8. Vizualizări materializate și tehnici de rescriere a cererilor

Cuprins

8. Vizualizări materializate și tehnici de rescriere a cererilor

- Sistemul *Oracle* utilizează rescrierea cererilor ca tehnică de optimizare, folosind pentru execuția cererilor vizualizările materializate în locul tabelelor de bază specificate în cerere.
  - Această operație este transparentă aplicațiilor și utilizatorilor.

8.1. Vizualizări materializate

- O vizualizare materializată, cunoscută în versiunile anterioare sub numele de *snapshot*, este un obiect al schemei ce stochează rezultatele unei cereri și care este folosit pentru a rezuma, calcula, replica și distribui date.
- Din mai multe puncte de vedere vizualizarea materializată se comportă asemenea unui index:
  - scopul vizualizării materializate este de a mări performanța la execuție a cererilor;
  - existența unei vizualizări materializate este transparentă aplicațiilor *SQL*, astfel că administratorul poate crea sau șterge vizualizări materializate fără să afecteze aplicațiile;
  - o vizualizare materializată necesită spațiu de stocare;
  - conținutul vizualizărilor materializate trebuie să fie actualizat atunci când tabele de bază sunt modificate.

Avantajele utilizării vizualizărilor materializate

- Elimină costurile mari de calcul al operațiilor de *join* sau al agregatelor pentru multe clase importante de cereri.
- Pot fi utilizate pentru a replica date.
- Permit rescrierea cererilor.

Restricții pentru cererile care definesc o vizualizare materializată

- Nu pot conține pseudo-coloana *ROWNUM* sau funcția *SYSDATE*.
- Nu pot referi obiecte de tip *RAW*, *LONG RAW* sau *REF*.
- Nu pot conține operatori pe mulțimi (*UNION*, *MINUS*, *INTERSECT*).

Numele

- Dacă vizualizarea materializată este bazată pe o tabelă declarată *prebuilt*, atunci numele acesteia trebuie să fie aceelași cu numele tabelei.

Caracteristici de stocare

- Dacă vizualizarea nu este creată pe baza unei tabele *prebuilt*, atunci aceasta va necesita spațiu de stocare în baza de date.
- Dacă nu se cunoaște cât spațiu de stocare necesită vizualizarea, atunci se poate utiliza procedura *ESTIMATE_SIZE* din pachetul *DBMS_OLAP*, care permite estimarea spațiului de stocare necesar.

Metode de populare cu date

Metoda	**Descriere**
BUILD IMMEDIATE	Se creează vizualizarea și se populează imediat cu date.
BUILD DEFERRED	Se creează vizualizarea, dar fără să conțină date. Popularea ulterioară cu date se poate realiza utilizând procedura *REFRESH* din pachetul *DBMS_MVIEW*.

Momentul actualizării

Mod	**Descriere**
ON COMMIT	Indică declanșarea unei operații de reactualizare ori de câte ori sistemul permanentizează o tranzacție care operează asupra unei tabele de bază a vizualizării materializate. Aceasta ar putea determina creșterea timpului de execuție a operației *COMMIT*, întrucât reactualizarea va face parte din acest proces.
ON DEMAND	Este opțiune implicită și indică efectuarea reactualizării vizualizării materializate la cererea utilizatorului, prin intermediul procedurilor specifice din pachetul *DBMS_MVIEW* (*REFRESH*, *REFRESH_ALL_ MVIEWS*, *REFRESH_DEPENDENT*).

Modalități de actualizare

Opțiuni	Descriere
COMPLETE	Implică reactualizarea completă, care se realizează prin rularea cererii din definiția vizualizării materializate.
FAST	Indică metoda de reactualizare incrementală, care se efectuează corespunzător modificărilor survenite în tabelele de bază. Modificările sunt stocate într-un fișier *log* (jurnal) asociat tabelei de bază.
FORCE	Determină reactualizarea de tip *FAST*, dacă este posibil, în caz contrar actualizarea de tip *COMPLETE*.
NEVER	Indică faptul că vizualizarea nu va fi reactualizată.

- Procedura *DBMS_MVIEW.EXPLAIN_MVIEW* indică dacă o vizualizare materializată poate fi actualizată de tip *FAST* sau nu.

Rescrierea cererilor

- Înainte de a defini o vizualizare se pot afla, utilizând procedura *DBMS_MVIEW.EXPLAIN_MVIEW*, ce tipuri de rescriere a cererilor va permite aceasta. După ce vizualizarea a fost creată se poate afla, cu ajutorul procedurii *DBMS_MVIEW.EXPLAIN_REWRITE*, de ce aceasta va rescrie o cerere sau nu.
- O vizualizare materializată poate fi utilizată pentru rescrierea cererilor dacă în definiția sa este specificată clauza *ENABLE QUERY REWRITE*.

Clauza *ORDER BY*

- Clauza *ORDER BY* a comenzii *SELECT* care definește vizualizarea este utilizată doar la momentul creării acesteia.
  - Nu va fi utilizată în timpul operațiilor de actualizare a vizualizării.
  - Această clauză nu este considerată parte din definiția vizualizării.

Log-uri pentru vizualizările materializate

- Pentru a putea realiza actualizări de tip *FAST* sunt necesare fișiere *log* definite pentru tabele de bază.
- Acestea se creează prin comanda:

```
CREATE MATERIALIZED VIEW LOG ON nume_tabelă
  [ WITH { OBJECT ID
         | PRIMARY KEY
```

```
        | ROWID
        | SEQUENCE
        | (coloană [,coloană ]...)
        }
          [, { OBJECT ID
             | PRIMARY KEY
             | ROWID
             | SEQUENCE
             | (coloană [,coloană ]...)
             }
          ]...
      { INCLUDING | EXCLUDING } NEW VALUES
  ] ;
```

- Clauza *WITH* este utilizată pentru a indica dacă în fișierul *log* se înregistrează valorile pentru cheia primară, *rowid*-ul, *id*-ul obiectului sau o combinație a acestor identificatori atunci când o înregistrare din tabela de bază este modificată. De asemenea, se poate utiliza clauza *SEQUENCE* pentru a adăuga valorile unei secvențe în fișierul *log*, cu scopul de a menține informații despre ordonarea înregistrărilor.
- În fișierul *log* se pot înregistra valori ale coloanelor care nu sunt chei primare: coloane utilizate în filtre sau coloane de *join*.
- Clauza *NEW VALUES* determină dacă sunt salvate în fișierele *log* atât valorile vechi, cât și valorile noi ale operațiilor *UPDATE*.
- În plus de *ROWID*-uri, pentru vizualizările materializate care conțin agregate, trebuie incluse și clauzele *INCLUDING NEW VALUES* și *SEQUENCE*. Se recomandă utilizarea opțiunii *SEQUENCE* cu excepția cazurilor în care există siguranța că niciodată nu se vor folosi combinații de operații *LMD* pe mai multe tabele.

- Fișierul *log* este:
  - necesar pentru a realiza mai rapid operațiile de *refresh*;
  - creat implicit cu opțiunea *WITH PRIMARY KEY*;
  - localizat în aceeași schemă ca și tabela de bază;
  - populat cu ajutorul unui *trigger* creat automat asupra tabelei de bază;
  - creat asupra tabelei de bază (pentru o tabelă se poate defini un singur fișier *log*);
  - denumit implicit de sistem: *MLOG$_nume_tabelă*.

Exemplul 8.1

a. Implicit, fișierul *log* menține valorile cheii primare.

```
CREATE MATERIALIZED VIEW LOG ON produse;
```

b. Fișierul *log* menține valorile cheii primare și *rowid*-urile înregistrărilor.

```
CREATE MATERIALIZED VIEW LOG ON produse
WITH PRIMARY KEY, ROWID;
```

c. Fișierul *log*-ul determină actualizarea de tip *fast* a vizualizărilor materializate care conțin coloanele *factura*, *produs_id*, *timp_id*, *cantitate*, *pret_unitar_vanzare*.

```
CREATE MATERIALIZED VIEW LOG ON vanzari
WITH PRIMARY KEY,SEQUENCE
     (factura, produs_id, timp_id,
      cantitate, pret_unitar_vanzare)
INCLUDING NEW VALUES;
```

Comanda CREATE MATERIALIZED VIEW

- O vizualizare materializată se creează prin intermediul comenzii *CREATE MATERIALIZED VIEW*, care are următoarea formă sintactică:

```
CREATE MATERIALIZED VIEW [schema.]nume_viz_mat
[OF [schema.]tip_obiect] [ (constr_ref_domeniu) ]
[ORGANIZATION INDEX clauza_tabelă_org_index]
[{proprietăți_vm | ON PREBUILT TABLE
             [{WITH | WITHOUT} REDUCED PRECISION]}]
[{USING INDEX
 [{clauza_atribute_fizice | TABLESPACE nume_sp_tab}
 [{clauza_atribute_fizice | TABLESPACE nume_sp_tab}]]
| USING NO INDEX}]
[refresh_vm]
[FOR UPDATE]
[{DISABLE | ENABLE} QUERY REWRITE] AS subcerere;
```

  - Clauza *OF* [*schema.*]*tip_obiect* permite crearea unei vizualizări materializate obiect.

