

Mechanizmy wspierające tworzenie i użytkowanie Hurtowni Danych w systemie zarządzania bazą danych Oracle

Gerard Głowacki
Akademia Ekonomiczna w Poznaniu
e-mail: gerardg@novci2.ae.poznan.pl

Abstrakt. W artykule omawia się możliwości wykorzystania mechanizmów oferowanych przez bazę danych Oracle wspierających budowę i użytkowanie magazynów danych, a w szczególności : partycjonowanie, indeksy bitmapowe, ograniczenia integralnościowe, perspektywy materialne, wymiary i hierarchie, rozszerzenia języka zapytań SQL wspomagające wykonywanie zapytań o charakterze analitycznym do bazy danych.

1. Systemy magazynów danych, a systemy transakcyjne

Przeznaczeniem systemu magazynu danych jest wykonywanie złożonych analiz , w odróżnieniu od systemów transakcyjnych których celem działania jest przetwarzanie bieżących transakcji. W związku z tym system Hurtowni Danych posiada cechy odróżniające go od systemów transakcyjnych, do których można zaliczyć :

- optymalizacja pod kątem przeprowadzania złożonych zapytań o charakterze analitycznym,
- modyfikacje danych wprowadzane są w pewnych regularnych okresach, podczas gdy dane w systemie transakcyjnym powinny odzwierciedlać aktualny stan występujący w rzeczywistości ,
- projekt bazy danych jest często zdenormalizowany w celu dostosowania go do zadawanych zapytań, podczas gdy system transakcyjny jest w pełni znormalizowany aby zapobiec redundancji informacji ,
- w systemie magazynu danych przechowuje się dane historyczne z przeszłych okresów.

Centralnym punktem schematu magazynu danych są tabele faktów – zawierające informacje o faktach (np. dane o wielkości sprzedaży) oraz posiadające klucze obce do tabel wymiarów. Tabele wymiarów służą do segregowania danych w tabeli faktów. Najczęściej występujące tabele wymiarów to klient, produkt, czas.

Dane z tabel wymiarów posiadają najczęściej swoje hierarchie, czyli logiczną strukturę używaną do grupowania i sortowania np. hierarchie dla wymiaru czasu : ROK – KWARTAŁ – MIESIĄC .

W związku z zarysowanymi różnicami podczas budowy i użytkowania systemu Hurtowni Danych wykorzystuje się pewne specyficzne mechanizmy oferowane przez bazę danych Oracle.

Należą do nich: partycjonowanie, indeksy bitmapowe, ograniczenia integralnościowe, perspektywy materialne, wymiary i hierarchie, jak również rozszerzenia języka zapytań SQL wspomagające wykonywanie zapytań o charakterze analitycznym do bazy danych.

2. Partycjonowanie

Partycjonowanie jest mechanizmem umożliwiającym podzielenie dużych tabel (i indeksów) na mniejsze fragmenty (partycje), dla których można zdefiniować parametry fizyczne (np. umieścić partycje na różnych dyskach i wykorzystać w ten sposób równoległość odczytu), oraz wykonywać operacje DDL (np. usunąć lub dodać partycję). Wykorzystanie tego mechanizmu może ułatwić

strojenie poleceń SQL, pozwala uniknąć przeglądania całych dużych tabel (i indeksów), jak również ułatwia administrację dużymi bazami danych (np. wykonywanie kopii bezpieczeństwa).

Można wyróżnić następujące typy partycjonowania :

- partycjonowanie zakresowe (ang. Range) - dane są dzielone na podstawie określenia granicznych wartości dla kolumn będących kluczem partycjonowania. W systemach Hurtowni Danych najczęściej kluczem partycjonowania jest kolumna przechowująca dane o dacie - (często ze względów efektywnościowych kolumna ta posiada typ numeryczny). Partycjonowanie zakresowe umożliwia łatwą implementację przesuwanego aktualnego okna czasowego dla przechowywanych danych np. w sytuacji gdy chcemy przechowywać dane z ostatnich 12 miesięcy, ułatwia wykonywanie kopii bezpieczeństwa.
- partycjonowanie z użyciem klucza haszowego - podział danych następuje na podstawie algorytmu haszowego, który jest zastosowany do wskazanego przez użytkownika klucza partycjonowania. Algorytm haszowy rozrzuca dane równomiernie po partycjach, wszystkie partycje mają mniej więcej ten sam rozmiar. Rozmieszczenie danych w partycjach nie zależy od reguł biznesowych.
- partycjonowanie złożone - wykorzystuje cechy partycjonowania zakresowego i partycjonowania z wykorzystaniem klucza haszowego. Dane są dzielone na zasadzie partycjonowania zakresowego, a następnie w obrębie tak powstałych partycji dalej dzielone na podpartycje z wykorzystaniem funkcji haszowej.

Partycjonowane mogą być tabele jak i indeksy. W przypadku partycjonowania indeksu - można indeks partycjonować globalnie albo lokalnie. W przypadku indeksu globalnego jedna partycja indeksu zawiera odniesienie do wierszy tabeli znajdujących się w różnych partycjach. W przypadku indeksu lokalnego jedna partycja indeksu zawiera odniesienie tylko do wierszy tabeli znajdujących się w jednej partycji tabeli. Indeks lokalny pobiera atrybuty (sposób oraz klucz) partycjonowania na podstawie związanej z nim tabeli. Indeks globalny może być partycjonowany tylko w sposób zakresowy.

