**4.3 해시 조인**

▶ 조인 과정에서 인덱스를 사용하지 않음: 대량 데이터 조인할 때 NL 조인보다 훨씬 빠르고, 일정한 성능을 보임.

▶ 소트 머지 조인과 달리 항상 양쪽 테이블을 정렬하는 부담을 가지고 있지 않음.

1. 기본 매커니즘
2. Build 단계: 작은 쪽 테이블(Build Input)을 읽어 해시 테이블(해시 맵)을 생성.
3. Probe 단계: 큰 쪽 테이블(Probe Input)을 읽어 해시 테이블을 탐색하면서 조인.

▶ use\_hash: 옵티마이저가 해시 조인을 수행하도록 유도하는 힌트

▶ 해시 테이블은 PGA 영역에 할당된 Hash Area에 저장.

▶ 해시 테이블이 너무 커 PGA에 담을 수 없으면, Temp 테이블 스페이스에 저장

Ex> 해시 조인 SQL 수행과정 예제

select /\*+ ordered **use\_hash(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

**① Build: 조건에 만족하는 사원 데이터로 해시 테이블 생성**

**: 해시 키 = ‘사원번호’ | 데이터 = ‘사원명’, ‘입사일자’**

**| ‘사원번호’를 해시함수에 입력 | 해시 체인 반환 | 해시 체인에 데이터 연결**

SQL) select 사원번호, 사원명, 입사일자

from 사원 where 입사일자 >= ‘19960101’ and 부서코드 = ‘Z123’

**② Probe: 조건에 만족하는 고객 데이터로 Build 후 생성된 해시 테이블 탐색**

**: 해시 함수 매개변수 = ‘관리사원번호’ | 해시 체인 검색**

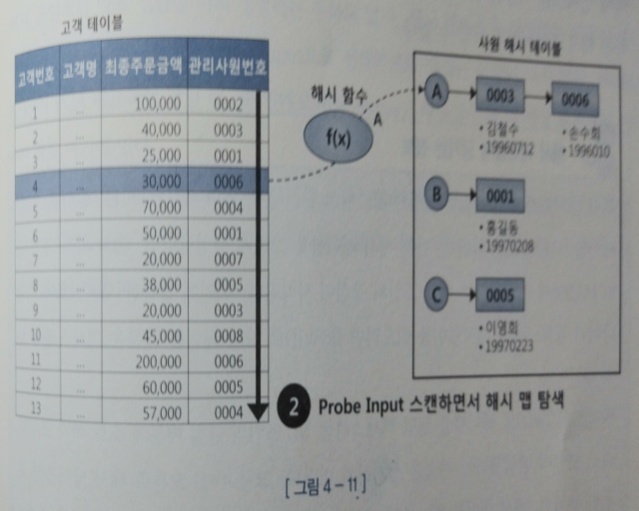
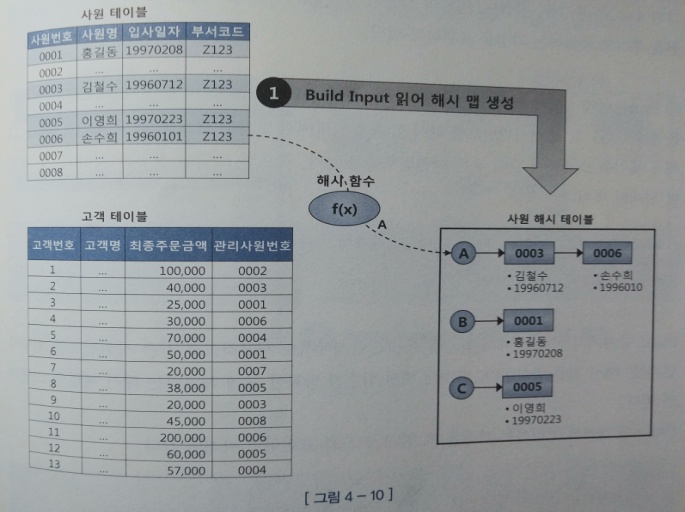
**| 검색된 해시 체인 스캔하여 고객 테이블의 ‘관리사원번호’ 와 같은 ‘사원번호’ 찾아 조인**.

