소트 튜닝

소트 수행 과정

- 소트는 기본적으로 PGA에 할당한 Sort Area에서 이루어진다.
- 메모리 공간인 Sort Area가 다 차면, 디스크 Temp 테이블 스페이스를 활용한다.

소트 유형

Sort Area에서 작업을 완료할 수 있는지에 따라 두 가지로 나눈다.

- **메모리 소트(In-memory Sort)**: 전체 데이터의 정렬 작업을 메모리 내에서 완료하는 것을 말하며, 'internal Sort'라고도 한다.
- **디스크 소트(To-Disk Sort)**: 할당받은 Sort Area 내에서 정렬을 완료하지 못해 디스크 공간까지 사용하는 경우를 말하며, 'External Sort'라고도 한다.

디스크 소트 과정

1. 소트할 대상 집합을 SGA 버퍼캐시를 통해 읽어들인다.
2. 일차적으로 Sort Area에서 정렬을 시도한다.
3. 양이 많을 때는 정렬된 중간집합을 Temp 테이블스페이스에 임시 세그먼트를 만들어 저장한다.
    a. Sort Area가 찰 때마다 Temp 영역에 저장해 둔 중간 단계의 집합을 'Sort Run'이라고 부른다.
4. 각 결과 집합을 Merge하여 정렬된 최종 결과집합을 가져온다.

소트 연산은 메모리 집약적(Memory-intensive)일 뿐만 아니라 CPU 집약적(CPU-intensive)이기도 하다. 처리할 데이터량이 많을 때는 Disk I/O까지 발생하므로 쿼리 성능을 좌우하는 매우 중요한 요소다.

디스크 소트가 발생하는 순간 SQL 수행 성능은 나빠질 수 밖에 없다.

많은 서버 리소스를 사용하고 디스크 I/O가 발생하는 것도 문제지만, 부분범위 처리를 불가능하게 함으로써 OLTP 환경에서 애플리케이션 성능을 저하시키는 주요인이 되기도 한다.

소트 오퍼레이션

Sort Aggregate

전체 로우를 대상으로 집계를 수행할 때 나타난다.

'Sort'라는 표현을 사용하지만, 실제로 데이터를 정렬하진 않는다. Sort Area를 사용한다는 의미로 이해하면 된다.

Sort Order By

데이터를 정렬할 때 나타난다.

Sort Group By

소팅 알고리즘을 사용해 그룹별 집계를 수행할 때 나타난다.

> 오라클 10gR2 버전에서 도입된 Hash Group By 방식으로 인해 Group By 절 뒤에 Order By 절을 명시하지 않으면 이제 대부분 Hash Group By 방식으로 처리한다.

ℹ️ **그룹핑 결과가 정렬 순서를 보장하지 않는다.**
정렬된 그룹핑 결과를 얻고자 한다면, 실행계획에 설령 'Sort Group By'라고 표시되더라도 반드시 Order By를 명시해야 한다.

Sort Unique

옵티마이저가 서브쿼리를 풀어 일반 조인문으로 변환하는 것을 '서브쿼리 Unnesting'이라고 한다.

Unnesting된 서브쿼리가 M쪽 집합이면, 메인 쿼리와 조인하기 전에 중복 레코드부터 제거해야 한다.

만약 PK/Unique 제약 또는 Unique 인덱스를 통해 Unnesting된 서브쿼리의 유일성이 보장된다면, Sort Unique 오퍼레이션은 생략된다.

추가적으로 Union, Minus, Intersect, Distinct를 사용할 때도 Sort Unique 오퍼레이션이 나타나낟.

> 오라클 10gR2부터는 Distinct 연산에도 Hash Unique 방식을 사용한다.

Sort Join

소트 머지 조인을 수행할 때 나타나낟.

Window Sort

윈도우 함수를 수행할 때 나타난다.

소트가 발생하지 않도록 작성하기

Union vs Union All

Union을 사용하면 옵티마이저는 상단과 하단 두 집합 간 중복을 제거하려고 소트 작업을 수행한다. 반면, Union All은 중복을 확인하지 않고 두 집합을 단순히 결합하므로 소트 작업을 수행하지 않는다.

따라서 될 수 있으면 Union All을 사용해야 한다.

- 집합 사이에 인스턴스 중복 가능성이 없을 경우 단순히 바꿔 사용할 수 있다.

```
select 결제번호
from 결제
where 결제일자 = '20180316'
UNION
select 결제번호
from 결제
where 주문일자 = '20180316'
```

위와 같이 중복 가능성이 있을 경우 다음과 같이 변경하여 사용할 수 있다.

```
select 결제번호
from 결제
where 결제일자 = '20180316'
UNION ALL
select 결제번호
from 결제
where 주문일자 = '20180316'
and 결제일자 <> '20180316'
```

Exists 활용

중복 레코드를 제거할 목적으로 Distinct 연산자를 종종 사용하는데, 이 연산자를 사용하면 조건에 해당하는 데이터를 모두 읽어서 중복을 제거해야 한다.

부분범위 처리는 당연히 불가능하고, 모든 데이터를 읽는 과정에 많은 I/O가 발생한다.

```
select DISTINCT p.상품번호, ...
from 상품 p, 계약 c
where p.상품유형코드 = :pclscd
and c.상품번호 = p.상품번호
and c.계약일자 between :dt1 and :dt2
and c.계약구분코드 = :ctpcd
```

```
select DISTINCT p.상품번호, ...
from 상품 p, 계약 c
where p.상품유형코드 = :pclscd
and EXISTS(select 'x' from 계약 c
           where c.상품번호 = p.상품번호
           and c.계약일자 between :dt1 and :dt2
           and c.계약구분코드 = :ctpcd)
```

Exists 서브쿼리는 데이터 존재 여부만 확인하면 되기 때문에 조건절을 만족하는 데이터를 모두 읽지 않는다.

조인 방식 변경

계약_X01 인덱스가 [지점ID + 계약일시] 순이면 소트 연산을 생략할 수 있다.

```
select c.계약번호, ...
from 계약 c, 상품 p
where c.지점ID = :brch_id
and p.상품코드 = c.상품코드
order by c.계약일시 desc
```

NL 조인하도록 조인 방식을 변경하면 솥트 연산을 생략할 수 있어 지점 ID 조건을 만족하는 데이터가 많고 부분범위 처리 가능한 상황에서 큰 성능 개선 효과를 얻을 수 있다.

```
select /*+ leading(c) use_nl(p) */c.계약번호, ...
from 계약 c, 상품 p
where c.지점ID = :brch_id
and p.상품코드 = c.상품코드
order by c.계약일시 desc
```