Oracle8i. Podręcznik administratora baz danych

Autorzy: Kevin Loney, Marlene Theriault
Tłumaczenie: Zbigniew Majewski, Anna Lenkiewicz, Grzegorz Stawikowski
ISBN: 83-7197-528-7
Tytuł oryginału: Oracle8i DBA Handbook
Format: B5, stron: 944

Baza danych Oracle8i to potężna struktura, pozwalająca obsługiwać kluczowe operacje zachodzące w przedsiębiorstwie – pozwala na łatwy dostęp i zarządzanie poprzez sieć WWW. „Oracle8i. Podręcznik administratora baz danych" dostarcza wielu rozwiązań związanych z codzienną pracą administratora. Z lektury książki dowiedzieć się można: jakie ustawiać parametry bazy danych w celu osiągnięcia maksymalnej wydajności, jak monitorować bazę danych, jak stroić bazę danych, w jaki sposób zabezpieczać dane zgromadzone w bazie.

W książce opisano także szczegóły dotyczące rozproszonych baz danych, a także własności związanych z architekturą klient-serwer, zarówno dla platformy UNIX jak i Windows NT. Informacje zawarte w książce dotyczą wszystkich kwestii związanych z nowymi rewolucyjnymi zmianami, jakie niesie ze sobą baza danych Oracle8i – w szczególności dotyczące dostępności bazy danych poprzez sieć WWW.

W książce zawarto najbardziej aktualne informacje odnośnie:

- Konfiguracji architektury bazy danych, zarówno jej fizycznego jak i logicznego układu
- Zarządzania procesem tworzenia
- Obsługi narzędzi i pakietów systemu Oracle, zarówno pakietów Oracle Financials, Oracle Designer jak i innych narzędzi
- Obsługi związanej ze strojeniem bazy danych, usuwaniem skutków awarii, archiwizacją oraz odtwarzaniem
- Narzędzi SQL*Net oraz Net8
- Zarządzania rozproszonymi bazami danych oraz konfiguracji architektury klient-serwer i środowiska sieci WWW.

Książka przeznaczona jest zarówno dla początkujących jaki i zaawansowanych administratorów baz danych Oracle. „Oracle8i. Podręcznik administratora baz danych" jest niezbędnym źródłem pozwalającym na utworzenie i administrowanie wysokowydajną bazą danych Oracle8i – przeznaczoną do rozwiązań internetowych.

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

Spis treści

Rozdział 5.

Zarządzanie procesem tworzenia aplikacji

Zarządzanie procesem tworzenia aplikacji jest niezwykle trudnym zagadnieniem. Z punktu widzenia administratora bazy danych najlepszym wyjściem byłoby bezpośrednie uczestnictwo w pracach zespołu zajmującego się aplikacją. W niniejszym rozdziale zamieszczono informacje dotyczące migracji aplikacji w bazach danych oraz szczegóły techniczne dotyczące implementacji. Szczegóły te dotyczą m.in. specyfikacji ról systemowych oraz zasad związanych z określaniem rozmiarów obiektów w bazie danych.

Szczególną uwagę poświęcono kontroli czynności związanych z procesem tworzenia obiektów w bazie danych na różnych etapach. Czynności te powinny odpowiadać czynnościom związanym z planowaniem bazy danych, a które opisano w rozdziałach 3., 4., 6. i 8. W niniejszym rozdziale przedstawiono również sposoby kierowania monitorowaniem oraz strojeniem bazy danych po jej utworzeniu.

Tworzenie aplikacji poprzez ograniczanie się do wykonania serii poleceń `create table` w bazie danych nie prowadzi do zintegrowania procesu tworzenia z innymi głównymi obszarami dotyczącymi aplikacji (planowanie, monitorowanie, strojenie). Administrator bazy danych musi włączyć się w proces tworzenia aplikacji, aby poprawnie zaprojektowana baza danych była odpowiednim elementem końcowego produktu, jakim jest aplikacja. Rozdział ten zawiera także wiele ważnych informacji odnośnie monitorowania oraz strojenia bazy danych.

Trzy podstawowe warunki sukcesu

Poprawne działanie bazy danych jest zależne od prawidłowego przeprowadzenia czterech etapów: planowania, tworzenia, monitorowania oraz strojenia. Elementy wpływające na pomyślny przebieg pracy bazy danych można opisać w trzech kluczowych kategoriach: właściwa współpraca zespołu, procesy zarządzania oraz technologia.

Zarządzanie wysiłkami twórczymi programistów wymaga współdziałania na wszystkich tych trzech płaszczyznach:

1. *Właściwa współpraca*: zespół programistów musi współpracować z administratorem bazy danych.
2. *Zarządzanie*: egzekwowanie stosowania przez programistów metodologii związanych z procesem tworzenia.
3. *Technologia*: Programiści i administratorzy bazy danych muszą zdefiniować mechanizmy w celu zapewnienia odpowiedniego poziomu uwagi i zaangażowania.

Próba wprowadzenia metodologii związanej z procesem tworzenia bez zbiorowego zaangażowania lub bez odpowiedniej technologii koordynującej wysiłki zespołu nie może przynieść długotrwałych korzyści.

Prawidłowa współpraca

Aby przełamać tradycyjne bariery pomiędzy administratorami baz danych a programistami, obydwie te grupy powinny odnosić się do siebie w sposób formalny przy pełnej akceptacji takiego nowego, lepszego układu. Jest to możliwe do osiągnięcia pod warunkiem obustronnego zrozumienia, że taki układ przyniesie korzyści podczas procesu tworzenia danej aplikacji. Konieczne jest wspólne zaangażowanie związane z przezwyciężaniem początkowych trudności, które mogą zakłócać ten proces już na jego starcie.

Połączenie sił administratorów oraz programistów przynosi korzyści związane z:

- budową aplikacji, które są łatwiejsze w obsłudze;
- budową aplikacji o właściwych wymiarach i organizacji, a więc nie powodujących przestojów podczas reorganizacji;
- tworzeniem właściwych indeksów w celu przyspieszenia działania aplikacji;
- określaniem tabel oraz indeksów, które będą najczęściej używane przez aplikację;
- wskazywaniem i poprawą źle skonstruowanych poleceń SQL w celu uniknięcia ich ujemnego wpływu na wydajność pracy aplikacji;
- wskazywaniem tabel statycznych, używanych tylko przez zapytania;
- budową rozsądnego, łatwego do zrozumienia interfejsu aplikacji;
- wcześniejszym identyfikowaniem problemów technicznych;
- wskazywaniem ewentualnych źródeł konfliktów pomiędzy pracą użytkowników a długotrwałymi operacjami wsadowymi.
- przyznaniem administratorowi uprawnień do większości obiektów w aplikacji, co pozwoli mu w późniejszym czasie na łatwiejsze zarządzanie nimi.

Jeszcze większe korzyści podczas wykonywania powyższych czynności przynieść może dobre zrozumienie przez administratora specyfiki aplikacji oraz potrzeb związanych z zadaniami, które ta aplikacja obsługuje. Jeśli administrator dobrze pojmuje te potrzeby, może również lepiej zrozumieć cele programisty w czasie tworzenia aplikacji. Następną korzyścią związaną z dobrym rozumieniem aplikacji jest większa możliwość porozumienia się z programistami oraz z użytkownikami aplikacji. Kolejną ważną korzyścią płynącą ze wspólnej pracy nad tworzeniem aplikacji jest możliwość doboru właściwych rozmiarów obiektów bazy danych używanej przez aplikację oraz odpowiednie jej nastrojenie.

Dzięki wspólnemu, zespołowemu wysiłkowi administratorów i programistów można znacznie ograniczyć koszty związane z obsługą, zamykaniem bazy danych, z czasem przeznaczanym na strojenie, z niewydajnym działaniem bazy i z niezadowoleniem użytkowników. Metodologia musi jasno definiować zakres obowiązków oraz role poszczególnych grup (administratorów i programistów) oraz musi być przez te grupy zaakceptowana. To bardzo ułatwia przestrzeganie metodologii także przez osoby trzecie.

Proces zarządzania

Aby właściwie kierować wysiłkami związanymi z tworzeniem aplikacji, metodologia nie może opisywać jedynie związków pomiędzy różnymi funkcjonalnymi obszarami, ale musi także jasno definiować zadania wymagane w każdej fazie tworzenia aplikacji. Przykładowo, moment przejścia aplikacji do fazy testowej lub z fazy testowej do produkcyjnej jest ściśle definiowany przez metodologię.

Jeśli prace na etapie tworzenia są zakończone, sprawdzone i zaakceptowane, wtedy aplikacja może przejść do fazy testowania, a programista pracuje zgodnie z ograniczeniami występującymi w tej fazie.

Definiowanie środowiska

Większość środowisk aplikacyjnych jest określanych za pomocą dwóch do pięciu faz. Są to: tworzenie, testy systemowe, testy eksploatacyjne, testy przyjęcia oraz faza produkcyjna. Dla potrzeb niniejszej dyskusji trzy fazy związane z testami połączono w jedną. Dokładna liczba faz zależy ściśle od przyjętej metodologii.

Pomimo że w niniejszej części nie zwrócono na to szczególnej uwagi, należy pamiętać, że testy eksploatacyjne powinny być integralną częścią procesu tworzenia aplikacji. W praktyce test eksploatacyjny jest najbardziej efektywny, jeśli przeprowadzany jest przed testem przyjęcia. Problemy związane z użyciem ładowania powinny zostać rozwiązane przed fazą testów przyjęcia.

Metodologia nakazuje określenie cech produktu po przejściu każdej z faz oraz cele, które należy osiągnąć podczas przeprowadzania każdej następnej fazy. Dopiero po zdefiniowaniu powyższego można rozpocząć wprowadzanie bazy danych w poszczególne etapy tworzenia aplikacji.

Przykładowo, w fazie tworzenia użytkownicy mogą dowolnie wprowadzać zmiany związane z budową tabel, testowaniem nowych pomysłów oraz tworzeniem nowych obiektów. W celu koordynacji logicznego modelu tworzonej bazy należy wykorzystywać zintegrowane narzędzia typu CASE, takie jak Oracle Designer. Repozytorium narzędzi CASE powinno być używane do tworzenia pierwszego zestawu obiektów w bazie danych w każdym środowisku. Programiści powinni być odpowiedzialni za utrzymywanie słownika CASE.

W chwili, kiedy system przechodzi do fazy testowania, powinny być już określone rozmiary tabel, konta użytkowników oraz potrzeby wydajnościowe. Pozwoli to administratorowi na utworzenie właściwej bazy dla pierwszych prób aplikacji a także umożliwi mu zaobserwowanie ewentualnych problemów z osiąganiem założonej wydajności.

W fazie produkcyjnej programiści kończą swoją pracę. Warto zaznaczyć, że baza danych zmienia tylko grupę użytkowników. Wszystkie zmiany w fazie produkcyjnej powinny wpierw przejść przez fazę testową. Wszystkie wymagane przez system modyfikacje powinny być jasno określone w fazie testowej.

W celu właściwej obsługi uprawnień systemowych konta programistów powinny być konfigurowane zależnie od fazy, w której znajduje się aplikacja. W następnym podrozdziale opisano role systemowe zdefiniowane dla programistów i przyznawanych im w zależności od fazy, w jakiej znajduje się aplikacja.

Definicje ról

W procesie tworzenia aplikacji stosowane są trzy role z zestawu ról systemowych dostarczanych przez Oracle. Są to: CONNECT, RESOURCE oraz DBA. Pozostałe role związane są z administracją bazą danych i zostały opisane w rozdziale 9. Możliwe jest także tworzenie własnych ról będących zestawem dowolnych uprawnień systemowych. Praca z nimi może jednak stwarzać większe trudności niż w przypadku ról standardowych. Można przyznawać użytkownikom oraz programistom role oparte na uprawnieniach systemowych w zależności od środowiska, w jakim pracują.

Rola CONNECT

Rola CONNECT pozwala użytkownikowi na otwieranie sesji w bazie danych. Poza tym rola ta zawiera jeszcze inne uprawnienia systemowe, takie jak: ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SYNONYM, CREATE TABLE oraz CREATE VIEW. Daje to znacznie więcej możliwości, niż tylko połączenie się z bazą danych. Jednakże użytkownik z tą przypisaną rolą nie będzie mógł utworzyć tabeli lub klastra (obiekty te wykorzystują przestrzeń w bazie danych), jeżeli nie zostanie mu przyznany limit na odpowiedniej przestrzeni tabel, lub jeżeli nie zostanie mu przyznana rola RESOURCE (patrz dalej). W rozdziale 9. znajdują się szczegóły dotyczące przyznawanie limitów na przestrzeniach tabel.

Najczęściej rola CONNECT jest wystarczającym uprawnieniem użytkowników końcowych w większości środowisk pracy aplikacji. Niekiedy rola CONNECT może również odpowiadać potrzebom programistów. Jest tak, gdy programiści nie potrzebują uprawnień do tworzenia obiektów takich, jak procedury, pakiety, wyzwalacze lub inne obiekty użytkownika.

Jeżeli istnieje potrzeba ograniczenia uprawnień systemowych użytkowników aplikacji, można stworzyć własną rolę, np. o nazwie APPLICATION_USER, która nadawać będzie tylko uprawnienie pozwalające na otworzenie sesji:

```
create role APPLICATION_USER;
grant CREATE SESSION to APPLICATION_USER;
grant APPLICATION_USER to username;
```

Rola RESOURCE

Standardowa rola RESOURCE nadaje następujące uprawnienia systemowe: CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE. Użytkownicy, którym przyznano rolę RESOURCE, posiadają także uprawnienie UNLIMITED TABLESPACE. Mogą oni dzięki temu unieważnić zdefiniowane dla nich limity na przestrzeniach tabel. Rola RESOURCE powinna być przyznawana programistom, którzy tworzą obiekty PL/SQL takie, jak procedury i wyzwalacze. Jeśli programiści używają opcji obiektowych, rola RESOURCE poprzez uprawnienie CREATE TYPE umożliwia im tworzenie nowych typów oraz wykonywanie metod.

Jeżeli istnieje potrzeba ograniczenia uprawnień nadanych programistom, można utworzyć własną rolę i przydzielić jej odpowiednie uprawnienia systemowe. Przykładowo, jeżeli programiści mają mieć uprawnienia do tworzenia indeksów oraz obiektów proceduralnych, ale nie mają mieć możliwości tworzenia tabel oraz klastrów, należy utworzyć rolę, której przydziela się wszystkie uprawnienia systemowe roli RESOURCE za wyjątkiem tych, które pozwalają na tworzenie tabel oraz klastrów. W większości przypadków rola RESOURCE powinna być przydzielana tylko na etapie tworzenia aplikacji. W czasie testowania oraz w fazie produkcyjnej rola CONNECT powinna zapewniać wystarczające możliwości. Nadając programistom role RESOURCE w produkcyjnej bazie danych zmniejsza się kontrolę administratora bazy nad zmianami zachodzącymi w bazie danych.

Rola DBA

Rola DBA zawiera wszystkie uprawnienia systemowe nadane z opcją `with admin option`, co oznacza, że użytkownik, któremu przyznano tę rolę, będzie mógł przekazywać uprawnienia w niej zawarte innym użytkownikom. Nie należy nadawać roli DBA użytkownikom oraz programistom w żadnej z faz istnienia aplikacji. Jeśli programiści mają rolę DBA w czasie tworzenia aplikacji, mogą oni podczas tworzenia aplikacji błędnie zakładać, że będą oni posiadać te same uprawnienia także w momencie, kiedy aplikacja będzie już oddana do pracy. Zbyt pochopne przyznawanie uprawnień związanych z rolą DBA powoduje brak odpowiedniego poziomu zabezpieczenia dla danych w bazie. Zagwarantowanie odpowiedniego poziomu zabezpieczeń danych w bazie danych jest kluczowym zadaniem administratora.

Testowanie oraz wdrożenie

Właściwe określenie ról dla fazy testowania zależy od warunków, w jakich faza ta się odbywa. Jeśli testy ograniczają się do przeprowadzenia zadań, których wykonanie wymaga posiadania tych samych uprawnień, co w przypadku normalnej produkcyjnej

pracy aplikacji, wtedy przydzielane role są podobne do tych, które zostają przyznane w produkcyjnej bazie danych. Jeśli jednak programiści muszą w czasie testów dokonywać modyfikacji bazy danych, wtedy potrzebują oni podobnych uprawnień, jak w przypadku fazy tworzenia aplikacji.

Tabele oraz inne obiekty bazy danych używane przez aplikację są własnością pojedynczego użytkownika (konta). Jeśli jednak zachodzi potrzeba wykonania pewnych zmian za pomocą konta, które należy do określonego schematu (np. utworzenie powiązania między bazami danych), wtedy administrator może tymczasowo zalogować się na to konto i przeprowadzić niezbędne zmiany. Należy podkreślić, że programiści nie powinni posiadać roli RESOURCE w fazie testowania aplikacji. Wszystkie wymagane zmiany powinny zostać przeprowadzone w fazie tworzenia. Przeniesienie tych modyfikacji do fazy testowania następować powinno poprzez udokumentowany proces kontroli zmian.

Zadania

Odpowiednie zastosowanie metodologii wymaga określenia listy zadań, które muszą zostać wykonane podczas trwania fazy tworzenia aplikacji. Metodologia musi jasno określać zarówno format, jak i poziom szczegółów wymaganych do wykonania każdego zadania we wszystkich fazach rozwoju aplikacji. Metodologia powinna dokładnie określać zakres wymagań, w szczególności:

- diagram związków jednostek;
- diagram rozkładu fizycznego bazy danych;
- wymagania dotyczące przestrzeni;
- cele strojenia;
- wymagania związane z ochroną danych;
- wymagania związane z obsługą danych;
- plany wykonania;
- procedury związane z testem przyjęcia.

W kolejnych podrozdziałach znajduje się opis wyżej wymienionych pozycji.

Diagram związków jednostek

Diagram związków jednostek (E-R) przedstawia sposób powiązania ze sobą wszystkich elementów składających się na aplikację. Diagram ten jest kluczowym elementem pozwalającym zrozumieć cele stawiane systemowi. Pomaga też w identyfikacji punktów połączeń z innymi aplikacjami oraz zapewnia spójność bazy w odniesieniu do całego przedsiębiorstwa. Zasady związane z logicznym modelowaniem diagramów związków jednostek opisano w rozdziale 1.

Diagram fizycznego rozkładu bazy danych

Diagram fizycznego rozkładu bazy danych przedstawia fizyczne tabele utworzone na podstawie jednostek oraz kolumn powstałych z definicji atrybutów w modelu logicznym. Narzędzia służące do wykonywania fizycznych diagramów bazy danych zazwyczaj umożliwiają też generowanie pleceń DDL, które pozwalają utworzyć wszystkie obiekty aplikacji. Wszystkie zasady związane z tworzeniem fizycznych diagramów bazy danych zostały opisane w rozdziale 1.

Dzięki fizycznemu diagramowi bazy danych możliwe jest wskazanie tabel, które są najczęściej używane w czasie przetwarzania transakcji. Można też określić, które tabele są używane łącznie podczas operacji wprowadzania danych oraz podczas zapytań. Wszystkie te informacje pozwalają na rozłożenie tabel (oraz ich indeksów) na odpowiednie dostępne fizyczne urządzenia w celu skutecznego zmniejszenia rywalizacji operacji wejścia-wyjścia (patrz rozdział 3. i 4.).