  - Sintaxa clauzei *constr_ref_domeniu* este următoarea:

```
SCOPE FOR ( {ref_coloana | ref_atribut} )
IS [schema.]nume_tabelă_scope
[, SCOPE FOR ( {ref_coloana | ref_atribut} )
   IS [schema.]nume_tabelă_scope] …
```

  - Clauza poate fi utilizată pentru restricționarea domeniului referințelor la tabela *nume_tabelă_scope*. Valorile dintr-o coloană de tip *REF* vor adresa obiecte din

tabela identificată prin *nume_tabelă_scope*. În această tabelă sunt stocate instanțe de obiecte care au același tip ca și coloana *REF*.

- Opțiunea *ON PREBUILT TABLE* permite considerarea unei tabele existente ca fiind o vizualizare materializată predefinită. Tabela trebuie să aibă același nume și să se afle în aceeași schemă ca vizualizarea materializată rezultată. La ștergerea acestei vizualizări, tabela revine la statutul său inițial. Pentru o vizualizare materializată de acest tip, *alias*-urile de coloană din clauza *subcerere* trebuie să corespundă, ca număr și tip de date, coloanelor din tabelă.

- Clauza *WITH REDUCED PRECISION* permite ca precizia coloanelor tabelei sau vizualizării materializate să nu coincidă cu precizia coloanelor returnate de *subcerere*. Pentru a impune respectarea întocmai a preciziei, sintaxa dispune de opțiunea *WITHOUT REDUCED PRECISION*, care este implicită.

- Atributele fizice au o semantică asemănătoare celei descrise de *clauza_proprietăți_fizice* din cadrul comenzii *CREATE TABLE*. Spre deosebire de tabele, pentru o vizualizare materializată nu poate fi specificată opțiunea *ORGANIZATION EXTERNAL*.

- Clauza *TABLESPACE* specifică *tablespace*-ul în care urmează să fie creată vizualizarea materializată. În absența acesteia, vizualizarea va fi creată în *tablespace*-ul implicit al schemei care o conține.

- Clauza *USING INDEX* permite stabilirea de valori ale parametrilor *INITRANS*, *MAXTRANS* și *STORAGE* ai indexului implicit care este utilizat de sistemul *Oracle* pentru a întreține datele vizualizării materializate. Dacă este omisă clauza, sistemul va utiliza indexul implicit pentru ameliorarea vitezei de reactualizare incrementală a vizualizării materializate. În această clauză nu poate apărea parametrul *PCTFREE*. Pentru a suprima crearea indexului implicit, se poate utiliza opțiunea *USING NO INDEX*. Un index alternativ poate fi creat explicit prin comanda *CREATE INDEX*.

- Clauza *proprietăți_vm* este utilă pentru descrierea vizualizărilor materializate care nu se bazează pe o tabelă existentă (nu sunt construite cu opțiunea *ON PREBUILT TABLE*). Dintre proprietățile care pot fi specificate în această clauză, se menționează: *clauza_partiționare_tabelă*, *clauza_paralelism*, *CACHE* sau

NOCACHE. Acestea sunt similare celor tratate în cadrul comenzii *CREATE TABLE*. Pe lângă acestea, poate fi menționată opțiunea *BUILD IMMEDIATE | DEFERRED* care determină introducerea de înregistrări în vizualizarea materializată imediat, respectiv la prima operație de reactualizare (*refresh*). În acest ultim caz, până la prima operație de reactualizare, vizualizarea nu va putea fi utilizată pentru rescrierea cererilor. Opțiunea *IMMEDIATE* este implicită.

- Prin *refresh_vm* se specifică metodele, modurile și momentele la care sistemul va reactualiza vizualizarea materializată. Sintaxa clauzei este următoarea:

```
{ REFRESH
   [ {FAST | COMPLETE | FORCE} ]
   [ON {DEMAND | COMMIT} ]
   [START WITH data] [NEXT data]
   [WITH {PRIMARY KEY | ROWID} ]
   | USING
     {DEFAULT [ {MASTER | LOCAL} ] ROLLBACK SEGMENT
     | [{MASTER | LOCAL} ]
       ROLLBACK SEGMENT nume_segm_anulare }}
  | NEVER REFRESH}
```

- Clauza *WITH PRIMARY KEY* este implicită și permite ca tabelele de bază să fie reorganizate fără a afecta eligibilitatea vizualizării materializate pentru reactualizarea de tip *FAST*. Tabela de bază trebuie să conțină o constrângere *PRIMARY KEY*. Opțiunea nu poate fi specificată pentru vizualizări materializate obiect.

- Opțiunea *WITH ROWID* asigură compatibilitatea cu tabelele de bază din versiunile precedente lui *Oracle8*.

- Clauza *USING ROLLBACK SEGMENT* specifică segmentul de reluare distant care urmează să fie utilizat pentru reactualizarea vizualizării materializate. Cuvântul cheie *DEFAULT* determină ca sistemul să aleagă acest segment în mod automat. Opțiunile *MASTER* și *LOCAL* specifică segmentul de reluare distant care urmează să fie utilizat pe *site*-ul distant pentru vizualizarea materializată individuală, respectiv pentru grupul local de reactualizare care conține vizualizarea materializată. Opțiunea *LOCAL* este implicită.

- Clauza *FOR UPDATE* permite actualizarea unei vizualizări materializate.

- Opțiunea *AS* specifică cererea care definește vizualizarea materializată. Vizualizările materializate nu pot conține coloane de tip *LONG*. Dacă în clauza *FROM* a cererii din definiția vizualizării materializate se face referință la o altă vizualizare materializată, atunci aceasta va trebui reactualizată întotdeauna înaintea celei create în instrucțiunea curentă.

Modificarea vizualizărilor materializate

- Comanda *ALTER MATERIALIZED VIEW* permite intervenția asupra unei vizualizări materializate, într-unul dintre următoarele moduri:
  - modificarea caracteristicilor de stocare;
  - modificarea metodei, modului sau timpului de reactualizare (*refresh*);
  - modificarea structurii, astfel încât să devină un alt tip de vizualizare materializată;
  - activarea sau dezactivarea funcției de rescriere a cererilor.
- O sintaxă simplificată a acestei comenzi este următoarea:

```
ALTER MATERIALIZED VIEW nume_viz_materializată
 [ {REBUILD | alter_vm_refresh} ]
 [ { {ENABLE | DISABLE} QUERY REWRITE
   | COMPILE | CONSIDER FRESH} ];
```

  - Clauza *REBUILD* permite regenerarea operațiilor de reactualizare atunci când se modifică un tip care este referit în vizualizarea materializată. Specificarea acestei opțiuni interzice utilizarea altor clauze în aceeași instrucțiune *ALTER MATERIALIZED VIEW*.

  - Clauza *alter_vm_refresh* permite modificarea metodei, modului sau perioada de timp la care se va realiza actualizarea. În cazul modificării conținutului tabelelor de bază ale vizualizării materializate, datele din vizualizare trebuie reactualizate astfel încât să reflecte datele existente. Sintaxa clauzei este asemănătoare clauzei *refresh_vm*, prezentată în cadrul instrucțiunii *CREATE MATERIALIZED VIEW*. Spre deosebire de aceasta, clauza nu prevede opțiunea *NEVER REFRESH*.

  - Clauza *QUERY REWRITE*, prin opțiunile *ENABLE* și *DISABLE*, determină ca vizualizarea materializată să fie, sau nu, eligibilă pentru rescrierea cererilor.

  - Clauza *COMPILE* permite revalidarea explicită a vizualizării materializate. Dacă un obiect de care depinde vizualizarea materializată este eliminat sau modificat, atunci vizualizarea rămâne accesibilă, dar nu este eligibilă pentru rescrierea cererilor.

Clauza este utilă pentru revalidarea explicită a vizualizării materializate, astfel încât aceasta să devină eligibilă în operația de rescriere a cererilor.

- Opțiunea *CONSIDER FRESH* indică sistemului să considere vizualizarea materializată ca fiind reactualizată și deci eligibilă pentru rescrierea cererilor.

