**1. 다음 중 아래 스키마를 보고 평점이 3.0 이상인 학생들의 이름을 보여주는 SQL을 표현한 것으로 가장 적절한 것은?**

※ 밑줄친 속성은 기본키(Primary Key)이며 ENROLL의 학번은 STUDENT의 학번을 참조하는 외래키(Foreign Key)이고, ENROLL의 강좌번호는 CLASS의 강좌번호를 참조하는 외래키이다.

STUDENT(학번, 학과, 이름) <- (스키마)

CLASS(강좌번호, 시간, 강좌이름)

ENROLL(학번, 강좌번호, 학점)

------------------------------------------------------------------------------------------------------

① SELECT 학번, MIN(이름)③ SELECT S.학번, MIN(이름)

FROM STUDENT S, ENROLL EFROM STUDENT S, ENROLL E

GROUP BY E.학번WHERE S.학번=E.학번

HAVING AVG(E.학점) >= 3.0GROUP BY S.학번

HAVING AVG(E.학점) >= 3.0

② SELECT S.학번, MIN(이름)④ SELECT 학번, MIN(이름)

FROM STUDENT S, ENROLL EFROM STUDENT S, ENROLL E

WHERE S.학번=E.학번WHERE S.학번=E.학번

HAVING AVG(E.학??글>GROUP BY S.학번

HAVING AVG(E.학점) > 3.0

정답 ③

GROUP BY 절 이후에 조건을 적용하는 경우는 HAVING 절이 사용된 경우이다.

그룹핑된 값에 조건을 적용하는 경우 HAVING 절을 사용한다.

JOIN 절에 ALIAS를 사용한 경우, 2개 이상의 테이블에 공통적으로 사용되고 SELECT 절에 사용되는 칼럼에는 ALIAS 접두사를 붙여야 한다.





정답 : ④ FULL OUTER JOIN은 DEFT와 EMP 사이에서 조인이 성공한 건과 DEPT, EMP 건 중 조인이 실패한 건을 함께 표시한다.

**3. 어느 회사의 사원들과 이들이 부양하는 가족에 대한 것으로 밑줄 친 칼럼은 기본키(Primary Key)를 표시한 것이다. 다음 중 ‘현재 부양하는 가족들이 없는 사원들의 이름을 구하라’는 질의에 대해 아래 SQL 문장의 ㉠, ㉡ 에 들어 갈 내용으로 가장 적절한 것은?**

사원 (사번, 이름, 나이)

가족 (이름, 나이, 부양사번)

※ 가족 테이블의 부양사번은 사원 테이블의 사번을 참조하는 외래키(Foreign Key)이다.

[SQL 문장]

SELECT 이름

FROM 사원

WHERE ㉠ (SELECT \* FROM 가족 WHERE ㉡ )

------------------------------------------------------------------------------------------------------

① ㉠ : EXISTS ㉡ : 사번 = 부양사번

② ㉠ : EXISTS ㉡ : 사번 <> 부양사번

③ ㉠ : NOT EXISTS ㉡ : 사번 = 부양사번

④ ㉠ : NOT EXISTS ㉡ : 사번 <> 부양사번

정답 : ③ ‘가족들이 없는’조건 : NOT EXISTS, ‘현재 부양하는 가족들’조건 : 사번 = 부양사번

**4. 다음 중 아래 테이블 정의와 인덱스 현황을 참고하여, 인덱스를 효율적(또는 정상적)으로 액세스할 수 없는 검색조건을 2개 고르시오.**

**(단, Oracle의 Index Unique Scan, Index Range Scan 또는 SQL Server의 Index Seek 이외의 액세스 방식은 모두 비효율적이라고 가정한다.)**

create table 주문 (

주문번호 int not null

, 주문자명 varchar(20) null

, 주문금액 money null

, 주문일자 varchar(8) null

)

create unique index 주문\_pk on 주문 (주문번호)

create index 주문\_x01 on 주문 (주문자명)

create index 주문\_x02 on 주문 (주문일자 , 주문금액)

------------------------------------------------------------------------------------------------------

① where 주문번호 between 1 and 10

② where 주문자명 like '%홍길동%'

③ where 주문일자 >= '20100901'

④ where 주문일자 = 20100901

정답 : ② , ④

2번은 LIKE 검색 문자열 앞뒤에 모두 ‘%’ 기호를 붙였으므로 정상적인 Index Range Scan이 불가능하다. 4번은 내부적 형변환이 발생하므로 Index Range Scan이 불가능하다.

**5. 총 건수가 1,000만 건인 연도별지역별상품매출 테이블에 [출시연도 + 크기], [색상 + 출시연도] 순으로 구성된 두 개의 B\*Tree인덱스가 있었다. 다음 중 이 두 인덱스를 제거하고 아래와 같이 세 개의 비트맵(Bitmap) 인덱스를 생성했을 때, 개별 쿼리의 블록 I/O 측면에서 개선 효과가 가장 미미한 것은?**

create bitmap index 연도별지역별상품매출\_bx1 on 연도별지역별상품매출(크기);

create bitmap index 연도별지역별상품매출\_bx2 on 연도별지역별상품매출(색상);

create bitmap index 연도별지역별상품매출\_bx3 on 연도별지역별상품매출(출시연도);

■ Distinct Value

크기 = { NULL, SMALL, MEDIUM, BIG }

색상 = { NULL, RED, GREEN, BLUE, BLACK, WHITE }

출시연도 = { NULL, 2001, 2002, 2003, 2004, 2005 2006, 2007, 2008, 2009, 2010 }

■ 데이터 분포는 모두 균일

------------------------------------------------------------------------------------------------------

① select count(\*) from 연도별지역별상품매출 where 색상 is null;

② select count(\*) from 연도별지역별상품매출

where (크기 = ‘SMALL’ or 크기 is null)

and 색상 = ‘GREEN’

and 출시연도 = ‘2010’;

③ select 색상, count(\*) from 연도별지역별상품매출 group by 색상;

④ select sum(판매량), sum(판매금액)

from 연도별지역별상품매출

where 색상 = ‘BLUE’;

정답 : ④

1번과 3번은 기존에 Full Table Scan으로 처리됐을 것이므로 비트맵 인덱스를 생성하고 나면 블록 I/O가 크게 감소한다. 3번의 경우 만약 ‘색상is not null’ 조건을 추가하면 인덱스만 읽고 처리할 수 있지만 그렇더라도 B\*Tree 인덱스는 비트맵 인덱스에 비해 블록 I/O는 더 많이 발생한다.

2번은 Bitmap Conversion이 발생하지 않는 한, 기존에 두 B\*Tree 인덱스 중 어느 하나만 사용되고, 나머지 필터 조건을 처리하기 위해 테이블 랜덤 액세스가 불가피하므로 성능이 매우 안 좋았을 것이다. 반면, 비트맵 인덱스를 생성하고 나면 두 개의 비트맵 인덱스를 동시에 사용할 수 있고, 테이블을 랜덤 액세스도 생략되므로 성능 개선 효과가 클 것이다.

4번은 색상 = ‘BLUE’에 해당하는 건수만큼 대량의 테이블 액세스가 불가피하다. 비트맵 인덱스를 생성하고 나면 인덱스 스캔 단계에서 다소 블록 I/O가 감소하겠지만 테이블 랜덤 I/O는 줄지 않으므로 성능 개선 효과가 미미하게 나타난다.

**6. 다음 중 Syntax 오류가 있는 SQL문 2개를 고르시오.**

① SELECT 부서번호

FROM 직원HAVING COUNT(\*) > 3;

② SELECT 직원이름

FROM 직원 JOIN 부서

ON (직원.부서번호 = 부서.부서번호)

WHERE 부서명 = ‘인사과’

③ SELECT 전화번호

FROM 직원

WHERE 전화번호 = ‘777’

GROUP BY 전화번호

④ SELECT 직원이름

FROM 직원 JOIN 부서

USING (부서.부서번호)

WHERE 부서명 = ‘인사과’

정답 : ①, ④

① GROUP BY 절 없이 HAVING을 사용할 수 없다.

④ USING 조건절을 이용한 EQUI JOIN에서도 NATURAL JOIN과 마찬가지로 JOIN 칼럼에 대해서는 ALIAS나 테이블 이름과 같은 접두사를 붙일 수 없다.

(부서.부서번호 → 부서번호)

**7. 다음 중각 SQL 실행 결과를 가장 올바르게 설명한 것을 2개 고르시오.**

① SELECT COL1, COL2 FROM TAB1 WHERE 1 = 2 ;

→ 실행 시 에러가 발생한다.

② SELECT NVL(COL1,'X') FROM TAB1 WHERE 1 = 2 ;

→ 실행 결과로 'X'를 반환한다.

③ SELECT NVL(MIN(COL1), 'X') FROM TAB1 WHERE 1 = 2;

→ 실행 결과로 'X'를 반환한다.

④ SELECT COL1, COL2 FROM TAB1 WHERE 1 = 2 ;

→ 실행 결과가 없다.(공집합)

정답 : ③ , ④

조건절에 해당하는 결과집합이 없다고 에러가 발생하지는 않는다. 그리고 공집합에 NVL 함수를 사용한다고 값이 얻을 수 있는 것은 아니다.

