

과목 III

SQL 고급활용 및 튜닝

제1장 아키텍처 기반 튜닝 원리

- 제1절 데이터베이스 아키텍처
- 제2절 SQL 파싱 부하
- 제3절 데이터베이스 Call과 네트워크 부하
- 제4절 데이터베이스 I/O 원리

제2장 Lock과 트랜잭션 동시성제어

- 제1절 Lock
- 제2절 트랜잭션
- 제3절 동시성 제어

제3장 옵티마이저 원리

- 제1절 옵티마이저
- 제2절 쿼리변환

제4장 인덱스와 조인

- 제1절 인덱스 기본 원리
- 제2절 인덱스 튜닝
- 제3절 조인 기본 원리
- 제4절 고급 조인 기법

제5장 고급 SQL 튜닝

- 제1절 고급 SQL 활용
 - 제2절 소스 튜닝
 - 제3절 DML 튜닝
 - 제4절 파티션 활용
 - 제5절 배치 프로그램 튜닝
-

The background of the top banner is a grayscale photograph of a city skyline, possibly featuring a prominent pyramid-like structure. A white grid pattern is overlaid on the image, creating a technical or architectural feel.

SQL

Professional • Developer

제1장

아키텍처 기반 튜닝 원리

-
- 제1절 데이터베이스 아키텍처
 - 제2절 SQL 파싱 부하
 - 제3절 데이터베이스 Call과 네트워크 부하
 - 제4절 데이터베이스 I/O 원리
-

핵심정리

클라이언트가 서버 프로세스와 연결하는 Oracle의 예

- 1) 전용 서버
(Dedicated Server)
방식
- 2) 공유 서버
(Shared Server)
방식

SQL Server에서는
세그먼트는 테이블,
인덱스, Undo처럼
저장공간을 필요로 하는
데이터베이스 오브젝트다.
저장공간을 필요로 한다는
것은 한 개 이상의
익스텐트를 사용함을
뜻한다.

SQL Server에서는
세그먼트라는 용어를
사용하지 않지만,
힙 구조 또는 인덱스
구조의 오브젝트가
여기에 속한다.

DB 버퍼 캐시에 가해지는
모든 변경사항을 기록하는
파일을 Oracle은 'Redo
로그'라고 부르며, SQL
Server는 '트랜잭션
로그'라고 부른다.

1

다음 중 데이터베이스 연결(Connection)과 관련한 설명으로 가장 부적절한 것은?

- ① 데이터베이스 서버와 클라이언트 간 연결상태를 유지하면 서버 자원을 낭비하게 되므로 동시 사용자가 많은 OLTP 환경에선 SQL 수행을 마치자마자 곧바로 연결(Connection)을 닫아주는 것이 바람직하다.
- ② 연결(Connection) 요청에 대한 부하는 스레드(Thread) 기반 아키텍처보다 프로세스 기반 아키텍처에서 더 심하게 발생한다.
- ③ 전용 서버(Dedicated Server) 방식으로 오라클 데이터베이스에 접속하면 사용자가 데이터베이스 서버에 연결 요청을 할 때마다 서버 프로세스(또는 스레드)가 생성된다.
- ④ 공유 서버(Shared Server) 방식으로 오라클 데이터베이스에 접속하면 사용자 프로세스는 서버 프로세스와 직접 통신하지 않고 Dispatcher 프로세스를 거친다.

2

다음 중 Oracle이나 SQL Server 같은 데이터베이스의 저장 구조를 설명한 것으로 가장 부적절한 것은?

- ① 데이터를 읽고 쓰는 단위는 블록(=페이지)이다.
- ② 데이터파일에 공간을 할당하는 단위는 익스텐트다.
- ③ 같은 세그먼트(테이블, 인덱스)에 속한 익스텐트끼리는 데이터파일 내에서 서로 인접해 있다.
- ④ SQL Server에서는 한 익스텐트에 속한 페이지들을 여러 오브젝트가 나누어 사용할 수 있다.

3

데이터 변경 사항을 일단 데이터 버퍼 캐시에만 기록했다가 시간 간격을 두고 데이터 파일에 일괄 반영하려면 반드시 Redo(또는 Transaction) 로그의 도움이 필요하다. 그래야 DBMS에 문제가 발생하더라도 안전하게 복구할 수 있다. 이런 Redo 로그(또는 트랜잭션 로그) 메커니즘의 특징을 설명하는 여러 가지 용어가 있는데, 다음 중 아래와 관련된 것으로 가장 적절한 것은?

아 래

“버퍼 캐시 블록을 갱신하기 전에 변경사항을 먼저 로그 버퍼에 기록해야 하며, Dirty 버퍼를 디스크에 기록하기 전에 해당 로그 엔트리를 먼저 로그 파일에 기록해야 한다.”

- ① Write Ahead Logging
- ② Log Force at commit
- ③ Fast Commit
- ④ Delayed Block Cleanout

핵심정리

Response Time Analysis
 방법론은 Response
 Time을 정의하고, CPU
 Time과 Wait Time을 각각
 break down 하면서
 서버의 일량과 대기
 시간을 분석해 나간다.
 CPU Time은 파싱 작업에
 소비한 시간인지 아니면
 쿼리 본연의 오퍼레이션
 수행을 위해 소비한
 시간인지를 분석한다.
 Wait Time은 각각 발생한
 대기 이벤트들을 분석해
 가장 시간을 많이 소비한
 이벤트 중심으로
 해결방안을 모색한다.

4

다음 중 메모리 구조에 대한 설명으로 가장 부적절한

- ① DB 버퍼 캐시는 데이터 파일로부터 읽어 들인 데이터 블록을 담는 캐시 영역이다.
- ② /*+ append */ 힌트를 사용하면 Insert 시 DB 버퍼 캐시를 거치지 않고 디스크에서 직접 쓴다.
- ③ 클러스터링 팩터가 좋은 인덱스를 사용하면 Buffer Pinning 효과로 I/O를 줄일 수 있다.
- ④ LRU(Least Recently Used) 알고리즘에 따라, Table Full Scan 한 데이터 블록이 Index Range Scan 한 데이터 블록보다 DB 버퍼 캐쉬에 더 오래 머무른다.

5

다음 중 Response Time Analysis 성능관리 방법론을 설명할 때, 아래 괄호 안에 들어갈 가장 적절한 용어를 2개 고르시오.

아 래

Response Time = () Time + () Time

- ① CPU
- ② Elapsed
- ③ Idle
- ④ Queue

핵심정리

- 소프트 파싱
(Soft Parsing) :
SQL과 실행계획을
캐시에서 찾아 곧바로
실행단계로 넘어가는
경우를 말함
- 하드 파싱
(Hard Parsing) :
SQL과 실행계획을
캐시에서 찾지 못해
최적화 과정을 거치고
나서 실행단계로
넘어가는 경우를 말함

6

DB 인스턴스를 기동한 직후, 아래 SQL1)을 포함하는 프로그램을 사원 A와 B가 아래와 같이 각각 4회씩 연속적으로 수행하였다. SQL1)에 대한 Hard Parsing은 몇 번 발생하겠는가?

아래

SQL1)

```
select 고객명, 전화번호, 주소, 최종방문일시
from 고객
where 고객번호 = :custno
```

시점	사원 A	사원 B
t1	:custno에 1000 입력 후 조회	
t2		:custno에 2000 입력 후 조회
t3	:custno에 2000 입력 후 조회	
t4		:custno에 3000 입력 후 조회
t5	:custno에 3000 입력 후 조회	
t6		:custno에 3000 입력 후 조회
t7	:custno에 2000 입력 후 조회	
t8		:custno에 7000 입력 후 조회

- ① 1회 ② 4회
③ 6회 ④ 8회

7

SQL*Plus나 TOAD 같은 쿼리 툴로 오라클 데이터베이스 ORDER 계정에 접속해서 아래 SQL을 각각 한번씩 순차적으로 실행했다. 다음 중 SQL 파싱에 대한 설명으로 가장 적절한 것은?

아래

- (1) select 고객번호, 고객명, 휴대폰번호 from 고객 where
고객명 like '010%';
(2) SELECT 고객번호, 고객명, 휴대폰번호 FROM 고객 WHERE 고
객명 LIKE '010%';
(3) SELECT /* 고객조회 */ 고객번호, 고객명, 휴대폰번호 FROM
고객 WHERE 고객명 LIKE '010%';

- ① 첫 번째 수행될 때 각각 하드파싱을 일으키고, 다른 캐시 공간을 사용할 것이다.
② 1번과 3번 SQL은 하드파싱이 일어나지만, 2번은 하드파싱이 일어나지 않는다. 즉, 1번 SQL과 공유된다.
③ 1번과 2번 SQL은 하드파싱이 일어나지만, 3번은 하드파싱이 일어나지 않는다. 즉, 2번 SQL과 공유된다.
④ 실행계획이 서로 다를 수 있다.

핵심정리

바인드 변수
(Bind Variable) :
파라미터 Driven
방식으로 SQL을 작성하는
방법이 제공되는데 SQL과
실행계획을 여러 개
캐싱하지 않고 하나를
반복 재사용하므로 파싱
소요시간과 메모리
사용량을 줄여준다.

8

공통기술팀에서 개발표준 업무를 담당하는 고성능 씨는 개발팀에 SQL 작성을 위한 표준 가이드라인을 제시했다. OLTP 환경의 시스템인 점을 고려해 가급적 바인드 변수를 사용하도록 권고하지만, “Literal 상수 조건을 사용하는 것이 더 낫거나 바인드 변수를 사용하려고 애쓰지 않아도 되는 경우”를 보기와 같이 제시했다. 다음 중 가장 부적절한 것은?

- ① 수행빈도가 낮고 한 번 수행할 때 수십 초 이상 수행되는 SQL일 때
- ② 조건절 칼럼의 값 종류(Distinct Value)가 소수이고, 값 분포가 균일하지 않을 때
- ③ 사용자가 선택적으로 입력할 수 있는 조회 항목이 다양해서 조건절이 동적으로 바뀔 때
- ④ 사용자가 입력할 수 있는 조회 항목이 아니어서 해당 조건절이 불변일 때

9

다음 중 프로그램을 아래와 같이 작성할 때의 문제점과 가장 거리가 먼 것을 2개 고르시오.

아 래

```
create function get_count(p_table varchar2, p_column varchar2,
                          p_value varchar2)
return number
is
  l_sql long;
  l_count number;
begin
  l_sql := 'select count(*) from ' || p_table ;

  if p_column is not null and p_value is not null then
    l_sql := l_sql || ' where ' || p_column || ' = :1';
    execute immediate l_sql
      into l_count      -- 쿼리 결과 값을 저장한다.
      using p_value;    -- 바인드 변수(:1)에 값을 입력한다.
  else
    execute immediate l_sql into l_count;
  end if;

  return l_count;

end get_count;
```

- ① 불필요한 하드파싱을 많이 일으킨다.
- ② 테이블 통계정보를 활용하지 못한다.
- ③ 인덱스 전략 수립이 어렵다.
- ④ 실행계획을 제어하기 어렵다.

핵심정리

SQL 커서에 대한
작업 요청에 따른

데이터베이스

Call의 구분

- 1) Parse Call : SQL
파싱을 요청하는 Call
- 2) Execute Call :
SQL 실행을 요청하는
Call
- 3) Fetch Call :
SELECT문의 결과
데이터 전송을
요청하는 Call

10

다음 중 SQL 작성 방식에 대해 설명으로 가장 부적절한 것은?

- ① Static SQL이란, String형 변수에 담지 않고 코드 사이에 직접 기술한 SQL문을 말한다.
- ② Dynamic SQL이란, String형 변수에 담아서 실행하는 SQL문을 말한다.
- ③ Static SQL을 지원하는 개발환경에선 가급적 Static SQL로 작성하는 것이 바람직하다.
- ④ 루프(Loop) 내에서 반복적으로 수행되는 SQL에 Dynamic SQL을 사용하면, 공유 메모리에 캐싱된 SQL을 공유하지 못해 하드파싱이 반복적으로 일어난다.

11

다음 중 데이터베이스 Call에 대한 설명으로 가장 부적절한 것을 2개 고르시오.

- ① SELECT 문장을 수행할 땐 Execute, Parse, Fetch 순으로 Call이 발생한다.
- ② SELECT 문장에선 대부분 I/O가 Fetch Call 단계에서 일어난다.
- ③ Group By를 포함한 SELECT 문장에서 Group By 결과집합을 만들기까지의 I/O는 Execute Call 단계에서, 이후 결과집합을 전송할 때의 I/O는 Fetch Call 단계에서 일어난다.
- ④ INSERT, UPDATE, DELETE 문장에선 Fetch Call이 전혀 발생하지 않는다.

12

다음 중 SQL 트레이스에서 얻은 아래 Call Statistics를 통해 얻을 수 있는 정보와 가장 거리가 먼 것은?

아 래

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1000	0.18	0.14	0	0	0	0
Fetch	5000	1.21	4.25	0	200000	0	500000
total	6001	1.39	4.39	0	200000	0	500000

Misses in library cache during parse: 1

- ① SQL 파싱 부하가 최소화되도록 프로그램을 효과적으로 작성하였다.
- ② Order By, Group By 등 데이터 정렬이 필요한 연산을 포함하지 않는 SQL이다.
- ③ 실행할 때마다 평균적으로 200개 블록을 읽었고, 필요한 블록을 모두 버퍼캐시에서 찾았다.
- ④ 매번 실행할 때마다 비슷한 양의 결과집합을 반환했다면, Array (=Fetch) Size는 100 정도로 설정한 상태였을 것이다.

핵심정리

사용자 정의
함수/프로시저는
내장함수처럼 Native
코드로 완전 컴파일된
형태가 아니어서
가상머신(Virtual
Machine) 같은 별도의
실행엔진을 통해
실행된다.
실행될 때마다
컨텍스트 스위칭
(Context Switching)이
일어나며, 이 때문에
내장함수(Built-In)를
호출할 때와 비교해
성능을 상당히
떨어뜨린다.

13

다음 중 부분범위처리에 대한 설명으로 가장 부적절한 것은?

