



POZNAN UNIVERSITY OF TECHNOLOGY

Perspektywy zmaterializowane

Robert Wrembel

Politechnika Poznańska

Instytut Informatyki

Robert.Wrembel@cs.put.poznan.pl

www.cs.put.poznan.pl/rwrembel





Materializowanie wyników zapytania

- ⇒ **Cel → optymalizacja zapytań analitycznych**
- ⇒ **Mechanizmy i obiekty wykorzystywane w optymalizacji**
 - **Perspektywy zmaterializowane (materialized views - Oracle, SQL Server, summary tables, materialized query tables - IBM)**
 - **Przepisywanie zapytań (query rewrite)**
 - **Wymiary**

Perspektywa zmaterializowana

⇒ Definicja zawiera:

- zapytanie odwołujące się do tabel
- sposób odświeżania danych (pełne, przyrostowe)
- częstotliwość odświeżania danych

⇒ Standardowo tylko do odczytu

⇒ Implementacja

- tabela + indeks

Perspektywa zmaterializowana

⇒ Rodzaje (sposób identyfikowania rekordów)

▪ PRIMARY KEY

- tabela master musi posiadać włączone ograniczenie PRIMARY KEY
- klauzula SELECT musi zawierać wszystkie atrybuty wchodzące w skład klucza podstawowego tabeli master

▪ ROWID

⇒ Rodzaje (struktura zapytania)

▪ perspektywa prosta

- bazująca na jednej tabeli master
- brak klauzul: GROUP BY, CONNECT BY, DISTINCT
- brak funkcji, połączeń, operatorów zbiorowych

▪ perspektywa złożona

Definiowanie perspektywy zmaterializowanej

CREATE MATERIALIZED VIEW nazwa

BUILD

IMMEDIATE

DEFERRED

REFRESH

FAST

COMPLETE

FORCE

ON DEMAND

ON COMMIT

WITH

PRIMARY KEY

ROWID

START WITH 'data pierwszego odświeżenia'

NEXT 'data następnego odświeżenia'

AS SELECT ...;



Przykład

```
CREATE MATERIALIZED VIEW mv_suma_sprzedazy  
BUILD IMMEDIATE  
REFRESH FAST  
NEXT sysdate+(1/(24*60*30))  
as  
select sklep_id, produkt_id,  
       sum(l_sztuk), sum(l_sztuk*cena_jedn),  
       count(l_sztuk), count(l_sztuk*cena_jedn), count(*)  
from sprzedaz  
group by sklep_id, produkt_id;
```



Odświeżanie perspektywy zmaterializowanej

⇒ Sposób odświeżania

- **REFRESH FAST** ⇒ odświeżanie przyrostowe
- **REFRESH COMPLETE** ⇒ odświeżanie pełne
- **REFRESH FORCE** ⇒ automatyczny wybór metody odświeżania; jeżeli możliwe to Oracle wybiera FAST

⇒ Częstotliwość odświeżania

- **START WITH** ⇒ data pierwszego odświeżenia
- **NEXT** ⇒ wyrażenie określające częstotliwość odświeżania

⇒ Odświeżanie automatyczne gdy:

- wyspecyfikowana klauzula **NEXT**
- określona częstotliwość odświeżania
 - **REFRESH FAST START WITH sysdate NEXT sysdate+1**
 - **REFRESH FAST NEXT sysdate+1**

Odświeżanie przyrostowe

- ➔ Dla perspektyw prostych
- ➔ Musi istnieć **MATERIALIZED VIEW LOG** dla tabeli master
- ➔ Perspektywa wyliczająca agregaty: **count, sum, avg, variance, stdev**
 - dziennik utworzony z klauzulą **INCLUDING NEW VALUES**
 - dziennik zawiera wszystkie atrybuty wymienione po **SELECT**, również będące argumentami wywołania funkcji grupowych
 - dla **SUM(X)**, **AVG(X)** należy wyliczać **COUNT(X)** i **COUNT(*)**

Odświeżanie perspektywy zmaterializowanej

⇒ **ON COMMIT** można stosować jedynie, gdy:

- zapytanie korzysta z tabel lokalnych
- migawek opartych o jedną tabelę, bez wyliczania agregatów
- migawek, których zapytanie wyznacza agregaty w oparciu o pojedynczą tabelę
- migawek których zapytanie wykorzystuje łączenie tabel, ale bez wyliczania agregatów

⇒ **Uprawnienia wymagane do doświeżania ON COMMIT**

- uprawnienie obiektowe **ON COMMIT** dla wszystkich tabel, do których odwołuje się perspektywa zmaterializowana
- uprawnienie systemowe **ON COMMIT**

Odświeżanie perspektywy zmaterializowanej

- Odświeżanie ręczne, gdy:
 - wyspecyfikowano **ON DEMAND**
 - brak klauzuli **NEXT**
 - perspektywa odświeżona raz w momencie jej tworzenia jeśli wyspecyfikowano **BUILD IMMEDIATE**

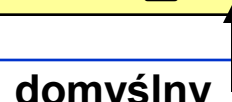
```
DBMS_MVIEW.REFRESH ('sn1, sn2, ..., snn', 'metoda')
```

- $sn_1, sn_2, \dots, sn_n$: perspektywy zmaterializowane
- metoda: metoda odświeżania
 - f lub F: FAST
 - c lub C: COMPLETE
 - ?: domyślny

```
DBMS_MVIEW.REFRESH ('s_dept, s_emp, s_emp1', 'C')
```