**8. 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?.**

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’)

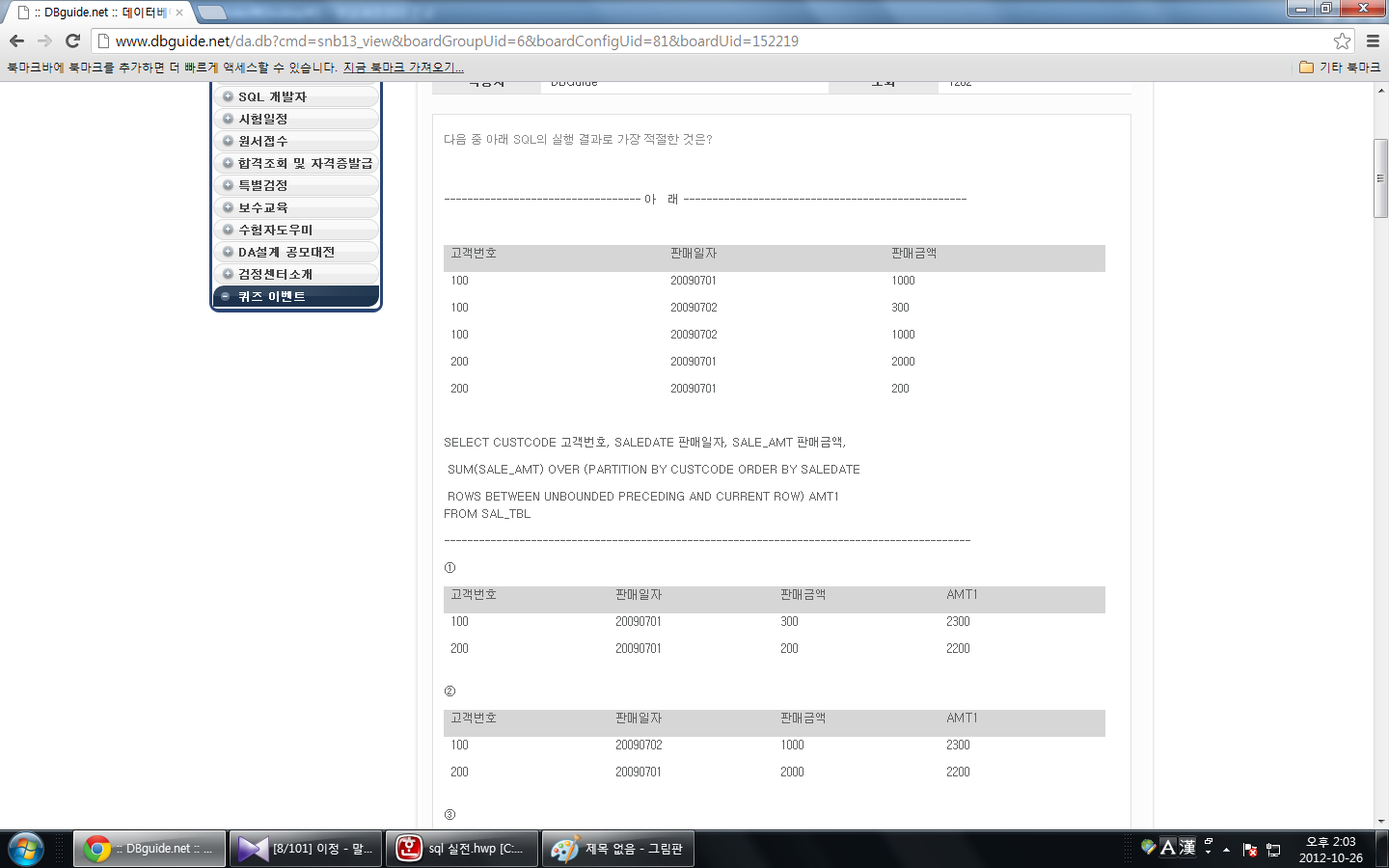
② ROWNUM

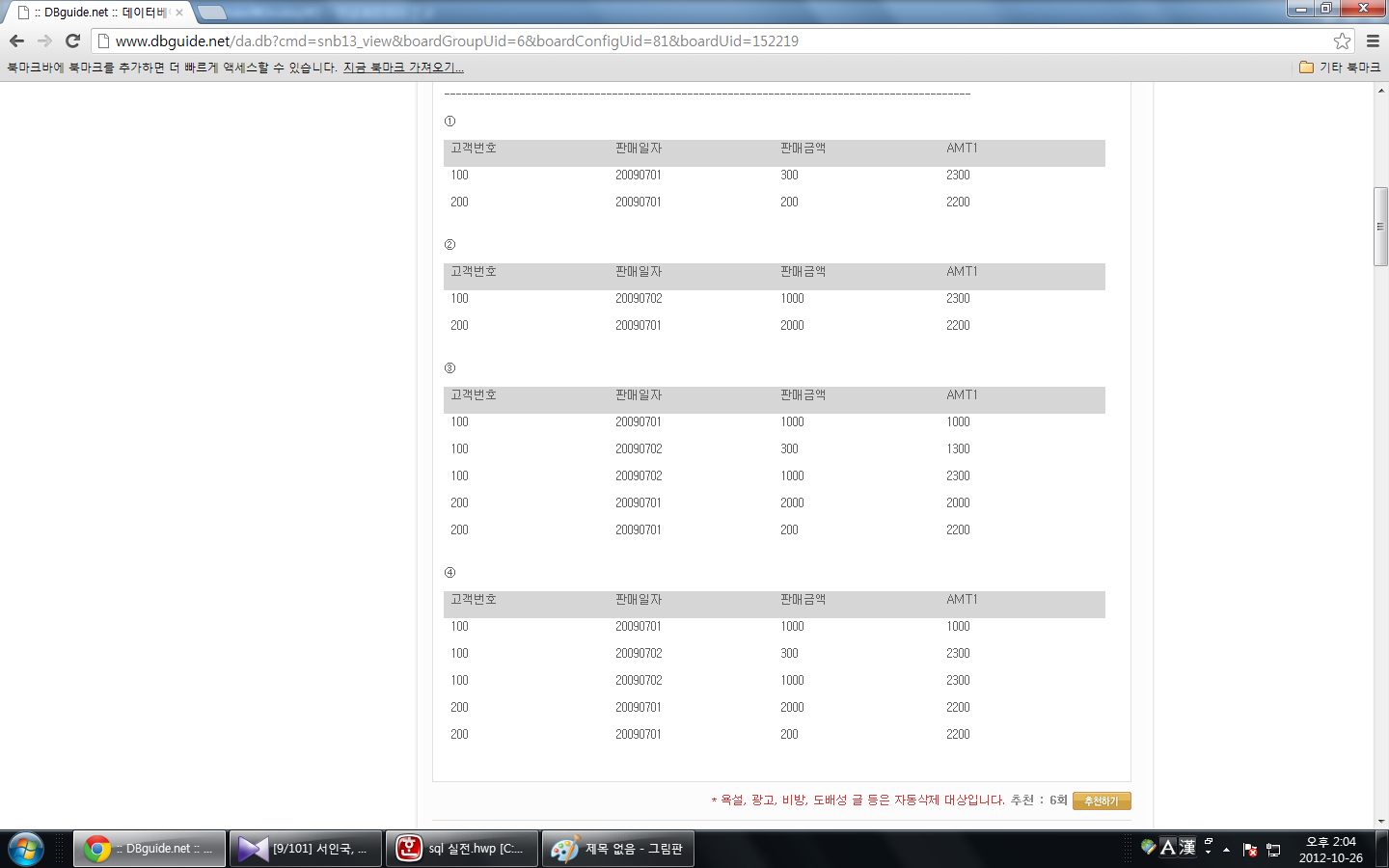
③ EMPNO

④ ROWID

정답 : ② 뷰(View) 안에 rownum을 사용하면 뷰 머징(View Merging)을 방지하는 효과가 나타난다.

**9.**





정답 :③ 윈도우 함수를 이용한 누적 합계(RUNNING SUMMARY)를 구하는 SQL이다.

**10. 다음 중 아래 SQL에 대한 설명으로 가장 부적절한 것은?**

SELECT SUM(NVL(A.C2, 0))

FROM TAB1 A, TAB2 B

WHERE A.C1 = :V1

------------------------------------------------------------------------------------------------------

① 조인 조건이 없어서 결과 건수가 여러 건이 된다.

② 조인 조건이 없다고 문법 오류가 발생하지는 않는다.

③ SUM(NVL(A.C2, 0))의 처리에 비효율이 존재한다.

④ 조인 조건이 없어서 CARTESIAN PRODUCT이 발생한다.

정답 :①

GROUP BY절이 없기 때문에 결과건수는 항상 1건이다.

해당 SQL문에서는 조인 조건의 부재로 결과 건수가 아니라 답 자체가 틀려질 수 있다. 즉, Cartesian Product으로 인해 합계가 틀려질 수 있다

