Oracle Technical Note

DB Tuning
Oracle SQL Tuning 실무사례 (9)

데이타 웨어하우스 아키텍처와 컴포넌트

이번 호에서는 배치(batch) 프로그램에서 활용할 수 있는 튜닝 방법을 소개하고자 한다. 일반적으로 배치 프로그램은 최소 1분에서 수십 시간까지의 실행시간을 가지는데, 이 동안 시스템의 자원을 최대한 끌어당겨 사용하여 처리하므로 같이 수행되는 다른 프로그램에 미치는 영향도 크고 또 실행하는 일의 양도 엄청나다고 할 수 있다. 이렇게 엄청난 일을 하는 배치 프로그램의 내부를 들여다 보면, 실제 처리해야 하는 데이타의 양이 많아서 오래 걸리는 경우는 어쩔 수 없지만 주로 문제가 되는 프로그램들은 동일한 유형의 SQL을 루핑(looping)을 돌리면서 반복처리하기 때문에 많은 시간이 소요되고 있음을 알 수 있다. 이렇게 루핑을 도는 SQL의 튜닝은 그 로직을 이해한 뒤 전혀 새로운 SQL을 만들어내는 방식인 로직 튜닝으로만 가능하며, 이러한 SQL은 주로 한두 개의 SQL내에서 필요한 데이타를 모두 가져와 처리하므로 수백, 수천 개의 낱개의 SQL을 오라클 커널이 일일이 처리하는 방식보다 수십, 수백 배의 성능 향상을 가져올 수 있다.

특히 그 처리 내용은 배치이지만 마치 온라인 프로그램처럼 사용하는 온라인성 배치 프로그램인 경우에 매우 유용하게 활용할 수 있다.

ORACLE®

사례 1 배치 집계 테이블을 이용한 온라인 처리

소개

이 경우는 배치로 처리하는 합계 테이블과 온라인으로 처리하는 테이블을 서로 같이 사용하여 어떤 범위의 조건이 들어와도 항상 온라인 처리 속도로 결과를 화면에 표시하는 요구 조건을 충족시키는 사례이다.

집계 테이블에는 항상 지난달인 경우 맨 마지막 날짜에 그 달의 합계가 들어 있고, 이번 달인 경우 항상 어제까지의 합계가 어제 날짜로 들어 있다. 만약 오늘이 96년 4월 27일이라면 집계 테이블의 데이타는 다음과 같이 들어 있다.

집계일자	COL1	COL2
960131	AAA	123
960131	ABC	321
960228	AAA	231
960331	AAA	351
960331	ABC	255
960427	AAA	142

이러한 상태로 운영하다가 4월 28일에 시스템 증설 작업 관계로 매일 수행해야 하는 집계 테이블 업데이트 작업을 수행하지 못했다. 이러한 상황에서 오는 4월 29일 현재까지의 SUM(COL2)을 COL1 별로 GROUP BY 해서 보고서를 작성해야 한다.

독립 데이타 마트(independent data mart) 아키텍처는 전사적 데이타 웨어하우스 구축 없이 소수의 사용자들(부서별)을 위한 제한된 주제를 가지고 소규모의 데이타 마트를 하나 또는 여러 개 구축하는 시스템이다. 주로 한 주제에 대한 다양한 분석, 예측을 위한 시스템이기 때문에 주로 MOLAP 솔루션을 사용하는 경우가 많다. 필요한 데이타 마트를 단기간에 구축할 수 있다는 장점이 있으나, 직접 데이타를 운영계 시스템에서 추출해야 하고, 이를 주기적으로 자동화해야 하며, 데이타 마트가 많아질 경우 데이타 추출에 문제가 생겨 전체 시스템 관리가 어렵게 된다는 단점도 있다.

문제점

4월 28일과 4월 29일에 집계 작업을 하지 못했으므로 집계 테이블의 데이타를 3월 31일 자료까지만 사용하고 4월 자료는 온라인 테이블에서 가져와서 작업하려고 하니 온라인 테이블의 건수가 너무 많아서 수행 속도가 매우 느리다.

또한 프로그램에 코딩된 내용은 지난 달까지의 집계 데이타의 합과 어제까지의 합계를 더하여 사용하도록 고정되어 있는데, 이번과 같이 어제의 집계 작업을 하지 못한 경우에는 어제까지의 집계 작업을 먼저 해 주어야만 보고서를 작성할 수 있다.

또한 어제까지의 합계를 온라인 테이블에서 구하는데, 월초인 경우는 데이타가 많지 않아서 별 문제가 없지만 월말인 경우는 온라인 테이블의 해당 건수가 많으므로 항상 문제가 되고 있다.

해결방안

우선 그날의 작업 마감 후 일일 배치 작업으로 합계를 구하는 방식은 그 전날까지의 합계에 그날의 합계를 더하여 집계 테이블을 업데이트하면 된다.

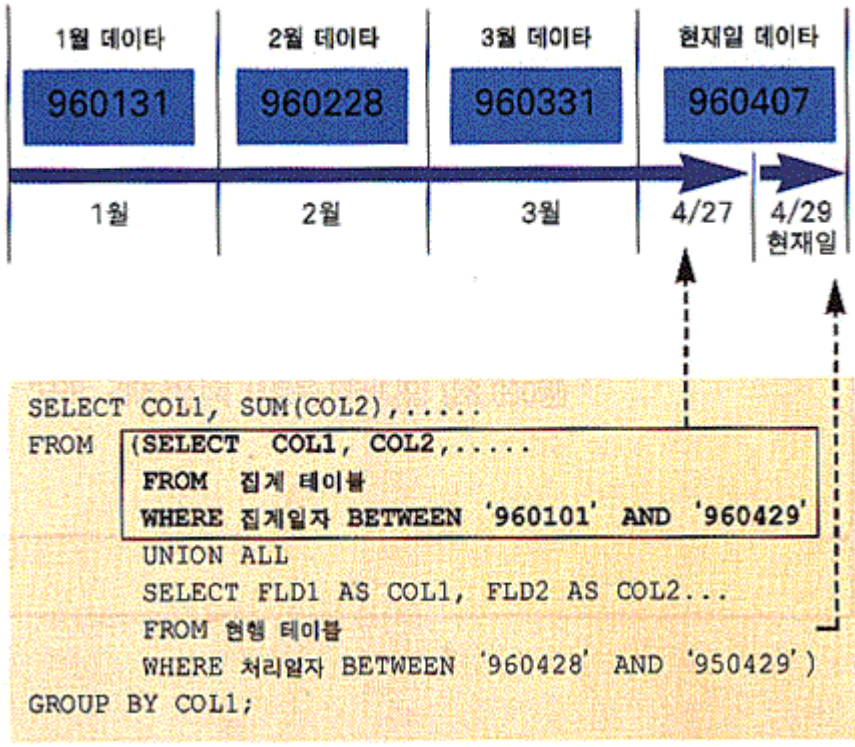
그 다음 현재 월에 대한 해결방안인데, 이는 최대한 집계 테이블의 데이타를 사용할 수 있는 데까지 사용하고 나머지를 온라인 테이블에서 구하는 방식을 사용한다.