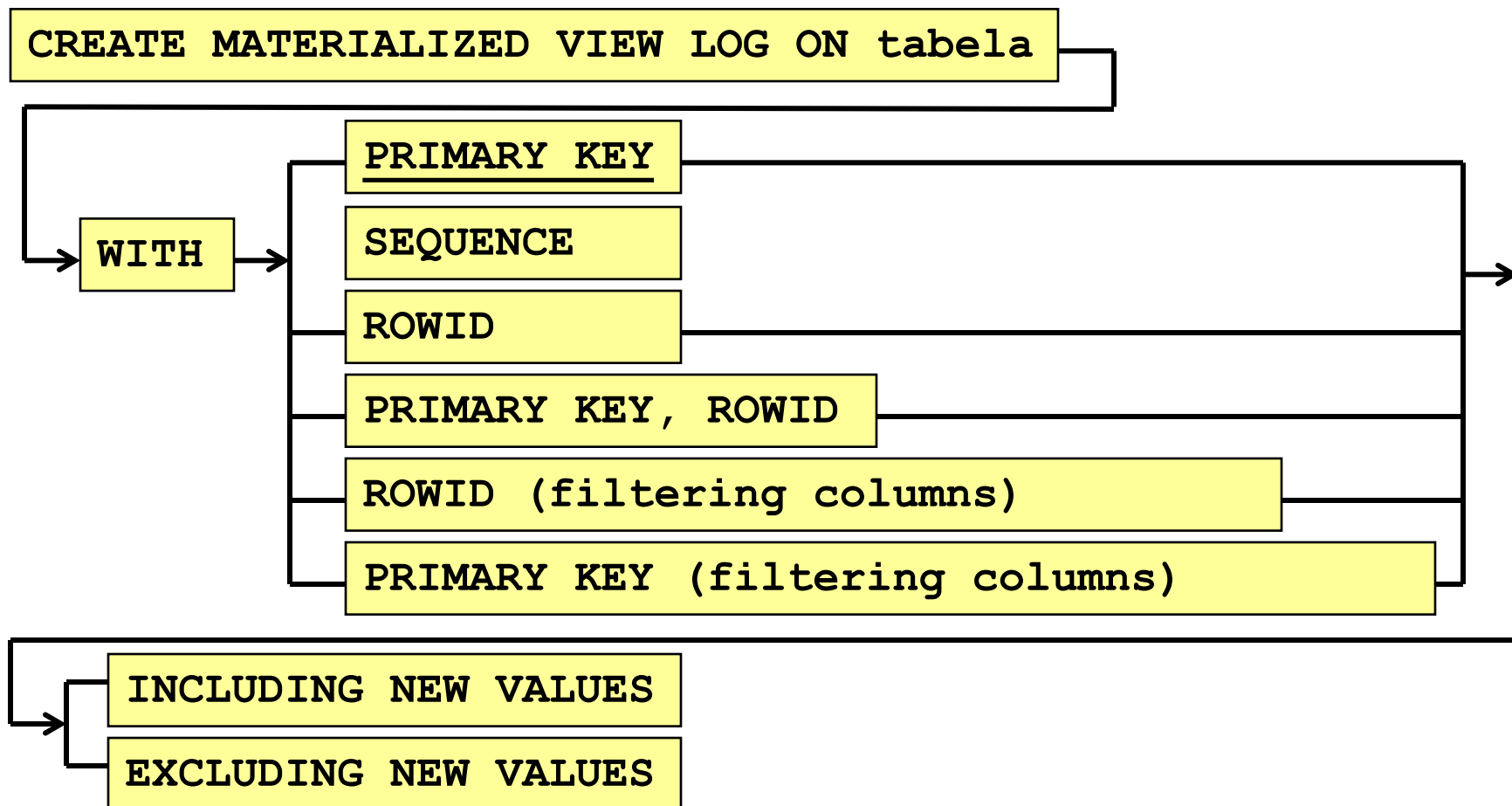


```
DBMS_MVIEW.REFRESH ('s_dept, s_emp, s_emp1', 'CF')
```



domyślny

Dziennik perspektywy zmaterializowanej





Dziennik perspektywy zmaterializowanej

⇒ Implementowany jako tabela
MLOG\$_nazwatabbazowej

⇒ **WITH PRIMARY KEY**

- w dzienniku rejestrowane wartości atrybutów wchodzących w skład klucza

⇒ **WITH ROWID**

- w dzienniku rejestrowane ROWID rekordów

⇒ **WITH PRIMARY KEY, ROWID**

- w dzienniku są rejestrowane wartości atrybutów kluczowych i ROWID



Dziennik perspektywy zmaterializowanej

➔ WITH SEQUENCE

- konieczne do odświeżania przyrostowego, gdy do tabeli bazowej kierowane są przemieszane polecenia IUD
- MV log będzie zawierał kolumnę SEQUENCE\$\$ przechowującą porządek wykonywania operacji IUD na tabeli bazowej

Dziennik perspektywy zmaterializowanej

- **Filtering column** $\Rightarrow$ atrybut występujący w:
 - klauzuli **WHERE** zapytania definiującego perspektywę (filtrowanie)
 - klauzuli **SELECT**
 - klauzuli **SELECT** jako argument wywołania funkcji grupowych
 - może również wymagać wskazania atrybutów połączeniowych

```
select sk.nazwa, sk.sklep_id
from scott.sklepy
where exists (select sp.sklep_id
              from scott.sprzedaz
              where sp.sklep_id=sk.sklep_id
              and sp.produkt_id=100
              and sp.data='23.01.2002'
              and sp.l_sztuk>1)
```

Dziennik perspektywy zmaterializowanej

■ INCLUDING NEW VALUES

- konieczne dla perspektyw odświeżanych przyrostowo zawierających agregaty
- MV log będzie zawierał poprzednie i nowe wartości kolumn (col1, ..., coln)
- pominięcie tej klauzuli powoduje przechowywanie w MV logu tylko poprzednich wartości (col1, ..., coln)

```
create materialized view log on BaseTable  
with primary key|ROWID (col1, ..., coln),  
sequence  
including new values;
```

Przykład

```
CREATE MATERIALIZED VIEW mv_suma_sprzedazy  
BUILD IMMEDIATE  
REFRESH FAST  
NEXT sysdate+(1/(24*60*30))  
as  
select sklep_id, produkt_id, sum(l_sztuk), sum(l_sztuk*cena_jedn),  
       count(l_sztuk), count(l_sztuk*cena_jedn), count(*)  
from sprzedaz  
group by sklep_id, produkt_id;
```

```
CREATE MATERIALIZED VIEW LOG on sprzedaz  
WITH PRIMARY KEY,  
     ROWID (l_sztuk, cena_jedn, sklep_id, produkt_id)  
INCLUDING NEW VALUES;
```



EXPLAIN_MVIEW

- ➔ Procedura **DBMS_MVIEW.EXPLAIN_MVIEW**
- ➔ Analizuje zapytanie definiujące perspektywę i sprawdza, czy perspektywa może być odświeżana przyrostowo
- ➔ W schemacie użytkownika należy uruchomić skrypt `%ORACLE_HOME%\rdbms\admin\utlxmv.sql` tworzący tabelę **MV_CAPABILITIES_TABLE** przechowującą wyniki analiz



EXPLAIN_MVIEW

```
BEGIN
dbms_mview.explain_mview (
'select sp.rowid sp_rowid, sp.produkt_id, sp.l_sztuk,
sk.rowid sk_rowid, sk.nazwa
from sprzedaz sp, sklepy sk
where sp.sklep_id=sk.sklep_id', '1' *; statement_id
END;
/
```

```
SELECT capability_name, possible,
       related_text "table", msgtxt explanation
FROM MV_CAPABILITIES_TABLE
where statement_id='1';
```