Wymagania dotyczące przestrzeni

Wymagania dotyczące przestrzeni dotyczą początkowej przestrzeni, jaką zajmować będą odpowiednie tabele oraz indeksy w bazie danych. Zalecenia dotyczące doboru właściwego rozmiaru tabel, klastrów oraz indeksów przedstawiono w podrozdziale „Rozmiary obiektów bazy danych” w dalszej części tego rozdziału.

Cele strojenia

Zmiany dokonywane w projekcie aplikacji mogą mieć znaczący wpływ na szybkość jej działania. Rozwiązania zastosowane podczas projektowania aplikacji oddziałują na późniejsze wysiłki związane ze strojeniem aplikacji. Z tego względu administrator bazy danych powinien także uczestniczyć w procesie projektowania bazy danych.

Określenie wymaganej wydajności pracy aplikacji oraz osiągnięcie tej wydajności musi być dokonane, zanim aplikacja trafi do użytkownika. Zwykle użytkownik oczekuje, że nowy system będzie pracował przynajmniej tak samo wydajnie, jak ten używany dotychczas. Administrator bazy danych i programiści muszą spełnić wszystkie wymagania użytkowników na akceptowalnym przez nich poziomie. Szacunkowe czasy odpowiedzi dla wszystkich najczęściej używanych elementów aplikacji muszą zostać zaaprobowane.

Ważne jest, aby w czasie tego procesu oszacować wszystkie ustawienia w dwojaki sposób: stawiając wymagania na umiarkowanym i na bardzo zawyżonym poziomie. Drugi z tych sposobów zakłada otrzymanie takich wyników, które w praktyce są rzadko spotykane z powodu sprzętowych i programowych powiązań ograniczających wydajność systemu. Jednak podejście takie pozwala skupić się na aspektach, które są naprawdę ważne pod względem wydajności zamiast na tych, które wykraczają poza podstawowy zakres systemu.

Wymagania związane z ochroną danych

Zespół tworzący aplikacje musi określić strukturę kont, które będą wykorzystywane podczas pracy tej aplikacji. Należy wziąć pod uwagę, czyją własnością są poszczególne obiekty bazy danych w aplikacji oraz sposób przekazywania uprawnień. Wszystkie role oraz uprawnienia powinny zostać jasno zdefiniowane. Wszystkie te elementy pozwalają na utworzenie optymalnej struktury kont oraz uprawnień w aplikacji produkcyjnej. W rozdziale 9. przedstawiono pełen opis systemu zabezpieczeń.

Niekiedy zachodzi konieczność utworzenia, poza zwykłymi kontami, także konta używanego podczas operacji wsadowych. Przykładowo, w odniesieniu do takiego konta można stosować właściwości związane z autologowaniem, podczas gdy pozostałe konta obsługiwane są w standardowy sposób. System ochrony danych musi obsługiwać obydwa typy użytkowników.

Podobnie, jak w przypadku realizacji wymagań związanych z przestrzenią, zaangażowanie administratora w planowanie zabezpieczeń związanych z ochroną danych jest szczególnie ważne. System zabezpieczeń powinien być zaprojektowany w sposób umożliwiający spełnienie wszystkich potrzeb związanych z aplikacją i powinien też mieścić się w ramach systemu zabezpieczeń bazy danych.

Wymagania związane z obsługą danych

Sposoby wprowadzania oraz wyszukiwania danych powinny zostać jasno zdefiniowane. Wszystkie sposoby wprowadzania danych powinny zostać przetestowane oraz zweryfikowane. Jakiekolwiek specjalne wymagania dotyczące archiwizacji danych powinny także zostać udokumentowane, ponieważ mogą one w znacznym stopniu zależeć od specyfiki aplikacji.

Należy także określić wymagania związane z archiwizacją i odtwarzaniem aplikacji. Wymagania te potem porównuje się z planem archiwizacji dla bazy danych (szczegóły dotyczące tego zagadnienia znajdują się w rozdziale10.). Wymagania dotyczące archiwizacji i odtwarzania bazy danych powinny zostać tak zmodyfikowane, aby uwzględniały one także potrzeby związane archiwizacją i odtwarzaniem aplikacji.

Plany wykonania

Plan wykonania uwzględnia poszczególne etapy wykonywania zapytania przez bazę danych. Jest on tworzony na podstawie poleceń `explain plan` lub `set autotrace`. Dokładniej opisano to w rozdziale 8. Sporządzanie planów wykonania dla najważniejszych zapytań w bazie danych może pomóc w planowaniu użycia indeksów oraz podczas strojenia aplikacji. Wygenerowanie ich przed rozpoczęciem użytkowania aplikacji w dużym stopniu upraszcza wysiłki związane ze strojeniem oraz z rozpoznawaniem potencjalnych problemów dotyczących wydajności pracy aplikacji. Pozwala to także na ułatwienie procesu przeglądania i oceniania kodu programu.

Procedury testu przyjęcia

Programiści oraz użytkownicy powinni bardzo jasno sprecyzować poziom funkcjonalności i wydajności aplikacji przed oddaniem jej do użytku. Założenia te powinny stać się podstawą procedur testowych wykonywanych na aplikacji podczas fazy testowania.

Procedury testowe powinny również określać sposób postępowania w przypadku niezgodności działania testowanej aplikacji z przyjętymi kryteriami. Powinny też jednoznacznie zdefiniować wszystkie warunki, jakim odpowiadać musi system przed udostępnieniem go końcowym użytkownikom. Istotne jest również określenie wymagań wobec mniej ważnych parametrów aplikacji. Rozdzielenie funkcjonalnych właściwości według wyznaczonych priorytetów pomaga w rozwiązywaniu ewentualnych, możliwych do przewidzenia, konfliktów oraz w przeprowadzeniu właściwych testów.

Nowe cechy wspierające zarządzanie procesem rozbudowy aplikacji

Od wersji Oracle8i istnieje możliwość używania dwóch nowych właściwości pomocnych w zarządzaniu procesem tworzenia aplikacji. Pierwsza z nich to *składowane plany wykonania* służące do migracji ścieżek wykonania pomiędzy instancjami. Drugą stanowi program Database Resource Manager przeznaczony do kontroli przydziału zasobów systemowych użytkownikom. Składowane plany wykonania oraz zarządzanie zasobami to ważne elementy w procesie zarządzania tworzeniem aplikacji. Program Database Resource Manager pozwala administratorom baz danych na pełniejsze kontrolowanie przydziałów zasobów systemowych, niż jest to możliwe za pomocą dostępnych narzędzi systemu operacyjnego.

Database Resource Manager

Program Database Resource Manager może być wykorzystywany do przydzielania zasobów systemowych różnym klasom użytkowników oraz zadań. Przykładowo, można przydzielić 75% dostępnych zasobów procesora użytkownikom pracującym na bieżąco w bazie danych, a 25% użytkownikom, którzy wykonują jedynie operacje wsadowe. Aby korzystać z programu Database Resource Manager trzeba utworzyć: plany zasobów, grupy użytkowników zasobów oraz wytyczne dotyczące planu zasobów. Skrypt *catm.sql* wywoływany z *catproc.sql* tworzy konieczne struktury dla programu Database Resource Manager.

Przed użyciem programu Database Resource Manager trzeba utworzyć „obszar przejściowy" dla swojej pracy. Do tego celu jest wykorzystywana procedura `CREATE_PENDING_AREA` pochodząca z pakietu `DBMS_RESOURCE_MANAGER`. Następnie przeprowadzić trzeba procedurę `VALIDATE_PENDING_AREA` w celu sprawdzenia poprawności nowych zestawów wytycznych oraz planów. Zmiany, jakie uzyskuje się w ten sposób, można przeglądać za pomocą polecenia `SUBMIT_PENDING_AREA` i usuwać za pomocą procedury `CLEAR_PENDING_AREA`. Procedury, które zarządzają obszarem przejściowym, nie wymagają żadnych parametrów wejściowych. Przykład użycia procedury `CREATE_PENDING_AREA` przedstawiono poniżej:

```
execute DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

Jeśli obszar przejściowy nie zostanie utworzony, otrzymamy następujący komunikat:

```
BŁĄD w linii 1:
ORA-29371: obszar przejściowy nie jest aktywny
ORA-06512: at "SYS.DBMS_RMIN", linia 249
ORA-06512: at "SYS.DBMS_RESOURCE_MANAGER", linia 26
ORA-06512: w linii 1
```

W razie otrzymania takiego zestawu komunikatów należy ponownie przeprowadzić procedurę CREATE_PENDING_AREA. Przykłady wykonania poleceń sprawdzających poprawność oraz umożliwiających przeglądanie parametrów obszaru przejściowego przedstawiono na końcu tego podrozdziału.

W celu utworzenia planu zasobów przeprowadza się procedurę CREATE_PLAN z pakietu DBMS_RESOURCE_MANAGER. System zarządzania zasobami jest nowością w Oracle8i, zatem składnia procedur jest oparta bardziej na PL/SQL, niż na SQL.

Składnia procedury CREATE_PLAN została przestawiona poniżej:

```
CREATE_PLAN
(plan                          IN VARCHAR2,
 comment                       IN VARCHAR2,
 cpu_mth                       IN VARCHAR2 DEFAULT 'EMPHASIS',
 max_active_sess_target_mth    IN VARCHAR2 DEFAULT
       'MAX_ACTIVE_SESS_ABSOLUTE',
 parallel_degree_limit_mth     IN VARCHAR2 DEFAULT
       'PARALLEL_DEGREE_LIMIT_ABSOLUTE')
```

Planowi nadaje się nazwę (zmienna plan) oraz komentarz (zmienna comment). Domyślną metodą przydzielania zasobów procesora będzie metoda emphasis. Zasoby będą mierzone w procentach. Poniższy przykład przedstawia tworzenie planu o nazwie DEVELOPERS (programiści):

```
Execute DBMS_RESOURCE_MANAGER.CREATE_PLAN -
(Plan => 'DEVELOPERS', -
 Comment => 'Programiści w bazie testowej');
```

Łącznik (-) podobnie jak w programie SQL*Plus pozwala na przeniesienie części polecenia do następnej linii.

Możliwość generowania planu zasobów i zarządzania nim oraz grupami użytkowników zasobów jest możliwa pod warunkiem posiadania podczas otwartej sesji uprawnień systemowych ADMINISTER_RESOURCE_MANAGER. Administratorzy posiadają takie uprawnienie nadane z opcją with admin option. Przydzielanie tego uprawnienia innym użytkownikom odbywa się za pomocą procedury GRANT_SYSTEM_PRIVILEGE z pakietu DBMS_RESOURCE_MANAGER_PRIVS. W przedstawionym poniżej przykładzie użytkownik MARTHA otrzymuje uprawnienia pozwalające na zarządzanie programem Database Resource Manager:

```
Execute DBMS_RESOURCE_MANAGER_PRIVS -
(grantee_name => 'Martha', -
 admin_option => TRUE);
```

Odbieranie uprawnień może być dokonane poprzez procedurę REVOKE_SYSTEM_PRIVILEGE z pakietu DBMS_RESOURCE_MANAGER.

Dzięki uprawnieniom ADMINISTER_RESOURCE_MANAGER można utworzyć grupę użytkowników zasobów. Robi się to za pomocą procedury CREATE_CONSUMER_GROUP z pakietu DBMS_RESOURCE_MANAGER. Składnię polecenia CREATE_CONSUMER_GROUP przedstawiono poniżej:

```
CREATE_CONSUMER_GROUP
(consumer_group   IN VARCHAR2,
 comment          IN VARCHAR2,
 cpu_mth          IN VARCHAR2 DEFAULT 'ROUND-ROBIN')
```

Teraz można przypisać użytkowników do grupy zasobów. Nazwa grupy powinna odpowiadać logicznemu podziałowi użytkowników w bazie danych. Poniższy przykład przedstawia polecenia tworzące dwie grupy. Pierwszą grupę stanowią programiści pracujący na bieżąco w bazie danych, a drugą programiści wykonujący operacje wsadowe:

```
execute DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP -
(Consumer_Group => 'Online_Developers', -
 Comment => 'Programiści pracujący na bieżąco');

Execute DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP -
(Consumer_Group => 'Batch_developers', -
 Comment => 'Programiści wykonujący operacje wsadowe');
```

Po utworzeniu planu oraz grup użytkowników zasobów należy ustalić wytyczne dotyczące planu oraz przypisać odpowiednich użytkowników do odpowiednich grup. W celu przypisania odpowiednich wytycznych dotyczących planu wykonuje się procedurę CREATE_PLAN_DIRECTIVE z pakietu DBMS_RESOURCE_MANAGER. Składnia polecenia CREATE_PLAN_DIRECTIVE przedstawiona została poniżej:

```
CREATE_PLAN_DIRECTIVE
    (plan                       IN VARCHAR2,
     group_or_subplan           IN VARCHAR2,
     comment                    IN VARCHAR2,
     cpu_p1                     IN NUMBER DEFAULT NULL,
     cpu_p2                     IN NUMBER DEFAULT NULL,
     cpu_p3                     IN NUMBER DEFAULT NULL,
     cpu_p4                     IN NUMBER DEFAULT NULL,
     cpu_p5                     IN NUMBER DEFAULT NULL,
     cpu_p6                     IN NUMBER DEFAULT NULL,
     cpu_p7                     IN NUMBER DEFAULT NULL,
     cpu_p8                     IN NUMBER DEFAULT NULL,
     max_active_sess_target_p1  IN NUMBER DEFAULT NULL,
     parallel_degree_limit_p1   IN NUMBER DEFAULT NULL)
```

Zmienne cpu w procedurze CREATE_PLAN_DIRECTIVE obsługują tworzenie wielopoziomowego przydziału zasobów procesora. Przykładowo, 75% wszystkich zasobów procesora można przydzielić grupie użytkowników pracujących na bieżąco w bazie danych (poziom 1). 50% z pozostałych zasobów przydzielić można drugiej grupie użytkowników (poziom 2). Pozostałą część zasobów można przydzielić pozostałym grupom na poziomie trzecim. Procedura CREATE_PLAN_DIRECTIVE obsługuje do ośmiu poziomów przydziału zasobów procesora.

Poniższy przykład przedstawia tworzenie wytycznych dla planu DEVELOPERS dla dwóch grup: programistów pracujących na bieżąco w systemie oraz programistów, którzy wykonują tylko operacje wsadowe:

```
execute DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE -
 (Plan => 'DEVELOPERS', -
  Group_or_subplan => 'Online_developers', -
  Comment => 'Programiści pracujący na bieżąco', -
  Cpu_p1 => 75, -
  Cpu_p2 => 0, -
  Parallel_degree_limit_p1 => 12);

execute DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE -
 (Plan => 'DEVELOPERS', -
  Group_or_subplan => 'Batch_developers', -
  Comment => 'Programiści wykonujący operacje wsadowe', -
  Cpu_p1 => 25, -
  Cpu_p2 => 0, -
  Parallel_degree_limit_p1 => 6);
```

Ograniczenie stopnia zrównoleglenia operacji wykonywanych przez użytkowników poszczególnych grup następuje nie tylko dzięki odpowiedniemu przydziałowi zasobów procesora, ale także dzięki wytycznym dotyczącym planu. W powyższym przykładzie grupie „programiści wykonujący operacje wsadowe" przypisano 6. stopień zrównoleglenia, podczas gdy grupie „programiści pracujący z bazą na bieżąco" — 12. stopień.

Aby przypisać użytkownika do określonej grupy użytkowników zasobów wykonuje się procedurę SET_INITIAL_CONSUMER_GROUP z pakietu DBMS_RESOURCE_MANAGER. Składnia procedury SET_INITIAL_CONSUMER_GROUP została przedstawiona poniżej:

```
SET_INITIAL_CONSUMER_GROUP
    (user           IN VARCHAR2,
     consumer_group IN VARCHAR2)
```

Jeśli użytkownik nie został nigdy przypisany do żadnej grupy poprzez procedurę SET_INITIAL_CONSUMER_GROUP, zostaje on automatycznie zapisany do grupy o nazwie DEFAULT_CONSUMER_GROUP.

Aby uaktywnić program w bazie danych należy w pliku *init.ora* podać nazwę planu zasobów dla danej instancji w parametrze RESOURCE_MANAGER_PLAN. Ogólny plan zasobów może zawierać plany podrzędne, możliwe jest zatem tworzenie warstw dla przydziału zasobów w instancji. Jeśli parametr RESOURCE_MANAGER_PLAN nie zostanie określony, zarządzanie zasobami w instancji za pomocą programu Resource Manager nie będzie możliwe.

Można dynamicznie zmieniać wykorzystywanie różnych planów przydziału zasobów przez instancję poprzez klauzulę set initial_consumer_group w poleceniu alter system. Przykładowo, można utworzyć jeden plan przydziału zasobów dla użytkowników pracujących tyko w ciągu dnia pracy (Daytime_users) a drugi dla użytkowników wykonujących operacje wsadowe (Batch_users). Można utworzyć procedurę, która uruchamia każdego dnia o godzinie 6:00 następujące polecenie:

```
alter system set initial_consumer_group = 'DAYTIME_USERS';
```

a o godzinie 18:00 polecenie, które zmieni grupę użytkowników:

```
alter system set initial_consumer_group = 'BATCH_USERS';
```

W ten sposób można zmieniać plan przydziału zasobów bez konieczności zatrzymywania instancji.

Kiedy używa się wielu planów przydziału zasobów, trzeba zwracać szczególną uwagę na unikanie przypadkowego uruchomienia niewłaściwego planu. Przykładowo, po zamknięciu i ponownym otworzeniu bazy danych trzeba upewnić się, że zadanie dokonujące zmian planów będzie wykonywane o odpowiedniej porze. Trzeba mieć na uwadze ewentualne skutki uruchomienia złego planu przydziału zasobów. Aby tego uniknąć, należy stosować jak najmniejszą liczbę różnych planów przydziału zasobów.

Aby uzupełnić informacje odnośnie procedur dotyczących planów przydziału zasobów przedstawione w tym podrozdziale, podano jeszcze kilka przykładów dodatkowych procedur pomocniczych. Procedura `UPDATE_PLAN` służy do modyfikacji istniejącego planu. Procedura `DELETE_PLAN` umożliwia usuwanie planu, a dzięki procedurze `DELETE_PLAN_CASCADE` można usuwać plan wraz z ewentualnymi planami podrzędnymi oraz wszystkie grupy użytkowników z nim związane. Aby usunąć bądź zmodyfikować grupy użytkowników zasobów stosuje się, odpowiednio, procedurę `DELETE_CONSUMER_GROUP` lub procedurę `UPDATE_CONSUMER_GROUP`. Wytyczne dotyczące planu przydziału zasobów można usuwać bądź też zmieniać za pomocą, odpowiednio, procedury `DELETE_PLAN_DIRECTIVE` lub procedury `UPDATE_PLAN_DIRECTIVE`.

Po dokonaniu zmian planu przydziału zasobów, grupy użytkowników lub wytycznych dotyczących planu należy zawsze sprawdzić i przetestować wykonane zmiany przed ich zastosowaniem. Aby przetestować wykonane zmiany, trzeba utworzyć przejściowy obszar roboczy za pomocą procedury `CREATE_PENDING_AREA` z pakietu `DBMS_RESOURCE_MANAGER`. Po dokonaniu planowanych zmian dokonuje się sprawdzenia działania systemu za pomocą polecenia `VALIDATE_PENDING_AREA`. Możliwe jest późniejsze przeglądanie przeprowadzonych zmian poprzez procedurę `SUBMIT_PENDING_AREA` lub usunięcie wszystkich zmian za pomocą procedury `CLEAR_PENDING_AREA`. Procedury te nie wymagają żadnych parametrów wejściowych. Poniżej przedstawiono przykład ich użycia:

```
execute DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
execute DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

Składowane plany wykonania zapytań

Jeżeli podczas pracy korzysta się z dostępu do kilku baz danych, zachodzi możliwość zmian ścieżki wykonania dla wydawanych zapytań. Ścieżki wykonania mogą zmieniać się z następujących powodów:

1. Poszczególne bazy danych mogą stosować różne optymalizatory (np. optymalizator kosztowy, optymalizator regułowy).
2. Poszczególne bazy danych mogą korzystać z różnych właściwości optymalizatorów.
3. Statystyki dla przeglądanych tabel mogą się różnić w zależności od bazy danych.

