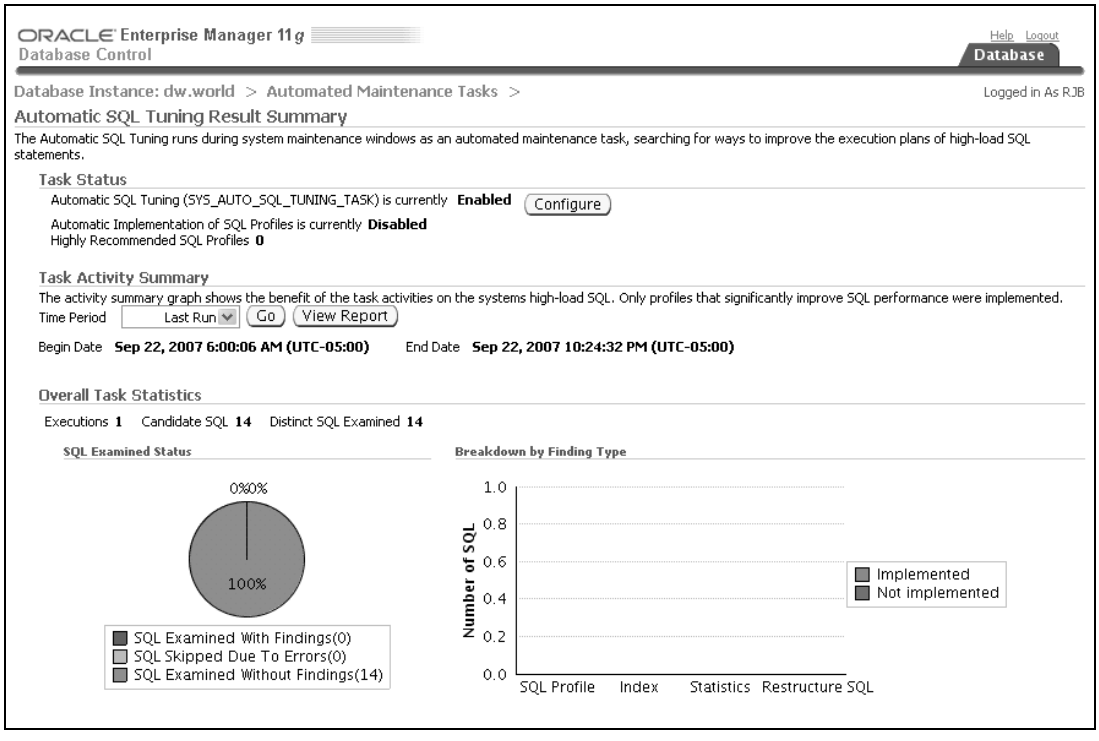
Rysunek 8.2. *Podsumowanie na stronie Advisor Central narzędzia OEM*

Kliknięcie odsyłacza dla zadania SQL Tuning Advisor spowoduje wyświetlenie strony *SQL Tuning Result Summary* widocznej na rysunku 8.3. W przypadku przykładowej bazy danych o niskim poziomie obciążenia narzędzie SQL Tuning Advisor znalazło 14 powtarzających się instrukcji SQL, które zaklasyfikowano jako mocno obciążające. Jednak nie zidentyfikowano metody zwiększenia wydajności tych instrukcji SQL.

Rozwiązania wykonujące strojenie

Niniejszy rozdział nie zawiera opisu wszystkich potencjalnych rozwiązań służących do strojenia. Istnieje jednak jednolity sposób postępowania, na którym bazują techniki i narzędzia zaprezentowane w tym rozdziale. Zanim zapadnie decyzja o poświęceniu czasu i zasobów na zaimplementowanie nowej funkcji, powinno się najpierw ustabilizować pracę i architekturę środowiska — serwera, bazy danych i aplikacji. Jeśli środowisko jest już stabilne, możliwe będzie szybkie osiągnięcie dwóch celów:

1. Odtworzenie problemu z wydajnością.
2. Wyizolowanie przyczyny problemu.



Rysunek 8.3. *Wyniki zwrócone przez narzędzie Automatic SQL Tuning Advisor*

Aby osiągnąć obydwa cele, konieczne może być utworzenie środowiska testowego, wykorzystywanego do testów wydajnościowych. Gdy problem zostanie już wyizolowany, można go obsłużyć, wykonując kroki opisane w tym rozdziale. Ogólnie mówiąc, podejście do strojenia systemu powinno odzwierciedlać kolejność punktów z niniejszego rozdziału:

1. Ocena projektu aplikacji.
2. Strojenie kodu SQL.
3. Strojenie użycia pamięci.
4. Strojenie mechanizmów przechowywania danych.
5. Strojenie operacji manipulujących danymi.
6. Strojenie fizycznych i logicznych mechanizmów przechowywania danych.
7. Strojenie ruchu w sieci.

Zależnie od natury aplikacji można zdecydować się na wykonanie odpowiednich czynności w innej kolejności albo połączenie ze sobą niektórych z nich.

Jeżeli nie można zmienić struktury aplikacji ani kodu SQL, można skupić się na strojeniu pamięci i dysków używanych przez aplikację. Po wprowadzeniu zmian w ustawieniach pamięci i dysków trzeba ponownie przeanalizować strukturę aplikacji oraz kod SQL, aby upewnić się, że wprowadzone zmiany nie wpłyną negatywnie na działanie aplikacji. Konieczność

ponownego przeanalizowania struktury aplikacji jest szczególnie ważna wówczas, gdy zastosowana zostanie metoda replikowania danych, ponieważ aktualność replikowanych danych może być źródłem problemów w realizacji procesów biznesowych obsługiwanych przez aplikację.