EXPLAIN_MVIEW

CAPABILITY_NAME	P table	EXPLANATION
-----	-----	-----
PCT	N	
REFRESH_COMPLETE	Y	
REFRESH_FAST	N	
REWRITE	Y	
PCT_TABLE	N SPRZEDAZ	relation is not a partitioned table
PCT_TABLE	N SKLEPY	relation is not a partitioned table
REFRESH_FAST_AFTER_INSERT	N DEMO.SPRZEDAZ	the detail table does not have a materialized vlog
REFRESH_FAST_AFTER_ONETAB_DML	N	see the reason why REFRESH_FAST_AFTER_INSERT is disabled
REFRESH_FAST_AFTER_ANY_DML	N	see the reason why REFRESH_FAST_AFTER_ONETAB_DML is disabled
REFRESH_FAST_PCT	N	PCT is not possible on any of the detail tables in the materialized view
REWRITE_FULL_TEXT_MATCH	Y	
REWRITE_PARTIAL_TEXT_MATCH	Y	
REWRITE_GENERAL	Y	
REWRITE_PCT	N	general rewrite is not possible and PCT is not possible on any of the detail tables

Prebuilt table

- W systemie znajdują się "zwykłe" tabele zawierające dane
- Zamiana zwykłych tabel na perspektywy zmaterializowane → wykorzystanie do przepisywania zapytań
- Klauzula **ON PREBUILT TABLE**

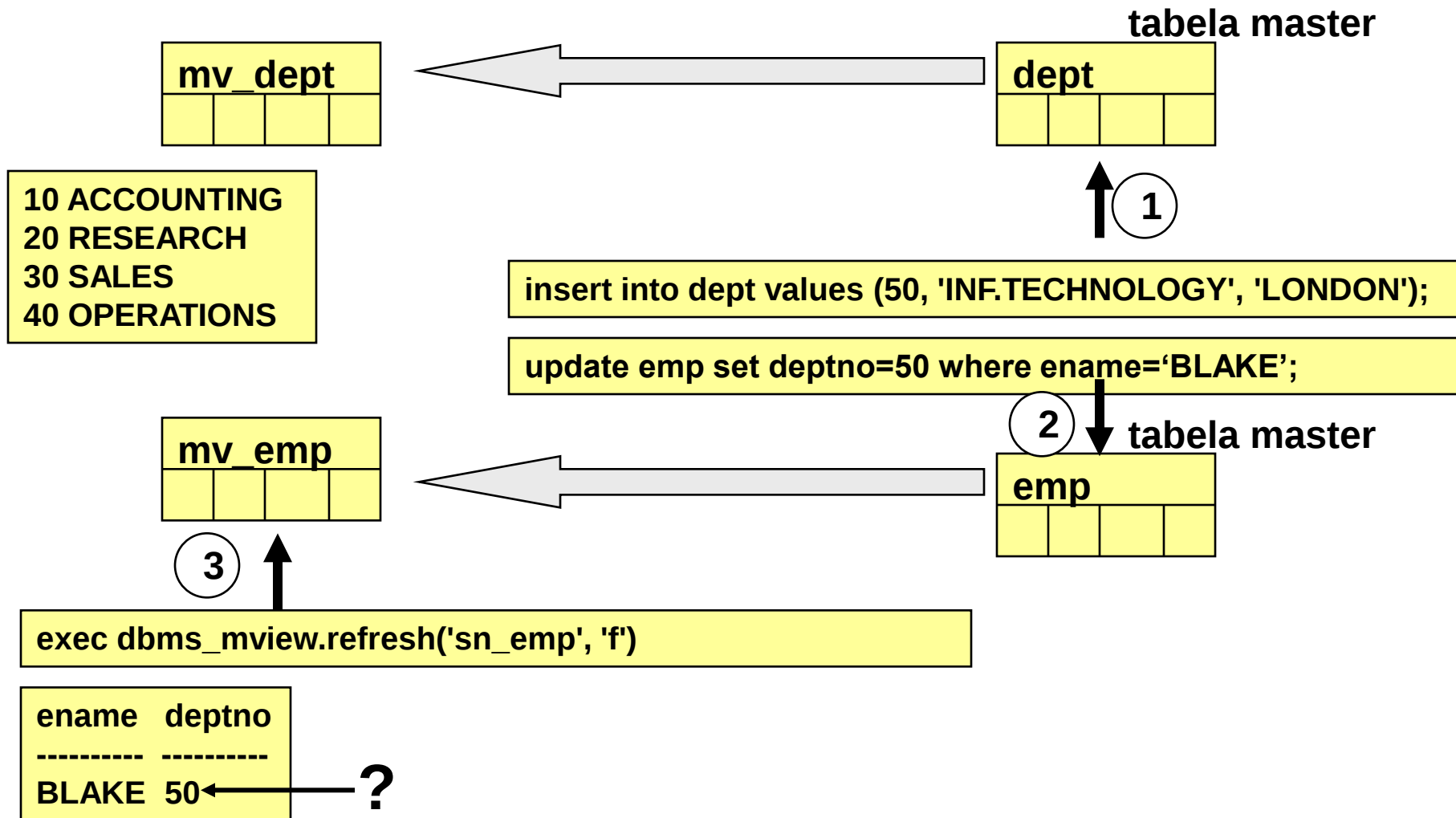
```
CREATE TABLE monthly_customer_sales  
( YEAR NUMBER(4), MONTH NUMBER(2),  
  CUSTOMER_ID VARCHAR2(10), DOLLAR_SALES NUMBER);
```

```
CREATE MATERIALIZED VIEW monthly_customer_sales  
ON PREBUILT TABLE  
ENABLE QUERY REWRITE  
AS  
SELECT t.year, t.month, c.customer_id,  
       SUM(f.purchase_price) dollar_sales  
FROM time t, purchases f, customer c  
WHERE f.time_key = t.time_key AND f.customer_id = c.customer_id  
GROUP BY t.year, t.month, c.customer_id;
```

Prebuilt table

- **Nazwa perspektywy zmaterjalizowanej budowanej na istniejącej tabeli musi być identyczna z nazwą tej tabeli**
- **System nie sprawdza poprawności zawartości tabeli w kontekście zapytania definiującego perspektywę**
 - **tabela może zawierać dane nieodpowiadające zapytaniu**
- **Perspektywa będzie odświeżana z wykorzystaniem standardowych mechanizmów, jeśli je zdefiniowano**

Zależności między perspektywami



Grupy odświeżania

DBMS_REFRESH.MAKE

(name ,	← nazwa grupy
list ,	← lista perspektyw przypisywanych do grupy
next_date ,	← data następnego odświeżenia
interval ,	← okres odświeżania
implicit_destroy)	← TRUE: usunięcie grupy po usunięciu ostatniej perspektywy; domyślnie FALSE

➤ lista perspektyw zmaterializowanych

- muszą być w tej samej bd
- mogą być w różnych schematach
- max. 100 migawek w grupie

