Strojenie aplikacji raportujących

Juliusz Jezierski
Instytut Informatyki
Politechnika Poznańska
e–mail: Juliusz.Jezierski@cs.put.poznan.pl

Abstrakt. Zwiększające się zapotrzebowanie na dane o charakterze syntetycznym wymaga budowania złożonych systemów raportowania. Aplikacje takich systemów przetwarzają duże wolumeny danych, wykonując wielokrotne łączenie kilkunastu, a nawet kilkudziesięciu tabel. W zależności od mocy dysponowanego sprzętu aplikacje te mogą się wykonywać od kilku do kilkudziesięciu godzin. W wielu przypadkach odbiorcy takich raportów nie są wstanie zaakceptować tak długiego czasu wykonania. W takiej sytuacji, wymagania efektywnościowe stanowią punkt wyjścia strojenia aplikacji raportujących. Poniższy artykuł przestawi metodykę strojenia i podstawowe techniki stosowane do zwiększenie efektywności aplikacji raportujących zbudowanych za pomocą *Oracle Reports*. W szczególności zostanie zaprezentowana problematyka efektywności raportów, materializacji i utrzymywania danych syntetycznych, efektywności parametryzowanych perspektyw oraz wykorzystywania operatorów ROLLUP i CUBE.

1. Wstęp

W ostatniej dekadzie obserwuje się rozszerzenie klasy użytkowników systemów informatycznych. Do tej pory głównymi celami budowy systemów informatycznych było wsparcie działań szeregowych pracowników przedsiębiorstwa, przykładowo: sprzedawców, magazynierów, kasjerów, księgowych. W kilku ostatnich latach jednym z głównych założeń systemu informatycznego jest dostarczenie narzędzi do analizy kondycji przedsiębiorstwa. Analizy takie umożliwiają kadrze kierowniczej odpowiednie zarządzanie zasobami, wyznaczanie i planowanie celów działania oraz odpowiednie dobieranie narzędzi do osiągania tych celów. Sprawny i wydajny system informatyczny stanowi silne narzędzie do utrzymywanie i zdobywania wysokiej pozycji na rynku.

Opracowanie danych wspierających decyzje kadry kierowniczej wymaga często przetworzenia wielkiego wolumenu danych. Przetwarzanie to angażuje bardzo kosztowne operacje łączenia, grupowania i sortowania danych pochodzących z wielu tabel. Implementacja aplikacji wymaga szczegółowej analizy wymagań efektywnościowych. Często takie aplikacje przetestowane na próbnej bazie danych o małym wolumenie nie dają się praktycznie uruchamiać na danych roboczych i wymagają strojenia.

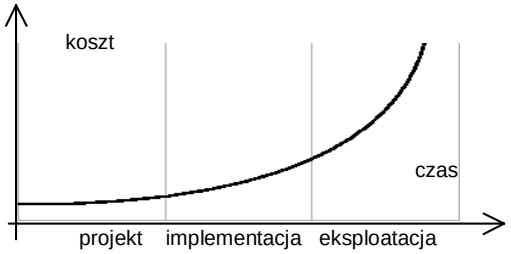
Do budowy aplikacji dla kadry kierowniczej korzysta się z różnego rodzaju narzędzi, począwszy od arkuszy kalkulacyjnych, poprzez systemy wizualizacji danych, aż po systemy eksploracji danych (ang. data mining) i systemy wspierania decyzji. Jednakże, najbardziej popularnym sposobem prezentacji danych są zestawienia analityczne sporządzone za pomocą aplikacji raportujących. Niniejszy artykuł poświęcony jest wybranym zagadnieniom strojenia aplikacji zbudowanych za pomocą narzędzi *Oracle Reports*.

W punkcie 2 została przedstawiona ogólna metodyka strojenia systemu informatycznego. Punkt 3 poświęcony jest przedstawieniu ogólnych zaleceń dotyczących efektywności aplikacji *Oracle Reports*. Punkt 4 poświęcony jest analizie efektywności parametryzowanych perspektyw. W punkcie 5 przedstawiono podstawowe techniki konstrukcji i utrzymywania materializowanych perspektyw. Punkt 6 przedstawia operatory CUBE i ROLLUP, rozszerzające składnie języka SQL. Ostatni punkt zawiera podsumowanie.

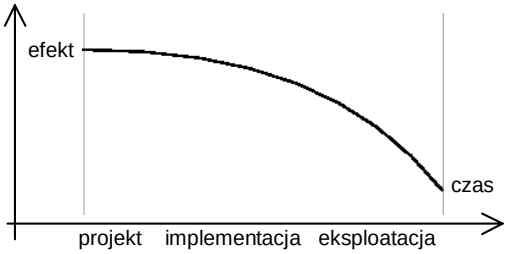
2. Metodyka strojenia systemów informatycznych

2.1. Strojenie na różnych etapach budowy systemu informatycznego

Strojenie systemu informatycznego obejmuje wszystkie fazy jego budowy, począwszy od projektu, przez implementację aplikacji, aż do wdrożenia. Kluczowymi etapami dla osiągnięcia odpowiednio dużej wydajności jest projekt i implementacja. Wykresy na rysunkach 1 i 2 przedstawiają odpowiednio koszt i efekt procesu strojenia systemu informatycznego w zależności od fazy budowy systemu, w której podjęto działanie zmierzające do poprawy wydajności. Koszt jest mierzony zaangażowaniem zasobów (głównie pracy ludzkiej) w poprawie wydajności. Efekt jest mierzony wzrostem wydajności całego systemu lub jego wybranych elementów.



Rys. 1. Wykres zależności między kosztem strojenia, a etapem budowy systemu informatycznego



Rys. 2. Wykres zależności między efektem strojenia, a czasem podjęcia decyzji o strojeniu

Największy efekt strojenia przy najmniejszym koszcie można uzyskać podejmując działania optymalizujące pracę na etapie projektu. Na tym etapie ustala się drogę obiegu dokumentów i ich kształt. Często okazuje się, że dzięki zastosowaniu systemu informatycznego z niektórych dokumentów można zrezygnować, a inne uprościć.

