실제화뷰

태그

리프레시

💡 마스터 테이블에 변경이 일어날때 이를 반영하도록 실체화 뷰를 갱신하는 동작을 리프레시라고 한다. 리프레시를 하면 실체화 뷰의 데이터는 질의의 결과와 일치 된다. 리프레시는 실체화뷰를 생성하는 조건에 따라 자동으로 수행될수 있으며 dbms_myview패키지의 리프래시함수를 이용하여 수동으로 수행할수도 있다. 리프레시는 완전리프레시와 빠른 리프레시가 있다.

```
완전 리프레시는 실체화뷰에 있는 기존의 데이터를 모두 삭제한 후 실체화뷰를 질의한 질의를 다시 수행하여 그 결과를 실체화뷰에 저장하는 방식이다.
빠른 리프레시는 마스터테이블의 변경된 부분만을 실체화뷰에 반영하는 방식이다. 완전 리프레시보다 빠르다.
```

일반적인 제약조건

💡 빠른 리프레시를 사용하기 위한 제약조건은 다음과 같다. 실체화뷰를 정의하는 질의는 다음의 제약조건을 만족해야한다.

```
실체화 뷰는 sysdate와 rownum과 같이 반복할수 없는 표현식을 포함하면 안된다.
실체화뷰는 long 또는 long raw데이터타입을 포함하면 안된다.
select리스트에 부질의를 포함하면 안된다.
select리스트에 분석함수를 포함하면 안된다.
having절에 부질의를 포함하면 안된다.
any, all, not exists를 포함하면 안된다.
start with connect by절을 포함하면 안된다.
서로 다른 원격에 있는 테이블을 포함하면 안된다. 즉 쿼리에 참가하는 모든테이블은 같은 서버에 있어야 한다.
원격 테이블이 포함한 경우 해당 서버가 tibero일경우만 된다.
union 등의 set연산자를 포함하면 안된다.
refresh on commit 실체화뷰를 정의한 질의는 원격 테이블을 포함하면 안된다.
```

집합 함수를 포함한 제약 조건

빠른 리프레시에어의 일반적인 제약조건을 모두 포함한다.
실체화 뷰의 모든 참조 테이블에는 실체화 뷰의 로그가 있어야하며 실체화 뷰의 로그는 다음의 제약조건을 만족해야 한다.
실체화뷰에서 참조하는 모든 칼럼을 포함해야 한다.
sequence, rowid와 including new values조건이 있어야 한다.
함수는 sum, count,avg,stddev,variance,min,max함수만 지원된다.
count(*)는 항상 포함해야 한다.
집합함수는 항상 표현식의 최상위에 있어야한다. 단, avg(count(x)), count(x) * count(x)는 허용하지 않는다. avg(expr)에 대응되는 count(expr)이 있어야 한다. select리스트에는 모든 group by칼럼이 잇어야한다. cube,rollup은 허용하지 않는다.

집단함수 count(expr), min(expr), max(expr), sum(expr), 필요집단함수 count(expr) 집단함수 sum(col) col에 not null제약 avg(expr), count(expr) , stddev(expr) - sum(expr), count(expr), variance(expr) -sum(expr), count(expr)

질의다시쓰기

실체화뷰는 가장 큰 장점은 질의 다시 쓰기 기능을 사용할 수 있다는 것이다. 질의 다시 쓰기는 주어진 질의를 분석한 후 실체화 뷰를 사용하여 동일한 결과를 내는 새로운 질의를 생성하는 기능이다. 실체화뷰가 복잡한 조인이나 집단함수의 결과로 정의되어있다면 질의다시쓰기를 통해 실행시간을 크게 단축할수있다. 질의 다시쓰기는 사용자가 명시한 명령 없이도 질의 최적화하기에 의해 수행되므로 실체화뷰를 인덱스처럼 사용할 수 있다. insert, delete,update,merge의 대상이 되는 부질의를 제외한 모든 select문에 동작한다.