여기서 집계 테이블의 배치 작업을 며칠간 하지 못했다 하더라도 자동적으로 집계 작업이 된 부분은 집계 테이블을 이용하고 나머지는 온라인 테이블에서 찾는 방법을 알아보자. | 그림 1 |

실제 값을 변수로 받아서 처리하는 완전한 SQL 문은 다음과 같다.

```
SELECT COL1, SUM(COL2), . . . . . .
FROM (SELECT  COL1, COL2, . . . . FROM 집계테이블
        WHERE   집계일자 BETWEEN :BEGIN_DATE AND
                                   TO_CHAR(SYSDATE,'YYMMDD')
      UNION ALL
      SELECT  FLD1 AS COL1, FLD2 AS COL2. . . . FROM 현행테이블
      WHERE   처리일자 > (SELECT /*+ INDEX_DESC(집계 테이블 집계일
                          자_IX) */ 집계일자
                          FROM 집계테이블
                          WHERE ROWNUM = 1)
      AND       처리일자 <= TO_CHAR(SYSDATE,'YYMMDD') )
GROUP BY COL1 ;
```

| 그림 1 |



사례 2 일별 집계 테이블에서 직접 온라인 조회

소개

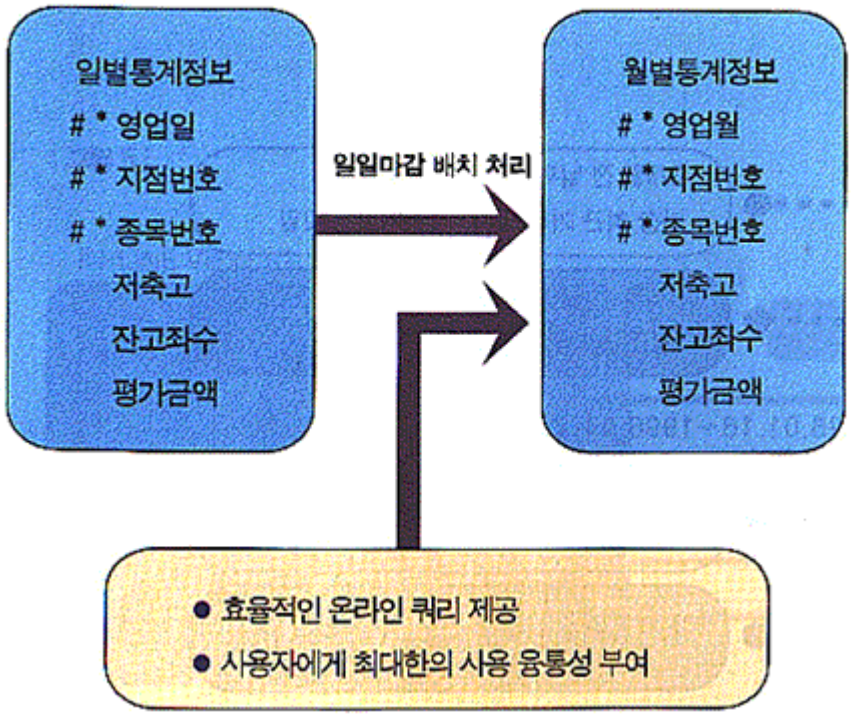
일일 마감 배치 작업으로 처리되는 일별통계정보 테이블과 월별통계정보 테이블을 동시에 이용하여 월별 평가 금액 합계를 온라인으로 직접 조회할 수 있는 사례이다.

이전에는 이런 월별 통계자료를 보기 위해서는 오랜 시간을 기다려야 결과를 볼 수 있었지만, 이제부터는 화면상에서 온라인 프로그램과 같은 응답 속도로 볼 수 있다.

월별통계정보에는 매월의 합계 값이 YYMM을 KEY로 하여 각각의 로우에 저장되어 있고, 일별통계정보에는 매일의 합계 값이 YYMMDD를 KEY로 하여 각각의 로우에 저장되어 있다. 사용자는 어떤 범위의 조건을 주더라도 자동적으로 일별통계정보와 월별통계정보 테이블을 가장 효율적으로 이용하여 평가 금액의 합계를 구하도록 한다.

예를 들어, 사용자가 1996.1.15부터 1996.5.10까지의 조건을 주었다면, 1996.1.15부터 1996.1.31까지와 1996.5.1부터 1996.5.10까지의 데이타는 일별통계정보 테이블을 이용하고 1996.2와 1996.3과 1996.4 데이타는 월별통계정보 테이블을 이용하여 구하도록 해야 한다. | 그림 2 |

| 그림 2 |



문제점

일별통계정보와 월별통계정보 테이블을 각각 읽어서 그 합계를 구하여 처리할 경우 온라인 프로그램과 같은 응답 속도가 나오지 않는다.