- ① 부분범위처리가 가능하도록 SQL을 작성하면 출력 대상 레코드가 많을수록 쿼리 응답 속도도 그만큼 빨라진다.
- ② 「INSERT INTO ... SELECT」 문장에서도 인덱스를 잘 활용하면 부분범위처리에 의한 성능 개선 효과를 얻을 수 있다.
- ③ Array 크기를 증가시키면 데이터베이스 Call 횟수가 감소한다.
- ④ Array 크기를 증가시키면 블록 I/O 횟수가 감소한다.

14

다음 중 사용자 정의 함수(User Defined Function)의 성능 특성에 대한 설명으로 가장 부적절한 것을 2개 고르시오.

- ① SQL을 포함하는 형태의 사용자 정의 함수라면, 대용량 쿼리에 그것을 사용하는 순간 성능이 크게 저하된다.
- ② SQL을 포함하지 않는 형태의 사용자 정의 함수라면, 대용량 쿼리에 그것을 사용해도 성능에 큰 영향은 없다.
- ③ 작은 코드 테이블로부터 코드명을 가져오는 정도의 사용자 정의 함수라면, 코드명을 가져오기 위해 매번 조인하는 것보다 오히려 성능상 유리하다.
- ④ 성능이 중요하다면, 함수 안에서 또다른 함수를 Recursive하게 호출하는 형태는 지양해야 한다.

15

다음 중 오라클에서 DB 저장형 함수(사용자 정의 함수)를 사용할 때 성능이 저하되는 원인과 거리가 가장 먼 것은?

- ① 함수를 실행할 때마다 컴파일하는 부하
- ② 가상머신(VM) 상에서 실행되므로 매번 바이트 코드를 해석하는 부하
- ③ 쿼리 문장의 조회 건수만큼 함수를 반복적으로 호출하는 부하
- ④ 함수에 내장된 쿼리가 있다면, 해당 문장을 Recursive하게 반복 수행하는 부하

핵심정리

Single Block I/O는 한번의 I/O Call에 하나의 데이터 블록만 읽어 메모리에 적재하는 방식이다.
MultiBlock I/O는 I/O Call이 필요한 시점에, 인접한 블록들을 같이 읽어 메모리에 적재하는 방식이다.

16

다음 중 아래 두 SQL의 수행 성능을 비교한 설명으로 가장 적절한 것은?

아래

```
가) select 고객명
    from 고객
   where 가입일자 = to_char(sysdate, 'yyyymmdd')
   order by 고객명;
나) select *
    from 고객
   where 가입일자 = to_char(sysdate, 'yyyymmdd')
   order by 고객명;
```

- ① 클라이언트에게 데이터를 전송할 때 발생하는 네트워크 트래픽은 두 SQL이 똑같다.
- ② 가입일자만으로 구성된 단일 컬럼 인덱스를 사용한다면, 두 SQL의 소트 공간 사용량은 똑같다.
- ③ 가입일자만으로 구성된 단일 컬럼 인덱스를 사용한다면 '나'보다 '가' SQL에 블록 I/O가 더 많이 발생한다.
- ④ (가입일자 + 고객명)을 선두로 갖는 인덱스를 사용한다면 '가'보다 '나' SQL에 블록 I/O가 더 많이 발생한다.

17

다음 중 I/O 효율화 튜닝 방안으로 가장 부적절한 것은?

- ① 필요한 최소 블록만 읽도록 쿼리를 작성한다.
- ② 전략적인 인덱스 구성은 물론 DBMS가 제공하는 다양한 기능을 활용한다.
- ③ 옵티마이저 행동에 영향을 미치는 가장 중요한 요소는 통계정보이므로 변경이 거의 없는 테이블일지라도 통계정보를 매일 수집해 준다.
- ④ 필요하다면, 옵티마이저 힌트를 사용해 최적의 액세스 경로로 유도한다.

18

다음 중 블록 I/O에 대한 설명으로 가장 부적절한 것은?

- ① Random I/O는 인덱스를 통해 테이블을 액세스할 때 주로 발생한다.
- ② Direct Path I/O는 병렬로 인덱스를 통해 테이블을 액세스할 때 주로 발생한다.
- ③ Single Block I/O는 인덱스를 통해 테이블을 액세스할 때 주로 발생한다.
- ④ Multiblock I/O는 인덱스를 이용하지 않고 테이블 전체를 스캔할 때 주로 발생한다.

핵심정리

버퍼 캐시 히트율
(Buffer Cache Hit Ratio) :
버퍼 캐시 효율을
측정하는 지표로서, 전체
읽은 블록 중에서 메모리
버퍼 캐시에서 찾은
비율을 나타낸다.
물리적인 디스크 읽기를
수반하지 않고 곧바로
메모리에서 블록을 찾은
비율을 말한다.

I/O튜닝의 핵심 원리

- Sequential 액세스에
의한 선택 비중을
높인다.
- Random 액세스
발생량을 줄인다.

19

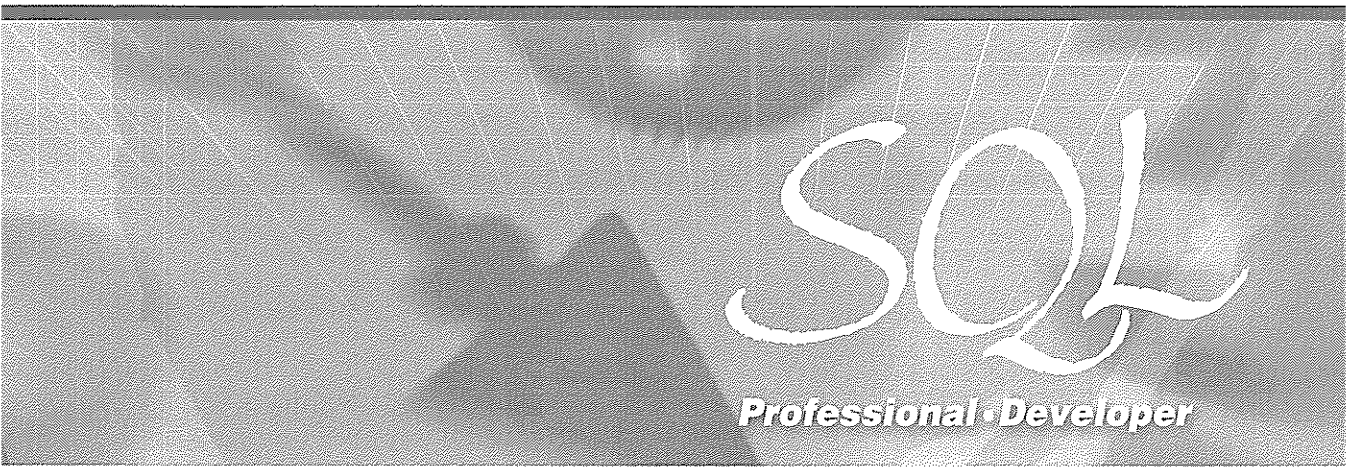
다음 중 데이터베이스 I/O 원리를 설명한 것으로 가장 부적절한 것은?

- ① 한 쿼리 내에서 같은 블록을 반복적으로 액세스하면 버퍼 캐시 히트율 (BCHR)은 높아진다.
- ② Multiblock I/O는 한번의 I/O Call로 여러 데이터 블록을 읽어 메모리에 적재하는 방식이다.
- ③ 테이블을 Full Scan할 때, 테이블이 작은 Extent로 구성되어 있을수록 더 많은 I/O Call이 발생한다.
- ④ 인덱스를 통해 테이블을 액세스할 때, 테이블이 큰 Extent로 구성돼 있으면 더 적은 I/O Call이 발생한다.

20

다음 중 데이터베이스 I/O 원리에 대한 설명으로 가장 부적절한 것은?

- ① 단 하나의 레코드를 읽더라도 해당 레코드가 속한 블록을 통째로 읽는다.
- ② I/O를 수행할 때 익스텐트 내에 인접한 블록을 같이 읽어들이는 것을 'Multiblock I/O'라고 한다.
- ③ 테이블 블록을 스캔(Scan)할 때는 Sequential I/O 방식을, 인덱스 블록을 스캔(Scan)할 때는 Random I/O 방식을 사용한다.
- ④ MPP(Massively Parallel Processing) 방식의 데이터베이스 제품에선 각 프로세스가 독립적인 메모리 공간을 사용하며, 데이터를 저장할 때도 각각의 디스크를 사용한다. 읽을 때도 동시에 각각의 디스크를 액세스하기 때문에 병렬 I/O 효과가 극대화된다.

The background of the top banner is a grayscale photograph of a city skyline, possibly featuring a prominent pyramid-like structure. A white grid pattern is overlaid on the image, creating a technical or architectural feel.

SQL

Professional • Developer

제2장

Lock과 트랜잭션 동시성제어

제1절 Lock

제2절 트랜잭션

제3절 동시성 제어

핵심정리

Lock에 의한 성능 저하를 최소화하는 방안

- ① 트랜잭션의 원자성을 훼손하지 않는 선에서 트랜잭션을 가능한 짧게 정의할 것
- ② 같은 데이터를 갱신하는 트랜잭션이 동시에 수행되지 않도록 설계할 것
- ③ 주간에 대용량 갱신 작업이 불가피하다면, 블로킹 현상에 의해 사용자가 무한정 기다리지 않도록 적절한 프로그래밍 기법을 도입할 것
- ④ 트랜잭션 격리성 수준을 불필요하게 상향 조정하지 않을 것
- ⑤ 트랜잭션을 잘 설계하고 대기 현상을 피하는 프로그래밍 기법을 적용하기에 앞서, SQL 문장이 가장 빠른 시간 내에 처리를 완료할 것

블로킹(Blocking)은, Lock 경합이 발생해 특정 세션이 작업을 진행하지 못하고 멈춰 선 상태를 말한다. 공유 Lock끼리는 호환되기 때문에 블로킹이 발생하지 않는다. 공유 Lock과 배타적 Lock은 호환되지 않아 블로킹이 발생할 수 있다.

21

공통기술팀에서 개발표준 업무를 담당하는 고성능 씨가 Lock 경합에 의한 성능 저하를 최소화하기 위해 개발팀에 제시한 가이드라인으로 가장 부적절한 것은?

- ① 트랜잭션의 원자성을 훼손하지 않는 선에서 트랜잭션을 가능한 짧게 정의할 것
- ② 같은 데이터를 갱신하는 프로그램이 가급적 동시에 수행되지 않도록 트랜잭션을 설계할 것
- ③ select 문장에 for update 문장을 사용하지 말 것
- ④ 온라인 트랜잭션을 처리하는 DML 문장을 1순위로 튜닝함으로써 조건 절에 맞는 최적의 인덱스를 제공할 것

22

MS-SQL Server에서 아래 UPDATE문과 블로킹 없이 동시 수행이 가능한 SQL문을 보기에서 고르시오. (단, Transaction Isolation Level을 조정하지 않았고, Snapshot 관련 데이터베이스 설정도 초기 값 그대로인 상황이다.)

아 래

EMP 테이블

EMPNO (PK)	ENAME	DEPTNO	SAL
7788	김철수	10	3000
7900	이정훈	20	5000
7903	정명훈	20	2000
8012	홍길동	30	6000

update emp set sal = sal * 1.1 where deptno = 20

- ① select * from emp where empno = 7900
- ② update emp set sal = sal * 1.1 where empno = 7900
- ③ delete from emp where empno = 7903
- ④ insert into emp(empno, ename, deptno, sal) values(8014, '이정훈', 40, 4000)

핵심정리

SQL Server의 공유 Lock은 트랜잭션이나 쿼리 수행이 완료될 때까지 유지되는 것이 아니라 다음 레코드가 읽히면 곧바로 해제된다. 단, 기본 트랜잭션 격리성 수준(Read Committed)에서만 그렇다. 격리성 수준을 변경하지 않고도 트랜잭션 내에서 공유 Lock이 유지되도록 하려면 테이블 힌트로 holdlock을 지정하면 된다. 또한, 두 트랜잭션은 상대편 트랜잭션에 의한 공유 Lock이 해제되기만을 기다리는 교착상태를 방지하려고 SQL Server는 갱신(Update) Lock을 두게 되었고, 이 기능을 사용하려면 uplock 힌트를 지정하면 된다.

테이블 Lock 종류로는 아래 5가지가 있다.

- Row Share(RS)
- Row Exclusive(RX)
- Share(S)
- Share Row Exclusive(SRX)
- Exclusive(X)

23

다음 중 오라클 PL/SQL로 작성한 아래 프로그램을 MS-SQL Server T-SQL 구문으로 변환하고자 한다. 아래 FOR UPDATE 구문을 대신하기 위해 사용할 SQL Server 힌트로 가장 적절한 것을 2개 고르시오.

아 래

```
select 적립포인트, 구매실적, 방문횟수, 최근방문일자
into v_적립포인트, v_구매실적, v_방문횟수, v_최근방문일자
from 고객
where 고객번호 = 100
FOR UPDATE;

-- do anything ...

update 고객 set 적립포인트 = v_적립포인트 where 고객번호 = 100;
commit;
```

- ① from 고객 WITH (HOLDLOCK)
- ② from 고객 WITH (UPDLOCK)
- ③ from 고객 WITH (READPAST)
- ④ from 고객 WITH (TABLOCK)

24

다음 중 아래와 같은 락(Lock) 모니터링 결과가 발생할 수 있는 SQL로 가장 적절한 것은?

아 래

```
SELECT a.SESSION_ID, a.LOCK_TYPE, a.MODE_HELD,
       a.MODE_REQUESTED, a.LOCK_ID1, a.LOCK_ID2
FROM   DBA_LOCK a, V$SESSION b
WHERE  a.SESSION_ID = b.SID
AND    a.LOCK_TYPE IN ( 'Transaction', 'DML')
AND    b.USERNAME = USER;
```

SESSION_ID	LOCK_TYPE	MODE_HELD	MODE_REQUESTED	LOCK_ID1	LOCK_ID2
135	Transaction	Exclusive	None	131074	3098
135	DML	Exclusive	None	92614	0

- ① INSERT INTO EMP (EMPNO, ENAME) VALUES (1000, 'SQLP');
- ② UPDATE EMP SET SAL = 1000 WHERE EMPNO = 7369;
- ③ INSERT /*+ APPEND */ INTO EMP SELECT * FROM SCOTT.EMP;
- ④ SELECT * FROM EMP WHERE EMPNO = 7369 FOR UPDATE;

25

아래는 두 세션에서 각각 UPDATE 문을 수행한 후의 오라클 Lock 관련 조회 결과이다. 다음 중 ☐ , ☐ 에 들어갈 용어로 가장 적절한 것은?