동작조건
질의 다시 쓰기 기능을 동작하기 위한 조건은 다음과 같다.
실체화뷰를 생성할때 enable query rewrite옵션을 추가해야 한다. query_rewirte_enabled파라미터의 값이 true나 force로 설정되어 있어야 한다. 질의 다시 쓰기가 query_Rewirte_integrity파라미터의 설정을 만족시켜야 한다. qury_rewrite_integrity = enforced | stable_totalrated

> enforced : 최신 데이터가 있는 실체화만 사용하여 원본 질의와 동일한 결과를 보장한다.

> stale_totalrated 최신 데이터 변경 내용이 반영되지 않은 실체화 뷰를 사용하기 때문에 원본 질의와 동일한 결과를 보장하지 않는다. 단, 질의 다시쓰기가 될 가능성은 커진다.

💡 동작방식 : 질의다시쓰기를 하려면 원본 질의와 사용할 실체화뷰의질의를 비교해서 원본 질의가 실체화뷰를 사용해서 얻어질수있는지를 확인해야 한다. 현재는 완전 문자열 비교와 문자열 비교방식을 지원한다.

```
완전 문자열 비교 : 원본 질의의 사용할 실체화뷰의 질의에대해 전체문자열을 비교한다. 이때 공백이나 대소문자의
 차이는 무시된다.
CREATE MATERIALIZED VIEW MV_SUM_SALARY ENABLE QUERY REWRITE AS SELECT DNAME, SUM(SAL)
 FROM DEPT, EMP WHERE DEPT.DEPTNO = EmP.DEPTNO
GROUP BY DNAME;
다음과 같이 질의 다시 쓰기가 동작한다.
SELECT * FROM MV_SUM_SALARY;
```

```
부분 문자열 비교
원본 질의와 사용할 실체화 뷰의 질의에 대해 FROM절부터 ORDER BY 절 앞까지 (존재할 경우)의 문자열을 비교하
고 SELECT 리스트와 ORDER BY절의 연산식에서 사용되는 칼럼의 적합성을 조사한다. 칼럼의 적합성은 원본 질의의
SELECT 리스트 칼럼이 실체화 뷰의 칼럼을 조합해서 얻어질수 있는지를 검사한다. 이때 조인에 의해 같아지는 칼럼
을 처리한다.
예를들어
CREATE MATERIALIZED VIEW MV_JOIN_DEPT_EMP ENABLE QUERY REWRITE AS
    SELECT ENAME, DNAME, DEPT.DEPTNO FROM DEPT, EMP WHERE DEPT.DEPTNO = EMP.DEPTNO;
다음과 같이 질의를 실행한다면
SELECT ENAME, DNAME, EMP.DEPTNO FROM DEPT, EMP WHERE DEPT.DEPTNO = EMP.DEPTNO;
다음과 같이 질의 다시 쓰기가 동작하게 된다.
SELECT ENAME, DNAME, DEPTNO FROM MV_JOIN_DEPT_EMP;
또한 연산의 교환, 결합, 분배 법칙을 이용하여 동등한 연산식의 칼럼을 처리한다. 예를들어

CREATE MATERIALIZED VIEW mymv ENABLE QUERY REWRITE AS
    SELECT a*(c+4)+b c1, sum(c+a)+1 c2 FROM base GROUP BY a*(c+4)+b, b+4*a+a*c;
다음과 같은 질의를 실행한다면,

SELECT b+4*a+a*c+(1+sum(a+c))*7 FROM base GROUP BY a*(c+4)+b, b+4*a+a*c;
다음과 같이 질의 다시 쓰기가 동작하게 된다.

SELECT c1+c2*7 FROM mymv;
집단 함수를 사용할때 집단 함수의 적합성을 조사하는 방법은 다음과 같다. 해당되는 집단 함수가 없어도 다른 집단
```

함수를 조합하여 원하는 집단 함수를 얻을 수 있다. 예를 들어 AVG(SALARY) 는 SUM(SALARY)/COUNT(SALARY)로 구할 수 있다.

