STUDIA INFORMATICA 2011
Volume 32 Number 2B (97)

Krzysztof CZAJKOWSKI, Marek BANACH
Politechnika Krakowska, Instytut Teleinformatyki,
Wydział Fizyki, Matematyki i Informatyki

AUTOMATYCZNE PRZEPISYWANIE ZAPYTAŃ Z WYKORZYSTANIEM WIDOKÓW ZMATERIALIZOWANYCH W OPTYMALIZACJI OPERACJI W BAZACH DANYCH

Streszczenie. Artykuł omawia operacje automatycznego przepisywania zapytań i ich zastosowanie w optymalizacji realizacji poleceń w bazach danych. Rozwiązania tego typu pozwalają znacząco skrócić czas wykonywania skomplikowanych zapytań w złożonych systemach, hurtowniach danych oraz środowiskach analitycznych. Artykuł porusza również kwestie dodatkowych mechanizmów, pomocnych w procesie dostrajania przepisywania zapytań, w tym m.in. analizy widoków i zapytań oraz statystyk. Wykonano eksperymenty w wybranych środowiskach bazodanowych, a ich wyniki zaprezentowano w artykule.

Słowa kluczowe: zapytanie, przepisywanie zapytań, widok zmaterializowany, optymalizacja.

AUTOMATIC QUERY REWRITE USING MATERIALIZED VIEWS IN OPERATIONS OPTIMIZATION IN DATABASES

Summary. The article discusses the operation of automatic query rewriting and its application in the optimization of the commands in the database. Solutions of this type allow you to significantly shorten the execution time of complex queries in complex systems, data warehouses and analytical environments. The article discusses the issues of additional mechanisms helpful in assisting the tuning process of rewriting queries, including analysis of views and queries, as well as statistics. Experiments were performed in selected database environments and their results are presented in the article.

Keywords: query, rewriting queries, materialized views, optimization.

1. Wstęp

Mechanizm przepisywania zapytań, stosowany w dużych bazach danych lub hurtowniach danych, daje ogromne możliwości. Głównym elementem całego procesu są widoki zmaterializowane (ang. *materialized views*), zwane również w przeszłości migawkami (ang. *snapshots*). Odpowiednio skonstruowane widoki zmaterializowane zawierają połączenia tabel oraz przeliczone funkcje agregujące. Proces przepisywania pozwala na bardzo szybką realizację złożonych zapytań SQL za pomocą widoków zmaterializowanych. Przepisywanie polega na zamianie zapytania SQL na wyrażenie mające dostęp do gotowych widoków zmaterializowanych, utworzonych z tabel szczegółowych.

W artykule omówiono mechanizm przepisywania zapytań na przykładzie systemu Oracle. Zaprezentowano przykłady obrazujące w praktyce zasady działania tego rozwiązania. W rozdziale 5. przedstawiono również funkcjonowanie tego rozwiązania w systemie Microsoft SQL Server.

2. Widoki zmaterializowane

W procesie przepisywania zapytań istotną rolę odgrywać mogą tzw. widoki zmaterializowane (ang. *materialized views*) [1]. Są one stosowane między innymi do zapisywania kopii danych, agregacji danych, wykonania połączeń pomiędzy tabelami, synchronizacji pomiędzy bazami danych. Innymi słowy, dane widoczne w widoku zmaterializowanym stanowią zapisany wynik zapytania (przeciwnie do widoków zwykłych, nieposiadających własnych segmentów danych). Zbiór danych jest utrwalany w pamięci, dzięki czemu raz utworzony widok może być wielokrotnie wykorzystywany. Z punktu widzenia bazy danych widoki traktowane są jak normalne tabele. Można je m.in. partycjonować i kompresować.

W omawianym zastosowaniu widoki wykorzystane są w celu optymalizacji procesu realizacji zapytań SQL. Widoki zmaterializowane usprawniają proces dynamicznego przepisania zapytania, co umożliwia zwiększenie wydajności i szybkości zapytań do bazy danych [12]. Głównymi obszarami wykorzystania są: wykonywanie podsumowań, łączenie tabel, wykonywanie obszernych kalkulacji, raportowanie itp. Tworzenie takich kopii może poprawić wydajność zapytań, w szczególności wtedy, gdy dane nie zmieniają się zbyt często. Kiedy bazowe tabele są modyfikowane przez operacje DML, widok zmaterializowany może zawierać nieaktualne dane [15]. Zmaterializowany widok zostanie oznaczony jako przestarzały, gdy tabele bazowe, na których jest oparty, zmieniły się [13]. Dane zawarte w widoku mogą być następnie odświeżane. Istnieją trzy tryby odświeżania widoków zmaterializowanych [3]:

a) fast – dostępny tylko wtedy, gdy system Oracle jest w stanie dopasować wiersze w zmaterializowanym widoku bezpośrednio do wierszy w tabelach bazowych. W odświeżaniu wykorzystywane są specjalne tabele, zwane dziennikami (ang. *materialized view log*);
b) complete – polega na powtórnym wykonaniu zapytania w celu ponownego wypełnienia widoku danymi;
c) force – oznacza odświeżanie w trybie fast, jeżeli jest ono możliwe. Jeżeli nie, to ustawione zostaje odświeżenie typu complete.

Aby dowiedzieć się, jaki rodzaj odświeżenia jest dostępny dla konkretnego widoku, należy wykorzystać procedurę `EXPLAIN_MVIEW`, dostępną w pakiecie `DBMS_MVIEW` [4]. W efekcie wypełniona zostanie tabela `MV_CAPABILITIES_TABLE`, której kolumna `CAPABILITY_NAME` określa, jaki typ przepisywania zapytań może obsłużyć dany widok oraz jaki rodzaj odświeżenia jest możliwy [3], np.:

```
CAPABILITY_NAME                    MSGTXT
------------------------------ ---------------------------------------
PCT_TABLE                      relacja nie jest tabelą podzieloną na
                               partycje
PCT_TABLE_REWRITE              relacja nie jest tabelą podzieloną na
                               partycje
REFRESH_FAST_AFTER_INSERT      tabela podrzędna nie ma dziennika
                               zmaterializowanej perspektywy
…
```

Z powyższego fragmentu raportu wynika, że nie jest możliwe przepisanie wykorzystujące mechanizm partycjonowania. Nie jest także możliwe odświeżanie w trybie fast, ponieważ nie istnieje dziennik widoku. Takie informacje pomagają lepiej projektować widoki, aby te mogły być jak najlepiej wykorzystywane przez mechanizm przepisywania zapytań.