Bardzo istotną cechą z punktu widzenia efektywności działania systemu występującą w przypadku partycjonowania jest możliwość eliminacji odczytu podczas wykonywania zapytania na podstawie zawartości klauzul FROM i WHERE niepotrzebnych partycji. Może to w znaczny sposób zmniejszyć ilość danych przenoszonych z dysku i mieć radykalny wpływ na całokształt przetwarzania.

Podczas utrzymywania systemu istotną kwestią jest posiadanie informacji o wielkości wypełnienia poszczególnych przestrzeni tabel. Informację taką można uzyskać z perspektyw słownika danych wykonując odpowiednie zapytania.

Do interesujących informacji mogą należeć:

- W jakim stopniu aktualnie składowane dane są pofragmentowane ?

```
select dt.tablespace_name, dt.owner, dt.table_name, dt.next_extent,
sum(de.bytes) USED , count(*) NO_EXTENTS
from dba_tables dt, dba_extents de where dt.tablespace_name =
de.tablespace_name and dt.table_name=de.segment_name group by
dt.tablespace_name, dt.table_name, dt.owner, dt.next_extent ;
```

- Ile jest wolnego miejsca na przestrzeni tabel ?

```
select dfs.tablespace_name, sum(dfs.bytes) from dba_free_space dfs group by
dfs.tablespace_name;
```

- W jakim stopniu wolna przestrzeń jest zdefragmentowana

```
select bytes, count(*), sum(bytes) from dba_free_space
```

```
where tablespace_name='Moja przestrzen'  
group by tablespace_name;
```

- Zapytanie wiążące informacje o aktualnej zajętości z informacją o ilości wolnego miejsca dla tabel niepartycjonowanych; (z1 - podzapytanie zwraca wielkość kolejnego rozszerzenia, maksymalną liczbę dostępnych rozszerzeń, sumaryczną ilość zajętego miejsca, ilość wykorzystanych rozszerzeń; z2 - ilość potencjalnie wolnego miejsca na przestrzeni tabel, z3 - zwraca ilość wolnego miejsca, które może być alokowane na potrzeby danej tabeli - oczywiście te wielkości zmieniają się dynamicznie w zależności od rozmieszczenia pozostałych innych elementów na przestrzeni tabel, a także ustawień PCT_INCREASE).

```
SELECT z1.tablespace_name TABLESPACE, z1.owner OWNER, z1.table_name TABLE_NAME,  
z1.used USED, z1.max_extents, z1.NO_EXTENTS_USED, z1.next_extent NEXT_EXTENT,  
z2.total_free TOTAL_FREE, z3.FREE  
from  
(select dt.tablespace_name, dt.owner, dt.table_name, dt.next_extent,  
dt.max_extents, sum(de.bytes) USED , count(*) NO_EXTENTS_USED  
from dba_tables dt, dba_extents de where dt.tablespace_name =  
de.tablespace_name and dt.table_name=de.segment_name group by  
dt.tablespace_name, dt.table_name, dt.owner, dt.next_extent, dt.max_extents )  
z1 ,  
( select dfs.tablespace_name, sum(dfs.bytes) TOTAL_FREE from dba_free_space dfs  
group by dfs.tablespace_name ) z2 ,  
( select dfs.tablespace_name, dt.table_name, sum(dfs.bytes) FREE from  
dba_free_space dfs, dba_tables dt where dfs.tablespace_name =  
dt.tablespace_name and dfs.bytes >= dt.next_extent  
group by dfs.tablespace_name, dt.table_name) z3  
where z1.tablespace_name = z2.tablespace_name and  
z1.tablespace_name=z3.tablespace_name and z1.table_name=z3.table_name;
```

Analogiczne zapytanie można zastosować w odniesieniu do tabeli partycjonowanej (budowa tego zapytania jest analogiczna, wprowadzenie rozbicia z1 na q1 i q2 wynika z braku możliwości grupowania na podstawie pola HIGH_VALUE);

```
SELECT q1.tablespace_name TABLESPACE,  
q1.partition_name PARTITION,  
q1.table_name TABLE_NAME,  
q2.suma USED,  
q2.no_extents_used,  
q2.max_extents,  
q1.next_extent NEXT_EXTENT,  
q4.total_free TOTAL_FREE,  
q3.free FREE  
q1.high_value HIGH_VALUE  
FROM  
( SELECT tablespace_name, partition_name, table_name, high_value, next_extent  
FROM dba_tab_partitions ) q1,  
( SELECT dtp.tablespace_name, dtp.partition_name, dtp.table_name,  
dtp.max_extents, SUM(de.bytes) suma , count(de.bytes) NO_EXTENTS_USED,  
FROM dba_tab_partitions dtp, dba_extents de  
WHERE dtp.tablespace_name=de.tablespace_name and dtp.partition_name =  
de.partition_name and dtp.table_name = de.segment_name  
GROUP BY dtp.tablespace_name, dtp.partition_name, dtp.table_name,  
dtp.max_extents) q2,  
( SELECT dfs.tablespace_name, dtp.partition_name, dtp.table_name,  
sum(dfs.bytes) FREE
```

```

FROM dba_free_space dfs, dba_tab_partitions dtp WHERE dfs.tablespace_name =
dtp.tablespace_name
and dfs.bytes >= dtp.next_extent GROUP BY dfs.tablespace_name,
dtp.partition_name, dtp.table_name) q3,
( SELECT tablespace_name, sum(bytes) TOTAL_FREE from dba_free_space
GROUP BY tablespace_name) q4
WHERE q1.tablespace_name = q3.tablespace_name and q1.partition_name =
q3.partition_name and
q1.table_name = q3.table_name and q1.tablespace_name = q4.tablespace_name and
q1.tablespace_name = q2.tablespace_name and q1.partition_name =
q2.partition_name and
q1.table_name = q2.table_name );