4. Częstotliwość zbierania statystyk w różnych bazach danych może być inna.
5. Bazy danych mogą się różnić wersjami.

Powyższe różnice mogą mieć negatywny wpływ na wykonywanie zapytań podczas przenoszenia bądź modyfikacji aplikacji. Aby zmniejszyć negatywny wpływ tych różnic na szybkość wykonywania zapytań, od wersji Oracle8i wprowadzono nową właściwość nazywaną składowanymi planami wykonania.

Składowany plan wykonania przechowuje zestawy wskazówek dla zapytania. Wskazówki te używane są podczas każdego wykonania zapytania. Stosowanie tych wskazówek zwiększa prawdopodobieństwo użycia przez zapytanie każdorazowo tych samych ścieżek wykonania. Wskazówki nie wymuszają określonych ścieżek wykonania (jedynie je wskazują), ale przyspieszają wykonanie zapytań.

Aby zapewnić sobie możliwość tworzenia wskazówek dla wszystkich zapytań, ustawia się parametr CREATE_STORED_OUTLINES w pliku *init.ora* na wartość TRUE. Wtedy wszystkie plany wykonania będą zachowane w domyślnej (DEFAULT) kategorii. Można także tworzyć własne kategorie poprzez ustawienie parametru CREATE_STORED_OUTLINES na wartość, która będzie zarazem nazwą kategorii. Przykład utworzenia takiej kategorii przedstawiono poniżej:

```
CREATE_STORED_OUTLINES = development
```

Efektem wykonania powyższego polecenia jest przechowywanie planów wykonania dla zapytań w kategorii `development`.

Aby mieć możliwość tworzenia nowych planów wykonania, trzeba posiadać uprawnienie systemowe `CREATE ANY OUTLINE`. W celu utworzenia nowego planu wykonania dla zapytania używa się polecenia `create outline`, jak pokazano w przykładzie poniżej:

```
create outline YTD_SALES
   for category Development
    on
select YEAR_to_Date_Sales
  from SALES
where region = 'SOUTH'
  and period = 1;
```

Jeśli nazwa planu jest nieokreślona, zostanie nadana mu nazwa systemowa.

Ustawienie parametru CREATE_STORED_OUTLINES na wartość TRUE spowoduje, że system sam będzie tworzył plany wykonania dla wydawanych zapytań. Polecenie `create outline` daje większą kontrolę nad tworzonymi planami.

Możliwe jest także tworzenie planów dla poleceń DML oraz dla poleceń typu `create table as select`.

Czasami zachodzi potrzeba modyfikacji już istniejących planów. Przykładowo, jest konieczna zmiana planu ze względu na zmiany w rozkładzie i rozmiarze danych. Zmiany istniejących planów przeprowadza się za pomocą klauzuli `rebuild` w poleceniu `alter outline`, zmieniając w ten sposób wskazówki podczas wykonywania zapytania:

```
alter outline YTD_SALES rebuild;
```

Nazwę planu zmienia się za pomocą klauzuli `rename` w poleceniu `alter outline`:

```
alter outline YTD_SALES to rename to YTD_SALES_REGION;
```

Klauzula `change category` pozwala na zmianę kategorii planu:

```
alter outline YTD_SALES_REGION change category to DEFAULT;
```

Spowodowanie, aby optymalizator wykorzystywał składowane plany, wymaga ustawienia parametru USE_STORED_OUTLINES w pliku *init.ora* na wartość TRUE lub przypisania do niego nazwy kategorii (np. `Development`). Kiedy wykorzystywanie składowanych planów jest uaktywnione, wszystkie zapytania wykonywane ze składowanymi planami będą mogły korzystać ze wskazówek wygenerowanych podczas ich tworzenia. Parametr USE_STORED_OUTLINES może także być ustawiany dla danej sesji przy pomocy polecenia `alter session`.

Do zarządzania składowanymi planami używamy pakietu `OUTLN_PKG`. Pakiet ten umożliwia:

- usuwanie nigdy nie używanych składowanych planów;
- usuwanie składowanych planów z danej kategorii;
- przenoszenie planów z jednej kategorii do drugiej.

Każdej z tych wymienionych cech odpowiada właściwa procedura z pakietu `OUTLN_PKG`. Aby usunąć nigdy nie używane składowane plany wykonuje się procedurę `DROP_UNUSED`:

```
execute OUTLN_PKG.DROP_UNUSED;
```

W celu usunięcia wszystkich planów z danej kategorii wykonuje się procedurę `DROP_BY_CAT`. W procedurze tej stosuje się jeden parametr wejściowy, jest nim nazwa kategorii. W poniższym przykładzie usuwane są wszystkie plany w kategorii `Development`:

```
Execute OUTLN_PKG.DROP_BY_CAT -
  (category_name => 'DEVELOPMENT');
```

Aby zmienić przypisane plany i przenieść je do innej kategorii, używa się procedury `UPDATE_BY_CAT`:

```
Execute OUTLN_PKG.UPDATE_BY_CAT -
 (old_category_name => 'DEVELOPMENT', -
  new_category_name => 'TESTS';
```

Aby usunąć określony plan, należy użyć polecenia `drop outline`.

Rozmiary obiektów bazy danych

Wybór właściwych rozmiarów dla obiektów bazy danych jest decyzją strategiczną. Programiści powinni oszacować wymagania dotyczące przestrzeni jeszcze przed utworzeniem pierwszego obiektu w bazie danych. W późniejszych fazach istnienia bazy istnieje możliwość dokonywania poprawek w celu doskonalszego spełniania wymagań dotyczących przestrzeni na podstawie aktualnych statystyk. W niniejszym podrozdziale przedstawiono metody pozwalające oszacować rozmiary tabel, indeksów oraz klastrów. Opisano tu także metody pozwalające na określenie właściwych wielkości parametrów `pctfree` oraz `pctused`.

Dlaczego właściwe rozmiary obiektów bazy danych są tak istotne?

Istnieją cztery powody, dla których należy dobierać odpowiednie rozmiary dla obiektów bazy danych:

1. Zmniejszenie późniejszych wysiłków związanych z zarządzaniem wymaganiami dotyczącymi wykorzystania przestrzeni.
2. Umożliwienie racjonalnej gospodarki przestrzenią.
3. Wyeliminowanie potencjalnych problemów związanych z operacjami wejścia-wyjścia.
4. Zwiększenie prawdopodobieństwa ponownego wykorzystania zwalnianych obszarów przez poszczególne segmenty.

Wszystkie wyżej wymienione cele są osiągalne dzięki metodologii opisanej w bieżącym podrozdziale. Metodologia ta jest oparta na wewnętrznych metodach systemu Oracle służących właściwemu przydziałowi przestrzeni dla obiektów bazy danych. Metody te oparte są raczej na przybliżeniach, niż na szczegółowych obliczeniach. Upraszcza to znacznie proces określania właściwych rozmiarów obiektów bazy danych.

W jaki sposób system Oracle dokonuje przydziału przestrzeni

Jeśli do obliczeń dotyczących wykorzystywania przestrzeni nie używa się wewnętrznych metod systemu Oracle, takie obliczenia przeważnie okazują się nieadekwatne do rzeczywistego wykorzystania przestrzeni. System Oracle dokonuje przydziału przestrzeni zgodnie z następującymi, wewnętrznymi zasadami:

- zawsze przydzielane są całe bloki, nigdy ich fragmenty;
- przydzielane są zestawy bloków, przeważnie po 5 bloków w zestawie;
- możliwy jest przydział mniejszych lub większych zestawów bloków, w zależności od ilości wolnej miejsca w przestrzeni tabel.

Można zastosować te wewnętrzne zasady podczas przydzielania przestrzeni. Przykładowo, podstawowy blok w bazie danych ma rozmiar równy 4 KB. W bazie danych zostały utworzone następujące tabele:

Nazwa tabeli	Parametr Initial	Parametr Next	Parametr Pctincrease
Mała_tabela	7K	7K	0
Średnia_tabela	103K	103K	20

Przed utworzeniem tych tabel administrator powinien starannie oszacować wymagania dotyczące przestrzeni za pomocą szczegółowych kalkulacji dostępnych w dokumentacji systemu Oracle. Kontynuując powyższy przykład poniżej przedstawiono sposób, w jaki system Oracle dokonuje przydziału przestrzeni.

Dla małej tabeli administrator ustalił rozmiar pierwszego obszaru na 7 KB. Jednak podstawowy blok bazy danych ma rozmiar 4 KB. System Oracle nie przydziela fragmentów bloków, zatem rozmiar tego obszaru zostanie zaokrąglony do 8 KB. Jednak 8 KB to tylko 2 bloki. Dla większości środowisk system Oracle ustala rozmiar obszaru początkowego jako 5 bloków, czyli 20 KB. Podobne kalkulacje system Oracle przeprowadzi przydzielając przestrzeń dla następnego obszaru, gdzie wartość określona w parametrze `next` wynosi znów 7 KB.

W przypadku średniej tabeli wartość parametru `initial` dla pierwszego obszaru została ustalona przez administratora na 103 KB. Wartość ta musi zostać tak zmodyfikowana, aby była wielokrotnością rozmiaru podstawowego bloku Oracle. Zatem system Oracle zaokrągli tę wartość do 104 KB (26 bloków). W tym punkcie system Oracle analizuje ilość wolnego miejsca w przestrzeni tabel. Idealnym rozwiązaniem byłoby zaokrąglenie do następnej wartości będącej wielokrotnością 5 bloków, czyli do 30 bloków. Jeśli w przestrzeni tabel znajduje się wolny obszar o rozmiarze pomiędzy 26 a 30 bloków, system Oracle może użyć tego obszaru dla średniej tabeli.

Podczas przydzielania średniej tabeli następnego obszaru system Oracle wykona podobną analizę i przydzieli obszar o wielkości 30 bloków (120 KB).

Przydział trzeciego obszaru średniej tabeli będzie wyglądał nieco inaczej. System Oracle nie posłuży się już parametrem `next`. Średnia tabela posiada parametr `pctincrease` ustawiony na wartość 20, dlatego też trzeci obszar będzie o 20% większy od drugiego obszaru. Jeśli system Oracle opierałby się na swych kalkulacjach, rozpocząłby proces przydziału przestrzeni dla trzeciego obszaru od wielkości 36 bloków (30 bloków * 1.2), następnie poszukiwałby wolnego obszaru o wielkości 40 bloków (wielokrotność 5 bloków). Jednakże system Oracle przyjmuje jako wartość początkową wartość określoną w parametrze `next` (103 KB), powiększa ją o 20% (wartość parametru `pctincrease`), co daje w efekcie 123,6 KB. Dla podstawowego bloku danych o wielkości 4 KB wynosi to 30,9 bloków, a po zaokrągleniu 31. Następnie ta wielkość jest zaokrąglana do kolejnej wielokrotności 5 bloków, czyli do 35 bloków.

Różnice w wielkościach obszarów określonych przez administratora i przydzielonych przez system Oracle przedstawiono poniżej:

Tabela	Initial	Next	Pctincrease	Pierwszy obszar	Drugi obszar	Trzeci obszar
Mała_tabela	7K	7K	0	20K	20K	20K
Średnia_tabela	103K	103K	20	120K	120K	140K

Rezultaty te mogą zniechęcić administratora, który przeprowadza kalkulacje dotyczące wykorzystania przestrzeni. Mała tabela potrzebuje jedynie 14 KB dla swych pierwszych dwóch obszarów, a system Oracle przydzielił jej 40 KB. Średnia tabela potrzebuje zaś jedynie 206 KB dla swych pierwszych dwóch obszarów, a system Oracle przydzielił jej 240 KB. Poza tym trzeci obszar miał być większy od drugiego o 20% (parametr `pctincrease`), a przydzielona mu przestrzeń jest większa jedynie o 16,6%.

Jak wynika z powyższego przykładu, podczas szacowania rozmiarów obiektów bazy danych zawsze trzeba brać pod uwagę sposób, w jaki system Oracle przydziela wolną przestrzeń. Jeśli wszystkie te czynniki zostaną uwzględnione podczas takich kalkulacji, ich wyniki będą tylko minimalnie różniły się od wartości obliczonych przez system Oracle.

Przed rozpoczęciem dobierania rozmiarów obszarów należy jeszcze rozważyć wpływ rozmiaru obszarów na wydajność bazy danych.

Wpływ rozmiaru obszarów na wydajność bazy danych

Zmniejszenie liczby obszarów w tabeli nie daje bezpośrednich korzyści związanych z wydajnością pracy bazy danych. W niektórych sytuacjach (np. w przypadku równoległego przetwarzania zapytań) większa liczba obszarów w tabelach może znacząco zmniejszyć rywalizację operacji wejścia-wyjścia i poprawić w ten sposób wydajność. Poza liczbą obszarów w tabeli trzeba również odpowiednio dobierać ich rozmiary.

System Oracle odczytuje dane z tabel na dwa sposoby: poprzez identyfikator wiersza (łączy się to przeważnie z natychmiastowym dostępem do indeksu) lub poprzez pełny przegląd tabeli. Jeśli dane są odczytywane poprzez identyfikator wiersza, liczba obszarów w tabeli nie wpływa na wydajność odczytu. System Oracle odczytuje każdy wiersz (określony przez jego identyfikator) i zwraca dane.

Jeśli dane odczytywane są poprzez pełny przegląd tabeli, wielkość obszarów może mieć wpływ na wydajność pracy bazy danych. Podczas odczytywania danych w czasie pełnego przeglądu tabeli system Oracle odczytuje wiele bloków naraz. Liczba bloków, które odczytywane są jednocześnie, jest określana przez parametr DB_FILE_MULTIBLOCK_READ_COUNT w pliku *init.ora*. Maksymalna liczba jednocześnie odczytywanych bloków jest ograniczona przez rozmiar systemowego buforu operacji wejścia-wyjścia. Przykładowo, jeśli podstawowy blok bazy danych ma rozmiar 4 KB, a rozmiar systemowego buforu wejścia-wyjścia wynosi 64 KB, wtedy możliwy jest jednoczesny odczyt do 16 bloków podczas jednego odczytu w czasie pełnego przeglądu tabeli. W takim przypadku ustawianie dla parametru DB_FILE_MULTIBLOCK_READ_COUNT wartości większej niż 16, nie przyniesie żadnych korzyści związanych z poprawą wydajności w czasie pełnego przeglądu tabeli.

Należy tak dobierać rozmiary obszarów, aby wykorzystać właściwości systemu Oracle związane z odczytem wielu bloków danych podczas jednego odczytu w czasie pełnego przeglądu tabeli. Tak więc, jeśli wielkość systemowego buforu wejścia-wyjścia wynosi 64 KB, rozmiary obszarów powinny być wielokrotnością 64 KB.

Jako przykład niech posłuży tabela, która posiada 10 obszarów. Każdy z nich ma rozmiar 64 KB. Systemowy bufor wejścia-wyjścia także ma rozmiar 64 KB. W celu wykonania pełnego przeglądu tej tabeli system Oracle musi wykonać 10 odczytów. Jeśli dane zostałyby skupione w jednym obszarze o rozmiarze 640 KB, system Oracle także musiałby wykonać 10 odczytów. Widać z tego, że skupianie danych w pojedynczych większych obszarach nie przynosi poprawy wydajności.

Jeżeli rozmiary obszarów nie są wielokrotnością rozmiaru systemowego buforu wejścia-wyjścia, wtedy liczba wykonywanych przez bazę danych odczytów może się zwiększyć. Przedstawiono to na poniższym przykładzie. Dla tej samej tabeli o rozmiarze 640 KB można stworzyć 8 obszarów po 80 KB każdy. Aby odczytać pierwszy obszar, system Oracle musi wykonać dwa odczyty: pierwszy odczytuje 64 KB a drugi pozostałe 16 KB. W celu odczytania wszystkich obszarów tej tabeli system Oracle musi wykonać po dwa odczyty na każdy obszar, czyli w sumie 16 odczytów. W tym przypadku zmniejszenie liczby obszarów z 10 do 8 zwiększyło liczbę odczytów o 60%.

W celu uniknięcia takich sytuacji podczas doboru rozmiarów obszarów należy kierować się jedną z dwóch poniższych strategii:

1. Tworzyć obszary, których rozmiary są znacząco większe od rozmiaru systemowego buforu wejścia-wyjścia. Jeśli uda się to osiągnąć, liczba dodatkowych odczytów jest bardzo niewielka. Dzieje się tak nawet, jeśli rozmiar obszaru nie jest wielokrotnością rozmiaru systemowego buforu wejścia-wyjścia.
2. Tworzyć obszary, których rozmiar jest wielokrotnością rozmiaru systemowego buforu wejścia-wyjścia.

Jeśli rozmiar systemowego buforu wejścia-wyjścia wynosi 64 KB, wtedy rozmiary obszarów powinny wynosić np: 64 KB, 128 KB, 192 KB, 256 KB itd. W następnym podrozdziale znajdują się dalsze informacje dotyczące optymalnego doboru rozmiarów obszarów.

Zwiększenie liczby ponownie używanych obszarów

Kiedy segment danych jest usuwany, jego obszary są dodawane do puli dostępnych, wolnych obszarów. Mogą one być wykorzystane przez inne segmenty. Jeśli obszary są wymiarowane w sposób konsekwentny, istnieje większe prawdopodobieństwo, że po zwolnieniu zostaną one ponownie wykorzystane przez system Oracle. Rezultatem tego będzie bardziej efektywne wykorzystanie miejsca w przestrzeni tabel.

Brak konsekwentnego wymiarowania obszarów może spowodować konieczność większego nakładu czasu przeznaczonego na zarządzanie wolną przestrzenią (np. defragmentacja przestrzeni tabel). Przykładowo, jeśli utworzy się tabelę z początkowym obszarem o wielkości 100 KB, system Oracle przydzieli 100 KB dla tego obszaru. Po usunięciu tej tabeli jej obszar zostanie oznaczony jako wolny. Zgodnie z tym, co napisano w rozdziale 4., system Oracle automatycznie łączy sąsiadujące wolne obszary, jeśli parametr `pctincrease` dla danej przestrzeni tabel został ustawiony na wartość różną od zera. Kontynuując przykład z poprzedniego podrozdziału założono, że w sąsiedztwie tego wolnego obszaru o wielkości 100 KB nie występują żadne inne wolne obszary. Otoczony jest on jedynie przez obszary zawierające dane. Jeśli zajdzie potrzeba

przydzielenia przestrzeni dla kolejnego segmentu, system Oracle wykorzysta w tym celu ten wolny obszar. Przykładowo, jeśli ten nowy segment będzie potrzebował dwóch obszarów o wielkości 60 KB każdy, system Oracle wybierze pierwsze 60 KB ze 100 KB wolnego obszaru dla nowego segmentu, pozostawiając pozostałe 40 KB. W wyniku tego działania 40% wolnej przestrzeni omawianego obszaru zostanie niewykorzystane.

