2018년 4월 26일 목요일 오후 5:44

SQL 튜닝 목차

■ 1. INDEX 튜닝

- ▶ 1.1 index range scan
- ▶ 1.2 index unique scan
- ▶ 1.3 index skip scan
- ▶ 1.4 index full scan
- ▶ 1.5 index fast full scan
- ▶ 1.6 index merge scan
- ▶ 1.7 index bitmap merge scan
- ▶ 1.8 index join

■ 2. JOIN 튜닝

- ▶ 2.1 join 튜닝
- ▶ 2.2 nested loop join
- ▶ 2.3 hash join
- ▶ 2.4 sort merge join
- ▶ 2.5 Outer join 문장 튜닝

■ 3. 서브쿼리 문장 튜닝

- ▶ 3.1 옵티마이져란?
- ▶ 3.2 쿼리변환이란?
- ▶ 3.3 subquery unnesting
- ▶ 3.4 view merging
- ▶ 3.5 push_pred, no_push_pred

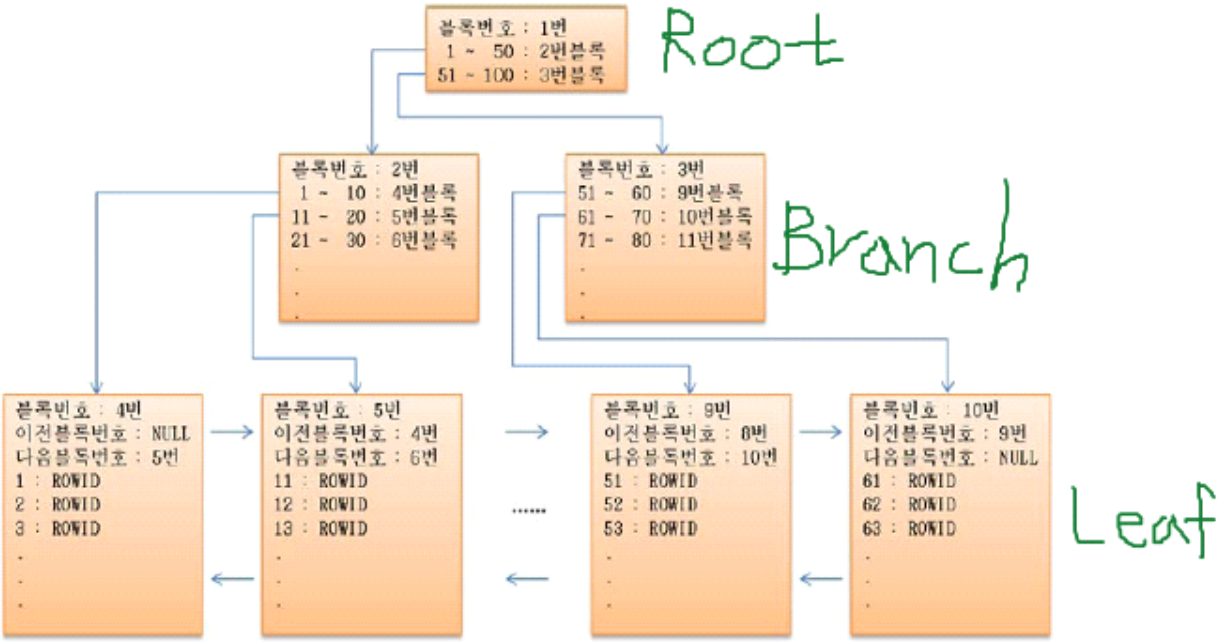
▶ 3.6 스칼라 서브쿼리문를 이용한 튜닝
▶ 3.7 수정가능 조인 뷰
▶ 3.8 분석함수를 이용한 SQL 튜닝
▶ 3.8 WITH 절을 이용한 튜닝

0. 인덱스

2018년 5월 2일 수요일 오후 6:48

우리에게 필요한 데이터를 빨리 찾으려면 인덱스의 도움이 필요하다.

0.1 인덱스 구조



[그림 Ⅲ-4-1] B*Tree 인덱스 구조

인덱스 사용 예시) 키 값이 53인 레코드를 찾아보자.

1. 우선 루트 블록에서 53이 속한 키 값을 찾는다. =3번 블록으로 ..