Faza implementacji również jest dobrym momentem do podjęcia działań zwiększających efektywność systemu. Działania te koncentrują się wokół zmian w modelu danych wprowadzających nadmiarowość danych (materializowanie danych syntetycznych), doboru odpowiednich struktur danych (np. indeksów) oraz zaplanowaniu struktury logicznej aplikacji. Dzięki odpowiednim zmianom wprowadzonym w tym etapie można kilkukrotnie zwiększyć wydajność systemu.

Najtrudniej zwiększyć efektywność systemu w fazie wdrażania lub eksploatacji. W tej fazie administratorzy starają się maksymalnie wykorzystać zasoby sprzętu i systemu operacyjnego przez ustalanie wielkości różnego rodzaju buforów, rozpraszanie i równoważenia obciążenia na urządzenia dyskowe, zrównoleglenie wykonywania operacji w środowisku wieloprocesorowym oraz eliminacji obciążenia niezwiązanego z działaniem systemu informatycznego (np. poczta elektroniczna). Pomimo dużego nakładu sił i środków strojenie w tej fazie przynosi rezultat jedynie w postaci kilkudziesięciu procent zwiększenia wydajności.

Działania podejmowane pod koniec budowy systemu informatycznego często nie przynoszą oczekiwanych rezultatów. W takim przypadku niezbędne jest przeanalizowanie i przebudowanie rezultatów poprzednich etapów. Konieczność ta potęguje koszty budowy i strojenia systemu informatycznego. Z tego powodu należy przykładać odpowiednio dużą wagę do problemu wydajności w trakcie całego cyklu budowy systemu informatycznego, począwszy już od analizy strategicznej.

2.3. Miary wydajności cel i metody strojenia

Podstawowymi miarami wydajności systemu są *czas odpowiedzi* i *przepustowość*. Czas odpowiedzi jest to czas, po którym użytkownik otrzymuje wynik swojego działania. Miara ta jest przydatna do określania wydajności aplikacji interakcyjnych np. formatek ekranowych. Przepustowość jest to liczba zadań wykonywanych przez system w jednostce czasu, przykładowo:

liczba wykonanych zleceń SQL na sekundę. Ta miara stosowana jest do opisywania aplikacji o charakterze wsadowym np. raporty.

Przed przystąpieniem do strojenia należy określić cel strojenia, np. czas odpowiedzi aplikacji służącej do rejestrowania pracowników nie może być dłuższy niż 5 sek. Strojenie ma charakter cykliczny, składa się z fazy testowania i fazy wnioskowania. Każdy cykl przybliża do rozwiązania optymalnego. W fazie testowania symulowane jest obciążenie robocze systemu i zbierane są statystyki. Statystyki winny być zbierane po pewnym czasie od rozpoczęcia symulacji, aby wypełniły się zasoby systemu. W fazie wnioskowania następuje przetworzenie zebranych statystyki, porównanie z założonymi wartościami czasu odpowiedzi i przepustowości oraz podjęcie decyzji co do zmiany konfiguracji systemu. Po tej fazie znów rozpoczyna się testowanie wprowadzonych zmian. Strojenie kończy się sukcesem, gdy osiągniemy określony cel lub się do niego zbliżymy. Rezultatem strojenia może być również zidentyfikowanie zasobu sprzętowego systemu komputerowego, które uniemożliwia osiągnięcie odpowiedniej wydajności - stanowi *wąskie gardło* systemu. W tym przypadku należy zwiększyć moc tego zasobu, np. przez zwiększenie liczby procesorów, kanałów we/wy, dysków lub ich wymianę.

3. Techniki zwiększenia efektywności wykonywania raportów

Operacje wykonywane za pomocą zleceń języka SQL zazwyczaj są wykonywane efektywniej niż za pomocą Oracle Reports lub języka PL/SQL. Poniżej przedstawiona lista zawiera najczęstsze sytuacje, gdy użycie SQL może zwiększyć wydajność:

- wykonywanie obliczeń w zapytaniu zamiast w formule lub podsumowaniu,
- użycie, do selekcji rekordów, klauzuli WHERE zamiast filtru rekordów lub wyzwalacza formatującego,
- użycie funkcji SQL SUBSTR zamiast stosowania mechanizmów Oracle Reports.

Język SQL wykonuje obliczenia szybciej niż Oracle Reports za pomocą formuł lub podsumowań. Klauzula WHERE zbędne operacje pobrania danych z serwera, ponieważ wykonywane są podczas wyszukiwania danych i serwer może wykorzystać dodatkowe struktury danych (np. indeksy). Użycie funkcji SUBSTR minimalizuje wolumen przesyłanych danych między serwerem i klientem.

Polecenia SQL w funkcji SRW.DO_SQL

Funkcja SRW.DO_SQL umożliwia wykonanie w raporcie dowolnego polecenia DDL lub DML. Funkcja ta jest bardzo pożyteczna, lecz może znacznie pogorszyć wydajność jeżeli będzie stosowana nierozsądnie. Polecenia SQL wykonywane przez funkcję SRW.DO_SQL są analizowane składniowo, otwierany jest dla nich kursor, a następnie są wykonywane. W przeciwieństwie do zapytań zdefiniowanych w modelu danych raportu, SRW.DO_SQL wykonuje wszystkie te operacje za każdym razem, gdy grupa związana z tą funkcją pobiera dane. Przykładowo: jeżeli funkcja SRW.DO_SQL związana jest grupą, która pobiera 10 rekordów to polecenie jest 10 razy analizowane składniowo, 10 razy jest otwierany kursor i 10 razy jest wykonywane. W celu uniknięcia takich przypadków, należy wykonać obliczenia za pomocą zapytania zdefiniowane w modelu da danych lub programu w PL/SQL zamiast funkcji SRW.DO_SQL związanej z grupą.

Procedura CDER_MM.GET_REF

Procedura CDE_MM.GET_REF została zaimplementowana w celu zmniejszenia wielkości tymczasowej przestrzeni zajmowanej przez Oracle Reports. Raporty nie buforują w pliku tymczasowym kolumn odczytanych przez funkcję CDE_MM.GET_REF. Chociaż wykorzystanie tej funkcji redukuje zapotrzebowanie na przestrzeń tymczasową, to powoduje również zmniejszenie wydajności, ponieważ wartości kolumny muszą być zawsze odczytane z bazy danych.