Przykłady

```
exec DBMS_REFRESH.MAKE
```

```
(name => 'orcl.rg_dept_emp', -  
list => 'orcl.sn_dept, orcl.sn_emp', -  
next_date => sysdate+1/48, -  
interval => 'next_day(trunc(sysdate), ''FRIDAY'')+10/24', -  
implicit_destroy => TRUE, -  
rollback_seg => 'rb1')
```

```
exec DBMS_REFRESH.ADD ('orcl.rg_dept_emp', 'orcl.sn_emp1')
```

```
exec DBMS_REFRESH.SUBTRACT('orcl.rg_dept_emp', 'orcl.sn_emp1')
```

```
exec DBMS_REFRESH.REFRESH('orcl.rg_dept_emp')
```

```
exec DBMS_REFRESH.DESTROY('orcl.rg_dept_emp')
```

usuwa grupę z perspektywami lub pustą



Metadane słownikowe

- ➔ **Zmaterializowane perspektywy**
 - **USER_MVIEWS**
- ➔ **Dzienniki zmaterializowanych perspektyw**
 - **USER_MVIEW_LOGS**
- ➔ **Informacje nt. odświeżania**
 - **USER_MVIEW_REFRESH_TIMES**
- ➔ **Grupy odświeżania**
 - **USER_REFRESH_CHILDREN**
- ➔ **Zmaterializowane perspektywy w grupie**
 - **USER_REFRESH**

Przepisywanie zapytań

CREATE MATERIALIZED VIEW nazwa

BUILD

.....

WITH

.....

START WITH ...

NEXT ...

ENABLE QUERY REWRITE

AS SELECT ...;

uwaga: klauzula ENABLE QUERY REWRITE może
kłócić się z NEXT,
wynikiem jest błąd:

ORA-030353

expression not supported for query rewrite

rozwiązanie:

utworzyć MView z NEXT

wykonać polecenie

alter materialized view ...

enable query rewrite



Przepisywanie zapytań - przykład

```
create materialized view mv_sprzedaz
build immediate
refresh force
next sysdate + (1/24)
ENABLE QUERY REWRITE
as
select sk.miasto, pr.prod_nazwa, cz.nazwa_miesiaca,
       sum(sp.l_sztuk) sprzedano, sum(sp.wartosc) wartosc
from sprzedaz sp, sklepy sk, produkty pr, czas cz
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=pr.produkt_id
and sp.data=cz.data
group by sk.miasto, pr.prod_nazwa,
         cz.nazwa_miesiaca;
```

```
select sk.miasto, pr.prod_nazwa,
       sum(sp.wartosc) wartosc
from sprzedaz sp, sklepy sk,
       produkty pr, czas cz
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=pr.produkt_id
and sp.data=cz.data
and sk.miasto='Poznań'
group by sk.miasto, pr.prod_nazwa,
         cz.nazwa_miesiaca;
```

Execution Plan

```
-----
0          SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=1 Bytes=37)
1    0      TABLE ACCESS (FULL) OF 'MV_SPRZEDAZ' (Cost=2 Card=1 Bytes=37)
```

Wskazówki

➤ NOREWRITE

➤ REWRITE (perspektywa zmaterializowana)

```
select /*+ REWRITE (mv_sprzedaz) */
      sk.wojewodztwo, pr.prod_nazwa, sum(sp.wartosc) wartosc
from sprzedaz sp, sklepy sk, produkty pr, czas cz
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=pr.produkt_id
and sp.data=cz.data
and sk.wojewodztwo='wielkopolskie'
having sum(sp.wartosc)>190
group by sk.wojewodztwo, pr.prod_nazwa, cz.nazwa_miesiaca;
```

```
select /*+ NOREWRITE */
      sk.wojewodztwo, sum(sp.wartosc) wartosc
from sprzedaz sp, sklepy sk
where sp.sklep_id=sk.sklep_id
and sk.wojewodztwo='wielkopolskie'
having sum(sp.wartosc)>190
group by sk.wojewodztwo;
```

Konfigurowanie (1)

- **Przepisanie zapytania jest możliwe tylko wówczas, gdy:**
 - **system wykorzystuje optymalizator kosztowy (ważne do 10g)**
 - parametr konfiguracyjny instancji **OPTIMIZER_MODE=CHOOSE**
 - dynamiczny wybór optymalizatora dla sesji

```
alter session set optimizer_mode='choose';  
alter session set optimizer_mode='all_rows';
```

- dla tabel bazowych i perspektyw zmaterializowanych zebrano statystyki

```
exec dbms_stats.gather_table_stats('USER_STAR','sprzedaz');
```

Konfigurowanie (2)

➤ Parametr konfiguracyjny

- **QUERY_REWRITE_ENABLED=TRUE**
- wówczas możliwe dynamiczne włączenie przepisывania dla sesji

```
alter session set query_rewrite_enabled=true;
```

- **QUERY_REWRITE_ENABLED=FORCE**

- jeżeli istnieje odpowiednia zmaterializowana perspektywa, wówczas zostanie ona wykorzystana do przepisania zapytania niezależnie od tego, czy przepisane zapytanie będzie mniej lub bardziej efektywne → nie jest porównywany koszt wykonania zapytania

➤ Blokowanie przepisывania dla wybranej perspektywy

```
alter materialized view nazwa_perspektywy  
{enable | disable} query rewrite;
```

Konfigurowanie (3)

- **Uprawnienia użytkownika niezbędne do wykorzystania przepisывania zapytań:**
 - uprawnienie obiektowe **QUERY REWRITE** na wszystkich tabelach, do których odwołuje się perspektywa zmaterializowana
 - lub uprawnienie systemowe **GLOBAL QUERY REWRITE**
- **Uwaga:** perspektywa z operatorem **BETWEEN AND** wskazującym zakres dat nie będzie wykorzystana do przepisывania zapytań

ORA-30353: expression not supported for query rewrite

- **Techniki przepisывania zapytań**
 - wyliczanie agregatów (aggregate rollup/computation)
 - join-back
 - filtrowanie (filtered data)
 - wykorzystanie obiektu **DIMENSION** (using dimension)

Aggregate roll-up

- **Wynik zapytania użytkownika jest wyznaczany za pomocą agregowania zawartości perspektywy zmaterializowanej**
 - **perspektywa udostępnia poniższe dane**

Name	Null?	Type
-----	-----	-----
MIASTO	NOT NULL	VARCHAR2 (10)
PROD_NAZWA	NOT NULL	VARCHAR2 (30)
NAZWA_MIESIACA		VARCHAR2 (15)
SPRZEDANO		NUMBER
WARTOSC		NUMBER

- **zapytanie użytkownika**

```
select pr.prod_nazwa, sum(sp.wartosc) wartosc
from sprzedaz sp, produkty pr
where sp.produkt_id=pr.produkt_id
group by pr.prod_nazwa;
```

Join-back (1)