2. 3번 블록에서 다시 53이 속한 키 값을 찾는다. =9번 블록으로 ..

3. 찾아간 9번은 리프 블록이므로 거기서 값을 찾거나 못 찾거나 둘 중 하나다. 다행히 세 번째 레코드에서 찾아지므로 함께 저장된 ROWID를 이용해 테이블 블록을 찾아간다.
ROWID를 분해해 보면, 오브젝트 번호, 데이터 파일번호, 블록번호, 블록 내 위치 정보를 알 수 있다.

4. 테이블 블록에서 레코드를 찾아간다.

5. 인덱스가 Unique 인덱스가 아닌 한, 값이 53인 레코드가 더 있을 수 있기 때문에 9번 블록에서 레코드 하나를 더 읽어 53인 레코드가 더 있는지 확인한다.
53인 레코드가 더 이상 나오지 않을 때까지 스캔하면서 ④번 테이블 액세스 단계를 반복한다. 만약 9번 블록을 다 읽었는데도 계속 53이 나오면 10번 블록으로 넘어가서 스캔을 계속한다.

b*tree구조는 맨 위쪽 뿌리(Root)에서부터 가지(Branch)를 거쳐 맨 아래 나뭇잎(Leaf)까지 연결되는 구조다.
루트에서 리프 블록까지의 거리를 인덱스 깊이(Height)라고 부르며, 인덱스를 반복적으로 탐색할 때 성능에 영향을 미친다.

루트 & 브랜치 block

하위 노드들의 데이터 값 범위 (**키 값**), 키 값에 해당하는 **블록을 찾는 주소 정보**를 가진다.

리프 block
인덱스 키 값, rowid 를 가진다.
키 값이 같을 때는 ROWID 순으로 정렬된다는 사실도 기억하기 바란다. 리프 블록은 항상 인덱스 키(Key) 값 순으로 정렬돼 있기 때문에 '범위 스캔(Range Scan, 검색조건에 해당하는 범위만 읽다가 멈추는 것을 말함)'이 가능하고, 정방향(Ascending)과 역방향(Descending) 스캔이 둘 다 가능하도록 양방향 연결 리스트(Double linked list) 구조로 연결돼 있다.

■ **oracle ,SQLserver 인덱스 차이점**

- Oracle는 인덱스 구성 **칼럼 중 하나라도 null 값이 아닌 레코드**는 인덱스에 저장한다.
- SQL Server는 인덱스 구성 칼럼이 모**두 null인 레코드도 인덱스에 저장**한다.
- null 값을 Oracle은 **맨 뒤**에 저장하고, SQL Server는 **맨 앞**에 저장한다.

0.2 인덱스 탐색

1. 수평적 탐색
인덱스 **리프 블록에 저장된 레코드끼리** 연결된 순서에 따라 좌에서 우, 또는 우에서 좌로 스캔하기 때문에 '수평적'이라고 표현한다

2.수직적 탐색
수평적 탐색을 위한 시작 지점을 찾는 과정이라고 할 수 있으며, 루트에서 리프 블록까지 아래쪽으로 진행하기 때문에 '수직적'이다.

0.3 인덱스 스캔 방식

0.3.1 Index range scan

사용조건 : 인덱스를 구성하는 **선두 칼럼이 조건절에 사용**
-- 힌트로 강제로 수행하면 Idex full scan 방식으로 수행한다.

Index Range Scan은 [그림 Ⅲ-4-2]처럼 인덱스 루트 블록에서 리프 블록까지 수직적으로 탐색한 후에 리프 블록을 필요한 범위(Range)만 스캔하는 방식이다.
B*Tree 인덱스의 가장 일반적이고 정상적인 형태의 액세스 방식이라고 할 수 있다.

[그림 Ⅲ-4-2] Index Range Scan

실행계획 상에 Index Range Scan이 나타난다고 해서 항상 빠른 속도를 보장하는 것은 아니다. 인덱스를 스캔하는 **범위**

(Range)를 얼마만큼 줄일 수 있느냐, 그리고 **테이블로 액세스하는 횟수를 얼마만큼 줄일 수 있느냐**가 관건이다.

Index Range Scan 과정을 거쳐 생성된 결과집합은 인덱스 칼럼 순으로 정렬된 상태가 되기 때문에 이런 특징을 잘 이용하면 sort order by 연산을 생략하거나 min/max 값을 빠르게 추출할 수 있다.

