Result Cache 동작원리 및 활용방안

㈜엑셈 컨설팅본부/DB컨설팅팀 김 철환

개요

ORACLE DBMS 를 사용하는 시스템에서 QUERY 성능은 무엇보다 중요한 요소 중 하나이며 그 성능과 직접적인 관련이 있는 것이 I/O 이다.

많은 건수를 ACCESS 해야만 원하는 결과값을 얻을 수 있는 QUERY 을 실행하게 된다면 BLOCK I/O 가 많이 발생하게 된다. 이런 이유로 QUERY 의 성능은 좋지 못 할 것이다. 코드 테이블과 같이 적은 데이터를 동시에 많은 사용자들이 조회 한다면 BUFFER CACHE 에서 원하는 BLOCK 을 찾아 결과를 전송 하겠지만 동시에 많은 사용자들이 사용하기 때문에 HOT BLOCK 이 되고 ORACLE 대기 이벤트가 발생하여 빠른 성능을 기대하기가 어렵다.

I/O 에 따라 QUERY 의 성능이 좌우 되기 때문에 I/O 을 최소 한으로 발생시키고 QUERY 를 실행한다면 성능은 향상될 것이다. ORACLE 11G 에서 I/O 의 개선을 위해 선보인 기능이 RESULT CACHE 기능이다.

RESULT CACHE 는 SHARED POOL 에 RESULT CACHE MEMORY 로 불리는 영역에 SQL 및 PL/SQL FUNCTION 의 결과를 저장하고, 이후 동일 QUERY 조회 시 RESULT CACHE 에 저장되어 있는 QUERY 결과 값을 그대로 활용하는 기능이다. 반복적으로 동일한 결과 값을 조회하는 QUERY 에서 사용되며 많은 데이터를 처리하여 적은 결과 건수를 보고자 할 때 BLOCK I/O 을 발생시키지 않고 결과를 전송하기 때문에 조회 성능에 있어서 매우 빠르다.

이제부터 RESULT CACHE QUERY 의 동작 원리와 설정 방법을 알아 본 후에 기능 활용에 대해 알아 보도록 하자.

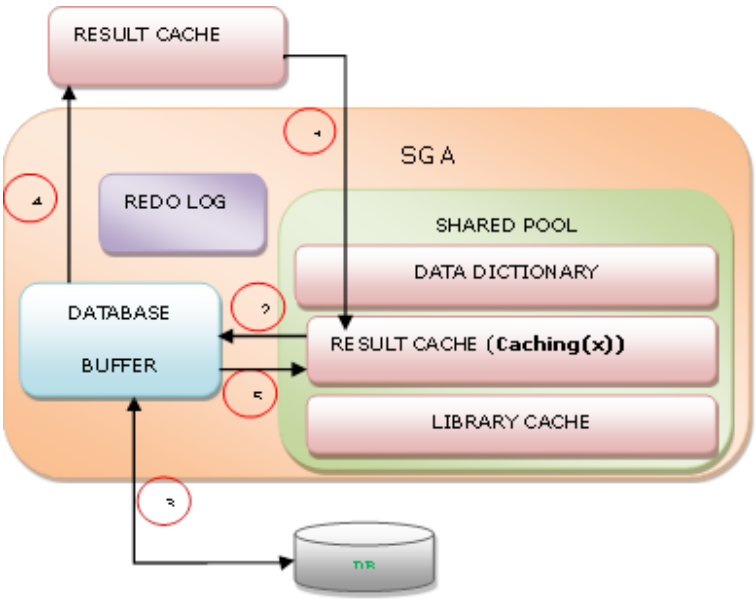
RESULT CACHE QUERY 동작 원리

일반적으로 QUERY 가 수행되어 BUFFER CACHE 에 원하는 블록이 존재한다면 요청한 세션에게 블록을 전송한다. 만약 BUFFER CACHE 영역에 존재하지 않는다면 DISK I/O 발생 후 BUFFER CACHE 에 BLOCK 을 올려 놓고 요청한 세션에게 결과 값을 전송한다.