Stosowanie wielu zapytań w modelu danych

W miarę możliwości należy zmniejszyć liczbę zapytań w modelu danych. Mniejsza liczba zapytań (nawet bardziej złożonych) często jest wykonywana przez raport szybciej. Model z wieloma zapytaniami jest łatwiejszy do zrozumienia i przeanalizowania, lecz model z jednym zapytaniem jest o wiele bardziej wydajny.

Model z wieloma zapytaniami należy stosować jedynie w przypadku, gdy:

- odczyt dotyczy wielu dużych kolumn w grupie nadrzędnej i jedynie niewielu małych kolumn z grupy podrzędnej,
- należy wykonać operację, która nie jest wspierana bezpośrednio prze polecenie SELECT (np. wielokrotne połączenie zewnętrzne),
- należy wykorzystać skomplikowaną perspektywę (np. zawierającą zapytania rozproszone lub operację grupowania,
- nie można użyć perspektyw.

W prostych raportach, dla pobrania rekordów nadrzędnych i podrzędnych, jest otwierany jedynie jeden kursor. W raportach z dwoma zapytaniami, Oracle Reports otwiera dwa kursowy (po jednym dla każdego zapytania) po uzupełnieniu klauzuli WHERE zapytania podrzędnego zmienną wiążącą. Dla każdego rekordu nadrzędnego, raporty muszą ustalić nową wartość zmiennej wiążącej, wykonać i pobrać wynik z zapytania podrzędnego.

Stosowanie indeksów

Stosowanie indeksów znacznie zwiększa wydajność działania raportów. W szczególności należy definiować indeksy na:

- kolumnach występujących w klauzuli WHERE,
- kluczach podstawowych, unikalnych i obcych,
- atrybutach wiążących zapytanie nadrzędne z zapytaniami podrzędnymi.

Stosowanie indeksów na tabeli nadrzędnej ma mały wpływ na wydajność raportów, ponieważ zapytanie kierowane do tych tabel wykonywane są jedynie raz. Indeksy znacząco zwiększają efektywność raportów typu *narzędny-podrzędny*. Wraz ze zmniejszaniem stosunku rekordów nadrzędnych do podrzędnych rośnie znaczenie indeksów zdefiniowanych dla tabeli podrzędnej. Indeksy są zalecane dla tabel podrzędnych, ponieważ Oracle Reports jawnie dodaje do klauzuli WHERE zapytania podrzędnego wyrażenie reprezentujące związek *narzędny-podrzędny*.

Łamanie grup rekordów

Liczebność kolumn w grupie definiujących łamanie grup rekordów będących wynikiem zapytania powinna być jak najmniejsza. Należy starać się aby na jedną taką grupę przypadała jedna kolumna. Wielkość takich kolumn też powinna być jak najmniejsza. Kolumna o mniejszym rozmiarze zazwyczaj daje lepszą wydajność niż kolumna o większym rozmiarze. Dla dużych kolumn, zwiększenie wydajności można uzyskać przez zastosowanie funkcji SUBSTR w celu zredukowania ich rozmiaru.

W przypadku zdefiniowania łamania grupy, Oracle Report dodaje do klauzuli ORDER BY zapytania wszystkie kolumny określające to łamanie. Wyjątek stanowi kolumna zdefiniowana za pomocą formuły. Dzięki zminimalizowaniu liczby kolumn w grupie określającej złamanie strony, minimalizuje się również liczbę kolumn w klauzuli ORDER BY. Przy mniejszej liczbie kolumn operacja sortowania jest mniej kosztowna. Również rozmiar klucza indeksu, który Oracle Reports wewnętrznie buduje na potrzeby sortowania jest mniejszy. Dzięki temu wydajność raportu zawierającego małą liczbę kolumn o małej wielkości jest większa.

Ograniczenie liczby wierszy i filtrowanie rekordów

Przy projektowaniu raportów, które przetwarzają wielkie wolumeny danych, często występuje potrzeba ograniczenia wielkości wyszukanych danych (np. podczas testowania) w celu skrócenia czasu wykonywania raportu. W takich przypadkach można skorzystać własności zapytania o nazwie *Maximum Rows*, własność ta ogranicza liczbę odczytanych przez zapytanie rekordów.

Własność *Maximum Rows* ogranicza liczbę przetworzonych rekordów do wskazanej wartości, natomiast filtrowanie określa, które rekordy mają znaleźć się na raporcie, a które mają być wyłączone. Z tego powodu w większości przypadków własność *Maximum Rows* jest bardziej efektywna niż filtrowanie.

Jeżeli stosuje się filtrowanie z własnościami *Last* lub *Conditional*, to Oracle Reports musi odczytać wszystkie rekordy w grupie i następnie zastosować kryteria filtrowania. W takich przypadkach zastosowanie własności zapytania *Maximum Rows* lub filtru z własnością *First* jest bardziej efektywne.

Nieużywane obiekty w modelu danych

Należy się upewnić, że wszystkie obiekty nieużywane w raporcie zostały usunięte lub ukryte. Jeżeli model danych zawiera zapytanie, które jest wykorzystywane w sporadycznych przypadkach (np. przez ustawienie parametru wywołania raportu), to można je ukryć warunkowo za pomocą procedury SRW.SET_MAXROW. Wywołanie procedury w postaci SRW.SET_MAXROW (queryname, 0) powoduje, że zapytanie nie pobierze żadnego rekordu.

Zbędne ramki

Należy usunąć wszystkie zbędne ramki z projektu wyglądu raportu. Kiedy Oracle Reports tworzy domyślny wygląd raportu, to umieszcza w własnych ramkach wszystkie obiekty raportu. W tem sposób zabezpiecza się obiekty przed nałożeniem innymi obiektami. Jeżeli wiadomo, że nie istnieje niebezpieczeństwo nałożenia się obiektów, to można usunąć ramki bez szkody na wygląd raportu. Dzięki mniejszej liczbie obiektów (ramek), przetwarzanie raportu jest mniej kosztowne i raport może być wykonany szybciej.