Proces odświeżania widoków może przebiegać w sposób zautomatyzowany (cyklicznie) lub też być wymuszany ręcznie. Za automatyzację procesu odpowiada klauzula *start with* np.:

```
start with SysDate next SysDate+7
```

Natomiast wymuszenie tego procesu jest możliwe przez procedurę REFRESH, np.:

```
BEGIN
DBMS_MVIEW.REFRESH('AGREGAT_2_MV','?');
END;
```

Powyższy przykład odświeża widok o nazwie `AGREGAT_2_MV` w trybie force. Inne tryby to: f (fast), c (complete), p (fast z wykorzystaniem PCT – śledzenie modyfikacji partycji) [4]. Śledzenie modyfikacji partycji umożliwia systemowi Oracle, podczas wykonywania odświeżania, ponowne przeliczanie w zmaterializowanym widoku tylko tych wierszy, które dotyczą zmodyfikowanych partycji w tabelach podstawowych.

W celu wykorzystania przepisywania zapytań widok zmaterializowany musi w swojej definicji posiadać klauzule ENABLE QUERY REWRITE.

3. Query Rewrite

3.1. Warunki funkcjonowania

Query Rewrite funkcjonuje dla zapytań i podzapytań w następujących wyrażeniach języka SQL:

- `SELECT …`
- `CREATE TABLE … AS SELECT …`
- `INSERT INTO … SELECT …`

Aby jednak zapytanie w języku SQL mogło zostać przepisane, musi przejść cykl kontroli. Jeżeli nie przejdzie pomyślnie wszystkich etapów kontroli, wówczas wykonywane jest bezpośrednio na tabelach szczegółowych, a nie z widoków zmaterializowanych. Obie sytuacje zasadniczo różnią się kosztem realizacji.

Optymalizator używa dwóch różnych metod w celu rozpoznania, czy dane zapytanie SQL nadaje się do „przepisania" pod względem warunków zmaterializowanego widoku. Pierwsza metoda opiera się na dopasowaniu tekstu zapytania SQL do tekstu widoku zmaterializowanego. Jeżeli pierwsza metoda się nie powiedzie, optymalizator używa drugiej, polegającej na porównaniu pomiędzy zapytaniem SQL a widokiem: połączeń tabel źródłowych, funkcji agregujących, kolumn danych, grupowania kolumn. System Oracle zastosuje przepisywanie zapytań w momencie, kiedy spełnione zostaną warunki:

- widok zmaterializowany musi być dostępny dla zapytania przepisywanego;
- poziom integralności przepisania powinien udostępniać użycie widoku zmaterializowanego. Na przykład, jeżeli widok zmaterializowany jest nieaktualny (tabele, z których korzysta widok mają zmienione dane) i integralność zapytania jest ustawiona na ENFORCED (wykorzystuje te widoki, które są aktualne), wówczas widok nie jest w użyciu (szczegóły opisane są w następnym podrozdziale);
- całość lub część wyników wymaganych przez zapytania musi być uzyskana z wyników jednego lub wielu widoków zmaterializowanych;
- musi być włączony mechanizm przepisywania zapytań dla każdego widoku zmaterializowanego.

Przepisywanie zapytań można podzielić na dwa etapy. Pierwszy to etap kwalifikacji (ang. *Eligibility Phase*) – następuje ustalenie grupy widoków zmaterializowanych, które mogą być użyte do przepisania zapytania. Drugi to etap przepisania zapytania (ang. *Rewrite Phase*) – zapytanie zostaje przepisane za pomocą widoków zmaterializowanych [16].

Aby umożliwić przepisywanie zapytań, należy ustawić parametry inicjalizacyjne [3]:

- QUERY_REWRITE_ENABLED – uaktywnienie procesu przepisywania zapytania:
  - TRUE zezwoli na przepisywanie;

  - FORCE wymusi przepisywanie (nawet, gdy szacowany koszt przy pominięciu przepisywania zapytania jest niższy).
- QUERY_REWRITE_INTEGRITY – manipulacje poziomem spójności:
  - ENFORCED to najwyższy poziom spójności. Wykorzystuje te widoki, które są aktualne (ustawiony domyślnie);
  - TRUSTED zakłada, że dane są aktualne oraz relacje między tabelami i w obiektach DIMENSION (obiekty bazy danych, wspierające organizację danych wymiarowych, zawartych w tabelach wymiarów – używane do budowy hurtowni danych) są aktualne;
  - STALE_TOLERATED to najniższy poziom. Dane zostaną przepisane nawet, gdy są nieaktualne.
- OPTIMIZER_MODE – ustawienie trybu pracy optymalizatora (na optymalizator kosztowy).

3.2. Weryfikacje

Aby zapytanie mogło zostać zakwalifikowana do przepisania, musi przejść pomyślnie proces weryfikacji. Istnieją następujące kontrole [3]:

- Join Compatibilty Check – w tej próbie złączenia z zapytania SQL są porównywane ze złączeniami używanymi w zmaterializowanym widoku. Wyniki porównania są klasyfikowane w trzech kategoriach:
  - Common join, które pojawiły się w pytaniu i w widoku. Złączenia pomiędzy dwoma parami muszą być takie same lub też to istniejące w zapytaniu musi się wywodzić z połączenia z widoku (np. jeżeli istnieje outer join tabeli A z tabelą B, a zmaterializowany widok posiada inner join na tabelach A oraz B, to wynik z inner join można otrzymać przez filtrowanie wierszy).
  - Delta join, które występują w zapytaniu, ale nie w zmaterializowanym widoku.
  - Delta join, które występuje w widoku, ale nie w zapytaniu.
- Data Sufficiency Check – w tym teście optymalizator określa, czy niezbędne kolumny danych wymagane przez pytanie można uzyskać z widoku zmaterializowanego.
- Grouping Compatibilty Check – sprawdzanie kompatybilności przy grupowaniu danych. To sprawdzanie jest wykorzystywane tylko wtedy, jeżeli zapytanie i widok zmaterializowany zawierają klauzule GROUP BY. Optymalizator określa, czy grupowanie danych w zapytaniu jest dokładnie takie samo, jak dane przegrupowane i przechowywane w zmaterializowanym widoku. Innymi słowy, poziom grupowania powinien być taki sam zarówno w zapytaniu, jak i w widoku zmaterializowanym.
- Aggregate Computability Check – to sprawdzenie jest wymagane, jeżeli zapytanie i widok zmaterializowany posiadają agregaty. Optymalizator określa, czy agregaty w zapytaniu są policzone z jednego lub z większej liczby agregatów przechowanych w zmaterializowa-

nym widoku. Na przykład, jeżeli zapytanie posiada agregat AVG(X) i widok zmaterializowany zawiera SUM(X), COUNT(X), wtedy AVG(X) może być wyliczone na podstawie operacji: SUM(X)/COUNT(X).

- Query Rewrite oraz obiekty Dimension – w hurtowniach danych wykorzystuje się dodatkowe obiekty bazy danych: wymiar (ang. *dimension*), hierarchia wymiaru (ang. *dimension hierarchy*), zależności funkcyjne (ang. *functional dependencies*) [2]. Definiowanie wymiaru zwiększa możliwości Query Rewrite, ponieważ pomaga tworzyć funkcjonalne zależności pomiędzy kolumnami. Ponadto, wymiar może wyrażać relacje wewnątrz tabeli, które nie mogą być wyrażone przez Constraint [3].

3.3. Rodzaje Query Rewrite

Wyróżnia się kilka typów przepisywania zapytań.

3.3.1. Text Match Rewrite

Mechanizm Query Rewrite zawsze najpierw porównuje tekst z przychodzących zapytań z tekstem potencjalnego widoku zmaterializowanego w celu przepisania zapytania. Istnieją dwie metody dopasowania tekstu, full text match rewrite oraz partial text match rewrite. W pierwszej metodzie cały tekst komendy jest porównany z całym tekstem widoku (wyrażenie SELECT), ignorując białe znaki. W drugiej metodzie porównujemy tekst od klauzuli FROM w zapytaniu, z tekstem z widoku zaczynającym się od klauzuli FROM. Jeżeli żadna z metod porównania się nie powiedzie, wtedy optymalizator używa metody General Query Rewrite (metody typu General Query Rewrite to wszystkie opisane w kolejnych punktach).

3.3.2. Join back

Jeżeli niektóre kolumny danych wymaganych przez zapytanie nie są możliwe do uzyskania przez widok zmaterializowany, optymalizator dalej określa, czy można uzyskać te dane, bazując na relacjach danych zwanych functional dependency. Gdy kolumna danych wymagana przez zapytanie nie jest dostępna z widoku, dane z tej kolumny można ciągle uzyskać przez złączenie z tabelą, która posiada kolumnę z danymi pod warunkiem, że widok zawiera klucz, który funkcjonalnie określi wymaganą kolumnę danych. Możemy zadeklarować zależność funkcjonalną na dwa sposoby:

- używając ograniczenia PRIMARY_KEY,
- używając klauzuli DETERMINES przy obiektach dimension.

3.3.3. Aggregate Computability

Przepisywanie zapytań może również wystąpić, gdy optymalizator określa, czy agregat w zapytaniu może być obliczony na podstawie innych agregatów przechowywanych w widoku zmaterializowanym. Jeżeli zapytanie zawiera AVG(X), a zmaterializowany widok zawiera SUM(X) oraz COUNT(X), to AVG(X) może być obliczone jako SUM(X)/COUNT(X). Aby tego dokonać, Oracle konwertuje wyrażenie agregatu zawierającego argument w postać kanoniczną tak, że dwa różne wyrażenia są przekształcone w taką samą formę kanoniczną. Na przykład: A*(B-C), A*B-C*A,(B-C)*A oraz –A*C+A*B są konwertowane w tę samą kanoniczną formę, a zatem są dobrze dopasowane.

3.3.4. Aggregate Rollup

Kiedy zapytanie żąda agregatów takich jak SUM(sprzedaż) na wyższym poziomie w hierarchii niż poziom, na którym agregaty są w zmaterializowanym widoku, wtedy zapytanie może być zapisane za pomocą zmaterializowanego widoku z wykorzystaniem zwinięcia jego składników (agregatów) do pożądanego poziomu. Na przykład SUM(sprzedaż) na poziomie miasta może być zwinięte do SUM(sprzedaż) na poziomie województw przez zsumowanie wszystkich agregatów SUM(sprzedaż) w grupie o tym samym poziomie województw.

3.3.5. Gdy zmaterializowane widoki obejmują tylko podzbiór danych

Oracle wspiera przepisywanie zapytań w taki sposób, że możliwe jest wykorzystywanie widoków zmaterializowanych, w których klauzule HAVING oraz WHERE zawężają zbiór danych. Do wykonania tego typu zapytań Oracle musi określić, czy wymagane dane w zapytaniu są zawarte w podzbiorze danych przechowywanych w zmaterializowanym widoku. Aby sprawdzić, czy przepisywanie zapytania może zadziałać na filtrowanych danych, sprawdzenie odbywa się zarówno na zapytaniu, jak i na widoku (jest wykonywane na klauzulach WHERE i HAVING). Jeżeli widok zawiera podzbiór danych, a rozważane zapytanie nie, wtedy zgodność selekcji sprawdza różnice, ponieważ widok jest bardziej restrykcyjny niż zapytanie.

Klauzule WHERE i HAVING widoku zmaterializowanego mogą zawierać połączenia lub polecenie SELECT lub obydwa na raz i widok wciąż może być użyty do przepisania pytania.