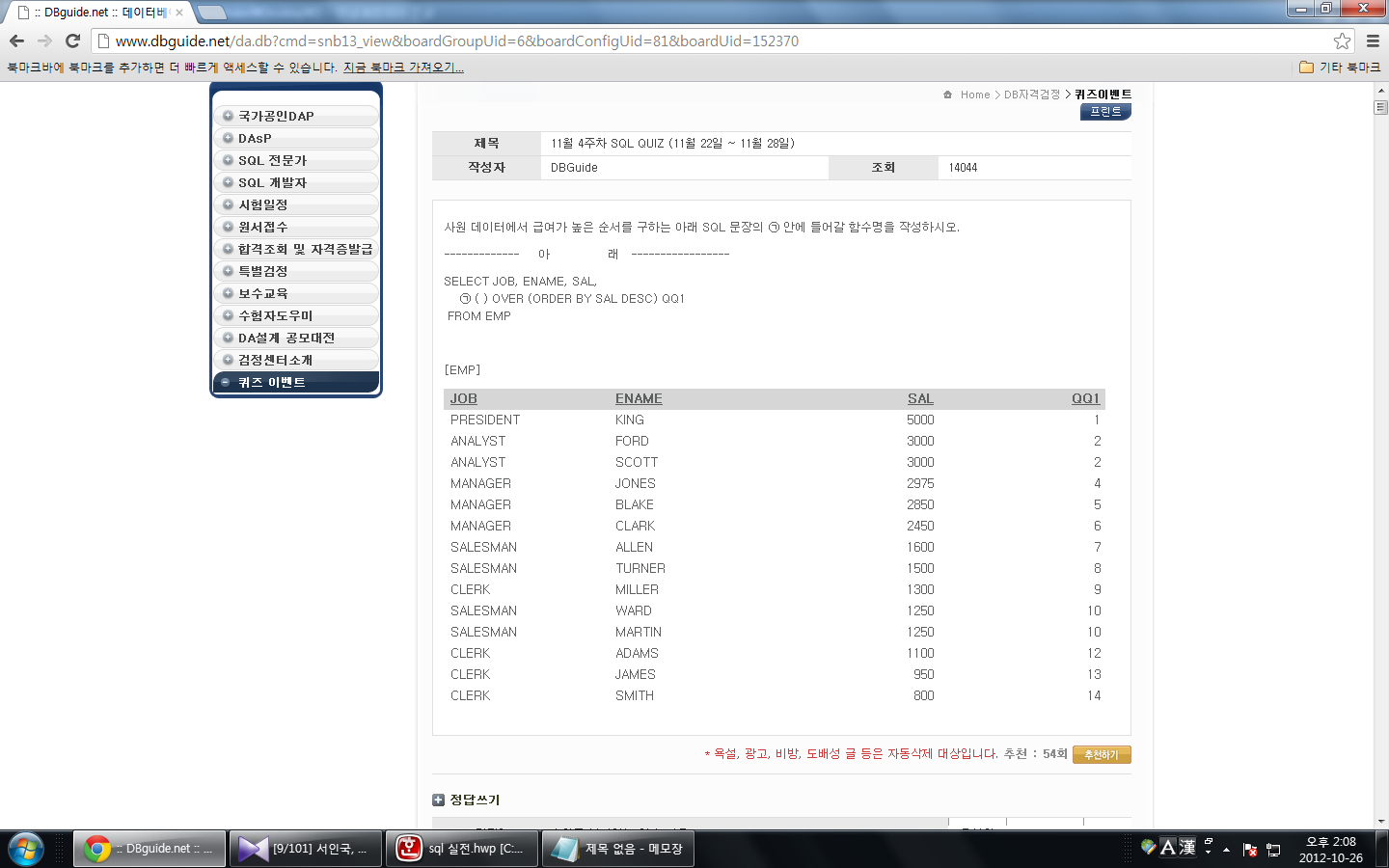
**11. (주관식) 사원 데이터에서 급여가 높은 순서를 구하는 아래 SQL 문장의 ㉠ 안에 들어갈 함수명을 작성하시오.**

------------------------------------------------------------------------------------------------------

SELECT JOB, ENAME, SAL,

㉠ ( ) OVER (ORDER BY SAL DESC) QQ1

FROM EMP



정답 : RANK

RANK 함수는 ORDER BY를 포함한 QUERY 문에서 특정 항목(칼럼)에 대한 순위를 구하는 함수이다. 이때 특정 범위(PARTITION) 내에서 순위를 구할 수도 있고 전체 데이터에 대한 순위를 구할 수도 있다. 또한 동일한 값에 대해서는 동일한 순위를 부여하게 된다.

**12. 전체 사원이 1,000명인 회사에서 아래 쿼리를 수행하려고 한다. 다음 중 옵티마이저가 예상하는 건수로 가장 적절한 것은?**

select \* from 사원 where 직급 = '부장' and 부서 = '영업';

\* 직급은 { 부장, 과장, 대리, 사원 }의 집합이다.

\* 부서는 { 인사, 총무, 회계, 마케팅, 영업 }의 집합이다.

\* 두 칼럼 모두 히스토그램 정보가 수집되어 있지 않다.

① 48 ② 50 ③ 55 ④ 60

정답 : ② 50 (직급의 Selectivity) × (부서의 Selectivity) × (전체 로우 수) = 1/4 × 1/5 × 1,000 = 50

**13. 다음 중 아래 t\_idx 인덱스가 해당하는 것으로 가장 적절한 것은?**

create table t ( a number, b char(3), c varchar2(10) )

partition by range(a) (

partition p1 values less than(100)

, partition p2 values less than(200)

, partition p3 values less than(maxvalue)

);

create index t\_idx on t( b ) local ;

------------------------------------------------------------------------------------------------------

① Global Prefixed Partition Index③ Local Prefixed Partition Index

② Global NonPrefixed Partition Index④ Local NonPrefixed Partition Index

정답 : ④ Local NonPrefixed Partition Index

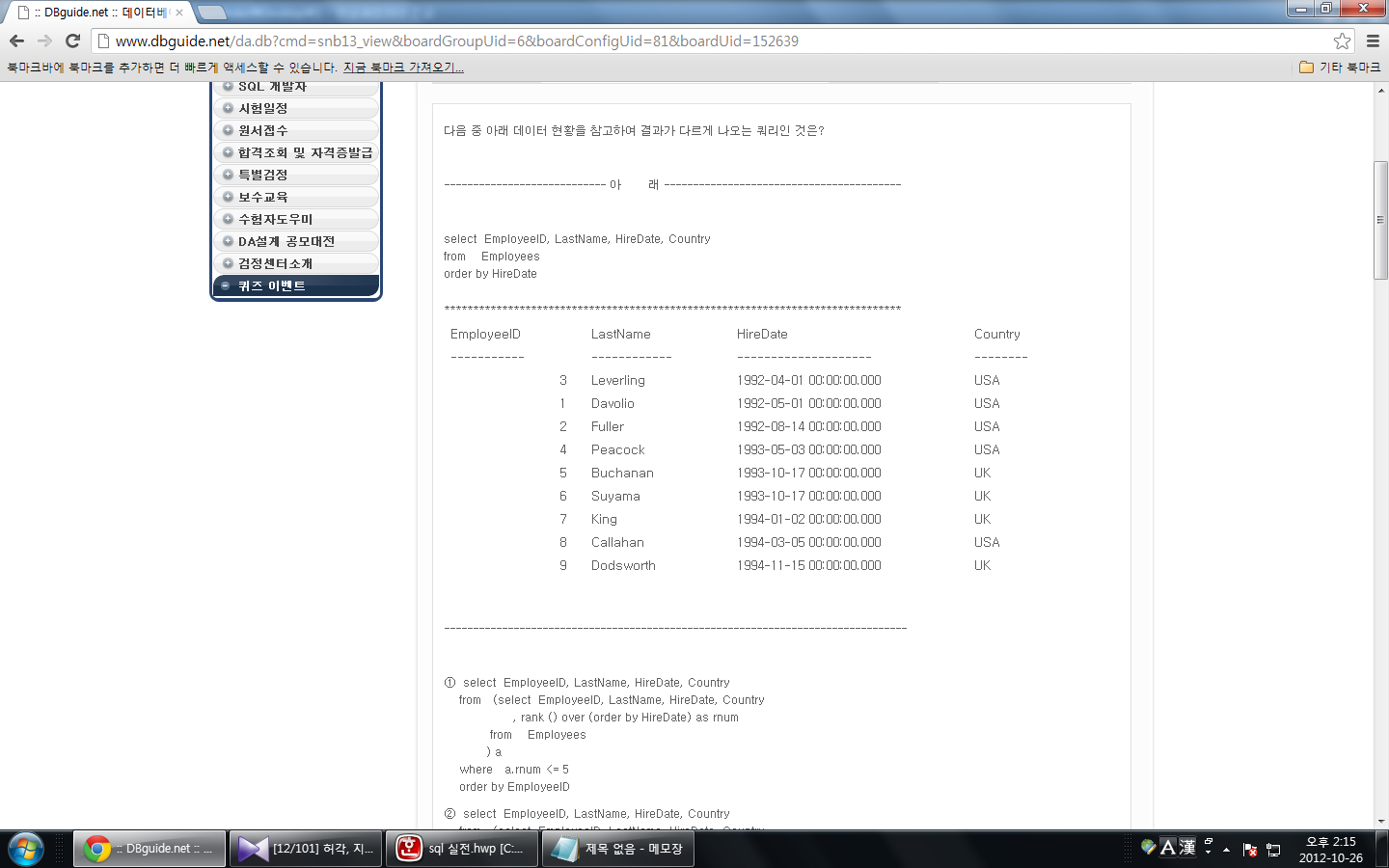
Local 인덱스이므로 t\_idx의 파티션 키는 테이블과 똑같이 a 칼럼이다. 그리고 파티션 키가 인덱스 선두 칼럼이 아니므로 NonPrefixed에 해당한다.

**14. 다음 중 아래 데이터 현황을 참고하여 결과가 다르게 나오는 쿼리인 것은?**

select EmployeeID, LastName, HireDate, Country

from Employees

order by HireDate



① select EmployeeID, LastName, HireDate, Country

from (select EmployeeID, LastName, HireDate, Country

, rank () over (order by HireDate) as rnum

from Employees

) a

where a.rnum <= 5

order by EmployeeID

② select EmployeeID, LastName, HireDate, Country

from (select EmployeeID, LastName, HireDate, Country

, row\_number() over (order by HireDate) as rnum

from Employees

) a

where a.rnum <= 5

order by EmployeeID

③ select EmployeeID, LastName, HireDate, Country

from Employees a

where HireDate in (select top 5 HireDate

from Employees

order by HireDate )

order by EmployeeID

④ select EmployeeID, LastName, HireDate, Country

from (select EmployeeID, LastName, HireDate, Country

, dense\_rank() over (order by HireDate) as rnum

from Employees

) a

where a.rnum <= 5

order by EmployeeID

정답 : ②

1, 4번은 5번째와 6번째 레코드가 rnum으로 똑같이 5를 부여받기 때문에 HireDate 순으로 6개 레코드가 선택된다. 참고로, 7번째 레코드는 각각 7과 6을 부여받는다

3번은 서브쿼리에서 5개 레코드가 선택되지만, 메인쿼리와 조인하고 나면 최종적으로 6개 레코드가 선택된다.