아 래

[SID: 31]

UPDATE 상품 a SET 상품명 = '상품1' WHERE a.상품코드 = 'A0001';

[SID: 267]

UPDATE 상품 a SET 상품명 = '상품2' WHERE a.상품코드 = 'A0001';

[오라클 Lock 관련 조회 결과]

SID	TP NAME	HELD	REQUESTED	BLOCKING	BLOCKED OBJ/XID	STATUS
31	TM DML	<input type="checkbox"/>	None		상품	INACTIVE
31	TX Transaction	<input type="checkbox"/>	None	1,267	21,3349	INACTIVE
267	TM DML	<input type="checkbox"/>	None		상품	ACTIVE
267	TX Transaction	None	<input type="checkbox"/>	1,31	3,21,3349	ACTIVE

- ① ☐ Row-X (SX), ☐ S/Row-X (SSX)
 ② ☐ Row-X (SX), ☐ Exclusive
 ③ ☐ Row-S (SS), ☐ S/Row-X (SSX)
 ④ ☐ Row-S (SS), ☐ Exclusive

26

다음 중 아래는 트랜잭션의 특징을 연결한 것으로 가장 적절한 것은?

아 래

가. 더 이상 분해가 불가능한 업무의 최소단위를 말한다.

나. 트랜잭션이 그 실행을 성공적으로 완료하면 언제나 일관성 있는 데이터베이스 상태로 변환한다. 즉, 트랜잭션 실행의 결과로 데이터베이스 상태가 모순되지 않는다.

다. 트랜잭션이 실행 중에 생성하는 연산의 중간 결과는 다른 트랜잭션이 접근할 수 없다.

라. 트랜잭션이 일단 그 실행을 성공적으로 완료하면 그 결과는 데이터베이스에 영속적으로 저장된다.

- ① 원자성 - 가. , 격리성 - 나.
 ② 일관성 - 가. , 격리성 - 다.
 ③ 영속성 - 나. , 일관성 - 다.
 ④ 영속성 - 라. , 일관성 - 나.

핵심정리

낮은 단계의 격리성 수준에서 발생할 수 있는 현상들

1) Dirty Read:

다른 트랜잭션에 의해 수정됐지만 아직 커밋되지 않은 데이터를 읽는 것을 말한다.

2) Non-Repeatable Read :

한 트랜잭션 내에서 같은 쿼리를 두 번 수행했는데, 그 사이에 다른 트랜잭션이 값을 수정 또는 삭제하는 바람에 두 쿼리 결과가 다르게 나타나는 현상을 말한다.

3) Phantom Read :

한 트랜잭션 내에서 같은 쿼리를 두 번 수행했는데, 첫 번째 쿼리에서 없던 유령(Phantom) 레코드가 두 번째 쿼리에서 나타나는 현상을 말한다.

27

아래와 같이 100번 세션(SID=100)과 200번 세션(SID=200)에서 쿼리를 순차적으로 실행했을 때 (가), (나)의 결과로 가장 적절한 것은?

아 래

[테이블]

```
CREATE TABLE t1
AS
SELECT    LEVEL AS c1
FROM DUAL
CONNECT BY LEVEL <= 10;
```

[실행]

세션 : 100	세션 : 200
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	
INSERT t1 VALUES (11);	
UPDATE t1 SET c1 = 1 WHERE c1 >= 6;	
COMMIT;	
SELECT COUNT (*) FROM t1 WHERE c1 >= 6; -- (가)	
INSERT INTO t1 VALUES (12);	
COMMIT;	
SELECT COUNT (*) FROM t1 WHERE c1 >= 6; -- (나)	

- ① (가) 0, (나) 5
- ② (가) 0, (나) 6
- ③ (가) 5, (나) 0
- ④ (가) 5, (나) 1

28

다음 중 대부분 DBMS가 채택하고 있는 기본 트랜잭션 격리성 수준 (Transaction Isolation Level)인 것은?

- ① Read Uncommitted
- ② Read Committed
- ③ Repeatable Read
- ④ Serializable

핵심정리

트랜잭션 격리성 수준

- Read Uncommitted :
트랜잭션에서 처리 중인 아직 커밋되지 않은 데이터를 다른 트랜잭션이 읽는 것을 허용한다.
- Read Committed :
트랜잭션이 커밋되어 확정된 데이터만 다른 트랜잭션이 읽도록 허용함으로써 Dirty Read를 방지해준다.
- Repeatable Read :
트랜잭션 내에서 쿼리를 두 번 이상 수행할 때, 첫 번째 쿼리에 있던 레코드가 사라지거나 값이 바뀌는 현상을 방지해 준다.
- Serializable Read :
트랜잭션 내에서 쿼리를 두 번 이상 수행할 때, 첫 번째 쿼리에 있던 레코드가 사라지거나 값이 바뀌지 않음은 물론 새로운 레코드가 나타나지도 않는다.

29

다음 중 트랜잭션 동시성 제어에 대한 설명으로 가장 부적절한 것은?

- ① 비관적 동시성 제어(Pessimistic Concurrency Control)는 두 트랜잭션이 같은 데이터를 동시에 수정할 것이라고 가정하고 데이터를 읽는 시점에 Lock을 설정하는 방식을 말한다.
- ② 낙관적 동시성 제어(Optimistic Concurrency Control)는 두 트랜잭션이 같은 데이터를 동시에 수정하지 않을 것이라고 가정하고 데이터를 읽는 시점에 Lock을 설정하지 않는 방식을 말한다. 이 방식에선 데이터를 수정하는 시점에, 앞서 읽은 데이터가 다른 트랜잭션에 의해 변경되었는지 반드시 확인해야 데이터 정합성을 유지할 수 있다.
- ③ 트랜잭션 격리성 수준(Transaction Isolation Level)을 Serializable로 상향 조정하면 일반적으로 동시성과 일관성이 같이 높아진다.
- ④ 트랜잭션 격리성 수준(Transaction Isolation Level)을 Serializable로 상향 조정하면 프로그램에서 별도의 동시성 제어를 하지 않아도 DBMS가 트랜잭션 단위 일관성을 보장해 준다.

30

EMP 테이블 7788번 사원의 SAL 값이 현재 1,000인 상황에서 아래 TX1, TX2 두 개의 트랜잭션이 동시에 수행되었다. 양쪽 트랜잭션이 모두 완료된 시점에 7788번 사원의 SAL 값은 Oracle과 SQL Server에서 각각 얼마인지 작성하시오. (Oracle과 SQL Server 순으로 적으시오. 단, Transaction Isolation Level을 조정하지 않았고, Snapshot 관련 데이터베이스 설정도 초기값 그대로인 상황이다.)

아 래

< TX1 >		< TX2 >	
update emp set sal = 2000 where empno = 7788 and sal = 1000 ; commit;	t1		
	t2	update emp set sal = 3000 where empno = 7788 and sal = 2000 ;	
	t3		
	t4	commit;	

핵심정리


다중버전 동시성 제어
(Multiversion
Concurrency Control,
이하 MVCC)

- 데이터를 변경할
때마다 그 변경사항을
Undo 영역에 저장해
둔다.
- 데이터를 읽다가
쿼리(또는 트랜잭션)
시작 시점 이후에
변경된(변경이 진행
중이거나 이미
커밋된) 값을
발견하면, Undo
영역에 저장된 정보를
이용해 쿼리(또는
트랜잭션)
시작 시점의 일관성
있는 버전(CR
Copy)을 생성하고
그것을 읽는다..

31

다중버전 동시성 제어(Multiversion Concurrency Control, 이하 MVCC) 모델을 채택하는 DBMS가 늘고 있다. 다음 중 DBMS의 읽기 일관성 메커니즘을 설명한 것으로 가장 부적절한 것은?

- ① 읽기 일관성을 위해 Undo 세그먼트(또는 버전 저장소)에 저장된 Undo (또는 Snapshot) 데이터를 활용한다.
- ② MVCC 모델은 기본적으로 완벽한 문장 수준 읽기 일관성을 보장한다.
- ③ MVCC 모델은 기본적으로 완벽한 트랜잭션 수준 읽기 일관성을 보장한다.
- ④ 트랜잭션 수준 읽기 일관성이란, 트랜잭션이 시작된 시점을 기준으로 일관성 있게 데이터를 읽어들이는 것을 말한다.



SQL

Professional • Developer

옵티마이저 원리

제1절 옵티마이저

제2절 쿼리변환

핵심정리

규칙기반 옵티마이저 (Rule-Based Optimizer, 이하 RBO)는 다른 말로 '휴리스틱(Heuristic) 옵티마이저'라고 불리며, 미리 정해 놓은 규칙에 따라 액세스 경로를 평가하고 실행계획을 선택한다. 여기서 규칙이란 액세스 경로별 우선순위로써, 인덱스 구조, 연산자, 조건절 형태가 순위를 결정짓는 주요인이다.

32

CBO(비용기반 옵티마이저)는 쿼리 최적화 과정에 비용(Cost)를 계산한다. 다음 중 비용(Cost)과 가장 거리가 먼 것은?

- ① 비용이란 기본적으로, SQL 수행 과정에 수반될 것으로 예상되는 I/O 일량을 계산한 것이다.
- ② 데이터베이스 Call 발생량도 옵티마이저의 중요한 비용 요소다.
- ③ 옵티마이저가 비용을 계산할 때, CPU 속도, 디스크 I/O 속도 등도 고려할 수 있다.
- ④ 최신 옵티마이저는 I/O에 CPU 연산 비용을 더해서 비용을 계산한다.

33

다음 중 규칙 기반 옵티마이저(RBO)가 사용하는 규칙으로 가장 부적절한 것은?

- ① 고객유형코드에 인덱스가 있으면, 아래 SQL에 인덱스를 사용한다.
`select * from 고객 where 고객유형코드 = 'CC0123'`
- ② 고객명에 인덱스가 있으면, 아래 SQL에 인덱스를 사용해 order by 소트 연산을 대체한다.
`select * from 고객 order by 고객명`
- ③ 연령과 연봉에 인덱스가 하나씩 있으면, 아래 SQL에 연봉 인덱스를 사용한다. between 조건(닫힌 조건)이 부등호 조건(열린 조건)보다 아무래도 스캔 범위가 작을 가능성이 높기 때문이다.
`select * from 사원 where 연령 >= 60 and 연봉 between 3000 and 6000`
- ④ 직급에 인덱스가 있고, 직급의 종류 개수가 5개 이상이면 인덱스를 사용한다.
`select * from 사원 where 직급 = '대리'`

핵심정리

- 선택도 → 카디널리티
→ 비용 → 액세스
방식, 조인 순서, 조인
방법 등 결정
- 카디널리티
= 총 로우 수 × 선택도
= num_rows /
num_distinct

34

다음 중 전체범위 최적화(ALL_ROWS) 방식의 옵티마이저 모드에 대한 설명으로 가장 거리가 먼 것은?

- ① 쿼리의 최종 결과 집합을 끝까지 Fetch하는 것을 전제로, 시스템 리소스를 가장 적게 사용하는 실행계획을 선택한다.
- ② 부분범위 최적화(FIRST_ROWS)와 비교할 때, Index Scan보다 Table Full Scan하는 실행계획을 더 많이 생성한다.
- ③ DML 문장은 옵티마이저 모드와 상관없이 항상 전체범위 최적화 방식으로 최적화된다.
- ④ 가장 빠른 응답속도(Response Time)를 목표로 한다.

35

다음 중 비용기반 옵티마이저(Cost-Based Optimizer)가 사용하는 비용 계산식으로 가장 적절한 것을 2개 고르시오.
(※ NDV = Number Of Distinct Value)

- ① 선택도(Selectivity) = $1 / NDV$
- ② 선택도(Selectivity) = $NDV / \text{총 로우 수}$
- ③ 카디널리티(Cardinality) = NDV
- ④ 카디널리티(Cardinality) = $\text{총 로우 수} / NDV$

36

다음 중 통계정보 수집 시 고려사항을 설명한 것으로 가장 부적절한 것은?

- ① 시간/주기 : 부하가 없는 시간대에 가능한 한 빠르게 수집을 완료해야 함
- ② 표본(Sample) 크기 : 가능한 한 많은 양의 데이터를 읽도록 해야 함
- ③ 정확성 : 표본(Sample) 검사하더라도 전수 검사할 때의 통계치에 근접해야 함
- ④ 안정성 : 데이터에 큰 변화가 없는데도 매번 통계치가 바뀌지 않아야 함

다음 중 동일 결과를 반환하는 SQL에서 가장 효율적인 것은? (단, ItemAmtFunc는 저장형 함수이고, 상품 테이블의 PK는 상품코드임)

- ① SELECT 상품코드,
 ItemAmtFunc(상품코드, SYSDATE) 당일매출,
 ItemAmtFunc(상품코드, SYSDATE-1) 전일매출,
 (ItemAmtFunc(상품코드, SYSDATE) -
 ItemAmtFunc(상품코드, SYSDATE-1))*100 /
 ItemAmtFunc(상품코드, SYSDATE) 증감율
 FROM 상품
 WHERE 상품분류코드='110'
- ② SELECT 상품코드, 당일매출, 전일매출,
 (당일매출-전일매출) * 100/당일매출 증감율
 FROM (SELECT ROWNUM, 상품코드,
 ItemAmtFunc(상품코드, SYSDATE) 당일매출,
 ItemAmtFunc(상품코드, SYSDATE-1) 전일매출,
 FROM 상품
 WHERE 상품분류코드='110')
- ③ SELECT 상품코드, 당일매출, 전일매출,
 (당일매출-전일매출) * 100 / 당일매출 증감율
 FROM (SELECT 상품코드,
 ItemAmtFunc(상품코드, SYSDATE) 당일매출,
 ItemAmtFunc(상품코드, SYSDATE-1) 전일매출,
 FROM 상품
 WHERE 상품분류코드='110')
- ④ SELECT 상품코드,
 (SELECT ItemAmtFunc(상품코드, SYSDATE)
 FROM DUAL) 당일매출,
 (SELECT ItemAmtFunc(상품코드, SYSDATE-1)
 FROM DUAL) 전일매출,
 (SELECT (ItemAmtFunc(상품코드, SYSDATE) -
 ItemAmtFunc(상품코드, SYSDATE-1))*100/
 ItemAmtFunc(상품코드, SYSDATE)
 FROM DUAL) 증감율
 FROM 상품
 WHERE 상품분류코드 = '110'

핵심정리

힌트를 사용하지 않고
뷰 머징(View Merging)
방지하는 방법

- 집합(set) 연산자
(union, union all,
intersect, minus)
- connect by절
- ROWNUM pseudo
칼럼
- select-list에 집계
함수(avg, count,
max, min, sum) 사용
- 분석 함수(Analytic
Function)

38

Oracle에서 no_merge 힌트를 사용하지 않고도 아래 SQL문에 뷰 머징 (View Merging)이 발생하지 않게 하려고 한다. 다음 중 ㉠ 안에 들어갈 키워드로 가장 적절한 것은?