Eliminarea vizualizărilor materializate

- Pentru a elimina o vizualizare materializată se utilizează instrucțiunea:

```
DROP MATERIALIZED VIEW nume_viz_materializată;
```

- La ștergerea unei tabele de bază, sistemul nu va elimina vizualizările materializate bazate pe acesta.
  - Atunci când se încearcă reactualizarea unei astfel de vizualizări materializate, va fi generată o eroare.

Exemple de intervale la care se poate realiza actualizarea

- `SYSDATE +1`
  - operația de *refresh* se realizează zilnic, la o zi de la ultimul *refresh*
- `NEXT_DAY(TRUNC(SYSDATE),'MONDAY')+15/24`
  - operația de *refresh* se realizează săptămânal, în zilele de Luni la ora 3:00 PM
- `ADD_MONTHS(SYSDATE,3)`
  - operația de *refresh* se realizează la fiecare 3 luni
- `SYSDATE +1/24`
  - operația de *refresh* se realizează din oră în oră

Informații despre vizualizările materializate

- Următoarele vizualizări din dicționarul datelor furnizează informații despre vizualizările materializate.
  - *USER_MVIEWS*
  - *USER_MVIEW_LOGS*
  - *USER_REFRESH*

8.2. Rescrierea cererilor

- Optimizatorul recunoaște în mod automat atunci când o vizualizare materializată poate și ar trebui utilizată pentru a satisface cerința utilizatorului.
  - Optimizatorul rescrie cererea pentru a putea utiliza vizualizarea materializată.
  - Cererile se execută direct pe vizualizarea materializată în loc de tabelele de bază.
  - În general, rescrierea cererilor folosind vizualizări materializate în locul tabelelor de bază îmbunătățește timpul de răspuns.
- În clauza *SELECT* din definiția vizualizării materializate nu se pot specifica tabele distante dacă se dorește menținerea avantajelor rescrierii cererii.
- În concluzie, rescrierea cererilor permite comenzilor *SQL* care sunt exprimate pe tabele sau vizualizări să fie transformate în comenzi *SQL* care să acceseze una sau mai multe vizualizări materializate ce au fost definite asupra tabelelor de bază.
  - Transformarea este transparentă utilizatorilor finali și aplicațiilor (nefiind nevoie de nicio intervenție a acestora).
  - Deoarece rescrierea cererilor este transparentă, vizualizările materializate pot fi adăugate sau șterse la fel ca în cazul indecșilor, fără invalidarea comenzilor *SQL* care le utilizau.

8.2.1 Concepte generale

- Înainte ca cererea să fie rescrisă trebuie realizate câteva verificări pentru a determina dacă aceasta poate fi utilizată în operația de rescriere. Dacă cererea nu trece validările, atunci aceasta se va executa asupra tabelelor de bază.
- Optimizatorul *Oracle* folosește două metode pentru a determina dacă o cerere poate fi rescrisă.
  - Prima metodă constă în compararea comenzii *SQL* candidată pentru rescriere cu comanda *SQL* care apare în definiția vizualizării materializate.
  - Dacă această metodă eșuează, atunci optimizatorul folosește o metodă generală, în care compară *join*-urile, selecțiile, coloanele de date, grupările de coloane și funcțiile agregat ale cererii cu cele ale vizualizării.
- Rescrierea cererilor operează pe:
  - cereri *SELECT*;
  - subcereri care apar în comanda *CREATE TABLE ... AS SELECT*;

  - subcereri care sunt conectate prin operatorii *UNION*, *UNION ALL*, *INTERSECT* sau *MINUS*;
  - subcereri care apar în comenzile *LMD* (*INSERT*, *DELETE*, *UPDATE*).
- Rescrierea sau nu a cererilor este influențată de anumiți factori:
  - activarea sau dezactivarea opțiunii de rescriere a cererilor;
  - folosirea comenzile *CREATE* sau *ALTER* pentru anumite vizualizări materializate;
  - setarea parametrului *QUERY_REWRITE_ENABLED*;
  - folosirea *hint*-urilor *REWRITE* sau *NOREWRITE* în comenzile *SQL*;
  - setarea nivelurilor de integritate ale rescrierii;
  - definirea constrângerilor și a obiectelor *dimension*.
- Sistemul furnizează procedura *EXPLAIN_REWRITE* din pachetul *DBMS_MVIEW.*care informează dacă rescrierea unei cereri este posibilă și ce vizualizare materializată va fi utilizată.
- Atunci când se utilizează rescrierea cererii, trebuie create vizualizări materializate care să satisfacă un număr mare de cereri.
  - De exemplu, dacă se identifică 20 de cereri care se aplică asupra tabelelor de fapte sau dimensiune, atunci se pot crea 5-6 vizualizări materializate care să satisfacă aceste cereri.
  - Definiția unei vizualizări materializate poate include oricâte funcții agregat și oricâte *join*-uri.
  - Sistemul *Oracle* oferă o serie de proceduri consultative, prin pachetul *DBMS_OLAP*, pentru designul și evaluarea vizualizărilor materializate destinate rescrierii cererilor. Aceste funcționalități sunt cunoscute sub denumirea de *Summary Advisor* sau *Advisor* (sunt disponibile și în *Oracle Enterprise Manager*).
  - Dacă vizualizarea materializată va fi utilizată pentru rescrierea cererilor, atunci aceasta trebuie stocată în aceeași bază de date ca și tabelele de fapte sau/și dimensiune pe care se bazează aceasta.
  - O vizualizare materializată poate fi partiționată și se pot defini vizualizări materializate pe tabele partiționate.

Exemple de vizualizări materializate utilizate la rescrierea cererilor

- Exemple de vizualizările materializate care conțin *join*-uri și agregate.

Exemplul 8.2

```
CREATE MATERIALIZED VIEW vm8_2_sum_domeniu_luna
  ENABLE QUERY REWRITE
  AS
SELECT p.categorie_4 AS domeniu, t.luna AS luna,
       SUM(cantitate * pret_unitar_vanzare) AS valoare
FROM   vanzari v, produse p, timp t
WHERE  t.id_timp = v.timp_id
AND    v.produs_id = p.id_produs
GROUP BY categorie_4, luna;
```

Exemplul 8.3

```
CREATE MATERIALIZED VIEW vm8_3_sum_prod_luna_client
  ENABLE QUERY REWRITE
  AS
SELECT p.denumire AS produs_denumire, t.luna AS luna,
       v.client_id AS client,
       SUM(cantitate * pret_unitar_vanzare) AS valoare
FROM   vanzari v, produse p, timp t
WHERE  t.id_timp = v.timp_id
AND    v.produs_id = p.id_produs
GROUP BY p.denumire, t.luna, v.client_id;
```

Exemplul 8.4

```
CREATE MATERIALIZED VIEW vm8_4_sum_domeniu_luna_oras
  ENABLE QUERY REWRITE
  AS
SELECT p.categorie_4 AS domeniu, t.luna AS luna,
       r.oras AS oras,
       SUM(cantitate * pret_unitar_vanzare) AS valoare,
       COUNT(cantitate * pret_unitar_vanzare) AS numar
FROM   vanzari v, produse p,
       timp t, regiuni r
WHERE  t.id_timp = v.timp_id
AND    v.produd_id = p.id_produs
AND    v.regiune_id = r.id_regiune
GROUP BY p.categorie_4, t.luna, r.oras;
```

- Vizualizările materializate care conțin doar *join*-uri.

Exemplul 8.5

```
CREATE MATERIALIZED VIEW vm8_5_produs_trimestru
  ENABLE QUERY REWRITE
  AS
SELECT denumire, trimestru,
       cantitate * pret_unitar_vanzare AS valoare
FROM   vanzari v, produse p, timp t
WHERE  t.id_timp = v.timp_id
AND    v.produd_id = p.id_produs;
```

Exemplul 8.6

```
CREATE MATERIALIZED VIEW vm8_6_produs_trimestru_outer
  ENABLE QUERY REWRITE
  AS
SELECT denumire, trimestru,
       cantitate * pret_unitar_vanzare AS valoare
FROM   vanzari v, produse p, timp t
WHERE  v.timp_id = t.id_timp(+)
AND    p.id_produs = v.produs_id (+)
and    v.produs_id is null;
```

- Pentru a permite optimizatorului să determine dacă poate utiliza rescrierea cererilor trebuie colectate statistici. Aceasta se poate realiza pentru fiecare obiect sau pentru toate obiectele noi create care nu au statistici.