2번은 HireDate 순으로 5개 레코드가 선택된다

**15. 아래 NVL 함수와 ISNULL 함수를 사용한 SQL 문장은 벤더 공통적으로 CASE 문장으로 표현할 수 있다. 아래 CASE SQL 문장의 ㉠안에 들어갈 내용을 작성하시오.**

[Oracle 사례]

SELECT PLAYER\_NAME 선수명, POSITION, NVL(POSITION,'없음') 포지션

FROM PLAYER

WHERE TEAM\_ID = 'K08'

[SQL Server 사례]

SELECT PLAYER\_NAME 선수명, POSITION, ISNULL(POSITION,'없음') 포지션

FROM PLAYER

WHERE TEAM\_ID = 'K08'

[CASE 문장 사례]

SELECT PLAYER\_NAME 선수명, POSITION,

CASE WHEN ( ㉠ )

THEN '없음'

ELSE POSITION

END AS 포지션

FROM PLAYER

WHERE TEAM\_ID = 'K08'

정답 : POSITION IS NULL

아래 NVL 함수와 ISNULL 함수를 사용한 SQL 문장은 벤더 공통적으로 CASE 문장으로 표현할 수 있다.

본 문제는 CASE 표현 중 SEARCHED\_CASE\_EXPRESSION에 들어갈 조건을 문의한 것임

CASE

WHEN CONDITION THEN RETURN\_EXPR

ELSE 표현절

END

**16. 다음 중 아래 두 SQL에 대한 설명으로 가장 적절한 것은?**

**(단, 국가는 100건, 수출실적은 100만 건. 국가 테이블 PK는 국가코드이다.)**

가. SELECT DISTINCT 국가명

FROM 국가 x, 수출실적 y

WHERE x.국가코드= y.국가코드

AND y.수출년월 BETWEEN '200001' AND '201012';

나. SELECT 국가명

FROM 국가 x

WHERE EXISTS (SELECT 1

FROM 수출실적 y

WHERE y.국가코드= x.국가코드

AND y.수출년월 BETWEEN '200001' AND '201012');

------------------------------------------------------------------------------------------------------

① ‘가’ SQL은 부분범위 처리가 가능하다.

② ‘나’ SQL이 더 효율적이다.

③ OLTP 환경이냐, DW 환경이냐에 따라 두 SQL의 효율성이 다르다.

④ 두 SQL의 결과가 다르므로 효율성을 판단하는 것은 의미가 없다.

답 : ②

두 SQL의 결과는 동일하며, OLTP 환경이냐 DW 환경이냐를 불문하고 두 번째 SQL이 더 효율적이다.

**17. (주관식) 아래 Call Statistics를 보고 Buffer Cache Hit Ratio를 구하시오.**

call count cpu elapsed disk query current rows

------ ------- ------ --------- ----- -------- -------- -------

Parse 50000 0.51 0.51 0 0 0 0

Execute 50000 3.18 2.91 5000 10000 15000 50000

Fetch 0 0.00 0.00 0 0 0 0

------ ------- ------ --------- ------ -------- --------- -------

total 100000 3.70 3.43 5000 10000 15000 50000

정답 : 80 Buffer Cache Hit Ratio = 100 × (1 - disk / (query+current))

**18. 다음 중 테이블 tab1에 아래와 같은 데이터가 존재할 경우에 대한 설명으로 가장 적절한 것은? (각 칼럼의 타입은 number이다.)**

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.

정답 : ③

칼럼끼리 연산할 때 null을 포함하면 결과는 null이다.

레코드끼리 연산할 때 null을 포함하면 결과가 null이 아니며, 이유는 null을 연산에서 제외하기 때문이다.

**19. 다음 중 NL(Nested Loops) Join의 특징을 설명한 것으로 가장 부적절한 것은?**

① 조인을 한 로우씩 차례대로 진행하므로 부분범위 처리에 자주 사용된다.

② 먼저 액세스한 테이블의 처리 범위에 따라 전체 일량이 결정된다.

③ Inner쪽 조인 칼럼에 대한 인덱스 전략이 중요하지만, 조인 칼럼이 모두 포함되지 않은 인덱스라도 사용 가능하다.

④ 선행(Driving) 집합도 반드시 인덱스를 통해 액세스해야 효율적이다.

정답 : ④

선행(Driving) 집합은 주어진 조건절에 따라 Full Table Scan이 유리할 수도 있다.

**20. 다음 중 Oracle에서 인덱스를 스캔하는 방식 중 하나인 Index Full Scan의 특징으로 가장 부적절한 것은?**

① 인덱스 구조를 따라 스캔하므로 결과집합의 순서가 보장된다(=일정하다).

② Single Block Read 방식을 사용한다.

③ 파티션 돼 있지 않다면 병렬 스캔이 불가능하다.

④ 필요한 칼럼이 모두 인덱스에 포함돼 있을 때만 사용 가능하다.

정답 : ④

필요한 칼럼이 모두 인덱스에 포함돼 있을 때만 사용 가능한 것은 Index Fast Full Scan의 특징이다.

**21. Oracle에서 TAB1, TAB2 순으로 NL 조인하도록 유도하고 싶다. 다음 중 ㉠ 안에 넣을 바른 힌트 사용법을 2개고 르시오.**

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)

정답 : ①, ③

테이블 Alias가 있은 상황에선 반드시 Alias를 사용해야 한다.

**22. 다음 중 아래 SQL문과 실행계획을 보고, 두 테이블의 조인 순서와 inner 테이블에 대한 조인 회수로 가장 적절한 것은?**

select \*

from t\_small a, t\_big b

where a.id = b.id

and a.colid = b.colid

and a.number = b.number

and b.name = 'password'

and a.length <= 10

order by

a.length desc

go

Rows Executes StmtText

----- ----------- ---------------------------------------------------------------

0 1 select \*

0 1 |--Sort(ORDER BY:([a].[length] DESC))

0 1 |--Filter(WHERE:([t\_small].[length] as [a].[length]<=(10

30 1 |--Nested Loops(Inner Join, OUTER REFERENCES:([Bmk1

30 1 |--Nested Loops(Inner Join, OUTER REFERENCES:(

30 1 | |--Table Scan(OBJECT:([t\_big] AS [b]),

30 30 | |--Index Seek(OBJECT:([t\_small].[t\_small\_

30 30 |--RID Lookup(OBJECT:([t\_small] AS [a]), SEEK:

(8개 행 적용됨)

-----------------------------------------------------------------------------------

① 조인 순서 : t\_small → t\_big, inner 테이블 조인 횟수 : 30

② 조인 순서 : t\_big → t\_small, inner 테이블 조인 횟수 : 60

③ 조인 순서 : t\_small → t\_big, inner 테이블 조인 횟수 : 1

④ 조인 순서 : t\_big → t\_small, inner 테이블 조인 횟수 : 30

정답 : ④

실행계획 상 위 쪽에서 아래 쪽으로 조인이 진행된다.

NL 조인의 경우 위쪽에 있는 Outer 집합에서 출력된 결과 건수(Rows)만큼 Inner 집합으로 조인 시도가 일어난다.

**23. 다음 중 아래 SQL과 트레이스 결과를 분석하고, 보기에서 제시한 튜닝 방안 중 가장 적절한 것은?**

SQL> SELECT ...

2 FROM (SELECT DEII\_DT, NEII\_VAL, NACT\_VAL, NSTD\_VAL

3 , NVL(ROUND(AVG(NACT\_VAL/NSTD\_VAL\*100) OVER(), 2), 0.00) AVG\_VAL

4 , NVL(ROUND(STDDEV(NACT\_VAL/NSTD\_VAL\*100) OVER(), 2), 0.00) STDDEV\_VAL

5 , ROWNUM RN

6 FROM EII\_SUMMARY

7 WHERE TO\_CHAR(DEII\_DT, 'YYYYMMDDHH24MISS') BETWEEN :B3 AND :B4

8 ORDER BY DEII\_DT) MR

9 LEFT OUTER JOIN EII\_TARGET ET

10 ON ET.DEII\_DT BETWEEN TRUNC(MR.DEII\_DT, 'MM')

11 AND TRUNC(LAST\_DAY(MR.DEII\_DT)+1)-1/60/24/24

12 AND ET.NCODE\_NO IN ( :B1, :B2 )

13 GROUP BY FLOOR((MR.RN-1)/:B5 ), MR.AVG\_VAL, MR.STDDEV\_VAL

14 ORDER BY FLOOR((MR.RN-1)/:B5 ) ;

call count cpu elapsed disk query current rows

------- ------ ------- -------- ------ ------- -------- -------

Parse 446 0.00 0.00 0 0 0 0

Execute 7578 0.03 0.12 22 564 2 8

Fetch 13522 128.03 129.16 22 6676902 0 10442

------- ------ ------- -------- ------ ------- -------- -------

total 21546 128.06 129.30 44 6677466 2 10450

Rows Row Source Operation

---- ---------------------------------------------------

2 SORT GROUP BY (cr=7340 pr=7 pw=0 time=221191 us)

240 NESTED LOOPS OUTER (cr=7340 pr=7 pw=0 time=221460 us)

120 VIEW (cr=20 pr=7 pw=0 time=90776 us)

120 FILTER (cr=20 pr=7 pw=0 time=90410 us)

120 WINDOW SORT (cr=20 pr=7 pw=0 time=89447 us)

120 COUNT (cr=20 pr=7 pw=0 time=79490 us)

120 TABLE ACCESS (FULL) OF 'EII\_SUMMARY' (TABLE) (cr=20 pr=7 pw=0 time=79250 us)

240 VIEW (cr=7320 pr=0 pw=0 time=74760 us)

240 TABLE ACCESS FULL EII\_TARGET (cr=7320 pr=0 pw=0 time=74414 us)

[인덱스 구성]

EII\_SUMMARY\_X01 : DEII\_DT

EII\_TARGET\_X01 : DEII\_DT

------------------------------------------------------------------------------------------

① 7번 라인을 아래와 같이 수정한다. WHERE DEII\_DT BETWEEN TO\_DATE(:B3, 'YYYYMMDDHH24MISS')

AND TO\_DATE(:B4, 'YYYYMMDDHH24MISS')

② 9번 라인의 'LEFT OUTER JOIN'을 'INNER JOIN'으로 변경함으로써 EII\_TARGET 테이블이 먼저 드라이빙 될 수 있게 한다. Row Source를 분석해 보면, Outer 집합에서 Inner 집합으로 조인 시도한 건수만큼 모두 성공하므로 Outer Join은 불필요하다.

