**4장 조인 튜닝**

**1. NL 조인**

**1.1** 기본 메커니즘

- 조인이란 두 개 이상 테이블을 하나의 집합으로 만드는 연산

- NL 조인은 프로그래밍에서 사용하는 반복문과 유사한 방식으로 수행

- Outer 테이블 : 반복문 외부에 있는 테이블, 선행 테이블, 외부 테이블

- Inner 테이블 : 반복문 내부에 있는 테이블, 후행 테이블, 내부 테이블

- 양쪽 테이블 모두 인덱스 이용 => NL 조인은 ‘인덱스를 이용한 조인 방식’

**- NL 조인하는 과정**

① 선행 테이블의 인덱스에서 조건을 만족하는 첫 번째 행을 찾는다.

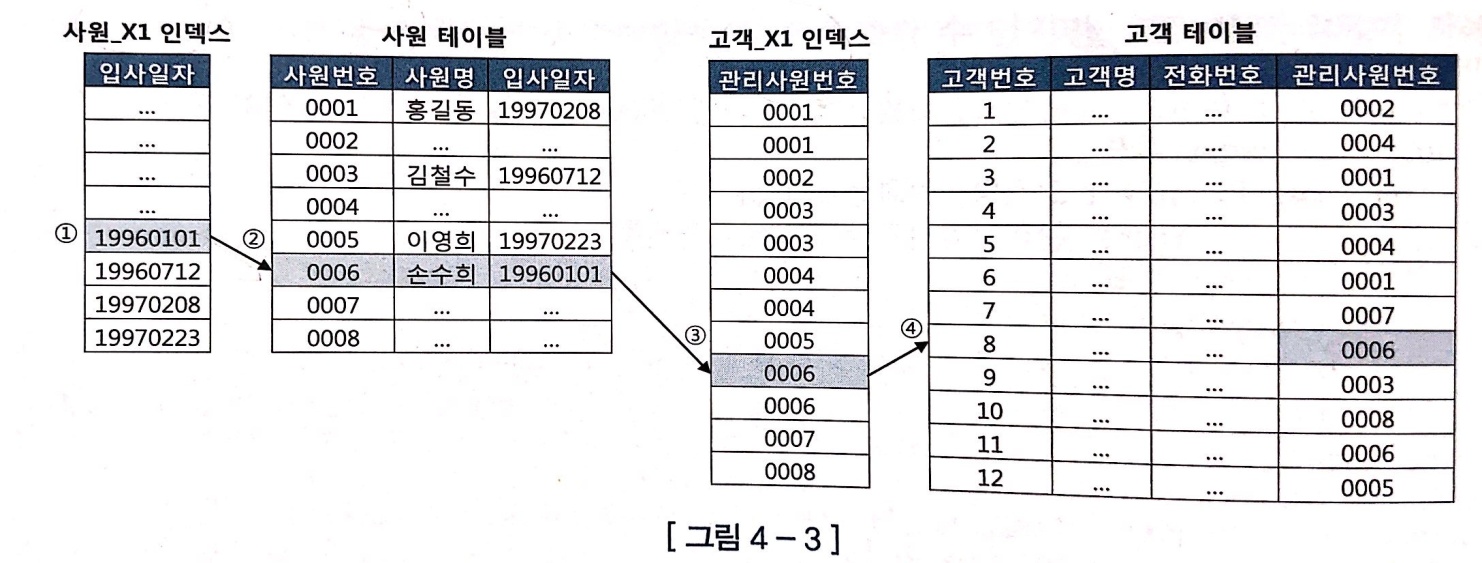
② 인덱스에서 읽은 ROWID로 선행 테이블을 액세스한다.

③ 선행 테이블에서 조건을 만족하는 행의 조인키를 가지고 후행 테이블의 인덱스에

조인키가 존재하는지 찾는다.

④ 인덱스에서 찾은 조인키의 ROWID를 가지고 후행 테이블을 액세스한다.

⑤ ①~④ 작업을 반복 수행한다.