Na przykład:

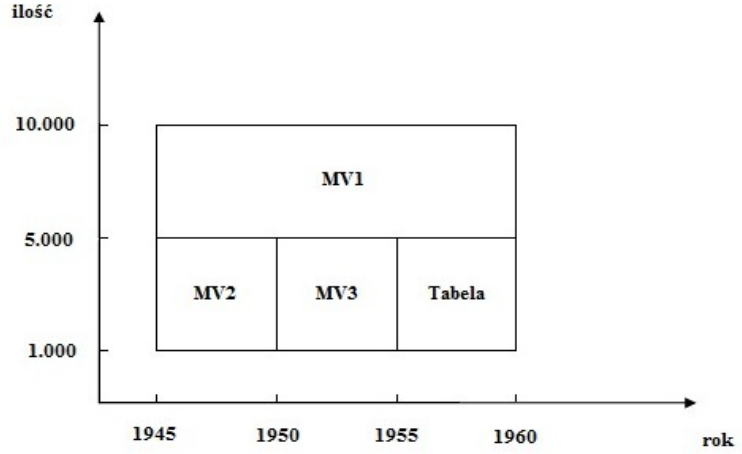
1. Zapytanie zawiera klauzule `WHERE kli_id=157`, a widok zawiera `WHERE kli_id BETWEEN 0 AND 300`, wtedy lewe strony są równe, a prawa strona zapytania mieści się w ramach widoku (w zakresie 0 – 300).
2. Zapytanie zawiera klauzulę `WHERE (sales.amount_sold * 0.07) BETWEEN 1000 AND 2000`, natomiast widok posiada klauzulę `WHERE (sales.amount_sold * 0.07) BETWEEN 0 AND 10000`. Lewe strony są równe, a prawa strona zapytania mieści się w zakresie widoku. Przepisanie jest możliwe.

3.3.6. Partition Change Tracking (PCT) Rewrite

W sytuacjach gdy mamy do czynienia z tabelami partycjonowanymi, może zostać wykorzystane rozwiązanie PCT. PCT Rewrite umożliwia optymalizatorowi przepisanie zapytań przy użyciu aktualnych danych, używając widoku zmaterializowanego, którego dane są tylko częściowo aktualne. Aby tego dokonać, Oracle śledzi, które partycje w tabelach szczegółowych zostały zaktualizowane. Optymalizator jest w stanie korzystać z tych części (partycji), które są aktualne. Optymalizator wykorzysta tę metodę, jeżeli mamy ustawiony poziom integralności na ENFORCED lub TRUSTED. Nie mamy możliwości użycia PCT w przypadku wykorzystania trybu STALE_TOLERATED, ponieważ świeże dane nie są wówczas rozpatrywane. Obsługiwane metody partycjonowania to: list, range oraz range-list. Partycjonowanie hash nie jest obsługiwane.

3.3.7. Multiple Materialized View

Query Rewrite posiada możliwość wykorzystywania wielu widoków zmaterializowanych. Zapytanie może zostać przepisane przy wykorzystaniu kilku widoków.



Rys. 1. Schemat bazy danych
Fig. 1. Database scheme

Dla poniższego zapytania, przy założeniu, że widoki zmaterializowane (rys. 1) obejmują tylko pewne fragmenty poszukiwanych informacji, może zostać zrealizowane połączenie widoków (i ewentualnie tabel, jeżeli jest taka potrzeba):

```
SELECT K.IMIE, K.NAZWISKO
FROM KLIENT K, SPRZEDAZ SP, FAKTURA_SPRZEDAZY FS
WHERE K.ID_KLIENT = SP.ID_KLIENT AND SP.ID_FAKTURA_SP = FS.ID_FAKTURA_SP AND
DATA_WYSTAWIENIA BETWEEN 1945 AND 1960 AND ILOSC BETWEEN 1000 AND 10000;
```

4. Przykłady wykorzystania

4.1. Przykład 1

Zakładamy, że istnieje widok zmaterializowany AGREGAT_2_MV, który posiada dane aktualne, a jego definicja wygląda następująco:

```
CREATE MATERIALIZED VIEW AGREGAT_2_MV
TABLESPACE SKLEP
BUILD IMMEDIATE
REFRESH COMPLETE
ENABLE QUERY REWRITE AS
SELECT KL.WOJEWODZTWO,  SUM(FS.ILOSC *T.CENA) as CENA, COUNT(FS.ILOSC *T.CENA)
as ILOSC_SPRZEDANYCH
FROM SPRZEDAZ SP, FAKTURA_SPRZEDAZY FS, TOWAR T, KLIENT KL
WHERE
FS.ID_FAKTURA_SP = SP.ID_FAKTURA_SP AND
T.ID_TOWAR = SP.ID_TOWAR AND
KL.ID_KLIENT = SP.ID_SPRZEDAZ
GROUP BY KL.WOJEWODZTWO;
```

Zapytanie jest następujące:

```
SELECT /*+ NOREWRITE */ KL.WOJEWODZTWO,
ROUND(AVG(FS.ILOSC *T.CENA),3) as SREDNIA_CENA_ZAKUPU
FROM SPRZEDAZ SP, FAKTURA_SPRZEDAZY FS, TOWAR T, KLIENT KL
WHERE
FS.ID_FAKTURA_SP = SP.ID_FAKTURA_SP AND
T.ID_TOWAR = SP.ID_TOWAR AND
KL.ID_KLIENT = SP.ID_SPRZEDAZ
GROUP BY KL.WOJEWODZTWO;
```

Wykonując zapytanie, za pierwszym razem bez wykorzystania mechanizmu przepisywania zapytań (wskazówka optymalizatora norewrite), a za drugim razem wykorzystując je (usuwając wskazówkę), otrzymujemy dwa plany wykonania:

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		9	432	2705 (2)	00:00:33
1	HASH GROUP BY		9	432	2705 (2)	00:00:33
* 2	HASH JOIN		200K	9375K	2697 (1)	00:00:33
3	TABLE ACCESS FULL	TOWAR	642	5778	3 (0)	00:00:01
* 4	HASH JOIN		200K	7617K	2692 (1)	00:00:33
5	TABLE ACCESS FULL	FAKTURA_SPRZEDAZY	349K	2734K	345 (2)	00:00:05
* 6	HASH JOIN		200K	6054K	1604 (1)	00:00:20
7	TABLE ACCESS FULL	KLIENT	200K	3320K	548 (1)	00:00:07
8	TABLE ACCESS FULL	SPRZEDAZ	349K	4785K	345 (2)	00:00:05

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		9	342	3 (0)	00:00:01
1	MAT_VIEW REWRITE ACCESS FULL	AGREGAT_2_MV	9	342	3 (0)	00:00:01

Jak można zauważyć, zapytanie zostało przepisane za pomocą zmaterializowanego widoku AGREGAT_2_MV. Średnia cena zakupu została obliczona za pomocą dwóch kolumn z widoku: CENA oraz ILOSC_SPRZEDANYCH. Wyeliminowany zostaje w ten sposób nadmiar kosztów związany z gromadzeniem i czerpaniem danych przy każdorazowym wykonywaniu zapytania [14].

