Goodus 기술노트 [29 회]

Materialized View

Author	박제헌
Creation Date	**2008-04-26**
Last Updated	**2008-04-26**
Version	**1.0**

Version	변경일자	변경자(작성자)	주요내용
1	2008-04-26	박제헌	문서 최초 작성
2			
3			

Goodus
Breathing Life into Technology

Contents

Materialized View

- 본 문서는 window XP 장비, ORACLE 10.2.0.1 version 에서 test 되었다.

1.Materialized View 란?

- Materialized View(이하 Mview 라한다)는 oracle 8i 이전에는 snapshot 으로 제공되었다. sum,max,min 과 같이 cost 가 높은 쿼리들의 값을 미리 구하여 Mview 테이블에 저장하여 조회시 cost 를 감소시키고, 응답속도의 향상을 기대할수 있으며, Remote server 의 데이터를 복제하여 Local Table 저장하여 조회함으로써, 네트웍을 통한 부하를 절감할수있다.

  View 와는달리 실제 같은이름의 물리적인 테이블에 저장이 되며, dba_segments 에서 조회할수 있다.

```
select owner,object_name,object_type from dba_objects where object_name='M_DEPART_SAL';

OWNER OBJECT_NAME          OBJECT_TYPE
----- -------------------- -------------------
HR    M_DEPART_SAL         TABLE
HR    M_DEPART_SAL         MATERIALIZED VIEW
```

M_DEPART_SAL 이라는 Mview 를 생성하면 같은 이름의 Table 이 생성되는 것을 볼수있다.

2. Materialized View 생성

```
Create materialized view view_name
Build [immediate][deferred]
Refresh
[Complete][fast][force]
On [demand][commit]
Enable query rewrite
As
select ……
```

2.1 Build

Build immediate

- Mview를 생성하는 시점에 데이터도 같이 생성한다.

Build deferred

- Mview는 생성하되 데이터는 생성하지 않는다.

Build deferred 옵션을 사용하여 Mview를 생성한 예제.

```
Create materialized view test
Build deferred
Refresh
Complete
On demand
Enable query rewrite
As
select department_name D_NAME,sum(salary) SUM_SALARY from departments
d,employees e
 where d.department_id=e.department_id
   group by department_name;

  select owner,object_name,object_type from dba_objects where object_name='TEST';

  select * from test;
선택된 레코드가 없습니다.
```

2.2 Refresh

- mview는 Refresh옵션을 사용하여 소스테이블에서 변경된 데이터를 Mview테이블에 refresh 하여 데이터 동기화를 할수있다. Resfesh종류는 On Commit 과 On Demand방식이 있다.

On Commit- 소스테이블에 commit 이 일어날 때 refresh 된다.

On Demand(default) - package(dbms_mview,dbms_refresh, dbms_jobs)를 이용하여 수동으로 Refresh 한다.

2.2.1 Compleate

- Refresh 시 Mview 테이블을 truncate 하여 데이터를 모두 삭제하고 재 query 를 하여 데이터를 Refresh 하는 방법이다.

2.2.2 Fast

- 가장 많이 사용하는 옵션으로, mview log 를 이용하여, 변경된 데이터만을 적용하는 방법이다.

2.2.3 Force(default)

- Refresh 시 Fast 를 사용하고, Fast 방식이 실패시, Compleate 를 이용한다.

2.2.4 Never

- Refresh 를 전혀 사용하지 않는다.

3.Materialized View 활용

3.1 query rewrite

사용자가 Query 를 실행하였을경우, 같은 쿼리(혹은 비슷한쿼리)들은 이미 생성해놓은 mview 를 이용하여 cost 를 줄일수있다.

- Query rewrite 시 필요한 파라미터

OPTIMIZER_MODE : all_rows,first_rows,choose(통계치정보가있어야함)

QUERY_REWRITE_ENABLED : {true(default)|false|force}

true - Cost-based Rewrite

false - Rewrite 없음

force - 강제 Rewrite

QUERY_REWRITE_INTEGRITY

- stale_tolerated - 데이터 일관성이 보장되지 않아도 query rewrite 수행.
- trusted - 데이터 일관성이 보장된다고 가정하고 query rewrite 수행.
- enforced(default) - 변경전 sql 과 변경후 sql 의 결과값이 같다는 것을 옵티마이져가 확신하면 query rewrite 수행.