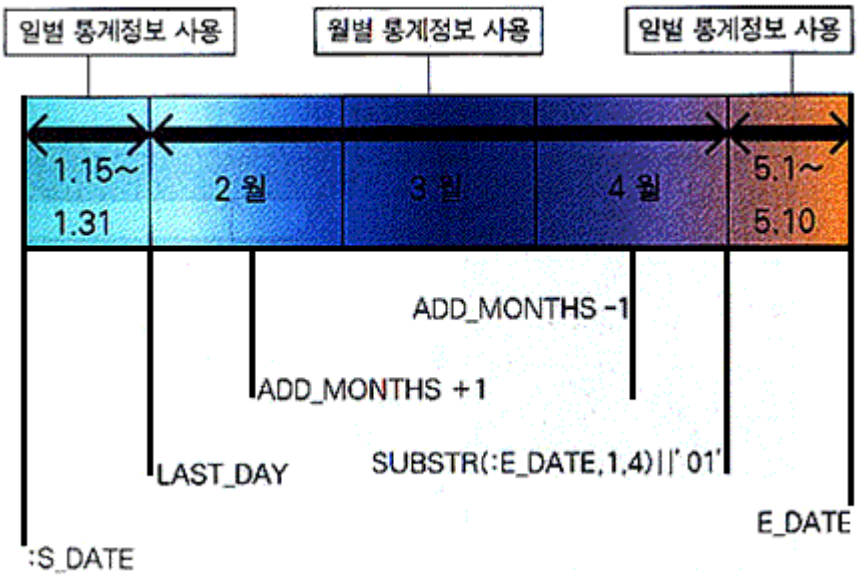
해결방안

IN-LINE VIEW를 사용하여 하나의 SQL 문으로 처리하는 방법을 찾아보자. 우선 일별통계정보와 월별통계정보 테이블을 사용할 범위를 서로 구분하는 것이 가장 우

선되어야 한다. 시작일(:S_DATE)과 종료일(:E_DATE)이 속하는 범위는 일별통계정보 테이블을 사용하고, 시작일의 다음달과 종료일의 이전달까지는 월별통계정보 테이블을 사용한다. 시작일이 속하는 월의 마지막 일자를 알기 위해 LAST_DAY 함수를 사용하고 시작일 다음달을 알기 위해 ADD_MONTHS 함수를 이용했다.| 그림 3 | 이를 정확하게 SQL로 구현하면 다음과 같이 된다.

```
SELECT V.영업월, V.평가금액합
FROM (SELECT SUSTR(영업일,1,4) AS 영업월 , SUM(평가 금액) AS 평
가금액합
        FROM  일별통계정보
        WHERE 지점번호 = :B1
           AND ( (영업일 BETWEEN :S_DATE AND
                                TO_CHAR(LAST_DAY(TO_DATE(
                                :S_DATE,'YYMMDD')),
                                'YYMMDD'))
           OR (영업일 BETWEEN SUBSTR(:E_DATE,1,4) || '01'
              AND :E_DATE) )
        GROUP BY SUSTR(영업일,1,4)
        UNION ALL
        SELECT 영업월, 평가 금액 AS 평가금액합
        FROM   월별통계정보
        WHERE  지점번호 = :B1
        AND     영업월 >= SUBSTR(TO_CHAR(ADD_MONTHS(TO_DATE
        (:S_DATE
                                ,'YYMMDD'),1),'YYMMDD'),1,4)
        AND    영업월 <= SUBSTR(TO_CHAR(ADD_MONTHS(TO_DATE
                                (:E_DATE,'YYMMDD'),-1),'
                                YYMMDD'),1,4) ) V
ORDER BY V.영업월 ;
```

| 그림 3 |



사례 3 어떤 범위의 평균 잔액이라도 온라인으로 조회

소개

I이번 사례는 전형적으로 배치 작업을 통해서만 결과를 얻을 수 있었던 것을 온라인으로 조회할 수 있도록 한 통계 처리 SQL 문의 대표적인 형태라 하겠다. 집계 테이블 없이 거래 실적 테이블의 데이타를 이용하여 바로 실시간으로 조회하는 것이며 일반적인 은행에서 대표적으로 적용되는 경우이다. 다음과 같이 보관된 일별 거래기록 데이타를 이용하여 설명한다.

계좌번호	거래마감일	잔액
23-12345	1996.01.08	1000000
23-12345	1996.01.12	600000
37-34567	1996.01.16	250000
23-12345	1996.02.07	800000
37-34567	1996.02.15	1100000
23-12345	1996.02.17	1350000
23-12345	1996.03.18	500000
37-34567	1996.03.30	1200000
23-12345	1996.04.13	1000000
36-34567	1996.04.15	2000000
23-12345	1996.04.17	500000

위와 같은 일별 거래기록을 갖고 있을 때 1996.4.16일에 계좌번호 23-12345에 대한 3개월 간 평균 잔액을 구하는 요구 조건이다.

사용자가 평균 잔액을 구하는 기간을 어떻게 지정하더라도 자동적으로 지정 기간의 조건에 맞는 계좌에 대해서만 합계를 구하여 지정 기간의 일자로 나누어 평균 잔액을 산출해 내어야 한다. 위와 같은 경우에 평균 잔액을 구하는 과정은 다음과 같다.

96.04.16기준3개월해당기간	예치일자	금액
1996.01.16~1996.02.06	21일간	600,000
1996.02.07~1996.02.16	9일간	800,000
1996.02.17~1996.03.17	29일간	1,350,000
1996.03.18~1996.04.12	25일간	500,000
1996.04.13~1996.04.16	3일간	1,000,000

위의 표에 의거하여 3개월 평균 잔액을 계산하면 다음과 같다.

1. 1996. 1. 16 ~ 1996. 4. 16 기간의 일자 수 ; 91일 간

2. 3개월 간 잔액 합계 = (21*600000 + 9*800000 + 29*1350000 + 25*500000 + 3*1000000) ÷ 91 = 818,131 원