```

3. Indeksy

Celem indeksu jest przyspieszenie dostępu do danych poprzez zapewnienie łatwego dostępu do wskaźnika przechowującego adres do wierszy w tabeli.

W regularnym indeksie B*drzewa jest to uzyskane przez przechowywanie listy rowid (jest to numer identyfikujący jednoznacznie wiersz) dla każdego z wierszy w powiązaniu z wartością klucza. W indeksie bitmap'owym, bitmapa jest tworzona dla każdej wartości klucza zamiast listy rowid.

Indeksy bitmap'owe są szczególnie efektywne w sytuacji gdy ilość różnych wartości w kluczu jest mała. Indeksy bitmap'owe głównie są wykorzystane przez systemy DSS, gdzie przeważają operacje odczytu nad operacjami modyfikacji danych. Indeksy bitmap'owe nie sprawdzają się w sytuacji, gdy wiele równoległych transakcji dokonuje modyfikacji danych.

Istotnym parametrem umożliwiającym określenie przydatności wykorzystania indeksów bitmap'owych jest tzw. kardynalność kolumny – czyli ilość różnych wartości w stosunku do liczby wszystkich rekordów w tabeli. Generalnie rzecz biorąc indeksy bitmap'owe powinny być używane dla kolumn o małej kardynalności, natomiast indeksy B*drzewa dla kolumn o dużej kardynalności, zwłaszcza w sytuacji, gdy typowe zapytania odwołują się do indeksowanych kolumn i zwracają niewiele rekordów.

W przypadku gdy tabela jest partycjonowana, indeks na niej oparty może być globalny lub lokalny. Indeksy globalne muszą być w całości przebudowywane w przypadku wprowadzania nowych rekordów do tabeli. Indeks lokalny jest indeksem prefixowanym (ang. local prefixed index), jeżeli kluczem partycjonowania jest lewy podzbiór kolumn indeksu (w przeciwnym wypadku indeks nie jest prefixowany). W przypadku indeksu globalnego klucze indeksu w jednej partycji mogą odwoływać się do rekordów znajdujących się w wielu partycjach tabeli. Indeks globalny może być podobnie jak lokalny - prefixowany lub nieprefixowany.

Zaleca się w systemach Hurtowni Danych stosowanie indeksów lokalnych.

Podczas utrzymywania systemu istotną kwestią jest posiadanie informacji dotyczących budowy i rozłożenia indeksów na przestrzeni tabel.

Do często zadawanych zapytań należą :

- jakie indeksy są założone na podanej tabeli :

```

SELECT table_name , index_name, index_type , uniqueness, partitioned FROM
dba_indexes WHERE table_name= 'Moja_tabelka'

```

- z jakich kolumn zbudowany jest indeks :

```
SELECT * FROM dba_ind_columns WHERE table_name = 'Moja_tabelka' ORDER BY
index_name, column_position;
```

- jakie jest rozmieszczenie indeksu (nie partycjonowanego) w stosunku do tabeli

```
SELECT dt.table_name, dt.tablespace_name, di.index_name, di.tablespace_name
FROM dba_tables dt, dba_indexes di
WHERE dt.table_name = di.table_name and
dt.owner = di.table_owner;
```

- jakie kolumny są kluczem partycjonowania :

```
SELECT * FROM dba_part_key_columns WHERE name = 'Mój_indeks';
```

- jakie sa parametry indeksu partycjonowanego (lokalny, globalny, ilość partycji) :

```
SELECT index_name, partitioning_type, partition_count, partitioning_key_count,
locality, alignment FROM dba_part_indexes WHERE index_name = 'My index' ;
```

- ile miejsca zajmuje indeks, ile jest jeszcze wolnego .(dla indeksu nie partycjonowanego)

```
SELECT q1.tablespace_name TABLESPACE, q1.owner OWNER, q1.index_name INDEX_NAME,
q1.table_name TABLE_NAME, q1.USED USED, q1.next_extent NEXT_EXTENT,
q1.EXTENTS_NO, q2.total_free TOTAL_FREE, q3.free FREE
from
( SELECT di.tablespace_name, di.owner, di.index_name, di.table_name,
di.next_extent, sum(de.bytes) USED, COUNT(*) EXTENTS_NO FROM dba_indexes di,
dba_extents de
WHERE di.tablespace_name = de.tablespace_name and
di.index_name = de.segment_name
GROUP BY di.tablespace_name, di.owner, di.index_name, di.table_name,
di.next_extent) q1,
( SELECT dfs.tablespace_name, sum(dfs.bytes) TOTAL_FREE from dba_free_space dfs
GROUP BY dfs.tablespace_name ) q2,
( SELECT dfs.tablespace_name, di.index_name, sum(dfs.bytes) FREE
FROM dba_free_space dfs, dba_indexes di
WHERE dfs.tablespace_name = di.tablespace_name and
dfs.bytes >= di.next_extent
GROUP BY dfs.tablespace_name, di.index_name ) q3
WHERE q1.tablespace_name = q2.tablespace_name and
q1.tablespace_name = q3.tablespace_name and
q1.index_name = q3.index_name;
```