- Query rewrite 시 필요한 권한

grant query rewrite (자신이 만든 mview 에대해서 query rewrite 권한제공)

grant global query rewrite(자신이 만든 mivew 뿐아니라 참조하는 object 가 자신의 object 가 아니더라도 query rewrite 수행.

실습을통해 알아보자.

Query_rewrite_enabled= true 인경우

먼저, 고객이름과 그 고객이 산 제품, 개수를 구하는 쿼리를 작성한후 질의한결과의 실행계획이다.

```
select b.prod_name,c.CUST_FIRST_NAME||' '||c.CUST_LAST_NAME NAME,a.QUANTITY_SOLD
from sales a,products b,customers c
where a.prod_id=b.prod_id and a.cust_id=c.cust_id;
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop
|
-------------------------------------------------------------------------------------------------------
-
|   0 | SELECT STATEMENT      |           |   918K|    54M|       |  1943   (5)| 00:00:24 |       |    |
|*  1 |  HASH JOIN            |           |   918K|    54M|       |  1943   (5)| 00:00:24 |       |    |
|   2 |   TABLE ACCESS FULL   | PRODUCTS  |    72 |  2160 |       |     3   (0)| 00:00:01 |       |    |
|*  3 |   HASH JOIN           |           |   918K|    28M|  1736K|  1924   (5)| 00:00:24 |       |    |
|   4 |    TABLE ACCESS FULL  | CUSTOMERS | 55500 |  1083K|       |   331   (2)| 00:00:04 |       |    |
|   5 |    PARTITION RANGE ALL|           |   918K|    10M|       |   431  (10)| 00:00:06 |     1 |    28
|
|   6 |     TABLE ACCESS FULL | SALES     |   918K|    10M|       |   431  (10)| 00:00:06 |     1 |    28
```

|

---

query rewrite 의 기능을 활용하기 위해서 필요한 파라미터 및 값들을 셋팅한 후 mview 를 생성하였다

```
alter system set QUERY_REWRITE_ENABLED=TRUE;
grant query rewrite to sh;
grant global query rewrite to sh;
```

.

```
Create materialized view m_sale_list
Build immediate
Refresh
Complete
On demand
Enable query rewrite
As
select b.prod_name,c.CUST_FIRST_NAME||' '||c.CUST_LAST_NAME NAME,a.QUANTITY_SOLD
from sales a,products b,customers c
where a.prod_id=b.prod_id and a.cust_id=c.cust_id;
```

mview 생성후, 같은 쿼리를 질의해서 실행계획을 보면 m_sale_list mview 를 이용하여 cost 를 줄인 것을 확인할수있다.

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |   805K|    55M|  1405   (3)| 00:00:17 |
|   1 |  MAT_VIEW REWRITE ACCESS FULL| M_SALE_LIST |   805K|    55M|  1405   (3)| 00:00:17 |
```

단, 옵티마이저가 판단시, query rewrite 를 수행한 실행계획보다 옵티마이저가 세운 실행계획의 cost 가 더 낮다면, 기본적으로 query rewrite 를 수행하지않는다.

아래의 결과로 비교해보자.

Custermers 의 fist_name,last_name 컬럼을 빼고 query rewrite 를 이용한결과와 이용하지않

은 cost 의결과

```
Select b.prod_name,a.QUANTITY_SOLD
from sales a,products b,customers c
where a.prod_id=b.prod_id and a.cust_id=c.cust_id;
```

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name         | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |              |   918K|    41M|   750  (48)| 00:00:09 |       |       |
|   1 |  NESTED LOOPS        |              |   918K|    41M|   750  (48)| 00:00:09 |       |       |
|*  2 |   HASH JOIN          |              |   918K|    36M|   450  (13)| 00:00:06 |       |       |
|   3 |    TABLE ACCESS FULL | PRODUCTS     |    72 |  2160 |     3   (0)| 00:00:01 |       |       |
|   4 |    PARTITION RANGE ALL|             |   918K|    10M|   431  (10)| 00:00:06 |     1 |    28 |
|   5 |     TABLE ACCESS FULL| SALES        |   918K|    10M|   431  (10)| 00:00:06 |     1 |    28 |
|*  6 |   INDEX UNIQUE SCAN  | CUSTOMERS_PK |     1 |     5 |     0   (0)| 00:00:01 |       |       |
-----------------------------------------------------------------------------------------------------------
```