문제점

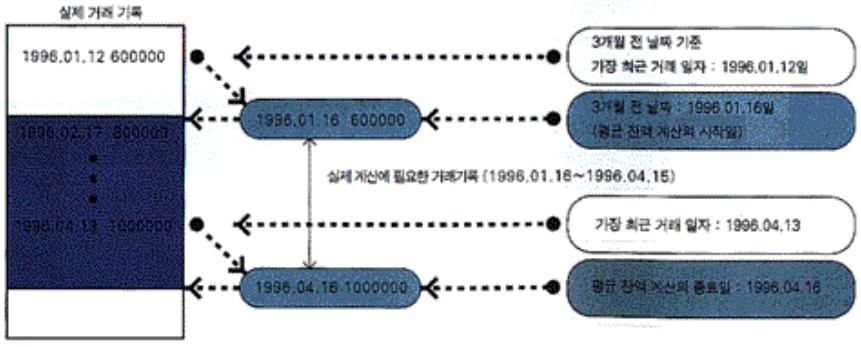
일반적인 구현 형태는 다음과 같이 프로그래밍으로 해결하며 루프를 돌리며 처리하므로 속도가 느리다.

```
SELECT    ADD_MONTHS ( SYSDATE, -3 ) INTO :3개월전일자 FROM
DUAL;
DECLARE CURSOR
      SELECT  거래마감일, 잔액
      FROM    일별거래기록
      WHERE   거래마감일 >= ( SELECT MAX(거래마감일) FROM
                              일별거래기록
                              WHERE 계좌 번호 = '23-12345'
                               AND 거래마감일 <= :3개월전일자 )
      AND     계좌 번호 = '23-12345' AND 거래마감일 <= SYSDATE
      ORDER BY 거래마감일 ;
OPEN CURSOR
FOR ( ; ; )
   FETCH CURSOR INTO :거래마감일, :잔액 ;
                        . . . . . .
                        . . . . . .
```

해결방안

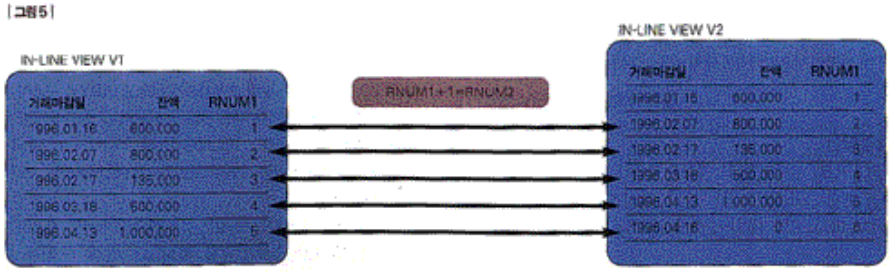
IN-LINE VIEW를 이용하여 다음과 같이 하나의 SQL 문으로 해결할 수 있다. 우선 3개월 전 일자를 기준으로 가장 가까운 과거 거래일자를 찾아서 그 잔액을 가져온다. 위의 데이타를 사용하여 예를 들면, 오늘 4.16일부터 3개월 전이라면 1.16일이 되는데, 1.16일에 거래가 없으므로 1.16일 기준으로 가장 최근에 거래한 1.12일자를 알아내어 그때의 금액을 가져와 1.16일부터 계산하는 데 사용한다. 마찬가지로 계산 종료일인 4.16일에도 실제 거래가 없으므로 가장 최근에 거래한 4.13일자를 알아내어 그때의 금액을 가져와 사용한다.| 그림 4 |

|그림4|



이렇게 하여 시작일과 종료일의 금액을 알아내었다면 이제는 예치 기간을 알아내야 하는데, 이는 원래의 데이타와 원래의 행을 한 칸씩 뒤로 시프트시킨 데이타의 ROWNUM을 이용하면 알아낼 수 있다.| 그림 5 |

|그림 5|



해결방안

이와 같은 아이디어를 갖고 구현한 실제 SQL 문은 다음과 같다.

```
SELECT SUM ( (V2.거래마감일 - V1.거래마감일) * V1.잔액 )
          / (SYSDATE - (ADD_MONTHS(SYSDATE, -3)) ) AS 평균잔액
FROM (SELECT ROWNUM AS RNUM1 , 잔액 ,
                DECODE(SIGN(거래마감일 - ADD_MONTHS
                        (SYSDATE,-3) ), >> ①
                        -1, ADD_MONTHS(SYSDATE,-3), 거래마감
                        일) AS 거래마감일
        FROM  일별거래기록
        WHERE 계좌 번호 = '23-12345' AND 거래마감일 < SYSDATE
        AND  거래마감일  >= (SELECT NVL(MAX(거래마감일),
        ADD_MONTHS(SYSDATE,-3)) >> ②
                        FROM 일별거래기록
                        WHERE 계좌 번호 = '23-12345'
                           AND 거래마감일 <= ADD_MONTHS
                               (SYSDATE, -3)) ) V1, >> ③
        (SELECT ROWNUM AS RNUM2, 거래마감일, 잔액

        FROM (SELECT 잔액, DECODE(SIGN(거래마감일 -
                ADD_MONTHS(SYSDATE, -3)-1,
                ADD_MONTHS(SYSDATE,-3), 거래마감일) AS 거래마감일
                FROM 일별거래기록
               WHERE 계좌 번호 = '23-12345' AND 거래마감일 < SYSDATE
                     AND 거래마감일 >= (SELECT NVL(MAX(거래마감일),
ADD_MONTHS(SYSDATE,-3) )
                FROM 일별거래기록
                WHERE 계좌 번호 = '23-12345'
                  AND 거래마감일 <= ADD_MONTHS(SYSDATE , -3) )
        UNION ALL
        SELECT 0 AS 잔액, SYSDATE AS 거래마감일 >> ④
        FROM DUAL ) V2 >> ⑤
WHERE  V1.RNUM1 + 1 = V2.RNUM2 ;
```

①은 일별 거래기록 테이블의 거래 마감일 데이타와 3개월 전 일자(1996.1.16.)를 비교하여 3개월 이내이면 거래 마감일을 가져오고 3개월 이전이면 정확히 3개월 전 일자를 가져온다. 위의 데이타를 이용하여 예를 들어 보면, 거래 마감일이 1996.02.07이거나 1996.04.13이면 그냥 그 날짜를 가져오고 만약 3개월 범위를 벗어나는 1996.1.12일이면 1996.01.16일자로 변환하여 가져온다.