③ 10~12번 라인을 위해 EII\_TARGET\_X01 인덱스를 [NCODE\_NO + DEII\_DT] 순으로 구성한다.

④ 14번 라인의 ORDER BY는 불필요하므로 제거한다.

정답 : ③

7번 라인에 대한 Row Source를 보면, 20개 블록을 읽어서 120개 로우를 반환하므로 굳이 인덱스를 사용하도록 튜닝하지 않아도 된다.

9번 라인에 대한 Row Source만 보고 Left Outer Join이 불필요하다고 판단할 수 없다.

10~11번 라인 조인 칼럼에 인덱스가 있는데도 옵티마이저가 이를 사용하지 않고 Full Table Scan으로 처리한 이유는, NCODE\_NO 필터링을 위해 다량의 테이블 랜덤 액세스가 발생하기 때문이다. 인덱스 뒤에 NCODE\_NO만 추가해도 성능이 많이 개선되겠지만, 순서까지 바꿔 [NCODE\_NO + DEII\_DT] 순으로 구성하는 것이 최적이다.

14번 라인의 ORDER BY를 제거하면 결과집합의 출력순서가 달라질 수 있다.

**24. 일별고객별판매집계 테이블의 PK는 [판매일시 + 고객번호] 순으로 구성되었다. 이 테이블에 DML을 수행하는 다른 트랜잭션이 없는 상황에서 Oracle은 아래 ‘가’, ‘나’, ‘다’ 문장을 어떤 순서로 수행하느냐에 따라 마지막 ‘라’ 문장의 블록 I/O 횟수가 달라진다. 제시된 4가지 수행 순서 보기 중 마지막 ‘라’ 문장을 수행했을 때 블록 I/O가 가장 적게 발생하는 것을 고르시오.**

**(‘라’ 문장은 PK 인덱스를 이용한다는 사실을 상기하기 바란다.)**

가) 3일 이전에 발생한 판매 데이터를 삭제한다.

delete from 일별고객별판매집계 where 판매일시 < trunc(sysdate) - 2;

나) 고객별로 집계한 금일 판매 데이터를 추가한다.

insert into 일별고객별판매집계

select to\_char(sysdate, 'yyyymmdd'), 고객번호, sum(판매량), sum(판매금액)

from 판매

where 판매일시 between trunc(sysdate) and trunc(sysdate+1)-1/24/60/60

group by 고객번호;

다) commit;

라) select count(\*) from 일별고객별판매집계;

------------------------------------------------------------------------------------------------------

① 가 -> 나 -> 다 -> 라

② 가 -> 다 -> 나 -> 다 -> 라

③ 나 -> 가 -> 다 -> 라

④ 나 -> 다 -> 가 -> 다 -> 라

정답 : ②

‘가’를 수행하고‘다’를 수행하기 전에 commit을 수행하면, ‘가’에서 삭제된 빈 공간을 ‘다’에서 재사용하므로 Index Skew 현상을 방지할 수 있다.

**25. 다음 중 아래 SQL과 인덱스 구성에 대한 설명으로 가장 부적절한 것을 2개 고르시오.**

------------------------------------- 아 래 ---------------------------------------

SELECT ………………

FROM TAB1

WHERE COL1 = :VAL1

AND COL2 LIKE :VAL2 || '%'

AND COL3 IN ('1','5')

AND COL4 BETWEEN :DATE1 AND :DATE2

■ 인덱스구성 : COL1 + COL2 + COL3

------------------------------------------------------------------------------------------

① COL1 조건이 없었다면 정상적으로 인덱스를 이용할 수 없다.

② COL2 조건은 인덱스 읽는 범위를 줄여주는데 기여하지 못한다.

③ COL3 조건은 인덱스 읽는 범위를 줄여주는데 기여한다.

④ COL4 조건은 테이블에서 필터링된다.

정답 : ② , ③

Like, Between, 부등호 같은 범위검색조건이더라도 선행 칼럼이 누락없이 모두 ‘=’ 조건으로 제공되면 인덱스 스캔 범위를 줄이는데

기여한다.

**26. 다음 중 아래와 같은 테이블에서 SQL의 실행 결과로 가장 적절한 것은?**

주문번호주문일자주문금액부가세주문월

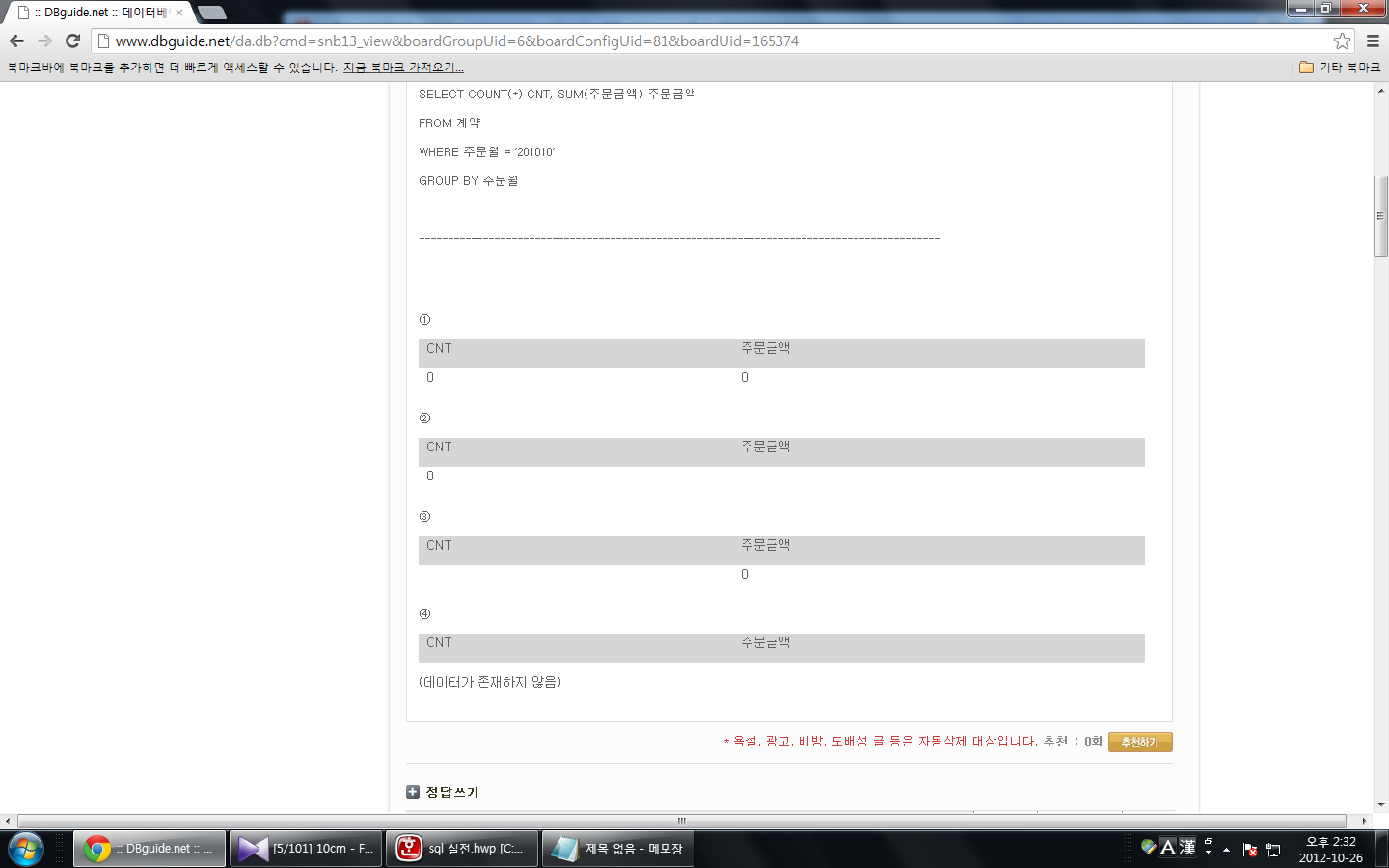
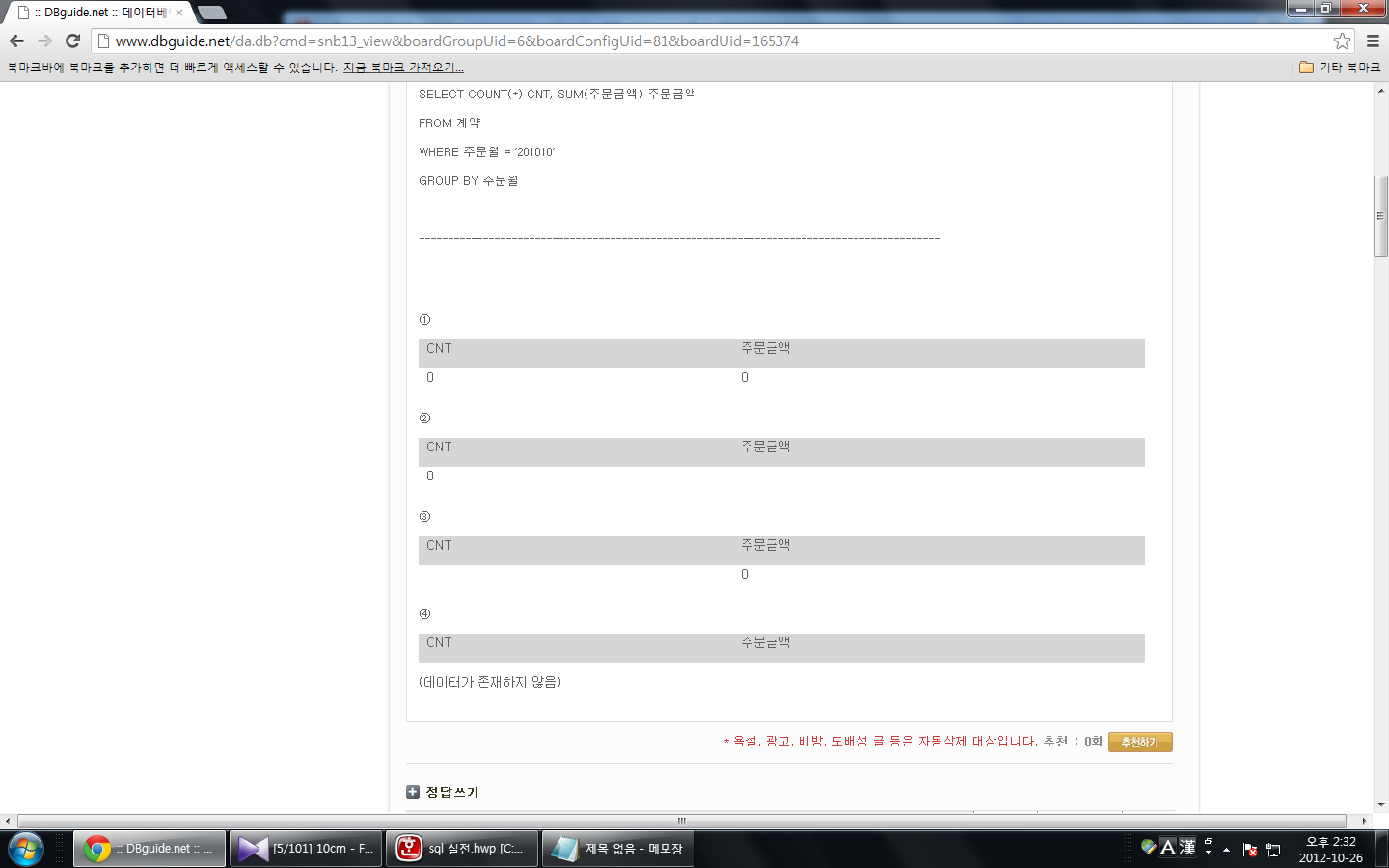
1012010/01/0150050201001