아 래

```
SELECT *
FROM (SELECT ㉠ , EMP_NAME, HIRE_DATE, SAL,
            DEPT_NO
      FROM EMP
      WHERE DEPTNO = 30) X
WHERE HIRE_DATE BETWEEN TO_DATE('20100101','YYYYMMDD') AND
                        TO_DATE('20101231','YYYYMMDD')
```

- ① TO_DATE(SYSDATE,'YYYYMMDD')
- ② ROWNUM
- ③ EMPNO
- ④ ROWID

39

다음 중 뷰 머징(View Merging)을 불가능하게 하는 경우가 아닌 것은?

- ① 뷰 안에 ROWNUM을 사용한 경우
- ② 뷰 안에 Group By를 사용한 경우
- ③ 뷰 안에 윈도우 함수(Window Function)를 사용한 경우
- ④ 뷰 안에 UNION 연산자를 사용한 경우

40

다음 중 아래 SQL에 대한 설명으로 가장 적절한 것은? (단, 쿼리변환(Query Transformation)이 동작하는 것으로 가정함)

아 래

```
select *
from (select deptno, empno, ename, job, sal, sal * 1.1 sal2, hiredate
      from emp
      where job = 'CLERK'
      union all
      select deptno, empno, ename, job, sal, sal * 1.2 sal2, hiredate
      from emp
      where job = 'SALESMAN' ) v
where v.deptno = 30
```

- ① emp 테이블에 job + deptno로 구성된 인덱스를 만들면, job 조건만 인덱스 액세스 조건으로 사용되고, deptno 조건은 필터로 처리된다.
- ② emp 테이블 job에 단일 컬럼 인덱스를 만들면, 이 인덱스를 정상적으로 사용할 수 없다. 즉, Index Range Scan할 수 없다.
- ③ emp 테이블에 deptno + job으로 구성된 인덱스를 만들면, job과 deptno에 대한 조건 모두를 인덱스 액세스 조건으로 사용할 수 있다.
- ④ emp 테이블 deptno에 단일 컬럼 인덱스를 만들면, 이 인덱스를 정상적으로 사용할 수 없다. 즉, Index Range Scan할 수 없다.

41

다음 중 아래 SQL을 처리하는 데 있어 옵티마이저가 선택할 수 있는 옵션으로 가장 부적절한 것은?

아 래

```
alter table 과금 add constraint 과금_pk primary key(고객번호, 과금액);
alter table 수납 add constraint 수납_pk primary key(고객번호, 수납일자);

select e1.고객번호, e1.과금액, e2.수납액, e1.과금액-e2.수납액 미수금액
from (select 고객번호, sum(과금액) 과금액 from 과금 group by 고객번호)
e1
, (select 고객번호, sum(수납액) 수납액
   from 수납
   where 고객번호 = 10
   group by 고객번호) e2
where e1.고객번호 = e2.고객번호
and e2.수납액 > 0
```

- ① 두 인라인 뷰(Inline View)를 풀어(View Merging) 고객번호 조인을 먼저 처리한 후에, 고객번호로 group by하면서 과금액과 수납액을 구한다.
- ② 인라인 뷰 e2에 있는 「고객번호 = 10」 조건을 e1에 전달해 줌으로써 과금_pk 인덱스를 사용해 처리한다.
- ③ 메인 쿼리에 있는 「수납액 > 0」 조건을 인라인 뷰 e2에 제공함으로써 조인 연산 전에 필터링이 일어나도록 한다.
- ④ 사용자가 작성한 SQL 형태 그대로, 각 인라인 뷰를 따로 최적화한 후에 조인한다.

42

아래 SQL과 트레이스 결과를 보고, 최적의 튜닝 방안을 선택하시오.

아 래

주문
 주문번호
 고객번호 (FK)
 주문일시
 주문금액

→

고객
 고객번호
 고객명
 연락처
 주소

```

select o,주문번호, o,주문일시, o,고객번호, o,주문금액
from 주문 o inner join 고객 c on c,고객번호 = o,고객번호
where o,주문일시 = :ord_dt
    
```

Call	Count	CPU Time	Elapsed Time	Disk	Query	Current	Rows
Parse	1	0.000	0.028	0	0	0	0
Execute	1	0.000	0.000	0	0	0	0
Fetch	45939	9.234	12.642	13563	149480	0	459379
Total	45941	9.234	12.670	13563	149480	0	459379

Rows	Row Source Operation
459379	NESTED LOOPS (cr=149480 pr=13563 pw=0 time=19337719 us)
459379	TABLE ACCESS BY INDEX ROWID 주문 (cr=103541 pr=13562 pw=0 time=8766716 us)
459379	INDEX RANGE SCAN 주문_주문일시_IDX (cr=46902 pr=968 pw=0 time=1879909 us)
459379	INDEX UNIQUE SCAN 고객_PK (cr=45939 pr=1 pw=0 time=5375998 us)

① 힌트를 이용해 고객 테이블을 먼저 드라이빙하도록 유도한다.

```

select /*+leading(c)use_nl(o)*/ o,주문번호, o,주문일시, o,고객번호,
o,주문금액
    
```

```

from 주문 o inner join 고객 c on c,고객번호 = o,고객번호
where o,주문일시 = :ord_dt
    
```

② 힌트를 이용해 해시 조인으로 유도한다.

```

select /*+use_hash(oc)*/ o,주문번호, o,주문일시, o,고객번호,
o,주문금액
    
```

```

from 주문 o inner join 고객 c on c,고객번호 = o,고객번호
where o,주문일시 = :ord_dt
    
```

③ 고객과의 조인문을 Exists 서브쿼리로 변환한다.

```

select o,주문번호, o,주문일시, o,고객번호, o,주문금액
from 주문 o
where o,주문일시 = :ord_dt
and exists (select 'x' from 고객 where 고객번호 = o,고객번호)
    
```

④ 고객과의 조인을 제거한다.

```

select 주문번호, 주문일시, 고객번호, 주문금액
from 주문
where 주문일시 = :ord_dt
    
```

SQL

Professional • Developer

인덱스와 조인

제1절 인덱스 기본 원리

제2절 인덱스 튜닝

제3절 조인 기본 원리

제4절 고급 조인 기법

핵심정리

Index Range Scan은
인덱스 루트 블록에서
리프 블록까지 수직적으로
탐색한 후에 리프 블록을
필요한 범위(Range)만
스캔하는 방식이다.

43

인덱스 탐색 과정은 수직적 탐색과 수평적 탐색으로 나뉘어 설명할 수 있다. 수평적 탐색은 인덱스 리프 블록에 저장된 레코드끼리 연결된 순서에 따라 좌에서 우, 또는 우에서 좌로 스캔하기 때문에 '수평적'이라고 표현한다. 수직적 탐색은 수평적 탐색을 위한 시작 지점을 찾는 과정이라고 말할 수 있다. 다음 중 아래 SQL에서 EMP_IDX 인덱스의 수평적 탐색 시작 지점의 값을 보기에서 고르시오.

아 래

```
< 인덱스 구성 >
emp_idx : deptno + sal + comm

select /*+ index_asc(e emp_idx) */
      empno, ename, sal, comm, hiredate
from   emp e
where  deptno = 20
and    sal between 2000 and 3000
and    comm <= 100
```

- ① 인덱스 정렬 순서 상 deptno = 20 조건을 만족하는 첫 번째 레코드
- ② 인덱스 정렬 순서 상 deptno = 20 and sal = 2000 조건을 만족하는 첫 번째 레코드
- ③ 인덱스 정렬 순서 상 deptno = 20 and sal = 2000 and comm = 100 조건을 만족하는 첫 번째 레코드
- ④ 인덱스 정렬 순서 상 deptno = 20 and sal = 3000 and comm = 100 조건을 만족하는 첫 번째 레코드

44

다음 중 아래와 같은 인덱스 상황에서 Index Range Scan이 불가능한 SQL은?

아 래

```
create index emp_idx on emp( deptno, job, ename );
```

- ① select empno, ename, job, hiredate
 from emp
 where deptno = :deptno
 and ename = :ename;
- ② select empno, ename, job, hiredate
 from emp
 where deptno = :deptno
 and job is null;
- ③ select empno, ename, job, hiredate
 from emp
 where deptno is null
 and job = :job;
- ④ select empno, ename, job, hiredate
 from emp
 where job = :job
 and ename = :ename;

핵심정리

Index Skip Scan은 루트 또는 브랜치 블록에서 읽은 칼럼 값 정보를 이용해 조건에 부합하는 레코드를 포함할 “가능성이 있는” 하위 블록(브랜치 또는 리프 블록)만 골라서 액세스하는 방식이다.

Index Full Scan은 수직적 탐색없이 인덱스 리프 블록을 처음부터 끝까지 수평적으로 탐색하는 방식으로, 대개는 데이터 검색을 위한 최적의 인덱스가 없을 때 차선으로 선택된다.

Index Unique Scan은 수직적 탐색만으로 데이터를 찾는 스캔 방식으로, Unique 인덱스를 ‘=’ 조건으로 탐색하는 경우에 작동한다.

Index Fast Full Scan은 Index Full Scan보다 빠르다. Index Fast Full Scan이 Index Full Scan보다 빠른 이유는, 인덱스 트리 구조를 무시하고 인덱스 세그먼트 전체를 Multiblock Read 방식으로 스캔하기 때문이다.

45

다음 중 힌트를 사용하더라도 Index Skip Scan 방식으로 실행되지 않는 SQL인 것은?

아 래

```
create index 고객_x01 on 고객(거주지역, 성별, 가입일자);
create index 고객_x02 on 고객(고객등급, 생일);
```

- ① select * from 고객
where 거주지역 = '충청'
and 가입일자 between '20110201' and '20110228'
- ② select * from 고객
where 가입일자 between '20110201' and '20110228'
- ③ select * from 고객
where 고객등급 between 'A' and 'C'
and 생일 = '0326'
- ④ select * from 고객
where 고객등급 in ('A', 'B', 'C')
and 생일 = '0326'

46

다음 중 인덱스 구성이 아래와 같을 때 SQL을 처리하는 데 활용 가능한 인덱스 스캔 방식을 2개 고르시오.

아 래

```
create index emp_x01 on emp(deptno, sal);

select empno, ename, deptno, sal
from emp
where sal between 2000 and 4000;
```

- ① index range scan
- ② index full scan
- ③ index fast full scan
- ④ index skip scan

핵심정리

비트맵 인덱스는 Lock에 의한 DML 부하가 심한 것이 단점이다. 레코드 하나만 변경되더라도 해당 비트맵 범위에 속한 모든 레코드에 Lock이 걸린다. OLTP성 환경에 비트맵 인덱스를 쓸 수 없는 이유가 여기에 있다. 비트맵 인덱스는 읽기 위주의 대용량 DW(특히, OLAP) 환경에 아주 적합하다.

47

다음 중 비트맵 인덱스에 대한 설명으로 가장 부적절한 것은?

- ① B*Tree 인덱스에 비해 테이블 Random 액세스를 획기적으로 줄여주므로 성별처럼 Distinct Value 개수가 적은 컬럼에도 좋은 성능을 나타낸다.
- ② 고객번호처럼 Distinct Value 개수가 아주 많을 때, B*Tree 인덱스 보다 훨씬 많은 공간을 차지한다.
- ③ 여러 비트맵 인덱스를 동시에 사용 가능하고, Bitwise Not 연산도 가능하다.
- ④ Lock에 의한 DML 부하가 심각해서 OLTP성 환경에는 부적합하다.

48

다음 중 SQL을 처리하기에 가장 효율적인 인덱스 구성을 고르시오.(단, 클러스터링 팩터는 고려하지 않는다.)

아 래

[SQL문]

```
select *
from 주문
where 주문유형코드 = :a
and 지점코드 = :b
and 계좌번호 = :c
and 주문일자 between :d and :e;
```

[값의 종류 개수(Number Of Distinct Value)]

주문유형코드	:	5
계좌번호	:	500,000
지점코드	:	100
주문일자	:	1,800

- ① 지점코드 + 주문일자 + 계좌번호
- ② 계좌번호 + 주문일자 + 지점코드
- ③ 주문일자 + 계좌번호 + 지점코드
- ④ 주문유형코드 + 지점코드 + 주문일자 + 계좌번호

49

다음 중 아래 SQL문을 튜닝하기 위한 분석을 진행 중 분석한 내용으로 가장 부적절한 것은?

아 래

```
create index 주문_idx on 주문(주문일자);

select  고객ID, 연락처, 고객등급
from    주문
where   주문일자 = '20110725'
and     배송상태 = 'ING';
```

- ① [주문일자 = '20110725'] 조건만으로 인덱스 탐색이 이루어진다.
- ② [배송상태 = 'ING'] 조건에 대한 필터링은 테이블 액세스 단계에서 이루어진다.
- ③ SQL 문장에 [배송상태 = 'ING'] 조건절이 없다면 테이블은 액세스하지 않아도 된다.
- ④ [배송상태 = 'ING'] 조건을 만족하는 레코드 비중이 아주 낮다면, 주문일자_idx 인덱스가 주문일자와 배송상태 두 컬럼을 모두 포함하도록 구성함으로써 쿼리 성능을 향상시킬 수 있다. 이 쿼리만 고려한다면, 두 컬럼 간 순서는 자유롭게 선택해도 무방하다.