Wykorzystując Explain Rewrite, otrzymujemy następujący raport, który potwierdza korzyści z zastosowanie Query Rewrite:

```
----------------------- ANALYSIS OF QUERY REWRITE ----------------------
>>
>> QRY BLK #: 0
>> MESSAGE  : QSM-01209: zapytanie ponownie zapisane przy użyciu zmaterializowa-
nej perspektywy, AGREGAT_2_MV, z wykorzystaniem algorytmu uzgadniania tekstu
>> RW QUERY : SELECT AGREGAT_2_MV.WOJEWODZTWO
WOJEWODZTWO,DECODE(AGREGAT_2_MV.CENA,0,0,NULL,NULL,AGREGAT_2_MV.CENA/AGREGAT_2_M
V.ILOSC_SPRZEDANYC) SREDNIA_CENA_ZAKUPU FROM ADM.AGREGAT_2_MV AGREGAT_2_MV
>> ORIG COST: 2705,43716216747                  RW COST: 2,70134510157229
```

Jeżeli do tabeli wprowadzone zostaną nowe rekordy, zapytanie nie zostanie przepisane. Wynik z Explain Rewrite:

```
----------------------- ANALYSIS OF QUERY REWRITE -----------
>>
>> MESSAGE  : QSM-01031: zmaterializowana perspektywa, AGREGAT_2_MV, jest nieak-
tualna z trybem spójności TRUSTED
>> ORIG COST: 0                   RW COST: 0
>> MESSAGE OUTPUT BEFORE VIEW MERGING...
```

Możliwe jest przeciwdziałanie wykonywaniu zapytań bez uwzględnienia widoków zmaterializowanych i mechanizmu przepisywania zapytań. Jest to szczególnie istotna kwestia w przypadku zapytań o długim czasie realizacji. Możliwość taką oferuje wskazówka optymalizatora (hint) REWRITE_OR_ERROR, której wykorzystanie, w przypadku nieuwzględnienia QR, generuje błąd: „ORA-30393: blok zapytania w instrukcji nie został ponownie zapisany” [17].

4.2. Przykład 2

System Oracle umożliwia realizację procesu przepisywania zapytań z wykorzystaniem mechanizmu partycjonowania [18]. Wykorzystując mechanizm Partition Change Tracking (opisany w rozdziale 3.3.6), możliwe jest zarówno odświeżanie tylko tych części danych w widoku zmaterializowanych, które dotyczą zaktualizowanej partycji, jak również przepisywanie zapytań z wykorzystaniem tabel, których dane uległy zmianie, pod warunkiem, że zapytanie dotyczy danych z innych (nieobjętych modyfikacjami) partycji.

Poniższy przykład zakłada istnienie tabel produkt oraz sprzedaz, z których druga jest tabelą partycjonowaną zakresowo, posiadającą m.in. partycje: sprzedaz_2010_12 i sprzedaz_2011_01. Pomimo wstawiania kolejnych danych w miesiącu styczniu 2011, nie ma problemu z przepisaniem zapytania i wykorzystaniem widoku zmaterializowanego w przypadku poniższego zapytania, ze względu na to, iż niezbędna partycja nie uległa zmianie.

```
SELECT ps.czas_klucz, p.kategoria,
SUM(ps.sprzedaz_cena) as suma_sprzedazy
FROM produkt p, sprzedaz ps
WHERE ps.produkt_id = p.produkt_id and
ps.czas_klucz BETWEEN TO_DATE('01-12-2010', 'DD-MM-YYYY')
AND TO_DATE('31-12-2010', 'DD-MM-YYYY')
GROUP BY ps.czas_klucz, p.kategoria;
```

Plan wykonania:

Id	Operation	Name	Cost
0	SELECT STATEMENT		3
1	MAT_VIEW REWRITE ACCESS FULL	SUMA_KATEGORIA_SPRZEDAZ_MV	3

4.3. Przykład 3

Kolejną użyteczną właściwością przepisywania zapytań jest możliwość jego wykorzystania do odświeżania widoków zmaterializowanych [18]. Oracle jest w stanie wykorzystywać dane składowane w jednym widoku zmaterializowanym w celu odświeżania innego widoku.

W przypadku gdy istnieje widok, który jest odświeżany codziennie, oraz widok odświeżany raz w miesiącu, istnieje możliwość odświeżania widoku miesięcznego za pomocą widoku dziennego. Tylko widoki świeże mogą zostać użyte w takiej operacji.

Aby przepisanie mogło funkcjonować, konieczny jest tryb integralności ENFORCED. W przypadku wykorzystania przepisania z trybem TRUSTED, niezbędne jest umieszczenie specjalnej klauzuli w widoku zmaterializowanym: USING TRUSTED CONSTRAINTS, jak w poniższym przykładzie.

```
CREATE MATERIALIZED VIEW sprzedaz_kategoria_produkt_mv
REFRESH FORCE
USING TRUSTED CONSTRAINTS
ENABLE QUERY REWRITE
AS
SELECT ps.czas_klucz, p.kategoria, SUM(ps.cena_sp) as
suma¬_sprz
FROM produkt p, sprzedaz PS
WHERE ps.produkt_id = p.produkt_id
GROUP BY  ps.czas_klucz, p.kategoria
```

5. Przepisywanie zapytań w Microsoft SQL Server

5.1. Zasady funkcjonowania

W systemie Microsoft SQL Server również istnieje rozwiązanie o podobnej funkcjonalności jak w środowisku Oracle. Możliwe jest wykorzystanie fizycznie zapisanych widoków w celu przepisania zapytań. Możliwości te zostały udostępnione od wersji SQL Server 2000 [8]. W SQL Server, w celu przepisywanie zapytań, wykorzystane są widoki indeksowane. Zwykły widok nie posiada swojej reprezentacji fizycznej. Jeżeli utworzymy unikalny, sklaste-

ryzowany indeks na widoku, następuje jego zapis. Z punktu widzenia omawianego rozwiązania, interesujące są indeksy typu:

- Clustered – dane tabeli są fizycznie posortowane i zapisane według klucza tabeli wskazanego do indeksowania.
- Unique – klucz indeksu zapewnia unikatowość danych.