1. INDEX 튜닝

2018년 4월 19일 목요일 오전 9:44

1. INDEX 튜닝

- 1.1 index range scan
- 1.2 index unique scan
- 1.3 index skip scan
- 1.4 index full scan
- 1.5 index fast full scan
- 1.6 index merge scan
- 1.7 index bitmap merge scan
- 1.8 index join

0. SQL 튜닝

" 쿼리의 검색속도를 높이기 위해서 반드시 알아야 되는 기술 "

■ SQL 튜닝 목차

1.인덱스 튜닝
2.조인 문장 튜닝
3.서브쿼리 문장 튜닝

■ 인덱스 엑세스 방법 8가지

1.index range scan ★
2.index unique scan
3.index skip scan ★
4.index full scan
5.index fast full scan
6.index merge scan
7.index bitmap merge scan
8.index join

Hiredate 의 인덱스는 10개 , sal은 1개만 스캔하면 되므로 emp_sal 인덱스를 사용하는게 더 빠르다.

오라클이 만약 더 느린 인덱스를 스캔했다면 힌트를 사용해서 빠른 인덱스를 사용하도록 하자.

```
SELECT STATEMENT Optimizer Mode=ALL_ROWS
  TABLE ACCESS BY INDEX ROWID          SCOTT.EMP
    INDEX FULL SCAN                    SCOTT.EMP_ENAME
```

2. JOIN 튜닝

2018년 4월 20일 금요일 오후 1:52

2. JOIN 튜닝

- 2.1 join 튜닝
- 2.2 nested loop join
- 2.3 hash join
- 2.4 sort merge join
- 2.5 Outer join 문장 튜닝

1. Index range scan

2018년 4월 30일 월요일 오후 8:18

" 인덱스를 부분 검색하는 스캔 방법 "

```
ex) select ename, sal
        from emp
        where job = 'SALESMAN';
```

문제1. 직업에 인덱스를 생성하고 직업의 인덱스의 구조가 어떻게 생겼는지 조회 하시오.

```
CREATE INDEX emp_job ON emp(job);
SELECT job, ROWID
  FROM EMP
  WHERE  job > ' ';   -- where 절에 인덱스가 있는 컬럼을 사용해서, order by 를 사용하지 않아도 정렬된다.
```

SELECT STATEMENT Optimizer Mode=ALL_ROWS	-	14	252	1
INDEX RANGE SCAN	SCOTT.EMP_JOB	14	252	1

	JOB	ROWID
1	ANALYST	AAAFDqAABAAALLZAAJ
2	ANALYST	AAAFDqAABAAALLZAAL
3	CLERK	AAAFDqAABAAALLZAAH
4	CLERK	AAAFDqAABAAALLZAAK
5	CLERK	AAAFDqAABAAALLZAAM
6	CLERK	AAAFDqAABAAALLZAAN
7	MANAGER	AAAFDqAABAAALLZAAB
8	MANAGER	AAAFDqAABAAALLZAAC
9	MANAGER	AAAFDqAABAAALLZAAD
10	PRESIDENT	AAAFDqAABAAALLZAAA
11	SALESMAN	AAAFDqAABAAALLZAAE
12	SALESMAN	AAAFDqAABAAALLZAAF
13	SALESMAN	AAAFDqAABAAALLZAAG
14	SALESMAN	AAAFDqAABAAALLZAAI

[인덱스의 구조]

ROWID	ENAME	SAL	JOB
AAAFDqAABAAALLZAAA	KING	5000	PRESIDENT
AAAFDqAABAAALLZAAB	BLAKE	2850	MANAGER
AAAFDqAABAAALLZAAC	CLARK	2450	MANAGER
AAAFDqAABAAALLZAAD	JONES	2975	MANAGER
AAAFDqAABAAALLZAAE	MARTIN	1250	SALESMAN
AAAFDqAABAAALLZAAF	ALLEN	1600	SALESMAN
AAAFDqAABAAALLZAAG	TURNER	1500	SALESMAN
AAAFDqAABAAALLZAAH	JAMES	950	CLERK
AAAFDqAABAAALLZAAI	WARD	1250	SALESMAN
AAAFDqAABAAALLZAAJ	FORD	3000	ANALYST
AAAFDqAABAAALLZAAK	SMITH	800	CLERK

[emp 테이블]

문제 2. 사원이름에 인덱스를 걸고 아래의 SQL이 어느 컬럼의 인덱스를 사용하는지 확인 하시오.