Całkowita liczba stron

W miarę możliwości przy numerowaniu stron należy ograniczać stosowanie całkowitej liczby stron jako źródło pola (*Total Logical Pages*). Kiedy stosuje się źródło *Total Logical Pages* raport musi zapisać wszystkie strony raportu w przestrzeni tymczasowej aby określić liczbę stron. Operacja ta znacząco zwiększa zaalokowaną na dysku przestrzeń tymczasową i generuje dodatkowe operacje we/wy, które obniżają wydajność raportu.

Wyzwalacze formatujące

Wyzwalacze formatujące należy definiować dla obiektów o najniższej możliwej częstotliwości. Zazwyczaj umieszczenie wyzwalacza na poziomie ramki zamiast na poziomie pola zwiększa szybkość raportu.

Podprogramy PL/SQL definiujące wyzwalacze formatujące wykonywane są dla każdego wystąpienia obiektu. Dla obiektów o małej częstotliwości drukowania, kod PL/SQL jest wykonywany rzadziej i efekcie raport zakończy swoje działanie wcześniej.

Integracja Oracle Graphics

Jeżeli wykres Oracle Graphics wywoływany przez raport wykorzystuje wybrane lub wszystkie dane wyszukane przez raport, to należy dane te przekazać z raportu do wykresu. Taka technika redukuje liczbę wykonywanych zapytań i wolumen przesłanych danych między serwerem i aplikacją. Jeżeli dane nie zostaną przekazane to dane są przetwarzane dwukrotnie: pierwszy raz dla raportu, drugi raz dla wykresu.

4. Parametryzowane perspektywy

Perspektywy, zdefiniowane w postaci zapytań SQL składowanych w słowniku bazy danych, umożliwiają budowanie prostego, spójnego interfejsu dla aplikacji raportujących. Utrzymanie jednolitego sposobu dostępu do złożonych struktur danych jest niezwykle istotne dla projektów, w którym uczestniczy duża grupa osób. Dzięki takiemu interfejsowi unika się nieporozumień w interpretacji związków między poszczególnymi tabelami modelu danych. W skrajnym przypadku różna interpretacja znaczenia poszczególnych elementów danych może doprowadzić do skonstruowania aplikacji, których wynik działania dla tego samego zbioru danych jest diametralnie różny.

Dużą zaletą perspektyw jest możliwość wykorzystywania ich w wielu aplikacjach. Jednakże zakres danych przetwarzanych przez te aplikacje może się różnić. W celu zwiększenia elastyczności perspektyw stosuje się ich parametryzację. Wynik zapytania skierowanego do parametryzowanej perspektywy zależy od parametrów, których wartość ustala się przed wykonaniem zapytania. Najprostszym sposobem konstrukcji parametryzowanej perspektywy jest wykorzystanie, w zapytaniu definiującym tą perspektywę, specjalnej tabeli zawierającej wartości parametrów. Tabela ta zawiera jeden rekord, a poszczególne pola reprezentują wartości parametrów.

Parametryzowane perspektywy są wygodnym sposobem przekazania kryteriów filtrowania danych z aplikacji formularza do aplikacji raportu. Aplikacja formularza wstawia do tabeli rekord zawierający parametry ustalone przez użytkownika i następnie wywołuje raport. Raport uruchamia zapytanie oparte na perspektywie, a jego wynik odpowiada parametrom wprowadzonych przez użytkownika. Poprawne, współbieżne wykorzystanie tej samej parametryzowanej perspektywy przez wielu użytkowników można zagwarantować stosując unikalne identyfikatory rekordów w tabeli z parametrami. Wartości tych parametrów można przekazywać przy wywoływaniu raportu z formularza. Innym sposobem jest wykorzystanie własności izolacji transakcji: formularz po wstawieniu rekordu z parametrami nie zatwierdza zmian i w ramach tej samej transakcji uruchamia raport, po zakończeniu działania raportu następuje wycofanie zmian wprowadzonych do tabeli z parametrami. Zmiany w tabeli z parametrami nie są obserwowane przez inne równolegle wykonywane raporty. Stosowanie samodzielnego formularza w stosunku do zintegrowanego z raportami formularza parametrów umożliwia wielokrotne wykorzystania tego formularza oraz stosowanie bardziej zaawansowanego interfejsu (np. zależnych od siebie list wartości).

Poniżej przedstawiono przykładową parametryzowaną perspektywę o nazwie *kadry*. Tabela *pracownicy* zawiera informację o pracownikach, tabela *zespoły* zawiera dane o zespołach zatrudniających pracowników, natomiast tabela *parametry* zawiera rekord parametryzujący perspektywę.

```
create view kadry as
select nazwisko, etat, nazwa_zespołu
from pracownicy p, zespoły z, parametry par
where p.id_zesp=z.id_zesp
and
    (p.etat= par.etat or par.etat='%')
and
    (z.nazwa_zespołu=par.nazwa_zespołu or par.nazwa_zespołu='%')
```

Część definicji parametryzująca perspektywę została zaznaczona kursywą. Powyższa perspektywa została sparametryzowana ze względu na etat, na którym jest zatrudniony pracownik i nazwę zespołu, do którego należy. W tabeli *parametry*, atrybuty *etat* i *nazwa_zespołu* umożliwiają odfiltrowanie danych. Podstawienie pod parametr znaku procent umożliwia wskazanie, że dane nie mają być filtrowane ze względu na dany parametr.

Frazę parametryzującą perspektywę można zapisać również z wykorzystaniem operatora LIKE, np:

```
(p.etat like par.etat).
```

Jednakże użycie operatora LIKE uniemożliwia optymalizatorowi wykorzystanie indeksu zbudowanego na atrybucie *etat* tabeli *pracownicy*.

Wraz ze wzrostem złożoności perspektywy (w szczególności liczby tabel) i liczby parametrów rośnie wykładniczo liczba kombinacji według których można połączyć tabele. W przypadku bardzo dużych perspektyw autor zaobserwował, że optymalizator opracowywał plan wykonania, który był bardzo daleki od optymalnego. Optymalizator wykorzystywał fakt małej liczności rekordów w tabeli z parametrami i rozpoczynał łączenie od tej tabeli zakładając, silne zawężenie wynikowego zbioru danych. Założenie to jest zbyt uproszczone, gdyż stopień odfiltrowania danych głównie zależy od wartości rekordu w tabeli z parametrami.