4. Ograniczenia integralnościowe

Ograniczenia integralnościowe mają na celu zapewnienie, że dane w bazie danych będą posiadały pewne własności określone przez projektanta systemu. Ograniczenia integralnościowe są

najczęściej używane w celu :

- „oczyszczenia” danych – wykorzystywane w magazynach danych przy wprowadzaniu danych do bazy
- wsparcia optymalizacji zapytań – są wykorzystywane przy opracowywaniu planów zapytania – jest to szczególnie istotne przy przeliczaniu perspektyw materialnych.

Ograniczenia integralnościowe mogą znajdować się w różnych stanach :

- **ENABLEd** – w celu zapewnienia, że wszystkie modyfikacje danych w tabeli, będą weryfikowane pod kątem spełniania określonych warunków. W przypadku modyfikacji, w rezultacie której ograniczenie byłoby zakłócone, operacja modyfikacji zakończy się błędem;
- **VALIDATED** – wszystkie dane, które aktualnie są przechowywane w tabeli muszą spełniać ograniczenie. Jest to stan niezależny od stanu **ENABLEd** tzn. ograniczenie może być w stanie **validated**, i jednocześnie nie znajdować się w trybie **ENABLEd** albo odwrotnie.
- **RELY** – w pewnych sytuacjach twórca systemu zakłada, że jakieś ograniczenie integralnościowe jest spełnione np. w sytuacji, gdy pewne reguły są sprawdzane programowo.

Do typowych ograniczeń integralnościowych występujących w systemach magazynach danych należą :

- Ograniczenie unikatowe - po utworzeniu ograniczenie unikalne jest w stanie **ENABLEd** oraz **VALIDATED**. Na przykład :

```
ALTER TABLE Pracownicy ADD CONSTRAINT prac_unique UNIQUE ( Prac_id );
```

Dla ograniczenia w tym stanie (a dokładniej w stanie **ENABLEd**) automatycznie tworzony jest indeks unikatowy. Indeks jest potrzebny dla zapewnienia możliwości szybkiej weryfikacji, że wprowadzane modyfikacje nie zakłócają ograniczenia. Jednak indeks w pewnych sytuacjach, choćby ze względu na zajmowanie przestrzeni dyskowej może być niepożądany. Alternatywnym rozwiązaniem jest utworzenie ograniczenia w stanie **DISABLE VALIDATE**. Na przykład :

```
ALTER TABLE Pracownicy ADD CONSTRAINT prac_unique UNIQUE (Prac_id)
DISABLE VALIDATE;
```

Jeżeli ograniczenie jest w stanie **DISABLE**, wówczas indeks unikalny nie jest tworzony i niemożliwe jest wykonanie jakiegokolwiek operacji **DML** dokonującej modyfikacji kolumny na którą nałożone jest ograniczenie unikatowe. W tej sytuacji aby przeprowadzić modyfikację należy usunąć ograniczenie, dokonać zmian i ponownie nałożyć ograniczenie w trybie **DISABLED**.

- Ograniczenie typu klucz obcy.
W systemach magazynów danych ograniczenie to tworzone jest w celu zapewnienia poprawności relacji pomiędzy tabelą faktów, a tabelami wymiarów. Na przykład :

```
ALTER TABLE Pracownik ADD CONSTRAINT fk_dept
```

```
FOREIGN KEY ( Prac_Dept_ID) REFERENCES Dept (Dept_ID)
ENABLE NOVALIDATE;
```

Dla tego ograniczenia przydatnym jest stan ENABLE NOVALIDATE. Na przykład w sytuacji gdy dane wprowadzane są do tabeli faktów codziennie, natomiast uzupełnianie tabeli wymiarów odbywa się co tydzień. Wówczas w trakcie tygodnia może dojść do niespełnienia ograniczenia. Jednak wprowadzenie ograniczenia na typu ENABLE NOVALIDATE może mieć na celu zabezpieczenie przed wprowadzeniem niespójności poza normalną procedurą wprowadzania danych do systemu.

- Ograniczenie w stanie RELY.

W wielu sytuacjach weryfikacja poprawności istnienia wymaganych ograniczeń, następuje programistycznie podczas procedury wprowadzania danych do systemu. Wówczas można nałożyć ograniczenie w trybie RELY, na przykład :

```
ALTER TABLE Pracownik ADD CONSTRAINT fk_dept
FOREIGN KEY ( Prac_dept_ID ) REFERENCES Dept(Dept_ID)
DISABLE NOVALIDATE RELY;
```

Tego typu ograniczenie, które nie jest wykorzystywane do zapewnienia istnienia odpowiednich zależności, może być przydatne dla celów optymalizacji zapytań, oraz umożliwienia innym narzędziom łatwej identyfikacji schematu magazynu danych, na podstawie informacji znajdujących się w słowniku danych.