- W celu wyznaczenia wyników zapytania użytkownika, perspektywa zmaterializowana jest łączona z jedną z jej tabel bazowych
- Perspektywa musi zawierać albo klucz podstawowy tej tabeli bazowej lub ROWID rekordów tabeli bazowej

```
create materialized view mv_sprzedaz1
build immediate
refresh complete
enable query rewrite
as
select sk.miasto, pr.produkt_id, cz.nazwa_miesiaca,
       sum(sp.wartosc) as wartosc
from sprzedaz sp, sklepy sk, produkty pr, czas cz
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=pr.produkt_id
and sp.data=cz.data
group by sk.miasto, pr.produkt_id, cz.nazwa_miesiaca;
```

Join-back (2)

```
select sk.miasto, pr.prod_nazwa, sum(sp.wartosc) wartosc
from sprzedaz sp, sklepy sk, produkty pr, czas cz
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=pr.produkt_id
and sp.data=cz.data
group by sk.miasto, pr.prod_nazwa, cz.nazwa_miesiaca;
```

- **zależność funkcyjna `PRODUKT_ID` → `PROD_NAZWA`**
 - wyznaczona na podstawie klucza

⇒ Uwaga do Oracle9i

- technika join-back zostanie wykorzystana jeśli w klauzuli **FROM** zapytania perspektywy i użytkownika występują identyczne zbiory tabel

```
create materialized view mv_suma_sprzedazy1 ...
select ...
from sprzedaz sp, produkty pr ...
```

```
select ...
from sprzedaz sp, produkty pr, sklepy sk ...
```

Filtrowanie (1)

```
create materialized view mv_sprzedaz4
build immediate
refresh complete
enable query rewrite
as
select sk.miasto, pr.prod_nazwa, cz.nazwa_miesiaca,
       sum(sp.l_sztuk) as sprzedano, sum(sp.wartosc) as wartosc
from sprzedaz sp, sklepy sk, produkty pr, czas cz
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=pr.produkt_id
and sp.data=cz.data
group by sk.miasto, pr.prod_nazwa, cz.nazwa_miesiaca;
```

```
select sk.miasto, pr.prod_nazwa, sum(sp.wartosc) wartosc
from sprzedaz sp, sklepy sk, produkty pr, czas cz
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=pr.produkt_id
and sp.data=cz.data
and sk.miasto='Poznań'
having sum(sp.wartosc)>190
group by sk.miasto, pr.prod_nazwa, cz.nazwa_miesiaca;
```

Filtrowanie (2)

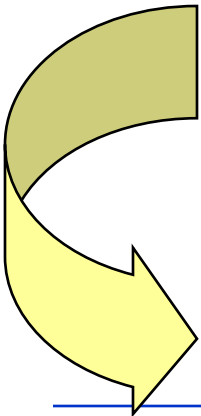
- Wyliczanie AVG na podstawie SUM i COUNT

```
create materialized view mv_sprzedaz4
build immediate
refresh complete
enable query rewrite
as
select sk.miasto, pr.prod_nazwa, cz.nazwa_miesiaca,
       sum(sp.l_sztuk) as sprzedano, sum(sp.wartosc) as wartosc,
       count(sp.wartosc) as ilosc
from sprzedaz sp, sklepy sk, produkty pr, czas cz
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=pr.produkt_id
and sp.data=cz.data
group by sk.miasto, pr.prod_nazwa, cz.nazwa_miesiaca;
```

```
select sk.miasto, pr.prod_nazwa, avg(sp.wartosc) wartosc
from sprzedaz sp, sklepy sk, produkty pr, czas cz
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=pr.produkt_id
and sp.data=cz.data
group by sk.miasto, pr.prod_nazwa, cz.nazwa_miesiaca;
```

Wykorzystanie DIMENSION (1)

```
create materialized view mv_sprzedaz6
refresh complete
enable query rewrite
as
select sk.miasto, pr.prod_nazwa, cz.nazwa_miesiaca,
       sum(sp.l_sztuk) as sprzedano, sum(sp.wartosc) as wartosc
from sprzedaz sp, sklepy sk, produkty pr, czas cz
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=pr.produkt_id
and sp.data=cz.data
group by sk.miasto, pr.prod_nazwa, cz.nazwa_miesiaca;
```



```
select sk.wojewodztwo, pr.prod_nazwa, sum(sp.wartosc) wartosc
from sprzedaz sp, sklepy sk, produkty pr, czas cz
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=pr.produkt_id
and sp.data=cz.data
and sk.wojewodztwo='wielkopolskie'
group by sk.wojewodztwo, pr.prod_nazwa, cz.nazwa_miesiaca;
```

wynik wyznaczony w oparciu o tabele wskazane zapytaniem

Wykorzystanie DIMENSION (2)

⇒ Utworzenie wymiaru **d_sklepy**

```
create dimension d_sklepy
  level l_sklep is sklepy.sklep_id
  level l_miasto is sklepy.miasto
  level l_wojewodztwo is sklepy.wojewodztwo
hierarchy h_sklepy
  (l_sklep child of l_miasto child of l_wojewodztwo);
```

- ⇒ Optymalizator wykorzysta zależność hierarchczną:
- **miasto → wojewodztwo**

Wykorzystanie DIMENSION (3)

```
create materialized view mv_suma_sprzedazy7
build immediate
refresh force on commit
enable query rewrite
as
select pr.prod_nazwa, cz.nr_miesiaca, sum(wartosc)
from sprzedaz sp, produkty pr, czas cz
where sp.produkt_id=pr.produkt_id
and sp.data=cz.data
group by pr.prod_nazwa, cz.nr_miesiaca;
```

➔ Optymalizator wykorzysta zależność hierarchiczną:

▪ miesiąc → kwartał → rok

```
create dimension d_czas
level 1_data is czas.data
level 1_miesiac is czas.nr_miesiaca
level 1_kwartal is czas.nr_kwartalu
level 1_rok is czas.rok
hierarchy czas_hier (
  1_data child of
  1_miesiac child of
  1_kwartal child of
  1_rok);
```

```
select pr.prod_nazwa, cz.rok, sum(wartosc)
from sprzedaz sp, produkty pr, czas cz
where sp.produkt_id=pr.produkt_id
and sp.data=cz.data
group by pr.prod_nazwa, cz.rok;
```

Wykorzystanie DIMENSION (4)

Execution Plan

```
-----
0      SELECT STATEMENT Optimizer=CHOOSE (Cost=13 Card=2 Bytes=92)
1      0      SORT (GROUP BY) (Cost=13 Card=2 Bytes=92)
2      1      HASH JOIN (Cost=8 Card=682 Bytes=31372)
3      2      TABLE ACCESS (FULL) OF 'MV_SUMA_SPRZEDAZY7' (Cost=2 Card=5
                                                Bytes=100)

4      2      VIEW (Cost=5 Card=409 Bytes=10634)
5      4      SORT (UNIQUE) (Cost=5 Card=409 Bytes=10634)
6      5      TABLE ACCESS (FULL) OF 'CZAS' (Cost=2 Card=409
                                                Bytes=10634)
```