Exemplul 8.7

Colectarea statisticilor pentru un obiect (de exemplu, pentru vizualizarea materializată *vm8_5_produs_trimestru*):

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS
     ('MASTER_DW','VM8_5_PRODUS_TRIMESTRU',
      estimate_percent=>100,block_sample=>TRUE,
      cascade=>TRUE);
```

Colectarea statisticilor pentru toate obiectele noi create, la nivel de schemă:

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS
      ('MASTER_DW', options => 'GATHER EMPTY',
       estimate_percent=>100, block_sample=>TRUE,
       cascade=>TRUE);
```

8.2.2. Activarea rescrierii cererilor

- Pentru activarea rescrierii cererilor trebuie urmați anumiți pași:
  1. vizualizările materializate ce vor fi folosite la rescriere trebuie să fie definite cu opțiunea *ENABLE QUERY REWRITE*;
  2. parametrul de inițializare *QUERY_REWRITE_ENABLED* trebuie să fie setat la valoarea *TRUE*;
  3. parametrul de inițializare *OPTIMIZER_MODE* trebuie să aibă una dintre valorile *ALL_ROWS*, *FIRST_ROWS* sau *CHOOSE*.
  4. parametrul de inițializare *OPTIMIZER_FEATURES_ENABLE* trebuie să nu fie setat (să nu i se dea nicio valoare); dacă totuși parametrul are setată o valoare, atunci aceasta trebuie să fie cel puțin 8.1.6.
  5. utilizatorul să dețină privilegiul sistem *QUERY REWRITE* sau *GLOBAL QUERY REWRITE*;
  6. cererea să nu utilizeze *hint*-ul *NOREWRITE*;
  7. depinde de nivelul de integritate al cererii care este dat de parametrul *QUERY_REWRITE_INTEGRITY*.

Parametrul *QUERY_REWRITE_INTEGRITY*

- Acest parametru determină gradul de integritate pentru care sistemul *Oracle* aplică rescrierea cererilor. Implicit acesta are valoarea *ENFORCED*.

```
QUERY_REWRITE_INTEGRITY =
   {STALE_TOLERATED | TRUSTED | ENFORCED}
```

  - Valoarea parametrului poate fi setată la nivel de:
    - sesiune (*ALTER SESSION*);
    - sistem (*ALTER SYSTEM*).
  - Valoarea *ENFORCED* presupune că sistemul *Oracle* aplică și garantează consistența și integritatea. Optimizatorul va folosi doar vizualizările materializate care conțin date actualizate (*fresh*) și va folosi doar acele relații care sunt bazate pe constrângeri active și validate de cheie primară, unicitate, cheie externă.
  - Valoarea *TRUSTED* presupune că sistemul *Oracle* permite rescrierea cererilor care utilizează relații ce au fost declarate, dar care nu au fost aplicate. Optimizatorul are încrederea că datele vizualizărilor materializate sunt actualizate și că relațiile

declarate în obiectele *dimension* și constrângerile de tip *RELY* sunt corecte. În acest mod, optimizatorul va folosi vizualizări materializate *prebuilt* sau vizualizări materializate bazate pe alte vizualizări. De asemenea, va folosi atât relațiile validate cât și pe cele nevalidate. Astfel, optimizatorul are încredere în constrângerile de cheie primară, unicitate și relațiile specificate în obiectele *dimension*, declarate fără clauza *ENABLED VALIDATED*.

- Valoarea *STALE_TOLERATED* presupune că sistemul *Oracle* permite rescrierea folosind relații nevalidate. Optimizatorul folosește atât vizualizările materializate care sunt valide, dar care conțin date neactualizate (*stale*), cât și vizualizările cu date actualizate (*fresh*). Acest mod oferă capacitatea maximă de rescriere a cererilor, dar determină riscul generării de rezultate incorecte.

- Dacă nivelul de integritate al rescrierii este cel mai sigur – *ENFORCED* – atunci optimizatorul va aplica constrângerile de cheie primară și pe cele referențiale pentru a asigura că rezultatele cererii sunt aceleași cu rezultatele obținute în cazul în care cererea ar accesa direct tabelele de bază.
- Dacă nivelul de integritate al rescrierii este altul decât *ENFORCED*, atunci apar câteva situații în care rezultatul obținut prin rescriere poate fi diferit de cel obținut dacă s-ar utiliza tabelele de bază.
  1. O vizualizare materializată poate să nu fie sincronizată cu datele tabelelor de bază. Aceasta se poate întâmpla atunci când procedura de actualizare a vizualizării este în așteptare după o încărcare masivă cu date sau după operațiile *LMD* asupra unuia sau mai multor tabele de bază ale vizualizării respective. Aceeași situație este întâlnită dacă vizualizarea este actualizată la anumite intervale de timp.
  2. Relațiile implementate în obiectele *dimension* pot fi invalide. De exemplu, valorile unui anumit nivel ierarhic nu au exact un părinte.
  3. Valorile stocate în tabela *prebuilt* care determină vizualizarea materializată pot fi incorecte.
  4. Operațiile asupra submulțimiilor (*DROP* și *MOVE PARTITION*) tabelelor de bază partiționate pot afecta rezultatele vizualizării materializate.

Parametrul *OPTIMIZER_MODE*

- Acest parametru stabilește modul implicit pentru alegerea unei abordări de optimizare. Valoarea implicită a parametrului este *CHOOSE*.

```
OPTIMIZER_MODE =
          {FIRST_ROWS_[1 | 10 | 100 | 1000]
         | FIRST_ROWS | ALL_ROWS | CHOOSE | RULE}
```

Hint-uri pentru rescrierea cererilor

- *Hint*-ul *NOREWRITE* într-o cerere previne ca optimizatorul să rescrie cererea.

Exemplul 8.8

```
SELECT /*+ NOREWRITE */
       p.categorie_4 AS domeniu, t.luna AS luna,
       SUM(cantitate * pret_unitar_vanzare) AS valoare
FROM   vanzari v, produse p, timp t
WHERE  t.id_timp = v.timp_id
AND    v.produs_id = p.id_produs
GROUP BY categorie_4, luna;
```

- *Hint*-ul *REWRITE* fără niciun argument forțează optimizatorul să folosească o vizualizare materializată pentru a rescrie cererea, fără a ține cont de costuri.
- *Hint*-ul *REWRITE*(*mv1*,*mv2*,...) selectează cele mai potrivite vizualizări materializate pentru a rescrie cererea.

Exemplul 8.9

```
SELECT /*+ REWRITE(vm8_2_sum_domeniu_luna) */
       p.categorie_4 AS domeniu, t.luna AS luna,
       SUM(cantitate * pret_unitar_vanzare) AS valoare
FROM   vanzari v, produse p, timp t
WHERE  t.id_timp = v.timp_id
AND    v.produs_id = p.id_produs
GROUP BY categorie_4, luna;
```

- Dacă cererea constă din mai multe blocuri *SELECT*, atunci se pot specifica *hint*-uri pentru fiecare bloc *SELECT*.

Privilegii pentru a putea activa rescrierea cererilor

- Privilegiul sistem *QUERY REWRITE* permite utilizarea pentru rescrierea cererilor a vizualizărilor materializate din schema proprie care referă tabele din acea schemă.
- Privilegiul sistem *GLOBAL QUERY REWRITE* permite utilizarea vizualizărilor materializate pentru rescrierea cererilor, chiar dacă aceasta referă tabele din alte scheme.

AICI M-AM OPRIT!!!!!

8.2.3. Metode de rescriere a cererilor

8.2.3.1. Metode de rescriere prin comparare la nivel de text SQL

Optimizatorul poate folosi două astfel de metode:

- comparare completă;
- comparare parțială.

Dacă este utilizată metoda de comparare completă, atunci este comparat textul întreg al comenzii cu definiția vizualizării materializate (întreaga expresie *SELECT*), ignorându-se spațiile.

Exemplul 8.10

```
SELECT p.categorie_4 AS domeniu, t.luna AS luna,
       r.oras AS oras,
       SUM(cantitate * pret_unitar_vanzare) AS valoare,
       COUNT(cantitate* pret_unitar_vanzare) AS numar
FROM   vanzari v, dim_produse p,
       dim_timp t, dim_regiuni r
WHERE  t.id_timp = v.timp_id
AND    v.produd_id = p.id_produs
AND    v.regiune_id = r.id_regiune
GROUP BY p.categorie_4, t.luna, r.oras;
```

Această cerere se potrivește complet cu definiția vizualizării materializate *vm3_sum_domeniu_luna_oras* și va fi rescrisă astfel:

```
SELECT domeniu, luna, oras, valoare, numar
FROM   vm3_sum_domeniu_luna_oras;
```

Dacă operația de comparare completă se încheie cu o eșuare, atunci Optimizatorul încearcă compararea parțială. În acesta metodă, se compară textul care apare în clauzele *FROM*.