5. Perspektywy materialne

Jedną z technik wykorzystywaną w systemach magazynu danych w celu poprawy efektywności działania zapytań, jest tworzenie tabel agregacyjnych. Tabele te są one pewnym specyficznym rodzajem półproduktów, w których przeliczane są pewne wartości pośrednie, potrzebne do wykonania raportu np. suma sprzedaży na region i produkt.

W Oracle 8i istnieje obiekt wraz z zespołem mechanizmów, umożliwiających zautomatyzowanie procesu generowania i uaktualniania tabel sumarycznych. Obiektem tym jest perspektywa materialna. W systemach magazynu danych perspektywy materialne mogą być używane do przeliczeń i przechowywania zagregowanych danych np. sum sprzedaży. Inne może być przeznaczenie w systemach rozproszonych, gdzie perspektywy materialne służą replikowaniu danych w sposób rozproszony i synchronizacji modyfikacji dokonywanych w różnych miejscach czasoprzestrzeni (ang. Snapshot).

Perspektywa materialna może przyspieszyć zadawanie złożonych zapytań poprzez jednorazowe wykonanie zapytania, obliczenie agregatów i przechowanie rezultatów w bazie danych. Optimalizator automatycznie, po spełnieniu pewnych warunków, rozpoznaje, że istniejąca perspektywa może być przydatna dla wykonania aktualnie zadawanego zapytania, i wówczas zapytanie wykonywane jest na perspektywie zamiast na tabelach źródłowych (mechanizm zamiany zapytań). Perspektywa materialna posiada cechy, upodabniające ją do indeksu. Podobnie jak w przypadku indeksu, celem zastosowania jest wzrost efektywności działania aplikacji, perspektywa materialna wymaga pewnej zajętości na dyskach, a jej zawartość jest modyfikowana wraz z zawartością tabeli źródłowej.

W mechanizmie umożliwiającym funkcjonowanie perspektywy materialnej można wyróżnić kilka komponentów :

- zestaw komend potrzebnych do definiowania perspektywy materialnej i wymiarów ,
- mechanizm odświeżania, którego celem jest zapewnienie, że perspektywa zawiera najbardziej

aktualne informacje ,

- mechanizm umożliwiający zamianę zapytań, tak aby zamiast do tabel źródłowych były kierowane do perspektywy materialnej ,
- zestaw narzędzi rekomendujących jaka perspektywa powinna zostać utworzona.

Z punktu widzenia struktury zapytania definiującej perspektywę materialną można wyróżnić następujące rodzaje perspektyw :

- perspektywy zawierające powiązania między tabelami i dane zagregowane,
- perspektywy zawierające tylko dane zagregowane,
- perspektywy zawierające tylko powiązania między tabelami;

Z punktu widzenia momentu wypełniania danymi można wyróżnić :

- perspektywy budowane natychmiast - BUILD IMMEDIATE - wypełniane danymi w momencie tworzenia perspektywy;
- perspektywy tworzone pierwotnie bez danych - BUILD DEFERRED - wypełniane danymi później przy pomocy perspektywy DBMS_MVIEW.REFRESH .

Z punktu widzenia momentu odświeżania perspektywy materialnej można wyróżnić :

- perspektywy utworzone z klauzulą ON COMMIT - operacja odświeżania uruchamiana jest bezpośrednio po zakończeniu transakcji na tabeli źródłowej. Ten typ odświeżania nie może być używany dla perspektyw zawierających jednocześnie powiązania między tabelami i dane zagregowane;
- perspektywy utworzone z klauzulą ON DEMAND - operacja odświeżania uruchamiania jest wtedy kiedy użytkownik uruchomi procedurę z pakietu DBMS_MVIEW (REFRESH, REFRESH_ALL_MVIEWS, REFRESH_DEPENDENT) ;

Z punktu widzenia sposobu odświeżania perspektywy materialnej można wyróżnić :

- perspektywy z opcją COMPLETE - odświeżanie odbywa się przez całościowe wykonanie zapytania będącego podstawą perspektywy;
- perspektywy z opcją FAST - odświeżanie odbywa się inkrementalnie (przyrostowo) w miarę dodawania danych do tabeli źródłowej (wykorzystany jest log perspektywy materialnej).
- perspektywy z opcją FORCE - odświeżanie jest dokonywane w trybie FAST, jeżeli nie jest możliwe wykonanie odświeżania w trybie FAST dokonywane jest odświeżanie w trybie COMPLETE;
- perspektywy z opcją NEVER - te perspektywy nie będą wcale odświeżane.

Aby perspektywa mogła być odświeżana w trybie FAST muszą być w zależności od struktury zapytania spełnione pewne dodatkowe warunki. Są one dokładnie wyspecyfikowane w dokumentacji systemu Oracle pt. „Data Warehousing Guide”.

Wiele systemów posiada zaimplementowane perspektywy materialne w normalnych tabelach, które są wypełniane za pośrednictwem odpowiednich procedur. Istnieje możliwość rejestracji takiej tabeli jako perspektywy materialnej przy pomocy polecenia CREATE MATERIALIZED VIEW ... ON PREBUILT TABLE.

Perspektywa materialna może być partycjonowana. Definicja i sposób partycjonowania analogicznie jak w przypadku tabeli, podawany jest w momencie tworzenia perspektywy. Występują tutaj analogiczne zalety jak w przypadku tabeli a więc m.in. mechanizm partycjonowania ułatwia administrację, umożliwia skalowalność, oraz tworzenie indeksów lokalnych, które są łatwiejsze w utrzymaniu. Bardzo często kluczem partycjonowania jest wymiar czasowy, co ułatwia archiwizację starych informacji.