Powyższa parametryzacja ma charakter statyczny - tekst polecenia SQL jest znany podczas kompilacji aplikacji. Rozwiązanie to okazało się mało skalowalne, przy bardzo dużych perspektywach i przy wielkiej liczbie parametrów znalezienie optymalnego planu wymaga przeanalizowanie gigantycznej liczby możliwych kombinacji połączenia tabel i stosownej do rozmiaru problemu mocy obliczeniowej.

Rozwiązaniem tego problemu jest zastosowanie dynamicznej parametryzacji perspektyw. W tym podejściu nie stosuje się tabeli z parametrami, dzięki czemu maleje liczba kombinacji połączenia tabel i optymalizator ma większe możliwości znalezienia optymalnego planu wykonania zapytania. Sposób filtrowania danych jest opracowywany po wprowadzeniu kryteriów użytkownika do formularza. Przed wywołaniem raportu następuje konstrukcja klauzuli WHERE uwzględniająca kryteria użytkownika. Tekst tej klauzuli jest przekazywany do raportu i następnie raport dynamicznie dołącza ten tekst do zapytania zdefiniowanego w modelu danych raportu, które odwołuje się do wcześniej zbudowanej perspektywy. Dodatkową zaletą tego podejścia jest możliwość wykorzystania przez optymalizator kosztowy zaawansowanych statystyk danych (tzw. histogramów). Dynamiczne parametryzowanie perspektyw można zaimplementować za pomocą mechanizmu *składniowych referencji* (ang. *lexical references*). Mechanizm ten umożliwia dynamiczną konstrukcję zapytania podczas wykonywania raportu.

Do zilustrowania dynamicznej parametryzacji perspektyw zostanie wykorzystana perspektywa *kadry2* zdefiniowana następująco:

```
create view kadry2 as
select nazwisko, etat, nazwa_zespołu
from pracownicy p, zespoły z
where p.id_zesp=z.id_zesp
```

W celu wykorzystania składniowej referencji należy zdefiniować w raporcie parametr użytkownika typu tekstowego, w naszym przykładzie o nazwie *my_where*. Następnie można przystąpić do budowy zapytania w modelu danych, które odwołuje się do perspektywy i wcześniej zdefiniowanego parametru. Przykładowe takie zapytanie podano poniżej.

```
select nazwisko, etat, nazwa_zespołu
from kadry2
&my_where
```

Po zdefiniowaniu wyglądu raportu, można go uruchomić. Zakres wyświetlonych danych zależy od wartości parametru *my_where*, którego wartość można ustalić tuż przed uruchomieniem zapytania. Przykładowo, jeżeli dane mają być zawężone jedynie do zespołu o nazwie *Administracja,* to parametr *my_where* powinien przyjąć wartość *where nazwa_zespołu='Administracja'*. Jeżeli natomiast zawężenie ma również etatu *Sekretarka,* to wartość parametru *my_where* przyjmie kształt: *where nazwa_zespołu='Administracja' and etat='Sekretarka'*. Jeżeli raport ma zawierać wszystkie dane dostępne przez perspektywę *kadry2*, to parametr *my_where* powinien zostać ustawiony na wartość pustą.

5. Materializowanie i utrzymywanie agregatów

Pomimo stosowania różnego rodzaju technik optymalizacji aplikacji raportujących, nie zawsze można zagwarantować satysfakcjonująco krótki czas wykonania raportu na podstawie danych źródłowych. W takich sytuacjach należy wyznaczyć wartości niezbędnych wskaźników przed uruchomieniem raportu i zapisać je w bazie danych. Taki zabieg nazywamy materializacją agregatów.

Wartości zmaterializowanych agregatów powinny nadążać za zmianami w danych źródłowych. W zależności od preferencji użytkowników, wymagań wydajnościowych i możliwości sprzętowych mogą być one uaktualniane: synchronicznie wraz ze zmianami w bazie danych (na poziomie polecenia lub transakcji), z zadanym interwałem czasowym lub na żądanie.

Utrzymywanie agregatów może być wykonywane techniką przyrostową. W technice tej nie wylicza się agregatów na podstawie całego zbioru danych źródłowych, lecz na podstawie poprzedniej wartości agregatu i zbioru wprowadzonych zmian na danych źródłowych. Technika ta ma pewne ograniczenia, przykładowo: nie ma możliwości przyrostowego utrzymywania agregatów zawierających wyrażenie: *SUM(DISTINCT wyrażenie)*.

W Oracle8i istnieje szereg mechanizmów umożliwiających budowanie i utrzymywanie zmaterializowanych agregatów, do których można zaliczyć:

- wyzwalacze (ang. triggers),
- zadania (ang. jobs),
- materializowane perspektywy (ang. materialized views), zwane również migawkami (ang. snapshots).

W dalszej części punktu zostaną opisane i zilustrowane przykładami powyższe techniki.

5.1 Wyzwalacze

Wyzwalacze są jednostkami, składowanych w bazie danych, programów PL/SQL lub Javy, które są uruchamiane przez serwer w momencie zmiany stanu bazy danych. Stan bazy danych może być zmieniony jednym z trzech poleceń DML: INSERT, UPDATE, DELETE. Wyzwalacze są prostym mechanizmem umożliwiającym implementację przyrostowego utrzymywania zmaterializowanych danych. Dzięki temu, że wyzwalacze mogą być uruchamiane w tej samej transakcji co polecenie modyfikujące bazę danych, to technika ta gwarantuje całkowitą spójność między agregatami i danymi źródłowymi. Wadą tego podejścia jest mała efektywność, gdyż procedura implementująca wyzwalacz musi być uruchomiona dla każdego wstawianego, zmienianego lub usuwanego rekordu z tabeli źródłowej.