* SQLplus에서 실행계획 확인하기

```
Set autot traceonly explain   -- 실행계획만 보겠다.

Select ename, sal, job
        From emp
        Where job='SALESMAN' and ename = 'ALLEN';
```

```
SQL> set autot traceonly explain
SQL> select ename, sal, job
  2  from emp
  3  where job ='SALESMAN' AND ename ='ALLEN';

Execution Plan
----------------------------------------------------------
Plan hash value: 106684950

---------------------------------------------------------------------------------------------
| Id  | Operation                   | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |            |     1 |    26 |     1   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| EMP        |     1 |    26 |     1   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | EMP_ENAME  |     1 |       |     1   (0)| 00:00:01 |
---------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("JOB"='SALESMAN')
   2 - access("ENAME"='ALLEN')

Note
-----
   - dynamic sampling used for this statement (level=2)

SQL>
```

Emp_ename 인덱스와 emp_job 인덱스 두개가 있는데 왜 emp_ename 인덱스를 스캔했을까???

--> 인덱스 구조를 확인해 봐야한다.

	JOB	ROWID
1	ANALYST	AAAFDqAABAAALLZAAJ
2	ANALYST	AAAFDqAABAAALLZAAL
3	CLERK	AAAFDqAABAAALLZAAH
4	CLERK	AAAFDqAABAAALLZAAK
5	CLERK	AAAFDqAABAAALLZAAM
6	CLERK	AAAFDqAABAAALLZAAN
7	MANAGEF	AAAFDqAABAAALLZAAB
8	MANAGEF	AAAFDqAABAAALLZAAC
9	MANAGEF	AAAFDqAABAAALLZAAD
10	PRESIDEN	AAAFDqAABAAALLZAAA
11	SALESMA	AAAFDqAABAAALLZAAE
12	SALESMA	AAAFDqAABAAALLZAAF
13	SALESMA	AAAFDqAABAAALLZAAG
14	SALESMA	AAAFDqAABAAALLZAAI

[emp_job 인덱스 구조]

	ENAME	ROWID
1	ADAMS	AAAFDqAABAAALLZAAM
2	ALLEN	AAAFDqAABAAALLZAAF
3	BLAKE	AAAFDqAABAAALLZAAB
4	CLARK	AAAFDqAABAAALLZAAC
5	FORD	AAAFDqAABAAALLZAAJ
6	JAMES	AAAFDqAABAAALLZAAH
7	JONES	AAAFDqAABAAALLZAAD
8	KING	AAAFDqAABAAALLZAAA
9	MARTIN	AAAFDqAABAAALLZAAE
10	MILLER	AAAFDqAABAAALLZAAN
11	SCOTT	AAAFDqAABAAALLZAAL
12	SMITH	AAAFDqAABAAALLZAAK
13	TURNER	AAAFDqAABAAALLZAAG
14	WARD	AAAFDqAABAAALLZAAI

[emp_ename 인덱스 구조]

*emp_job은 4개가 일치하고 emp_ename은 1개가 일치하므로 검색범위가 적은 emp_ename에서 찾는게 빠르다.

문제 3. 문제2번에서 만약 emp_name 인덱스를 엑세스 하지 않고 emp_job 인덱스를 엑세스 했을때 인덱스 검색 범위가 더 작은 emp_ename 인덱스를 타게 하려면 어떻게 해야하는가?

*힌트를 사용해야한다.

```
SELECT /*+ index(emp emp_job) +*/ ename, sal, job
        FROM EMP
    WHERE job = 'SALESMAN' AND ename = 'ALLEN';
```

SELECT STATEMENT Optimizer Mode=ALL_ROWS		1	26	1
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP	1	26	1
INDEX RANGE SCAN	SCOTT.EMP_JOB	1		1

