

82. chatgpt 를 활용하여 PL/SQL 튜닝하기

for loop문과 forall loop문의 문법 비교

1. 관련 테이블 생성

```
drop table test1 purge;
drop table test2 purge;
create table test1 (c1 number);
create table test2 (c1 number);
```

2. for loop문을 사용한 프로시저 생성

```
create or replace procedure for_pro
is
begin
    for i in 1..1000000 loop
        insert /*+for */ into test1
            values(i);
    end loop;
end;
/
```

3. forall loop문을 사용한 프로시저 생성

```
create or replace procedure for_all_pro
is
    type tnt is table of pls_integer;
    l_v1 tnt:= tnt();
begin
    for i in 1..1000000 loop
        l_v1.extend;
        l_v1(i):=i;
    end loop;

    forall i in l_v1.first ..l_v1.last
        insert/*+ for_all*/ into test2
            values(l_v1(i));
end;
/
```

4. 두 프로시저의 수행 속도 차이 확인

```
set timing on
exec for_pro;
exec for_all_pro;
```

5. chatgpt한테 sql 튜닝 시키기

데이터 삽입 프로시저의 성능을 향상시키기 위해, 이미 제시한 **FORALL** 사용 방식이 가장 효과적인 접근 방법 중 하나입니다. 그러나 여기에서는 기존의 **FORALL** 기반 프로시저를 더 최적화하는 방법에 초점을 맞추겠습니다. 주요 목표는 가능한 최소한의 코드 변경으로 성능을 향상시키는 것입니다.

튜닝된 **FORALL** 기반 프로시저

```
sql Copy code

CREATE OR REPLACE PROCEDURE for_all_pro_optimized
IS
    TYPE tnt IS TABLE OF pls_integer INDEX BY PLS_INTEGER;
    l_v1 tnt;
BEGIN
    -- 데이터를 배열에 할당
    FOR i IN 1..1000000 LOOP
        l_v1(i) := i;
    END LOOP;

    -- BULK INSERT 실행
    FORALL i IN 1..1000000
        INSERT INTO test2 VALUES (l_v1(i));

    -- 성능을 위해 단일 COMMIT 사용
    COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        -- 에러 처리
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

최적화 포인트

1. 단일 **COMMIT** 사용: 대량의 데이터 삽입 후 한 번만 **COMMIT**을 실행하여 트랜잭션 관리 오버헤드를 줄입니다. 이 방식은 로그 쓰기 작업을 최소화하고, 성능을 크게 향상시킬 수 있습니다. 다만, 실제 운영 환경에서는 삽입 데이터의 양과 데이터베이스의 용량, 트랜잭션의 크기에 따라 적절한 **COMMIT** 전략을 선택해야 합니다.
2. 예외 처리 추가: 대량의 데이터를 처리할 때 예기치 않은 에러가 발생할 수 있으므로, 예외 처리 로직을 추가하여 오류 발생 시 **ROLLBACK**을 수행하고, 에러 메시지를 출력합니다. 이는 데이터 일관성을 유지하고, 문제 발생 시 빠른 진단을 가능하게 합니다.
3. 인덱스 사용: **FORALL** 문 내에서 사용된 배열(**l_v1**)은 **INDEX BY PLS_INTEGER**로 선언되어 있어, 인덱스 기반 컬렉션을 사용합니다. 이는 PL/SQL에서 대량의 데이터를 효율적으로 처리할 수 있게 해줍니다.

이러한 최적화를 통해, 이미 효율적인 **FORALL** 접근 방식을 더욱 최적화할 수 있습니다. 실제 성능 향상은 데이터베이스의 현재 상태, 하드웨어 자원, 실행 환경 등 다양한 요소에 따라 달라질 수 있습니다. 따라서, 최적화한 코드를 실제 환경에서 테스트하여 그 효과를 확인하는 것이 중요합니다.