**↑**

조인키

- 선행 테이블의 조건을 만족하는 행을 추출하여 후행 테이블을 읽으면서 조인 수행하기

때문에 선행 테이블의 조건을 만족하는 모든 행의 수만큼 반복 수행

=> 즉, 결과 행의 수가 적은 테이블을 선행 테이블로 선택해야 전체 일량 줄일 수 있음

- NL 조인은 랜덤 액세스이기 때문에 처리 범위가 좁은 것이 유리

**1.2** NL 조인 실행계획 제어

- NL 조인 제어

SELECT **/\*+ ordered use\_nl(c) \*/**

FROM 사원 e, 고객 c

WHERE e.입사일자 >= ‘19960101’

AND c.관리사원번호 = e.사원번호

-> ordered 힌트는 FROM 절에 기술한 순서대로 조인

-> use\_nl 힌트는 NL 방식으로 조인

SELECT **/\*+ ordered use\_nl(B) use\_nl(C) use\_hash(D) \*/**

FROM A, B, C, D

WHERE ....

-> ordered가 있으므로 FROM 절 순서대로 조인( A -> B -> C -> D)

-> A, B, C 는 NL 방식으로 조인, D와는 해시방식으로 조인

SELECT **/\*+ leading(C, A, D, B) use\_nl(A) use\_nl(D) use\_hash(B) \*/**

FROM A, B, C, D

WHERE ....

-> leading 힌트는 순서를 지정하여 조인

-> C, A, D 는 NL 방식으로 조인, B와는 해시방식으로 조인

SELECT **/\*+ use\_nl(A, B. C. D) \*/**

FROM A, B, C, D

WHERE ....

-> 모두 NL 방식으로 조인, 순서는 옵티마이저가 정함.

**1.3** NL 조인 수행 과정 분석

SELECT **/\*+ ordered use\_nl(c) index(e) index(c) \*/**

e.사원번호, e.사원명, e.입사일자,

c.고객번호, c.고객명, c.전화번호, c.최종주문금액

FROM 사원 e, 고객 c

WHERE c.관리사원번호 = e.사원번호

AND e.입사일자 >= ‘19960101’

AND e.부서코드 = ‘Z123’

AND c.최종주문금액 >= 20000

< 인덱스 >

사원\_PK : 사원번호

**사원\_X1 : 입사일자**

고객\_PK : 고객번호

**고객\_X1 : 관리사원번호**

고객\_X2 : 최종주문금액

-> 힌트 : 사원 테이블을 기준으로 고객 테이블과 조인하고 두 테이블 모두

인덱스를 이용해서 액세스한다. 어떤 인덱스를 사용할지는 옵티마이저가 결정.

-> 조건절 순서

① : 사원 테이블의 인덱스는 입사일자로 이루어져 있으며 조건절에 입사일자 범위 조건이

있으므로 사원\_X1 인덱스를 Range 스캔한다.

② : 인덱스에서 읽은 ROWID로 사원 테이블을 액세스하여 조건(**부서코드 = ‘Z123’**) 에

맞는 레코드를 찾는다.

③ : 사원 테이블에서 찾은 레코드의 조인키(사원번호) 값을 고객 테이블의 인덱스에서

찾는다. ( **c.관리사원번호 = e.사원번호** )

④ : 고객 테이블의 인덱스에 읽은 ROWID로 고객 테이블을 액세스하여

조건 (**최종주문금액 >= 20000**) 에 맞는 레코드를 찾는다.

-> 각 단계를 완료하고 다음 단계로 넘어가지 않고 한 레코드씩 순차적으로 진행

**1.4** NL 조인 튜닝 포인트

- 위에서는 단일 인덱스인 사원\_X1에서 입사일자를 ‘>=’ 조건으로 스캔 했을 때 비교적 비효율

없이 읽고 그만큼 테이블 랜덤 액세스 발생함.

하지만, 범위 조건 결과 값이 많고 테이블에서 필터링 되는 비율이 높다면 사원\_X1 인덱스에

부서코드 컬럼을 추가하여 테이블을 액세스하는 양을 줄인다.

- 사원 테이블의 결과 건수 = 고객 테이블의 인덱스를 탐색하는 횟수 = 조인 액세스 횟수이므로 사원 테이블 즉, 선행 테이블의 결과 건수를 줄이는 것이 중요하다.

- 고객\_X1 인덱스를 읽고 고객 테이블을 액세스할 때 조건절에 의해 필터링 되는 비율이 높다면

고객\_X1 인덱스에 최종주문금액 컬럼을 추가한다.

- 맨 처음 사원\_X1 인덱스에서 얻은 결과 건수에 의해 전체 일량이 좌우되므로

결과 행의 수가 적은 테이블을 선행 테이블로 선택해서 전체 일량 줄이도록 한다.