Aby unikać tak niedoskonałego gospodarowania przestrzenią, powinno się używać takich zestawów obszarów, których rozmiary spełniają następujący warunek: *każdy rozmiar większego obszaru powinien być całkowitą wielokrotnością rozmiaru każdego mniejszego obszaru*.

Proszę rozważyć pierwsze 6 wartości rozmiarów obszarów, które zostały przedstawione w poprzednim podrozdziale:

64 KB, 128 KB, 192 KB, 256 KB, 320 KB, 384 KB

Okazuje się, że:

- 64 KB to wartość bazowa;
- 128 KB jest całkowitą wielokrotnością 64 KB, wartość ta spełnia więc powyższy warunek;
- 192 KB jest wartością, która nie jest całkowitą wielokrotnością 126 KB, a więc nie spełnia wymaganego warunku;
- 256 KB spełnia ten warunek;
- 320KB oraz 384KB nie są całkowitymi wielokrotnościami 256KB, a więc nie spełniają powyższego warunku.

Natychmiast można zauważyć, że każdy następny rozmiar obszaru musi być dwa razy większy od poprzednigo. Zatem wartości, które spełniają powyższy warunek to:

64 KB, 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB itd.

Dobierając rozmiary obszarów w sposób opisany powyżej zmniejsza się w znaczny sposób potencjalne problemy związane z operacjami wejścia-wyjścia oraz zwiększa się prawdopodobieństwo ponownego wykorzystania zwalnianych obszarów.

Ostatnia przeszkoda

Zgodnie z tym co napisano wcześniej, system Oracle zazwyczaj zaokrągla liczbę bloków przydzielanych do danego obszaru do wielokrotności liczby 5. Dla bloku o rozmiarze 4 KB lista możliwych rozmiarów obszarów (mierzonych w blokach) wygląda następująco:

16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192

Żadna z tych wartości nie jest podzielna przez 5, więc system Oracle zaokrągli je do wartości:

20, 35, 65,130, 260, 525, 1025, 2050, 4100, 8195

Co prawda, nie jest to zgodne z wyszczególnionymi powyżej zasadami dotyczącymi poprawnego doboru rozmiarów obszarów, jednak system Oracle nie zawsze zaokrągla te wartości. Podczas procesu przydziału przestrzeni system Oracle przeszukuje przestrzenie tabel w celu znalezienia wolnych obszarów. Jeśli znajdzie on obszar zawierający np. 32 bloki i obszar ten będzie dostępny, wtedy system użyje go zamiast obszaru o rozmiarze 35 bloków. Ponadto pamiętać należy, że większy rozmiar obszaru pozwala na zmniejszenie wpływu jego zaokrąglania na wydajność odczytów.

Oszacowanie wymaganej przestrzeni dla tabel nieklastrowych

Aby poprawnie oszacować ilość wymaganej przestrzeni dla tabeli, jest konieczna znajomość czterech wartości:

- rozmiar bloku danych w bazie danych;
- wartość dla parametru `pctfree` dla danej tabeli;
- średnią długość wiersza;
- spodziewaną liczbę wierszy w tabeli.

Dokładne obliczenie wymaganej ilości przestrzeni dla danej tabeli wymaga jeszcze więcej informacji (np. liczba kolumn). Jednak dla przybliżonego oszacowania powyższe informacje będą zupełnie wystarczające.

Rozmiar bloku danych w bazie danych określa parametr DB_BLOCK_SIZE w pliku *init.ora*. Rozmiaru tego nie można już zmienić po utworzeniu bazy danych. Jedynym sposobem zmiany rozmiaru bloku danych jest ponowne utworzenie bazy danych (poprzez polecenie `create database`) i zaimportowanie danych wcześniej wyeksportowanych.

Każdy blok bazy danych posiada obszar używany jako nagłówek bloku. Szacunkowa wielkość nagłówka dla bloku wynosi około 90 bajtów. Dlatego też dostępna przestrzeń w bloku bazy danych przedstawia się następująco:

Rozmiar bloku bazy danych (w bajtach)	Dostępna przestrzeń (w bajtach)
2 048	1 958
4 096	4 006
8 192	8 102

Część przestrzeni musi pozostać wolna, dostępna dla modyfikowanych wierszy, które wcześniej zostały wstawione do bloku. Parametr `pctfree` na poziomie definicji tabeli określa rozmiar wolnej przestrzeni, która jest niedostępna dla operacji wstawiania (`insert`). Na podstawie tego parametru można wyliczyć ilość przestrzeni zarezerwowanej dla modyfikowanych wierszy. Odejmując tę wartość od całości dostępnej przestrzeni w bloku otrzymuje się przestrzeń dostępną dla wstawianych wierszy.

Przykładowo, dla bloku o rozmiarze 4 KB i tabeli, dla której parametr `pctfree` ustawiono na 10, ilość dostępnej przestrzeni wylicza się w następujący sposób:

```
4006 bajtów - (0.1*4006 bajtów) = 3605,4 bajtów, zaokrąglamy do 3605 bajtów.
```

W każdym bloku danych w tym przykładzie dostępnych jest 3605 bajtów dla nowych rekordów.

Następnie należy oszacować średnią długość wiersza. Przybliżona długość pola typu DATE wynosi 8 bajtów, a przybliżona długość pola typu NUMBER 4 bajty. Dla kolumn typu VARCHAR2, należy oszacować długość w zależności od aktualnie przechowywanych danych w kolumnach.

Te szacunkowe obliczenia nie zawierają dodatkowego narzutu związanego z nagłówkiem kolumny. W rzeczywistości kolumna typu DATE przechowuje 7 bajtów, a kolumna typu NUMBER przeważnie 3 bajty.

Przykładowo, tabela zawiera 10 kolumn oraz oszacowana średnia długość wiersza jest równa 60 bajtów. Ponieważ na jeden blok przypada 3605 bajtów, liczba wierszy na jeden blok wynosi:

```
3605 bajtów na jeden blok / 60 bajtów na jeden wiersz = 6 wierszy na blok.
```

Teraz trzeba oszacować liczbę wierszy w tabeli. Jeśli przykładowa tabela zawiera 25 000 wierszy, wtedy liczba bloków wyniesie:

```
25 000 wierszy / 6 wierszy na blok = 4166 bloków.
```

Tabela będzie potrzebowała w przybliżeniu 4166 bloków. Jednak liczba ta nie odpowiada żadnemu z wymienionych wcześniej zalecanych rozmiarów dla obszarów. W takiej sytuacji można:

1. Stworzyć początkowy obszar o rozmiarze 16 MB (4096 bloków) oraz następny obszar o rozmiarze 512 KB (128 bloków).
2. Jeśli istnieje dostępna przestrzeń oraz przewidywany jest rozrost tabeli w przyszłości, można stworzyć początkowy obszar o rozmiarze 32 MB.

W przypadku zastosowania pierwszego rozwiązania przydzielona przestrzeń (4224 bloki) różnić się będzie od tej wcześniej oszacowanej o zaledwie 1%. Tabela będzie posiadała obszary o odpowiednich wymiarach pod względem zarówno wydajności pracy bazy, jak pod względem możliwości ich powtórnego użycia.

W następnym podrozdziale opisano sposoby szacowania ilości potrzebnej przestrzeni dla indeksów, następnie przedstawiono szczegółowe obliczenia dotyczące tabel, indeksów oraz klastrów. W praktyce szczegółowe obliczenia nie są konieczne. Użycie efektywnie oszacowanych rozmiarów obszarów w dużym stopniu upraszcza zarządzanie przestrzenią.

Oszacowanie wymaganej przestrzeni dla indeksów

Proces szacowania rozmiarów dla indeksów przebiega równolegle z procesem szacowania rozmiarów dla tabel. Opisana w tym podrozdziale metoda nie określa dokładnych wymagań związanych z przydziałem przestrzeni, ale jest pomocna w szybkiej ocenie wymagań związanych z przestrzenią. Ułatwia również dopasowanie tych wymagań do zestawów standardowych rozmiarów obszarów. W celu oszacowania wymaganej ilości przestrzeni dla indeksu należy znać cztery wartości:

- rozmiar bloku danych w bazie danych;
- wartość dla parametru `pctfree` dla danego indeksu;
- średnią długość indeksu;
- spodziewaną liczbę wpisów w indeksie.

Rozmiar bloku danych w bazie danych określa parametr DB_BLOCK_SIZE w pliku *init.ora*. Każdy blok bazy danych posiada obszar używany jako nagłówek bloku. Można oszacować wielkość nagłówka dla indeksu na 161 bajtów. Dlatego też ilość dostępnej przestrzeni w bloku bazy danych przedstawia się następująco:

Rozmiar bloku bazy danych (w bajtach)	Dostępna przestrzeń (w bajtach)
2 048	1 887
4 096	3 935
8 192	8 031

Część dostępnej przestrzeni musi pozostać wolna w zależności od wartości parametru `pctfree` dla indeksu. Jednakże wartości indeksu nie są często zmieniane. Dlatego wartość parametru `pctfree` dla indeksów jest ustawiana najczęściej na wartość poniżej 5. Na podstawie tego parametru można wyliczyć ilość przestrzeni zarezerwowanej dla modyfikowanych indeksów. Odejmując tę wartość od całości dostępnej przestrzeni w bloku, otrzymuje się ilość przestrzeni dostępnej dla wstawianych indeksów.

Przykładowo, dla bloku danych o rozmiarze 4 KB oraz dla indeksu, dla którego wartość parametru `pctfree` jest równa 2, dostępna przestrzeń wynosi:

```
3935 bajtów - (0.02*3935 bajtów) = 3856.3 bajtów, w zaokrągleniu 3856 bajtów.
```

W każdym bloku danych w tym przykładzie dostępnych jest 3856 bajtów dla nowych indeksów.

Następnie trzeba oszacować średnią długość wiersza dla indeksu. Jeśli indeks jest indeksem złożonym, należy oszacować długość każdej kolumny i zsumować je wszystkie razem, aby otrzymać całkowitą długość wiersza indeksu. Przybliżona długość pola typu DATE wynosi 8 bajtów, a przybliżona długość pola typu NUMBER 4 bajty. Długość kolumn typu VARCHAR2 szacuje się w zależności od danych przechowywanych aktualnie w kolumnach.

Te szacunkowe obliczenia nie zawierają dodatkowego narzutu związanego z nagłówkiem kolumny. W rzeczywistości dla typu DATE przechowane jest 7 bajtów, a dla typu NUMBER przeważnie 3 bajty.

Przykładowo, dany indeks istnieje na trzech kolumnach, a jego szacunkowa średnia długość wiersza wynosi 17 bajtów. Ponieważ na jeden blok przypada 3856 bajtów, liczba wpisów indeksowych na jeden blok wynosi:

```
3856 bajtów na blok / 17 bajtów na wpis = 226 wpisów na blok.
```

Teraz należy oszacować liczbę wpisów w indeksie. Jeśli przykładowy indeks zawiera 25 000 wpisów, wtedy liczba bloków wyniesie:

```
25 000 wierszy / 226 wpisów na blok = 111 bloków.
```

Dany indeks będzie wymagał w przybliżeniu 111 bloków. Jednakże rozmiar ten (444 KB) nie odpowiada żadnemu z wymienionych wcześniej zalecanych rozmiarów obszarów. W takiej sytuacji można:

1. Stworzyć początkowy obszar o rozmiarze 256 KB i następny obszar o rozmiarze 64 KB. Parametr `minextent` ustawić na 4, a parametr `pctincrease` na 0 (112 bloków).
2. Jeśli istnieje dostępna przestrzeń oraz jeśli przewidywany jest rozrost tabeli, utworzyć można początkowy obszar o rozmiarze 512 KB (128 bloków).

W przypadku zastosowania pierwszego rozwiązania przydzielona przestrzeń (112 bloków) różnić się będzie od tej wcześniej oszacowanej zaledwie o 1 blok. Jeśli zastosowane zostanie drugie rozwiązanie, zostanie przydzielone o 13% więcej przestrzeni, niż oszacowane zostało to wcześniej. Dzięki tym rozwiązaniom można utworzyć indeks, którego obszary posiadają wymiary odpowiednie pod względem zarówno wydajności pracy bazy, jak pod względem możliwości ich powtórnego użycia.

W następnych podrozdziałach zamieszczono szczegółowe obliczenia dotyczące wymagań związanych z przydziałem przestrzeni dla tabel, indeksów oraz klastrów. Można użyć tych obliczeń, aby porównać je z obliczeniami szacunkowymi. Przed dokonaniem szczegółowych obliczeń powinno się zawsze przeprowadzić wstępne obliczenia szacunkowe. Bez względu na metodę przydziału przestrzeni trzeba upewnić się, że rozmiary obszarów odpowiadają standardowym rozmiarom obszarów przyjętych dla naszej bazy danych.

Uwagi na temat przybliżonych obliczeń

System Oracle dostarcza niezmiernie dużą liczbę sposobów dokonywania szczegółowych obliczeń, które należy przeprowadzić w celu określenia rozmiarów dla tabel oraz indeksów. Jednak tak wysoki poziom szczegółowości tych obliczeń jest zbyteczny, ponieważ tę wyliczoną — wymaganą przestrzeń zawsze należy dodatkowo powiększyć o 10 do 20%.

Wykonując więc te szczegółowe obliczenia należy brać pod uwagę, że będą one obarczone przynajmniej 10% błędem.

Błąd w obliczeniach szacunkowych wynika ze sposobu, w jaki system Oracle zarządza przestrzenią już po utworzeniu obiektów. Podczas wprowadzania, modyfikacji bądź usuwania wierszy system Oracle zarządza przestrzenią wewnątrz bloków bazy danych. Sposób, w jaki to następuje, powoduje, że przestrzeń przydzielana przez administratora na etapie planowania bazy jest przeważnie mniejsza od wartości optymalnej. W rezultacie tego istnieje potrzeba jej zwiększenia — w niektórych przypadkach nawet podwojenia.

Ze względu na dynamiczny charakter procesu zarządzania przestrzenią w systemie Oracle dokładne obliczenia są wiarygodne jedynie w czasie ładowania danych statycznych do statycznych tabel. Ponieważ dotyczy to tylko niewielkiej liczby istniejących tabel, obliczenia umieszczone w tym rozdziale pozwalają jedynie na przybliżenie, a nie dokładne określenie potrzeb związanych z przestrzenią. We wszystkich przypadkach różnica pomiędzy obliczeniami dokładnymi a przybliżonymi wynosi mniej niż 5% — co zgodne jest z przyjętą granicą błędu założoną przez system Oracle.

Obliczenia wymaganej przestrzeni dla tabel nieklastrowych

Oprócz początkowych wymagań związanych z przestrzenią należy też oszacować procentowy wzrost liczby rekordów w każdej tabeli w stosunku rocznym. W pewnych przypadkach powinno się także określić maksymalną liczbę rekordów w tabeli.

Jeżeli znane są już definicje kolumn tabeli oraz rozmiary danych, można przejść do obliczeń związanych z wymaganiami dotyczącymi składowania. Są to rozważania teoretyczne, ponieważ rzeczywiste rozmiary danych oraz długość wierszy nie są do końca znane przed utworzeniem tabeli. Ważne jest, aby przykładowe dane dostępne były podczas wykonywania obliczeń, co pozwoli na otrzymanie dokładniejszych rezultatów.

W pierwszej kolejności oblicza się ilość przestrzeni zajmowanej przez nagłówek bloku. Nagłówek bloku służy do zarządzania danymi w bloku. Rozmiar nagłówka bloku wynosi w przybliżeniu 90 bajtów. Jeśli używa się bloku o rozmiarze 2 KB, pozostaje 1958 wolnych bajtów. Dla bloku o rozmiarze 4 KB pozostaje 4006 wolnych bajtów:

```
4096 - 90 = 4006 dostępnych bajtów.
```

Następnym czynnikiem, jaki należy wziąć pod uwagę, jest parametr `pctfree` dla danej tabeli. Mnożąc wartość tego parametru przez ilość wolnej przestrzeni otrzymuje się ilość przestrzeni zarezerwowanej dla operacji modyfikacji wierszy.

Jeśli wartość dla `pctfree` jest równa 10, wtedy mnożąc dostępną przestrzeń przez 0.1 otrzymuje się:

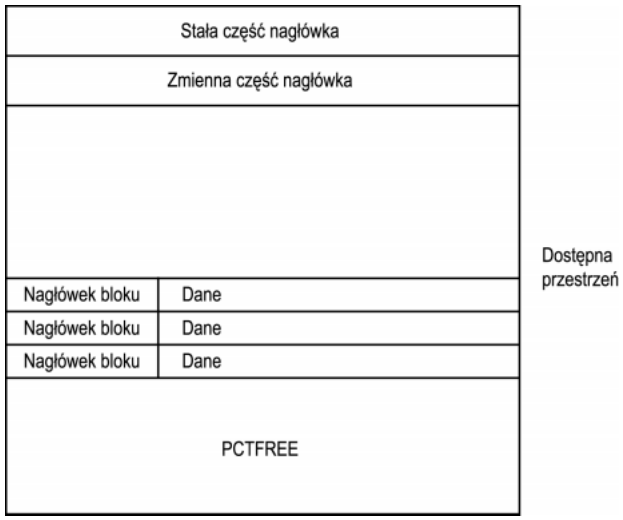
```
4006*(pctfree/100) = 4006*0.1 = 401 (po zaokrągleniu).
```

401 bajtów z dostępnej wolnej przestrzeni będzie zarezerwowanych dla modyfikowanych wierszy. Pozostała dostępna wolna przestrzeń wynosi:

```
4006 - 401 = 3605 dostępnych bajtów.
```

Po dokonaniu obliczeń z 4096 bajtów w bloku pozostaje 3605. Są one dostępne dla składowanych wierszy (patrz rysunek 5.1).

Następnie oblicza się ilość przestrzeni zajmowanej przez dany wiersz. W tym celu trzeba oszacować średnią długość wiersza. Średnia długość wiersza to całkowita średnia długość wszystkich wartości w wierszu. Jeśli żadne dane nie są dostępne, można oszacować aktualną średnią długość wartości w kolumnie. Nie używa się całej długości kolumny, jeśli dane nie wypełniają jej w całości.



Rysunek 5.1. *Rozdział przestrzeni w blokach*

Przykładowo, tabela zawiera trzy kolumny, wszystkie są typu VARCHAR2(10). Średnia długość wiersza przekracza 30, a jego aktualna długość jest zależna od ilości danych aktualnie przechowywanych w danym wierszu.