Wykorzystanie DIMENSION (5)

- Wykorzystanie zależności funkcyjnych między atrybutami tabeli, zdefiniowanych w wymiarze

```
create table sklepy
(sklep_id number(3),
nazwa varchar2(20) not null,
rodzaj_sklepu varchar2(10) not null,
miasto varchar2(15) not null,
wojewodztwo varchar2(30) not null);
```

```
create materialized view
      mv_suma_sprzedazy6
build immediate
refresh force on commit
enable query rewrite
as
```

```
select sk.sklep_id, sum(l_sztuk*cena_jedn), count(l_sztuk*cena_jedn)
from sprzedaz sp, sklepy sk
where sp.sklep_id=sk.sklep_id
group by sk.sklep_id;
```

```
create dimension SKLEPY
  level l_sklep is sklepy.sklep_id
  level l_miasto is sklepy.miasto
  level l_wojew is sklepy.wojewodztwo
HIERARCHY sklepy_hier (
  l_sklep child of
  l_miasto child of
  l_wojew)
ATTRIBUTE l_sklep
  DETERMINES (rodzaj_sklepu);
```

Wykorzystanie DIMENSION (6)

```
select sk.rodzaj_sklepu, sum(l_sztuk*cena_jedn)
from sprzedaz sp, sklepy sk
where sp.sklep_id=sk.sklep_id
group by sk.rodzaj_sklepu;
```

Execution Plan

```
-----
0      SELECT STATEMENT Optimizer=CHOOSE (Cost=6 Card=2 Bytes=30)
1      0      SORT (GROUP BY) (Cost=6 Card=2 Bytes=30)
2      1      NESTED LOOPS (Cost=4 Card=2 Bytes=30)
3      2      TABLE ACCESS (FULL) OF 'MV_SUMA_SPRZEDAZY6' (Cost=2
                                Card=2 Bytes=8)
4      2      TABLE ACCESS (BY INDEX ROWID) OF 'SKLEPY' (Cost=1
                                Card=1 Bytes=11)
5      4      INDEX (UNIQUE SCAN) OF 'SKLEPY_PK' (UNIQUE)
```

- **Uwaga: jeżeli tabela SPRZEDAZ będzie posiadała klucz podstawowy, to powyższe zapytanie zostanie również przepisane nawet jeśli nie zdefiniowano zależności funkcyjnych w wymiarze**

QUERY_REWRITE_INTEGRITY (1)

⇒ **Zależności hierarchiczne wymiaru (DIMENSIONS) zostaną wykorzystane jeśli:**

- parametr konfiguracyjny
 - **QUERY_REWRITE_INTEGRITY = TRUSTED** lub
 - **QUERY_REWRITE_INTEGRITY = STALE_TOLERATED**

⇒ **Wartość parametru ustalana**

- na poziomie instancji (init.ora, spfile.ora)
- na poziomie sesji

```
alter system  
set QUERY_REWRITE_INTEGRITY=TRUSTED;
```

```
alter session  
set QUERY_REWRITE_INTEGRITY=TRUSTED;
```

QUERY_REWRITE_INTEGRITY (2)

- Parametr określający jakie perspektywy zmaterializowane, ograniczenia integralnościowe i obiekty typu DIMENSION zostaną wykorzystane do przepisywania zapytań
- Dopuszczalne wartości dla QUERY_REWRITE_INTEGRITY
 - **ENFORCED** (wartość domyślna)
 - Oracle zapewnia, że **wynik** zapytania przepisanego będzie **identyczny** z wynikiem oryginalnego zapytania na tabelach bazowych
 - sprawdza poprawność danych weryfikując za każdym razem ograniczenia integralnościowe PK-FK (w trybie VALIDATE)
 - **nie wykorzystuje** obiektów typu DIMENSION
 - **STALE_TOLERATED**
 - Oracle wykorzysta perspektywy zmaterializowane, których zawartość albo **nie jest aktualna** albo **nie odzwierciedla** (zagregowanych) danych z tabel bazowych (dla perspektyw modyfikowalnych)

QUERY_REWRITE_INTEGRITY (3)

- **TRUSTED** (zalecany w hurtowniach danych) pod warunkiem, że integralność danych zostanie zapewniona przez oprogramowanie zarządzające magazynem (aplikacje, proces ETL)
 - Oracle wykorzysta:
 - obiekty typu **DIMENSION**
 - ograniczenia integralnościowe (PK - FK) w trybie **NOVALIDATE** oznaczone jako poprawne → **RELY**
- ```
ALTER TABLE nazwa MODIFY CONSTRAINT nazwa_ogr RELY;
```
- perspektywy zbudowane w oparciu o istniejące tabele (ON PREBUILT TABLE)
  - tylko te perspektywy zmaterializowane, które zawierają dane aktualne (USER\_MVIEWS.STALENESS)



# QUERY\_REWRITE\_INTEGRITY (4)

- **Uwaga:** jeżeli zapytanie nie zostanie przepisane w trybie **STALE\_TOLERATED**, to w innych trybach na pewno nie zostanie przepisane
- Określenie aktualności danych zmaterializowanej perspektywy
  - **USER\_MVIEWS.STALENESS** → wartości:
    - **FRESH** → perspektywa zawiera dane aktualne
    - **NEEDS\_COMPILE (STALE)** → zawartość tabel bazowych została zmieniona
      - po zatwierdzeniu transakcji modyfikującej tabele bazowe wartość **STALENESS** jest automatycznie zmieniana na **NEEDS\_COMPILE**
      - po odświeżeniu perspektywy wartość automatycznie ustawiana na **FRESH**

# Procedura EXPLAIN\_REWRITE

- Analizuje możliwość przepisania zapytania w oparciu o wskazaną perspektywę zmaterializowaną
- Wynik analizy w tabeli **REWRITE\_TABLE**
  - REWRITE\_TABLE utworzyć w schemacie użytkownika → skrypt ORACLE\_HOME\rdbms\admin\utlxrw.sql

```
BEGIN
DBMS_MVIEW.EXPLAIN_REWRITE
('select sk.nazwa, pr.prod_nazwa, sum(l_sztuk*cena_jedn),
count(l_sztuk)
from sprzedaz sp, sklepy sk, produkty pr
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=pr.produkt_id
group by GROUPING SETS ((sk.nazwa, pr.prod_nazwa))',
'MV_SUMA_SPRZEDAZY7');
END;
```

```
select mv_name, message from rewrite_table;
```



# Procedura TUNE\_MVIEW

---

- Analizuje definicję perspektywy pod kątem:
  - możliwości odświeżania przyrostowego
- Procedura **DBMS\_ADVISOR.TUNE\_MVIEW**
- Wynik dostępny przez perspektywę **USER\_TUNE\_MVIEW**