**1.5** NL 조인 특징 요약

- 랜덤 액세스 위주의 조인 방식

-> 레코드 하나를 읽으려고 블록을 통째로 읽는 비효율 발생

- 한 레코드씩 순차적으로 진행

-> 부분범위 처리를 활용하면 빠른 응답 속도를 낼 수 있음.

- 인덱스 구성 전략이 중요

-> 조인 컬럼에 대한 인덱스의 유무와 구성에 따라 조인 효율이 달라짐.

**1.6** NL 조인 튜닝 실습

SELECT /\*+ ordered use\_nl(c) index(e) index(c) \*/

e.사원번호, e.사원명, e.입사일자,

c.고객번호, c.고객명, c.전화번호, c.최종주문금액

FROM 사원 e, 고객 c

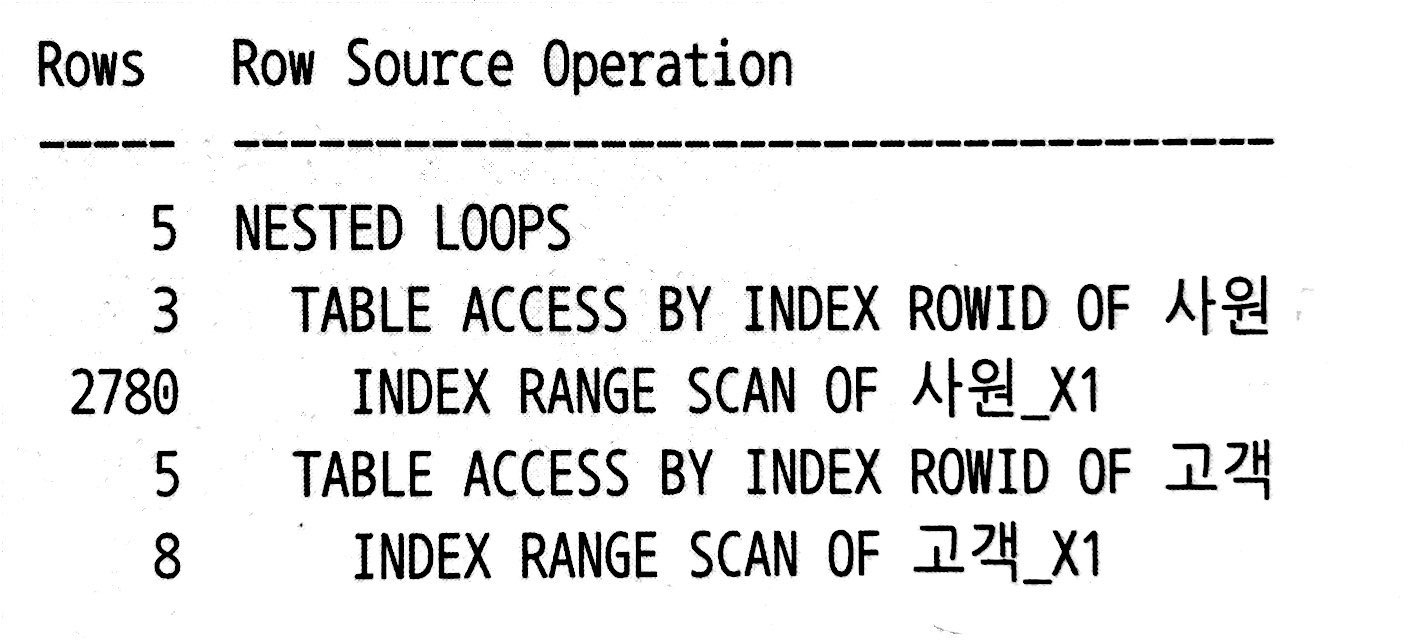
WHERE c.관리사원번호 = e.사원번호

AND **e.입사일자 >= ‘19960101’**

AND **e.부서코드 = ‘Z123’**

AND c.최종주문금액 >= 20000

- 위 쿼리에 대한 SQL 트레이스



**사원\_X1 : 입사일자**

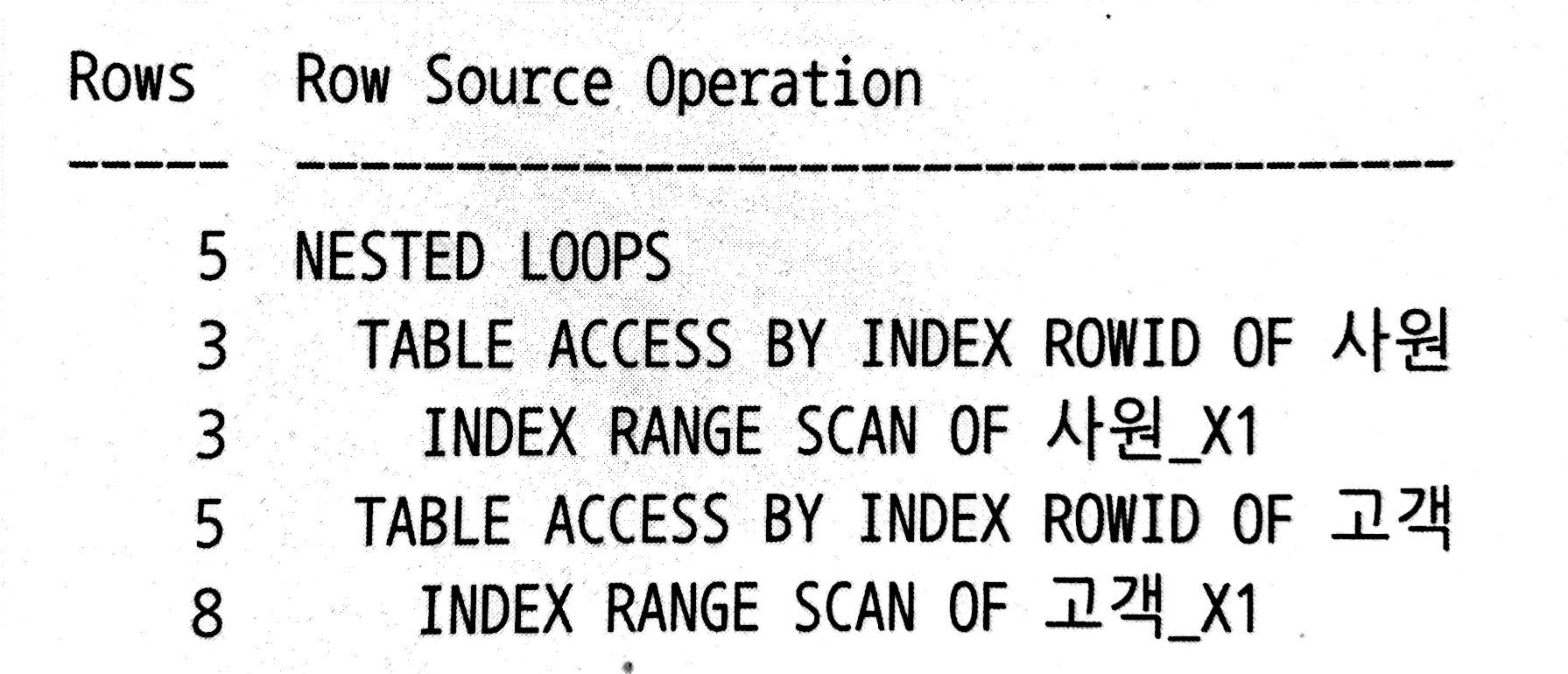
-> 사원\_X1 인덱스 스캔하고 사원 테이블을 액세스 한 횟수 : 2780