RESULT CACHE 기능은 일반적인 QUERY 동작 방식과 같지만 그 결과 값을 CACHING 한다는 점이 다르다.

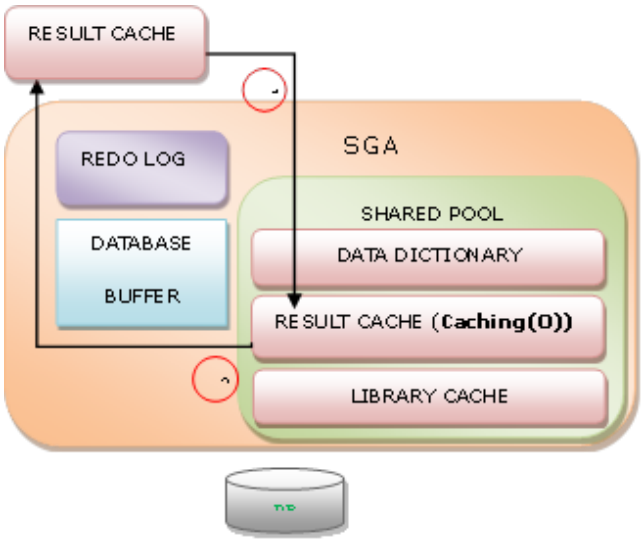
RESULT CACHE QUERY 최초 실행.

1. RESULT CACHE QUERY 가 실행되면 가장 먼저 SHARED POOL 영역의 RESULT CACHE 메모리에서 OBJECT 의 CACHING 여부를 확인한다.
2. OBJECT 가 CACHING 되어 있지 않다면 BUFFER CACHE 에서 BLOCK 을 찾는다.
3. BUFFER CACHE 에 원하는 BLOCK 이 존재하지 않으면 DISK 에서 BLOCK 을 읽어 BUFFER CACHE 에 전송한다.
4. 그 결과 값을 QUERY 한 세션에 전송한다.
5. 마지막으로 RESULT CACHE 영역에 QUERY 결과 값을 저장한다.



RESULT CACHE QUERY 반복 실행.

1. RESULT CACHE QUERY 가 실행되면 가장 먼저 SHARED POOL 영역의 RESULT CACHE 메모리에서 OBJECT 의 CACHING 여부를 확인한다.
2. CACHING 된 정보가 존재 하면 I/O 을 발생시키지 않고 CACHING 된 결과 값을 요청한 세션에게 전송한다.



RESULT CACHE 기능 설정 및 해지

RESULT CACHE 기능 설정하기

PARAMETER 설정으로 RESULT CACHE 기능을 사용할 수 있다. 설정에 필요한 주요 PARAMETER 들을 살펴보도록 하겠다.

RESULT_CACHE_MODE

RESULT_CACHE_MODE PARAMETER 값에 따라 2 가지 MODE 의 설정이 가능한데 MANUAL 과 FORCE 이다. MANUAL 일 경우에는 SQL 마다 /*+ RESULT_CACHE */ HINT 을 주어

RESULT CACHE 기능을 사용할 수 있고 FORCE 일 경우에는 모든 SQL 이 RESULT CACHE 의 대상이 된다. 단 /*+ NO_ RESULT_CACHE */ HINT 을 제외 된다. RESULT_CACHE_MODE PARAMETER 의 DEFAULT 값은 MANUAL 이다.

RESULT_CACHE_MAX_SIZE

RESULT CACHE 기능을 사용 하려면 RESULT_CACHE_MAX_SIZE 값도 명시적으로 지정 되어 있어야 한다. 이 PARAMETER 의 최대 값은 SHARED POOL 의 최대 75%까지만 설정할 수 있다.

RESULT_CACHE_MAX_RESULT

하나의 SQL 에 결과 집합에 대한 전체 캐시 영역에서 최대 할당할 수 있는 메모리 크기이며 DEFAULT 값은 5%이다.