Jeżeli istnieje dostęp do przykładowych danych, wtedy można użyć funkcji `VSIZE` w celu określenia aktualnej średniej długości wiersza. Kontynuując przykład tabeli o trzech kolumnach, poniżej przedstawiono odpowiednie zapytanie:

```
select AVG(NVL(VSIZE(Column1),0)+
       AVG(NVL(VSIZE(Column2),0)+
       AVG(NVL(VSIZE(Column3),0)) Średnia_długość_wiersza
from TABLENAME;
```

W przykładzie tym została określona średnia długość każdej kolumny, a następnie na tej podstawie określono średnią długość wiersza.

Jeżeli tabela jest bardzo duża, wykonanie powyższego zapytania może zająć dużo czasu. Alternatywnie można wykorzystać polecenie `analyze table`:

```
analyze table TABLENAME compute statistics;
```

Dla bardzo dużych tabel można także użyć klauzuli `estimate statistics`:

```
analyze table TABLENAME estimate statistics;
```

Po przeprowadzeniu analizy dla tabeli związane z nią statystyki odnaleźć można w słowniku danych:

```
select Avg_Row_Len
from USER_TABLES
where Table_Name = 'TABLENAME';
```

Wartość `Avg_Row_Len` (średnia długość wiersza) uwzględnia też dodatkowe bajty przeznaczone na nagłówki kolumn.

Dla danej przykładowej tabeli, złożonej z trzech kolumn, założono, że średnia długość wiersza wynosi 24 bajty. Do całości dodać należy 1 bajt na każdą kolumnę w tabeli, czyli razem 27 bajtów na wiersz. Jeżeli tabela posiada kolumny, które zawierają dane o długości przekraczającej 250 znaków, dodać należy dodatkowy bajt dla każdej kolumny. Dodać też trzeba 3 bajty dla nagłówka wiersza.

```
Przestrzeń używana przez jeden wiersz = Średnia długość wiersza
                                        + 3
                                        + liczba kolumn
                                        + liczba długich kolumn.
```

Dla danej, przykładowej tabeli przestrzeń używana przez jeden wiersz wynosi:

```
Przestrzeń używana przez jeden wiersz = 24
                                        +3
                                        +3
                                        +0
                                      = 30 bajtów na wiersz.
```

Mając do dyspozycji 3605 bajtów oraz obliczone 30 bajtów na wiersz, można w jednym bloku umieścić 120 wierszy.

```
Liczba wierszy na blok = TRUNC(3605 wolnych bajtów/30 bajtów na wiersz)
                       = 120 wierszy na blok.
```

Ponieważ w jednym bloku mieści się 120 wierszy, można oszacować liczbę potrzebnych bloków. Jak zauważono w poprzednim podrozdziale, są to jedynie przybliżone obliczenia wymagań związanych z przestrzenią w tabeli. Wymagania te się zwiększają podczas manipulacji rekordami w tabeli. Duża liczba przeprowadzanych operacji modyfikacji (`update`) oraz usuwania (`delete`) wymaga większej ilości przestrzeni.

Obliczanie właściwej wartości dla pctfree

Odpowiednia wartość dla `pctfree` musi być określona dla każdej tabeli. Wartość ta określa procentowo ilość zarezerwowanej wolnej przestrzeni w bloku danych. Przestrzeń ta jest wykorzystywana w momencie, kiedy przechowywany wiersz na skutek modyfikacji powiększa się lub też podczas modyfikacji pól, które poprzednio posiadały wartości nieokreślone (NULL).

Nie ma jednej standardowej wielkości dla `pctfree`, odpowiedniej dla wszystkich tabel w bazie danych. Ze względu na to, że `pctfree` łączy wszystkie występujące sposoby modyfikacji wierszy w aplikacji, określenie jego wielkości jest prostym procesem. Dzięki `pctfree` możliwa jest kontrola liczby rekordów przechowywanych w bloku oraz w tabeli. W celu poprawnego określenia wartości dla `pctfree` jest potrzebna znajomość liczby wierszy w bloku. Zgodnie z tym, co przedstawiono w poprzednim podrozdziale, można w tym celu użyć polecenia `analyze`. W poniższym przykładzie w poleceniu `analyze` użyto klauzuli `compute statistics` w celu wygenerowania statystyk dla danej tabeli. Statystyki te dostępne są w słowniku danych.

```
analyze table TABLENAME compute statistics;
```

Jeśli używa się optymalizatora regułowego, trzeba usuwać statystyki po przeprowadzeniu obliczeń związanych z wartością dla pctfree.

Po dokonaniu analizy tabeli jej wyniki znaleźć można w perspektywie USER_TABLES w kolumnach Num_Rows (liczba wierszy) oraz Blocks (liczba bloków). Dzieląc liczbę wierszy przez liczbę bloków otrzymujemy liczbę wierszy przechowywanych w bloku, jak pokazano poniżej:

```
select Num_Rows,                  /*liczba wierszy*/
       Blocks,                    /*liczba bloków*/
       Num_Rows/Blocks            /*liczba wierszy w bloku*/
 from USER_TABLES
where Table_Name = 'TABLENAME';
```

Kiedy liczba wierszy przypadająca na jeden blok jest już znana, należy dokonać modyfikacji rekordów w tabeli, symulując w ten sposób normalną pracę bazy danych. Następnie jeszcze raz przeprowadza się analizę tabeli i powtórnie wykonuje powyższe zapytanie. Jeżeli parametr pctfree nie posiada wystarczająco wysokiej wartości, wtedy niektóre wiersze, których długość przekroczy dostępną przestrzeń, mogą zostać przeniesione do nowych bloków. Jeżeli liczba bloków nie zmieni się, wtedy wartość dla pctfree jest odpowiednia.

Proces przenoszenia wierszy ze względu na niewłaściwą wartość dla pctfree nazywa się migracją wierszy. Migracja wierszy ma negatywny wpływ na wydajność transakcji.

Wartość pctfree nie może być także zbyt duża, ponieważ efektem tego będzie niewłaściwa gospodarka przestrzenią. Dzięki poleceniu analyze, które zostało przedstawione wcześniej, można też otrzymać wartości dla kolumny Avg_Space (średnia liczba bajtów w bloku) w perspektywie USER_TABLES. W kolumnie tej znajduje się średnia liczba bajtów w każdym bloku danych. Jeśli wartość ta zwiększa się po wykonaniu modyfikacji wierszy, oznacza to, że wartość parametru pctfree może zostać zmniejszona.

Określenie właściwej wartości parametru pctused

Wartość pctused określa, kiedy używany blok jest ponownie dodawany do listy bloków, do których mogą być wprowadzane dane. Przykładowo, niech wartość pctfree dla danej tabeli jest równa 20, a wartość dla pctused wynosi 50. Podczas wstawiania wierszy do tabeli system Oracle pozostawia 20% wolnej przestrzeni (na potrzeby ewentualnej modyfikacji wcześniej wprowadzonych rekordów). Jeśli teraz rekordy z bloku zostaną usunięte, system Oracle nie zwalnia automatycznie wolnej przestrzeni w blokach. Nowe wiersze nie będą mogły być wprowadzone do bloku, zanim ilość używanej w bloku przestrzeni nie spadnie poniżej wartości pctused — 50%.

Domyślna wartość pctused to 40%. Jeżeli w danej aplikacji następuje częste usuwanie wierszy i jest używana domyślna wartość dla pctused, wtedy w tabeli występuje wiele bloków, które są wykorzystane tylko w 40%.

Aby uzyskać możliwie najlepsze wykorzystanie przestrzeni, należy tak ustawić wartość pctused, aby suma wartości pctused i pctfree była równa 85. Przykładowo, jeśli wartość dla pctfree jest równa 20%, wartość pctused powinna być równa 65%. W ten sposób przynajmniej 65% każdego bloku jest wykorzystywana, a 20% jest zachowana na potrzeby ewentualnych modyfikacji oraz rozszerzania wierszy.

Obliczanie rozmiarów indeksów

Proces obliczania rozmiarów indeksów jest bardzo podobny do procesu obliczania rozmiarów tabel, który przedstawiono wcześniej. Istnieje jednak kilka różnic wynikających z różnic w strukturze tabel i indeksów. Podobnie jak w przypadku tabel, wszystkie obliczenia zawarte w tym podrozdziale są obliczeniami przybliżonymi.

Jeżeli definicje kolumn indeksu oraz rozmiary danych są znane, można przystąpić do obliczeń związanych z ich wymaganiami dotyczącymi składowania. Są to rozważania teoretyczne, ponieważ rzeczywiste rozmiary danych oraz długość wierszy nie są do końca znane przed utworzeniem tabeli. Ważne jest, aby przykładowe dane dostępne były podczas wykonywania obliczeń, co pozwoli na otrzymanie dokładniejszych rezultatów.

W pierwszej kolejności oblicza się ilość przestrzeni używanej przez nagłówek bloku. Nagłówek bloku służy do zarządzania danymi w bloku. Rozmiar nagłówka bloku wynosi w przybliżeniu 161 bajtów. Jeśli używa się bloku o rozmiarze 4 KB, pozostaje 3935 bajtów wolnych:

```
4096 - 161 = 3935 dostępnych bajtów.
```

Następnym czynnikiem, jaki trzeba wziąć pod uwagę, jest parametr pctfree dla danej tabeli. Mnożąc wartość dla tego parametru przez ilość wolnej przestrzeni otrzymuje się ilość przestrzeni zarezerwowanej dla operacji modyfikacji wierszy.

Jeśli wartość dla pctfree jest równa 5, wtedy mnożąc dostępną przestrzeń przez 0.05 otrzymujemy:

```
3935*(pctfree/100) = 3935*0.05 = 197 (po zaokrągleniu).
```

197 bajtów z dostępnej wolnej przestrzeni będzie zarezerwowanych na potrzeby ewentualnych modyfikacji wierszy. Pozostała dostępna wolna przestrzeń wynosi:

```
3935 - 197 = 3738 dostępnych bajtów.
```

Po dokonaniu obliczeń z 4096 bajtów w bloku otrzymuje się 3738 bajtów, które dostępne są dla wpisów indeksowych.

Następnie oblicza się przestrzeń zajmowaną przez jeden indeks. W celu oszacowania przestrzeni używanej przez jeden indeks trzeba oszacować średnią długość kolumn w wierszu indeksu. W przypadku tabel obliczano średnią długość wszystkich kolumn, a w przypadku indeksów oblicza się tylko długość kolumn dotyczących indeksu. Średnia długość wiersza to całkowita średnia długość wszystkich wartości w kolumnach indeksu. Jeśli nie ma żadnych dostępnych danych, można oszacować aktualną długość wszystkich wartości w kolumnie. Nie stosuje się całej długości kolumny, jeśli dane nie wypełniają jej w całości.

Przykładowo, rozważono indeks zawierający dwie kolumny, obydwie są typu VARCHAR2(10). Średnia długość wiersza indeksu przekracza 20, a jego aktualna długość jest zależna od ilości danych aktualnie przechowywanych w danym wierszu.

Jeżeli istnieje dostęp do przykładowych danych, wtedy istnieje możliwość wykorzystania funkcji VSIZE w celu określenia średniej długości wiersza. Kontynuując przykład indeksu posiadającego dwie kolumny, należy wykonać poniższe zapytanie:

```
select AVG(NVL(VSIZE(Column1),0)+
       AVG(NVL(VSIZE(Column2),0))  Średnia_długość_wiersza
from TABLENAME;
```

W przykładzie tym określono średnią długość każdej kolumny, a następnie na tej podstawie określono średnią długość wiersza.

Dla danej przykładowej tabeli złożonej z dwóch kolumn założono, że średnia długość wiersza wynosi 16 bajtów. Do całości dodano 1 bajt na każdą kolumnę w indeksie, czyli razem 18 bajtów na wiersz. Jeśli indeks posiada kolumny zawierające dane, których długość przekracza 127 znaków, dodać należy dodatkowy bajt dla każdej kolumny. Wreszcie dodaje się też 8 bajtów dla nagłówka indeksu.

```
Przestrzeń używana przez jeden wiersz = Średnia długość wiersza
                                      + liczba kolumn
                                      + liczba długich kolumn
                                      + 8 bajtów nagłówka bloku.
```

Dla naszej przykładowej tabeli przestrzeń używana przez jeden wiersz wynosi:

```
Przestrzeń używana przez jeden wiersz = 24
                                        +2
                                        +0
                                        +8
                                      = 26 bajtów na indeks.
```

Jeśli indeks jest indeksem unikalnym, dodaje się 1 bajt do całości:

```
26 + 1 = 27 bajtów na wiersz.
```

Mając do dyspozycji 3738 bajtów oraz obliczone 27 bajtów na wiersz, można w jednym bloku umieścić 138 indeksów.

```
Liczba wierszy na blok = TRUNC(3738 wolnych bajtów/27 bajtów na wiersz)
                       = 138 indeksów na blok
```

Ponieważ w jednym bloku mieści się 138 indeksów, można oszacować liczbę potrzebnych bloków. Jak zauważono w poprzednim podrozdziale, są to jedynie przybliżone obliczenia wymagań związanych z przestrzenią w tabeli. Wymagania te zwiększają się w razie manipulowania rekordami w tabeli. Duża liczba operacji modyfikacji (update) oraz usuwania (delete) wymaga większej ilości przestrzeni.

Przestrzeń usuwana w obrębie indeksów jest rzadko ponownie wykorzystywana, a więc indeksy mogą wciąż przyrastać, nawet jeśli związana z nimi tabela nie zmienia swych wymiarów. Przykładowo, po usunięciu 100 wierszy z tabeli możliwe jest ponowne

wykorzystanie zwolnionej po usuniętych rekordach przestrzeni podczas wstawiania nowych 100 wierszy. Dzięki temu rozmiar przestrzeni używanej w tabeli pozostaje bez zmian. Inaczej jest w przypadku tabel zawierających indeksy. Przestrzeń zwolniona po usunięciu rekordów nie jest używana ponownie. Przestrzeń zajmowana przez indeksy więc wzrasta.

Obliczanie rozmiarów tabel klastrowych

Klastry używane są do przechowywania danych z różnych tabel w tych samych fizycznych blokach. Powinny one być stosowane, gdy rekordy z tych tabel są często przeglądane razem poprzez zapytania. Umieszczając rekordy w tych samych blokach, zmniejsza się liczbę bloków odczytywanych w czasie wykonywania zapytania, co poprawia wydajność pracy bazy danych. Jednak pamiętać należy, że użycie klastrów może mieć też negatywny wpływ na wydajność bazy danych. Dzieje się tak podczas transakcji (manipulacja danymi) oraz w przypadku zapytań używanych tylko do przeglądu jednej tabeli w klastrze.

Ze względu na specyficzną strukturę tabele klastrowe posiadają inne wymagania związane ze składowaniem, niż tabele nieklastrowe. W każdym klastrze przechowywane są dane z tabel oraz indeks klastra wykorzystywany podczas sortowania danych.

Kolumny w indeksie klastra nazywane są kluczem klastra. Jest to wspólny zestaw kolumn dla tabel w klastrze. Ponieważ kolumny w kluczu klastra określają fizyczne położenie wierszy w klastrze, nie powinny być często modyfikowane. Kluczem klastra jest zazwyczaj klucz obcy z jednej tabeli, który odwołuje się do klucza głównego w drugiej tabeli klastra.

Po wygenerowaniu klastra na kolumnach klucza klastra tworzony jest indeks klastra . Po utworzeniu indeksu opartego na kluczu klastra dane mogą być wprowadzane do tabel składowanych w klastrze. Kiedy wprowadza się wiersze do klastra, baza danych przechowuje klucz klastra oraz związane z nim wiersze w każdym z bloków klastra.

Ze względu na złożoną strukturę dobieranie odpowiednich rozmiarów dla klastrów jest dużo bardziej złożone, aniżeli w przypadku tabel czy indeksów. Dzieje się tak nawet w razie wykorzystania uproszczonych metod.

Proces dobierania wymiarów klastra zawiera w sobie elementy z analogicznych procesów dla tabel oraz indeksów. Kontynuując przykład z poprzedniego podrozdziału, dana jest tabela zawierająca trzy kolumny. Wszystkie kolumny są typu VARCHAR2(10). Jeśli tabela ta jest często łączona z drugą tabelą, wtedy właściwym rozwiązaniem jest umieszczenie obydwu tych tabel razem w klastrze. Dla celów tego przykładu założono, że w drugiej tabeli znajdują się dwie kolumny: pierwsza typu VARCHAR2(10), a druga typu VARCHAR2(5). Pierwsza kolumna jest używana do łączenia obu tabel.

Ponieważ obydwie tabele połączone są przez kolumnę typu VARCHAR2(10), kolumna ta znajduje się w kluczu klastra.

W pierwszej kolejności oszacować należy ilość przestrzeni używanej przez nagłówek bloku. Nagłówek bloku służy do zarządzania danymi w bloku. Rozmiar nagłówka bloku klastra wynosi w przybliżeniu 110 bajtów. Jeśli używa się bloku o rozmiarze 2 KB, pozostaje 1938 bajtów wolnych. Zatem dla bloku o rozmiarze 4 KB, pozostaje 3986 wolnych bajtów.

```
2048 - 110 = 1938 dostępnych bajtów.
```

Następnym czynnikiem, jaki trzeba wziąć pod uwagę, jest parametr `pctfree` dla danej tabeli. Mnożąc wartość dla tego parametru przez ilość wolnej przestrzeni otrzymuje się ilość przestrzeni zarezerwowanej dla operacji modyfikacji wierszy.

Jeśli wartość dla `pctfree` jest równa 10, wtedy mnożąc dostępną przestrzeń przez 0.1, otrzymuje się:

```
1938*(pctfree/100) = 1938*0.1 = 194 (po zaokrągleniu).
```

194 bajty z dostępnej wolnej przestrzeni będzie zarezerwowanych dla modyfikowanych wierszy. Pozostała dostępna wolna przestrzeń wynosi:

```
1938 - 194 = 1744 dostępne bajty.
```

Następnie odejmuje się przestrzeń używaną dla nagłówka tabeli. Przestrzeń buforu jest czterokrotnością liczby tabel plus 4 bajty. Dla dwóch tabel w klastrze ilość dostępnej przestrzeni jest równa:

```
Liczba dostępnych bajtów = 1744 dostępne bajty
                            - 4 bajty
                            - 4*liczba tabel
                         = 1744
                            - 4
                            - 8
                         = 1732 dostępne bajty.
```

Z 2048 bajtów w bloku 1732 bajty są dostępne dla wierszy klastra.

Następnie oblicza się przestrzeń wymaganą dla jednego wiersza w każdej tabeli, włączając w to długość kolumn (kolumny) w kluczu klastra.

Jeżeli możliwy jest dostęp do przykładowych danych, wtedy używa się funkcji `VSIZE` w celu określenia aktualnej przestrzeni używanej przez dane. Ponownie założono, że pierwsza tabela posiada trzy kolumny, a druga posiada dwie kolumny. Aby określić średnią długość wiersza, wykonać trzeba poniższe zapytanie:

```
select AVG(NVL(VSIZE(Column1),0))+
       AVG(NVL(VSIZE(Column2),0)) Średnia_długość_wiersza_1
from TABLE1;
select AVG(NVL(VSIZE(Column1),0)) Średnia_długość_wiersza_2
from TABLE2;
```

W przykładzie tym określono średnią długość każdej kolumny, która nie jest w kluczu klastra. Średnie te są zsumowane, aby określić średnią długość wiersza. W przykładzie tym założono, że kluczem klastra są: kolumna Column3 w pierwszej tabeli oraz kolumna Column2 w drugiej tabeli.