-> 테이블에서 조건을 필터링 한 결과건수 : 3

-> 불필요한 테이블 액세스가 많고, 필터링 되는 비율이 높음

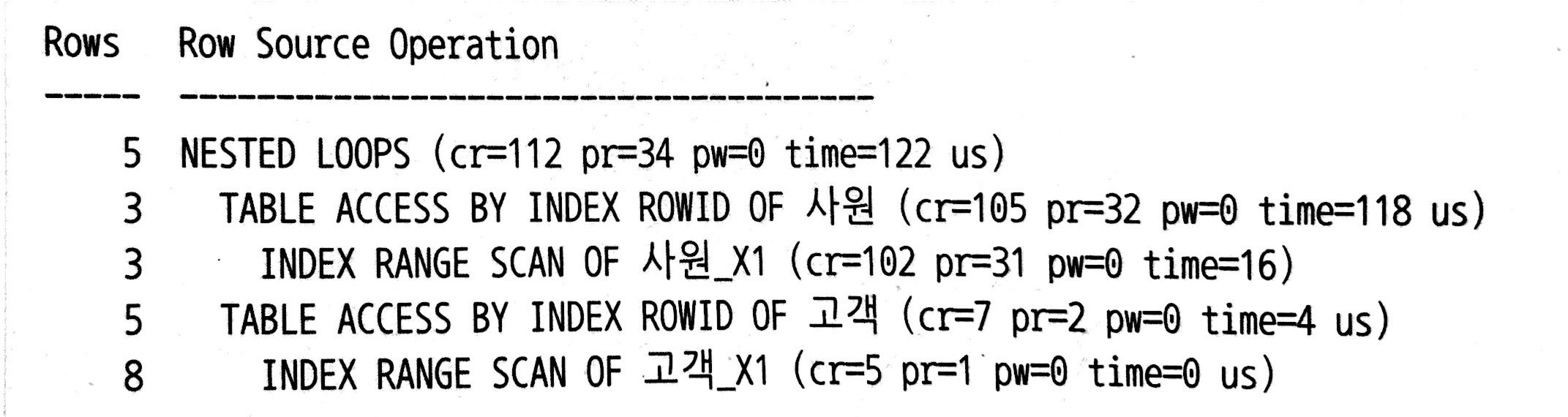
**=> 인덱스에 테이블 필터 조건 컬럼을 추가한다.**

- 인덱스 컬럼 추가 후 SQL 트레이스



**사원\_X1 : 입사일자 + 부서코드**

-> 비효율적인 테이블 액세스는 없어졌지만 액세스하기 전 인덱스 스캔 단계 일량 확인



cr=논리적인 블록 요청 횟수

pr=디스크에서 읽은 블록 수

pw=디스크에 쓴 블록 수

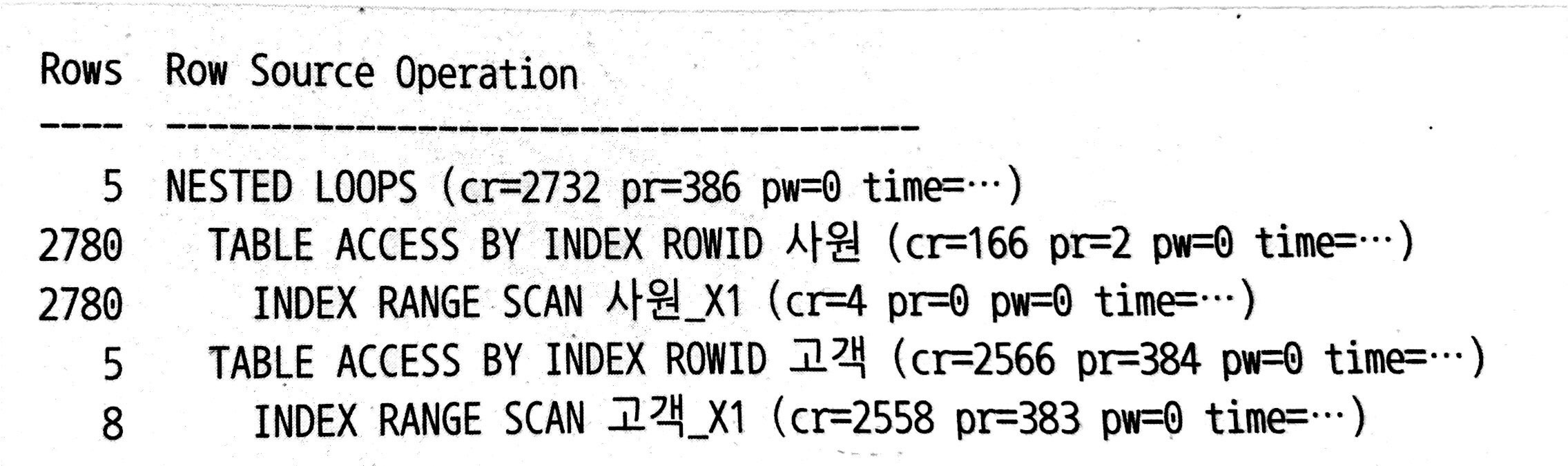
-> 사원\_X1 인덱스로부터 읽은 블록 : 102

-> 한 블록당 평균 500개 레코드가 있다고 가정하면 인덱스에서 3건을 얻기 위해 50,000개의 레코드를 읽은 셈이다.

-> 입사일자 조건이 범위조건이고 부서코드는 ‘=’ 조건이기 때문에 입사일자 조건에 해당하는 레코드가 아주 많다면 발생하는 비효율이다.