Exemplul 8.11

```
SELECT p.categorie_4 AS domeniu, t.luna AS luna,
       r.oras AS oras,
       AVG(cantitate * pret_unitar_vanzare) AS medie
FROM   vanzari v, dim_produse p,
       dim_timp t, dim_regiuni r
```

```
WHERE  t.id_timp = v.timp_id
AND    v.produd_id = p.id_produs
AND    v.regiune_id = r.id_regiune
GROUP BY p.categorie_4, t.luna, r.oras;
```

Cererea de mai sus este rescrisă astfel:

```
SELECT domeniu, luna, oras, valoare/numar AS medie
FROM   vm3_sum_domeniu_luna_oras;
```

Dacă nici compararea parțială nu se încheie cu succes, atunci Optimizatorul va încerca alte metode de rescriere.

De remarcat este faptul că Optimizatorul realizează o comparare *case-sensitive* (de exemplu, un cuvânt cheie precum *FROM* trebuie scris la fel în cerere cât și în definiția vizualizării).

8.2.3.2. Metode generale pentru rescrierea cererilor

Pentru a determina dacă o cerere poate fi rescrisă, sistemul *Oracle* realizează mai multe tipuri de verificări:

- compatibilitatea selecției;
- compatibilitatea *join*-ului;
- suficiența datelor;
- compatibilitatea grupărilor;
- compatibilitatea agregărilor.

Metodele generale de rescriere a cererilor ce pot fi utilizate

Metode de verificare a rescrierii	**VM doar cu *join*-uri**	**VM cu *join*-uri și agregăte**	**VM cu agregate pe o singură tabelă**
Compatibilitatea selecției	X	X	X
Compatibilitatea *join*-ului	X	X	-
Suficiența datelor	X	X	X
Compatibilitatea grupărilor	-	X	X
Compatibilitatea agregărilor	-	X	X

Dimensiuni și constrângeri necesare pentru rescrierea cererilor

Metode de verificare a rescrierii	**Dimensiuni**	**Constrângeri de cheie primară, cheie externă și *Not Null***
Compatibilitatea selecției	nu sunt necesare	nu sunt necesare
Compatibilitatea *join*-ului	nu sunt necesare	necesare
Suficiența datelor	necesare	necesare
Compatibilitatea grupărilor	necesare	necesare
Compatibilitatea agregărilor	nu sunt necesare	nu sunt necesare

Constrângeri definite pe vizualizări

Constrângerile definite pe vizualizări trebuie create cu opțiunea *DISABLE NOVALIDATE*. Pentru a permite rescrierea cererilor complexe, starea unei constrângeri definită pe o vizualizare poate fi *RELY* sau *NORELY*. De exemplu, o constrângere pe o vizualizare care este în starea *RELY* permite rescrierea cererii atunci când nivelul de integritate al cererii este *TRUSTED*.

Stare Constrângere / Mod integritate rescriere	**RELY**	**NORELY**
ENFORCED	Nu	Nu
TRUSTED	Da	Nu
STALE_TOLERATED	Da	Nu

Pentru a demonstra capacitatea de rescriere a cererilor pe vizualizări, trebuie extinsă schema cu următoarea vizualizare:

```
CREATE VIEW v_timp AS
SELECT id_timp,
       TO_NUMBER(TO_CHAR(id_timp,'ddd'))AS zi_in_an
FROM dim_timp;
```

Acum se poate stabili o relație referențială în modul *RELY*, între vizualizare și tabelul de fapte. Rescrierea va merge pentru acest exemplu în modul *TRUSTED*.

```
ALTER VIEW v_timp
ADD CONSTRAINT pk_v_timp PRIMARY KEY (id_timp)
    DISABLE NOVALIDATE;
ALTER VIEW timp_view
MODIFY CONSTRAINT timp_view_pk RELY;
ALTER TABLE vanzari
ADD CONSTRAINT timp_view_fk FOREIGN key (id_timp)
    REFERENCES timp_view(timp_id)
    DISABLE NOVALIDATE);
ALTER TABLE vanzari
MODIFY CONSTRAINT timp_view_fk RELY;

CREATE MATERIALIZED VIEW vanzari_pcat_cal_zi_mv
ENABLE QUERY REWRITE
AS
SELECT p_categorie, zi_in_an,
       SUM(svaloare_sold) as sum_valoare_sold
FROM  timp_view, vanzari, produse
WHERE id_timp = timp_id AND prod_id = id_prod
GROUP BY p_categorie, zi_in_an;
```

Următoarea cerere, care nu utilizează tabelul *produse*, va fi rescrisă fără relația referențială, deoarece *join*-ul suprimat dintre tabelul de fapte *vanzari* și tabelul dimensiune *produse* nu este necesar.

```
SELECT zi_in_an,
       SUM(valoare_sold) AS sum_valoare_sold
FROM   timp_view, vanzari
WHERE  timp_id = id_timp
GROUP BY zi_în_an;
```

Totuși, dacă vizualizarea materializată *vanzari_pcat_cal_zi_mv* a fost definită numai asupra vizualizării *timp_view*, atunci nu se poate rescrie următoarea cerere:

```
SELECT p_categorie,
       SUM(valoare_sold) AS sum_valoare_sold
FROM   vanzari, produse
WHERE  prod_id = id_prod
GROUP BY p_categorie;
```

Potrivirea expresiei

O expresie ce apare în cerere poate fi înlocuită cu o coloană simplă din vizualizarea materializată, care reprezintă acea expresie precalculată. Dacă cererea poate fi rescrisă cu expresia precalculată, atunci cererea se va executa mai rapid.

***Exemplu*:**

Se consideră vizualizarea materializată *vm2_sum_prod_luna_client* și următoarea cerere *SQL*:

```
SELECT p.denumire, t.luna, v.client_id,
       SUM(cantitate* pret_unitar_vanzare)
FROM   vanzari v, dim_produse p, dim_timp t
WHERE  t.id_timp = v.timp_id
AND    v.produs_id = p.id_produs
GROUP BY p.denumire, t.luna, v.client_id;
```

Cererea va fi rescrisă astfel:

```
SELECT produs, luna, client, valoare
FROM   vm2_sum_prod_luna_client;
```

Formatul datei calendaristice

Reprezintă o formă specifică a metodei de rescriere prin potrivirea expresiei. În acest tip de rescriere data calendaristică din cerere este inclusă într-o dată calendaristică echivalentă, având gradul de granularitate mai mare. Data care o cuprinde reprezintă o granulă mai mare ca de exemplu luni, trimestre sau ani.

Potrivirea expresiei se realizează prin folosirea formelor canonice pentru datele calendaristice.

Tipul *DATE* este un tip de date care reprezintă unități ordonate de timp ca secunde, zile, luni și incorporează o ierarhie precum secundă → minut → oră → zi → lună → trimestru → an. De exemplu, data 01-01-2010 poate fi inclusă în anul 2010 sau în primul trimestru al anului 2010, adică 2010-1 sau în prima lună a anului 2010, adică 2010-01.

Deoarece valorile datelor calendaristice sunt ordonate, un predicat specificat pe o coloană de tip *DATE* poate fi inclus într-un nivel mai mare de granularitate. De exemplu, predicatul *date_col BETWEEN '01-01-2010' AND '01-07-2010'* poate fi inclus în granula lună sau granula trimestru, folosind funcția *TO_CHAR*.

***Exemplu*:**

Se consideră următoarea vizualizare materializată:

```
CREATE MATERIALIZED VIEW vm_categorii_lunare
ENABLE QUERY REWRITE
AS
SELECT categorie_5 AS categorie,
       TO_CHAR(timp_id,'YYYY-MM') AS luna,
       SUM(cantitate* pret_unitar_vanzare) AS valoare
FROM   vanzari v, dim_produse p
WHERE  v.produs_id = p.id_produs
GROUP BY categorie_5, TO_CHAR(timp_id,'YYYY-MM');
```

Se consideră cererea care listează valoarea vânzărilor pe categorii de produse pentru anul 2007.

```
SELECT categorie_5 AS categorie,
       SUM(cantitate* pret_unitar_vanzare)
FROM   vanzari v, dim_produse p
WHERE  v.produs_id = p.id_produs
AND    TO_CHAR(v.timp_id, 'YYYY-MM') BETWEEN '2007-01'
       AND '2007-12'
GROUP BY p_categorie;
Cererea va fi rescrisă astfel:
SELECT categorie, valoare
FROM   vm_vanzari_lunare
WHERE  luna BETWEEN '2007-01' AND '2007-12';
```

Compatibilitatea selecției

Sistemul *Oracle* permite rescrierea cererilor astfel încât să folosească vizualizări materializate în care clauza *HAVING* sau *WHERE* conține o selecție a unei submulțimi de date dintr-o tabelă.