50

성능 향상을 위해 아래 왼쪽 SQL을 오른쪽과 같이 변환하였다. 인덱스 구성을 보기와 같이 ①②③④ 순으로 바꿔 가며 SQL을 실행할 때, 오른쪽 변환된 SQL의 블록 I/O가 오히려 늘어나는 인덱스 구성을 2개 고르시오.

아 래

<pre>select * from 고객 where 가입일자 like '2010%' and 고객등급 between 'A' and 'B'</pre>	➔	<pre>select * from 고객 where 가입일자 like '2010%' and 고객등급 = 'A' union all select * from 고객 where 가입일자 like '2010%' and 고객등급 = 'B'</pre>
--	---	--

- ① 가입일자
- ② 고객등급
- ③ 가입일자 + 고객등급
- ④ 고객등급 + 가입일자

핵심정리

인덱스 설계를 위해 고려해야 할 요소

- 쿼리 수행 빈도
- 업무상 중요도
- 클러스터링 팩터
- 데이터량
- DML 부하(= 기존
인덱스 개수, 초당
DML 발생량, 자주
갱신되는 칼럼 포함
여부 등)
- 저장 공간
- 인덱스 관리 비용 등

결합 인덱스 구성

- 첫 번째 기준은, 조건절에
항상 사용되거나,
적어도 자주 사용되는
칼럼들을 선정
- 두 번째 기준은, 그렇게
선정된 칼럼 중 '≠'
조건으로 자주
조회되는 칼럼을
앞쪽에 두어야 한다.
- 세 번째 기준은, 소트
오퍼레이션을
생략하도록 하기 위해
칼럼을 추가

51

아래 트레이스 결과를 가장 적절히 설명한 보기를 2개 고르시오.

아래

Call	Count	CPU Time	Elapsed Time	Disk	Query	Current	Rows
Parse	1	0.010	0.012	0	0	0	0
Execute	1	0.000	0.000	0	0	0	0
Fetch	78	10.150	49.199	27830	266468	0	1909
Total	80	10.160	49.231	27830	266468	0	1909
Rows Row Source Operation							
1909 TABLE ACCESS BY INDEX ROWID TAB1 (cr=266468 pr=27830 pw=0 time=58480816 us)							
266476 INDEX RANGE SCAN TAB1_X01 (cr=511 pr=300 pw=0 time=1893462 us)OF ...							

- ① TAB1_X01 인덱스의 클러스터링 팩터는 매우 좋은 상태다.
- ② TAB1_X01 인덱스에 컬럼을 추가하면 성능을 높이는 데 매우 큰 도움이 된다.
- ③ TAB1_X01 인덱스 컬럼 순서를 조정하면 성능을 높이는 데 매우 큰 도움이 된다.
- ④ CPU Time과 Elapsed Time 간 39초가량 차이가 발생한 이유는, 27,000여 개 디스크 블록을 읽기 위한 I/O Call 과정에 발생한 대기 현상 때문일 가능성이 가장 높다.

52

다음 중 SQL에 사용된 다양한 액세스 유형을 분석해 시스템 전체적인 관점에서 전략적인 인덱스 설계를 하려고 한다. 결합 인덱스 키 칼럼을 선정하는 데 있어 가장 중요한 선택 기준 두 가지를 순서대로 나열한 것으로 가장 적절한 것은?

아래

- ㄱ. 데이터 분포
- ㄴ. '=' 조건으로 자주 조회되는지
- ㄷ. 조건절에 항상 또는 자주 사용되거나
- ㄹ. 정렬 기준 칼럼으로 자주 사용되는지

- ① ㄱ → ㄴ
- ② ㄴ → ㄱ
- ③ ㄷ → ㄴ
- ④ ㄷ → ㄹ

53

다음 중 아래 인덱스 구성과 SQL을 고려할 때, 실행계획 맨 아래(ID=5) 주문_IDX 인덱스의 액세스 조건으로 가장 적절한 것은?

아 래

< 인덱스 구성 >

```
create index 주문_IDX on 주문(주문일자, 고객번호);
```

```
select /*+ ordered use_nl(o) */ *
from   고객 c, 주문 o
where  c.가입일자 = '20130414'
and    o.고객번호 = c.고객번호
and    o.주문일자 = '20130414'
and    o.상품코드 = 'A123';
```

Execution Plan

```
-----
0      SELECT STATEMENT Optimizer=ALL ROWS
1  0    TABLE ACCESS (BY INDEX ROWID) OF '주문' (TABLE)
2  1      NESTED LOOPS
3  2      TABLE ACCESS (BY INDEX ROWID) OF '고객' (TABLE)
4  3          INDEX (RANGE SCAN) OF '고객_X01' (INDEX)
5  2          INDEX (RANGE SCAN) OF '주문_IDX' (INDEX)
```

- ① o.고객번호 = c.고객번호
- ② o.주문일자 = '20130414'
- ③ o.주문일자 = '20130414' and o.고객번호 = c.고객번호
- ④ o.주문일자 = '20130414' and o.고객번호 = c.고객번호 and
o.상품코드 = 'A123'

54

SQL 트레이스를 수집한 결과 Row Source Operation이 아래와 같았다. 튜닝을 위해 가장 우선적으로 검토할 사항으로 가장 적절한 것은? (단, 한 달간 주문 건수는 평균 50만 건이다.)

아 래

```
select c.고객명, c.연령, c.전화번호, o.주문일자, o.주문총금액, o.배송지주소
from 고객 c, 주문 o
where o.고객번호 = c.고객번호
and c.고객등급 = 'A'
and c.연령 between 51 and 60
and o.주문일자 between '20101201' and '20101231'
```

Rows	Row Source Operation
10	NESTED LOOPS
23	TABLE ACCESS BY INDEX ROWID 고객
2978	INDEX RANGE SCAN 고객_IDX
10	TABLE ACCESS BY INDEX ROWID 주문
28	INDEX RANGE SCAN 주문_IDX

- ① 고객_IDX 인덱스 칼럼 순서를 조정한다.
- ② 고객_IDX 인덱스에 칼럼을 추가한다.
- ③ 주문_IDX 인덱스에 칼럼을 추가한다.
- ④ 테이블 조인 순서를 변경한다.

55

테이블과 인덱스 구성은 아래와 같다. INDEX RANGE SCAN으로 데이터를 액세스할 수 있는 SQL로 가장 적절한 것은?

아 래

테이블 명 : 고객

[칼럼 리스트]

고객 ID : NOT NULL, VARCHAR2(10)

고객명 : NOT NULL, VARCHAR2(20)

직업코드 : NULLABLE, VARCHAR2(3)

[인덱스 구성]

고객_PK : 고객ID

고객_X01 : 직업코드, 고객명

- ① SELECT * FROM 고객 WHERE SUBSTR(고객ID,1,3) = '100'
- ② SELECT * FROM 고객 WHERE 고객명 = '김시협'
- ③ SELECT * FROM 고객 WHERE 직업코드 IS NULL
- ④ SELECT * FROM 고객 WHERE 고객ID = 1000000009

핵심정리

Hash Join 성능을 좌우하는 키 포인트

- 한 쪽 테이블이 가용 메모리에 담길 정도로 충분히 작아야 함
- Build Input 해시 키 컬럼에 중복 값이 거의 없어야 함

Hash Join 선택 기준

- 조인 컬럼에 적당한 인덱스가 없어 NL Join이 비효율적일 때
- 조인 컬럼에 인덱스가 있더라도 NL Join 드라이빙 집합에서 Inner 쪽 집합으로의 조인 액세스량이 많아 Random 액세스 부하가 심할 때
- Sort Merge Join 하기에는 두 테이블이 너무 커 소트 부하가 심할 때
- 수행빈도가 낮고 쿼리 수행 시간이 오래 걸리는 대용량 테이블을 조인할 때

서브쿼리 중에서 함수처럼 한 레코드당 정확히 하나의 값을 리턴하는 서브쿼리를 Scalar Subquery라고 한다. Scalar Subquery는 주로 select-list에서 사용되지만 몇 가지 예외사항을 뺀다면 컬럼이 될 수 있는 대부분 위치에서 사용 가능하다.

56

Oracle에서 TAB1, TAB2 순으로 NL 조인하도록 유도하고 싶다. 다음 중 ㉠ 안에 넣을 바른 힌트 사용법을 2개 고르시오.

아 래

```
SELECT /*+ ( ㉠ ) */ ...
FROM TAB1 A, TAB2 B
WHERE A.KEY = B.KEY
```

- ① ORDERED USE_NL(B)
- ② ORDERED USE_NL(TAB2)
- ③ LEADING(A) USE_NL(B)
- ④ DRIVING_SITE(A) USE_NL(B)

57

다음 중 Hash Join에 대한 설명으로 가장 부적절한 것을 2개 고르시오.

- ① 조인 연결고리에 equi-join 조건이 하나라도 있어야 한다.
- ② 일반적으로, 큰 집합으로 해시 테이블을 생성하고서 작은 집합을 읽으면서 이 해시 테이블을 탐색하는 게 유리하다.
- ③ OLTP 환경에서 수행빈도가 아주 높은 쿼리라도 부담 없이 사용할 수 있다.
- ④ Build Input으로 선택된 집합의 조인 컬럼에는 중복 값이 거의 없어야 효과적이다.

58

다음 중 스칼라 서브쿼리의 특징을 설명한 것으로 가장 부적절한 것은?

- ① 스칼라 서브쿼리를 이용한 조인은 NL 조인처럼 한 레코드씩 순차적으로 진행한다.
- ② 메인 쿼리의 레코드마다 스칼라 서브쿼리를 통해 하나의 레코드, 하나의 값만 리턴한다.
- ③ 조인에 실패할 경우, Null을 리턴한다.
- ④ 입력 값(=조인 컬럼)과 출력 값(=리턴 값)을 Shared Pool에 있는 Result Cache에 캐싱했다가 같은 입력 값에 대해서는 캐싱된 값을 리턴함으로써 조인 부하를 줄여준다.

다음 중 아래 SQL1과 SQL2의 실행계획에 대한 설명으로 가장 부적절한 것은?

아 래

```
[SQL1]
SELECT /*+ LEADING(A) USE NL(B) */
      b.상품분류코드, a.기준일자
      , SUM (a.판매금액) AS 판매금액
      , SUM (a.판매수량) AS 판매수량
FROM 일별매출 a, 상품 b
WHERE a.기준일자 BETWEEN TO_DATE (:v_시작기준일자, 'YYYYMMDD')
                        AND TO_DATE (:v_종료기준일자, 'YYYYMMDD')
      AND b.상품코드 = a.상품코드
      AND b.판매시작일자 >= TO_DATE (:v_판매시작일자, 'YYYYMMDD')
GROUP BY b.상품분류코드, a.기준일자;
```

Id	Operation	Name
0	SELECT STATEMENT	
1	HASH GROUP BY	
* 2	FILTER	
3	NESTED LOOPS	
4	NESTED LOOPS	
5	PARTITION RANGE ITERATOR	
6	TABLE ACCESS BY LOCAL INDEX ROWID	일별매출
* 7	INDEX RANGE SCAN	일별매출_X1
* 8	INDEX RANGE SCAN	상품_PK
* 9	TABLE ACCESS BY INDEX ROWID	상품

```
[SQL2]
SELECT /*+ LEADING(A) USE NL(B) */
      b.상품분류코드, a.기준일자
      , SUM (a.판매금액) AS 판매금액
      , SUM (a.판매수량) AS 판매수량
FROM (SELECT a.기준일자
      , a.상품코드
      , SUM (a.판매금액) AS 판매금액
      , SUM (a.판매수량) AS 판매수량
FROM 일별매출 a
WHERE a.기준일자 BETWEEN TO_DATE (:v_시작기준일자, 'YYYYMMDD')
                        AND TO_DATE (:v_종료기준일자, 'YYYYMMDD'))
      a
      , 상품 b
WHERE b.상품코드 = a.상품코드
      AND b.판매시작일자 >= TO_DATE (:v_판매시작일자, 'YYYYMMDD')
GROUP BY b.상품분류코드
      , a.기준일자;
```

Id	Operation	Name
0	SELECT STATEMENT	
1	HASH GROUP BY	
2	NESTED LOOPS	
3	NESTED LOOPS	
4	VIEW	
5	HASH GROUP BY	
* 6	FILTER	
7	PARTITION RANGE ITERATOR	
8	TABLE ACCESS BY LOCAL INDEX ROWID	일별매출
* 9	INDEX RANGE SCAN	일별매출_X1
* 10	INDEX UNIQUE SCAN	상품_PK
* 11	TABLE ACCESS BY INDEX ROWID	상품

[테이블]

```
CREATE TABLE 상품 (
  상품코드 VARCHAR2(10) NOT NULL
, 상품명 VARCHAR2(100) NOT NULL
, 상품분류코드 VARCHAR2(5) NOT NULL
, 판매시작일자 DATE NOT NULL
, 판매종료일자 DATE NULL);
```

```
CREATE UNIQUE INDEX 상품_PK ON 상품 (상품코드);
CREATE INDEX 상품_X1 ON 상품 (판매시작일자);
```

```
CREATE TABLE 일별매출 (
  기준일자 DATE NOT NULL
, 상품코드 VARCHAR2(10) NOT NULL
, 판매금액 NUMBER(10) NOT NULL
, 판매수량 NUMBER(10) NOT NULL);
```

```
CREATE UNIQUE INDEX 일별매출_PK ON 일별매출 (상품코드, 기준일자);
CREATE INDEX 일별매출_X1 ON 일별매출 (기준일자);
```

- 단, 상품 테이블은 판매시작일자별 상품 수가 비슷한 것으로 가정한다.
- 단, 일별매출 테이블은 기준일자로 파티셔닝되어 있다고 가정한다.