Query rewrite 힌트를 준경우

```
select/*+rewrite*/ b.prod_name,a.QUANTITY_SOLD
from sales a,products b,customers c
where a.prod_id=b.prod_id and a.cust_id=c.cust_id;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |   918K|    35M|  1409   (3)| 00:00:17 |
|   1 |  MAT_VIEW REWRITE ACCESS FULL| M_SALE_LIST |   918K|    35M|  1409   (3)| 00:00:17 |
-------------------------------------------------------------------------------------------
```

Custermers 의 fist_name,last_name 컬럼을 빼고 수행한경우는 query rewrite 를 이용하지 않은 경우가 cost 가 더 낮으므로 옵티마이져가 query rewrite 를 이용하지 않았다.이 경우는

QUERY_REWRITE_INTEGRITY 옵션을 바꾸어도 동일하게 적용되었다.

c.gender 컬럼이 추가된 경우에는 모드,옵션에 상관없이 query_rewrite 를 이용하지 않았다.

*참고 *

,sales 의 QUANTITY_SOLD 컬럼을 뺀 경우는 힌트를 주지 않아도 query rewrite 을 이용하였다

```
select b.prod_name,c.CUST_FIRST_NAME||' '||c.CUST_LAST_NAME NAME
from sales a,products b,customers c
where a.prod_id=b.prod_id and a.cust_id=c.cust_id;
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name         | Rows  | Bytes | Cost (%CPU)| Time
|
-------------------------------------------------------------------------------------------------
-
|   0 | SELECT STATEMENT                   |              |   918K|    40M|  1405   (3)| 00:00:17
|
|   1 |  MAT_VIEW REWRITE ACCESS FULL| M_SALE_LIST |   918K|    40M|  1405   (3)| 00:00:17
|
-------------------------------------------------------------------------------------------------
-
```

Query_rewrite_enabled= force 인경우..

```
Select b.prod_name,a.QUANTITY_SOLD
from sales a,products b,customers c
where a.prod_id=b.prod_id and a.cust_id=c.cust_id;
```

query_rewrite_enable 모드가 force 이므로 cost 가 높더라도 query_rewrite 를 이용한다.

```
-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |   934K|    25M|  1409   (3)| 00:00:17 |
|   1 |  MAT_VIEW REWRITE ACCESS FULL| M_SALE_LIST |   934K|    25M|  1409   (3)| 00:00:17 |
-------------------------------------------------------------------------------------------
```

Query rewrite 제약사항

- remote table 은 query rewrite 사용불가능. local table 에서만 사용가능.
- sys user 의 detail-table 이나 mview 는 지원하지 않음
- mview 에 group by 절이 사용되었다면 그 컬럼은 select 문에 명시해야함.
- avg(avg(x), avg(a) + avg(b) 연산은 지원하지 않음
- connect by 절은 허용하지 않음.

3.2 Materialized view for replication

- 원격지에있는 Data 를 로컬로 복사를 하여 원격지의 Data 를 억세스 하지 않고, 로컬 Data 를 조회함으로, 네트웍 부하를 줄일수있으며, mview log 를 이용하여 Data 를 동기화할 수 있다.

employee table 에 job_id 별 salary 합계를 구하는 쿼리를 만들어보자.

먼저, 원격지 DB employee table 에 mview log 를 생성한다.

```
create materialized view log on employees
with rowid (salary,job_id)
including new values
/
```

로컬에서 db_link 를 생성하고, mview 를 생성한다.

```
create database link hr connect to hr identified by hr  using 'TEST';
```

```
create materialized view m_employees
refresh fast start with sysdate next sysdate +(1/60/24)
as select job_id,sum(salary) from employees@HR group by job_id;
```

user_job 에 등록되어 1 분마다 refresh 됨을 알수있다.

```
select job,last_date,next_date,what from user_jobs;
```

```
JOB LAST_DATE              NEXT_DATE              WHAT
--- ---------------------- ---------------------- ------------------------
 22 2008-apr-13 18:16:42   2008-apr-13 18:17:42   dbms_refresh.refresh('"TE
                                                  STP"."M_EMPLOYEES"');
```