SQL) select 고객번호, 고객명, 전화번호, 최종주문금액, 관리사원번호

from 고객 where 최종주문금액 >= 20000

**등치(=) 조인 적용 - 해시 함수에 사용할 변수**

[ Ex – SQL 구문 해시 조인 처리 구성도 ]



[Ex - Probe 단계 조인 과정을 나타낸 PL/SQL 구문(p.284 참조)]

begin

for outer in (select 고객번호, …from 고객 where 최종주문금액 >= 20000)

loop -- outer 루프

for inner in (select 사원번호, … from PGA에 생성한\_사원\_해시맵 where 사원번호=outer.관리사원번호)

loop -- inner 루프

dbms\_output.put\_line( … );

end loop;

end loop;

end;

☞ Probe 단계는 NL 조인과 다르지 않음.

1. 해시 조인이 빠른 이유
2. 해시 조인 vs. NL 조인

① 해시 테이블을 PGA 영역에 할당: NL 조인과 같이 Outer 테이블 레코드마다 Inner 쪽 테이블 레코드를 읽기 위해 래치 획득 및 캐시 체인 스캔 과정의 반복이 없음.

② 해시 조인도 Build Input과 Probe Input 각 테이블을 읽을 때는 DB 버퍼캐시를 경유하고, 이때 인덱스를 이용하기도 함 – 이 과정에서 발생하는 버퍼캐시 탐색 비용과 랜덤 액세스 부하는 피할 수 없음

1. 해시 조인 vs. 소트 머지 조인

: 둘 다 조인 오퍼레이션을 PGA에서 처리하지만 대량의 데이터를 조인할 때 둘 간의 사전 준비작업 방식의 차이로 인해 일반적으로 해시 조인이 더 빠름

■ 해시 조인 vs. 소트 머지 조인 사전 준비작업 비교

|  |  |  |
| --- | --- | --- |
|  | 해시 조인 | 소트 머지 조인 |
| PGA 할당 대상 | 한쪽  : 해시 테이블, Build Input 대상 | 양쪽  : 대상 집합을 모두 정렬한 후 |
| Temp  테이블스페이스 사용 | 둘 중 작은 집합을 해시 맵 Build Input으로 선택하므로 Temp 영역 사용 가능성 낮음 | 두 집합 중 어느 하나가 중대형 이상이면, Temp 테이블스페이스 사용(디스크에 쓰는 작업) |

☞ NL 조인 처럼 조인 과정에서 발생하는 랜덤 액세스 부하가 없고, 소트 머지 조인처럼 양쪽 집합을 미리 정렬하는 부하도 없음.

▶ 해시 테이블에 담기는 정보: 조인 키 값 + SQL에 사용한 모든 컬럼 데이터

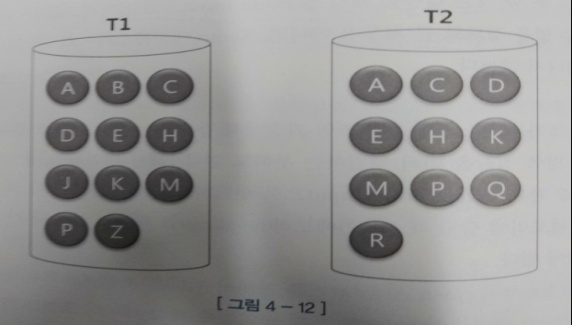
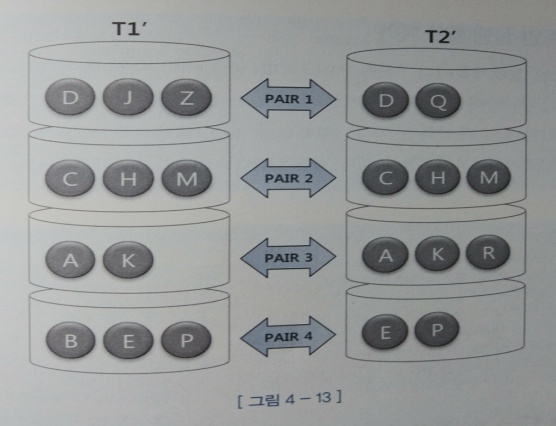
1. 대용량 Build Input 처리