Widok może zostać użyty w zapytaniu na dwa sposoby. Zapytanie może odwoływać się do widoku bezpośrednio lub przez optymalizator, który może wybrać cały bądź część widoku w celu przepisania zapytania, co powinno znacznie zmniejszyć czas wykonania widoku [8].

W wersjach SQL Server Developer oraz Enterprise można używać przepisywania zapytań bez odnoszenia się do widoków przez nazwy (odbywa się to automatycznie). W pozostałych wersjach konieczne jest użycie parametru NOEXPAND i odniesienie się do widoku przez nazwę. Parametr powoduje, że widok traktowany jest jak zwykła tabela i jest zapewnione skorzystanie z widoku, a nie z tabel, z których widok został utworzony [8], np.:

```
SELECT Kol1, Kol2, ... FROM Tabela1, Widok1 WITH (NOEXPAND) WHERE ...
```

Na wyłączenie trybu przepisywania pozwala klauzula EXPAND VIEWS:

```
SELECT Kol1, Kol2, ... FROM Tabela1, Widok1 WHERE ... OPTION (EXPAND VIEWS)
```

Optymalizator wykorzystuje kilka warunków, aby określić, czy widok indeksowany może objąć całe zapytanie lub tylko część. Optymalizator znajduje dopasowania między kolumnami indeksu widoku i elementami w zapytaniu, takie jak:

- warunki wyszukiwania w klauzula WHERE,
- funkcje agregujące,
- klauzule GROUP BY,
- odwołania do tabel.

W celu utworzenia widoku musi zostać spełnionych kilka warunków:

- widok oraz tabele muszą znajdować się w tej samej bazie danych i muszą mieć tego samego właściciela;
- widok indeksowany nie musi zawierać wszystkich tabel, które występują w głównym zapytaniu; może zawierać ich część;
- unikalny, klastrowy indeks musi zostać utworzony zanim zostaną utworzone inne indeksy na widoku;
- następujące opcje muszą zostać włączone: ANSI_NULLS (określa zachowanie zgodne z ISO równości (=) i nierówności (<>) podczas porównania operatorów, gdy są używane z wartościami null), ANSI_PADDING (kontroluje sposób przechowywania wartości kolumn krótszych niż zdefiniowany rozmiar kolumny i sposób przechowywania wartości mających spacje na końcu w typach char, varchar, binary, varbinary), ANSI_WARNINGS

(określa zachowanie standardu ISO dla kilku rodzajów błędu), ARITHABORT (warunek, który powoduje zakończenie wykonania zapytania podczas błędu przy dzieleniu przez zero lub przepełnieniu programu), CONCAT_NULL_YIELDS_NULL (kontroluje, czy wyniki łączeń są traktowane jako wartości null czy jako ciąg pusty), QUOTED_IDENTIFIER (powoduje, że SQL Server wykonuje reguły ISO dotyczące identyfikatorów ograniczników i literałów);

- widok musi zostać utworzony z opcją SCHEMABINDING, dlatego należy stosować nazwę dwuczęściową, np. `dbo.SPRZEDAZ`. Opcja SCHEMABINDING wiąże schemat widoku ze wszystkimi obiektami bazowymi, użytymi do zdefiniowania danego widoku; dzięki temu usunięcie obiektów bazowych będzie niemożliwe;
- jeżeli zapytanie dokonuje agregacji danych, widok musi zawierać funkcję COUNT_BIG(*).

Widoki te są automatycznie odświeżane. W momencie zmian na tabelach bazowych indeksowany widok zostaje zmieniony automatycznie. Należy dobrze zaplanować, kiedy stosować tego typu widoki, aby nie ponosić dodatkowych, znaczących kosztów.

5.2. Przykład 1

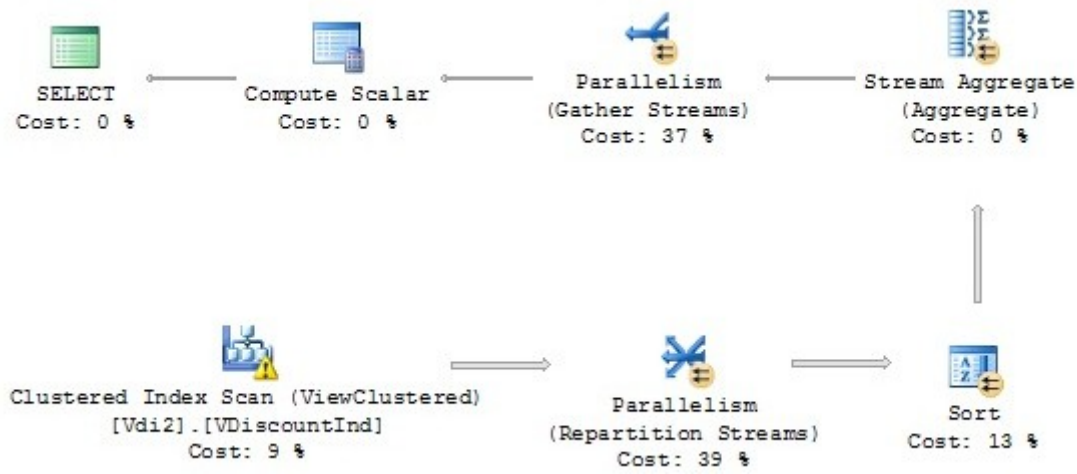
Polecenia tworzące widok oraz indeks:

```
CREATE VIEW Vdi2 WITH SCHEMABINDING AS
SELECT T.ID_TOWAR, TY.PODTYP ,
T.NAZWA ,sum(FS.ILOSC) as ILOSC_SPRZEDANYCH_EGZ ,
sum(FZ.ILOSC) as ILOSC_KUPIONYCH_EGZ, COUNT_BIG(*) as liczba
FROM dbo.SPRZEDAZ as SP, dbo.FAKTURA_SPRZEDAZY as FS, dbo.TOWAR as T,
dbo.TYP_PRODUKTU as TY, dbo.KUPNO as KUP, dbo.FAKTURA_ZAKUPU as FZ
WHERE
FS.ID_FAKTURA_SP = SP.ID_FAKTURA_SP AND
T.ID_TOWAR = SP.ID_TOWAR AND T.ID_TYP = TY.ID_TYP AND
T.ID_TOWAR = KUP.ID_TOWAR AND KUP.ID_FAKTURA_ZA = FZ.ID_FAKTURA_ZA
GROUP BY  T.ID_TOWAR,T.NAZWA ,TY.PODTYP
GO
CREATE UNIQUE CLUSTERED INDEX VDiscountInd ON Vdi2 (ID_TOWAR)
```