- ① SQL1은 조인 과정에서 과도한 I/O가 발생할 수 있다.
- ② SQL1은 판매시작일자 조건을 만족하는 상품 건수가 적다면, 조인순서 변경만으로 성능이 개선될 가능성이 높다.
- ③ SQL2는 v_판매시작일자 변수에 오랜 과거일자를 입력할 때보다 최근 일자를 입력할 때 비효율이 더 적다.
- ④ Query Transformer는 쿼리변환을 수행하여 SQL1에 대해 SQL2와 유사한 실행계획을 수립하기도 한다.

60

다음 중 아래와 같은 SQL에서 가능한 조인 방법을 모두 나열한 것은?

아 래

```
SELECT emp.ename, salgrade.grade
FROM emp t1, salgrade t2
WHERE t1.sal BETWEEN t2.losal AND t2.hisal
```

- ① Nested Loop Join
- ② Nested Loop Join, Sort Merge Join
- ③ Sort Merge Join, Hash Join
- ④ Nested Loop Join, Sort Merge Join, Hash Join

핵심정리

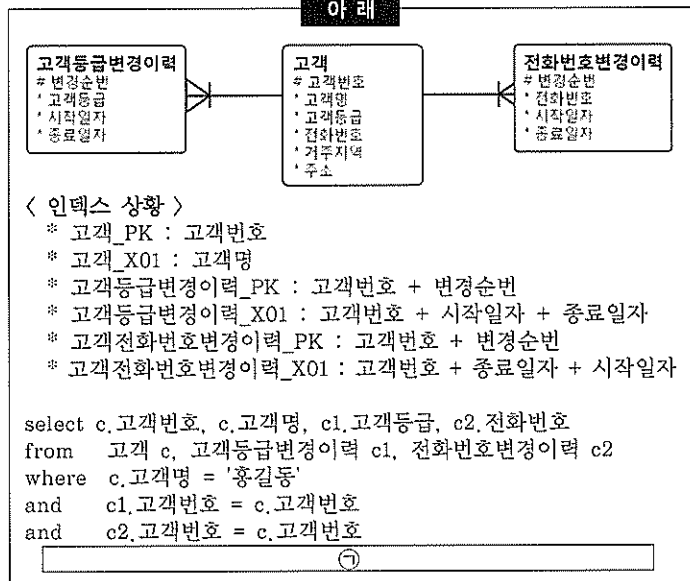
선분이력이란?

예를 들어 고객별연체금액
변경이력을 관리할 때
이력의 시작시점만을
관리하는 것을 '점이력'
모델이라고 하고,
시작시점과 종료시점을
함께 관리하는 것을
'선분이력' 모델이라고
한다

61

아래와 같이 고객 테이블과 2개의 변경이력 테이블이 있다. 두 변경이력 테이블에 있는 시작일자, 종료일자는 선분이력으로 관리된다. 즉, 시작 일자에는 이력 레코드 생성일자가 입력되고, 종료일자에는 처음에 '99991231'로 입력되었다가 다음 번 새 이력 레코드가 생성되는 순간 바로 전 일자로 갱신된다. 이 세 테이블에서 이름이 '홍길동'인 고객의 1998년 5월 29일자 고객등급과 전화번호를 조회하려고 한다. 아래 ㉠에 들어갈 조건절로 가장 적절한 것은? (단, 고객번호와 변경순번을 제외한 나머지 속성은 모두 문자형)

아래



- ① and c1.변경순번 = (select max(변경순번)
from 고객등급변경이력
where 시작일자 <= '19980529')
and c2.변경순번 = (select max(변경순번)
from 전화번호변경이력
where 시작일자 <= '19980529')
- ② and c1.변경순번 = (select 변경순번
from 고객등급변경이력
where '19980529' between 시작일자 and 종료일자)
and c2.변경순번 = (select 변경순번
from 전화번호변경이력
where '19980529' between 시작일자 and 종료일자)
- ③ and '19980529' between c1.시작일자 and c1.종료일자
and '19980529' between c2.시작일자 and c2.종료일자
- ④ and '19980529' between c1.시작일자 and c1.종료일자
and c1.종료일자 >= c2.시작일자
and c1.시작일자 <= c2.종료일자

핵심정리

Local 파티션 인덱스 :
테이블 파티션과 1:1로
대응되도록 파티셔닝한
인덱스.
인덱스 파티션 키를
사용자가 따로 지정하지
않으며, 테이블과 1:1
관계를 유지하도록
DBMS가 자동으로
관리해 줌.
SQL Server에선
'정렬된(aligned) 파티션
인덱스'라고 부른다.

62

다음 중 Local 파티션 인덱스의 특징과 거리가 먼 것은?

- ① 테이블 파티션과 1:1 대응 관계
- ② 테이블에 파티션 ADD/DROP/SPLIT/EXCHANGE 작업 시, 인덱스 파티션도 자동 관리됨
- ③ Local 파티션 인덱스의 경우, 테이블 파티션 키(=인덱스 파티션 키)가 인덱스 키 선두 컬럼에 위치해야 함. 예를 들어 테이블과 인덱스 파티션 키가 '주문일자'이면, Local 파티션 인덱스 키는 '주문일자'로 시작해야 함
- ④ 테이블 파티션 키가 SQL 조건절에 없을 때, 인덱스 사용 시 비효율 발생

63

우리가 개발하고자 하는 시스템은 업무적으로 7~9월에 매출이 집중돼 있다. 아래 월별매출집계 테이블(보관주기 3년)을 파티셔닝하기로 했고, 매출연월 기준으로 각 파티션에 데이터를 고르게 분산 저장하고자 할 때, 다음 중 사용할 파티션 전략으로 가장 적절한 것을 2개 고르시오.

아 래

월별매출집계

매출연월
고객번호
판매지점코드
상품구분코드
* 판매금액
*
*

- ① Range 파티션
- ② List 파티션
- ③ Hash 파티션
- ④ Manual 파티션

Oracle이 지원하는 파티션 유형

1) Range 파티셔닝

- 파티션 키 값의 범위(Range)로 분할
- 파티셔닝의 가장 일반적인 형태이며, 주로 날짜 칼럼을 기준으로 함에) 판매 데이터를 월별로 분할

2) Hash 파티셔닝

- 파티션 키 값에 해시 함수를 적용하고, 거기서 반환된 값으로 파티션 매핑
- 데이터가 모든 파티션에 고르게 분산되도록 DBMS가 관리
 - 각 로우의 저장 위치 예측 불가
- 파티션 키의 데이터 분포가 고른 칼럼이어야 효과적
예) 고객번호, 주문일련번호 등
- 병렬처리 시 성능효과 극대화
- DML 경합 분산에 효과적

3) List 파티셔닝

- 불연속적인 값의 목록을 각 파티션에 지정
- 순서와 상관없이, 사용자가 미리 정한 그룹핑 기준에 따라 데이터를 분할 저장
예) 판매 데이터를 지역별로 분할

4) Composite 파티셔닝

- Range나 List 파티션 내에 또 다른 서브 파티션(Range, Hash, List) 구성
예) Range + List 또는 List + Hash 등
 - Range나 List 파티션이 갖는 이점 + 각 서브 파티션 구성의 이점
-

SQL

Professional • Developer

고급 SQL 튜닝

-
- 제1절 고급 SQL 활용
 - 제2절 소스 튜닝
 - 제3절 DML 튜닝
 - 제4절 파티션 활용
 - 제5절 배치 프로그램 튜닝
-

핵심정리

소트와 관련된

오퍼레이션 유형

- 1) Sort Aggregate :
전체 로우를 대상으로
집계를 수행할 때
나타나며, Oracle
실행계획에 sort라는
표현이 사용됐지만
실제 소트가 발생하진
않는다.
- 2) Sort Order By :
정렬된 결과집합을
얻고자 할 때
나타난다.
- 3) Sort Group By :
Sorting 알고리즘을
사용해 그룹별 집계를
수행할 때 나타난다.
- 4) Sort Unique :
선택된 결과집합에서
중복 레코드를
제거하고자 할 때
나타난다. Union
연산자나 아래와 같이
Distinct 연산자를
사용할 때가
대표적이다.
- 5) Sort Join :
Sort Merge Join을
수행할 때 나타난다.
- 6) Window Sort :
윈도우 함수를 수행할
때 나타난다.

64

다음 중 테이블 tab1에 다음과 같은 데이터가 존재할 경우, 올바르게 설명한 것은?(각 칼럼의 타입은 number이다.)

아 래

col1	col2	col3
10	20	NULL
15	NULL	NULL
50	70	20

- ① select sum(col2) from tab1 의 결과는 NULL이다.
- ② select sum(col1 + col2 + col3) from tab1 의 결과는 185 이다.
- ③ select sum(col2 + col3) from tab1 의 결과는 90 이다.
- ④ select sum(col2) + sum(col3) from tab1 의 결과는 90 이다.

65

다음 중 아래 실행계획과 보기의 SQL을 서로 연결할 때, 보기 SQL 중 실행계획이 없는 것은?

아 래

Id	Operation	Id	Operation
0	SELECT STATEMENT	0	SELECT STATEMENT
1	SORT AGGREGATE	1	SORT ORDER BY
2	TABLE ACCESS FULL	2	TABLE ACCESS FULL
Id	Operation	Id	Operation
0	SELECT STATEMENT	0	SELECT STATEMENT
1	SORT UNIQUE	1	SORT GROUP BY
2	TABLE ACCESS FULL	2	TABLE ACCESS FULL

- ① select * from emp order by deptno;
- ② select distinct deptno from emp;
- ③ select deptno, count(*) from emp group by deptno;
- ④ select empno, ename, sal, avg(sal) over (partition by deptno) from emp;

66

다음 중 정렬(Sort) 오퍼레이션을 포함하지 않는 SQL은?

- ① select * from emp order by ename
- ② select * from emp where deptno = 20
union all
select * from emp where deptno = 30
- ③ select empno, ename, rank() over (partition by deptno order by sal desc) from emp
- ④ select * from dept d, emp e where d.deptno = e.deptno
option(force order, merge join)

67

다음 중 아래 ERD와 Dictionary 조회 결과를 고려할 때, 보기 중 union 대신 union all을 사용해도 가능한 것으로 가장 적절한 것은?

아 래

EMP

EMPNO
* ENAME
* DEPTNO
o JOB
o MGR
o SAL

SQL> select column_name, num_distinct
2 from user_tab_columns
3 where table_name = 'EMP';

COLUMN_NAME	NUM_DISTINCT
EMPNO	14
ENAME	14
DEPTNO	3
JOB	5
MGR	6
SAL	12

- ① select deptno, job, mgr from emp where empno = 7499
union
select deptno, job, mgr from emp where empno = 7654;
- ② select job, mgr from emp where deptno = 10
union
select job, mgr from emp where deptno = 20;
- ③ select deptno, job, mgr from emp where deptno = 10
union
select deptno, job, mgr from emp where deptno = 20;
- ④ select empno, job, mgr from emp where deptno = 10
union
select empno, job, mgr from emp where deptno = 20;

68

모니터링 결과, 아래 쿼리의 수행빈도가 가장 높아 시스템에 미치는 영향이 큰 것으로 조사됐다. 다음 중 고려할 튜닝 방안으로 가장 부적절한 것은?

아 래

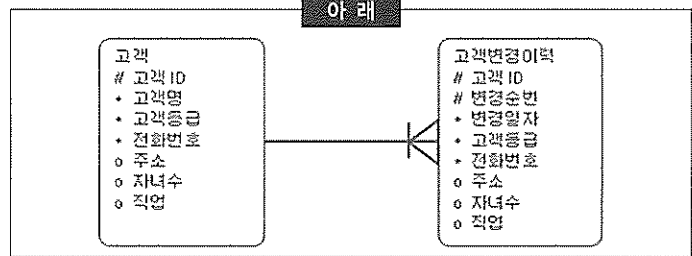
```
create index 게시판_idx on 게시판(작성일시, 게시판구분);

select *
from (
    select *
    from   게시판
    where  게시판구분 = :gubun
    and    작성일시 >= trunc(sysdate) - 1
    order by 작성일시, 작성자명
)
where rownum <= 100;
```

- ① rownum 조건을 인라인 뷰 안에 사용한다.
- ② 게시판 구분의 데이터 분포(선택도)를 고려해 인덱스를 [작성일시 + 작성자명] 순으로 변경할지 검토한다. 최종 결정 시 다른 SQL에 미치는 영향도도 검토해야 한다.
- ③ 게시판 구분의 데이터 분포(선택도)를 고려해 인덱스를 [게시판구분 + 작성일시] 순으로 변경할지 검토한다. 최종 결정 시 다른 SQL에 미치는 영향도도 검토해야 한다.
- ④ 작성자명이 데이터 정렬 순서로서 의미 있는 것인지 현업 사용자에게 문의한 후, 불필요하다면 제거한다.

69

다음 중 아래의 고객과 고객변경이력 테이블에서 전체 고객을 대상으로 2010년 12월 4일자 이력을 조회하려고 할 때, 가장 효과적인 SQL인 것은?



- ① select b.고객ID, b.변경순번, b.전화번호, b.주소, b.자녀수, b.직업, b.고객등급
from 고객 a, 고객변경이력 b
where b.고객번호 = a.고객번호
and b.변경순번 = (select max(변경순번)
from 고객변경이력
where 고객번호 = a.고객번호
and 변경일자 <= '20101204')
- ② select b.고객ID, b.변경순번, b.전화번호, b.주소, b.자녀수, b.직업, b.고객등급
from (select 고객ID, max(변경순번) 변경순번
from 고객변경이력
where 변경일자 <= '20101204'
group by 고객ID) a
, 고객변경이력 b
where b.고객번호 = a.고객번호
and b.변경순번 = a.변경순번
- ③ select 고객ID, 변경순번, 전화번호, 주소, 자녀수, 직업, 고객등급
from (select 고객ID, 변경순번
, rank() over (partition by 고객ID order by 변경순번 desc) rnum
, 전화번호, 주소, 자녀수, 직업, 고객등급
from 고객변경이력
where 변경일자 <= '20101204')
where rnum = 1
- ④ select 고객ID, 변경순번, 전화번호, 주소, 자녀수, 직업, 고객등급
from (select 고객ID, 변경순번
, max(변경순번) over (partition by 고객ID) 변경순번2
, 전화번호, 주소, 자녀수, 직업, 고객등급
from 고객변경이력
where 변경일자 <= '20101204')
where 변경순번 = 변경순번2

핵심정리

Oracle에서 Direct
Path Insert 방식으로
데이터를 입력하는 방법

- insert select 문장에
/*+ append */ 힌트
사용
- 병렬 모드로 insert
- direct 옵션을
지정하고
SQL*Loader(sqlldr)
로 데이터를 로드
- CTAS(create table
... as select)
문장을 수행

70

아래 SQL은 v_주식선물구분 바인드 변수 입력에 따라 선택적으로 주식
월별시세 또는 선물월별시세 테이블에 데이터를 입력한다. 실행 정보와
같이 바인드 변수를 입력하고 쿼리를 100번(SID=100)과 200번
(SID=200) 세션에서 순차적으로 수행하였을 때 200번 세션의 상태로
올바른 것은?