문제4. 81년도에 입사했고 월급이 3000인 사원의 이름, 입사일, 월급을 출력하는데 월급에도 인덱스를 걸고, 입사일에도 인덱스를 걸어서 쿼리 하시오.

```
CREATE INDEX emp_sal ON  EMP(sal);
CREATE INDEX emp_hiredate ON EMP(hiredate);

SELECT ename, hiredate, sal
        FROM EMP
           WHERE TO_CHAR(hiredate,'RRRR') = '1981' AND SAL = 3000;

           -- 좌변을 가공했기때문에 인덱스 사용 못함

SELECT ename, hiredate, sal
        FROM EMP
           WHERE hiredate BETWEEN TO_DATE('1981/01/01', 'RRRR/MM/DD') AND
TO_DATE('1981/12/31','RRRR/MM/DD')
                    AND SAL = 3000;
```

	ENAME	HIREDATE	SAL
1	FORD	1981-12-11 오전 12:00:00	3000

	HIREDATE	SAL
1	1980-12-09 오전 12:00:00	800
2	1981-02-11 오전 12:00:00	1600
3	1981-02-23 오전 12:00:00	1250
4	1981-04-01 오전 12:00:00	2975
5	1981-05-01 오전 12:00:00	2850
6	1981-05-09 오전 12:00:00	2450
7	1981-08-21 오전 12:00:00	1500
8	1981-09-10 오전 12:00:00	1250
9	1981-11-17 오전 12:00:00	5000
10	1981-12-11 오전 12:00:00	950
11	1981-12-11 오전 12:00:00	3000
12	1982-01-11 오전 12:00:00	1300
13	1982-12-22 오전 12:00:00	3000
14	1983-01-15 오전 12:00:00	1100

```
SELECT /*+ index(emp emp_sal)  */ ename, hiredate, sal
        FROM EMP
```

```
WHERE hiredate BETWEEN TO_DATE('1981/01/01', 'RRRR/MM/DD')
        AND TO_DATE('1981/12/31','RRRR/MM/DD')
         AND SAL = 3000;
```

문제 6. 부서번호와 커미션에 각각 인덱스를 사용하고 부서번호가 30번이고 커미션이 300인 사원의 이름, 월급, 커미션, 부서번호를 출력 하시오.

```
SELECT ename, sal, comm, deptno
    FROM EMP
  WHERE comm = 300 AND deptno = 30;
```

	ENAME	SAL	COMM	DEPTNO
1	ALLEN	1600	300	30

SELECT STATEMENT Optimizer Mode=ALL_ROWS	-
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP
INDEX RANGE SCAN	SCOTT.EMP_COMM

2. Index unique scan

2018년 4월 30일 월요일 오후 8:18

" **Primary key 또는 unique 제약**을 컬럼에 걸면 자동으로 unique 인덱스가 생성되는데 이 인덱스를 엑세스 하는 스캔 방법 "

2.1 예제

예제문제 : 사원 테이블에 사원번호에 primary key를 거시오

```
ALTER TABLE EMP
add CONSTRAINT emp_empno_pk
      PRIMARY KEY(empno);
```

문제 7. 사원 테이블에 걸린 인덱스 리스트를 조회 하시오.

```
SELECT *
      FROM user_indexes
        WHERE table_name = 'EMP';
```

	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME
1	EMP_JOB	NORMAL	SCOTT	EMP
2	EMP_ENAME	NORMAL	SCOTT	EMP
3	EMP_HIREDATE	NORMAL	SCOTT	EMP
4	EMP_SAL	NORMAL	SCOTT	EMP
5	EMP_COMM	NORMAL	SCOTT	EMP
6	EMP_DEPTNO	NORMAL	SCOTT	EMP
7	EMP_EMPNO_PK	NORMAL	SCOTT	EMP

제약이름과 똑같다.

문제 8. 사원번호에 empno에 인덱스의 구조를 확인하시오.

```
SELECT empno, ROWID
      FROM EMP
   WHERE empno >= 0;
```

	EMPNO	ROWID
1	7369	AAAFDqAABAAALLZAAK
2	7499	AAAFDqAABAAALLZAAF
3	7521	AAAFDqAABAAALLZAAI
4	7566	AAAFDqAABAAALLZAAD
5	7654	AAAFDqAABAAALLZAAE
6	7698	AAAFDqAABAAALLZAAB
7	7782	AAAFDqAABAAALLZAAC
8	7788	AAAFDqAABAAALLZAAL