목표 집단 함수	필요 집단 함수
AVG(expr)	SUM(expr), COUNT(expr)
SUM(expr)	AVG(expr), COUNT(expr_
STDDEV(expr)	VARIANCE(expr)
	(STDDEV_SAMP(expr), STDDEV_POP(expr), VAR_SAMP(expr),VAR_POP(expr)) 중 하나, COUNT(expr)
	SUM(expr * expr), SUM(expr), COUNT(expr)
STDDEV_SAMP(expr)	VAR_SAMP(expr)
	STDDEV(expr), STDDEV_POP(expr), VARIANCE(expr), VAR_POP(expr)) 중 하나, COUNT(expr)

비용기반 최적화
외부질의와 다시 쓰여진 질의로부터 비용 기반 최적화 기법에 의해 각각의 실행계획이 생성되고 최종으로 두 시행계획 중에서 비용이 적은 쪽이 선택된다. 질의 다시쓰기는 실행비용에 따라 최종선택이 달라지기 때문에 원본 질의가 참조하는 테이블과 실체화뷰를 저장하고 있는 테이블의 통계정보로 생성한다.

원격 저장소를 가진 실체화뷰

💡 이기종 데이터베이스에서 실체화뷰를 이용하여 tibero에 있는 베이스 테이블의 데이터를 동기화하고 싶을때 사용하는 기능이다. 이기종데이터베이스에서 tibero에 있는 베이스 테이블을 조회하는 실체화뷰를 생성해도 빠를 리프레시는 수행이 불가능하다. 따라서 이 기능을 통해 실체화 뷰 관리를 tibero쪽 데이터 베이스가 담당하게 하고 실제 저장소는 이기종 데이터베이스에 위치하게 하여 조회가 가능하게 한다. 현재 이기종 데이터베이스는 oracle만 지원한다. 아래 설명을 하기 전에 베이스 테이블이 위치한 데이터베이스를 A, 실체화 뷰 저장소 테이블이 위치할 오라클 서버를 B이라 가정한

실체화 뷰 생성 사전 작업
실체화 뷰 생성에 하기 전에 아래와 같은 사전 작업이 필요하다. 베이스 테이블이 위치한 데이터베이스를 A, 실체화 뷰 저장소 테이블이 위치할 오라클 서버를 B라 가정한다.
B의 해당 계정에 Tibero에서 제공하는 $TB_HOME/scripts/mvview_remote_install.sql스크립트를 수행하여 기능에 필요한 PSM패키지를 설치한다.
A에서 B로의 데이터베이스 링크를 생성한다.
PREBUILT옵션만 지원하므로 저장소 테이블을 사전에 생성한다