**=> [사원\_X1 : 부서코드 + 입사일자] 로 인덱스 컬럼 순서를 조정한다.**

- 최종 트레이스 결과



**사원\_X1 : 부서코드 + 입사일자**

-> 인덱스에서 스캔한 블록 4개, 테이블 액세스하고 필터링 되는 레코드 X

-> 사원 테이블을 읽는 부분에서 일량은 많지만, 비효율은 없음

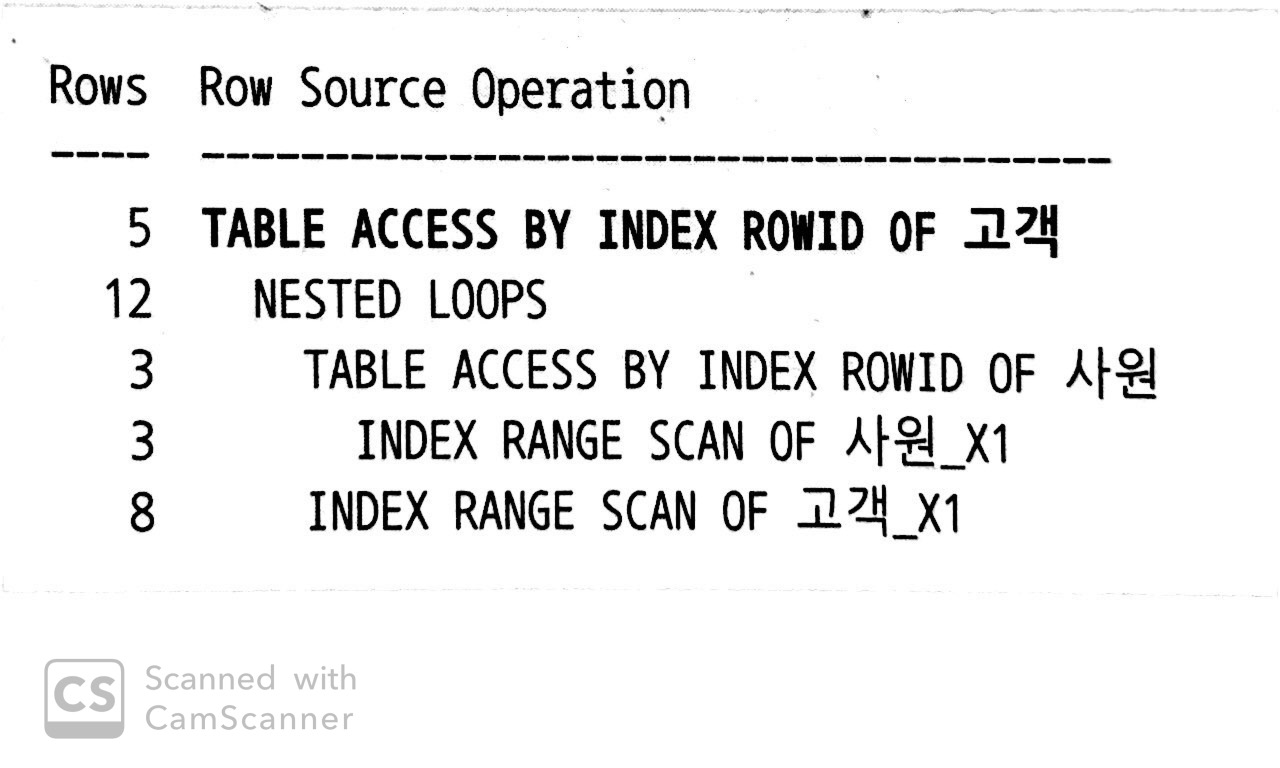
-> 하지만, 고객테이블과 조인하는 부분에서 2780번 조인 시도를 했지만 최종 결과집합은 5건이 므로 조인 순서 변경 고려(순서 변경도 비효율이 발생하면 소트 머지 조인과 해시 조인 검토)

**1.7** NL 조인 확장 메커니즘

- 오라클은 NL 조인 성능을 높이기 위해 ‘테이블 Prefetch’, ‘배치 I/O’ 기능 도입

- **테이블 Prefetch** : 인덱스를 이용해 테이블을 액세스하다가 디스크 I/O가 필요하면 곧 읽게 될 블록까지 미리 읽어서 버퍼캐시에 적재하는 기능

-> 표현 방식



- **배치 I/O** : 디스크 I/O Call을 미뤘다가 블록이 일정량 쌓이면 한꺼번에 처리하는 기능

-> 표현 방식