6. Wymiary

Wymiar jest strukturą, która dokonuje kategoryzacji danych. Najczęściej występujące wymiary to : KLIENT, PRODUKT, TIME. Wymiary nie muszą być definiowane, jednak ułatwiają przetwarzanie bardziej złożonych zapytań. Wymiary są zorganizowane w odpowiednie hierarchie. Przykładowo : rozważając wymiar lokalizacji można zdefiniować tabelę i wymiar lokalizacji wraz z odpowiednią hierarchią.

```
CREATE TABLE lokalizacja (
  Miasto    varchar2(30) ,
  Stan      varchar2(30) ,
  Kraj      varchar2(30) );
CREATE DIMENSION lokalizacja_dim (
  LEVEL Miasto is lokalizacja.miasto
  LEVEL Stan is lokalizacja.stan
  LEVEL Kraj is lokalizacja.kraj ),
HIERARCHY lokalizacja_rollup (
  Miasto    CHILD OF
  State     CHILD OF
  Kraj      )
```

Istnieje możliwość zdefiniowania kilku hierarchii np. w zakresie wymiaru czasowego.

Przykładowo :

```
CREATE TABLE czas (
  Data      DATE,
  Miesiac   INTEGER,
  Kwartał   INTEGER,
  Rok       INTEGER,
  Sezon     INTEGER,
  Numer_tygodnia  INTEGER,
  Dzień_tygodnia  VARCHAR2(30),
  Nazwa_miesiaca  VARCHAR2(30) );
CREATE DIMENSION czas_dim
  LEVEL Data      IS czas.Data
  LEVEL Miesiac   IS czas.miesiac
  LEVEL Kwartał   IS czas.kwartał
  LEVEL Rok       IS czas.Rok
  LEVEL Sezon     IS czas.Sezon
  LEVEL Numer_tygodnia  IS czas.Numer_tygodnia
HIERARCHY kalendarz (
  Data    CHILD OF
  Miesiac  CHILD OF
  Kwartał  CHILD OF
  Rok      )
```

```

HIERARCHY tydzien (
    Data CHILD OF
    Numer_tygodnia )

HIERARCHY sezon (
    Data CHILD OF
    Sezon )

ATTRIBUTE Data DETERMINES czas.dzien_tygodnia
ATTRIBUTE Miesiac DETERMINES czas.nazwa_miesiaca ;

```

Aby uzyskać informacje o utworzonych wymiarach można wykorzystać pakiet DEMO_DIM, zawierający procedury PRINT_DIM, oraz PRINT_ALLDIMS.

DEMO_DIM.PRINT_DIM – wydruk informacji o konkretnym wymiarze,
DEMO_DIM.PRINT_ALLDIMS - wydruk informacji o wszystkich wymiarach.

7. Rozszerzenia języka zapytań SQL wspomagające wykonywanie zapytań o charakterze analitycznym do bazy danych

Rozszerzenia języka zapytań SQL wspomagających przetwarzanie analityczne to :

- operatory CUBE i ROLLUP w klauzuli GROUP BY ;
- rodzina funkcji analitycznych ;
- funkcje wspomagające regresję liniową ;
- wyrażenie CASE ;

7.1. Operatory CUBE i ROLLUP w klauzuli GROUP BY

Operator ROLLUP umożliwia kalkulację wielopoziomowych podsumowań według wyspecyfikowanych wymiarów od najbardziej szczegółowych do globalnych. Na przykład :

```

SELECT Data, Region, Rodzaj, Sum(Zysk), FROM Dane_Sprzedaz
GROUP BY ROLLUP ( Data, Region, Rodzaj );

```

Po wykonaniu tej komendy uzyskamy :

Data	Region	Rodzaj	Profit
1996	Centrum	Leasing	10000
1996	Centrum	Sprzedaż	20000
1996	Centrum	NULL	30000
1996	Wschód	Leasing	50000
1996	Wschód	Sprzedaż	30000
1996	Wschód	NULL	80000
1996	NULL	NULL	110000

NULL NULL NULL 110000

Wartości NULL zwracane przez operatory ROLLUP i CUBE wskazują, że dany wiersz zawiera podsumowanie. Operator ROLLUP może być zastosowany do podzbioru kolumn występujących w klauzuli GROUP BY. Na przykład :

```
SELECT Data, Region, Rodzaj, Sum(Zysk), FROM Dane_Sprzedaz
GROUP BY ROLLUP Data, ( Region, Rodzaj );
```

Wówczas w wyniku nie wystąpi wiersz z totalnym podsumowaniem.

Analogiczne wyniki można uzyskać stosując zestaw zapytań SELECT połączonych operatorem UNION ALL. Jednak efektywność takiego zapytania jest znacznie gorsza, ponieważ wymaga wielokrotnego przebiegu po tabeli, podczas gdy ROLLUP dokonuje wymaganych przeliczeń w trakcie jednego dostępu do tabeli.

Wykorzystanie operatora ROLLUP jest szczególnie przydatne do przeliczania podsumowań dla wymiarów posiadających ścisłą hierarchię, takich jak czas albo lokalizacja.