아 래

[SQL]

```
INSERT /*+ APPEND */ ALL
  WHEN :v_주식선물구분 = '주식'
  THEN INTO 주식월별시세 (종목코드, 거래일자, 증가)
  WHEN :v_주식선물구분 = '선물'
  THEN INTO 선물월별시세 (종목코드, 거래일자, 증가)
SELECT   a.종목코드
         , :v_기준일자 AS 거래일자
         , AVG (a.증가) AS 증가
  FROM   주식일별시세 a
  WHERE  :v_주식선물구분 = '주식'
  AND a.거래일자 BETWEEN ADD_MONTHS (:v_기준일자, -1) AND :v_기준일자
  GROUP BY a.종목코드
UNION ALL
SELECT   a.종목코드
         , :v_기준일자 AS 거래일자
         , AVG (a.증가) AS 증가
  FROM   선물일별시세 a
  WHERE  :v_주식선물구분 = '선물'
  AND a.거래일자 BETWEEN ADD_MONTHS (:v_기준일자, -1) AND :v_기준일자
  GROUP BY a.종목코드;
```

[실행]

```
-- 세션 100
EXEC :v_주식선물구분 := '주식';
/

-- 세션 200
EXEC :v_주식선물구분 := '선물';
/
```

- ① TM 락을 Row-X(SX) 모드로 요청하고 대기한다.
- ② TX 락을 Exclusive 모드로 요청하고 대기한다.
- ③ TM 락을 Exclusive 모드로 요청하고 대기한다.
- ④ 정상적으로 수행된다.

핵심정리

대량의 데이터를 일반 Update문으로 갱신하면 상당히 오랜 시간이 소요될 수 있다. 다음과 같은 이유 때문이며, Delete문일 때도 마찬가지다.

- 테이블 데이터를 갱신하는 본연의 작업
- 인덱스 데이터까지 갱신
- 버퍼 캐시에 없는 블록을 디스크에서 읽어 버퍼 캐시에 적재한 후에 갱신
- 내부적으로 Redo와 Undo 정보 생성
- 블록에 빈 공간이 없으면 새 블록 할당(→ Row Migration 발생)

71

다음 중 오라클에서 아래와 같이 nologging 옵션과 append 힌트를 사용하여 대용량 데이터를 INSERT할 때 극적인 성능개선 효과를 가져 오는 원리를 설명한 것으로 가장 부적절한 것은?

아 래

```
alter table 수납 nologging;

insert /*+ append */ into 수납
select * from 수납_임시
where 수납일시 between '20131204120000' and '20131204235959'
```

- ① Lock을 사용하지 않고 빠르게 입력한다.
- ② Redo와 Undo 발생량을 최소화한다.
- ③ 데이터 입력이 가능한 빈 블록을 찾기 위해 Freelist를 조회하지 않아도 된다.
- ④ 버퍼 캐시를 거치지 않고 데이터를 세그먼트 HWM 뒤쪽에 순차적으로 입력한다.

72

다음 중 DML 튜닝방안으로 가장 부적절한 것은?

- ① 대량 데이터에 대해 작업할 경우 인덱스를 Unusable 상태로 변경하고 작업한 후에 인덱스를 재생성하는 하는 것이 빠를 수 있다.
- ② 대량의 데이터를 빠르게 UPDATE하기 위해 테이블을 nologging 모드로 변경하고 작업을 시작한다.
- ③ 오라클의 수정가능 조인 뷰는 키-보존 테이블에만 입력, 수정, 삭제가 허용된다.
- ④ 대량 데이터에 대한 UPDATE나 DELETE문의 경우 기존 데이터를 임시 테이블에 저장하고 테이블을 TRUNCATE한 후 임시 테이블을 이용해 다시 입력하는 것이 빠르다.

다음 중 (A), (B) 두 SQL에 대한 설명으로 가장 부적절한 것은?

아 래

```
(A)
UPDATE 급여지급 T
  SET T.월급여 = (
    SELECT S.월급여
    FROM 사원 S
    WHERE S.사원번호 = T.사원번호
  )

WHERE T.급여월 = '201101'
AND EXISTS (
  SELECT 1
  FROM 사원 S
  WHERE S.사원번호 = T.사원번호
  AND S.부서코드 = '30'
)
```

```
(B)
MERGE INTO 급여지급 T
USING (
  SELECT S.사원번호
    ,S.월급여
  FROM 사원 S
  WHERE S.부서코드 = '30'
) S
ON (T.급여월 = '201101'
    AND T.사원번호 = S.사원번호)
WHEN MATCHED THEN
  UPDATE SET T.월급여 = S.월급여
```

- ① (A)와 (B)의 수정되는 데이터 건수는 같다.
- ② 둘 다 NL 방식으로 조인한다면, (A)가 (B)보다 사원 테이블의 반복 액세스가 많아 발생하는 I/O도 많다.
- ③ 둘 다 해시 방식으로 조인한다면, 아무리 처리량이 많아도 성능에 차이가 없다.
- ④ (B)는 INSERT 처리도 같이 하도록 문장을 재구성할 수 있다.

핵심정리

파티셔닝이 필요한 이유

- 관리적 측면 :
파티션 단위 백업,
추가, 삭제, 변경
- 성능적 측면 :
파티션 단위 조회 및
DML 수행, 경합 및
부하 분산

74

야간에 단독으로 DML을 수행하는 대용량 배치 프로그램 속도를 항상 시키려고 한다. 다음 중 고려할 만한 튜닝 방안으로 가장 부적절한 것은?

- ① 인덱스를 제거했다가 작업 완료 후 다시 생성한다.
- ② 수정 가능 조인 뷰(Updatable Join View)나 Merge문을 활용한다.
- ③ Oracle이라면 update문을 수행하기 전에 테이블을 nologging 모드로 변경한다.
- ④ SQL Server라면 최소 로깅(minimal logging) 모드 Insert 기능을 활용한다.

75

다음 중 테이블 또는 인덱스를 파티셔닝하는 이유로 가장 부적절한 것은?

- ① 가용성 향상
- ② 저장효율 개선
- ③ 조회성능 개선
- ④ 경합 분산

76

다음 중 파티션에 대한 설명으로 가장 부적절한 것은?

- ① 파티션은 애플리케이션에 투명하다. 즉, 파티션되지 않은 테이블을 새롭게 파티셔닝하더라도 쿼리는 수정하지 않아도 된다.
- ② 기존에 사용되던 인덱스를 파티셔닝하면 그 인덱스를 액세스하던 일부 SQL 성능이 오히려 느려질 수 있다.
- ③ 파티션 칼럼에 대한 검색조건을 (변수가 아닌) 상수 값으로 제공해야 Partition Pruning이 작동한다.
- ④ 파티션 단위로 인덱스를 재생성할 수 있다.

77

다음 중 아래와 같이 주문 테이블을 생성하고 주문 데이터가 월평균 100만 건이라고 가정할 때, 보기 중 블록 I/O 측면에서 비효율이 없는 SQL을 2개 고르시오. (단, ④번 항목 일자 테이블에 대한 블록 I/O는 소량이므로 무시하기로 함)

아 래

```
create table 주문 (
  고객번호 varchar2(12), 주문일자 varchar2(8), 주문시각 varchar2(6), ... , ...
)
partition by range(주문일자) (
  partition m201101 values less than('20110201')
, partition m201102 values less than('20110301')
, partition m201103 values less than('20110401')
, partition m201104 values less than('20110501')
, partition m201105 values less than('20110601')
, partition m201106 values less than('20110701')
, partition mmaxval values less than(maxvalue)
);
```

- ① select * from 주문 where 주문일자 between '20110101' and '20110331'
- ② select * from 주문 where 주문일자 between '20110101' and '20110131'
union all
select * from 주문 where 주문일자 between '20110201' and '20110228'
union all
select * from 주문 where 주문일자 between '20110301' and '20110331'
- ③ select * from 주문 where substr(주문일자, 1, 6) in ('201101', '201102', '201103')
- ④ select /*+ ordered use_nl(b) */ b.*
from 일자 a, 주문 b
where a.일자 between '20110101' and '20110331'
and b.주문일자 = a.일자

핵심정리

파티션 인덱스

- Local Prefixed
파티션 인덱스
- Local NonPrefixed
파티션 인덱스
- Global Prefixed
파티션 인덱스
- Global NonPrefixed
파티션 인덱스 (—
Oracle Not Support)
- 비파티션
(NonPartitioned)
인덱스

78

거래 테이블이 아래와 같을 때, 다음 중 LOCAL PREFIXED 파티션 인덱스로 가장 적절한 것은?

아 래

```
create table 거래 ( 고객번호 varchar2(10), 중목코드 varchar2(20),
거래일시 date, ... )
partition by range (거래일시) (
partition p2010 values less than(to_date('20110101', 'yyyymmdd'))
, partition p2011 values less than(to_date('20120101', 'yyyymmdd'))
, partition p2012 values less than(to_date('20130101', 'yyyymmdd'))
, partition p2013 values less than(to_date('20140101', 'yyyymmdd'))
, partition pmax values less than(maxvalue)
);
```

- ① create index 거래_N1 on 거래(거래일시) local;
- ② create index 거래_N2 on 거래(고객번호) local;
- ③ create index 거래_N3 on 거래(중목코드) local;
- ④ create index 거래_N4 on 거래(중목코드, 거래일시) local;

79

다음 중 아래 DDL 스크립트를 보고, 보기 중 거래_IDX1과 거래_IDX2 인덱스 각각에 해당하는 가장 적절한 인덱스 유형을 2개 고르시오.

아 래

```
create table 거래 (
계좌번호 number, 상품번호 varchar2(6)
, 거래일자 varchar2(8), 거래량 number, 거래금액 number )
partition by range(거래일자) (
partition p1 values less than('20110101')
, partition p2 values less than('20120101')
, partition px values less than( maxvalue )
);

create index 거래_idx1 on 거래(거래일자, 상품번호) GLOBAL
partition by range(거래일자) (
partition p1 values less than('20120101')
, partition px values less than( maxvalue )
);

create index 거래_idx2 on 거래(계좌번호, 거래일자) LOCAL;
```

- ① Global Prefixed
- ② Global Nonprefixed
- ③ Local Prefixed
- ④ Local Nonprefixed

핵심정리

배치 프로그램의 특징

- 사용자와의 상호작용 없이
- 대량의 데이터를 처리하는
- 일련의 작업들을 묶어
- 정기적으로 반복 수행하거나
- 정해진 규칙에 따라 자동으로 수행

80

다음 중 배치(Batch) 프로그램 튜닝 방안으로 가장 부적절한 것은?

- ① 시스템 사용자의 업무시간이 종료되자마자, 동시에 수행 가능한 모든 배치 프로그램을 집중적으로 수행함으로써 총 소요시간을 단축한다. 그래야 문제가 생겼을 때 대처할 수 있는 시간을 확보할 수 있다.
- ② 개별 프로그램 측면에서도 최초 응답속도보다는 전체 처리속도 최적화를 목표로 설정해야 한다.
- ③ 파티션과 병렬처리를 적절히 활용한다.
- ④ 여러 배치 프로그램에서 공통적으로 사용하는 집합을 정의해 이를 임시 테이블로 생성한다.

81

다음 중 오라클 병렬 프로세싱에 대한 설명으로 가장 부적절한 것은?

- ① 일련의 처리 과정이 모두 병렬로 진행되도록 해야 병렬 효과를 극대화할 수 있다.
- ② 옵티마이저는 병렬 프로세스 간 통신량을 최소화하려고 노력한다.
- ③ 병렬 DML 시, Exclusive 모드 TM Lock이 걸려 해당 테이블을 갱신하는 다른 트랜잭션이 모두 블록킹되므로 주의해야 한다.
- ④ 병렬 Update는 Redo Log를 생성하지 않는다.

핵심정리

pq_distribute 힌트를 사용함으로써 옵티마이저의 선택을 무시하고 사용자가 직접 조인을 위한 데이터 분배 방식을 결정할 수 있다.

- 옵티마이저가 파티션된 테이블을 적절히 활용하지 못하고 동적 재분할을 시도할 때
- 기존 파티션 카를 무시하고 다른 키 값으로 동적 재분할하고 있을 때
- 통계정보가 부정확하거나 통계정보를 제공하기 어려운 상황(—옵티마이저가 잘못된 판단을 하기 쉬운 상황)에서 실행계획을 고정시키고자 할 때
- 기타 여러 가지 이유로 데이터 분배 방식을 변경하고자 할 때

82

주문 테이블은 주문일자 기준으로 Range 파티셔닝, 고객번호 기준으로 Hash 서브 파티셔닝 되어 있는 상태다. 고객 테이블은 비파티션(Non-Partitioned) 테이블이다. 등록된 고객 수는 100만 명이고, 월 평균 주문 레코드는 2,000만 건이다. 이 상태에서 두 테이블을 조인하는 병렬 쿼리를 수행했더니 평소보다 3배 이상 오래 걸렸고, 실행계획을 확인해 보니 아래와 같았다. Oracle에서 이 병렬 쿼리의 속도를 향상시키기 위해 추가해야 할 힌트로 가장 적절한 것은?