데이터 insert 후 mview log 를 통해, mview 가 refresh 된 것을 알수있다.

```
Insert into EMPLOYEES  (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER,
   HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID,
   DEPARTMENT_ID)  Values(219,  'test',  'test',  'test',  '650.507.9822',TO_DATE('05/23/1998  00:00:00',
'MM/DD/YYYY HH24:MI:SS'), 'SH_CLERK', 3000, NULL, 124,  50);
COMMIT;
Select * from m_employees;
```

```
JOB_ID               SUM(SALARY)
-------------------- -----------
AC_MGR                     24000
AC_ACCOUNT                 16600
```

```
JOB_ID               SUM(SALARY)
-------------------- -----------
AC_MGR                     24000
AC_ACCOUNT                 16600
```

IT_PROG	28800	IT_PROG	28800
ST_MAN	36400	ST_MAN	36400
AD_ASST	8800	AD_ASST	8800
PU_MAN	11000	PU_MAN	11000
SH_CLERK	72100	SH_CLERK	75100
AD_VP	51000	AD_VP	51000
FI_ACCOUNT	39600	FI_ACCOUNT	39600
MK_MAN	26000	MK_MAN	26000
PR_REP	20000	PR_REP	20000

- 90 만건의 데이터(180M 의 sql 문,로컬 insert 시간 30 여분)를 mview log 를 이용하여 동기화 시키는 TEST 를 수행보았다.

refresh 주기는 1 분,로컬테이블에서 데이터가 연속적으로 insert 되는 상태에서의 원격지테이블 refresh 상태에 대해서 알아보기 위해 진행하였다.

1. 로컬에서 테이블 껍데기만 생성한다.

```
create table sales_bk_bk as select * from sales where 1=2;
```

2.로컬에서 mview log 를 생성한다.

```
create materialized view log on sales_bk_bk
with rowid
```

Mview log

sales_bk_bk 테이블에 mview log 를 생성하면 mlog$_sales_bk_bk 이라는 테이블이 생성된다.

이 테이블은 sales_bk_bk 테이블에 변화가 있을 때 그 정보를 저장해두었다가, 동기화시 사용되고 비워진다.

```
SQL> desc mlog$_sales_bk_bk;
 이름                                      널?      유형
 ----------------------------------------- -------- ----------------------------
 M_ROW$$                                            VARCHAR2(255)
 SNAPTIME$$                                         DATE
 DMLTYPE$$                                          VARCHAR2(1)
 OLD_NEW$$                                          VARCHAR2(1)
 CHANGE_VECTOR$$                                    RAW(255)
```

```
M_ROW$$           SNAPTIME       D O CHANGE_VECTOR$$
AAAM8sAAEAAAAL2ADE  00/01/01      D O         00
```

3.원격지에서 mview를 refresh 주기를 1분으로 생성한다.

```
create materialized view m_sales_bk_bk
refresh fast with rowid start with sysdate next sysdate +(1/60/24)
as select * from sales_bk_bk@test;
```

4.원격지데이터베이스에 job에 등록되었는지 확인한다.

```
select job,last_date,next_date,what from dba_jobs;
JOB LAST_DATE                NEXT_DATE                WHAT
---------- -------------------- -------------------- ----------------------------------------
    181                          2008-4월 -17 20:00:05     dbms_refresh.refresh('"SH"."M_SALES_BK"');
```

*refresh 가 자동으로 수행되지 않는 경우는(수동으로 refresh 는잘됨).mview 를 생성한 테이블의 job_queue_processes 가 0 으로 설정되어있기때문이다. 이값을 적당한값으로 조정 후 job 등록확인하면 job 프로세서가 잘 동작되고있음을 알수있다.

```
JOB LAST_DATE           NEXT_DATE           WHAT
---------- -------------------- -------------------- ----------------------------------------
        8 2008-apr-22 17:52:42 2008-apr-22 17:53:42 dbms_refresh.refresh('"TESTP"."M_SALES_BK_BK"');
```

5.sales_bk_bk 에 DATA insert!

