**제 5장 고급 SQL 튜닝**

1. **고급 SQL 활용**

(1) CASE문 활용

- 여러 번 테이블을 ACCESS하는 경우 CASE문을 활용하여 1번만 액세스

EX) NVL( SUM (CASE WHEN COLUMN = VALUE THEN COLUMN END), 0 )

(2) 데이터 복제 기업 활용 -> 고급 조인 테크닉 참조

(3) UNION ALL을 활용한 M:M 관계의 조인

- FULL OUTER JOIN을 대체할 용도로 UNION ALL 활용 가능

-> FULL OUTER JOIN을 할 경우 테이블을 각각 2회씩 ACCESS, UNION ALL 1번

(4) 페이징 처리 -> SKIP

(5) WINDOW 함수 활용 -> 전에 했음

(6) WITH 구문 활용

- 자주 쓰는 SELECT FROM 절일 경우 WITH 구문을 통해 한번 선언하고 테이블처럼 어디에서나 이용 가능하게 활용 가능하다.

**2. 소트 튜닝**

(1) 소트와 성능

가) 메모리 소트 / 디스크 소트

- 메모리 소트 : 정렬작업을 메모리 내에서. INTERNAL (OPTIMAL) SORT라고도 함

- 디스크 소트 : 메모리 내에서 못하여 디스크 공간까지 사용. 느림. EXTERNAL SORT

- PGA -> CGA / UGA

- CGA : CALL이 진행되는 동안만 필요한 정보 저장 (PRIVATE 변수)

- UGA : CALL을 넘어 다음 CALL 까지 계속 참조되는 정보 저장 (PUBLIC 변수)

- DML 문장, SELECT 문장 중 한 단계 앞선 단계의 정렬은 CGA

- SELECT 문장 중 가장 마지막 단계에서 정렬된 데이터는 UGA

나) 소트를 발생시키는 오퍼레이션

(1) SORT AGGREGATE : AGGRREGATE 함수 사용시 발생. BUT 실제 소트 발생 X

(2) SORT ORDER BY : ORDER BY 절 사용할 때

(3) SORT GROUP BY : 그룹별 집계를 수행할 때 발생. ORDER BY 있을 땐 SORT GROUP BY, 없을 경우 HASH GROUP BY. 결과값이 정렬 되지 않음

(4) SORT UNIQUE -> 쿼리 변환 본 후 볼 것!

(5) SORT JOIN : 소트머지 조인 할 경우

(6) WINDOW SORT : 분석함수를 수행할 때 발생

다) 소트튜닝 요약

(1) 데이터 모델 측면에서의 검토 : 모델만 잘 구성하면 단순하게 SQL 작성 가능

(2) 소트가 발생하지 않도록 SQL 작성

- UNION을 UNION ALL로 대체 : UNION은 중복값 없게 하므로

**- DISTINCT를 EXISTS로 대체** : DISTINCT를 할 경우 모든 테이블을 다 읽어야 하지만 EXISTS는 SEMI JOIN으로 하나만 찾으면 끝나는 JOIN이다.

- 불필요한 COUNT 연산 제거 : ROWNUM <= 1로 더 이상 읽지 않도록 대체할 것.

(3) 인덱스를 이용한 소트 연산 대체

- 인덱스는 정렬이므로 INDEX에 들어간 컬럼이 ORDER BY될 경우 SORT가 없음

- GROUP BY도 같은 원리로 SORT GROUP BY가 이루어짐

(4) SORT AREA를 적게 사용하도록 SQL 작성

- SORT를 완료하고 데이터를 가공

- TOP-N 쿼리 : ROWNUM을 활용함 -> STOPKEY가 나와서 적게 SORT함

-> 이 경우 처음 정해진 범위만큼의 레코드가 정렬된 상태로 맨 좌우측 값과 비교 후 그것과 큰거나 작은 것이 나온 경우 배열 내에서 재정렬 후 최종 정해진 범위만큼만 남음

- WINDOW SORT시에도 RANK() 및 ROW\_NUMBER() 사용시 작동됨

- MAX() 등의 함수보다 부하 경감됨

- TOP-N 쿼리의 경우 INLINE VIEW 바로 뒤에 써야 적용이 된다.

(5) 소트 영역 크기 조정

- WORK AREA SIZE 관리 : SORT, HASH, BITMAP\_MERGE, CREATE\_BITMAP 등

- 오라클 8i 이하는 DBA에 의한 크기 조정, 9i 부터는 자동

-> WORKAREA\_SIZE\_POLICY = AUTO / MANUAL 중 하나로 선택해 사용가능

- SORT AREA 할당 및 해체 : 최초(수행 직전), 수행 중(수행 중이지만 결과 출력 X), 완료 후(결과 출력 중이지만 FETCH X), 커서를 닫은 후(다른 쿼리 수행 OR 끝까지 FETCH)

- 수동 PGA 메모리(AREA\_SIZE) 관리 방식 시에 PARALLEL DEGREE에 주의 할 것!

**3. DML 튜닝**

(1) 인덱스 유지 비용

- 테이블 데이터 변경시 인덱스도 변경

- 변경할 인덱스 레코드 찾는 비용, REDO, UNDO를 생성하는 비용 추가

- 인덱스가 많을수록 DML 성능 저하

- UPDATE 수행 시 테이블은 직접 변경, 인덱스는 DELETE & INSERT 방식

- 인덱스는 항상 정렬된 상태로 유지, 따라서 인덱스 유지를 위한 UNDO도 2개씩 기록

- UPDATE시 관련 인덱스 수에 따라 성능 좌우

- INSERT, DELETE는 인덱스 모두에 변경을 가해야 하므로 총 인덱스 수에 비례

- 대량의 데이터 입력/수정/삭제 시 인덱스 DROP또는 UNUSABLE 상태 변경 후 인덱스 재생성이 더 빠를 수도 있음

(2) INSERT 튜닝

- DIRECT PATH INSERT : 일반 HEAP TABLE과 달리 FREELIST를 거치지 않고 HWM 상위 영역에 버퍼캐시를 거치지 않고 직접 데이터 파일에 입력하고 UNDO에 데이터를 쌓지 않아 속도 향상된다. (사용자가 COMMIT할 때만 HWM을 상향 조정하면 되기 때문)

-> INSERT SELECT 문장에 /\*+ APPEND \*/ 힌트를 사용하며 병렬 모드로 INSERT. DIRECT 옵션을 지정하고 SQL LOADER(SQLLDR)로 데이터 로드.

- NOLOGGING MODE INSERT : 테이블 속성을 NOLOGGING으로 변경 시 REDO LOG까지 최소화. -> DATA DICTIONARY 변경 시에만 로깅, DIRECT PATH INSERT 일 때만 작용. 일반적인 INSERT엔 X. “ALTER TABLE T NOLLOGING;” 형식으로 사용

\*주의 : EXCLUSIVE MODE TABLE LOCK으로 DDL LOCK으로 OLTP 사용 불가하며 입력한 데이터 장애 발생 시 복구가 불가능

(3) UPDATE 튜닝 : 조인 부분 참고