▶ 조인 대상 테이블이 모두 대용량 테이블이어서 인메모리 해시 조인이 불가능한 상황

▶ 분할 \* 정복(Divide & Conquer) 방식 적용

1. 파티션 단계

: 파티션 짝(pair) 생성 단계 – 독립적으로 처리할 수 있는 여러 개의 작은 서브 집합으로 분할

1. 조인하는 양쪽 집합의 조인 컬럼에 해시 함수를 적용
2. 반환된 해시 값에 따라 동적으로 파티셔닝(그림 4-13 참조)

☞ 양쪽 집합(T1, T2)을 읽어 디스크 Temp 공간에 저장(T1’, T2’)

※ 파티셔닝(Partitioning): 테이블 또는 인덱스 데이터를 특정 컬럼(파티션 키) 값에 따라 별도 세그먼트에 나눠서 저장하는 것. - [ 6.3.1 참조 ]

1. 조인 단계

: 각 파티션 짝(pair)에 대해 하나씩 조인을 수행

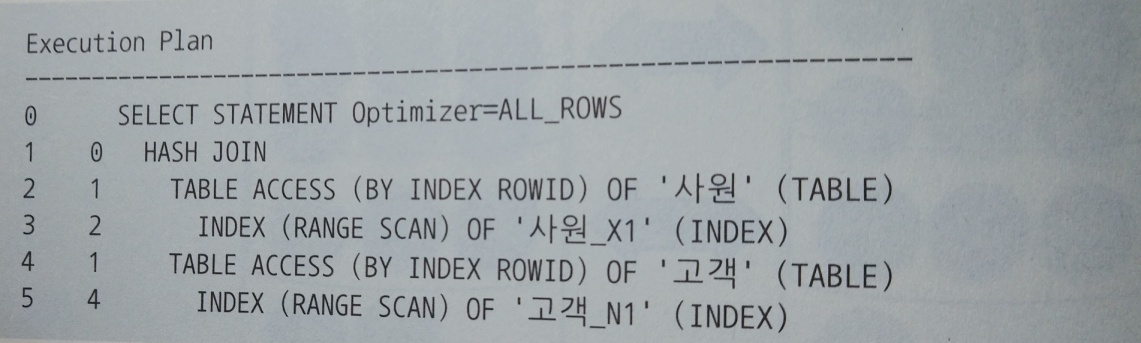
각각에 대한 Build Input과 Probe Input은 독립적으로 결정됨. 즉, 파티션하기 전 어느 쪽이 작은 테이블이었는지에 상관없이 각 파티션 짝(pair)별로 작은 쪽을 Build Input으로 선택하고 해시 테이블을 생성

반대편 파티션 로우를 하나씩 읽으면서 해시 테이블을 탐색, 모든 파티션 짝에 대한 처리를 마칠 때까지 이 과정을 반복.

▶ **In-Memory Hash Join(한 테이블의 카디널리티가 적은 경우)와 대용량 Build Input 비교**

|  |  |
| --- | --- |
| In-Memory Hash Join 처리 단계 | 대용량 Build Input 처리 단계 |
| 1. 작은 쪽 테이블(Build Input)을 읽어 해시 테이블(해시 맵)을 생성 2. 큰 쪽 테이블(Probe Input)을 읽어 해시 테이블을 탐색하면서 조인 | 1. 파티션 짝(pair) 생성 및 Temp 공간에 저장 2. Temp 공간에 생성된 각 파티션 로우 짝 중 작은 쪽을 Build Input으로 선택하고 해시 테이블 생성 3. 반대편 파티션 짝(Probe Input) 로우를 하나씩 읽으면서 해시 테이블 탐색하면서 조인 |

1. 해시 조인 실행계획 제어



☞ 위쪽(HASH JOIN 바로 아래) 사원 데이터(Build Input)로 해시 테이블을 생성한 후, 아래쪽 고객 테이블(Probe Input)에서 읽은 조인 키값으로 해시 테이블을 탐색하면서 조인