결과, sales_bk_bk 테이블에 데이터가 insert 되면, mlog$$_sales_bk_bk 에 그 값들이 저장되고, mview 테이블에 값이 동기화 후 지워짐을 알수있다

```
select count(*) from sales_bk_bk
COUNT(*)
----------
    178014
SQL> /
  COUNT(*)
----------
    178014
SQL> /
  COUNT(*)
```

```
select count(*) from
MLOG$_SALES_BK_BK
SQL> /
  COUNT(*)
----------
     17000
SQL> /
  COUNT(*)
----------
     18000
```

```
select count(*) from
m_sales_bk_bk
  COUNT(*)
----------
    161014
SQL> /
  COUNT(*)
----------
    161014
SQL> /
```

```
----------
    179014
.....(중략)



COUNT(*)
----------
    987014
```

```
SQL> /

  COUNT(*)
----------
      3000
SQL> /

  COUNT(*)
----------
      4000
.....(중략)
  COUNT(*)
----------
         0
```

```
  COUNT(*)
----------
    177014
SQL> /

  COUNT(*)
----------
    177014
.....(중략)


  COUNT(*)
----------
    987014
```

3.3 Materialized view for DW

- sum,avg,join 등 집합연산등의 query 의 결과를 미리 구해놓으므로써, cost 를 줄일수있다.

salary 합계를 구하는 쿼리를 수행한 실행계획과, mivew 생성후 mview 를 이용한 쿼리 실행계획을 비교해본다.

salary 합계를 구하는 query

```
select department_name D_NAME,sum(salary) SUM_SALARY from hr.departments
d,hr.employees e
 where d.department_id=e.department_id
 group by department_name;
```

실행계획

```
-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |    27 |   540 |     5  (20)| 00:00:01 |
|   1 |  HASH GROUP BY               |             |    27 |   540 |     5  (20)| 00:00:01 |
|   2 |   NESTED LOOPS               |             |   117 |  2340 |     4   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL         | EMPLOYEES   |   118 |   708 |     3   (0)| 00:00:01 |
|   4 |    TABLE ACCESS BY INDEX ROWID| DEPARTMENTS |     1 |    14 |     1   (0)| 00:00:01 |
|* 5 |     INDEX UNIQUE SCAN        | DEPT_ID_PK  |     1 |       |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------
```

mview 생성후 실행계획

```
Create materialized view m_depart_sal
Build immediate
Refresh
Complete
On demand
Enable query rewrite
As
select department_name D_NAME,sum(salary) SUM_SALARY from hr.departments
d,hr.employees e
 where d.department_id=e.department_id
 group by department_name;
```

```
Select * from m_depart_sal;
-------------------------------------------------------------------------------
```

```
| Id  | Operation            | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |              |    11 |   330 |     2   (0)| 00:00:01 |
|   1 |  MAT_VIEW ACCESS FULL| M_DEPART_SAL |    11 |   330 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

Mview 를 생성하기 전보다 cost 가 훨씬 줄어든 것을 확인할수있다.

이는 query 결과 값을 미리 구하여 mview table 에 저장되는데 mivew 생성시의 스크립트를 보면 쉽게 확인할수있다.

```
CREATE TABLE M_DEPART_SAL
(
  D_NAME      VARCHAR2(30 BYTE)               NOT NULL,
  SUM_SALARY  NUMBER
)
…중략
Insert into M_DEPART_SAL
   (D_NAME, SUM_SALARY)
 Values
   ('Administration', 8800);
Insert into M_DEPART_SAL
   (D_NAME, SUM_SALARY)
 Values
   ('Accounting', 40600);

.......
```

5. Materialized View TroubleShooting

mview log 와 관련된 에러

ora-12032 : "HR"."EMPLOYEES"의 구체화된 뷰 로그에서 ROWID 열을 사용할 수 없음
원인 : m-view log 생성시 rowid 열 옵션을 지정하지 않았기 때문.
조치 : with rowid including new values 을 추가해준다.

ORA-12033: "HR"."EMPLOYEES"의 구체화된 뷰 로그에서 필터 열을 사용할 수 없음
원인 : m-view log 생성시 m-view 에서 사용할 필터열을 지정하지 않았기 때문
조치 : with rowid(miview 에서 사용할 필터열 지정) including new values

ORA-32401: "HR"."EMPLOYEES"에 대한 구체화된 뷰 로그에 새 값이 없음
원인 : m-view log 생성시 including new values 값을 추가해주지 않았기 때문
조치 : including new values 값을 추가

권한관련 에러
ORA-01031: "HR"."EMPLOYEES"에 대한 구체화된 뷰 로그에 새 값이 없음
원인 : mview 생성 권한과 global query 권한이 없기때문
조치 :권한부여 grant global query rewrite,create snapshot to user ;

mview 생성시 error
ORA-12014: 테이블 'SALES_BK_BK'은(는) 기본 키 제약조건을 포함하고 있지 않습니다
조치 : with rowid 옵션을 주어서 해결한다.