Operator CUBE umożliwia wyznaczenie podsumowań dla wszystkich możliwych kombinacji i kolejności grupowania wymiarów. Dla n wymiarów można więc wyznaczyć 2^n ilości podsumowań

Analogicznie jak w przypadku ROLLUP istnieje możliwość zastosowania operatora CUBE do podzbioru kolumn.

Operatory ROLLUP oraz CUBE mogą być wykorzystywane z funkcjami dostępnymi dla klauzuli GROUP BY, takich jak COUNT, AVG, MIN, MAX, STDDEV, VARIANCE.

Wraz z operatorem ROLLUP, oraz CUBE wprowadzona została funkcja GROUPING. Argumentem tej funkcji jest kolumna. Funkcja ta zwraca 1, jeżeli w tej kolumnie pojawi się NULL wygenerowany przez operator ROLLUP lub CUBE.

Na przykład :

```
SELECT Data, Region, Rodzaj, Sum(Zysk)
GROUPING (Time) AS T,
GROUPING (Region) AS R,
GROUPING (Rodzaj) AS D
FROM Dane_Sprzedaz
GROUP BY ROLLUP ( Data, Region, Rodzaj );
```

Przykładowy wynik :

Czas	Region	Rodzaj	Zysk	T	R	D
1996	Centrum	NULL	10000	0	0	1

Przy pomocy funkcji DECODE w połączeniu z funkcją GROUPING można w łatwy sposób dokonać opisu otrzymanych wyników np. DECODE (GROUPING(Rodzaj), 1, 'Wszystkie rodzaje', Rodzaj);

Funkcja GROUPING daje możliwość sortowania i filtrowania otrzymanych wyników np. wybrać tylko niektóre rodzaje podsumowań :

```
SELECT Data, Region, Rodzaj, Sum(Zysk) As Zysk,
GROUPING(Data) AS T, GROUPING(Region) AS R, GROUPING(Rodzaj) AS D
```

```

FROM Dane_Sprzedaz
GROUP BY CUBE ( Data, Region, Department )
HAVING ( GROUPING(Rodzaj) = 1 AND GROUPING(Region) = 1 AND GROUPING(Data)=1)
OR ( GROUPING(Region) = 1 AND GROUPING(Rodzaj) = 1))

```

7.2. Funkcje analityczne

System Oracle 8i oferuje następujące funkcje analityczne :

- RANK, DENSE RANK - umożliwiają określenie pozycji w zestawieniu, którą zajmuje określony rekord według pewnego określonego kryterium. Różnica pomiędzy tymi dwoma funkcjami polega na tym, że w przypadku gdy kilka rekordów zajmie tę samą pozycję, to funkcja DENSE_RANK umieści kolejny rekord na pozycji bezpośrednio następnej, natomiast funkcja RANK uwzględni ilość rekordów zajmujących tę samą pozycję. Na przykład : Jeżeli na drugim miejscu znajdują się trzy rekordy, to kolejnym miejscem w przypadku funkcji RANK będzie miejsce 5, a w przypadku DENSE_RANK miejsce 3.

Składnia jest następująca :

```

RANK() | DENSE_RANK() OVER (
[ PARTITION BY < value_expression1 > [ , ... ] ]
ORDER BY < value_expression2 > [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ] [ , ... ] )

```

Klauzula ORDER BY określa kolumny na podstawie których dokonywany jest ranking, oraz definiuje porządek sortowania.

Pożyteczną funkcjonalność zapewnia występujące w składni funkcji wyrażenie PARTITION BY, które umożliwia podzielenie wyników zapytania na grupy, i określenie pozycji rekordów w obrębie tych grup.

Przykładowe wykorzystanie :

```

SELECT region, produkt, SUM(sprzedaz)
RANK() OVER ( PARTITION BY region
ORDER BY SUM(sprzedaz) DESC
AS rank_region
FROM Dane_Sprzedaz
GROUP BY region, produkt;

```

- CUME_DIST - funkcja przyjmuje wartości od 0 do 1, i zwraca ułamek określający iloraz wartości mierzonej w wierszu do wartości maksymalnej występującej w całej grupie ;
- ROW_NUMBER - funkcja przyporządkowuje unikalny kolejny numer dla każdego wiersza w obrębie zdefiniowanej partycji według ustalonego porządku sortowania w klauzuli ORDER BY wewnątrz funkcji analitycznej ;
- Zestaw funkcji umożliwiających przeliczanie statystyk na podzbiórze rekordów - tzw. funkcje przedziałowe (ang. windowing functions).

Składnia funkcji przedziałowych jest następująca :

```

{SUM|AVG|MAX|MIN|COUNT|STDDEV|VARIANCE|FIRST_VALUE|LAST_VALUE}
( {<value expression1 > | * } ) OVER
( [PARTITION BY < value expression2 >[,...]]
ORDER BY < value expression3>

```

```

[ ASC | DESC ] [ NULLS FIRST | NULLS LAST ] [,...]
ROWS | RANGE
{{ UNBOUNDED PRECEDING | < value expression4 > PRECEDING
  | BETWEEN
    { UNBOUNDED PRECEDING | < value expression4 > PRECEDING
AND { CURRENT ROW | < value expression4> FOLLOWING }}

```

Klauzula PARTITION BY umożliwia definiowanie grupy rekordów do których odnosi się rozważana funkcja. Jeżeli ta klauzula zostanie pominięta, wówczas wynik zapytania jest przeliczany na wszystkich rekordach na których działa funkcja.