Przyjęto, że średnia długość wiersza dla nieklastrowych kolumn tabeli pierwszej wynosi 20, a średnia długość wiersza dla nieklastrowej kolumny tabeli drugiej wynosi 3.

```
Średnia długość wiersza = 23 bajty
```

Każdy wiersz klastra posiada nagłówek, w którym przechowywane są informacje o klastrze. Całkowita przestrzeń wymagana dla klastra zawiera też przestrzeń przeznaczoną na nagłówek klastra i jest wyliczana według formuły przedstawionej poniżej. „Długa" kolumna to taka, w której wartość danych przekracza 250 znaków długości. Rozróżnienie takie jest niezbędne ze względu na różnice w długości przechowywanych wartości.

```
Przestrzeń nagłówka wiersza = 4 bajty
                            + liczba kolumn
                            + liczba długich kolumn.
```

Sumując średnią długość wiersza oraz przestrzeń nagłówka wiersza otrzymano:

```
Przestrzeń używana przez 1 wiersz = średnia długość wiersza + przestrzeń nagłówka
wiersza
            = 23 + 4 + liczba kolumn + liczba długich kolumn
            = 23 + 4 + 3 + 0
            = 30 bajtów.
```

Tak więc każdy wpis w klastrze wymaga 30 bajtów. Przestrzeń ta nie uwzględnia wymagań związanych z przechowywaniem indeksu klastra.

Następnym etapem w procesie obliczania wymiarów klastra jest określenie wartości parametru `size`, który jest niepowtarzalną wartością dla klastrów. Parametr ten określa liczbę bajtów wymaganych dla klucza klastra oraz dla wierszy z nim związanych.

Parametr `size` zależy od rozkładu danych, czyli od tego, ile wierszy znajduje się w tabeli dla każdej wartości z klucza klastra. W celu obliczenia tej wartości sumuje się wszystkie wiersze w tabeli klastrowej, a następnie dzieli tę wartość przez liczbę różnych wartości klucza klastra.

```
select
    COUNT(DISTINCT(column name))/              /*liczba rekordów w tabeli*/
    COUNT(*) rows_per_key                      /*liczba wartości klucza klastra*/
from tablename;
```

Przykładowo, tabela TABLE1 posiada 30 wierszy na każdą wartość klucza klastra, a tabela TABLE2 zawiera 1 wiersz na każdą wartość klucza klastra.

Aby określić parametr `size`, trzeba także znać średnią długość wartości klucza klastra. Należy więc zastosować funkcję `VSIZE` dla tabeli klastrowej. Ponieważ wartość ta powinna być taka sama dla obu tabel (poprzez integralność referencyjną), wykorzystuje się tylko jedną tabelę.

```
select
   AVG(NVL(VSIZE(cluster key column),0)) Średnia_długość_klucza
from TABLE1;
```

W tym przykładzie średnia wartość kolumny klucza wynosi 5 bajtów.

Obliczamy zatem wartość dla parametru `size`:

```
SIZE =
     (Liczba wierszy na klucz klastra w tabeli TABLE1*Średni rozmiar wiersza dla
     tabeli TABLE1)+
     (Liczba wierszy na klucz klastra w tabeli TABLE2*Średni rozmiar wiersza dla
     tabeli TABLE2)+
     nagłówek klucza klastra+
     długość kolumny klucza klastra+
     średnia długość klucza klastra+
     2*(Liczba wierszy na klucz klastra w tabeli TABLE1 + Liczba wierszy na klucz
     klastra w tabeli TABLE2 + Liczba wierszy na klucz klastra dla innych tabel
     w klastrze)
```

Długość nagłówka klucza klastra wynosi 19 bajtów, zatem parametr `size`:

```
SIZE = (30 wierszy na klucz w tabeli TABLE1*20 bajtów na wiersz)+
       (1 wiersz na klucz w tabeli TABLE2*3 bajty na wiersz)+
       19 bajtów dla nagłówka klucza klastra+
       10 bajtów (długość kolumny klucza klastra)+
       5  bajtów (średnia długość klucza klastra)+
       2*(30 wierszy+1 wiersz)
     = (30*20)+(1*3)+19+10+5+(2*31)
     = 600+3+19+10+5+62
     = 699 bajtów
```

Zatem każdy klucz klastra wymaga 699 bajtów (w zaokrągleniu 700 bajtów). Jest on umieszczany w dostępnej przestrzeni bloku. Dostępna przestrzeń w bloku została obliczona wcześniej i wynosi 1732 bajty. Liczba kluczy klastra w jednym bloku wynosi:

```
Liczba kluczy klastrów na jeden blok = wolna przestrzeń/(SIZE + 42)
                                     = 1732/(700+42)
                                     = 1732/742
                                     = 2 (po zaokrągleniu)
```

W każdym bloku mogą być przechowywane wartości dla dwóch kluczy klastra.

Ponieważ w każdym bloku mogą być przechowywane wartości dla dwóch kluczy klastra, wymagana liczba bloków dla klastra jest połową tej liczby. Tak więc jeśli istnieje 40 różnych wartości kluczy klastra, trzeba przydzielić 20 bloków dla klastra.

Rozmiary indeksów funkcyjnych

Od wersji Oracle8i można tworzyć takie indeksy funkcyjne, które obejmują nie tylko kolumny, ale także mogą być użyte w poleceniach wykorzystujących funkcje. Przykładowo, można utworzyć indeks na funkcji `UPPER` (Name) a nie tylko na kolumnie Name (Nazwisko). Użycie indeksów funkcyjnych znacznie rozszerza możliwości strojenia zapytań. Dobierając wymiary dla indeksów funkcyjnych, można użyć tych samych metod, co w przypadku standardowych indeksów, które zostały przedstawione we wcześniejszym podrozdziale.

Indeksy z odwróconym kluczem

Od wersji Oarcle8 podczas tworzenia indeksów można określić parametr `reverse` w celu umożliwienia przechowywania bajtów w bloku indeksowym w odwróconym porządku. Możliwe jest zatem tworzenie indeksów z odwróconym kluczem. W indeksie z odwróconym kluczem wartości przechowywane są w odwróconej postaci, przykładowo, wartość 2201 będzie przechowywana jako 1022. W przypadku standardowych indeksów kolejne wartości sąsiadują ze sobą. W indeksie z odwróconym kluczem kolejne wartości nie są przechowywane blisko siebie. Jeśli więc często wykonywane są zapytania, które wykonują pełne przeglądy tabel, mogą zaistnieć problemy z rywalizacją operacji wejścia-wyjścia dotyczących indeksów. W takich przypadkach można rozważyć zastosowanie indeksów z odwróconym kluczem. Dobieranie wymiarów dla indeksów z odwróconym kluczem można przeprowadzać tymi samymi metodami, co w przypadku indeksów indeksów standardowych, które zostały przedstawione we wcześniejszym podrozdziale.

Rozmiary indeksów bitmapowych

Jeśli tworzony jest indeks bitmapowy, system Oracle dynamicznie kompresuje tworzone mapy bitowe. Wpływa to w dużym stopniu na oszczędności związane z ich składowaniem. Dobieranie wymiarów dla indeksów bitmapowych może być przeprowadzane za pomocą tych samych metod, co w przypadku standardowych indeksów (typu B*drzewo). Metody te zostały przedstawione we wcześniejszym podrozdziale. Po obliczeniu ilości wymaganej przestrzeni dla indeksów typu B*drzewo otrzymany wynik należy podzielić przez 10 i w ten sposób otrzymuje się rozmiar dla indeksu bitmapowego. W zasadzie indeksy bitmapowe zajmują od 5 do 10% przestrzeni zajmowanej przez porównywalne indeksy typu B*drzewo.

Rozmiary tabel indeksowych

Dane w tabeli indeksowej składowane są w porządku ustalonym przez klucz główny tabeli. Ilość wymaganej przez tabele indeksowe przestrzeni można oszacować podobnie, jak to opisano w odniesieniu do indeksów. Pewne różnice pojawiają się podczas obliczeń przestrzeni używanej przez wiersz. Dzieje się tak, ponieważ tabele indeksowe nie posiadają identyfikatorów wierszy (RowID).

W przypadku tabeli indeksowej przestrzeń zajmowana przez wiersz jest mniejsza, ponieważ nagłówek wiersza zajmuje nie 8, a 2 bajty.

```
Przestrzeń używana przez wiersz = średnia długość wiersza
                                + liczba kolumn
                                + liczba długich kolumn
                                + 2 bajty nagłówka
```

Rozmiary tabel zawierających duże obiekty (LOB)

Duże obiekty (typu BLOB oraz CLOB) przeważnie są przechowywane poza głównymi tabelami bazy danych. W celu określenia miejsca składowania dużych obiektów można użyć klauzuli `lob` w poleceniu `create table`. W głównych tabelach system Oracle przechowuje tylko wskaźniki odnoszące się do dużych obiektów. Szacunkowa długość takiego wskaźnika wynosi 24 bajty.

System Oracle nie zawsze przechowuje duże obiekty poza głównymi tabelami. Zasadniczo, jeśli rozmiar obiektu nie przekroczy 4 KB, może on być przechowywany w obrębie głównych tabel. Jeśli więc w bazie występują duże obiekty, których długość nie przekracza 4 KB, trzeba rozważyć ich wpływ na inne obiekty w głównych tabelach. Jeśli duże obiekty są mniejsze niż 4000 znaków, można dla przechowania danych zastosować typ VARCHAR2 zamiast typów LOB.

Rozmiary partycji

Od wersji Oracle8 można tworzyć partycje na tabelach. Partycjonowana tabela składa się z wielu oddzielnych partycji. Przykładowo, tabela SALES (sprzedaż) może być podzielona na cztery partycje: SALES _NORTH (sprzedaż na północy), SALES_ SOUTH (sprzedaż na południu), SALES_EAST (sprzedaż na wschodzie), SALES_ WEST (sprzedaż na zachodzie). Rozmiary każdej partycji oblicza się za pomocą metod odnoszących się do tabel, które przedstawiono w jednym ze wcześniejszych podrozdziałów. Na partycjach można tworzyć także indeksy, których rozmiary obliczane są w podobny sposób, jak rozmiary indeksów na tabelach. Metody te opisano powyżej. W rozdziale 12. znajdują się dalsze informacje dotyczące zarządzania partycjami.

Rozmiary tabel opartych na typach danych użytkownika

W wersji Oracle8 zostały wprowadzone struktury obiektowo-relacyjne. Kluczowe znaczenie mają dwie spośród tych struktur: typy danych użytkownika oraz metody konstruktora. Typy danych użytkownika używane są w celu definiowania nowych struktur danych — przykładowo, typ danych ADDRESS_TY (adres) może posiadać właściwości związane z danymi adresowymi oraz metody manipulowania tymi danymi. W momencie utworzenia typu ADDRESS_TY system Oracle automatycznie tworzy metodę konstruktora o nazwie ADDRESS_TY. Metoda ta zawiera parametry, które dopasowują właściwości tego typu danych, ułatwiając wprowadzanie nowych wartości w tym formacie danych. W dalszej części tego podrozdziału opisano sposoby tworzenia tabel, które wykorzystują typy danych zdefiniowane przez użytkownika. Znajdują się tam również informacje na temat doboru wymiarów oraz kwestie związane z zabezpieczeniem ich wykonania.

Od wersji Oarcle8 istnieje możliwość tworzenia tabel wykorzystujących typy danych użytkownika do definiowania kolumn. Przykładowo, można utworzyć typ danych opisujący dane adresowe:

```
create type ADDRESS_TY as object
(street   VARCHAR2(50),
 city     VARCHAR2(25),
 state    CHAR(2),
 zip      NUMBER);
/
```

Utworzony nowy typ danych może posłużyć podczas tworzenia tabeli, jak pokazano poniżej:

```
create table CUSTOMER
(Name   VARCHAR2(25),
 Address  ADDRESS_TY);
```

Podczas tworzenia nowego typu danych system Oracle generuje metodę konstruktora w celu obsługi operacji wprowadzania (`insert`) danych. Metoda konstruktora posiada taką samą nazwę, jak związany z nią typ danych, a jej parametry są atrybutami danego typu. Kontunuując powyższy przykład, podczas wprowadzania rekordów do tabeli CUSTOMER (Klient) trzeba użyć metody konstruktora dla typu ADDRESS_TY, aby umieścić w tabeli dane dotyczące adresu (pole `Address`).

```
Insert into CUSTOMER values
('Joe',ADDRESS_TY('My Street', 'Some City', 'ST', 10001));
```

W przykładzie tym polecenie `insert` wywołuje metodę konstruktora ADDRESS_TY w celu wprowadzania danych do pól typu ADDRESS_TY.

Stosowanie nowych typów danych zwiększa zapotrzebowanie na przestrzeń wykorzystywaną przez tabelę i jest to liczba 8 bajtów dla każdego nowego typu danych. Jeśli kolejny typ danych będzie zawierał kolejny nowo zdefiniowany typ danych, wtedy trzeba dodać kolejne 8 bajtów.

Używanie perspektyw obiektowych

Używanie nowych typów danych może zwiększyć złożoność środowiska programowego. Kiedy podczas wykonywania zapytania stosuje się typy danych zdefiniowane przez użytkownika, konieczne jest używanie innej składni, niż tej używanej podczas przeglądania tabel nie zawierających nowych typów danych. Jeśli nowe typy danych nie są stosowane we wszystkich tabelach bazy, należy oddzielić od siebie polecenia, które używają składni dla tabel zawierających nowe typy danych od tych, które na tych tabelach nie działają. Trzeba wcześniej wiedzieć, które zapytania wykorzystują typy danych zdefiniowane przez użytkownika.

Przykładowo, tabela CUSTOMER używa typu danych ADDRESS_TY, który został opisany w poprzednim podrozdziale.

```
create table CUSTOMER
(Name   VARCHAR2(25),
 Address  ADDRESS_TY);
```

Typ danych ADDRESS_TY posiada z kolei cztery atrybuty: Street (ulica), City (miasto), State (stan) oraz Zip (kod pocztowy). Aby przejrzeć wartości dla atrybutu Street z kolumny Address tabeli CUSTOMER, można wypróbować następujące zapytanie:

```
select Address.Street from CUSTOMER;
```

Niestety, zapytanie to nie będzie działać. Aby móc przeglądać atrybuty typów danych zdefiniowanych przez użytkownika, trzeba użyć zmiennych wiążących nazwę tabeli. W przeciwnym razie może wystąpić dwuznaczność w czasie dostępu do danego obiektu. Aby móc przeglądać atrybut Street, trzeba użyć zmiennej wiążącej (w tym przypadku „C”) dla tabeli CUSTOMER, jak pokazano w przykładzie poniżej:

```
select C.Address.Street from CUSTOMER  C;
```

Zmienne wiążące trzeba stosować we wszystkich zapytaniach używających atrybutów typów danych zdefiniowanych przez użytkownika, nawet jeśli zapytanie dotyczy dostępu tylko do jednej tabeli. Zatem istnieją dwie właściwości związane z zapytaniami

używającymi dostępu do atrybutów nowych typów danych: zapis stosowany w czasie dostępu do atrybutów oraz wymagania związane z użyciem zmiennych wiążących. Uzyskanie pełnej zgodności z typami danych zdefiniowanymi przez użytkownika jest możliwe pod warunkiem zmiany zasad SQL, co pozwoli na obsługę zmiennych wiążących. Należy pamiętać, że nawet w razie stałego korzystania ze zmiennych wiążących, mogą wystąpić problemy z zapisem wymaganym podczas dostępu do atrybutów typów danych zdefiniowanych przez użytkownika. Jest to spowodowane tym, że nie można stosować podobnych zapisów podczas dostępu do tabel, które nowych typów danych nie używają.

Perspektywy obiektowe są efektywnym, kompromisowym rozwiązaniem tych niezgodności. W wyżej podanym przykładzie tworzenia tabeli CUSTOMER założono, że typ danych ADDRESS_TY już istnieje. Jednakże czasami zachodzi potrzeba wprowadzania struktur obiektowo-relacyjnych (takich, jak typy danych zdefiniowane przez użytkownika) do już istniejącej tabeli, będącej częścią aplikacji opartą na relacyjnej bazie danych. Takim celom służą perspektywy obiektowe, które zostały utworzone specjalnie na potrzeby związane z definiowaniem takich obiektów w już istniejących, relacyjnych tabelach.

Jeśli tabela CUSTOMER już istnieje, można stworzyć nowy typ danych ADDRESS_TY i użyć perspektyw obiektowych w celu powiązania nowego typu z tabelą CUSTOMER. W poniższym przykładzie tabela CUSTOMER została utworzona przy użyciu jedynie standardowych typów danych.

```
create table CUSTOMER
(Name        VARCHAR2(25) primary key,
 street      VARCHAR2(50),
 city        VARCHAR2(25),
 state       CHAR(2),
 zip         NUMBER);
```

Aby utworzyć następną tabelę, która by przechowywała informacje o osobach i adresach, można użyć typu danych ADDRESS_TY. W tabeli CUSTOMER powinno się więc zastosować ten typ danych. W kolejnych przykładach pokazano stosowanie typu danych (ADDRESS_TY), którego utworzenie przedstawiono na przykładzie zamieszczonym w poprzednim podrozdziale.

Można utworzyć perspektywę obiektową opartą na tabeli CUSTOMER, używając wszystkich zdefiniowanych typów danych. Generując perspektywę obiektową używamy polecenia `create view`. W poleceniu tym należy określić zapytanie, na którym będzie opierała się perspektywa. Poniżej przedstawiono przykład tworzenia perspektywy obiektowej CUSTOMER_OV:

```
create view CUSTOMER_OV (Name, Address) as
select Name,
       ADDRESS_TY(Street, City, State, Zip)
from CUSTOMER;
```

Perspektywa CUSTOMER_OV posiada dwie kolumny: Name oraz Address (druga z nich jest zdefiniowana za pomocą typu ADDRESS_TY). Należy zwrócić uwagę, że w poleceniu `create view` nie zastosowano klauzuli `object`.

Analizując powyższy przykład należy zwrócić uwagę na kilka ważnych kwestii dotyczących składni. Kiedy tabela jest zbudowana na podstawie istniejącego już, nowego typu danych, wtedy podczas wybierania kolumny w trakcie zapytania odwoływuje się do niej poprzez jej nazwę (np. Name), a nie przez metodę konstruktora. Natomiast po utworzeniu perspektyw obiektowych w trakcie zapytań należy odwoływać się do nazw metod konstruktora. W zapytaniu, na którym opiera się perspektywa, można także użyć klauzuli `where`, co ogranicza liczbę wierszy zwracanych przez perspektywę.

Jeśli używa się perspektyw obiektowych, nie trzeba zmieniać sposobu administrowania tabelami relacyjnymi . Nadal trzeba zarządzać uprawnieniami dotyczącymi typów danych (w następnym podrozdziale znajdują się informacje dotyczące zabezpieczeń związanych z typami danych zdefiniowanymi przez użytkownika), ale struktura tabel oraz indeksów pozostaje taka sama. Korzystanie ze starych struktur może uprościć zadania związane z administrowaniem, pozwalając programistom na dostęp do obiektów poprzez perspektywy obiektowe.