문제 9. 사원번호가 7654인 사원의 사원번호와 이름을 조회하는데 인덱스를 통해서 조회될 수 있도록 힌트를 주시오.

```
SELECT  empno, ename
        FROM EMP
          WHERE empno = 7654;
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS	-	1
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP	1
INDEX UNIQUE SCAN	SCOTT.EMP_EMPNO_PK	1

문제 10. 문제9번 SQL이 실행될 때의 인덱스 엑세스와 테이블 엑세스를 그림으로 그리시오.

Index unique scan은 딱 한 건만 조회 (중복된 데이터가 있으면 primary key 제약을 못주니까)

	EMPNO	ROWID
1	7369	AAAFDqAABAAALLZAAK
2	7499	AAAFDqAABAAALLZAAF
3	7521	AAAFDqAABAAALLZAAI
4	7566	AAAFDqAABAAALLZAAD
5	7654	AAAFDqAABAAALLZAAE
6	7698	AAAFDqAABAAALLZAAB
7	7782	AAAFDqAABAAALLZAAC
8	7788	AAAFDqAABAAALLZAAL
9	7839	AAAFDqAABAAALLZAAA
10	7844	AAAFDqAABAAALLZAAG
11	7876	AAAFDqAABAAALLZAAM
12	7900	AAAFDqAABAAALLZAAH
13	7902	AAAFDqAABAAALLZAAJ
14	7934	AAAFDqAABAAALLZAAN

[emp_no 인덱스]

	ROWID	EMPNO	ENAME	JOB
1	AAAFDqAABAAALLZAAA	7839	KING	PRESIDENT
2	AAAFDqAABAAALLZAAB	7698	BLAKE	MANAGER
3	AAAFDqAABAAALLZAAC	7782	CLARK	MANAGER
4	AAAFDqAABAAALLZAAD	7566	JONES	MANAGER
5	AAAFDqAABAAALLZAAE	7654	MARTIN	SALESMAN
6	AAAFDqAABAAALLZAAF	7499	ALLEN	SALESMAN
7	AAAFDqAABAAALLZAAG	7844	TURNER	SALESMAN
8	AAAFDqAABAAALLZAAH	7900	JAMES	CLERK
9	AAAFDqAABAAALLZAAI	7521	WARD	SALESMAN
10	AAAFDqAABAAALLZAAJ	7902	FORD	ANALYST
11	AAAFDqAABAAALLZAAK	7369	SMITH	CLERK
12	AAAFDqAABAAALLZAAL	7788	SCOTT	ANALYST
13	AAAFDqAABAAALLZAAM	7876	ADAMS	CLERK
14	AAAFDqAABAAALLZAAN	7934	MILLER	CLERK

[emp 테이블]

문제 11. 사원번호가 7654이고 이름이 MARTIN인 사원의 사원번호, 이름, 월급을 조회하고 사원번호의 인덱스를 엑세스 했는지 사원이름의 인덱스를 엑세스 했는지 확인하시오.

```
SELECT empno,ename, sal
    FROM EMP
    WHERE ename = 'MARTIN' AND empno = 7654;
```

index range Scan

unique Scan

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP	1
INDEX UNIQUE SCAN	SCOTT.EMP_EMPNO_PK	1

*unique index와 non unique index 중에 unique index를 사용한다

Why? **Primary key 제약**이 걸렸다는 것은 이미 **데이터가 유니크** 하다는 게 보장된 상태이므로

문제 12. 문제 11번의 인덱스를 ename에 걸린 인덱스를 엑세스 하도록 힌트를 주고 실행 하시오.

```
SELECT /*+ index(emp emp_ename) */ empno,ename, sal
```

FROM EMP
WHERE ename = 'MARTIN' AND empno = 7654;

```
SELECT STATEMENT Optimizer Mode=ALL_ROWS
  TABLE ACCESS BY INDEX ROWID          SCOTT.EMP
    INDEX RANGE SCAN                   SCOTT.EMP_ENAME
```