RESULT_CACHE_REMOTE_EXPIRATION

REMOTE DATABASE OBJECT 의 보관 주기를 시간(분) 지정이 가능하며 DEFAULT 값은 0 이다. The default is 0 which means that resultsets dependant on remote OBJECT

RESULT CACHE 기능 해지하기

RESULT CACHE 기능을 해지하는 방법에는 특정 INSTANCE 가 RESULT CACHE 기능을 사용할 수 없게 설정하는 방법과 RESULT CACHE 기능은 사용이 가능하지만 CACHE 에서 OBJECT 을 해지 하는 2 가지 방법이 있다. INSTANCE 가 RESULT CACHE 기능을 사용하지 않으려면 RESULT_CACHE_MODE 값을 0 의 값으로 설정 하고, INSTANCE 을 재 기동 하면 된다. RESULT CACHE 기능은 사용 가능하지만 CACHING 되어 있는 OBJECT 을 해지하는 방법은 RESULT CACHE 패키지를 이용해 해지할 수 있다. CACHE 에서 해지하는 방법은 예제를 통해 알아 보도록 하겠다.

CACHING 된 OBJECT 해지하는 방법

RESULT CACHE 기능을 사용 하다가 CACHING 된 여러 OBJECT 중에서 특정 OBJECT 만 CACHE 에서 해지하려면 DBMS_RESULT_CACHE.INVALIDATE 패키지를 사용하여 해지할 수 있다.

[QUERY 실행]

```
SELECT  ID,
        TYPE,
        STATUS,
        BUCKET_NO,
        HASH,NAME
FROM    V$RESULT_CACHE_OBJECT;
```

[결과 값]

```
ID    TYPE  STATUS  BUCKET_NO  HASH     NAME
----- ---- ------- ---------- --------- ----------------------------
 0 Dependency Published   660 319061 DBAADM.TB_RC_TEST_YYYYMMDD
 1 RESULT     Published  2011 159411      SELECT       /*+ RESULT_CACHE */
                                             SUBSTR(SDATE,1,6) SDATE,
                                             PROD,
                                             SUM(AMT1) AMT1,
                                             SUM(amt2) AMT2,
                                             SUM(AMT3) AMT3
                                          FROM          TB_RC_TEST_YYYYMMDD
                                               GROUP BY SUBSTR(SDATE,1,6),
                                                        PROD
```

[패키지 실행]

```
EXEC DBMS_RESULT_CACHE.INVALIDATE('DBAADM',' TB_RC_TEST_YYYYMMDD')
```

[QUERY 실행]

```
SELECT
        ID,
        TYPE,
        STATUS,
        BUCKET_NO,
        HASH,NAME
FROM    V$RESULT_CACHE_OBJECT;
```

[결과 값]

```
no rows selected.
```

[패키지 실행]

```
EXEC DBMS_RESULT_CACHE.FLUSH
```

[QUERY 실행]

```
SELECT
       ID,
       TYPE,
       STATUS,
       BUCKET_NO,
       HASH,NAME
FROM   V$RESULT_CACHE_OBJECT;
```

[결과 값]

```
no rows selected.
```

RESULT CACHE 에 등록된 모든 OBJECT 들을 DBMS_RESULT_CACHE.FLUSH 패키지를 통해 일괄 해지할 수 있다.

[QUERY 실행]

```
SELECT /*+ RESULT_CACHE */ 1 FROM DUAL;
SELECT /*+ RESULT_CACHE */ 2 FROM DUAL;
SELECT /*+ RESULT_CACHE */ 3 FROM DUAL;
```

[QUERY 실행]

```
SELECT
       ID,
       TYPE,
       STATUS,
       BUCKET_NO,
       HASH,NAME
FROM   V$RESULT_CACHE_OBJECT;
```

[결과 값]

```
ID  TYPE          STATUS    BUCKET_NO HASH            NAME
--- ----------    --------- --------  --------  -------------------------------------
2   RESULT        Published 3870      83370270  SELECT /*+ RESULT_CACHE */ 3 FROM DUAL
1   RESULT        Published 2222      93590190  SELECT /*+ RESULT_CACHE */ 2 FROM DUAL
0   RESULT        Published 3414      7988310   SELECT /*+ RESULT_CACHE */ 1 FROM DUAL
```

```
[패키지 실행]
EXEC DBMS_RESULT_CACHE.FLUSH

[QUERY 실행]
SELECT
       ID,
       TYPE,
       STATUS,
       BUCKET_NO,
       HASH,NAME
FROM   V$RESULT_CACHE_OBJECT;

[결과 값]
no rows selected
```