Perspektywy obiektowe mogą także służyć do symulacji powiązań używanych przez wiersze obiektowe. Wiersze obiektowe są to wiersze w tabeli obiektowej. Aby utworzyć perspektywę obiektową, obsługującą wiersze obiektowe, trzeba najpierw utworzyć typ danych, który będzie posiadał taką samą strukturę, jak tabela.

```
create or replace type CUSTOMER_TY as object
(Name        VARCHAR2(25),
 street      VARCHAR2(50),
 city        VARCHAR2(25),
 state       CHAR(2),
 zip         NUMBER);
```

Następnie tworzy się perspektywę obiektową opartą na typie CUSTOMER_TY, przypisując identyfikatory obiektów (OID) rekordom tabeli CUSTOMER.

```
Create view CUSTOMER_OV of CUSTOMER_TY
  With object OID (Name) as
Select Name, Street, City, Zip
  from CUSTOMER;
```

W pierwszej części polecenia `create view` nadano nazwę perspektywie (CUSTOMER _OV) oraz określono jej typ (CUSTOMER_TY). OID jest identyfikatorem obiektu (dla wierszy obiektowych). W tym przypadku identyfikatorem obiektu była kolumna Name.

Jeśli istnieje druga tabela, która łączy się z tabelą CUSTOMER poprzez powiązanie klucza głównego i klucza obcego, można wtedy utworzyć perspektywę obiektową powiązaną z perspektywą CUSTOMER_OV. Przykładowo, tabela CUSTOMER_CALL (odwołanie do tabeli Klient) zawiera klucz obcy do tabeli CUSTOMER, jak pokazano poniżej:

```
create table CUSTOMER_CALL
(Name          VARCHAR2(25),
 Call_Number   NUMBER,
 Call_Date     DATE,
 constraint CUSTOMER_CALL_PK
     primary key (Name, Call_Number),
constraint CUSTOMER_CALL_FK foreign key (Name)
    references CUSTOMER(Name));
```

Kolumna Name w tabeli CUSTOMER_CALL jest powiązana z kolumną o tej samej nazwie w tabeli CUSTOMER. Ze względu na to, że symulowane są identyfikatory obiektów (nazywane pkOID) oparte na tabeli CUSTOMER, trzeba utworzyć powiązanie między identyfikatorami. System Oracle dostarcza operatora o nazwie MAKE_REF, dzięki któremu można tworzyć te powiązania (nazywane pkREF). W poniższym przykładzie operator MAKE_REF używany jest do utworzenia powiązania pomiędzy perspektywą obiektową na tabeli CUSTOMER_CALL a perspektywą obiektową na tabeli CUSTOMER:

```
Create view CUSTOMER_CALL_OV as
select MAKE_REF(CUSTOMER_OV, Name) Name
       Call_Number,
       Call_Date
from CUSTOMER_CALL;
```

Podczas tworzenia perspektywy CUSTOMER_CALL_OV określono nazwę perspektywy powiązanej z nią oraz kolumny tworzące to powiązanie. Dzięki temu można wykonywać zapytania na perspektywie CUSTOMER_OV z wewnątrz perspektywy CUSTOMER_CALL_OV, za pomocą operatora DEREF na kolumnie (identyfikatorze) tabeli CUSTOMER.

```
Select DEREF(CCOV.Name)
       from CUSTOMER_CALL_OV CCOV
where Call_Date = Trunc(SysDate);
```

Można w ten sposób korzystać z danych w tabeli CUSTOMER nie wydając zapytania bezpośrednio do tej tabeli. W powyższym przykładzie kolumna Call_date (data odwołania) ogranicza liczbę wierszy zwracanych przez zapytanie.

Niezależnie od tego, czy używa się wierszy obiektowych, czy też kolumn obiektowych, można używać perspektyw obiektowych jako osłony naszych tabel. W takim razie tabele nie są modyfikowane i można administrować nimi w dotychczasowy sposób. Jedyną różnicą dla użytkownika jest to, że może on mieć dostęp do wierszy tabeli CUSTOMER, jeśli są one wierszami obiektowymi.

Rozmiary tabel obiektowych oraz typów REF

Podczas tworzenia tabel obiektowych system Oracle tworzy identyfikatory obiektów (OID) dla każdego wiersza w tabeli. Na skutek tego średnia długość wiersza wzrasta o 16 bajtów. Tabela, która połączona jest z tabelą obiektową, zawiera kolumnę typu REF. Podczas szacowania wymagań dotyczących przestrzeni dla tabeli można przyjąć długość dla kolumny typu REF równą 16 bajtom.

Ochrona typów danych zdefiniowanych przez użytkownika

W przykładach zamieszczonych w poprzednim podrozdziale założono, że właścicielem nowego typu danych ADDRESS_TY oraz tabeli CUSTOMER jest ten sam użytkownik. Zdarza się jednak, że właścicielem nowego typu danych nie jest właściciel tabeli. Możliwe jest również, że kolejny użytkownik tworzy nowy typ danych oparty na typie, który został już utworzony. Należy ustalić wytyczne dotyczące użycia oraz właściwości typów danych definiowanych przez użytkownika w środowisku programistycznym.

Przykładowo, użytkownik o nazwie DORA jest właścicielem typu danych ADDRESS_TY, a inny użytkownik o nazwie GEORGE chce utworzyć nowy typ danych PERSON_TY (osoba). Użytkownik GEORGE wykonuje następujące polecenie:

```
create type PERSON_TY as object
(Name     VARCHAR2(25),
 Address  ADDRESS_TY);
```

Jeżeli użytkownik GEORGE nie jest właścicielem typu danych ADDRESS_TY, po wykonaniu powyższego polecenia otrzyma komunikat:

```
Ostrzeżenie: Typ został utworzony z błędami kompilacji.
```

Błędy kompilacji wynikają z problemów podczas tworzenia metody konstruktora po utworzeniu nowego typu danych. Spowodowane jest to brakiem możliwości odwołania się do typu danych ADDRESS_TY ze względu na to, że użytkownik GOERGE nie jest właścicielem typu o tej nazwie. Użytkownik GEORGE może wykonać ponownie polecenie `create type` (używając klauzuli `or replace`) odwołując się do typu danych ADDRESS_TY użytkownika DORA.

```
create or replace type PERSON_TY as object
(Name     VARCHAR2(25),
 Address  Dora.ADDRESS_TY);
/

Ostrzeżenie: Typ został utworzony z błędami kompilacji.
```

Aby sprawdzić, jakie błędy pojawiły się po wykonaniu polecenia `create type`, należy wykonać polecenie `show errors`, jak pokazano poniżej:

```
show errors
Błędy dla typu TYPE PERSON_TY:
LINIA/KOL  BŁĄD
---------  -------------------------------------------------------------------
0/0        PL/SQL: Zakończono analizę kompilacyjną modułu
3/11       PLS-00201: Identyfikator 'Dora.ADDRESS_TY' musi być zadeklarowany
```

Użytkownik GEORGE nie będzie mógł utworzyć typu danych PERSON_TY (który zawiera typ ADDRESS_TY), jeżeli wcześniej nie otrzyma uprawnienia EXECUTE do typu ADDRESS_TY od użytkownika DORA. Polecenie `grant` służące do przyznawania uprawnień w tym przypadku wygląda następująco:

```
grant EXECUTE on ADDRESS_TY to George;
```

Teraz użytkownik GEORGE może utworzyć już nowy typ danych oparty na typie danych ADDRESS_TY użytkownika DORA.

```
create or replace type PERSON_TY as object
(Name     VARCHAR2(25),
 Address  Dora.ADDRESS_TY);
```

Typ danych PERSON_TY został pomyślnie utworzony. Jednak istnieją jeszcze inne problemy związane z używaniem typów danych innych użytkowników. Przykładowo, w czasie operacji wprowadzania wierszy, trzeba zawsze określać nazwę właściciela

każdego używanego typu danych. Użytkownik GEORGE może utworzyć tabelę opartą na typie danych PERSON_TY (który zawiera typ ADDRESS_TY użytkownika DORA), jak pokazano poniżej:

```
create table GEORGE_CUSTOMERS
(Customer_ID  VARCHAR2(25),
 Address      Dora.ADDRESS_TY);
```

Jeśli użytkownik GEORGE byłby właścicielem typów PERSON_TY oraz ADDRESS_TY, wtedy wprowadzając rekordy do tabeli GEORGE_CUSTOMERS (klienci GEOORGE'A) mógłby posłużyć się poleceniem:

```
insert into GEORGE_CUSTOMERS values
(1, PERSON_TY('SomeNane',
    ADDRESS_TY('StreetValue','CityValue','ST',11111)));
```

Ponieważ użytkownik GEORGE nie jest właścicielem typu danych ADDRESS_TY, polecenie to nie zakończy się powodzeniem. W czasie wykonywania polecenia `insert` na typie danych ADDRESS_TY jest używana metoda konstruktora, której właścicielem jest użytkownik DORA. Dlatego też polecenie `insert` musi zostać zmodyfikowane w celu określenia, że właścicielem typu danych ADDRESS_TY jest użytkownik DORA. Poniższy przykład przedstawia poprawione polecenie `insert` z odwołaniem do użytkownika DORA:

```
insert into GEORGE_CUSTOMERS values
(1, PERSON_TY('SomeName',
    Dora.ADDRESS_TY('StreetValue','CityValue','ST',11111)));
```

Proszę przeanalizować poniższy przykład. Użytkownik GEORGE próbuje utworzyć synonim ADDRESS_TY dla typu danych użytkownika DORA:

```
create synonym ADDRESS_TY for Dora.ADDRESS_TY;
```

Jednakże utworzenie tego synonimu się nie powiedzie:

```
create type PERSON_TY as object
(Name     VARCHAR2(25),
 Address  ADDRESS_TY);

create type PERSON_TY as object
*
Błąd w linii 1:
ORA-22863: synonim dla typu danych DORA.ADDRESS_TY jest niedopuszczalny
```

Z opisu błędu wynika, że nie można tworzyć synonimów dla typów danych innych użytkowników. Dlatego też zawsze trzeba odwoływać się do nazwy właściciela typu danych podczas wykonywania polecenia `insert`.

W czasie tworzenia synonimu nie jest sprawdzana prawdziwość wykorzystywanego obiektu. Jeśli wykonujemy polecenie `create synonym x for y`, nie jest sprawdzane, czy nazwa oraz typ obiektu „y" są prawidłowe. Sprawdzanie dostępności obiektu odbywa się dopiero podczas próby dostępu do obiektu poprzez synonim.

W relacyjnej implementacji systemu Oracle istnieje możliwość przyznawania uprawnienia EXECUTE, które dotyczy obiektów proceduralnych, takich jak procedury i pakiety. W obiektowo-relacyjnej implementacji systemu Oracle rozszerzono możliwości przyznawania uprawnienia EXECUTE także na typy danych zdefiniowane przez użytkownika — funkcje oraz procedury PL/SQL, które wykorzystują te typy. Jeśli użytkownik przyzna komuś uprawnienie do używania typów danych przez siebie zdefiniowanych, jednocześnie przyzna uprawnienia do wykonywania metod zdefiniowanych na podstawie tych typów danych. Dlatego też uprawnienia typu EXECUTE powinny być przydzielane ostrożnie i z rozwagą.

Kontunuując rozważanie przykładu przedstawionego powyżej, pomimo że użytkownik DORA nie zdefiniował jeszcze żadnych metod związanych z typem ADDRESS_TY, automatycznie tworzone są specjalne procedury zwane metodami konstruktora, które używane są w czasie dostępu do danych. Każdy obiekt (taki jak PERSON_TY), który używa typu danych ADDRESS_TY, używa też związanych z tym typem metod konstruktora. Tak więc, jeśli nawet użytkownik nie utworzył żadnych metod dla swojego typu danych, zawsze istnieją metody z nim związane.

Nie można tworzyć publicznych typów danych (dostępnych dla wszystkich użytkowników) oraz publicznych synonimów dla typów danych użytkownika. Dlatego też zawsze trzeba odwoływać się do nazwy właściciela typu lub też tworzyć typy w obrębie danego konta, na którym należy tworzyć tabele w bazie danych. Żadne z tych rozwiązań nie upraszcza zarządzania typami danych użytkownika.

Indeksowanie atrybutów typów danych zdefiniowanych przez użytkownika

W poprzednim przykładzie tabela GEORGE_CUSTOMERS została utworzona w oparciu o typy danych PERSON_TY oraz ADDRESS_TY. Jak pokazano poniżej, tabela GEORGE_CUSTOMERS zawiera zwykłą kolumnę o nazwie Customer_ID (identyfikator klienta) oraz kolumnę Person (osoba), która jest zdefiniowana na podstawie typu danych PERSON_TY.

```
create table GEORGE_CUSTOMERS
(Customer_ID    NUMBER,
 Person         PERSON_TY);
```

Dzięki definicjom przedstawionym w poprzednim podrozdziale można zauważyć, że typ PERSON_TY zawiera kolumnę Name oraz kolumnę Address, która jest oparta na typie danych ADDRESS_TY.

Kiedy w czasie zapytań bądź podczas operacji modyfikacji (`update`) lub usuwania (`delete`) zachodzi potrzeba odwołania się do typu zdefiniowanego przez użytkownika, określa się pełną ścieżkę do atrybutów tego typu. Przykładowo, przedstawione poniżej zapytanie zwraca wartości dla kolumn Customer_ID oraz Name. Kolumna Name jest atrybutem typu danych, który definiuje kolumnę Person. Trzeba więc odwołać się do niej poprzez kolumnę Person.Name.

```
select C.Customer_ID, C.Person.Name
from GEORGE_CUSTOMERS C;
```

Do atrybutów typu danych ADDRESS_TY można odwoływać się poprzez określenie pełnej ścieżki do związanych z tym typem kolumn. Przykładowo, podczas odwoływania się do kolumny Street pełna ścieżka będzie wyglądać następująco: Person.Address.Street. Jest to pełen opis lokalizacji kolumny Street w strukturze tabeli. W poniższym przykładzie widzimy dwukrotne odwołanie do kolumny City, pierwszy raz na liście kolumn polecenia `select`, a drugi w klauzuli `where`.

```
select C.Person.Name,
       C.Person.Address.City
from   GEORGE_CUSTOMERS C
where  C.Person.Address.City like 'C%';
```

Kolumna City znajduje się w klauzuli `where` ograniczającej zakres przeszukiwania, więc optymalizator może zastosować indeks w czasie wykonywania zapytania. Jeśli na kolumnie City założony jest indeks, szybko odnajdywane są wszystkie wiersze, dla których spełniony jest warunek podany w klauzuli `where`, czyli wszystkie wartości z kolumny City zaczynające się na literę „C”.

W celu utworzenia indeksu na kolumnie, która jest częścią typu danych zdefiniowanego przez użytkownika, trzeba określić pełną ścieżkę do kolumny jako część polecenia `create index`. Tworząc indeks na kolumnie City (która jest częścią kolumny Address) można wykonać następujące polecenie:

```
create index I_GEORGE_CUSTOMERS$CITY
on GEORGE_CUSTOMERS(Person.Address.City);
```

Polecenie to tworzy indeks o nazwie I_GEORGE_CUSTOMERS$CITY na kolumnie Person.Address.City. Podczas każdorazowego odwołania się do kolumny City użyte do tego polecenie zostanie przeanalizowane przez optymalizator, który oceni użyteczność użycia indeksu pod kątem poprawy szybkości dostępu do danych.

Przed utworzeniem tabel opartych na typach danych zdefiniowanych przez użytkownika, należy rozważyć sposób odwoływania się do tych kolumn. Jeżeli, jak w poprzednim przykładzie, pewne kolumny są często używane z zastosowaniem klauzuli `where`, ograniczającej zakres przeszukiwania, wtedy na kolumnach tych powinien być założony indeks. Pod tym względem przedstawienie kilku kolumn w pojedynczym typie danych może pogarszać wydajność aplikacji, ponieważ określenie kolumn, które wymagają indeksu, może być nie do końca jasne.

Podczas tworzenia nowych typów danych grupy kolumn przeważnie traktuje się, jak pojedyncze jednostki, tak jak w przypadku przykładowej kolumny Address lub kolumny Person. Należy pamiętać, że w czasie przetwarzania zapytania kolumny te są traktowane przez optymalizator jako osobne jednostki. Dlatego też stosując indeksy w typach danych zdefiniowanych przez użytkownika należy odnosić je do kolumn. Dodatkowo trzeba pamiętać, że na przykład indeksując kolumnę City w jednej tabeli, w której używa się typu danych ADDRESS_TY, nie indeksuje się jej automatycznie w innych tabelach, w których też używane są tego typu dane. Jeżeli więc na przykład w tabeli o nazwie BRANCH (oddział) zastosowano typ danych ADDRESS_TY, kolumna City nie będzie indeksowana, jeżeli nie stworzyono dla niej indeksu. Oznacza to, że założenie indeksu na kolumnie City w tabeli CUSTOMER nie ma wpływu na kolumnę City w tabeli BRANCH.

Tworzenie iteracyjne

Metodologie tworzenia iteracyjnego zazwyczaj zawierają serię bardzo szybko rozwijanych rozwiązań prototypowych. Rozwiązania prototypowe używane są w celu zdefiniowania wymagań systemowych w czasie tworzenia aplikacji. Metodologie te są atrakcyjne, gdyż pozwalają na przedstawianie przyszłym użytkownikom możliwości aplikacji już w czasie procesu tworzenia. Jednakże często popełniane są błędy, które zmniejszają efektywność tworzenia iteracyjnego.

Pierwsza możliwość popełnienia błędu wynika z tego, że nie zawsze jest używana odpowiednia wersja produktu. Po utworzeniu kilku wersji aplikacji zmienia się niektóre jej części, inne zaś pozostawia bez zmian. W ten sposób pewne moduły aplikacji są w fazie tworzenia, kiedy inne są już w fazie testowania. Wtedy możliwe jest, że zbyt często jedna wersja aplikacji jest używana przez wszystkie kroki iteracyjne. W rezultacie końcowy produkt nie wykazuje wystarczającej elastyczności (której osiągnięcie jest przypuszczalnym celem tworzenia iteracyjnego).

Inny, często popełniany błąd, to stosowanie rozwiązań prototypowych jako aplikacji użytkowej. Rozwiązania prototypowe tworzone są, aby przedstawić końcowym użytkownikom pewien zarys końcowego produktu. Nie powinny być one brane pod uwagę, jako podstawa do utworzenia końcowego produktu, gdyż osiągnięcie maksymalnych jego możliwości dotyczących stabilności oraz elastyczności może okazać się nierealne. Podczas procesu tworzenia iteracyjnego rozwiązania prototypowe powinny być traktowane jako rozwiązania tymczasowe. Nie należy osadzać nowych systemów na rozwiązaniach tymczasowych, podobnie należy traktować rozwiązania prototypowe.

Kolejne niebezpieczeństwo wynika z tego, że różnice pomiędzy poszczególnymi etapami rozwoju aplikacji (tworzenie, testowanie, praca) zacierają się. Metodologia tworzenia iteracyjnego musi bardzo jasno definiować warunki przejścia aplikacji z jednego etapu do drugiego. Najlepszym rozwiązaniem jest całkowite oddzielenie rozwiązań prototypowych od pełnych aplikacji.

Ostatnią pułapką tworzenia iteracyjnego jest przyjmowanie nierealistycznego czasu wykonania. Zadania stosowane w metodologii strukturalnej stosowane są także w metodologii iteracyjnej. Fakt, że aplikacja jest tworzona w przyspieszonym tempie, nie oznacza, że wszystkie zadania będą także szybciej wykonane.

