Na początku połączyłem się z bazą jako *system/manager* i wykonałem polecenia tworzące perspektywę i partycje:

```
ALTER SESSION SET CURRENT_SCHEMA = SH;
@rdbms/admin/utlxrw.sql
@costs_promo_mv.sql
@partitions.sql
```

Następnie sprawdziłem, czy wszystko działa poprzez wykonanie zapytania

```
SELECT p.promo_subcategory, sum(unit_cost)
FROM costs c, promotions p
WHERE c.promo_id = p.promo_id
GROUP BY promo_subcategory;
```

Teraz już mogłem przystąpić do sprawdzenia, na co zostało przepisane to zapytanie. Zrobiłem to w ten sposób:

```
BEGIN
DBMS_MVIEW.EXPLAIN_REWRITE(
'SELECT p.promo_subcategory, sum(unit_cost)
FROM costs c, promotions p
WHERE c.promo_id = p.promo_id
GROUP BY promo_subcategory',
'SH.COSTS_PROMO_MV',
'Z1');
END;
/
SELECT message FROM REWRITE_TABLE;
SELECT rewritten_txt FROM REWRITE_TABLE;
```

Naturalnie musiałem wykonać to polecenie jako użytkownik *SH*, administrator bazy danych na koncie *system* nie ma dostępu do tak zaawansowanych i potencjalnie niebezpiecznych funkcji, które dla podniesienia poziomu bezpieczeństwa wypluwają mu enigmatyczny komunikat o nieistnieniu tabeli... W wyniku otrzymałem optymistycznie brzmiący komunikat:

```
MESSAGE
----------------------------------------------------------------
QSM-01151: query was rewritten
QSM-01033: query rewritten with materialized view, COSTS_PROMO_MV

REWRITTEN_TXT
----------------------------------------------------------------
SELECT COSTS_PROMO_MV.PROMO_SUBCATEGORY PROMO_SUBCATEGORY,
SUM(COSTS_PROMO_MV.SUM_UNITS) SUM(UNIT_COST)
FROM SH.COSTS_PROMO_MV COSTS_PROMO_MV
GROUP BY COSTS_PROMO_MV.PROMO_SUBCATEGORY
```

Tytułem wyjaśnienia, nasz widok przechowuje zagregowane koszty jednostkowe z poszczególnych dni. Nie ma powodu, by przy wykonaniu zapytania znowu sumować wszystkie pojedyńcze koszty, skoro można się mniej napracować i zsumować już gotowe wyniki z kolejnych dni. Zapewne optymalizator zastosował takie rozumowanie przy układaniu planu. Zgodnie z kolejnym poleceniem usunąłem wszystkie wiersze z partycji COST_Q1_2004. Usunięto ok. 82 tysięcy wierszy. Od tej chwili rewrite nie następuje z uwagi na wykonanie operacji DML na tabeli:

```
SM-01150: query did not rewrite
QSM-01279: query rewrite not possible because DML operation occurred on a table
referenced by materialized view COSTS_PROMO_MV
```

Naprawienie tej sytuacji jest proste - trzeba zaktualizować widok. Aby taka sytuacja nie zachodziła w przyszłości warto zmienić tryb odświeżania widoku z „REFRESH FAST ON DEMAND" na „REFRESH FAST ON COMMIT". Ja ograniczyłem się do wywołania

```
EXEC DBMS_MVIEW.refresh('COSTS_PROMO_MV', 'f')
```

Wykonanie EXPLAIN_REWRITE potwierdziło, że wszystko znowu działa. Mogłem przejść do dalszej części zadania:

```
ALTER SESSION SET QUERY_REWRITE_INTEGRITY = TRUSTED;
EXEC DBMS_MVIEW.EXPLAIN_REWRITE(
'SELECT p.promo_category, sum(unit_cost)
FROM costs c, promotions p
WHERE c.promo_id = p.promo_id
AND c.time_id >= TO_DATE('''01-JAN-2000''','''DD-MON-YYYY''')
AND c.time_id < TO_DATE('''01-JAN-2001''','''DD-MON-YYYY''')
GROUP BY promo_category',
'SH.COSTS_PROMO_MV',
'Z2')
SELECT message FROM REWRITE_TABLE WHERE statement_id = 'Z2';
```