아 래

```
select /*+ ordered use_hash(o) parallel(c 2) parallel(o 2) */
      o.고객번호, sum(o.주문금액), min(c.등급코드)
from   고객 c, 주문 o
where  o.고객번호 = c.고객번호
and    o.주문일자 between '20100901' and '20100930'
group by o.고객번호
```

Id	Operation	Name	Pstart	Pstop	TQ	IN-OUT
0	SELECT STATEMENT					
1	PX COORDINATOR					
2	PX SEND QC (RANDOM)	:TQ10003			Q1,03	P->S
3	SORT GROUP BY				Q1,03	PCWP
4	PX RECEIVE				Q1,03	PCWP
5	PX SEND HASH	:TQ10002			Q1,02	P->P
6	SORT GROUP BY				Q1,02	PCWP
7	HASH JOIN				Q1,02	PCWP
8	PX RECEIVE				Q1,02	PCWP
9	PX SEND HASH	:TQ10000			Q1,00	P->P
10	PX BLOCK ITERATOR				Q1,00	PCWC
11	TABLE ACCESS FULL	고객			Q1,00	PCWP
12	PX RECEIVE				Q1,02	PCWP
13	PX SEND HASH	:TQ10001			Q1,01	P->P
14	PX BLOCK ITERATOR		1	8	Q1,01	PCWC
15	TABLE ACCESS FULL	주문	1	8	Q1,01	PCWP

- ① pq_distribute(o none none)
- ② pq_distribute(o none broadcast)
- ③ pq_distribute(o partition none)
- ④ pq_distribute(o hash hash)



SQL

Professional • Developer

SQL 전문가 실기문제

실기문제 1

아래 좌측과 같은 월별지점매출 테이블을 읽어서 우측과 같은 형태, 즉 각 지점별로 판매월과 함께 증가하는 누적매출(running total)을 구하는 SQL 을, 아래 두 가지 방식으로 작성하시오.

- ① 윈도우 함수를 이용한 방식으로 작성
- ② 윈도우 함수나 스칼라 서브쿼리를 지원하지 않는 DBMS 에서 활용할 수 있는 방식으로 작성
(단, 전체범위처리에 최적화된 방식으로 작성할 것)

아 래

	지점	판매월	매출
1	10	1	521
2	10	2	684
3	10	3	590
4	20	1	537
5	20	2	650
6	20	3	500
7	20	4	919
8	20	5	658
9	30	1	631
10	30	2	736
11	30	3	513
12	30	4	970
13	30	5	939
14	30	6	666

누적매출

	지점	판매월	매출	누적매출
1	10	1	521	521
2	10	2	684	1205
3	10	3	590	1795
4	20	1	537	537
5	20	2	650	1187
6	20	3	500	1687
7	20	4	919	2606
8	20	5	658	3264
9	30	1	631	631
10	30	2	736	1367
11	30	3	513	1880
12	30	4	970	2850
13	30	5	939	3789
14	30	6	666	4455

실기문제 2

아래 SQL 과 트레이스 결과를 분석해서 개선된 SQL 을 작성하시오.

① 원하는 실행계획이 정확히 나오도록 힌트도 함께 기술할 것

② 최적의 인덱스 구성방안도 함께 제시할 것

(성능개선에 도움이 되지 않는, 필요 이상의 컬럼을 추가하는 것은 감점 요인이므로 주의)

아 래

[SQL 문]

```
select o.주문일시, o.주문번호, c.고객번호, c.고객명, c.연락처, o.주문금액, o.배송지
from   고객 c, 주문 o
where  o.주문일시 between to_date('20150301', 'yyyymmdd')
and to_date('20150314235959', 'yyyymmddhh24miss')
and    o.고객번호 = c.고객번호
and    c.거주지역코드 || c.고객명 in ('02김철수', '05홍길동')
order by o.주문일시, c.고객명
```

[SQL 트레이스]

Call	Count	CPU Time	Elapsed Time	Disk	Query	Current	Rows
Parse	1	0.000	0.002	0	0	0	0
Execute	1	0.000	0.000	0	0	0	0
Fetch	2	0.828	7.136	65296	114341	0	5
Total	4	0.828	7.138	65296	114341	0	5

Rows Row Source Operation

```
5  SORT ORDER BY (cr=114341 pr=65296 pw=0 time=0 us cost=21342)
5  HASH JOIN  (cr=114341 pr=65296 pw=0 time=28 us cost=21127)
20 TABLE ACCESS FULL 고객 (cr=76929 pr=36924 pw=0 time=38 us cost=21019)
45185 PARTITION RANGE SINGLE PARTITION: 1 1 (cr=37412 pr=28372 pw=0 time=33696 us cost=107)
45185 TABLE ACCESS FULL 주문 PARTITION: 1 1 (cr=37412 pr=28372 pw=0 time=22209 us cost=107)
```

[인덱스 구성]

고객 테이블

고객_PK : 고객번호

주문 테이블

주문_PK : 주문번호

실기문제 3

같은 데이터를 두 번 읽지 않고도 같은 결과집합을 출력하도록 아래 두 SQL 을 각각 재작성하시오.
(단, 부분범위처리 불가능한 상황임. 즉, 전체범위처리 기준으로 튜닝할 것)

- '주문일자'의 데이터 타입은 문자형 8자리
- 거래 업체는 10,000개
- 월평균 주문건수는 100만 건

아 래

[SQL1]

```
select b.주문번호, b.업체번호, b.주문일자, b.주문금액, a.총주문횟수, a.평균주문금액, a.최대주문금액
from (
  select  업체번호, count(*) 총주문횟수, avg(주문금액) 평균주문금액, max(주문금액) 최대주문금액
  from    주문
  where   주문일자 like '201509%'
  group by 업체번호
) a, 주문 b
where b.업체번호 = a.업체번호
and   b.주문일자 like '201509%'
order by a.평균주문금액 desc
```

[SQL2]

```
select b.주문번호, b.업체번호, b.주문일자, b.주문금액
from (
  select  업체번호, max(주문번호) 마지막주문번호
  from    주문
  where   주문일자 like '201509%'
  group by 업체번호
) a, 주문 b
where b.업체번호 = a.업체번호
and   b.주문번호 = a.마지막주문번호
```

실기문제 4

주문 테이블 구조는 아래와 같으며, 파티셔닝하지 않았다.

아 래

COLUMN NAME	IS_PK	Nullable	DATA TYPE
주문번호	Y	NO	NUMBER
고객번호		NO	NUMBER
주문일시		NO	DATE
주문금액		NO	NUMBER
우편번호		NO	VARCHAR2(6)
배송지		NO	VARCHAR2(100)
연락처		YES	VARCHAR2(14)
메모		YES	VARCHAR2(100)

하루 주문 건수는 평균 2만 건이며, 10년치 데이터가 저장돼 있다.

주문 데이터를 조회하는 화면은 아래와 같다. 고객번호는 입력하지 않을 수 있지만, 주문일자는 항상 입력해야 한다. 주문일자로는 보통 3일을 입력하며, 최대 1주일까지 입력할 수 있다.


① 조회 버튼을 누를 때 수행할 최적의 SQL을 작성하시오.

개발 정책 상, Dynamic SQL은 사용할 수 없다. 주문일시 기준 역순으로 정렬해야 하며, 부분범위처리는 허용되지 않는다. 즉, 조회된 결과 집합 전체를 그리드(Grid)에 출력해야 한다.

② 최적의 인덱스 구성안을 제시하시오.

아 래

아 래



```

graph LR
    A[고객] --> B[고객접속이력]
    style A fill:#fff,stroke:#333,stroke-width:1px
    style B fill:#fff,stroke:#333,stroke-width:1px
  
```

고객

- # 고객번호
- * 고객명
- * 등록일시
- * 고객상태코드
- o 연락처
- o 주소

고객접속이력

- # 접속일시
- # 고객번호
- * 접속경로
- *
- o
- o

두 조회 버튼에 대한 ① 최적의 SQL을 각각 작성하고, ② 최적의 인덱스 구성안을 제시하시오.

아 래

【 요 건 】

1. 각 조회버튼에 대한 성능요건

- ① 조회/다음 : 응답속도(Response Time)를 빠르게 튜닝하는 것이 가장 중요
- ② 파일로 출력 : 전체 처리속도와 시스템 리소스 사용량을 최소화하는 것이 가장 중요

2. (조회/다음 버튼 클릭 시) 화면 페이징 처리 요건

- ① 조회/다음 버튼을 계속 눌러 뒤쪽 페이지로 많이 이동하는 경우가 간혹 있지만, 대개 3페이지 이내만 조회하고 멈추는 업무임(→ 페이지마다 인덱스 스캔 시작점을 찾기 위해 UNION ALL 방식으로 복잡하게 구현하지 않아도 된다는 의미임)
- ② 페이징 방식으로 조회하는 동안 새로운 데이터가 등록되거나 기존 데이터가 삭제되는 경우를 고려하지 않아도 됨
- ③ 향후에 혹시 인덱스 구성이 변경되더라도 결과집합은 정확히 보장되도록 구현해야 함

3. View Merging, Join Predicate Pushdown 등 Query Transformation이 작동하지 않는 DBMS 버전을 사용 중임

4. 인덱스 설계 시, 성능에 도움이 안 되는 컬럼을 추가하면 오히려 감점이 될 수 있으므로 주의

5. 병렬처리 불가

【 데이터 분포 및 테이블 구성 】

고객 테이블

- 비파티션
- 총 고객수 = 10만명
- 고객상태코드 'AC'인 고객수 = 2만명

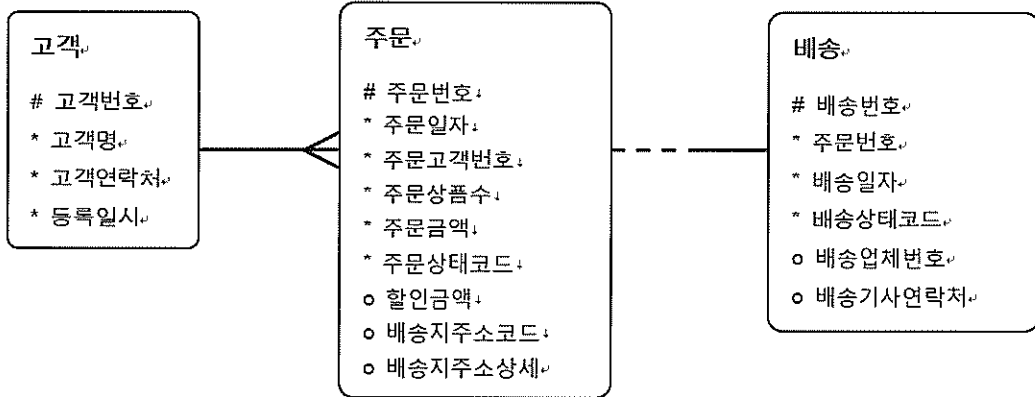
고객접속이력 테이블

- 총 데이터 건수 : 1,000만건
- 접속일시 기준 월단위 Range 파티션
- 고객접속이력_PK은 Local Partitioned Index

실기문제 6

주문, 배송, 고객 정보를 읽어 주문배송 테이블에 입력하는 야간 배치(Batch) 프로그램을 튜닝하려고 한다. 대상 주문 데이터는 2016년 6월부터 8월까지 3개월치다. 월별 주문건수는 1,000만 건이다. 월별 배송건수는 900만 건이다. 배송은 주문이 완료된 후에 시작된다. 고객 수는 500만 명이다.

아 래



【 파티션 구성 】

주문 : 주문일자 기준 월단위 Range 파티션

배송 : 배송일자 기준 월단위 Range 파티션

【 인덱스 구성 】

주문 테이블

주문_PK : 주문번호

주문_N1 : 주문상태코드 + 주문일자 → Local Partition

주문_N2 : 주문고객번호 + 주문일자 → Local Partition

배송 테이블

배송_PK : 배송번호

배송_N1 : 주문번호 + 배송일자 → Local Partition

배송_N2 : 배송일자 + 배송상태코드 → Local Partition

고객 테이블

고객_PK : 고객번호

고객_N1 : 고객명 + 고객번호

아래 병렬 SQL과 예상실행계획을 분석해 가장 빠르게 수행할 수 있도록 SQL을 재작성하시오.

- 옵티마이저 힌트 변경이 필요하다면 SQL문장에 정확히 기술
- 야간 배치용 SQL이므로 다른 트랜잭션에 의한 동시 DML 없음
- 세션 파라미터 변경이 필요하다면, 설정 값 제시
- 인덱스 구성 변경이 필요하다면, 변경안 제시
- 파티션 구성은 변경 불가
- 시스템 운영 정책상 허용된 최대 Parallel Degree = 4

아 래

```
insert into 주문배송 t
select /*+ leading(o) use_nl(d) index(d) full(o) parallel(o 4) */
      o.주문번호, o.주문일자, o.주문상품수, o.주문상태코드, o.주문고객번호
    , (select 고객명 from 고객 where 고객번호 = o.주문고객번호) 고객명
    , d.배송번호, d.배송일자, d.배송상태코드, d.배송업체번호, d.배송기사연락처
from    주문 o, 배송 d
where   o.주문일자 between '20160601' and '20160831'
and     o.주문번호 = d.주문번호
```

Id	Operation	Name	Rows	Pstart	Pstop	TQ	IN-OUT
0	INSERT STATEMENT		30M				
1	LOAD TABLE CONVENTIONAL	주문배송					
2	TABLE ACCESS BY INDEX ROWID	고객	1				
* 3	INDEX UNIQUE SCAN	고객_PK	1				
4	PX COORDINATOR						
5	PX SEND QC (RANDOM)	:TQ10000	30M			Q1,00	P->S
6	NESTED LOOPS		30M			Q1,00	PCWP
7	NESTED LOOPS		30M			Q1,00	PCWP
8	PX BLOCK ITERATOR		30M	62	64	Q1,00	PCWC
* 9	TABLE ACCESS FULL	주문	30M	62	64	Q1,00	PCWP
10	PARTITION RANGE ALL		1	1	65	Q1,00	PCWP
* 11	INDEX RANGE SCAN	배송_N1	1	1	65	Q1,00	PCWP
12	TABLE ACCESS BY LOCAL INDEX ROWID	배송	1	1	1	Q1,00	PCWP

Predicate Information (identified by operation id):

- 3 - access("고객번호"=:B1)
- 9 - filter("O"."주문일자"<='20160831')
- 11 - access("O"."주문번호"="D"."주문번호")

