```
/*+ MERGE */
/*+ NO_MERGE */

뷰(INLINE VIEW)의 처리가 메인쿼리와 MERGE 되지 않을 때 강제로 MERGE 하도록 하는 힌트.
뷰의 MERGE 란, 뷰를 통해 데이터를 가져오는 작업을 최적화하고자 메인쿼리의 조건들과 병합하여 최소한으로 테이블에 접근할 수 있도록 내부적으로 SQL 문장을 변형시키는 것을 의미한다.
많은 경우 뷰의 MERGE 로 성능이 개선되는 것을 볼 수 있으며, 반대로 뷰가 MERGE 되지 않아야 성능이 좋아지는 경우도 종종 발생한다. 이런 경우 NO_MREGE 힌트를 사용하면 뷰가 MERGE 되는 것을 강제로 막을 수 있다.
```

```
/*+ UNNEST */
/*+ NO_UNNEST */

서브쿼리와 메인쿼리를 합쳐 조인 형태로 실행 계획 변형을 유도하는 힌트.
서브쿼리와 메인쿼리를 합쳐 조인 형태로 실행 계획이 변경되는 것을 막고자 할 때는 NO_UNNEST 힌트를 사용할 수 있으며, 반대로 일부러 조인 형태로 변형하고자 할 때는 UNNEST 힌트를 사용하면 된다.


-- 서브쿼리에서 사용한 사원 테이블을 메인쿼리에서 사용하고 있는 부서 테이블과 직접 조인
select a.부서명
  from 부서 a
 where a.부서번호 in (select b.부서번호
                        from 사원 b
                       where b.부서번호 = a.부서번호)
;

-----------------------------------------------------
Id  Operation                          Name
-----------------------------------------------------
0   SELECT STATEMENT
1     NESTED LOOP
2       SORT UNIQUE
3         INDEX FULL SCAN              IDX_사원_부서
4       TABLE ACCESS BY INDEX ROWID    부서
5         INDEX UNIQUE SCAN            PK_부서
-----------------------------------------------------
```

MERGE: 뷰나 인라인 뷰를 해체해라

UNNEST: 서브쿼리를 해체해라

(해쉬 테이블로 올릴 테이블을 직접 정하고자 할 때 사용)

MERGE와 UNNEST의 공통점

NO_MERGE: 뷰나 인라인 뷰에 사용

(인라인 뷰 = FROM절의 서브쿼리)

NO_UNNEST: 인라인 뷰 외의 WHERE절 등의 서브쿼리에 사용

NO_MERGE와 NO_UNNEST의 공통점은

서브쿼리 (또는 뷰나 인라인 뷰)
를 해체하지 말아라.

라는 의미를 갖고 있습니다.

★ 서브쿼리문에서 unnest의 의미는?

nest ~> 감싸다

un ~> 감싼것을 풀어내는 것

중첩된 상자(서브쿼리)를 풀어내는것 ---> unnest

중첩된 상자(서브쿼리)를 풀어내지 않는 것---> no_unnest

(2) 서브쿼리 Unnesting의 의미

- nest : 상자 등을 차곡차곡 포개넣다. 중첩
- unnest : 중첩된 상태를 풀어낸다.

- 중첩된 서브쿼리는 메인쿼리와 부모와 자식이라는 종속적이고 계층적인 관계가 존재한다.
  따라서 논리적인 관점에서 그 처리과정은 IN, Exists를 불문하고 필터 방식이어야 한다.
  즉, 메인 쿼리에서 읽히는 레코드마다 서브쿼리를 반복 수행하면서 조건에 맞지 않는 데이터를 골라내는 것이다.
  하지만 서브쿼리를 처리하는 데 있어 필터 방식이 항상 최적의 수행속도를 보장하지 못하므로 옵티마이저는 아래 둘 중 하나를 선택한다.
  1. 동일한 결과를 보장하는 조인문으로 변환하고 나서 최적화한다. 이를 일컬어 '서브쿼리 Unnesting'이라고 한다.
  2. 서브쿼리를 Unnesting하지 않고 원래대로 둔 상태에서 최적화한다. 메인쿼리와 서브쿼리를 별도의 서브플랜으로 구분해 각각 최적화를 수행하며, 이때 서브쿼리에 필터(Filter) 오퍼레이션이 나타난다.

(3) 서브쿼리 Unnesting의 이점

- 서브쿼리를 메인쿼리와 같은 레벨로 풀어낸다면 다양한 액세스 경로와 조인 메소드를 평가할 수 있다.
- 서브쿼리 Unnesting과 관련한 힌트로는 아래 두 가지가 있다.
  - unnest : 서브쿼리를 Unnesting 함으로써 조인방식으로 최적화하도록 유도한다.
  - no_unnest : 서브쿼리를 그대로 둔 상태에서 필터 방식으로 최적화하도록 유도한다.

(4) 서브쿼리 Unnesting 기본 예시

'correlated'가 아니어도, No Unnest의 Nested Subquery는 'FILTER' operation을 유발함.

```
select *
  from emp
 where deptno in (select /*+ no_unnest */ deptno from dept)
```

```
-----------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |     5 |   185 |     3   (0)| 00:00:01 |
|*  1 |  FILTER            |         |       |       |            |          |
|   2 |   TABLE ACCESS FULL| EMP     |    14 |   518 |     3   (0)| 00:00:01 |
|*  3 |   INDEX UNIQUE SCAN| PK_DEPT |     1 |     3 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter( EXISTS (SELECT /*+ NO_UNNEST */ 0 FROM "DEPT"
              "DEPT" WHERE "DEPTNO"=:B1))
   3 - access("DEPTNO"=:B1)
```

- 옵티마이저가 서브쿼리를 별도의 서브플랜으로 최적화
- 이처럼, Unnesting하지 않은 서브쿼리를 수행할 때는 메인 쿼리에서 읽히는 레코드마다 값을 넘기면서 서브쿼리를 반복 수행

핵심!! Nested Subquery에 'in' 또는 'EXISTS'가 쓰였을 때 'filter'

□ 서브쿼리와 메인쿼리를 실행 시 조인형태로 실행계획을 수립하도록 하는 힌트가 UNNEST 힌트이며, 반대로 서브쿼리를 그대로 둔 상태에서 필터(FILTER) 방식으로 실행계획이 수립되게 하려면, 즉 조인으로 풀리는 것을 막으려면 NO_UNNEST 힌트를 사용하면 된다.

□ WHERE절에 사용되는 서브쿼리를 중첩 서브쿼리(Nested Subquery)라고 하며 IN, EXISTS 관계없이 메인쿼리에서 읽히는 FILTER방식으로 처리되어 메인레코드 하나 읽을 때마다 서브쿼리를 반복적으로 수행하면서 조건에 맞는 데이터를 추출하는 것이다. 이러한 필터방식이 최적의 성능을 보장하지 않으므로 옵티마이저는 조인문으로 변경후 최적화(Unnesting) 하거나 메인과 서브쿼리를 별도의 SUB PLAN으로 분리하여 각각 최적화를 수행하는데 이때 서브쿼리에 FILTER 연산이 나타난다.

□ 서브 쿼리를 Unesting 하지 않는다면 메인쿼리의 건 마다 서브쿼리를 반복 수행하는 FILTER 연산자를 사용하기에 Unesting 힌트는 효율적으로 사용한다면 성능 향상에 도움이 될 것이다.