Przykładowe zapytanie:

```
SELECT  TY.PODTYP ,sum(FS.ILOSC) as ILOSC_SPRZEDANYCH_EGZ,
sum(FZ.ILOSC) as ILOSC_KUPIONYCH_EGZ,
sum(FZ.ILOSC) - sum(FS.ILOSC) as ILE_ZOSTALO ,
(sum(FS.ILOSC) * 100) /sum(FZ.ILOSC)   as PROCENT_SPRZEDANYCH
FROM SPRZEDAZ SP, FAKTURA_SPRZEDAZY FS, TOWAR T, TYP_PRODUKTU TY, KUPNO KUP,
FAKTURA_ZAKUPU FZ
WHERE
FS.ID_FAKTURA_SP = SP.ID_FAKTURA_SP AND
T.ID_TOWAR = SP.ID_TOWAR AND
T.ID_TYP = TY.ID_TYP AND
T.ID_TOWAR = KUP.ID_TOWAR AND
KUP.ID_FAKTURA_ZA = FZ.ID_FAKTURA_ZA
GROUP BY TY.PODTYP
ORDER BY TY.PODTYP;
```

Plan wykonania zapytania z użyciem widoku indeksowanego przedstawiono na rys. 2.



Rys. 2. Plan wykonania
Fig. 2. Execution plan

Jak widać na powyższym planie, zapytanie zostało przepisanie z wykorzystaniem widoku indeksowanego.

5.3. Przykład 2

Polecenia tworzące widok oraz indeks:

```
CREATE VIEW Vdi2 WITH SCHEMABINDING AS
SELECT T.ID_TOWAR, TY.PODTYP ,
T.NAZWA ,sum(FS.ILOSC) as ILOSC_SPRZEDANYCH_EGZ ,
sum(FZ.ILOSC) as ILOSC_KUPIONYCH_EGZ, COUNT_BIG(*) as liczba
FROM dbo.SPRZEDAZ as SP, dbo.FAKTURA_SPRZEDAZY as FS, dbo.TOWAR as T,
dbo.TYP_PRODUKTU as TY, dbo.KUPNO as KUP, dbo.FAKTURA_ZAKUPU as FZ
WHERE
FS.ID_FAKTURA_SP = SP.ID_FAKTURA_SP AND
T.ID_TOWAR = SP.ID_TOWAR AND T.ID_TYP = TY.ID_TYP AND
T.ID_TOWAR = KUP.ID_TOWAR AND KUP.ID_FAKTURA_ZA = FZ.ID_FAKTURA_ZA
GROUP BY  T.ID_TOWAR,T.NAZWA ,TY.PODTYP
GO
CREATE UNIQUE CLUSTERED INDEX VDiscountInd ON Vdi2 (ID_TOWAR)
```

Przykładowe zapytanie:

```
SELECT  TY.PODTYP ,sum(FS.ILOSC) as ILOSC_SPRZEDANYCH_EGZ,
sum(FZ.ILOSC) as ILOSC_KUPIONYCH_EGZ,
sum(FZ.ILOSC) - sum(FS.ILOSC) as ILE_ZOSTALO ,
(sum(FS.ILOSC) * 100) /sum(FZ.ILOSC)   as PROCENT_SPRZEDANYCH
FROM SPRZEDAZ SP, FAKTURA_SPRZEDAZY FS, TOWAR T, TYP_PRODUKTU TY, KUPNO KUP,
FAKTURA_ZAKUPU FZ
WHERE
FS.ID_FAKTURA_SP = SP.ID_FAKTURA_SP AND
T.ID_TOWAR = SP.ID_TOWAR AND
T.ID_TYP = TY.ID_TYP AND
T.ID_TOWAR = KUP.ID_TOWAR AND
KUP.ID_FAKTURA_ZA = FZ.ID_FAKTURA_ZA
GROUP BY TY.PODTYP
ORDER BY TY.PODTYP;
```

W tym wypadku przepisanie miało polegać na skorzystaniu z widoku zmaterializowanego oraz połączeniu widoku z tabelą `TYP_TOWARU` w celu pobrania dodatkowej kolumny

PODTYP. Microsoft SQL Server nie wykonał połączenia. Plan wykonania zapytania, w przypadku gdy widok indeksowany istnieje, oraz w przypadku gdy nie istnieje, jest dokładnie taki sam (składa się z 29 kroków). Zapytanie nie zostało przepisane, a czas jego wykonania przed i po utworzeniu widoku nie uległ zmianie.