Dla zilustrowania tej techniki załóżmy, że w tabeli *zespoły* znajduje się kolumna *liczba_pracowników*, która zawiera wartość odpowiadającą liczbie pracowników w danym zespole. Kolumna ta może być zainicjowana za pomocą następującego polecenia

```
update zespoły z
set liczba_pracowników=(select count(*) from pracownicy
                        where id_zesp=z.id_zesp)
```

W celu utrzymywania aktualnej wartości tej kolumny, można zbudować wyzwalacz przedstawiony poniżej.

```
create or replace trigger utrzymuj_agregat
after insert or delete or update of id_zesp
on pracownicy
for each row
begin
 if inserting or updating then
```

```
    update zespoly
    set liczba_pracownikow=liczba_pracownikow+1
    where id_zesp=:new.id_zesp;
 end if;
 if deleteting or updating then
    update zespoly
    set liczba_pracownikow=liczba_pracownikow-1
    where id_zesp=:old.id_zesp;
 end if;
end;
```

Wyzwalacz *utrzymaj_agregat* zostanie uruchomiony dla każdego rekordu wstawionego lub usuniętego z tabeli *pracownicy*, co odpowiada zatrudnieniu lub zwolnieniu pracownika. Uruchomienie wyzwalacza nastąpi również w przypadku zmiany wartości atrybutu *id_zesp*, oznacza to zmianę miejsca zatrudnienia pracownika. W przypadku zatrudnienia nowego pracownika zostanie zwiększona liczba pracowników w zespole, którego identyfikator odpowiada zespołowi do którego pracownik został przydzielony. Taka sama operacja zostanie również wykonana w przypadku zmiany zespołu przez pracownika. Jeżeli pracownik zostanie zwolniony lub zmieni zespół, w którym pracuje, to dotychczasowa liczba pracowników jego zespołu zostanie zmniejszona.

5.2 Zadania

Zadania (ang. jobs) są mechanizmem umożliwiającym cykliczne wykonywanie składowanych w bazie danych programów PL/SQL i Javy. Mogą być one wykorzystane do okresowego wyznaczania wartości zmaterializowanych wartości. Zaletą tego podejścia jest całkowita integracja mechanizmu uruchamiania procedur z systemem bazy danych. Technika ta zostawia projektantowi pełną swobodę w implementacji sposobu wyznaczenia wartości zmaterializowanych agregatów. Poziom spójności danych źródłowych i ich agregatów zależy od zadeklarowanej częstotliwości uruchamiania zadań.

W celu prezentacji opisanej techniki do tabeli *zespoły* dodano kolumnę *liczba_etatów* reprezentującą liczbę różnych etatów, na których zatrudniono pracowników w danym zespole. Wykorzystanie techniki wyzwalaczy do utrzymywania tego agregatu jest mało efektywne, gdyż nie ma możliwości utrzymywania go przyrostowo.

Do obsługi zadań służy pakiet DBMS_JOB. Procedura SUBMIT z tego pakietu służy do zlecenia systemowi wykonywania ze wskazanym okresem podanego programu PL/SQL. Poniżej zamieszczono program PL/SQL wykorzystujący procedurę SUBMIT do cyklicznego uruchamiania polecenia wyznaczającego różnych liczbę etatów, na których zatrudniono pracowników w danym zespole.

```
declare
nr_zadania number;
tekst_plsql varchar2(300);
begin
tekst_plsql:='update zespoly z
           set liczba_etatow =(select count(distinct etat)
                               from pracownicy
                               where z.id_zesp=id_zesp);';
dbms_job.submit(nr_zadania, tekst_plsql, sysdate, 'sysdate+1/24');
end;
```

Pierwszy parametr procedury SUBMIT zwraca identyfikator uruchomionego zadania. Identyfikator ten służy do dalszej obsługi zadania za pomocą innych procedur pakietu DBMS_JOB, np. zatrzymanie zadania. Wartość tego identyfikatora można również odczytać z systemowej perspektywy o nazwie USER_JOBS. Drugi parametr przekazuje tekst polecenia SQL (lub programu PL/SQL), które ma być wykonywane w ramach definiowanego zadania. Trzeci parametr przekazuje

datę pierwszego wykonania zadania. Czwarty parametr przekazuje tekst wyrażenia, które służy do wyznaczenia kolejnego wykonania zadania. W powyższym przykładzie zostanie uruchomione zadanie wykonujące tekst polecenia przekazanego zmienną *tekst_plsql*. Pierwsze wykonanie zadania odbędzie się bezzwłocznie po jego uruchomieniu. Kolejne wykonania zostaną zainicjowane godzinę po zakończeniu poprzedniego wykonania.

W celu uruchomienia kolejki zadań należy ustawić parametr inicjacyjny JOB_QUEUE_PROCESSES (w pliku init<SID>.ora) na wartość odpowiadającą liczbie wymaganych procesów do obsługi tej kolejki. Liczba ta powinna odpowiadać średniej liczbie współbieżnie wykonywanych zadań.

5.3 Materializowane perspektywy

Mechanizm materializowanych perspektyw (ang. materialized views), wprowadzony w Oracle8i, jest rozwinięciem koncepcji migawek (ang. snapshots) – mechanizmu dostępnego już w Oracle7. Mechanizm ten opiera się na deklaratywnym definiowaniu zmaterializowanych agregatów za pomocą zapytania SQL. Istnieje możliwość uaktualniania agregatów synchronicznie wraz ze zmianami w bazie danych (na poziomie transakcji), z zadanym interwałem czasowym lub na żądanie. Utrzymywanie agregatów może być, z pewnymi ograniczeniami, wykonywane przyrostowo. Przyrostowe utrzymywanie agregatów wymaga utworzenia dzienników dla wszystkich tabel, które wchodzącą w skład definicji perspektywy. Dzienniki te rejestrują wszystkie zmiany wykonane na wskazanych tabelach.

W porównaniu do wcześniejszej wersji tego mechanizmu (tj. migawek):

- rozszerzono klasę materializowanych perspektyw, które mogą być utrzymywane przyrostowo (np. włączono zapytania zawierające operację połączenia i funkcje grupowe),
- dodano możliwość zapewnienia spójności między agregatami i danymi źródłowymi na poziomie transakcji,
- umożliwiono przekierowywanie zapytań (ang. query rewrite) – zapytania odwołujące się do danych źródłowych mogą być wykonane na podstawie zawartości zmaterializowanych perspektyw.