SELECT COUNT(\*) CNT, SUM(주문금액) 주문금액

FROM 계약

WHERE 주문월 = ‘201010’

GROUP BY 주문월



정답 : ④

주어진 조건을 만족하는 건이 없다. 만약 SQL에서 GROUP BY절을 사용하지 않았다면 4번이 정답이다.

**27. 다음 중 아래 두 SQL문에 대한 설명으로 가장 적절한 것은? (단, DEPT : EMP = 1 : M)**

SELECT A.ENAME, A.SAL, B.DNAME

FROM EMP A, DEPT B

WHERE A.DEPTNO = B.DEPTNO

AND B.DEPTNO = 20;

SELECT A.ENAME, A.SAL, B.DNAME

FROM EMP A, DEPT B

WHERE A.DEPTNO = 20

AND B.DEPTNO = 20;

------------------------------------------------------------------------------------------------------

① 두번째 SQL문은 배치쿼리 시 유리하다.

② 두 SQL문의 수행속도는 현저한 차이가 난다.

③ 두 SQL문의 결과는 동일하다.

④ 옵티마이저 모드나 버전에 따라 다른 결과가 나올 수 있다.

정답 : ③

b.deptno = 20이고 a.deptno = b.deptno이므로 a.deptno = 20이다. 따라서 위 두 SQL의 처리 결과는 동일하다

**28. 다음 중 절차형 SQL 모듈에 대한 설명으로 가장 부적절한 것은?**

① 데이터의 무결성과 일관성을 위해서 사용자 정의 함수를 사용한다.

② 사용자 정의 함수는 절차형 SQL을 로직과 함께 데이터베이스 내에 저장해 놓은 명령문의 집합을 의미하며 다른 SQL문장에 포함되어 실행될 수 있다. 단독적으로 실행되기 보다는 다른 SQL문을 통하여 호출되고 그 결과를 리턴하는 SQL의 보조적인 역할을 한다.

③ 트리거는 특정한 테이블에 INSERT, UPDATE, DELETE와 같은 DML문이 수행되었을 때 데이터베이스에서 자동으로 동작하도록 작성된 프로그램이다.

④ 스토어드 프로시져는 절차형 SQL을 로직과 함께 데이터베이스 내에 저장해 놓은 명령문의 집합을 의미한다.

정답 : ①

Stored Module(ex: PL/SQL, LP/SQL, T-SQL)로 구현 가능한 기능은 ②,③,④ 세가지이며, ① 데이터의 무결성과 일관성을 위해서 사용자 정의 함수를 사용하는 것은 트리거의 용도이다.

**29. 다음 중 관계형 데이터베이스의 인덱스(Index)에 대한 설명으로 가장 적절한 것은?**

① 기본 인덱스(Primary Key Index)에 중복된 키 값들이 나타날 수 있다.

② 기본 인덱스에 널 값(Null Value)들이 나타날 수 없다.

③ 보조 인덱스(Secondary Index)에는 고유한 키 값들만 나타날 수 있다.

④ 자주 변경되는 속성은 인덱스를 정의할 좋은 후보이다.

정답 : ②

기본 인덱스 = 기본키 인덱스(PK)는 UNIQUE & NOT NULL의 제약조건을 가진다.

보조 인덱스는 NON UNIQUE 성격을 가질 수 있다.

자주 변경되는 속성은 인덱스로 적절하지 않다.

**30. 다음 중 Oracle 계층형 질의에 대한 설명으로 가장 부적절한 것은?**

① START WITH절은 계층 구조의 시작점을 지정하는 구문이다.

② 루트 노드의 LEVEL 값은 0이다.

③ 순방향전개란 부모 노드로부터 자식 노드 방향으로 전개하는 것을 말한다.

④ ORDER SIBLINGS BY절은 형제 노드 사이에서 정렬을 지정하는 구문이다.

정답 : ② Oracle 계층형 질의에서 루트 노드의 LEVEL 값은 1이다.

**31.다음 중 서브쿼리에 대한 설명으로 가장 부적절한 것은?**

① 메인 쿼리는 조직 테이블(1)이고, 서브쿼리는 사원 테이블(M)을 사용할 경우, 질의 결과는 M레벨인 사원 레벨로 나온다.

② 메인쿼리에서 서브쿼리 칼럼을 사용할 수 없다.

③ 서브쿼리에서 메인쿼리 칼럼은 사용할 수 있다.

④ 서브쿼리의 연결 연산자로서 ‘=’을 사용할 경우, 서브쿼리 파트의 결과는 1건 이하여야 한다.

정답 : ① 서브쿼리를 사용한 경우, 질의 결과는 항상 메인쿼리 레벨과 동일하다.

**32. 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.

정답 : ③ 익스텐트 내 블록들은 서로 인접하지만, 익스텐트끼리 서로 인접하지는 않는다.

**33. 다음 중 논리 데이터 모델을 물리 데이터 모델로 변환할 때 슈퍼/서브타입 데이터 모델에 대한 변환방법으로 가장 부적절한 것은?**

① 공통화된 슈퍼타입에 대해 데이터가 너무 많거나 칼럼이 너무 많은 경우 수직분할과 수평분할을 적용하여 성능을 향상 시키도록 한다.

② 슈퍼타입, 서브타입 각각에 대해 테이블이 독립적으로 구분되어 1:1관계의 테이블이 유지될 수 있도록 한다.

③ 슈퍼타입과 서브타입을 결합하여 한 개의 테이블로 생성하여 서브타입의 개수만큼 테이블을 생성한다.

④ 전체 테이블을 하나의 테이블에 속성을 모두 포함하도록 한다. 다만 서브타입일 경우 속성마다 Not Null을 지정할 수 있지만 통합되면 개별적인 특징을 반영하지 못할 수 있다.

정답 : ④ 슈퍼타입과 서브타입을 변환하는 방식에서는 수직분할과 수평분할 방식이 존재하지 않는다.

**34. 다음 중 SELECT절에 사용하는 서브쿼리인 스칼라 서브쿼리(Scalar Subquery)에 대한 설명으로 가장 적절한 것을 2개 고르시오.**

① 내부적으로 Unique 작업을 실행한다.

② 하나의 로우에 해당하는 결과 건수는 1건 이하여야 한다.

③ Min 또는 Max 함수를 사용해야 한다.

④ 결과 칼럼의 개수는 1개여야 한다.