RESULT CACHE 기능 활용하기

과거 고객사의 예를 들어 보면 온라인 시간에 상품들에 대해 실적을 보는 업무가 있었다. 실적 테이블은 1 년 동안 보관되며 NON PARTITION 된 테이블이 12 개 존재하고 있다. 특정 월에 해당하는 실적을 보려면 해당 테이블에 모든 데이터를 읽어야만 한다. 그렇게 되면 I/O 가 많이 발생해 빠른 성능을 기대하기 어렵다. 그래서 매일 야간 배치가 실행되어 전일 기준으로 실적 집계 테이블을 갱신 하고 있었다. 많은 개선의 효과가 있지만 여전히 읽어야 할 데이터가 많기 때문에 많은 블록 I/O 가 발생하고 있었다. 여기서 각 달에 해당하는 실적 조회를 RESULT CACHE 기능을 사용 한다면 많은 성능 개선이 있을 것이다. 특정 월에 해당하는 실적 테스트 데이터를 생성하여 RESULT CACHE 기능을 활용해 보도록 하자.

테이블 생성 스크립트

```
[DDL 실행]
CREATE TABLE TB_RC_TEST_YYYYMMDD
AS
SELECT
       TO_CHAR(ADD_MONTHS(SYSDATE,-1) ,'YYYYMM')||
       TRIM(TO_CHAR(ROUND(DBMS_RANDOM.VALUE(1,31)),'09'))||
```

```
        TRIM(TO_CHAR(ROUND(DBMS_RANDOM.VALUE(0,23)),'09'))||
        TRIM(TO_CHAR(ROUND(DBMS_RANDOM.VALUE(00,59)),'09')) SDATE,
        DBMS_RANDOM.STRING('U',2) PROD,
        ROUND(DBMS_RANDOM.VALUE(100,100000)) AMT1,
        ROUND(DBMS_RANDOM.VALUE(10,10000)) AMT2,
        ROUND(DBMS_RANDOM.VALUE(1000,100)) AMT3
FROM    DUAL
CONNECT BY LEVEL <= 10000000 ;
```

RESULT CACHE 기능을 사용하지 않은 경우(X)

```
[QUERY 실행]
SELECT
        SUBSTR(SDATE,1,6) SDATE,
        PROD,
        SUM(AMT1) AMT1,
        SUM(AMT2) AMT2,
        SUM(AMT3) AMT3
FROM    TB_RC_TEST_YYYYMMDD
GROUP BY SUBSTR(SDATE,1,6),
           PROD
```

[실행계획]

```
call     count       cpu    elapsed       disk      QUERY    current        rows
------- ------  -------- ---------- ---------- ----------  ----------  ----------
Parse        1      0.00       0.00          0          0           0           0
Execute      1      0.00       0.00          0          0           0           0
Fetch       47     15.62      15.80      46944      46948           0         676
------- ------  -------- ---------- ---------- ----------  ----------  ----------
total       49     15.62      15.80      46944      46948           0         676

Rows (1st) Row Source OPERATION
---------- ---------------------------------------------------
676 SORT GROUP BY (cr=46948 pr=46944 pw=0 time=15803245)
10000000 TABLE ACCESS FULL TB_RC_TEST_YYYYMMDD (cr=46948 pr=46944 pw=0 time=3144528)
```

실행 결과를 보면 38,670 개의 BLOCK I/O 가 발생 하였다.

RESULT CACHE 기능을 이용하여 테스트를 다시 수행해 보자.