Compatibilitatea selecției poate fi:

- simplă – selecții de genul *<expresie> operator_relational <constanta>*;
- complexă – selecții de genul *<expresie> operator_relational <expresie>*;
- de tip *range* – selecții de genul *<expresie> BETWEEN <const1> AND <const2>*;
- de tip *in list* – selecții de genul *lista_coloane IN lista_valori*; de remarcat este faptul că selecțiile de forma *coloana1='v1' OR coloana1='v2' OR coloana1='v3' OR ....* sunt tot de tipul listă;
- de tip *IS* [*NOT*] *NULL*;
- de tip [*NOT*] *LIKE*.

Atunci când se compară selecția din cerere cu cea din vizualizarea materializată se verifică dacă partea stângă a celor două selecții coincide. Dacă se folosește un interval de valori, atunci se compară partea dreaptă, ca în exemplul următor.

***Exemplul 1*:**

În cerere:

```
WHERE prod_id = 102
```

În vizualizare:

```
WHERE prod_id BETWEEN 0 AND 200
```

***Exemplul 2*:**

În cerere:

```
WHERE prod_id > 10 AND prod_id < 50
```

În vizualizare:

```
WHERE prod_id BETWEEN 0 AND 200
```

***Exemplul 3*:**

În cerere:

```
WHERE vanzari.valoare_sold * .07 BETWEEN
      1 AND 100
```

În vizualizare:

```
WHERE vanzari.valoare_sold *. 07 BETWEEN
           0 AND 200
```

***Exemplul 4*:**

În cerere:

```
WHERE cost.unit_pret * 0.95 >
      cost_unit_cost * 1.25
```

În vizualizare:

```
WHERE cost.unit_pret * 0.95 >
      cost_unit_cost * 1.25
```

***Exemplul 5*:**

Selecțiile din cerere nu sunt constrânse de nici o selecție din vizualizare, dar valorile din dreapta trebuie conținute de vizualizare. Cererea următoare va fi rescrisă.

În cerere:

```
WHERE p_nume = 'pantaloni'
AND   p_categorie = 'barbati'
```

În vizualizare:

```
WHERE p_categorie = 'barbati'
```

***Exemplul 6*:**

Vizualizarea este mai restrictivă decât cererea. Cererea următoare nu va fi rescrisă.

În cerere:

```
WHERE p_categorie = 'barbati'
```

În vizualizare:

```
WHERE p_nume = 'pantaloni'
AND   p_categorie = 'barbati'
```

***Exemplul 7*:**

În cerere:

```
WHERE prod_id, client_id IN
      ((1022, 1000), (1033, 2000))
```

În vizualizare:

```
WHERE prod_id IN (1022,1033)
AND   client_id IN (1000, 2000)
```

***Exemplul 8*:**

În cerere:

```
WHERE prod_id = 1022
AND   client_id IN (1000, 2000)
```

În vizualizare:

```
WHERE (prod_id, client_id) IN
(1022, 1000), (1022, 2000))
```

***Exemplul 9*:**

Rescrierea eșuează.

În cerere:

```
WHERE prod_id = 1022
AND   client_id IN (1000, 2000)
```

În vizualizare:

```
WHERE (prod_id, client_id, c_oras)
            IN ((1022, 1000, 'Bucuresti'),
                (1022, 2000, 'Iasi'))
```

Compatibilitatea selecțiilor se poate realiza și pentru selecții complexe de genul (*selecție AND selecție AND* ...) *OR* (*selecție AND selecție AND* ...)

Fiecare grup de selecții relaționate prin operatorul *AND* este în disjuncție cu un alt grup. De fapt se obține o disjuncție de conjuncții.

***Exemplul 10*:**

Cererea va fi rescrisă:

În cerere:

```
WHERE populatie_oras > 15000
```

```
AND   populatie_oras < 25000
     AND   judet= 'Vrancea'
```

În vizualizare:

```
WHERE (populatie_oras < 5000
       AND judet = 'Bucuresti')
     OR    (populatie_oras BETWEEN 10000 AND 50000
            AND judet = ' Vrancea ')
```

***Exemplul 11*:**

```
CREATE MATERIALIZED VIEW cal_luna_vanzari_id_mv
BUILD IMMEDIATE
REFRESH FORCE
ENABLE QUERY REWRITE
AS
SELECT    calendar_luna_desc,
          SUM(valoare_sold) AS suma
FROM      vanzari,timp
WHERE     timp_id = id_timp AND id_client = 10
GROUP BY  calendar_luna_desc;
```

Cererea următoare poate fi rescrisă deoarece întoarce date despre clientul 10.

```
SELECT   calendar_luna_desc,
         SUM(valoare_sold) AS suma
FROM     vanzari,timp
WHERE    timp_id = id_timp AND id_client = 10
GROUP BY calendar_luna_desc;
```

Cererea va fi rescrisă astfel:

```
SELECT calendar_luna_desc, suma
FROM   cal_luna_vanzari_id_mv;
```

***Exemplul 12*:**

```
CREATE MATERIALIZED VIEW produs_vanzari_mv
   BUILD IMMEDIATE
   REFRESH FORCE
   ENABLE QUERY REWRITE
   AS
   SELECT   p_nume,
            SUM(valoare_sold) AS suma
   FROM     produse, vanzari
   WHERE    prod_id = id_prod
```

```
   GROUP BY p_nume
   HAVING SUM(valoare_sold) BETWEEN 5000 AND 50000;
```

Se dă următoarea cerere:

```
SELECT   p_nume, SUM(valoare_sold) AS suma
FROM     produse, vanzari
WHERE    prod_id = id_prod
GROUP BY p_nume
HAVING SUM(valoare_sold) BETWEEN 10000 AND 20000;
```

Cererea va fi rescrisă astfel:

```
SELECT p_nume, suma
FROM   produs_vanzari_mv
WHERE  suma > 10000 AND suma < 20000;
```

***Exemplul 13*:**

```
CREATE MATERIALIZED VIEW
       vanzari_valentines_day_99_mv
BUILD IMMEDIATE
REFRESH FORCE
ENABLE QUERY REWRITE
AS
  SELECT  prod_id, id_client, valoare_sold
  FROM    vanzari,timp
  WHERE   timp_id = id_timp
  AND timp_id=TO_DATE('14-FEB-1999','DD-MON-YYYY');
```

Se dă următoarea cerere:

```
SELECT  prod_id, id_client, valoare_sold
FROM    vanzari,timp
WHERE   timp_id = id_timp
AND timp_id=TO_DATE('14-FEB-1999', 'DD-MON-YYYY');
```

Cererea va fi rescrisă astfel:

```
SELECT * FROM vanzari_valentines_day_99_mv;
```

***Exemplul 14*:**

```
CREATE MATERIALIZED VIEW promo_vanzari_mv
```

```
BUILD IMMEDIATE
REFRESH FORCE
ENABLE QUERY REWRITE
AS
  SELECT  promo_nume,
          SUM(valoare_sold) AS sum_valoare_sold
  FROM    promotii, vanzari
  WHERE   promo_id = id_promo
  AND     promo_nume IN
             ('cupon', 'premium', 'fluturas')
  GROUP BY promo_name;
```

Se dă următoarea cerere:

```
SELECT promo_nume, SUM(valoare_sold)
FROM   promotii, vanzari
WHERE  promo_id = id_promo
AND    promo_nume IN ('cupon', 'premium')
GROUP BY promo_name;
```

Cererea va fi rescrisă astfel:

```
SELECT *
FROM   promo_vanzari_mv
WHERE  promo_nume IN ('coupon', 'premium');
```

Cererea următoare va și ea rescrisă ca mai jos:

```
SELECT promo_nume, SUM(valoare_sold)
FROM   promotii, vanzari
WHERE  promo_id = id_promo
AND    promo_nume  = 'cupon'
GROUP BY promo_name;
HAVING   SUM(valoare_sold) > 1000;

SELECT *
FROM   promo_vanzari_mv
WHERE  promo_nume = 'cupon'
AND    sum_valoare_sold > 1000;
```

Compatibilitatea join-ului

Join-urile din cerere sunt comparate cu *join*-urile din definiția vizualizării materializate. În general, această comparare poate fi clasificată în trei categorii:

1. *join*-uri comune, care apar atât în cerere cât și în vizualizare;
2. *join*-uri delta, care apar în cerere, dar nu și în vizualizare (cereri *delta join*);
3. *join*-uri delta, care apar în vizualizare, dar nu și în cerere (vizualizări *delta join*).

Join-uri comune

***Exemplul 1*:**

```
SELECT p_nume, zi_sf_saptamana, SUM(valoare_sold)
FROM   vanzari, produse, timp
WHERE  timp_id=id-timp AND prod_id = id_prod
AND    zi_sf_saptamana BETWEEN
       TO_DATE('01-AUG-1999', 'DD-MON-YYYY')
       AND TO_DATE('10-AUG-1999', 'DD-MON-YYYY')
GROUP BY p_nume, zi_sf_saptamana;
```

Join-urile comune între această cerere și vizualizarea materializată *join_vanzari_timp_produs_mv* sunt *timp_id=id_timp* și *prod_id = id_prod*.