Do utworzenia zmaterializowanej perspektywy wymagany jest przywilej systemowy CREATE MATERIALIZED VIEW. Obsługa zmaterializowanych perspektyw wymaga dodatkowo ustawienia parametrów inicjalizacyjnych (w pliku init<SID>.ora) JOB_QUEUE_PROCESSES i JOB_QUEUE_INTERVAL.

Dalsza część punktu zostanie poświęcona przykładom definiowania materializowanych perspektyw. Poniższe polecenie umożliwia zbudowanie perspektywy zawierającej informacje o liczbie różnych etatów, na których są zatrudnieni pracownicy w poszczególnych zespołach.

```
create materialized view liczba_etatów
build deferred
refresh complete
on demand
as
select id_zesp, count(distinct etat) as liczba_etatów
from pracownicy
group by id_zesp
```

W powyższym przykładzie, klauzula BUILD DEFFERED oznacza, że po utworzeniu perspektywa nie zawiera zmaterializowanych danych, zostanie ona wypełniona przy pierwszym uaktualnieniu. Klauzula REFRESH COMPLETE określa, że perspektywa będzie uaktualniana przez pełne wykonanie zapytania na danych źródłowych. Klauzula ON DEMAND wskazuje, że perspektywa będzie odświeżana na żądanie. Do obsługi materializowanych perspektyw służy pakiet

DBMS_MVIEW. Za pomocą procedury REFRESH można uaktualnić wskazaną perspektywę. Poniższe polecenie wymusza uaktualnienie zmaterializowanej perspektywy *liczba_etatów*.

```
SQL> exec dbms_mview.refresh('liczba_etatów')
```

Poniższy przykład ilustruje konstrukcję zmaterializowanej perspektywy *budżety_zespołów*, która zawiera informację o sumie płac pracowników w każdy zespole.

```
create materialized view budżety_zespołow
refresh complete
start with sysdate next sysdate+1
as
select nazwa_zespołu, sum(placa) as suma_płac
from pracownicy p, zespolu z
where p.id_zesp=z.id_zesp
group by nazwa_zespołu
```

Klauzule START WITH oraz NEXT służą do zdefiniowania cyklicznego uaktualniania perspektywy. Pierwsze uaktualnienie nastąpi natychmiast po utworzeniu perspektywy, kolejne po upływie jednej doby po zakończeniu poprzedniego uaktualnienia.

Kolejny przykład przedstawia zmaterializowaną perspektywę, która jest uaktualniana przyrostowo. Perspektywa ta, o nazwie *średnie_dochody*, zwiera dane na temat średniej płacy pracowników zatrudnionych na poszczególnych etatach. Przyrostowe utrzymywanie zmaterializowanych perspektyw wymaga zdefiniowania dzienników zmian dla tabel źródłowych. Poniższe polecenie przedstawia sposób konstrukcji takiego dziennika dla tabeli *pracownicy*.

```
create materialized view log on pracownicy
with rowid (etat, płaca)
log new values
```

Klauzula WITH ROWID wymusza przechowywanie w dzienniku identyfikatora rekordu, który został zmieniony w tabeli źródłowej. Klauzula LOG NEW VALUES umożliwia składowanie w dzienniku starych i nowych wartości atrybutów *etat* oraz *płaca*. Są one niezbędne do przyrostowego utrzymywania agregatów. Po utworzeniu dziennika można przystąpić do konstrukcji zmaterializowanej perspektywy *średnie_dochody*, której tekst przedstawiono poniżej.

```
create materialized view średnie_dochody
refresh fast
on commit
enable query rewrite
as
select etat, count(*) as liczba_prac, count(płaca) as liczba_płac,
          avg(płaca) as średnia
from pracownicy
group by etat
```

Klauzula REFRESH FAST wskazuje przyrostowe utrzymywanie agregatów w zmaterializowanej perspektywie. Spójność między danymi źródłowymi i perspektywą wymusza klauzula ON COMMIT. Wszystkie zmiany wprowadzone do tabel źródłowych będą bezzwłocznie uwzględnione w perspektywie przy zatwierdzeniu transakcji, która je wprowadziła. Klauzula ENABLE QUERY REWRITE umożliwia przekierowywanie zapytań dotyczących tabel źródłowych do materializowanej perspektywy. W celu zagwarantowania możliwości przyrostowego utrzymania agregatu *avg(płaca)*, klauzula SELECT zapytania musi zawierać wszystkie wyrażenie z klauzuli GROUP BY, funkcję grupową *count(*)* oraz *count(płaca)*.

W celu skorzystania z własności przekierowania zapytań należy ustawić w pliku init<SID>.ora parametr inicjacyjne QUERY_REWRITE_ENABLED na wartość *true*, parametr OPTIMIZER_MODE na wartość *choose*, oraz parametr COMPATIBLE na wartość *8.1.0* (lub

większą). Ponadto użytkownik wykonujący zapytanie powinien posiadać przywilej systemowy QUERY REWRITE.

Mechanizm przekierowania zapytań wykorzystuje optymalizację kosztową do określenia, czy skorzystanie ze zmaterializowanych danych jest bardziej opłacalne niż dostęp do danych źródłowych. Z tego powodu należy zebrać statystyki dla wszystkich tabel źródłowych oraz zmaterializowanych perspektyw, dla których chcemy wykorzystać ten mechanizm. Poniżej przedstawiono polecenia wyliczające pełne statystki dla tabeli *pracownicy* i perspektywy *średnie_dochody*.

```
SQL> analyze table pracownicy compute statistics;

SQL> analyze table średnie_dochody compute statistics;
```

Po spełnieniu powyższych wymagań pewna grupa zapytań do tabeli *pracownicy* może być przekierowana do zmaterializowanej perspektywy *średnie_dochody*. Przykładowo poniższe zapytanie, znajdujące sumę płac pracowników w poszczególnych zespołach, może zostać przekierowane do perspektywy *średnie_dochody*.

```
select etat, sum(płaca)
from pracownicy
group by etat
```

Powyższe zapytanie w celu przekierowania do perspektywy *średnie_dochody* zostanie przetransformowane na przedstawione poniżej polecenie.

```
select etat, avg(płaca)*count(płaca)
from średnie_dochody
```

Powyższe zapytanie wykorzystuje zmaterializowane wartości średniej i liczby płac pracowników w celu obliczenia wartości średniej. Z planem zapytania, przekierowanego do zmaterializowanej perspektywy, można się zapoznać za pomocą polecenia EXPLAIN PLAN.