▶ Build Input 대상 지정

1) 옵티마이저: 일반적으로 조인 대상 테이블 중 카디널리티(조건절에 대한 카디널리티)가 작은 테이블을 선택

2) 사용자 지정 - [p.291 쿼리 참조]

▷ leading이나 ordered 힌트를 사용

Ex) /\*+ leading(e) use\_hash(c) \*/ or /\*+ ordered use\_hash(c) \*/

▷ swap\_join\_inputs: Build Input 대상을 명시적으로 선택

Ex) /\*+ leading(e) use\_hash(c) swap\_join\_inputs(c) \*/: 고객 테이블 Build Input

/\*+ leading(e) use\_hash(c) swap\_join\_inputs(e) \*/: 사원 테이블 Build Input

▣ 세 개 이상 테이블 해시 조인

▷ 해시 조인에서 leading 힌트 첫 번째 파라미터로 지정한 테이블은 무조건 Build Input으로 선택 됨.

Ex) /\*+ leading(T1, T2, T3) use\_hash(T2) use\_hash(T3) \*/ - [p.293 쿼리 참조, 실행 계획 참조]

▷ swap\_join\_inputs: Build Input 대상을 명시적으로 선택.

Ex) /\*+ leading(T1, T2, T3) swap\_join\_inputs(T2) \*/ - [p.294 쿼리 참조, 실행 계획 참조]

☞ **swap\_join\_inputs** 옵션 적용에 따른 실행 계획 비교

|  |  |
| --- | --- |
| SQL> select \* from T1, T2, T3 where T1.key = T2.key and T2.key = T3.key | |
| /\*+ leading(T1, T2, T3) use\_hash(T2) use\_hash(T3) \*/ | /\*+ leading(T1, T2, T3) swap\_join\_inputs(T2) \*/ |
| 20200420_234234(1).jpg | 20200420_234304(1).jpg |

▷ no\_swap\_join\_inputs: Probe Input 대상을 지정(오라클 10.1.0.3 버전부터 제공).

Ex) /\*+ leading(T1, T2, T3) no\_sqap\_join\_inputs(T3) \*/ - [p.295 첫 번째 쿼리 참조]

※ no\_swap\_join\_inputs 없을 시 인라인 뷰(from 절 뒤에 오는 서브쿼리)를 생성하고 그 인라인 뷰 Alias로 제어 - [p.295 두 번째 쿼리 참조]

5. 조인 메소드 선택 기준

|  |  |
| --- | --- |
| 조인 메소드 | 선택 기준 |
| NL 조인 | 소량 데이터(랜덤 액세스가 많아 만족할 만한 성능을 낼 수 없는) 조인할 때 |
| 해시 조인 | 대량 데이터 조인할 때 – 조인 조건식에 등치(=) 조건 사용 |
| 소트 머지 조인 | 대량 데이터 조인인데 해시 조인으로 처리할 수 없을 때 |

▶ 수행빈도가 매우 높은 쿼리 처리 선택 기준

① (최적화된) NL 조인과 해시 조인 성능이 같으면, NL 조인

② 해시 조인이 약간 더 빨라도 NL 조인

③ NL 조인보다 해시 조인이 매우 빠른 경우, 해시 조인

▶ 조인 메소드를 선택할 때 NL 조인을 가장 먼저 고려해야 하는 이유

: NL 조인에 사용하는 인덱스는 영구적으로 유지하면서 다양한 쿼리를 위해 공유 및 재사용하는 자료구조인 반면, 해시 테이블은 단 하나의 쿼리를 위해 생성하고 조인이 끝나면 소멸하는 자료구조

☞ 수행시간이 짧으면서 수행빈도가 매우 높은 쿼리를 해시 조인으로 처리하면 CPU와 메모리 사용률이 크게 증가

▶ 해시 조인을 사용하는 SQL문 조건

① 수행 빈도가 낮고

② 쿼리 수행 시간이 오래 걸리는

③ 대량 데이터 조인할 때

☞ 배치 프로그램, DW, OLAP성 쿼리의 특성을 가짐.