정답 : ②, ④ 스칼라 서브쿼리의 결과는 1칼럼의 1건이다.

**35. 다음 중 NULL의 설명으로 가장 부적절한 것은?**

① 모르는 값을 의미한다.

② 값의 부재를 의미한다.

③ 공백문자(Empty String) 혹은 숫자 0을 의미한다.

④ NULL과의 모든 비교는 알 수 없음(Unknown)을 반환한다.

정답 : ③ NULL은 공백문자(Empty String) 혹은 숫자 0과 동일하지 않다.

**36. 다음 중 Oracle과 SQL Server의 내부 자료 구조 및 프로세스를 서로 같은 역할을 하는 것끼리 연결할 때 잘못된 것은?**

① Log Buffer = Log Cache

② Online Redo Log = Transaction Log

③ Database Writer = LazyWriter

④ Program Global Area = Procedure Cache

정답 : ④

Oracle에서 Program Global Area는 프로세스에 종속적인 고유 데이터를 저장해 두는 메모리 공간이며, 다른 말로 Private Global Area, Process Global Area라고도 한다. SQL Server의 Procedure Cache는 SQL과 실행계획, 저장형 함수/프로시저를 캐싱해 두는 메모리 공간이다.

**37. 다음 중 바인드 변수에 대한 설명으로 가장 부적절한 것을 2개 고르시오.**

① 바인드 변수를 사용하면 옵티마이저가 칼럼 히스토그램뿐만 아니라 각종 통계정보를 이용하지 못하게 된다.

② 파티션 테이블을 쿼리할 때 파티션 키 칼럼을 바인드 변수로 조회하면 옵티마이저가 파티션 레벨 통계를 활용하지 못한다.

③ 바인드 변수를 사용하면 옵티마이저는 평균 분포를 가정한 실행계획을 수립한다.

④ 바인드 변수 Peeking(또는 Parameter Sniffing) 기능이 도입되면서 칼럼 히스토그램을 사용할 수 있게 되었고, 이 때문에 바인드 변수 사용에 따른 부작용은 거의 해소되었다.

정답 : ①, ④

바인드 변수를 사용하더라도 칼럼 히스토그램을 제외한 나머지 통계정보는 활용할 수 있다.

그리고 바인드 변수 Peeking(또는 Parameter Sniffing) 기능은 오히려 부작용이 많아 사용을 꺼리는 추세고, 따라서 바인드 변수 사용에 따른 부작용을 해소하려고 각 DBMS가 아직 노력 중이다.

**38. 다음 중 SQL과 옵티마이저에 대한 설명으로 가장 부적절한 것을 2개 고르시오.**

① 두 SQL 결과가 같다면, 수행속도도 별 차이가 없다.

② 옵티마이저에 의해 내부적인 쿼리 변환이 이루어지더라도 결과집합에는 영향을 주지 않는다.

③ 사용자가 제시하는 SQL은 결과에 대한 요구사항일 뿐이며 처리절차는 옵티마이저가 결정한다.

④ 사용자가 부여한 조건은 실행계획에 영향을 주지 않는다.

정답 : ①, ④

두 SQL의 결과가 같더라도 SQL 형태에 따라 수행속도가 크게 다를 수 있다.

실행계획에 가장 큰 영향을 미치는 것은 조건절이다.

**39. 다음 중 아래 인덱스를 사용함으로써 정렬(Sort) 연산을 피할 수 있는 경우를 2개 고르시오.**

**(지문 중 SQL이 2개인 것은 Oracle과 SQL Server 구문을 각각 표시한 경우이다.)**

t1\_x01 : c1 + c2 + c3

① select \* from t1 where c1 = ? order by c2 desc, c3 ;

② select \* from (select \* from t1 order by c1, c2, c3) where rownum <= 1 ;select top 1 \* from t1 order by c1, c2, c3 ;

③ select \* from t1 where c1 = ? order by c2 ;

④ select \* from (select \* from t1 where c1 = ? and c2 >= ? order by c3) where rownum <= 10 ;select top 10 \* from t1 where c1 = ? and c2 >= ? order by c3 ;

정답 : ②, ③

인덱스는 기본적으로 오름차순(Ascending)으로 정렬된다. 따라서 order by 칼럼 중 일부만 내림차순일 때는 인덱스로 정렬을 대체할 수 없다. 따라서 ①은 정답이 될 수 없다.

인덱스 뒤쪽 칼럼이 order by 절에 사용된 경우, 그 앞쪽 인덱스 칼럼이 모두 '=' 조건이어야 정렬을 대체할 수 있다. 따라서 ④번도 정답이 될 수 없다.

**40. 다음 중 아래의 인덱스 현황을 고려할 때, (어떤 방식으로 스캔하든) t1\_x02 인덱스만 읽고도 처리할 수 있는 경우로 가장 부적절한 것은? (참고로, Oracle, SQL Server 중 어느 것을 기준으로 하더라도 답은 같다.)**

[Oracle 기준]

create table t1 ( c1, c2, c3, c4, c5, constraint t1\_pk primary key (c1, c2) )

organization index;

create index t1\_x02 on t1 ( c3, c4 );

[SQL Server 기준]

create unique clustered index t1\_pk on t1 ( c1, c2 )

go

create index t1\_x02 on t1 ( c3, c4 )

go

------------------------------------------------------------------------------------------------------

① select c5 from t1 where c3 = ?

② select count(\*) from t1 where c4 >= ?

③ select max(c2) from t1 where c3 between ? and ?

④ select count(\*) from t1 where c3 = ? and c4 is null

정답 : ①

Oracle 기준 : 인덱스 키와 PK 이외의 칼럼을 참조하면 테이블 액세스가 발생한다.

SQL Server 기준 : Non-Clustered Index 키와 Clustered Index 키 이외의 칼럼을 참조하면 테이블 액세스가 발생한다.

**41. 야간에 단독으로 DML을 수행하는 대용량 배치 프로그램의 속도를 향상시키려고 한다. 다음 중 고려할 만한 튜닝 방안으로 가장 부적절한 것은?**

① 인덱스를 제거했다가 나중에 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.

정답 : ③

nologging 옵션은 insert 문장에만 효과가 있으며, update나 delete문에 대해 로깅을 남기지 않는 방법은 제공되지 않는다.

오라클에서의 nologging 기능의 올바른 사용방법은 다음과 같다.

alter table t nologging;

insert /\*+ append \*/ into t select ... from ...;

또는

insert /\*+ parallel(t 2) \*/ into t select ... from ...;

아래와 같이 CTAS 문장으로 테이블을 생성하면서 nologging 옵션을 주어도 같은 효과가 나타난다.

create table t

nologging

as

select ... from ... ;

그 외, SQL\*Loader에서 direct=TRUE 옵션을 사용하는 방법도 있다.

참고로, 아래와 같이 힌트를 사용하는 것은 올바른 사용방법이 아니다. (nologging 힌트는 존재하지 않는다.)

insert /\*+ nologging \*/ into t

select ... from ... ;

아래와 같이 테이블 옆에 nologging을 지정하는 것도 무용지물이며, 이 경우 nologging은 테이블 Alias 역할을 할 뿐이다.

insert into t nologging

select ... from ... ;

DML 성능을 높이기 위해 생각해 볼 수 있는 또 다른 방안은, Target 테이블의 인덱스를 제거했다가 나중에 다시 생성하는 것이다. 왜냐하면, DML 성능이 저하되는 가장 큰 원인 중 하나가 인덱스 관리에 있기 때문이다.

예를 들어, 인덱스가 3개 달린 테이블에 100만 건을 Insert 하려면 테이블 외에 인덱스에도 총 300만 개 레코드를 Insert 해야 한다. 인덱스는 그 특성상 정렬상태를 유지해야 하기 때문에 Freelist에 의해 관리되는 테이블에 Insert 하는 것보다 훨씬 부하가 크다.

따라서 Insert나 Delete의 경우, 전체 인덱스를 Drop(또는 Unusable 상태로 변경)했다가 작업 완료 후 다시 생성하는 것을 고려해 볼 수 있다. Update의 경우는, Update되는 컬럼을 포함하는 인덱스를 제거했다가 나중에 다시 생성하는 것을 고려해 볼 수 있다.

단, 인덱스를 재생성하는 비용도 만만치 않으므로 전체 테이블 건수 중 Insert/Update/Delete 되는 건수의 비중을 따져본 후 결정해야 한다.

**43. 다음 중 아래에서 인덱스를 정상적으로 사용하지 못하거나 비효율적으로 사용하는 SQL을 튜닝하려고 하는데, 튜닝 후에 오히려 성능이 나빠지거나 결과가 틀릴 수 있는 경우는?**

- 단, 인덱스를 사용했을 때의 테이블 액세스량은 손익분기점 미만이라고 가정

- 데이터 타입

\* 주문상태코드 char(1) not null

\* 배송상태코드 char(3) not null

\* 주문수량 number(5) null

------------------------------------------------------------------------------------------------------

① 주문\_X01 인덱스 : 주문상태코드 + 배송상태코드

[ 튜닝 전 ] select \* from 주문 where 주문상태코드 || 배송상태코드 = 'A038'

[ 튜닝 후 ] select \* from 주문 where 주문상태코드 = 'A' and 배송상태코드 = '038'

② 주문\_X03 인덱스 : 주문수량

[ 튜닝 전 ] select \* from 주문 where nvl(주문수량, 0) >= 100

[ 튜닝 후 ] select \* from 주문 where 주문수량 >= 100

③ 주문\_X03 인덱스 : 주문수량

[ 튜닝 전 ] select \* from 주문 where 주문수량 \* 1.1 > 2200

[ 튜닝 후 ] select \* from 주문 where 주문수량 > 2200/1.1

④ 주문\_X02 인덱스: 주문일자 + 배송상태코드 + 주문상태코드

[ 튜닝 전 ]

select \* from 주문

where 주문일자 between '20100901' and '20100930'

and 주문상태코드 || 배송상태코드 in ('A038', 'B045')

[ 튜닝 후 ]

select \* from 주문

where 주문일자 between '20100901' and '20100930'

and 주문상태코드 = 'A'

and 배송상태코드 = '038'

union all

select \* from 주문

where 주문일자 between '20100901' and '20100930'

and 주문상태코드 = 'B'

and 배송상태코드 = '045'

정답 : ④

인덱스 선두 컬럼이 Like, Between, 부등호 같은 범위검색 조건일 때 SQL을 Union All로 분기하면, 인덱스에서 같은 범위를 2번 스캔하기 때문에 비효율이 2배로 증가한다.