Iteracyjne definicje kolumn

W czasie trwania procesu tworzenia aplikacji definicje kolumn mogą się często zmieniać. Od wersji Oracle8i, można usuwać kolumny z istniejących już tabel. Ta właściwość wcześniej nie była dostępna. Można usuwać kolumny z tabel, bądź oznaczać je jako „nieużywane”, przeznaczone do późniejszego usunięcia. Jeśli kolumna jest usuwana natychmiast, może mieć to wpływ na wydajność pracy bazy danych. Jeśli kolumna jest tylko zaznaczona jako nieużywana, nie ma to wpływu na wydajność. Kolumna może być w rzeczywistości usunięta w późniejszym czasie, kiedy baza danych jest mniej obciążona.

W celu usunięcia kolumny używa się albo klauzuli `set unused`, albo klauzuli `drop` w poleceniu `alter table`. Nie można usuwać pseudokolumn, kolumn w tabelach zagnieżdżonych oraz kolumn będących częścią klucza partycji. W dodatku A znajduje się pełna składnia oraz opis ograniczeń związanych z poleceniem `alter table`.

W poniższym przykładzie usunięto kolumnę Col2 z tabeli TABLE1:

```
alter table TABLE1 drop column Col2;
```

Poniżej przedstawiono sposób oznaczenia kolumny jako nieużywanej.

```
alter table TABLE1 set unused column Col2;
```

Zaznaczenie kolumny jako nieużywanej nie spowoduje zwolnienia przestrzeni zajmowanej przez tę kolumnę. Możliwe jest to po usunięciu nieużywanej kolumny:

```
alter table Table1 drop unused columns;
```

Perspektywy systemowe: USER_UNUSED_COL_TABS, DBA_UNUSED_COL oraz ALL_UNUSED_COL_TABS dostarczają informacji o wszystkich tabelach, których kolumny zostały zaznaczone jako „nieużywane".

Jeśli zaznaczy się kolumnę jako „nieużywaną", nie można już ponownie użyć tej kolumny.

Można także za pomocą jednego polecenia usunąć kilka kolumn jednocześnie.

```
alter table TABLE1 drop (Col4, col5);
```

Kiedy usuwa się większą liczbę kolumn, nie należy używać słowa kluczowego `column` w poleceniu `alter table`, ponieważ spowoduje to błąd składniowy. Wyszczególnione nazwy usuwanych kolumn powinny być umieszczone w nawisach, jak pokazano w powyższym przykładzie.

Jeśli usuwane kolumny są częścią klucza głównego lub więzów unikalnych, wtedy należy zastosować klauzulę `cascade constraints` w poleceniu `alter table`. Jeżeli usuwa się tabelę, która należy do klucza głównego, usuwany jest także indeks klucza głównego.

Technologia

Elementy oprogramowania muszą być dostępne w czasie trwania procesu tworzenia aplikacji. Ze względu na to, że w skład większości zespołów wchodzi wielu programistów (oraz przynajmniej jeden administrator), muszą zostać ustalone sposoby komunikacji pomiędzy poszczególnymi członkami zespołu. Kanały komunikacyjne umożliwiają utrzymanie konsekwentnego planowania i wykonywania aplikacji.

W celu zastosowania odpowiedniej metodologii potrzebne są cztery rozwiązania techniczne. Obecnie stosowane są cztery oddzielne technologie. Są to narzędzia typu CASE, dzielone katalogi, bazy kontroli projektu oraz dyskusyjne bazy danych.

Narzędzia typu CASE

Narzędzia typu CASE są pomocne podczas tworzenia diagramów związków jednostek oraz diagramów fizycznego układu bazy danych. Oracle Designer jest wielostanowiskowym narzędziem typu CASE, za pomocą którego można tworzyć diagramy związków jednostek. Narzędzie to posiada także zintegrowany słownik danych. Za pomocą narzędzia Oracle Designer możliwe jest współużytkowanie wszystkich elementów w obrębie aplikacji, można także przechowywać informacje odnośnie rozmiarów tabel i wierszy. Dzięki tym właściwościom możliwe się staje rozwiązywanie problemów, z których kilka opisano w tym rozdziale. Wielodostęp stosowany w programie Oracle Designer pozwala na zapewnienie spójności pracy wszystkich programistów, dzięki czemu można utrzymywać kontrolę nad wszystkimi wersjami aplikacji.

Polecenia SQL, za pomocą których tworzymy obiekty w bazie danych, powinny być generowane bezpośrednio za pomocą narzędzi CASE. Można także używać narzędzi CASE w celu tworzenia ogólnych wersji aplikacji opartych na zdefiniowanych obiektach bazy danych.

Współdzielone katalogi

Nie do wszystkich zadań związanych np. z archiwizacją przypisano ściśle określone narzędzia, za pomocą których są one wykonywane. Poszczególne zadania powinny być realizowane za pomocą dostępnych i odpowiednich do tego celu narzędzi. Pliki, będące rezultatem tych działań, powinny być przechowywane we współdzielonych katalogach, gdzie są dostępne dla wszystkich członków zespołu. Format oraz konwencje nazewnicze powinny być ustalone we wcześniejszym etapie procesu tworzenia aplikacji.

Bazy danych kontroli projektu

Aby osoby spoza zespołu pracującego nad tworzeniem aplikacji mogły być informowane o aktualnym jej stanie oraz o zadaniach z nią związanych, należy utworzyć bazę danych kontroli projektu. Za pomocą takiej bazy osoby spoza zespołu będą miały wgląd w aktualny stan projektu. Osoby związane np. z kierownictwem firmy mogą dzięki bazie kontroli projektu sterować kolejnymi zadaniami tworzenia aplikacji. Za pomocą bazy danych kontroli projektu można także planować zmiany, kontrolować tempo ich wykonania oraz analizować strategiczne ścieżki rozwiązań. Dzięki tym analizom można dokonać modyfikacji w zasobach przypisanych do poszczególnych zadań w projekcie.

Dyskusyjne bazy danych

Większość informacji zawartych w opisanych wcześniej współdzielonych obszarach (narzędzia CASE, współdzielone katalogi oraz bazy danych kontroli projektu) reprezentuje pewien zasób uzgodnionych opinii. Przykładowo, jeśli kilku członków zespołu programistów może mieć pewne uwagi dotyczące strategii archiwizacji, kierownictwo oraz administratorzy muszą także się do nich ustosunkować. W celu ułatwienia komunikacji można utworzyć dyskusyjne bazy danych (zazwyczaj używa się do tego celu oprogramowania do pracy grupowej w lokalnej sieci). Wstępne projekty mogą być do nich przesyłane, zanim ich końcowe rozwiązania trafią do współdzielonych katalogów.

Zarządzanie pakietami

Proszę wyobrazić sobie środowisko programistyczne o następujących cechach:

- nie określono żadnych standardów współpracy;
- wszystkie obiekty tworzone są w schematach użytkowników SYS lub SYSTEM;
- właściwe rozłożenie oraz wymiary tabel i indeksów zostały określone tylko pobieżnie;
- każda aplikacja jest projektowana tak, jakby miała być jedyną pracującą w bazie danych.

Właściwe zarządzanie pakietami umożliwi uniknięcie scenariusza opisanego powyżej.

Odpowiednie zarządzanie implementacją pakietów wymaga wielu podobnych zabiegów, jak w przypadku procesu tworzenia aplikacji. Czynności te opisano w poprzednich podrozdziałach. W tym podrozdziale przedstawiono najlepszy sposób wykorzystania pakietów w procesie tworzenia aplikacji. W rozdziale 11. przedstawiono szczegóły dotyczące zarządzania aplikacjami pakietowymi.

Tworzenie diagramów

Większość narzędzi typu CASE posiada możliwość wykorzystywania pakietów technicznych podczas tworzenia fizycznych diagramów bazy danych. Składa się na to analiza struktur tabel oraz tworzenie fizycznych diagramów bazy danych na podstawie tych struktur. Dzieje się tak zazwyczaj poprzez analizę nazw kolumn oraz indeksów w celu zidentyfikowania kolumn kluczowych. Jednak w normalnych warunkach fizyczny diagram bazy danych nie jest odzwierciedleniem diagramu związków jednostek w stosunku jeden do jednego. Diagramy związków jednostek dla pakietów są zazwyczaj dostarczane z pakietem sprzedawcy. Są one niezwykle pomocne podczas planowania powiązań z pakietami bazy danych.

Wymagania dotyczące przestrzeni

Większość pakietów systemu Oracle zapewnia bardzo dokładne obliczenia dotyczące zużycia zasobów w czasie pracy produkcyjnej bazy danych. Jednakże obliczenia te najczęściej nie zdają egzaminu, kiedy bierze się pod uwagę ich wymagania dotyczące przestrzeni w czasie ładowania danych oraz uaktualniania oprogramowania. Z tej przyczyny rozważnym krokiem jest utworzenie przestrzeni tabel dla specjalnych segmentów wycofania (RBS_2), która byłaby używana podczas obsługi procesu ładowania danych. Także przestrzeń tabel zawierająca dane zapasowe może okazać się potrzebna, jeśli pakiet utworzy kopie wszystkich tabel w czasie trwania aktualizacji oprogramowania. W rozdziale 11. znajdują się dodatkowe wskazówki dotyczące zarządzania pakietami.

Cele strojenia

W przypadku aplikacji mówi się o pewnych celach związanych ze strojeniem, podobnie ma się w przypadku pakietów. Ustalenie i śledzenie wartości kontrolnych pomaga w identyfikacji obszarów pakietów wymagających strojenia (patrz rozdział 8.).

Wymagania związane z ochroną danych

Niestety, większość pakietów używanych przez bazę danych Oracle przynależy do jednej z poniższych dwóch kategorii: albo zostały przeniesione do systemu Oracle z innego systemu baz danych, albo używane są z założeniem, że posiadają pełne uprawnienia administratora dla swoich kont właścicieli obiektów.

Jeśli pakiety zostały wcześniej utworzone w innym systemie baz danych, wtedy podczas pracy w systemie Oracle nie zawsze jest możliwe wykorzystywanie ich wszystkich właściwości. Do właściwości tych należą: blokady na poziomie wierszy, użycie sekwencji, wyzwalaczy oraz metod. Strojenie takich pakietów w celu osiągnięcia oczekiwanych potrzeb może wymagać modyfikacji kodu źródłowego.

Jeśli pakiety używane są z założeniem, że posiadają pełne uprawnienia administratora, nie powinny być przechowywane w tej samej bazie danych, co inne strategiczne aplikacje. Większość pakietów wymaga pełnych uprawnień administratora w celu dodawania nowych użytkowników do bazy danych. Trzeba dokładnie określić, jakich uprawnień systemowych wymaga aktualnie konto administratora (zazwyczaj są to uprawnienia CREATE SESSION oraz CREATE USER) używającego pakiet. Można stworzyć specjalne role zawierające ograniczenia odpowiednich uprawnień systemowych dla administratora pakietu.

Pakiety, które były wcześniej utworzone w innym systemie baz danych, mogą wymagać użycia tego samego konta, co inny pakiet znajdujący się w systemie Oracle. Przykładowo, kilka aplikacji może dokonywać prób skorzystania z tego samego konta o nazwie SYSADM. Jedynym rozwiązaniem tego problemu jest tworzenie takich pakietów w osobnych bazach danych.

Wymagania związane z obsługą danych

Wszystkie wymagania związane z pakietami, a szczególnie dotyczące danych, muszą być jasno zdefiniowane. Jest to zazwyczaj dokładnie opisane w dokumentacji pakietu.

Wymagania związane z wersjami

W obsługiwanych aplikacjach mogą występować pewne zależności związane z wersjami i właściwościami systemu Oracle. Przykładowo, dana aplikacja pakietowa może być zalecana do pracy z wydaniami 8.0.5.1 oraz 8.1.6, ale już nie jest zalecana do pracy z wydaniem 8.1.5. Jeśli używa się aplikacji pakietowych, należy podstawową wersję uaktualniać zgodnie z dostarczanymi przez dostawcę wytycznymi, dotyczącymi obsługi wersji systemu Oracle. Ponadto dostawca może na przykład zmienić wykorzystywany optymalizator z używanego w wydaniu 8.0.5.1 optymalizatora regułowego na używany w wydaniu 8.1.6 optymalizator kosztowy. Tworzona baza danych powinna być na tyle elastyczna, aby obsłużyć tego typu zmiany.

Ponieważ takie ograniczenia pozostają z reguły poza możliwościami kontroli, należy przeprowadzać próby umieszczenia każdej aplikacji w osobnej instancji. Jeżeli często wykonuje się zapytania poprzez aplikację, a każda aplikacja jest umieszczona w osobnej instancji, zwiększa się liczbę używanych powiązań baz danych. Trzeba ocenić, czy bardziej korzystne jest obsługiwanie kilku instancji, czy też kilku aplikacji umieszczonych w jednej instancji.

Plany wykonania

Tworzenie planów wykonania wymaga dostępu do instrukcji SQL, które uruchamiane są w bazie danych. Dzielony obszar SQL w SGA (patrz rozdział 1. i rozdział 6.) przechowuje wszystkie wykonywane instrukcje SQL w bazie danych. Dopasowanie odpowiednich instrukcji SQL do określonych części aplikacji jest niezwykle czasochłonnym procesem. Najlepiej jest określić, które obszary aplikacji są elementami krytycznymi pod względem funkcjonalności oraz wydajności i współpracować z odpowiednimi zespołami obsługującymi pakiety w celu rozwiązania problemów związanych z wydajnością.

Procedury testów przyjęcia

Procedury testów przyjęcia dla pakietu zazwyczaj wykonywane są po zainstalowaniu aplikacji. Jednakże pakiety powinny spełniać te same wymagania funkcjonalne, co aplikacje. Testy przyjęcia powinny zostać przeprowadzone przed dokonaniem wyboru pakietów. Testy powinny być generowane na podstawie kryteriów wyboru pakietu. Ten sposób przeprowadzania testów pozwala na sprawdzenie właściwości, które interesują przyszłych użytkowników, a nie tych, które wybiorą programiści.

Należy przygotować plan postępowania na wypadek, gdy pakiet nie przejdzie pomyślnie fazy testu przyjęcia z powodów funkcjonalnych lub wydajnościowych. Warunki powodzenia testu przyjęcia muszą być respektowane, ponieważ od nich również zależy zadowolenie klienta - końcowego użytkownika aplikacji.

Obszar testowania

Podczas ustalania obszaru testowania należy kierować się następującymi wskazówkami:

1. Obszar ten musi być większy niż obszar pracy produkcyjnej aplikacji. Trzeba przewidzieć przyszły rozrost aplikacji.
2. Obszar musi zawierać znane zestawy danych, plany wykonania, wyniki wydajności oraz zestawy wyników.
3. Musi on być zastosowany zarówno dla każdego wydania bazy danych, narzędzi, jak i dla nowych właściwości.
4. Musi obsługiwać cały zestaw złożonych warunków testu, umożliwiających ocenę cech kosztów biznesowych. Nie należy polegać tylko na istocie analizy wyników. Najlepiej, jeśli można określić krzywe kosztów czy też korzyści, w zależności od wzrostu rozmiaru bazy danych.
5. Obszar ten musi być dość elastyczny, aby mógł uwzględniać różne opcje związane z licencjonowaniem.
6. Musi on być aktywnie używaną częścią technologii wykonywania metodologii.

Podczas prowadzenia implementacji i wykonywania testów w obszarze testowania zwykłe śledzenie ścieżki wykonania oraz wydajności pracy aplikacji podczas każdego zapytania w bazie danych jest niepraktyczne. W celu ułatwienia wykonania testów można wykorzystać dwie tradycyjne metody statystyczne — grupowania oraz próbkowania.

Aby otrzymać wyniki dotyczące wydajności, należy pogrupować zapytania oraz operacje i przeprowadzić testy dla całości. Jeśli wyniki dla całej grupy będą odbiegały od przyjętych kryteriów, można wtedy prześledzić indywidualnie poszczególne elementy danej grupy. Przykładowo, można pogrupować wszystkie operacje związane z ładowaniem danych w hurtowni danych. Jeśli wyniki wydajnościowe będą spełniały założone oczekiwania, nie ma potrzeby oceniania każdej operacji z osobna. Jeśli jednak wyniki nie spełniają przyjętych kryteriów, należy utworzoną wcześniej grupę rozbić na mniejsze grupy i ponownie przeprowadzić testy dla każdej z nich. Poprzez grupowanie operacji upraszcza się proces identyfikacji obszarów problemu oraz ich przyczyny.

Metoda próbkowania pozwala na określenie wpływu zmian na takie operacje, których nie można w łatwy sposób pogrupować. Przykładowo, można wybrać losowo reprezentatywną próbkę zapytań z naszej bazy danych (ręcznie lub poprzez kolumnę SQL_Text w perspektywie V$SQLAREA) i uruchomić je w celu określenia oczekiwanych wyników oraz planu wykonania. Następnie wykonuje się te zapytania w obszarze testowania i porównuje wyniki otrzymane z wynikami oczekiwanymi. Kiedy bada się dużą populację (np. ludność kraju) wielkość próbki uznawana za reprezentatywną wynosi

około 1/300 000. Dla aktywnej bazy danych, która posiada tysiące zapytań, powinno się brać pod uwagę większość najczęściej stosowanych zapytań, nie mniej jednak niż 100 operacji SQL w jednym zestawie.

W rzeczywistości próbka nie powinna być wybrana losowo. Próbki powinny reprezentować każdą z poniższych grup:

- zapytania, które wykorzystują klastry, przynajmniej po dwa z każdej z wymienionych grup: klastry, pętle zagnieżdżone, klastry rozszerzone oraz klastry haszowane;
- zapytania korzystające z powiązań baz danych;
- polecenia DML, które wykorzystują powiązania baz danych;
- przynajmniej po dwa polecenia z każdego typu poleceń DML (polecenia `insert`, `update` oraz `delete`);
- przynajmniej po jednym poleceniu z każdej głównej grupy poleceń DDL, uwzględniając tworzenie tabel, przebudowę indeksów, analizę operacji oraz przyznawanie uprawnień;
- przynajmniej jedno zapytanie, które używa opcji równoległych zapytań, jeśli opcja ta jest używana w danym środowisku.

Próbka zatem nie powinna być przypadkowa — powinna reprezentować odpowiednie operacje. Powinna ona odpowiadać przeglądowi głównych grup operacji, w tym także operacjom OLTP wykonywanym przez użytkowników. Wynik nie będzie odzwierciedleniem każdego działania w bazie danych, ale pozwoli na uświadomienie sobie następstw związanych z uaktualnieniami oraz pozwoli na zmniejszenie ryzyka i podejmowanie trafniejszych decyzji związanych z wprowadzaniem nowych opcji.

Zarządzanie środowiskiem

Rezultatem wprowadzenia trzech krytycznych elementów — właściwej współpracy, procesu zarządzania oraz technologii — będzie środowisko tworzenia aplikacji zawierające wbudowaną kontrolę jakości. Pozwala to na wprowadzanie ulepszeń do procesu tworzenia aplikacji. Korzyści z tego płynące są bardzo wymierne: jest to poprawa wydajności, lepsza integracja z innymi aplikacjami w przedsiębiorstwie oraz uproszczenie obsługi.