실체화 뷰 생성

A에서 실체화 뷰를 생성하지만 실제 조회가 가능한 저장소 테이블은 B에 있다.
CREATE MATERIALIZED VIEW를 생성할때 AT dblink_name 속성을 추가하여 a에서 b로의 데이터 베이스 링크를 지정한다.
리프레시 및 저장소 테이블 조회
실체화 뷰 리프레시는 A에서 수행한다. 리프레시가 된 결과는 B의 저장소 테이블에 저장된다. 그 외의 사항에 대해서는 일반 실체화 뷰와 동일하다.
제약사항
원격 저장소를 가진 실체화뷰는 아래와 같은 제약사항이 있다.
베이스 테이블이 이기종 데이터베이스에 있고 저장소 테이블을 Tibero에 저장하는 실체화 뷰는 지원하지 않는다.
빠른 리프레시의 경우 실체화 뷰 질의가 집합함수가 없는 단일 베이스 테이블에 대한 질의어야 한다.
그 외의 사항은 일반 실체화 뷰 질의의 제약조건과 동일하다.

```
1
2
3
4
5
6
7
8
9
10
CREATE TABLE MVIEW_TEST
(
 COL1 VARCHAR(10),
 COL2 VARCHAR(10),
 COL3 VARCHAR(10),
 COL4 VARCHAR(10),
 CONSTRAINT PK_MVIEW_TEST PRIMARY
KEY(COL1)
)
TABLESPACE TESTDATA;
?
1
```

```
2
3
4
5
6
7
8
9
10
CREATE TABLE MVIEW_TEST_M
(
 COL1 VARCHAR(10),
 COL2 VARCHAR(10),
 COL3 VARCHAR(10),
 COL4 VARCHAR(10),
 CONSTRAINT PK_MVIEW_TEST_M
PRIMARY KEY(COL1)
)
TABLESPACE TESTDATA;
?
1
2
3
4
5
6
7
8
9
10
11
12
CREATE MATERIALIZED VIEW LOG ON MVIEW_TEST_M
TABLESPACE TESTDATA
WITH PRIMARY KEY
EXCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW MVIEW_TEST (COL1,COL2,COL3,COL4)
ON PREBUILT TABLE WITH REDUCED PRECISION
REFRESH FAST ON DEMAND
WITH PRIMARY KEY
ENABLE QUERY REWRITE
AS
SELECT COL1, COL2, COL3, COL4 FROM MVIEW_TEST_M;
나. 99,999 ROWS 성능 비교
1. 성능 개선 전  2. 성능 개선 후
?
1
2
3
4
5
6
7
8
9
10
11
```

```
12
13
14
15
16
17
18
19
SQL> set timing on;

SQL> INSERT INTO MVIEW_TEST_M
 SELECT LEVEL AS COL1
 ,ROUND(DBMS_RANDOM.VALUE(1,1000),0)AS COL2
 ,ROUND(DBMS_RANDOM.VALUE(1,1000),0)AS COL3
 ,ROUND(DBMS_RANDOM.VALUE(1,1000),0)AS COL4
 FROM DUAL CONNECT BY LEVEL <100000;

99999 rows inserted.
Total elapsed time 00:00:17.532112

SQL> COMMIT;
Commit completed.
Total elapsed time 00:00:00.006475

SQL> EXEC DBMS_MVIEW.REFRESH('MVIEW_TEST', 'F');
PSM completed.
Total elapsed time 00:00:10.377328
?
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
SQL> set timing on;

SQL> INSERT INTO MVIEW_TEST_M
 SELECT LEVEL AS COL1
 ,ROUND(DBMS_RANDOM.VALUE(1,1000),0)AS COL2
 ,ROUND(DBMS_RANDOM.VALUE(1,1000),0)AS COL3
 ,ROUND(DBMS_RANDOM.VALUE(1,1000),0)AS COL4
 FROM DUAL CONNECT BY LEVEL <100000;

99999 rows inserted.
Total elapsed time 00:00:03.827569
```

```
SQL> COMMIT;
Commit completed.
Total elapsed time 00:00:00.016289

SQL> EXEC DBMS_MVIEW.REFRESH('MVIEW_TEST', 'F');
PSM completed.
Total elapsed time 00:00:15.657918
다. 999,999 ROWS 성능 비교
1. 성능 개선 전  2. 성능 개선 후
?
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
SQL> set timing on;

SQL> INSERT INTO MVIEW_TEST_M
 SELECT LEVEL AS COL1
 ,ROUND(DBMS_RANDOM.VALUE(1,1000),0)AS COL2
 ,ROUND(DBMS_RANDOM.VALUE(1,1000),0)AS COL3
 ,ROUND(DBMS_RANDOM.VALUE(1,1000),0)AS COL4
 FROM DUAL CONNECT BY LEVEL <1000000;

999999 rows inserted.
Total elapsed time 00:02:59.010955

SQL> COMMIT;
Commit completed.
Total elapsed time 00:00:00.023046

SQL> EXEC DBMS_MVIEW.REFRESH('MVIEW_TEST','F');
PSM completed.
Total elapsed time 00:01:52.257876
?
1
2
3
4
5
6
7
8
```

```
9
10
11
12
13
14
15
16
17
18
19
SQL> set timing on;

SQL> INSERT INTO MVIEW_TEST_M
 SELECT LEVEL AS COL1
 ,ROUND(DBMS_RANDOM.VALUE(1,1000),0)AS COL2
 ,ROUND(DBMS_RANDOM.VALUE(1,1000),0)AS COL3
 ,ROUND(DBMS_RANDOM.VALUE(1,1000),0)AS COL4
 FROM DUAL CONNECT BY LEVEL <1000000;

999999 rows inserted.
Total elapsed time 00:00:39.982500

SQL> COMMIT;
Commit completed.
Total elapsed time 00:00:00.022666

SQL> EXEC DBMS_MVIEW.REFRESH('MVIEW_TEST','F');
PSM completed.
Total elapsed time 00:02:37.267385
```