**44. 다음 중 아래 SQL의 실행 결과로 가장 적절한 것은?**

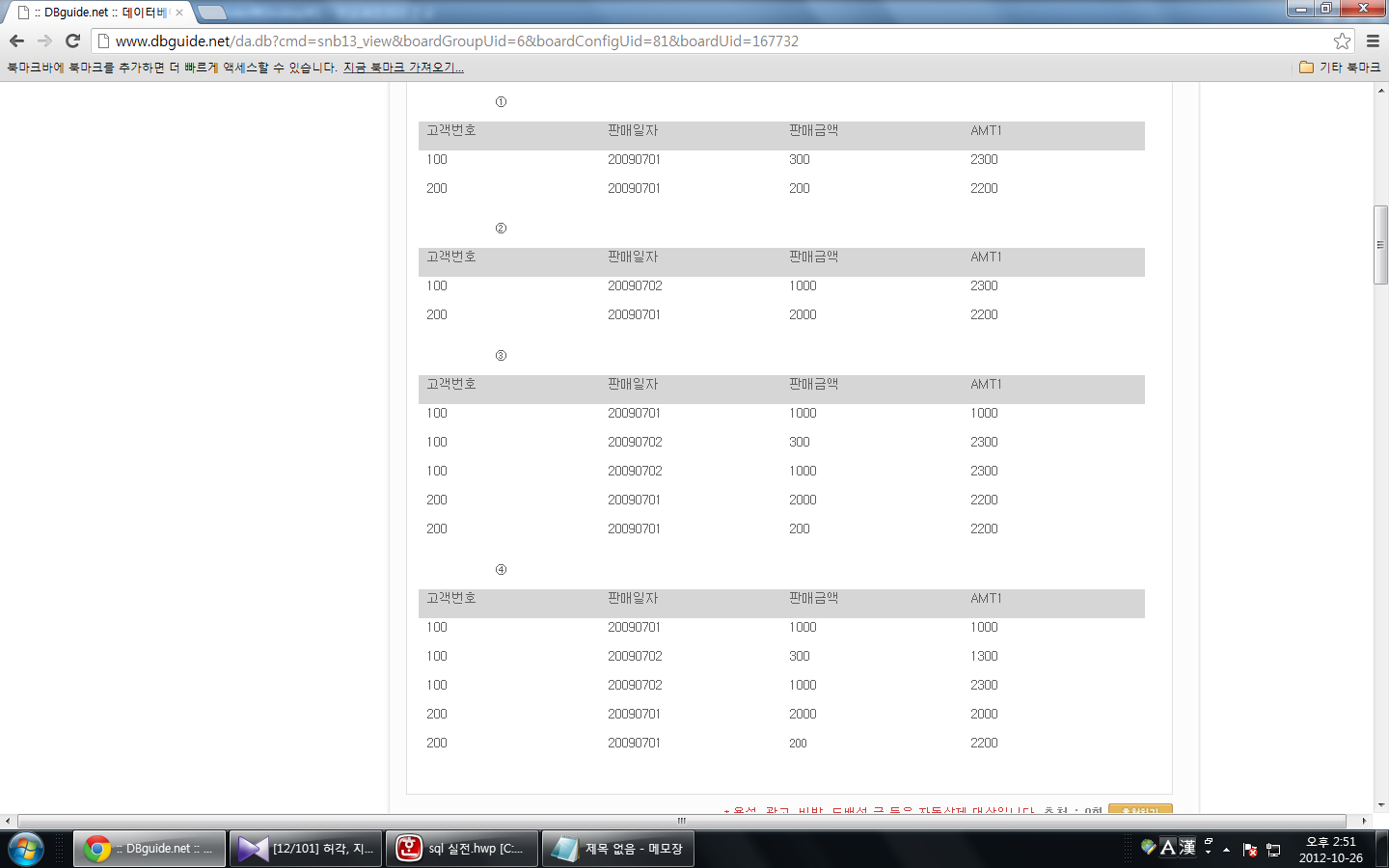
|  |  |  |
| --- | --- | --- |
| **고객번호** | **판매일자** | **판매금액** |
| 100 | 20090701 | 1000 |
| 100 | 20090702 | 300 |
| 100 | 20090702 | 1000 |
| 200 | 20090701 | 2000 |
| 200 | 20090701 | 200 |

SELECT CUSTCODE 고객번호, SALEDATE 판매일자, SALE\_AMT 판매금액,

SUM(SALE\_AMT) OVER (PARTITION BY CUSTCODE ORDER BY SALEDATE

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AMT1

FROM SAL\_TBL



정답 : ④

윈도우 함수를 이용한 누적 합계(RUNNING SUMMARY)를 구하는 SQL이다.

**45. 다음 중 아래와 같은 테이블에서 Full Outer Join의 결과로 가장 적절한 것은?**

[DEPT]

|  |  |
| --- | --- |
| DEPTNO | DNAME |
| 10 | RESARCH |
| 30 | SALES |

[EMP]

|  |  |
| --- | --- |
| DEPTNO | ENAME |
| 30 | KING |
| 40 | WARD |

SELECT D.DEPTNO, E.DEPTNO

FROM DEPT D FULL OUTER JOIN EMP E

ON (D.DEPTNO = E.DEPTNO)



정답 : ②

FULL OUTER JOIN은 DEFT와 EMP 사이에서 조인이 성공한 건과 DEPT, EMP 건 중에서 조인이 실패한 건을 함께 표시한다.

**46. 다음 중 서브쿼리에 대한 설명으로 가장 부적절한 것은?**

① 메인 쿼리는 조직 테이블(1)이고, 서브쿼리는 사원 테이블(M)을 사용할 경우, 질의 결과는 M레벨인 사원 레벨로 나온다.

② 메인쿼리에서 서브쿼리 칼럼을 사용할 수 없다.

③ 서브쿼리에서 메인쿼리 칼럼은 사용할 수 있다.

④ 서브쿼리의 연결 연산자로서 ‘=’을 사용할 경우, 서브쿼리 파트의 결과는 1건 이하여야 한다.

정답 : ①

서브쿼리를 사용한 경우, 질의 결과는 항상 메인쿼리 레벨과 동일하다.

**47. 다음 중 실행계획에 대한 설명으로 가장 부적절한 것은?**

① 실행계획은 SQL 처리를 위한 실행 절차와 방법을 표현한 것이다.

② 실행계획은 조인 방법, 조인 순서, 액세스 기법 등이 표현된다.

③ 동일 SQL문에 대해 실행계획이 다르면 실행 결과도 달라질 수 있다.

④ CBO(Cost Based Optimizer)의 실행계획에는 단계별 예상 비용 및 건수 등이 표시된다.

정답 : ③

실행계획 즉, 실행방법이 달라진다고 해서 결과가 달라지지는 않는다.

**48. 다음 중 아래 문장의 SQL명령으로 가장 적절한 것은?**

기본 테이블 T의 컬럼(P,Q,R) 조합으로 X라 불리는 인덱스를 생성한다.

각각은 P(오름차순), Q(내림차순), R(오름차순)으로 정렬한다.

------------------------------------------------------------------------------------------------------

① CREATE INDEX X ON T(P,Q DESC,R);

② CREATE INDEX X ON T(P,R,Q DESC);

③ CREATE INDEX T ON X(P,Q DESC,R);

④ CREATE INDEX T ON X(P,R,Q DESC);

정답 : ①

인덱스 생성 SQL 문장은 다음 포맷으로 만들어진다.

CREATE INDEX 인덱스명 ON 테이블명 (인덱스로 사용되는 칼럼 LIST);

**49. 다음 중 관계형 데이터베이스의 인덱스(Index)에 대한 설명으로 가장 적절한 것은?**

① 기본 인덱스(Primary Key Index)에 중복된 키 값들이 나타날 수 있다.

② 기본 인덱스에 널 값(Null Value)들이 나타날 수 없다.

③ 보조 인덱스(Secondary Index)에는 고유한 키 값들만 나타날 수 있다.

④ 자주 변경되는 속성은 인덱스를 정의할 좋은 후보이다.

정답 : ②

기본 인덱스 = 기본키 인덱스(PK)는 UNIQUE & NOT NULL의 제약조건을 가진다.

보조 인덱스는 NON UNIQUE 성격을 가질 수 있다.

자주 변경되는 속성은 인덱스로 적절하지 않다.

**50. 다음 SQL 문장 중 COLUMN1의 값이 널(NULL)이 아닌 경우를 찾아내는 문장으로 가장 적절한 것은? (ANSI 표준 기준)**

① SELECT \* FROM MYTABLE WHERE COLUMN1 IS NOT NULL

② SELECT \* FROM MYTABLE WHERE COLUMN1 <> NULL

③ SELECT \* FROM MYTABLE WHERE COLUMN1 != NULL

④ SELECT \* FROM MYTABLE WHERE COLUMN1 NOT NULL

정답 :