Klauzula ORDER BY specyfikuje uporządkowanie rekordów w obrębie zdefiniowanej klauzulą PARTITION BY grupy rekordów.

Operatory ASC, DESC określają porządek sortowania, a operatory NULLS FIRST, NULLS LAST specyfikują czy rekordy posiadające wartości NULL powinny pojawiać się na początku, czy na końcu.

Klauzula ROWS | RANGE definiuje okno, które może być określone jako fizyczny bądź logiczny zbiór wierszy. Klauzula UNBOUNDED PRECEDING określa, że okno rozpoczyna się pierwszym wierszem partycji, lub jeżeli partycja jest niezdefiniowana pierwszym wierszem w zbiorze danych. Analogicznie UNBOUNDED FOLLOWING określa, że okno kończy się na ostatnim wierszu partycji, lub jeżeli partycja jest niezdefiniowana na ostatnim wierszu w zbiorze danych.

Przykłady :

1. Poniższe zapytanie zwraca obrót na koncie oraz średnią krocącą z 7 ostatnich dni.

```

SELECT Numer_konta, Data, Obrot,
AVG(Obrot) OVER
( PARTITION BY Numer_konta ORDER BY Data
RANGE INTERVAL '7' DAY PRECEDING ) AS Mavg_7
FROM Ksiega_Glowna;

```

2. Poniżej zdefiniowane zapytanie zwraca średnią liczoną z okresu od miesiąca poprzedzającego do następującego w stosunku do rozpatrywanego wiersza.

```

SELECT Numer_konta, Data, Obrot,
AVG(Obrot) OVER
( PARTITION BY Numer_konta ORDER BY Data
RANGE BETWEEN INTERVAL '1' MONTH PRECEDING
AND INTERVAL '1' MONTH FOLLOWING ) AS M_Avg
FROM Ksiaga_Glowna;

```

- Funkcja RATIO_TO_REPORT dokonuje obliczenia ilorazu danej wartości w stosunku do sum wartości zbioru rekordów. Na przykład :

```

SELECT produkt, SUM(kwota), SUM(SUM(kwota)) OVER () AS total,
RATIO_TO_REPORT( SUM(kwota)) OVER () AS ratio_to_report
FROM Dane_Sprzedaz
GROUP BY produkt;

```

- Funkcje LAG, oraz LEAD umożliwiają dostęp do kilku wierszy tej samej tabeli bez konieczności tworzenia połączeń (ang. self join). Funkcja LAG umożliwia dostęp do wiersza oddalonego o określony offset wstecz w stosunku do aktualnego wiersza, natomiast funkcja

LEAD umożliwia dostęp do wiersza oddalonego o określony offset wprzód w stosunku do aktualnego wiersza. Na przykład :

```
SELECT czas, kwota,  
LAG(kwota,1) OVER ( ORDER BY czas ) AS LAG_kwota,  
LEAD(kwota,1) OVER ( ORDER BY czas ) AS LEAD_kwota  
FROM OBROT;
```

7.3. Funkcje statystyczne

Funkcje statystyczne umożliwiają m.in. obliczanie kowariancji, korelacji, i regresji liniowej. Do podstawowych funkcji należą :

- VAR_POP - wariancja populacji ,
- VAR_SAMP - wariancja próbki ,
- STDDEV_POP / STDDEV_SAMP - odchylenie standardowe dla próbki i populacji ,
- COVAR_POP, COVAR_SAMP - kowariancja dla populacji i próbki ,
- CORR - współczynnik korelacji ,

Oracle zapewnia zestaw funkcji umożliwiających wyznaczenie współczynników metodą regresji liniowej. Są to REGR_COUNT, REGR_AVGX, REGR_AVGY, REGR_SLOPE, REGR_INTERCEPT, REGR_R2, REGR_SXX, REGR_SYY, REGR_SXY.

7.4. Instrukcja CASE

Instrukcja CASE działa w podobny sposób jak instrukcja DECODE. Jest bardziej efektywna, umożliwia budowę bardziej złożonych warunków logicznych. Składnia tej instrukcji jest następująca :

```
CASE WHEN <cond1> THEN <v1> WHEN <cond2> THEN <v2> ... [ ELSE <vn+1> ] END
```

Na przykład poniższe wyrażenie znajduje średni zarobek pracowników, jednak jeżeli wynagrodzenie jest mniejsze od 2000\$, to w obliczeniach należy przyjąć 2000\$.

```
SELECT AVG(CASE when placa>2000 THEN placa ELSE 2000 end ) FROM Pracownik;
```

Bibliografia

1. David Austin : „Poznaj Oracle 8” , 1999, ISBN 83-7158-153-X
2. Dokumentacja do systemu : Oracle 8i, „Data Warehousing Guide”, 1999,
3. Ralph Kimball, W.H.Inmon : „The Data Warehouse Toolkit”, 2000, ISBN 0-471-15337-0
4. Wrembel, Jezierski, Zakrzewicz : „System zarządzania bazą danych Oracle 7 i Oracle 8”, ISBN 83-86969-34-2
5. Materiały z konferencji PLOUG 1999, PLOUG 2000.