```
SELECT /*+ qb_name(main) */ COL1, RND
FROM (SELECT /*+ qb_name(sub) NO_MERGE */ ROWNUM, COL1, RND
      FROM MY_T)
WHERE COL1 <= 10;
```

위와 같이 인라인 뷰 안에 ROWNUM 연산자를 사용한다면 다음과 같이 실행 계획이 생성된다.

```
-----------------------------------------------------------------------------------------------------
| Id | Operation                     |   Name  | Starts| E-Rows | E-Bytes | A-Rows |   A-Time    |
-----------------------------------------------------------------------------------------------------
|* 1 | VIEW                          |         |    1  |  100K  |  2539K  |   10   | 00:00:11.00 |
|  2 |    COUNT                      |         |    1  |        |         |  100K  | 00:00:10.68 |
|  3 |       TABLE ACCESS FULL       |   MY_T  |    1  |  100K  |  1269K  |  100K  | 00:00:09.98 |
-----------------------------------------------------------------------------------------------------
Query Block Name / Object Alias (identified by operation id):
-----------------------------------------------------------------------------------------------------
1 - SUB / from$_subquery$_001@MAIN
2 - SUB
3 - SUB / MY_T@SUB
Predicate Information (identified by operation id):
-----------------------------------------------------------------------------------------------------
1 - filter("COL1"<=10)
```

실행계획을 확인해 보면 ROWNUM이 없는 쿼리는 10건만 읽고 처리했지만 ROWNUM이 있는 쿼리는 대상 테이블을 전체 10만 건을 읽고 난 후에야 Inline-View에 대한 필터로 10건을 처리한 것을 확인할 수 있다. 집합 내에 ROWNUM 연산자가 존재 하는 경우 VIEW-MERGING이나 PUSH-PREDICATE가 발생하지 않고 인라인 뷰가 먼저 처리되는 특성이 있다. Outer 집합에 ROWNUM 값을 제공하기 전에 Inner 집합에서 이미 ROWNUM에 대한 값을 할당해야 하기 때문이다. 이렇게 단지 하나의 단어를 추가했을 뿐이지만 시간상으로는 0.01초에서 11초라는 어마어마한 차이가 존재하게 된다. 일반적으로 많이 사용되는 ROWNUM 연산자에 이러한 특징이 있다는 점을 인지하지 못하고 사용되는 경우가 많고 이로 인해 성능 문제가 발생된다면 의외로 문제를 찾지 못해 힘든 경우가 많다. ROWNUM의 기본적인 특징을 이용해 테이블 간의 조인 순서를 제어하는 등 효과적으로 튜닝하는 사례가 있기 때문에 관심을 가질 필요가 있다.