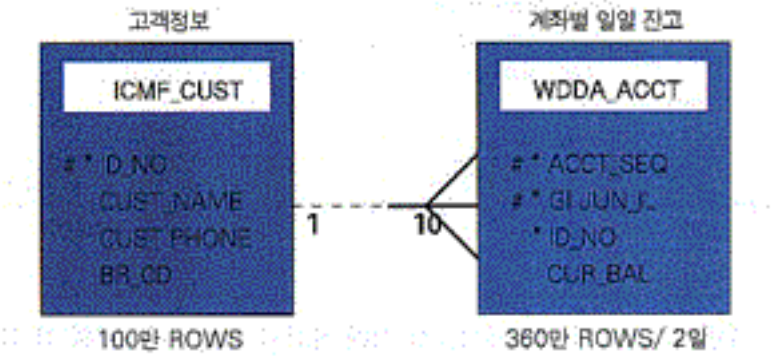
②는 3개월 이전 날짜를 기준으로 가장 최근의 일자를 가져온다. 즉, WHERE 조건이 거래마감일 <= ADD_MONTHS(SYSDATE,-3)이면서 MAX(거래 마감일)이라고 하면 된다. 이때 만약 최초 거래일 이후의 기간이 아직 3개월이 안된 고객이 있다면 WHERE 조건이 NULL이 되어 V1 전체의 결과가 1건도 나오지 않으므로 이를 방지하기 위해 NVL을 사용하여 NULL일 경우 ADD_MONTHS(SYSDATE, -3) 값으로 대치되도록 하였다.

③은 첫번째 IN-LINE VIEW이며, ④는 V2 IN-LINE VIEW에서 1칸을 시프트하기 위한 것이고, ⑤는 두 번째 IN-LINE VIEW이다.

사례 4 GROUP BY를 온라인 부분 범위 처리로 유도

소개

이 사례는 온라인으로 대량 데이타를 조회하여 보고자 하는 경우에 전체 범위로 처리하기 때문에 첫 화면이 나오기까지 많은 시간이 걸리는 부분을 부분 범위 처리로 유도함에 따라 첫 화면이 나오는 속도를 비약적으로 향상시켜서 응답 속도를 개선한 경우이다.



위와 같은 상태에서 영업 지점(A.BR_CD), 기준일(B.GIJUN_IL)이 WHERE 조건으로 들어왔을 때 잔액의 합계 즉, SUM(B.CUR_BAL)이 입력된 기준 금액(:ISUMBAL) 이상인 고객에 대한 리스트를 보여달라는 조건이다. 다음과 같이 SQL 문을 작성할 수 있다.

```
SELECT   A.CUST_NAME, SUM(B.CUR_BAL)
FROM     CMF_CUST A, WDDA_ACCT B
WHERE    A.ID_NO = B.ID_NO
AND      A.BR_CD = :IBRCD AND B.GIJUN_IL = :IDATE
GROUP BY A.CUST_NAME
HAVING SUM(B.CUR_BAL) >= :ISUMBAL ;
Excution Plan
SELECT
  FILTER
```

```
SORT ( GROUP BY )  >> 전체 범위의 원인
  NESTED  LOOPS
  TABLE ACCESS ( BY ROWID ) OF 'ICMF_CUST'
     INDEX ( RANGE SCAN ) OF 'ICMF_CUST_IDX1'
  TABLE ACCESS ( BY ROWID ) OF 'WDDA_ACCT'
     INDEX ( RANGE SCAN ) OF 'WDDA_ACCT_IDX1'
```

문제점

GROUP BY A.CUST_NAME 때문에 전체 범위로 처리된다. 따라서 첫 화면이 나오기까지의 수행 속도가 너무 오래 걸린다.

해결방안

GROUP BY 절이 사용되는 부분을 없애서 부분 범위 처리가 되도록 유도한다. GROUP BY 절이 없어지면 HAVING SUM (B.CUR_BAL)… 절도 같이 사용할 수 없으므로 서브쿼리로 대치한다.

```
SELECT A.CUST_NAME, GET_SUM(A.ID_NO, :IDATE)
FROM   ICMF_CUST A
WHERE  A.BR_CD = :IBRCD
AND   :ISUMBAL <= (SELECT SUM(CUR_BAL) FROM WDDA_ACCT B
                   WHERE A.ID_NO = B.ID_NO AND B.GIJUN_
                   IL = :IDATE) ;
```

이 SLQ 문 실행에 앞서 GET_SUM이라는 FUNCTION을 먼저 작성해 주어야 한다.

```
CREATE OR REPLACE FUNCTION GET_SUM(VIDNO IN NUMBER,
VIDATE IN DATE )
RETURN NUMBER
IS
SUM_BAL NUMBER;
BEGIN
  SELECT SUM(CUR_BAL) INTO SUM_BAL
      FROM  WDDA_ACCT
    WHERE  ID_NO = VIDNO
       AND  GIJUN_IL = VIDATE;
  RETURN  SUM_BAL;
END;
```

이때의 EXECUTION PLAN은 다음과 같다.

```
Excution Plan
 SELECT
  FILTER
    TABLE ACCESS ( BY ROWID ) OF 'ICMF_CUST'
     INDEX ( RANGE SCAN )  OF 'ICMF_CUST_IDX1' >> 부분 범위 처리화
    SORT AGGREGATE
     TABLE ACCESS ( BY ROWID ) OF 'WDDA_ACCT'
       INDEX ( RANGE SCAN )    OF 'WDDA_ACCT_IDX1'
```

해결방안

GROUP BY 절이 없어지려면 SUM 등의 함수를 MAIN SQL 내에서 사용할 수 없으므로 이를 따로 수행해 주어야 한다. 이를 위해서는 SUM 기능만 별도로 수행해 주는 FUNCTION의 사용이 불가피하게 되므로 사전에 별도의 FUNCTION을 미리 만들어 두어야 한다. 이렇게 되면 MAIN 쿼리에서 한 건이 나오면 그때마다 FUNCTION을 한 번씩 수행하여 SUM을 구하므로 부분 범위 처리가 되어 첫 화면이 나오는 속도가 매우 빨라지게 되어 사용자가 느끼는 응답 속도는 획기적으로 개선된다.

그러나 전체적으로 모든 범위의 데이타를 처리하는 데는 매 건마다 FUNCTION을 한 번씩 수행하므로 GROUP BY를 사용하는 것보다 느리다. 따라서 이와 같은 SQL은 반드시 첫 화면의 응답 속도를 중요하게 여기는 온라인 프로그램과 같은 분야에서만 사용되어야 하고 배치 형태의 처리는 당연히 GROUP BY를 사용하여 처리하는 것이 더 빠르다.