# Warunki niezbędne do przepisania zapytania - podsumowanie

- **Uprawnienia użytkownika:**
  - uprawnienie obiektowe **QUERY REWRITE** na wszystkich tabelach, do których odwołuje się perspektywa zmaterializowana
  - lub uprawnienie systemowe **GLOBAL QUERY REWRITE**
- **Włączony mechanizm przepisывania zapytań dla perspektywy zmaterializowanej**
  - klauzula **ENABLE QUERY REWRITE**
  - lub włączenie dynamicznie (`alter materialized view`)
- **Włączony mechanizm przepisывania w systemie**
  - parametr konfiguracyjny **QUERY\_REWRITE\_ENABLED=TRUE**
  - lub włączenie dynamicznie dla sesji (`alter session set query_rewrite_enabled=true`)
- **Wykorzystywany optymalizator kosztowy (9i)**
  - **OPTIMIZER\_MODE=CHOOSE** (parametr konfiguracyjny, ustawienie dla sesji)
  - dla tabel bazowych i perspektyw zmaterializowanych zebrano statystyki
- **QUERY\_REWRITE\_INTEGRITY= ENFORCED | STALE\_TOLERATED | TRUSTED** (parametr konfiguracyjny, ustawienie dla sesji)



# Warunki niezbędne do przepisania zapytania - podsumowanie

---

- ➔ **Uwaga: perspektywa z klauzulą NEXT (okres odświeżania) może nie być brana pod uwagę przy przepisywaniu, nawet jeśli wszystkie wcześniejsze warunki zostały spełnione**
  - **rozwiązanie: pominąć NEXT i dodać REFRESH ON DEMAND**

# Problematyka

---

- ➔ **Wyznaczenie właściwego zbioru perspektyw zmaterializowanych**
  - liczba zapytań w systemie wykorzystujących takie perspektywy
  - koszty odświeżania perspektyw
- ➔ **Problem trudny**
  - wiele prac badawczych (złożone algorytmy)
- ➔ **Wsparcie ze strony oprogramowania systemowego**
  - Oracle9i Summary Advisor
  - Oracle10g Access Advisor



# Access Advisor

## ➤ Rekomenduje:

- zbiory zmaterializowanych perspektyw
- zbiory indeksów

## ➤ Wymagane uprawnienie **ADVISOR**

## ➤ Interface:

- pakiet **DBMS\_ADVISOR**
- Enterprise Manager

## ➤ Wykorzystanie:

- utworzenie obiektu zadania (task) → **CREATE\_TASK**
- utworzenie obiektu obciążenia (workload) → **CREATE\_SQLWKLD**
- powiązanie zadania z obciążeniem → **ADD\_SQLWKLD\_REF**
- określenie zawartości obciążenia → **IMPORT\_SQLWKLD\_...**
- wykonanie zadania → **EXECUTE\_TASK**
- wygenerowanie skryptu sql → **GET\_TASK\_SCRIPT**

# Access Advisor - zadanie

```
DBMS_ADVISOR.CREATE_TASK (
 advisor_name IN VARCHAR2 NOT NULL,
 task_id OUT NUMBER,
 task_name IN OUT VARCHAR2,
 task_desc IN VARCHAR2 := NULL,
 task_or_template IN VARCHAR2 := NULL,
 is_template IN VARCHAR2 := 'FALSE')
```

```
v_task_name := 'T_MV1';
DBMS_ADVISOR.CREATE_TASK (
 DBMS_ADVISOR.SQLACCESS_ADVISOR,
 out_task_id,
 v_task_name,
 'wyznaczenie zbioru MV',
 DBMS_ADVISOR.SQLACCESS_WAREHOUSE,
 'FALSE')
```

⇒ Zbiór wartości dla **advisor\_name** jest dostępny m.in. w perspektywie **DBA\_ADVISOR\_DEFINITIONS**

↓ predefiniowany szablon

```
select * from DBA_ADVISOR_DEFINITIONS;
```

| ADVISOR_ID | ADVISOR_NAME         | PROPERTY |
|------------|----------------------|----------|
| 1          | ADDM                 | 1        |
| 2          | SQL Access Advisor   | 15       |
| 3          | Undo Advisor         | 1        |
| 4          | SQL Tuning Advisor   | 3        |
| 5          | Segment Advisor      | 3        |
| 6          | SQL Workload Manager | 0        |
| 7          | Tune MView           | 31       |

Automatic Database Diagnostic Monitor ⇒ wartości parametrów konf. zwiększające efektywność systemu

czy jest tworzony szablon zadania?

# Parametry zadania

```
DBMS_ADVISOR.SET_TASK_PARAMETER (
 task_name IN VARCHAR2
 parameter IN VARCHAR2,
 value IN VARCHAR2)
```

## ➔ Parameter=EXECUTION\_TYPE

- value=INDEX\_ONLY → rekomendacje dot. indeksów
- value=MVIEW\_ONLY → rekomendacje dot. perspektyw zmaterializowanych
- value=MVIEW\_LOG\_ONLY → rekomendacje dot. dzienników perspektyw
- value=FULL (domyślnie) → wszystkie powyższe rekomendacje

## ➔ Parameter=MODE

- value=LIMITED → rekomendacje na podstawie analizy przybliżonej (redukcja czasu wyznaczenia rekomendacji)
- value=COMPREHENSIVE → rekomendacje na podstawie pełnej analizy

```
DBMS_ADVISOR.SET_TASK_PARAMETER
(v_task_name, 'EXECUTION_TYPE', 'FULL')
```

# Access Advisor - szablon

## ➔ Szablon (template) określa m.in.:

- wzorce nazw indeksów i perspektyw zmaterializowanych
- nazwy przestrzeni tabel dla indeksów i perspektyw zmaterializowanych

```
VARIABLE template_id NUMBER;
VARIABLE template_name VARCHAR2(255);
EXECUTE :template_name := 'MY_TEMPLATE';

EXECUTE DBMS_ADVISOR.CREATE_TASK(-
 'SQL Access Advisor', -
 :template_id, -
 :template_name, -
 is_template => 'TRUE')
```

```
DBMS_ADVISOR.SET_TASK_PARAMETER
(:template_name,
 'INDEX_NAME_TEMPLATE',
 'SH_IDX$$_<SEQ>');
```

```
DBMS_ADVISOR.SET_TASK_PARAMETER
(:template_name,
 'MVVIEW_NAME_TEMPLATE',
 'SH_MV$$_<SEQ>');
```

```
DBMS_ADVISOR.SET_TASK_PARAMETER
(:template_name,
 'DEF_INDEX_TABLESPACE',
 'SH_INDEXES');
```

```
DBMS_ADVISOR.SET_TASK_PARAMETER
(:template_name,
 'DEF_MVIEW_TABLESPACE',
 'SH_MVIEWS')
```