RESULT CACHE 기능을 사용한 경우(O) (최초 실행)

```
[QUERY 실행]
SELECT /*+ RESULT_CACHE */
       SUBSTR(SDATE,1,6) SDATE,
       PROD,
       SUM(AMT1) AMT1,
       SUM(AMT2) AMT2,
       SUM(AMT3) AMT3
FROM   TB_RC_TEST_YYYYMMDD
GROUP BY SUBSTR(SDATE,1,6),
         PROD
```

[실행 계획]

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       47     15.62      15.80      46944      46948          0         676
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       49     15.62      15.80      46944      46948          0         676

Rows (1st) Row Source OPERATION
---------- ---------------------------------------------------
676 SORT GROUP BY (cr=46948 pr=46944 pw=0 time=15803245)
10000000 TABLE ACCESS FULL TB_RC_TEST_YYYYMMDD (cr=46948 pr=46944 pw=0 time=3144528)
```

수행한 결과 여전히 38,670 개의 BLOCK I/O 가 발생하고 있다. 최초 QUERY 실행 시 CACHING 된 영역에 OBJECT 가 존재하지 않았기 때문이다. 이제 VIEW 를 보면 CACHE 영역에 등록되어 있는 것을 볼 수 있다.

CACHING 된 OBJECT 보기

```
[QUERY 실행]
SELECT           ID,
       TYPE,
       STATUS,
       BUCKET_NO,
       HASH,
```

```
       NAME
FROM   V$RESULT_CACHE_OBJECT;
```

[결과 값]

```
ID         TYPE             STATUS    BUCKET_NO HASH         NAME
----    ---------------   --------  --------- ---------- --------- --------------------
 0      Dependency        Published 660       319061    DBAADM.TB_RC_TEST_YYYYMMDD
 1      RESULT            Published 2011      159411    SELECT    /*+ RESULT_CACHE */
```

이제 CACHE 등록이 되어 있는 것을 확인 했으니 다시 한번 수행해 보자.

RESULT CACHE 기능을 사용한 경우(O) (반복 실행))

[QUERY 실행]

```
SELECT /*+ RESULT_CACHE */
       SUBSTR(SDATE,1,6) SDATE,
       PROD,
       SUM(AMT1) AMT1,
       SUM(AMT2) AMT2,
       SUM(AMT3) AMT3
FROM   TB_RC_TEST_YYYYMMDD
GROUP BY SUBSTR(SDATE,1,6),
         PROD
```

[실행 계획]

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ----------  ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       47      0.00       0.00          0          0          0         676
------- ------  -------- ---------- ---------- ----------  ----------  ----------
total       49      0.00       0.00          0          0          0         676

Rows (1st) Row Source OPERATION
---------- ---------------------------------------------------
       676 RESULT CACHE  gnxqgpxppdavj80b6rddg4qsqj (cr=0 pr=0 pw=0 time=20 us)
         0  SORT GROUP BY (cr=0 pr=0 pw=0 time=0 )
         0   TABLE ACCESS FULL TB_RC_TEST_YYYYMMDD (cr=0 pr=0 pw=0 time=0 )
```

실행 계획을 보면 I/O 가 전혀 발생하지 않았다. I/O 가 전혀 발생되지 않아 SQL 을 반복 수행하더라도 성능을 향상시켜 준다. 여기서 RESULT CACHE 라는 OPERATION 을 볼 수 있다. 결과 값이 CACHING 된 것이다. INLINE VIEW 나 WITH 문과 같은 쿼리에서도 RESULT CACHE 기능을 사용할 수 있다. 독립적으로 QUERY 블록에 CACHING 이 가능하기 때문이다. 테스트를 통해 알아 보도록 하자.

테이블 생성 스크립트

```
[DDL 실행]
CREATE TABLE TB_RC_TEST_SYSDATE
AS
SELECT
       TO_CHAR(ADD_MONTHS(SYSDATE,-1) ,'YYYYMM')||
       TRIM(TO_CHAR(ROUND(DBMS_RANDOM.VALUE(1,31)),'09'))||
       TRIM(TO_CHAR(ROUND(DBMS_RANDOM.VALUE(0,23)),'09'))||
       TRIM(TO_CHAR(ROUND(DBMS_RANDOM.VALUE(00,59)),'09')) SDATE,
       DBMS_RANDOM.STRING('U',2) PROD,
       ROUND(DBMS_RANDOM.VALUE(100,100000)) AMT1,
       ROUND(DBMS_RANDOM.VALUE(10,10000)) AMT2,
       ROUND(DBMS_RANDOM.VALUE(1000,100)) AMT3
FROM   DUAL
CONNECT BY LEVEL <= 300000 ;
```