여기서 하나 더 생각할 수 있는 것은 FUNCTION을 사용하지 않기 위해 IN-LINE VIEW를 하나 더 씌워 다음과 같이 SUM(CUR_BAL)을 할 수도 있다.

```
SELECT V1.CUST_NAME, SUM(CUR_VAL)
FROM    WDDA_ACCT C,
        (SELECT ID_NO, CUST_NAME
         FROM   ICMF_CUST A
         WHERE  A.BR_CD = :IBRCD
           AND  :ISUMBAL <= (SELECT SUM(CUR_BAL)
                             FROM WDDA_ACCT B
                            WHERE A.ID_NO = B.ID_NO
                             AND B.GIJUN_IL = :IDATE)) V1
WHERE  C.ID_NO = V1.ID_NO
GROUP  BY V1.CUST_NAME ;
```

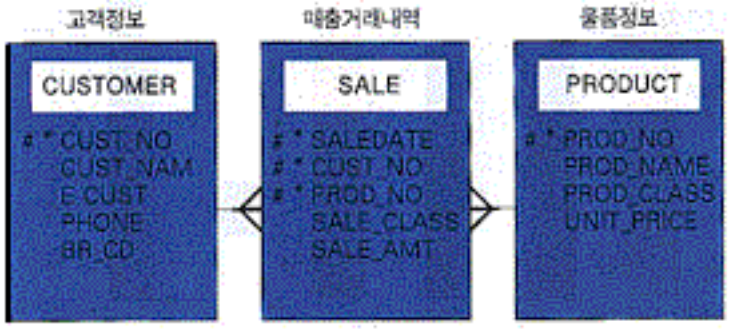
그러나 결국 위의 SQL 문도 맨 마지막에 GROUP BY 절이 들어가지 않으면 안되기 때문에 전체 범위로 처리되게 되므로 답은 같이 나올지 모르지만 원하는 부분 범위 처리가 되지 못한다.

결론적으로 GROUP BY 문을 없애고 이를 대신할 SUM을 구하는 FUNCTION을 사용해야만 하며 이렇게 하면 전체 범위 처리를 부분 범위 처리로 유도할 수 있다.

사례 5 EXISTS, IN LINE VIEW를 이용한 조인

소개

대부분의 마스터-디테일 관계에서 조인은 디테일의 테이블이 마스터 테이블의 내용을 확인하는 성격인데, 이 경우 마스터 테이블의 용도가 단순 확인을 위해서라면 조인 대신 IN-LINE VIEW나 EXISTS 문을 사용하여 수행 속도를 개선할 수 있다.

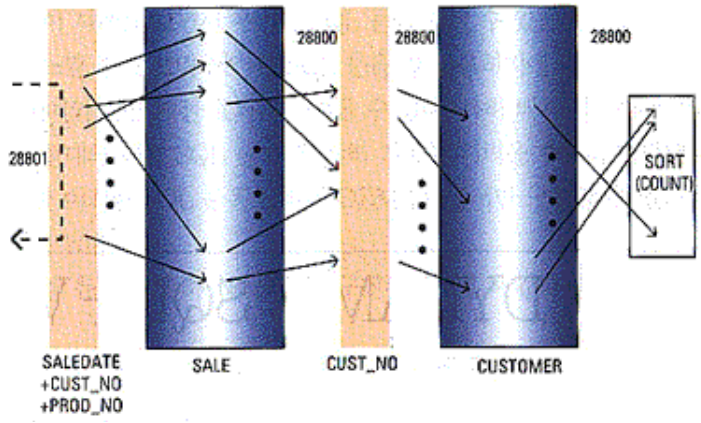


고객 정보(CUSTOMER) 테이블의 영업점(BR_CD)이 B10001에서 B10100 사이에 있는 고객에 대하여 매출거래내역(SALE) 테이블을 조사하여 1995년 1/4 분기의 매출액(SALE_AMT) 중 1000원 이상되는 매출 거래가 몇 건이나 되는지 알아보고자 한다. 이 요구사항을 만족하는 일반적인 SQL 문을 작성하면 다음과 같다.

```
SELECT COUNT(*) AS 거래건수
FROM   SALE A, CUSTOMER B
WHERE  A.SALEDATE BETWEEN '19950101' AND '19950331'
    AND  A.SALE_AMT >= 1000 AND  A.CUST_NO = B.CUST_NO
    AND  B.BR_NO BETWEEN 'B10001' AND 'B10100' ;>>수행 시간 :28.30초
 Rows       Excution Plan
    0   SELECT STATEMENT  HINT: CHOOSE
    0    SORT (AGGREGATE)
28800       NESTED LOOPS
28800      TABLE ACCESS ( BY ROWID ) OF 'SALE'
28801        INDEX ( RANGE SCAN ) OF 'SALE_PK'( UNIQUE )
28800      TABLE  ACCESS ( BY ROWID ) OF 'CUSTOMER'
28800        INDEX ( UNIQUE SCAN )OF'CUSTOMER_PK'( UNIQUE )
```

문제점

인덱스를 사용하기는 하나 연결하는 횟수가 많아서 비효율이 발생하고 따라서 수행 시간이 오래 걸린다. WHERE 조건에서 걸러져 나온 28,800건의 모든 SALE 데이타를 CUSTMER 테이블에 연결한 후 CUSTOMER 테이블에서 28,800건의 BR_NO 조건을 체크하여 일치되는 것만 카운트하여 구한다.