Przepisanie nie zachodzi, wyjaśnienie jest takie:

```
QSM-01150: query did not rewrite
QSM-01082: Joining materialized view, COSTS_PROMO_MV, with table, PROMOTIONS,
not possible
QSM-01102: materialized view, COSTS_PROMO_MV, requires join back to table,
PROMOTIONS, on column, PROMO_CATEGORY
QSM-01053: NORELY referential integrity constraint on table, PROMOTIONS, in
TRUSTED/STALE TOLERATED integrity mode
```

Pierwszym zasadniczym pytaniem jest, czemu tutaj następuje jakikolwiek join? Przecież wszystkie potrzebne informacje są zawarte w widoku COSTS_PROMO_MV! Po dokładniejszym przyjrzeniu się definicjom obiektów okazuje się, że tabela PROMOTIONS jest potrzebna do konsolidacji podkategorii w kategorie. Baza danych ma mgliste pojęcie o zależności między kategoriami

i podkategoriami dzięki informacji zawartej w obiekcie PROMOTIONS_DIM, jednak z widoku nie może wyczytać konkretnych wartości. Postanowiłem odrobinę wyklarować sytuację poprzez utworzenie kolejnego obiektu:

```
CREATE DIMENSION dim_specjal
  LEVEL subcategory IS (promotions.promo_subcategory)
  LEVEL category IS (promotions.promo_category)
  HIERARCHY roll (subcategory CHILD OF category)
```

Wygenerowało to następujący efekt:

```
QSM-01102: materialized view, COSTS_PROMO_MV, requires join back to table,
PROMOTIONS, on column, PROMO_CATEGORY
QSM-01151: query was rewritten
QSM-01110: query rewrite not possible with materialized view COSTS_PROMO_MV
because it contains a join between tables (PROMOTIONS and COSTS) that is not present
in the query and that potentially eliminates rows needed by the query
QSM-01033: query rewritten with materialized view, COSTS_PROMO_MV
```

Jak widać mamy demokrację, planer jest za, a nawet przeciw. Jestem skłonny uznać, że stosunkiem 2:1 wygrała opcja mówiąca, że rewrite jednak się odbyło. Świadczą o tym wygenerowane zapytania:

```
SELECT DISTINCT PROMOTIONS.PROMO_SUBCATEGORY PROMO_SUBCATEGORY,PROMOTIONS.PROMO_
CATEGORY PROMO_CATEGORY FROM PROMOTIONS PROMOTIONS GROUP BY PROMOTIONS.PROMO_SUB
CATEGORY,PROMOTIONS.PROMO_CATEGORY
```

```
SELECT from$_subquery$_005.PROMO_CATEGORY PROMO_CATEGORY,SUM(COSTS_PROMO_MV.SUM_
UNITS) SUM(UNIT_COST) FROM SH.COSTS_PROMO_MV COSTS_PROMO_MV, (SELECT DISTINCT PR
OMOTIONS.PROMO_SUBCATEGORY PROMO_SUBCATEGORY,PROMOTIONS.PROMO_CATEGORY PROMO_CAT
EGORY FROM PROMOTIONS PROMOTIONS) from$_subquery$_005 WHERE from$_subquery$_005.
PROMO_SUBCATEGORY=COSTS_PROMO_MV.PROMO_SUBCATEGORY AND COSTS_PROMO_MV.TIME_ID>=T
O_DATE("01-JAN-2000","DD-MON-YYYY") AND COSTS_PROMO_MV.TIME_ID<TO_DATE("01-JAN-2
001","DD-MON-YYYY") GROUP BY from$_subquery$_005.PROMO_CATEGORY
```

Planer (albo „przepisywaczka zapytań”) wydedukował sposób generacji kategorii z subkategorii, a potem użył go do przepisania. W moim odczuciu wygenerowane zapytanie jest całkiem sensowne i jest cień szansy, że realizuje zamierzenie pierwotnego zapytania. Innym sposobem na osiągnięcie takiego efektu byłoby wprowadzenie równoważności między id kategorii (subkategorii), a jej nazwą poprzez wprowadzenie dodatkowych dwóch atrybutów do PROMOTIONS_DIM. Zrezygnowałem z takiego rozwiązania, gdyż o ile przy danych w przykładowej bazie danych jest to prawdą, o tyle w ogólności takie zjawisko może nie mieć miejsca.