INLINE VIEW 와 WITH 문 사용예제

```
[INLINE VIEW 사용 QUERY]
SELECT SDATE, SUM(SUM_AMT)
FROM ( SELECT    SDATE,
                 NVL(AMT1,0) + NVL(AMT2,0) + NVL(AMT3,0) SUM_AMT
       FROM      ( SELECT /*+ RESULT_CACHE */
                          SUBSTR(SDATE,1,6) SDATE,
                          PROD,
                          SUM(AMT1) AMT1,
                          SUM(AMT2) AMT2,
                          SUM(AMT3) AMT3
```

```
                FROM     TB_RC_TEST_YYYYMMDD
                GROUP BY SUBSTR(SDATE,1,6),
                                    PROD  )
        UNION ALL
        SELECT
                SUBSTR(SDATE,1,6) SDATE,
                SUM(AMT1) + SUM(AMT2) + SUM(AMT3) SUM_AMT
        FROM      TB_RC_TEST_SYSDATE
        WHERE     SDATE < :SDATE
        GROUP BY SUBSTR(SDATE,1,6),
                 PROD
        )
GROUP BY SDATE;
```

[WITH 문 사용 QUERY]

```
WITH  W_TB_RC_TEST_YYYYMMDD
AS
(SELECT SDATE,
                 NVL(AMT1,0) + NVL(AMT2,0) + NVL(AMT3,0) SUM_AMT
FROM ( SELECT /*+ RESULT_CACHE */
                SUBSTR(SDATE,1,6) SDATE,
                PROD,
                SUM(AMT1) AMT1,
                SUM(AMT2) AMT2,
                SUM(AMT3) AMT3
        FROM      TB_RC_TEST_YYYYMMDD
        GROUP BY  SUBSTR(SDATE,1,6),
                                    PROD  ))
SELECT
        SDATE,
        SUM(SUM_AMT)
FROM    (SELECT   SDATE,
                  SUM_AMT
             FROM           W_TB_RC_TEST_YYYYMMDD
        UNION ALL
        SELECT    SUBSTR(SDATE,1,6) SDATE,
                  SUM(AMT1) + SUM(AMT2) + SUM(AMT3) SUM_AMT
        FROM      TB_RC_TEST_SYSDATE
        WHERE     SDATE < :SDATE
        GROUP BY SUBSTR(SDATE,1,6),
                  PROD
```

```
        )
GROUP BY SDATE;
```

[실행 계획]

```
call     count       cpu    elapsed       disk      QUERY    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.18       0.18          0       1413          0           2
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.18       0.19          0       1413          0           2

Rows (1st) Row Source OPERATION
---------- ---------------------------------------------------
         2 SORT GROUP BY (cr=1413 pr=0 pw=0 time=188846 )
      1352  VIEW  (cr=1413 pr=0 pw=0 time=2355 )
      1352   UNION-ALL  (cr=1413 pr=0 pw=0 time=2230 us)
       676    VIEW  (cr=0 pr=0 pw=0 time=578 )
       676     RESULT CACHE  gnxqgpxppdavj80b6rddg4qsqj (cr=0 pr=0 pw=0 time=126 us)
         0      SORT GROUP BY (cr=0 pr=0 pw=0 time=0 )
         0       TABLE ACCESS FULL TB_RC_TEST_YYYYMMDD (cr=0 pr=0 pw=0 time=0 )
       676    SORT GROUP BY (cr=1413 pr=0 pw=0 time=187432 )
    104850     TABLE ACCESS FULL TB_RC_TEST_SYSDATE (cr=1413 pr=0 pw=0 time=57449 )
```

RESULT CACHE 사용시 고려사항

DML 사용시 고려사항

CACHING 된 OBJECT 에 DML(변경사항)이 발생되면 변경된 시점에서 RESULT CACHE 기능은 무효화 된다. 그 이유는 CACHING 된 OBJECT 가 변경된다면 QUERY 의 결과 값에 대한 정합성을 보장할 수 없기 때문이다. 따라서 DML 이 많이 발생하는 OBJECT 는 RESULT CACHE 기능을 사용하지 않는 것이 좋다.