해결방안

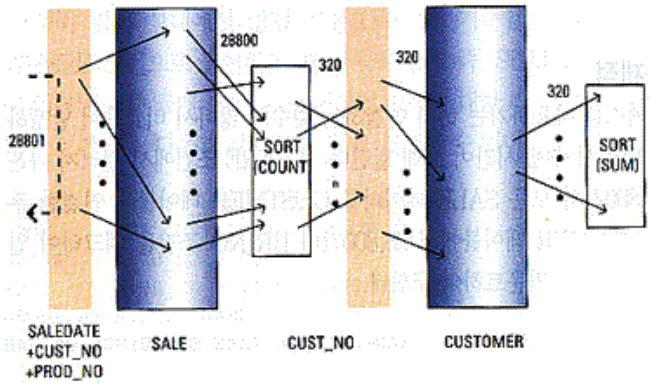
IN-LINE VIEW를 사용하여 해결

우선 SALE 테이블에서 먼저 GROUP BY 하여 COUNT를 구하고, 그 다음 CUSTOMER 테이블에 연결하여 BR_NO 조건을 체크한다. 이렇게 되면 SALE 테이블에서 각 CUST_NO에 대하여 한 번씩만 CUSTOMER 테이블을 연결해 보면 되므로(320번) 조인의 횟수가 28,800번에서 320번으로 줄어들고 그만큼 수행 속도가 빨라진다.

```
SELECT SUM(CNT) AS 거래건수
FROM    (SELECT  CUST_NO, COUNT(*) AS CNT
         FROM    SALE
         WHERE   SALEDATE BETWEEN '19950101' AND '19950331'
           AND   A.SALE_AMT >= 1000
         GROUP BY CUST_NO) V, CUSTOMER B
AND V.CUST_NO = B.CUST_NO
AND B.BR_NO BETWEEN 'B10001' AND 'B10100' ; >> 수행 시간 :12.11초
Rows     Excution Plan
     0     SELECT STATEMENT  HINT: CHOOSE
   320       SORT (AGGREGATE)
   320        NESTED LOOPS
   320          VIEW OF ' FROM$_SUBQUERY$_ 1 '
 28800            SORT (GROUP BY)
 28800             TABLE ACCESS (BY ROWID) OF 'SALE'
 28801                INDEX (RANGE SCAN) OF 'PK_SALE' (UNIQUE)
   320          TABLE ACCESS (BY ROWID) OF 'CUSTOMER'
   320            INDEX (UNIQUE SCAN) OF 'PK_CUSTOMER' (UNIQUE)
```

문제점

실행 계획을 그림으로 표시하면 다음과 같다.

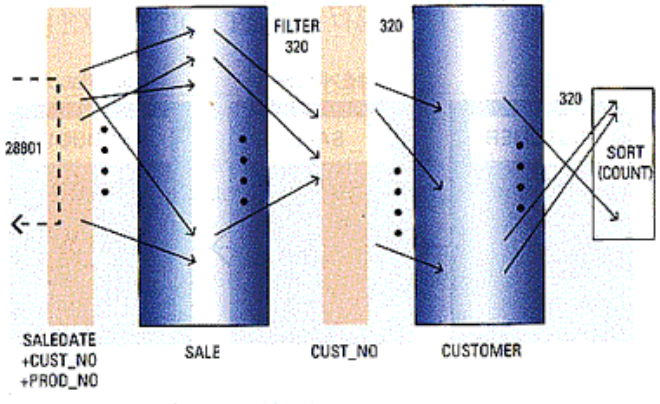


EXISTS를 사용하여 해결

위의 경우에서 IN-LINE VIEW 내에서 수행된 결과는 결국 CUSTOMER 테이블에 가서 BR_NO 조건의 CUST_NO가 있나 없나만 확인하는 절차이므로 이 경우는 EXISTS 문을 사용하여 다음과 같이 해결할 수도 있다.

```
SELECT  COUNT(*) AS 거래건수
  FROM  SALE A
WHERE   A.SALEDATE BETWEEN '19950101' AND '19950331'
   AND  A.SALE_AMT >= 1000
   AND  EXISTS (SELECT 'X' FROM CUSTOMER B
                WHERE A.CUST_NO = B.CUST_NO
                AND B.BR_NO BETWEEN 'B10001' AND 'B10100');
                >> 수행 시간 :12.28초
Rows     Excution Plan
   0   SELECT STATEMENT  HINT: CHOOSE
   0     SORT ( AGGREGATE )
28800      FILTER
28800        TABLE ACCESS ( BY ROWID ) OF 'SALE'
28801          INDEX ( RANGE SCAN ) OF 'PK_SALE' (UNIQUE)
  320       TABLE ACCESS ( BY ROWID ) OF 'CUSTOMER'
  320         INDEX ( UNIQUE SCAN ) OF 'PK_CUSTOMER' (UNIQUE)
```

실행되는 과정을 그림으로 표시하면 다음과 같다.

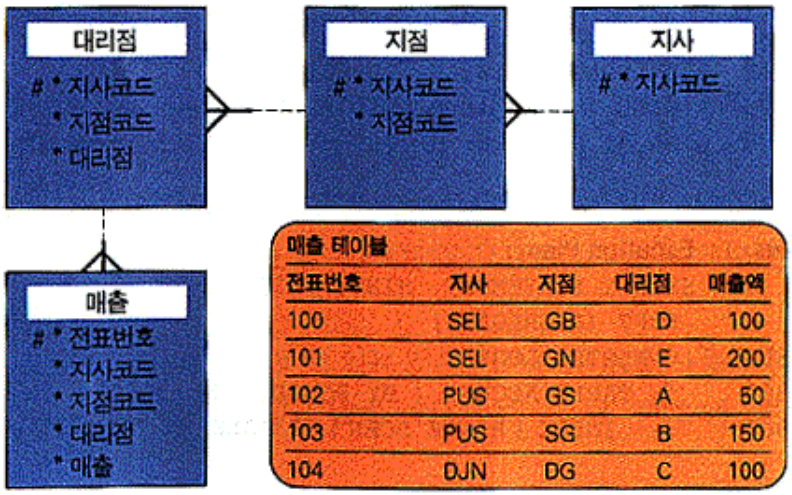


사례 6 DYNAMIC SQL을 WHERE...DECODE...로 해결

소개

이번 사례는 요구사항이 매우 복잡하여 DYNAMIC SQL로 구현해야만 할 것 같지만 발상의 전환으로 새로운 시각에서 바라보면 하나의 SQL로 통합될 수 있음을 보여 주는 사례이다. 이러한 사례를 통하여 우리는 SQL의 무한한 가능성을 엿볼 수 있다. 통계자료를 온라인으로 조회하는 프로그램인데 | 그림 6 | 과 같은 환경이다.

| 그림 6 |



이와 함께 사용되는 회계단위 테이블의 구조와 샘플 데이타는 | 표 1| 과 같다.