6. Operatory CUBE i ROLLUP

Operatory CUBE i ROLLUP są rozszerzeniem klauzuli GROUP BY i umożliwiają wyliczanie agregatów cząstkowych dla grup rekordów. Rozszerzenie to jest propozycją standardu SQL3 i zostało zaimplementowane w systemie Oracle w wersji 8.1. Stosowanie tych operatorów znacznie ułatwia kodowanie aplikacji raportujących, w szczególności raportów matrycowych. Dzięki temu, że sumy cząstkowe są wyliczane na serwerze to operacje te mogą być zrównoleglone i wykonane efektywniej. Stosowanie operatorów CUBE i ROLLUP umożliwia zminimalizowanie liczby zapytań do bazy danych, tym samym minimalizuje liczbę przesłanych pakietów danych między aplikacją klienta i serwerem oraz zmniejsza obciążenie łącza komunikacyjnego.

Poniżej przedstawiono zapytanie, które wykorzystuje operator ROLLUP dla wyliczenia średniej płacy pracowników na poszczególnych etatach w określonych zespołach.

```
select etat, id_zesp, avg(płaca)
from pracownicy
group by rollup(etat, id_zesp)

ETAT         ID_ZESP AVG(PLACA_POD)
---------- --------- --------------
ADIUNKT           20         617,75
ADIUNKT                      617,75
ASYSTENT          20      430,23333
ASYSTENT          30            480
ASYSTENT                    442,675
                           501,03333
```

Powyższe zapytanie oprócz wyliczenia wartości średnich dla określonych etatów i zespołów dokonuje wyliczenia średniej również dla wszystkich pracowników zatrudnionych na określonym etacie oraz dla wszystkich pracowników (fragment wydruku zaznaczony kursywą).

Operator CUBE dokonuje wyliczenia wartości agregatu dla wszystkich kombinacji grup rekordów, które można skonstruować na podstawie wyrażeń umieszczonych w klauzuli GROUP BY.

Poniższy przykład ilustruje wykorzystanie operatora CUBE.

```
select etat, id_zesp, avg(płaca)
from pracownicy
group by cube(etat, id_zesp)

ETAT         ID_ZESP AVG(PLACA_POD)
---------- --------- --------------
ADIUNKT           20         617,75
ADIUNKT                      617,75
ASYSTENT          20      430,23333
ASYSTENT          30            480
ASYSTENT                    442,675
                  20         505,24
                  30            480
                         501,03333
```

W powyższym przykładzie, zostały wyznaczone również średnie płace pracowników wyliczone dla poszczególnych zespołów (zaznaczone kursywą). Dla wskazania, że agregat cząstkowy dotyczy większej liczby grup, system wykorzystuje wartość pustą (ang. null value). W celu odróżnienia tego wskazania od wartości pustej składowanej w bazie danych można wykorzystać funkcję GROUPING. Argumentem tej funkcji jest wyrażenie z klauzuli GROUP BY. Funkcja GROUPING zwraca wartość 0, gdy wartość pochodzi z bazy danych, oraz wartość 1 gdy jest ona wskazaniem agregatu cząstkowego.

W poniższym przykładzie zastosowano funkcję GROUPING do wskazania, że agregat cząstkowy dotyczy większej liczby etatów i zespołów.

```
select decode(grouping(etat),1,'Wszystkie etaty', etat) as etat,
    decode(grouping(id_zesp),1,'Wszystkie zespoły',id_zesp) as id_zesp,
    avg(płaca)
from pracownicy
group by cube(etat, id_zesp)

ETAT            ID_ZESP                AVG(PŁACA)
--------------- ---------------------- --------------
ADIUNKT         20                           617,75
ADIUNKT         Wszystkie zespoły            617,75
ASYSTENT        20                        430,23333
ASYSTENT        30                              480
ASYSTENT        Wszystkie zespoły           442,675
Wszystkie etaty 20                           505,24
Wszystkie etaty 30                              480
Wszystkie etaty Wszystkie zespoły         501,03333
```

Po odpowiednim sformatowaniu wynik powyższego zapytanie może użyty zamiast raportu matrycowego. Poniżej przedstawiono przykład wydruku takiego raportu.

ŚREDNIE PŁACE	ADIUNKT	ASYSTENT	Wszystkie etaty
Zespół 20	617,75	430,23333	505,24
Zespół 30		480	480
Wszystkie zespoły	617,75	442,675	501,03333

7. Podsumowanie

W poniższej pracy przedstawiono podstawowe techniki strojenia aplikacji raportujących. Zaprezentowano problematykę związaną z efektywnością raportów, materializacją i utrzymywaniem danych syntetycznych, efektywnością parametryzowanych perspektyw oraz wykorzystywaniem operatorów ROLLUP i CUBE.

Należy wspomnieć, że opisane techniki nie wyczerpują wszystkich dostępnych środków dla zwiększenia wydajności raportów. Z powodu ograniczonej objętości niniejszej pracy nie poruszono szerokiej tematyki związanej z równoległym wykonywaniem zapytań z wykorzystaniem architektur masywno-równoległych oraz procesorów symetrycznych. Przedstawienie tej problematyki będzie celem kolejnej pracy.

Bibliografia

1. Wrembel R., Jezierski J. Zakrzewicz M., System zarządzania bazą danych Oracle7 i Oracle8, Wydawnictwo Nakom, Poznań 1999, ISBN 83-86969-34-2
2. Oracle Corporate Support, ORACLE REPORTS PERFORMANCE TIPS, Oracle Corporation 1995-96, ID:9402099.61
3. Bauer M. Et al., Oracle8i Tuning, Release 8.1.5, Oracle Corporation 1999, No. A67775-01
4. Raphaely D. et al., Application Developer's Guide - Fundamentals, Release 8.1.5, Oracle Corporation 1999, No. A68003-01