Ten sam eksperyment powtórzony w środowisku Oracle dał odmienny efekt. Po utworzeniu widoku zmaterializowanego realizacja zapytania została w bardzo istotny sposób uproszczona, a jej czas skrócony. Potwierdza to poniższy raport:

```
>> MESSAGE  : QSM-01151: zapytanie zostało ponownie zapisane
>> ORIG COST: 1427,45219327295     RW COST: 8,14414803529347
>> MESSAGE  : QSM-01033: zapytanie ponownie zapisane przy użyciu
   zmaterializowanej perspektywy EX3_MV
>> MESSAGE  : QSM-01102: zmaterializowana perspektywa, EX3_MV, wymaga złączenia
   wstecznego do tabeli, TYP_PRODUKTU, dla kolumny, PODTYP
```

6. Podsumowanie

W artykule omówiono rozwiązania umożliwiające przepisywanie zapytań na podstawie widoków zmaterializowanych. Przedstawiono zasady funkcjonowania tego mechanizmu w systemach Oracle oraz Microsoft SQL Server. Większe możliwości konfiguracyjne oraz wydajnościowe oferują serwery firmy Oracle. MS SQL Server wciąż jeszcze nie w pełni wykorzystuje potencjał tkwiący w możliwości przepisywania zapytań. W ramach prac wykonano wiele testów. Ze względu na ograniczenia licencyjne, precyzyjne porównania wydajności rozwiązania Query Rewrite w systemach Oracle i Microsoft nie mogły zostać zaprezentowane w artykule. Pewne jest, że z uwagi na nieustannie rosnące ilości danych gromadzonych i przetwarzanych w systemach bazodanowych oraz na coraz bardziej złożone wymagania użytkowników, dalszy rozwój tego typu rozwiązań jest bardzo potrzebny. Kolejne wersje systemów z pewnością udostępnią bardziej rozbudowane funkcjonalności w tym zakresie. Istotna jest jak najlepsza znajomość tych rozwiązań, aby w pełni wykorzystać tkwiące w nich możliwości.

BIBLIOGRAFIA

1. Kozielski S., Wrembel R. (eds.): New Trends in Data Warehousing and Data Analysis. Annals of Information Systems, Vol. 3, Springer, 2009.
2. Wrembel R., Koncilia C.: Data Warehouses and OLAP. IRM Press, 2007.
3. Oracle Database Data Warehousing Guide 11g.
4. Puget Sound Oracle Users Group – http:/psoug.org/reference/dbms_mview.html.

5. Loney K.: Oracle Database 11g Kompendium administrator. Helion, Gliwice 2010.
6. Freeman R.G., Nanda A.: Oracle Database 11g Nowe możliwości. Helion, Gliwice 2009.
7. Hanson E.N., Angelov Y.: Statistics Used by the Query Optimizer in Microsoft SQL Server 2008. SQL Server Technical Article, February 2009.
8. Erickson G., Kollar L., Ward J.: Improving Performance with SQL Server 2008 Indexed Views. SQL Server Technical Article, October 2008.
9. Stawiarski K.: Oracle 11g – nowe cechy w strojeniu wydajności i tworzeniu aplikacji dla bazy danych. XIV Konferencja PLOUG, październik 2008.
10. Dageville B., Das D., Dias K., Yagoub K., Zait M., Ziauddin M.: Automatic SQL Tuning in Oracle 10g. Oracle Corp., 2006.
11. Lewis J.: Cost-Based Oracle Fundamentals. Springer-Verlag, New York 2006.
12. Thiyagarajan M., Kumar P.: Inline view query rewrite using a materialized view. 2009.
13. Hobbs L.: Oracle Materialized View & Query Rewrite. An Oracle White Paper, May 2005.
14. Gupta A., Witkowski A.: Rewrite of queries containing rank or row number min/max aggregate functions using a materialized views. San Jose 2006.
15. Antognini C.: Troubleshooting Oracle Performance. Springer-Verlag, New York 2008.
16. Bello R.G., Panchapagesan B., Yu T., Raitto J. D.: Rewriting a query to use a set of materialized views and database object. Reedwood Shores, 2008.
17. Freeman R.G.: Oracle Database 10g New Features. The McGraw-Hill, California 2004.
18. Hobbs L., Smith P., Lawande S.: Oracle Database 10g Data Warehousing. Elsevier, 2004.

Recenzenci: Dr inż. Bożena Małysiak-Mrozek
Dr hab. Tadeusz Pankowski, prof. Uniwersytetu im. Adama Mickiewicza

Wpłynęło do Redakcji 31 stycznia 2011 r.

Abstract

The article discusses the operations of automatic query rewriting and its application in the optimization of the operations in the database. Solutions of this type allow you to shorten significantly the execution time of complicated queries in complex systems, data warehouses and analytical environments. The article discusses the issues of additional mechanisms helpful in assisting the tuning process of rewriting queries, including analysis of views, queries and statistics.

Query rewriting mechanism offers great potential when used in large databases or data warehouses. The main elements of the whole process are materialized views, also known as snapshots in the past. Well designed materialized views contain connection tables and converted aggregate functions. The process of rewriting permits quick execution of SQL queries using materialized views. Rewriting is to change the SQL query expressed by the conditions created on the tables or views into the expression that has an access to the prepared materialized views created from the detailed tables.

The principles of this mechanism in the Oracle and Microsoft SQL Server systems have been presented. Oracle server offers more configurable, as well as performance, possibilities. MS SQL Server is still not fully exploiting the potential of the possibility of rewriting queries. Within the confines of the works a series of tests has been performed. Due to licensing restrictions, precise comparisons of productivity of query rewrite solutions in Oracle and Microsoft systems could not be presented in the article. It is certain that, given the constantly increasing amount of data collected and processed in database systems and much more complex user requirements, the further development of such solutions is needed very much. Subsequent versions of the database systems will make available more advanced functionalities in this regard. It is essential to have the best knowledge of these solutions to fully exploit their possibilities.

Adresy

Krzysztof CZAJKOWSKI: Politechnika Krakowska, Wydział Fizyki, Matematyki i Informatyki, Instytut Teleinformatyki, ul. Warszawska 24, 31-155 Kraków, Polska, kc@pk.edu.pl.
Marek BANACH: Politechnika Krakowska, Wydział Fizyki, Matematyki i Informatyki, Instytut Teleinformatyki, ul. Warszawska 24, 31-155 Kraków, Polska, mbanach4@gmail.com.