# Access Advisor - obciążenie

```
DBMS_ADVISOR.CREATE_SQLWKLD (
workload_name IN VARCHAR2,
description IN VARCHAR2 := NULL,
template IN VARCHAR2 := NULL,
is_template IN VARCHAR2 := 'FALSE')
```

```
v_workload_name := 'W_MV1';
DBMS_ADVISOR.CREATE_SQLWKLD (
v_workload_name,
'przykładowe obciążenie')
```

## ➔ Powiązanie obiektu obciążenia z obiektem zadania

```
DBMS_ADVISOR.ADD_SQLWKLD_REF (
task_name IN VARCHAR2,
workload_name IN VARCHAR2)
```

```
DBMS_ADVISOR.ADD_SQLWKLD_REF (
v_task_name,
v_workload_name)
```

## ➔ Zdefiniowanie zawartości obciążenia

- obciążenie zdefiniowane przez użytkownika (user defined workload)
- obciążenie z zawartość bufora SQL (SQL cache workload)
- obciążenie na podstawie kodów poleceń (SQL statement workload)

# Obciążenie użytkownika

- Obciążenie jest zapisywane w tabeli **USER\_WORKLOAD** o ściśle określonym schemacie (zob. dokumentacja)
  - tabela zawiera dane z pliku trace (przetworzone tkprof)
- Obciążenie jest importowane i analizowane przez Access Advisor

```
CREATE TABLE user_workload (
 MODULE VARCHAR2(48),
 ACTION VARCHAR2(32),
 BUFFER_GETS NUMBER,
 CPU_TIME NUMBER,
 ELAPSED_TIME NUMBER,
 DISK_READS NUMBER,
 ROWS_PROCESSED NUMBER,
 EXECUTIONS NUMBER,
 OPTIMIZER_COST NUMBER,
 LAST_EXECUTION_DATE DATE,
 PRIORITY NUMBER,
 SQL_TEXT CLOB,
 STAT_PERIOD NUMBER,
 USERNAME VARCHAR2(30));
```

```
DBMS_ADVISOR.IMPORT_SQLWKLD_USER (
 workload_name IN VARCHAR2, ←
 import_mode IN VARCHAR2, ←
 owner_name IN VARCHAR2,
 table_name IN VARCHAR2,
 saved_rows OUT NUMBER,
 failed_rows OUT NUMBER)
```

← obiekt obciążenia

← APPEND  
NEW  
REPLACE

# Zawartość bufora SQL

➔ Obciążenie jest budowane na podstawie zawartości bufora SQL (SQL cache w SHARED POOL)

```
DBMS_ADVISOR.IMPORT_SQLWKLD_SQLCACHE (
workload_name IN VARCHAR2,
import_mode IN VARCHAR2,
priority IN NUMBER := 2,
saved_rows OUT NUMBER,
failed_rows OUT NUMBER)
```

← obiekt obciążenia

←  
←  
← ważność  
aktualnego zbioru  
obciążenia:

APPEND  
NEW  
REPLACE

1-HIGH, 2-MEDIUM, 3-LOW

# Kod polecenia SQL (1)

## ➔ Dodanie pojedynczego polecenia do obciążenia

```
DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT (
workload_name IN VARCHAR2 ,
module IN VARCHAR2 , ← nazwa aplikacji generującej obciążenie
action IN VARCHAR2 , ← operacja w ramach aplikacji
cpu_time IN NUMBER := 0 ,
elapsed_time IN NUMBER := 0 ,
disk_reads IN NUMBER := 0 ,
buffer_gets IN NUMBER := 0 ,
rows_processed IN NUMBER := 0 ,
optimizer_cost IN NUMBER := 0 ,
executions IN NUMBER := 1 ,
priority IN NUMBER := 2 ,
last_execution_date IN DATE := 'SYSDATE' ,
stat_period IN NUMBER := 0 , ← okres czasowy zbierania statystyk
username IN VARCHAR2 , ← użytkownik wykonujący analizowane
sql_text IN CLOB) polecenie
```

dane z pliku trace



# Kod polecenia SQL (2)

```
DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT (
 workload_name=>v_workload_name,
 username=>'USER_STAR',
 sql_text=>'select pr.prod_nazwa, sum(sp.wartosc) wartosc
 from sprzedaz sp, produkty pr
 where sp.produkt_id=pr.produkt_id
 group by pr.prod_nazwa')
```



# Wykonanie zadania

```
DBMS_ADVISOR.EXECUTE_TASK
(task_name IN VARCHAR2)
```

```
DBMS_ADVISOR.EXECUTE_TASK
(v_task_name)
```

➔ **Wynik wykonania dostępny przez**

- **perspektywę USER\_ADVISOR\_ACTIONS**
- **odczytywana zazwyczaj procedurą**  
**queryrewrite\create\_show\_recom.sql**

# Wyświetlanie rekomendacji

## ➤ Zapis do pliku skryptu tworzącego rekomendowane obiekty

```
DBMS_ADVISOR.CREATE_FILE(dbms_advisor.get_task_script(v_task_name), -
 'ADVISOR_DIR', - ←
 'plik_wynikowy.sql');
```

obiekt typu DIRECTORY

- demo queryrewrite\access\_advisor.sql

## ➤ Wydruk rekomendacji na ekranie w postaci opisowej

- demo queryrewrite\create\_show\_recom.sql
- queryrewrite\show\_recommendations.sql

## ➤ Uwaga: przed uruchomieniem ADVISOR'a należy zebrać statystyki przynajmniej dla tabel

- brak statystyk powoduje, że ADVISOR nie podaje podpowiedzi

# Quick Tune

## ➤ Wyświetla rekomendacje dla pojedynczego polecenia

```
BEGIN
 DBMS_ADVISOR.QUICK_TUNE (
 advisor_name => DBMS_ADVISOR.SQLACCESS_ADVISOR,
 task_name => 'zadanie1',
 attr1 => 'select sk.miasto, pr.prod_nazwa, cz.nazwa_miesiaca,
 sum(sp.l_sztuk) as sprzedano, sum(sp.wartosc) as wartosc
 from sprzedaz sp, sklepy sk, produkty pr, czas cz
 where sp.sklep_id=sk.sklep_id
 and sp.produkt_id=pr.produkt_id
 and sp.data=cz.data
 group by sk.miasto, pr.prod_nazwa, cz.nazwa_miesiaca');
END;
/
```

```
EXECUTE show_recm('zadanie1');
```

# Journaling

- Rejestrowanie komunikatów z pracy Access Advisor, na różnym poziomie szczegółowości
- Parametr **JOURNALING**
  - wartość 0-4
- Informacje dostępne za pomocą perspektywy **USER\_ADVISOR\_SQLW\_JOURNAL**
- Wyłączenie rejestrowania

```
DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(v_workload_name, 'JOURNALING', 0);
```

- Włączenie rejestrowania

```
DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(v_workload_name, 'JOURNALING', 4);
```