BIND 사용시 고려사항

BIND 변수의 변경이 많은 QUERY 에 대해서도 비효율이 발생된다. 아래 테스트 결과를 보면 BIND 변수 값에 따라 독립적으로 CACHING 되는 것을 볼 수 있다. 그렇기 때문에 BIND 변수가

많아지면 특정 QUERY 가 CACHE 의 영역을 모두 사용 할 것이고 CACHE 하고자 하는 각기 다른 쿼리들에 의해 CACHE 의 등록과 해지가 빈번하게 발생되어 CACHE 의 효율은 떨어지게 된다.

같은 QUERY에서 BIND 변수 값이 다양한 경우

```
[QUERY 실행]
VAR NUM NUMBER;
EXEC :NUM := 1;
SELECT /*+ RESULT_CACHE */ 'CACHING Count' FROM DUAL WHERE 1 = :NUM;

EXEC :NUM := 2;
SELECT /*+ RESULT_CACHE */ 'CACHING Count' FROM DUAL WHERE 1 = :NUM;

EXEC :NUM := 3;
SELECT /*+ RESULT_CACHE */ 'CACHING Count' FROM DUAL WHERE 1 = :NUM;

[QUERY 실행]
SELECT
       ID,
       TYPE,
       STATUS,
       BUCKET_NO,
       HASH,NAME
FROM   V$RESULT_CACHE_OBJECT;

[결과 값]
ID TYPE   STATUS      BUCKET_NO   HASH        NAME
- ----    --------- --------- ----------- ----------------------------------------
3 RESULT Published   1882 2742746   SELECT /*+ RESULT_CACHE */ 'CACHING Count' FROM
DUAL WHERE 1 = :A
2 RESULT Published   1714 7906098   SELECT /*+ RESULT_CACHE */ 'CACHING Count' FROM
DUAL WHERE 1 = :A
1 RESULT Published   3880 95977768  SELECT /*+ RESULT_CACHE */ 'CACHING Count' FROM
DUAL WHERE 1 = :A
0 RESULT Published   3157 98148181  SELECT /*+ RESULT_CACHE */ 'CACHING Count' FROM
DUAL WHERE 1 = :A
```

특정 OBJECT 사용시 고려사항

아래 내용은 쿼리 결과 집합을 CACHING 하지 못하는 경우이다.

- 임시 테이블 또는 DICTIONARY OBJECT (DBA_*, V$_*, GV$_* 등) 참조 시
- 시퀀스 CURRVAL 및 NEXTVAL COLUMN 호출 시
- QUERY 에서 SQL 함수를 사용 할 경우
  - CURRENT_TIMESTAMP, LOCAL_TIMESTAMP, SYS_GUID, SYSDATE, SYSTIMESTAMP 등

결론

기존 QUERY 들은 I/O 가 발생 해야만 결과 집합을 알 수 있다. 많지 않은 BLOCK 이라도 많은 세션들이 동시에 같은 BLOCK 을 사용 한다면 WAIT EVENT 가 발생한다. 이는 좋은 성능을 기대하기 어렵지만 RESULT CACHE 기능을 사용하게 되면 I/O 를 전혀 발생하지 않는다는 부분에 있어 상당한 성능 개선의 효과가 있다. 하지만 그 기능의 장점과 단점을 잘 알고 사용해야만 시스템을 안정적으로 사용할 수 있을 것이다. 또한 운영하고 있는 시스템을 살펴보면 RESULT CACHE 기능을 사용할 수 있는 부분이 있을 것이다. 이 기능을 활용해서 I/O 성능을 더욱 개선시킨다면 더욱 안정화되고 최적화된 시스템으로 발전하게 될 것이다.