Join-urile se potrivesc și cererea poate fi rescrisă astfel:

```
SELECT p_nume, zi_sf_saptamana, SUM(valoare_sold)
FROM   join_vanzari_timp_produs_mv
WHERE  zi_sf_saptamana BETWEEN
       TO_DATE('01-AUG-1999','DD-MON-YYYY')
       AND TO_DATE('10-AUG-1999','DD-MON-YYYY')
GROUP BY p_nume, zi_sf_saptamana;
```

De asemenea, cererea poate fi procesată și utilizând vizualizarea *join_vanzari_timp_produs_oj_mv* unde *join*-urile din cerere pot fi obținute din *outerjoin*-urile din vizualizare:

```
SELECT p_nume, zi_sf_sapatamana, SUM(valoare_sold)
FROM   join_vanzari_timp_produs_oj_mv
WHERE  zi_sf_saptamana BETWEEN
       TO_DATE('01-AUG-1999','DD-MON-YYYY')
       AND TO_DATE('10-AUG-1999','DD-MON-YYYY')
AND    prod_id IS NOT NULL
```

```
GROUP BY p_nume, zi_sf_saptamana;
```

***Exemplul 2*:**

```
SELECT DISTINCT p_nume
FROM   produse p
WHERE EXISTS
  (SELECT *
   FROM   vanzari s
   WHERE  p.prod_id=s.id_prod
   AND    s.valoare_sold > 1000);
```

Această cerere poate fi interpretată și astfel:

```
SELECT DISTINCT p_nume
FROM   produse p
WHERE  p.prod_id IN (SELECT s.id_prod
                     FROM   vanzari s
                     WHERE  s.valoare_sold > 1000);
```

Această cerere conține un *semijoin* între tabelul *produse* și tabelul *vanzari*:

```
s.id_prod = p.prod_id.
```

Cererea poate fi rescrisă astfel:

```
SELECT p_nume
FROM (SELECT DISTINCT p_nume
      FROM   join_vanzari_timp_produs_mv
      WHERE  valoare_sold > 1000);
```

Dacă vizualizarea materializată *join_vanzari_timp_produs_mv* este partiționată după *timp_id*, atunci această cerere va fi mai eficientă decât cererea originală, deoarece *join*-ul original dintre tabelul *vanzari* și tabelul *produse* poate fi evitat.

```
SELECT p_nume
FROM (SELECT DISTINCT p_nume
      FROM   join_vanzari_timp_produs_oj_mv
      WHERE  valoare_sold > 1000
      AND    prod_id IS NOT NULL);
```

Cereri delta join

***Exemplul 3*:**

```
SELECT p_nume, zi_sf_saptamana, c_oras,
       SUM(valoare_sold)
FROM   vanzari,clienti,timp,produse
WHERE  timp_id=id_timp
AND    prod_id=id_prod
AND    client_id = id_client
GROUP BY p_nume, zi_sf_saptamana, c_oras;
```

Folosind vizualizarea *join_vanzari_timp_produs_mv* *join*-urile comune sunt *timp_id=id_timp* și *prod_id=id_prod*. *Join*-ul delta este *client_id = id_client*.

Cererea va fi rescrisă astfel:

```
SELECT mv.p_nume, mv.zi_sf_saptamana, c.c_oras,
       SUM(mv.valoare_sold)
FROM   join_vanzari_timp_produs_mv mv, clienti c
WHERE  mv.id_client = c.client_id
GROUP BY mv.p_nume, mv.zi_sf_saptamana, c.c_oras;
```

Vizualizări materializate delta join

Pentru ca o cerere să poată fi rescrisă printr-o vizualizare materializată care conține *join*-uri suplimentare, acele *join*-uri nu trebuie să determine pierderea de înregistrări (înregistrări care nu îndeplinesc condiția de *join*). Pentru aceasta coloanele care intră în *join* trebuie să aibă declarate constrângerile *FOREIGN KEY*, *PRIMARY KEY* și *NOT NULL*.

***Exemplul 4*:**

```
SELECT zi_sf_saptamana, SUM(valoare_sold)
FROM   vanzari,timp
WHERE  timp_id =id_timp
AND    zi_sf_saptamana BETWEEN
       TO_DATE('01-AUG-1999', 'DD-MON-YYYY')
       AND TO_DATE('10-AUG-1999', 'DD-MON-YYYY')
GROUP BY zi_sf_saptamana;
```

Vizualizarea *join_vanzari_timp_produs_mv* are un *join* adițional *prod_id=id_prod* între tabelul *vanzari* și tabelul *produse*. Cererea poate fi rescrisă dacă *s.id_prod* este cheie externă și *p.prod_id* este *not null*.

```
SELECT zi_sf_saptamana, SUM(valoare_sold)
FROM   join_vanzari_timp_produs_mv
WHERE  zi_sf_saptamana BETWEEN
       TO_DATE('01-AUG-1999', 'DD-MON-YYYY')
       AND TO_DATE('10-AUG-1999', 'DD-MON-YYYY')
```

```
GROUP BY zi_sf_saptamana;
```

Cererea poate fi rescrisă și utilizând vizualizarea materializată *join_vanzari_timp_produs_oj_mv*. Această vizualizare conține un *outerjoin* între tabelul *vanzari* și tabelul *produse* (*s.id_prod=p.prod_id*(+)). Dacă *p.prod_id* este cheie primară, atunci cererea poate fi rescrisă astfel:

```
SELECT zi_sf_saptamana, SUM(valoare_sold)
FROM   join_vanzari_timp_produs_oj_mv
WHERE  zi_sf_saptamana BETWEEN
       TO_DATE('01-AUG-1999', 'DD-MON-YYYY')
       AND TO_DATE('10-AUG-1999', 'DD-MON-YYYY')
GROUP BY zi_sf_saptamana;
```

Suficiența datelor

În cazul acestei verificări, Optimizatorul determină dacă toate coloanele de date necesare cererii pot fi obținute din vizualizarea materializată. Pentru aceasta se utilizează echivalența coloanelor.

De exemplu, dacă avem un *inner join* între tabelele *A* și *B* cu predicatul de *join A.X = B.X*, atunci datele din coloana *A.X* vor fi egale cu datele din coloana *B.X* în rezultatul *join*-ului. Această proprietate a datelor este utilizată pentru a verifica potrivirea coloanei *A.X* din cerere cu coloana *B.X* din vizualizare.

***Exemplul 1*:**

```
SELECT p_nume, timp.timp_id, zi_sf_saptamana,
       SUM(valoare_sold)
FROM   vanzari, produse, timp
WHERE  timp_id=id_timp AND prod_id = id_prod
GROUP BY p_nume, timp_id, zi_sf_saptamana;
```

Această cerere poate fi rescrisă folosind vizualizarea materializată *join_vanzari_timp_produs_mv*, deși în vizualizare nu apare coloana *timp.timp_id*. Deoarece *timp.timp_id=vanzari.id_timp* Optimizatorul va rescrie cererea.

```
SELECT p_nume, id_timp, zi_sf_saptamana,
       SUM(valoare_sold)
FROM   join_vanzari_timp_produs_mv
GROUP BY p_nume, id_timp, zi_sf_saptamana;
```

Dacă o anumită coloană nu se poate obține direct din vizualizare, atunci Optimizatorul încerca să determine dacă poate obține coloana pe baza dependentelor funcționale (valorile unei coloane determină valorile altei coloane). De exemplu, codul unui produs determină funcțional numele produsului.

***Exemplul 2*:**

```
SELECT p_categorie, zi_sf_saptamana,
       SUM(valoare_sold)
FROM   vanzari, produse, timp
WHERE  timp_id=id_timp AND prod_id = id_prod
AND    p_categorie='barbati'
GROUP BY p_categorie, zi_sf_saptamana;
```

Vizualizarea materializată *sum_vanzari_pcat_sapt_mv* conține *p.prod_id*, dar nu și *p.p_categorie*. Totuși se poate realiza un *join* între vizualizare și tabelul *produse* pentru a obține categoria produsului, deoarece coloana *prod_id* determină în mod funcțional coloana *p_categorie*. Cererea va fi rescrisă astfel:

```
SELECT p_categorie, mv.zi_sf_saptamana,
       SUM(mv.sum_valoare_sold)
FROM   sum_vanzari_prod_sapt_mv mv, produse p
WHERE  mv.prod_id=p.prod_id
AND    p.prod_category='barbati'
GROUP BY p_categorie, mv.zi_sf_saptamana;
```

Dependențele funcționale se pot declara în două moduri:

- utilizarea constrângerii de cheie primară;
- utilizarea clauzei *DETERMINES* a unei dimensiuni.

