

LEAD 함수 사례

LEAD 함수 사례

현재 수강변경이력(EC_CHANGE_APPLY) 테이블에서는 적용개시일(apply_date)만 존재합니다. 이 적용개시일을 기준으로 이력을 생성하고자 합니다. 예를 들어서, 특정 기준일(예. 2002년 1월 30일)을 기준으로 card_type 유형을 분석하려고 할 때, SQ를 제외한 각 KEY에 대해서 '기준일 보다 작거나 같은 날자 중 최종'을 찾아내야 합니다. 일반 OLTP 성 업무에서는 별로 의미가 없는 내용 일수도 있으나 분석을 주로 하는 DW에서는 아주 빈번하게 사용되는 유형입니다. 다음과 같이 적용종료일이란 가상의 칼럼이 생성되었다고 가정해봅시다. apply_date 순서는 sq 순서와 일치합니다.

course_code	year	course_sq_no	member_no	sq	apply_date	apply_end_date	card_type
11	2002	1	test078619	1	2002-05-02	2002-05-03	01
11	2002	1	test078619	2	2002-05-03	2002-05-15	03
11	2002	1	test078619	3	2002-05-15	2002-05-17	03
11	2002	1	test078619	4	2002-05-17	2002-05-22	01
11	2002	1	test078619	5	2002-05-22	9999-12-31	03

위의 예에서와 같이, **적용개시일과 적용종료일 칼럼이 모두 존재한다면**, 조건 절에서 다음과 같이 사용할 수 있습니다.

where apply_date <= '기준일자' and '기준일자' < apply_end_date

그러나, 적용종료일 칼럼을 물리적으로 만들지 못했다면 SQL로 해결해야 합니다.

다음은 이력을 생성하기 위해서 테이블을 Self Join한 형태입니다. 각 이력에는 순번이 생성되어 있기 때문에 다음 순번의 적용개시일을 자신의 적용종료일로 생성하고 있습니다. 제일 마지막 레코드의 적용종료일은 9999-12-31로 세팅되어야 합니다.

사용된 인덱스는 다음과 같습니다.

EC_CHANGE_APPLY 테이블

EC_CHANGE_APPLY_PK : COURSE_CODE + YEAR + COURSE_SQ_NO + MEMBER_NO + SQ

```
select a.course_code, a.year, a.course_sq_no, a.member_no, a.sq, a.apply_date,
       nvl(b.apply_date, to_date('99991231', 'yyyymmdd')) apply_end_date,
       a.card_type
  from ec_change_apply a, ec_change_apply b
 where a.course_code = b.course_code(+)
       and a.year = b.year(+)
       and a.course_sq_no = b.course_sq_no(+)
       and a.member_no = b.member_no(+)
       and a.sq + 1 = b.sq(+);
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.01	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	1413	0.53	0.41	0	44723	4	21168
total	1415	0.54	0.41	0	44723	4	21168

Rows Row Source Operation

```
-----
21168 NESTED LOOPS OUTER
21169 TABLE ACCESS FULL EC_CHANGE_APPLY
  488 TABLE ACCESS BY INDEX ROWID EC_CHANGE_APPLY
21656 INDEX UNIQUE SCAN (object id 607755) EC_CHANGE_APPLY_PK
```

이력 관리를 NL 조인으로 처리하다 보니, EC_CHANGE_APPLY 테이블을 두 번 읽고 랜덤 액세스까지 사용되면서 **query가 많아졌음**을 볼 수 있습니다.

이러한 **문제를 해결하기 위해**, 조인을 하지 않고서도 이와 같은 이력을 생성 해내는 방법을 **LEAD Analytical Function으로 해결**해 보겠습니다.

LEAD 함수 사례

```
select course_code, year, course_sq_no, member_no, sq, apply_date,  
       lead(apply_date, 1, to_date('99991231', 'yyyymmdd'))  
       over (partition by course_code, year, course_sq_no, member_type, MEMBER_NO  
            order by apply_date) AS appl_end_date,  
       card_type  
from   ec_change_apply;
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.01	0.00	0	0	0	0
Execute	2	0.00	0.00	0	0	0	0
Fetch	1413	0.72	0.79	324	510	9	21168
total	1416	0.73	0.79	324	510	9	21168

Rows Row Source Operation

21168 WINDOW SORT

21168 TABLE ACCESS FULL EC_CHANGE_APPLY

이렇게 했을 경우에는 테이블을 **join** 하지 않고 **Window Sort**로 해결했기 때문에, 테이블을 **한번만 읽는 것으로 처리가 가능**합니다.

결과적으로 disk 및 query의 수를 비교해 보았을 때, 향상된 효과를 볼 수 있습니다.

참고로, 이러한 이력 분석이 중요한 시스템에서는 레코드 이력의 의미를 명확히 하기 위해서 순번 대신에 적용개시일/ 적용종료일을 Key에 포함시키기도 합니다.