회계단위 테이블

	유형그룹	유형	대상1	대상2	대상3	UNIT	ACCT
1	+	B1	SEL	PUS	DNS	1	1
2	+	B1	DGU	USN		1	2
3	+	B2	GB	GN	GS	2	1
4	+	B3	A	B		3	1
5	-	B1	SEL	PUS		1	2
6	-	B2	GN	GD		2	2
7	-	B3	A	B	C	3	2

소개

회계단위 테이블이란 이런 다양한 조건의 통계자료에 대해 여러 가지 WHERE 조건을 정의한 테이블이다. 읽는 방법은 유형 그룹이 '+' 인 기호는 지사, 지점, 대리점 등을 포함하여 통계를 내고 '-' 인 기호는 제외하고 통계를 내고 싶은 것이고, 유형이 B1은 지사의 조건이고 B2는 지점의 조건이고 B3는 대리점의 조건이다. 유형의 실제 값이 들어가는 대상은 대상1, 대상2, 대상3으로 여기서는 3가지만 입력 가능한 것으로 제한하였다. UNIT 컬럼은 ACCT 컬럼의 그때의 조건을 사용하는 회계단위를 UNIQUE하게 정의한 것이다.

예를 들어, 유형 그룹이 '+'인 ①번 행은 회계단위 11번으로 계산을 하는데 매출 테이블에서 서울, 부산, 대전 등 3개 지점의 매출액만 합계를 내고 싶은 것이고, ② 번 행은 회계단위가 12번으로 대구, 울산 지점의 합계를 내고 싶은 것이다. ③번 행은 회계단위 21번으로 강북, 강남, 강서 지점의 합계만 내고 싶은 것이고, 유형 그룹이 '-'인 ⑤번 행은 전체 매출 중에서 서울, 부산 지역을 제외한 매출 합계를 내고 싶은 것이다. 참고로 각 지사, 각 지점, 각 대리점 별로 모두 고유한 코드를 갖고 있다.

이와 같은 요구사항은 일반적인 프로그램 방식을 사용하여 코딩하면 다음과 같다.

```
. . . . .
CURSOR EXP_CURSOR IS SELECT * FROM 회계단위 ;
OPEN EXP_CURSOR ;
LOOP
FETCH EXP_CURSOR INTO EXP_REC;
IF EXP_REC.유형그룹 = '+' AND EXP_REC.유형 = 'B1' THEN
  SELECT SUM(매출액) INTO SUM_SALE
   FROM   매출 WHERE 지사 IN (EXP_REC.대상1, EXP_REC.대상2,
EXP_REC.대상3);
  INSERT INTO 통계테이블 (UNIT, ACCT, MECHUL)
          VALUES (EXP_REC.UNIT, EXP_REC.ACCT, SUM_SALE);
ELSIF EXP_REC.유형그룹 = '+' AND EXP_REC.유형 = 'B2' THEN
  SELECT SUM(매출액) INTO SUM_SALE
   FROM   매출 WHERE 지점 IN (EXP_REC.대상1, EXP_REC.대상2,
EXP_REC.대상3);
  INSERT INTO 통계테이블 (UNIT, ACCT, MECHUL)
     VALUES (EXP_REC.UNIT, EXP_REC.ACCT, SUM_SALE);
ELSIF EXP_REC.유형그룹 = '+' AND EXP_REC.유형 = 'B3' THEN
  SELECT SUM (매출액) INTO SUM_SALE
   FROM   매출 WHERE 대리점 IN (EXP_REC.대상1, EXP_REC.대상2,
EXP_REC.대상3);
  INSERT INTO 통계테이블 (UNIT, ACCT, MECHUL)
     VALUES (EXP_REC.UNIT, EXP_REC.ACCT, SUM_SALE);
ELSEIF EXP_REC.유형그룹 = '-' AND EXP_REC.유형 = 'B1' THEN
  SELECT SUM(매출액) INTO SUM_SALE
   FROM   매출 WHERE 지사 NOT IN (EXP_REC.대상1, EXP_REC.대상
2, EXP_REC.대상3);
  INSERT INTO 통계테이블 (UNIT, ACCT, MECHUL)
```

```
      VALUES (EXP_REC.UNIT, EXP_REC.ACCT, SUM_SALE);
ELSIF EXP_REC.유형그룹 = '-' AND EXP_REC.유형 = 'B2' THEN
  SELECT SUM(매출액) INTO SUM_SALE
   FROM   매출 WHERE 지점 NOT IN (EXP_REC.대상1, EXP_REC.대상
2, EXP_REC.대상3);
  INSERT INTO 통계테이블 (UNIT, ACCT, MECHUL)
      VALUES (EXP_REC.UNIT, EXP_REC.ACCT, SUM_SALE);
ELSIF EXP_REC.유형그룹 = '-' AND EXP_REC.유형 = 'B3' THEN
  SELECT SUM (매출액) INTO SUM_SALE
   FROM   매출 WHERE 대리점 NOT IN (EXP_REC.대상1, EXP_REC.대
상2, EXP_REC.대상3);
  INSERT INTO 통계테이블 (UNIT, ACCT, MECHUL)
      VALUES (EXP_REC.UNIT, EXP_REC.ACCT, SUM_SALE);
END LOOP;
. . . . .
```

한국오라클(주)
서울특별시 강남구 삼성동 144-17
삼화빌딩
대표전화 : 2194-8000
FAX : 2194-8001

한국오라클교육센타
서울특별시 영등포구 여의도동 28-1
전경련회관 5층, 7층
대표전화 : 3779-4242~4
FAX : 3779-4100~1

대전사무소
대전광역시 서구 둔산동 929번지
대전둔산사학연금회관 18층
대표전화 : (042)483-4131~2
FAX : (042)483-4133

대구사무소
대구광역시 동구 신천동 111번지
영남타워빌딩 9층
대표전화 : (053)741-4513~4
FAX : (053)741-4515

부산사무소
부산광역시 동구 초량동 1211~7
정암빌딩 8층
대표전화 : (051)465-9996
FAX : (051)465-9958

울산사무소
울산광역시 남구 달동 1319-15번지
정우빌딩 3층
대표전화 : (052)267-4262
FAX : (052)267-4267

광주사무소
광주광역시 서구 양동 60-37
금호생명빌딩 8층
대표전화 : (062)350-0131
FAX : (062)350-0130