Clauza *DETERMINES* din definiția unei dimensiuni poate fi uneori singura modalitate de a determina dependența funcțională atunci când coloana care determină altă coloană nu poate fi cheie primară. De exemplu, tabelul *produse* este o tabelă dimensiune denormalizat care are coloanele *prod_id*, *p_nume*, *p_subcategorie* și *p_categorie*. Atributul *p_subcategorie* determină funcțional *p_subcat_desc*, iar *p_categorie* determină funcțional *p_categ_desc*. Atât *p_subcateg* cât și *p_categ* nu pot fi chei primare în această tabelă.

***Exemplul 3*:**

```
CREATE DIMENSION produse_dim
LEVEL  produs IS (produse.prod_id)
LEVEL  subcategorie IS (produse.p_subcategorie)
```

```
LEVEL  categorie IS (produse.p_categorie)
HIERARCHY prod_rollup (
    produs        CHILD OF
    subcategorie  CHILD OF
    categorie)
ATTRIBUTE produs DETERMINES produse.prod_nume
ATTRIBUTE produs DETERMINES produse.prod_desc
ATTRIBUTE subcategorie
          DETERMINES produse.p_subcateg_desc
ATTRIBUTE categorie
          DETERMINES produse.p_categ_desc;
```

Se dă cererea următoare:

```
SELECT p_subcateg_desc, zi_sf_saptamana,
       SUM(valoare)
FROM   vanzari, produse, timp
WHERE  timp_id=id_timp AND prod_id = id_prod
AND    p_subcateg_desc LIKE '%barbati'
GROUP BY p_subcateg_desc, zi_sf_saptamana;
```

Această cerere va fi rescrisă astfel:

```
SELECT iv.p_subcateg_desc, mv.zi_sf_saptamana,
       SUM(mv.sum_valoare_sold)
FROM   sum_vanzari_pcat_sapt_mv mv,
       (SELECT DISTINCT p_subcategorie,
               p_subcateg_desc
        FROM produse) iv
WHERE  mv.p_subcategorie=iv.p_subcategorie
AND    iv.p_subcateg_desc LIKE '%barbati'
GROUP BY iv.p_subcateg_desc, mv.zi_sf_saptamana;
```

Compararea grupărilor

Această verificare este necesară dacă atât cererea cât și vizualizarea conțin clauza *GROUP BY*. Inițial Optimizatorul determină dacă gruparea datelor cerute de cerere este identică cu cea folosită în vizualizare. De exemplu, o cerere folosește o grupare după *p_subcategorie* și o vizualizare materializată stochează datele grupate după *p_subcateg* și *p_subcateg_desc*. Gruparea este aceeași în ambele cazuri determinând gruparea după atributul *p_subcategorie* care determină funcțional atributul *p_subcateg_desc*.

Vizualizarea *sum_vanzari_pcat_sapt_mv* grupează datele după *zi_sf_sapt* și *p_subcategorie*. Urmatoarea cerere grupează datele după *p_subcategorie*.

***Exemplul 1*:**

```
SELECT p_subcategorie,
       SUM(valoare_sold) AS sum_valoare
FROM   vanzari, produse
WHERE  prod_id=id_prod
GROUP BY p.p_subcategorie;
```

De aceea, Optimizatorul va rescrie cererea astfel:

```
SELECT p_subcategorie, SUM(sum_valoare_sold)
FROM   sum_vanzari_pcat_saptamana_mv
GROUP BY p_subcategorie;
```

***Exemplul 2*:**

```
SELECT p_categorie, zi_sf_saptamana,
       SUM(valoare_sold) AS sum_valoare
FROM   vanzari,produse,timp
WHERE  timp_id=id_timp AND prod_id=id_prod
GROUP BY p_categorie,zi_sf_saptamana;
```

Deoarece *p_subcategorie* determină funcțional *p_categorie*, vizualizarea *sum_vanzari_pcat_ saptamana_mv* poate fi folosită pentru rescriere.

```
SELECT pv.p_subcategorie, mv.zi_sf_saptamana,
       SUM(mv.sum_valoare_sold)
FROM   sum_vanzari_pcat_saptamana_mv mv,
       (SELECT DISTINCT p_subcategorie, p_categorie
        FROM produse) pv
WHERE  mv.p_subcategorie=mv.p_subcategorie
GROUP BY pv.p_subcategorie, mv.zi_sf_saptamana;
```

Compararea agregatelor

Această verificare este necesară doar dacă atât cererea, cât și vizualizarea conțin agregate. De exemplu, dacă cererea returnează *AVG*(*X*), iar vizualizarea conține *SUM*(*X*) și *COUNT*(*X*), atunci cererea poate fi rescrisă cu *SUM*(*X*)/*COUNT*(*X*).

Dacă verificarea compatibilității grupărilor determină că vizualizarea stochează agregate de tip *rollup*, atunci se încercă determinarea fiecărui agregat cerut de cerere prin utilizarea agregatelor de nivele inferioare ale vizualizării materializate. De exemplu,

SUM(*vanzari*) la nivel de județ se poate obține prin însumarea valorilor *SUM*(*vanzari*) la nivel de oraș care au aceeași valoare pentru județ.

***Exemplu*:**

```
SELECT  p_subcategorie, AVG(valoare) AS avg_vanzari
FROM    vanzari, produse
WHERE   prod_id = id_prod
GROUP BY p_subcategorie;
```

Optimizatorul va rescrie cererea astfel:

```
SELECT mv.p_subcategorie,
SUM(mv.sum_valoare_sold)/SUM(mv.count_valoare_sold)
       AS avg_vanzari
FROM   sum_vanzari_pcat_luna_oras_mv mv
GROUP BY mv.p_subcategorie;
```

Argumentul unei funcții agregat precum *SUM* poate fi o expresie aritmetică precum *A*+*B*. Optimizatorul verifică dacă agregatul *SUM*(*A*+*B*) care apare în cerere se potrivește cu un agregat de tipul *SUM*(*A*+*B*) sau *SUM*(*B*+*A*) în vizualizare.

Pentru a realiza verificările, sistemul *Oracle* convertește expresiile argument din funcția agregat în aceeași formă canonică. De exemplu, *A**(*B*-*C*), *A***B*-*C***A*, (*B*-*C*)**A* și -*A***C*+*A***B* pot fi convertite toate în aceeași formă canonică.

Bibliografie

1. Connolly T.M., Begg C.E., *Database Systems*: *A Practical Approach to Design, Implementation and Management*, 5th edition, Pearson Education, 2005
2. Dollinger R., Andron L., *Baze de date și gestiunea tranzacțiilor*, Editura Albastră, Cluj-Napoca, 2004
3. Inmon W.H., *Building the Data Warehouse*, 4th Edition, Wiley, 2005
4. Kimball R., *The Data Warehouse Toolkit*, 3rd Edition, Wiley, 2013
5. Kimball R., Ross M., Thornthwaite W., Mundy J., Becker B., *The Data Warehouse Lifecycle Toolkit*, 2nd Edition Wiley, 2008
6. Oracle and/or its affiliates, *Oracle Database Concepts*, 1993, 2024
7. Oracle and/or its affiliates, *Oracle Database Administrator's Guide*, 2001, 2025
8. Oracle and/or its affiliates, *Oracle Database Performance Tuning Guide*, 2013, 2025
9. Oracle and/or its affiliates, *Oracle Database SQL Language Reference*, 1996, 2025
10. Oracle and/or its affiliates, *Oracle Database PL/SQL Language Reference*, 1996, 2025
11. Oracle and/or its affiliates, *Oracle Database: SQL Tuning Workshop*, 2010, 2025
12. Oracle and/or its affiliates, *Oracle OLAP Customizing Analytic Workspace Manager*, 2006, 2019
13. Oracle and/or its affiliates, *Oracle OLAP DML Reference*, 1994, 2019
14. Oracle and/or its affiliates, *Oracle OLAP User's Guide*, 2003, 2019
15. Oracle and/or its affiliates, *Oracle Warehouse Builder Concepts*, 2000, 2021
16. Oracle and/or its affiliates, *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide*, 2000, 2021
17. Oracle and/or its affiliates, *Oracle Database Data Warehousing Guide*, 2001, 2021
18. Oracle University, *Oracle Database*: *PL/SQL Fundamentals*, *Student Guide*, 2009, 2025
19. Poe V., Klauer P., Brobst S., *Building A Data Warehouse for Decision Support*, 2nd Edition, Prentice Hall; 1997
20. Popescu I., Alecu A., Velcescu L., Florea (Mihai) G., *Programare avansată în Oracle9i*, Ed. Tehnică, 2004