```
-----------------------------------------------------------
|Operation                                  |   Object Name
-----------------------------------------------------------
3 SELECT STATEMENT Optimizer Mode=ALL_ROWS |
2 TABLE ACCESS BY INDEX ROWID              |SCOTT.EMP
| 1 INDEX RANGE SCAN                       |SCOTT.EMP_ENAME
```

	ENAME	ROWID
1	ADAMS	AAAFDqAABAAALLZAAM
2	ALLEN	AAAFDqAABAAALLZAAF
3	BLAKE	AAAFDqAABAAALLZAAB
4	CLARK	AAAFDqAABAAALLZAAC
5	FORD	AAAFDqAABAAALLZAAJ
6	JAMES	AAAFDqAABAAALLZAAH
7	JONES	AAAFDqAABAAALLZAAD
8	KING	AAAFDqAABAAALLZAAA
9	MARTIN	AAAFDqAABAAALLZAAE
10	MILLER	AAAFDqAABAAALLZAAN
11	SCOTT	AAAFDqAABAAALLZAAL
12	SMITH	AAAFDqAABAAALLZAAK
13	TURNER	AAAFDqAABAAALLZAAG
14	WARD	AAAFDqAABAAALLZAAI

	ROWID	EMPNO	ENAME
1	AAAFDqAABAAALLZAAA	7839	KING
2	AAAFDqAABAAALLZAAB	7698	BLAKE
3	AAAFDqAABAAALLZAAC	7782	CLARK
4	AAAFDqAABAAALLZAAD	7566	JONES
5	AAAFDqAABAAALLZAAE	7654	MARTIN
6	AAAFDqAABAAALLZAAF	7499	ALLEN
7	AAAFDqAABAAALLZAAG	7844	TURNER
8	AAAFDqAABAAALLZAAH	7900	JAMES
9	AAAFDqAABAAALLZAAI	7521	WARD
10	AAAFDqAABAAALLZAAJ	7902	FORD
11	AAAFDqAABAAALLZAAK	7369	SMITH
12	AAAFDqAABAAALLZAAL	7788	SCOTT
13	AAAFDqAABAAALLZAAM	7876	ADAMS
14	AAAFDqAABAAALLZAAN	7934	MILLER

***index range scan은**

index에서 martin을 검색하고 바로 다음 row가 martin 인지 확인한다.

-- why? **Non-Unique** 하고 인덱스에서 **정렬되어 있으므로** 다음꺼 까지만 확인해본다.

문제 13. (점심시간 문제) 직업이 SALESMAN인 사원들의 이름, 월급, 직업을 출력하는 쿼리의 실행계획 그림을 인덱스와 테이블로 나눠서 그리시오. (index range scan 그림이어야 함)

	ENAME	SAL	JOB
1	MARTIN	1250	SALESMAN
2	ALLEN	1600	SALESMAN
3	TURNER	1500	SALESMAN
4	WARD	1250	SALESMAN

```
|Operation                                  |  Object Name   |
-----------------------------------------------------------
| SELECT STATEMENT Optimizer Mode=ALL_ROWS |                |
|  TABLE ACCESS BY INDEX ROWID              |SCOTT.EMP       |
|   INDEX RANGE SCAN                        |SCOTT.EMP_JOB   |
```

	JOB	ROWID
1	ANALYST	AAAFDqAABAAALLZAAJ
2	ANALYST	AAAFDqAABAAALLZAAL
3	CLERK	AAAFDqAABAAALLZAAH
4	CLERK	AAAFDqAABAAALLZAAK
5	CLERK	AAAFDqAABAAALLZAAM
6	CLERK	AAAFDqAABAAALLZAAN

	ROWID	JOB	ENAME
1	AAAFDqAABAAALLZAAA	PRESIDENT	KING
2	AAAFDqAABAAALLZAAB	MANAGER	BLAKE
3	AAAFDqAABAAALLZAAC	MANAGER	CLARK
4	AAAFDqAABAAALLZAAD	MANAGER	JONES
5	AAAFDqAABAAALLZAAE	SALESMAN	MARTIN
6	AAAFDqAABAAALLZAAF	SALESMAN	ALLEN
7	AAAFDqAABAAALLZAAG	SALESMAN	TURNER

```
|Operation                                          |  Object Name   |
-----------------------------------------------------------------------
| SELECT STATEMENT Optimizer Mode=ALL_ROWS          |                |
|  TABLE ACCESS BY INDEX ROWID                      |SCOTT.EMP       |
|   INDEX RANGE SCAN                                |SCOTT.EMP_JOB   |
```

	JOB	ROWID
1	ANALYST	AAAFDqAABAAALLZAAJ
2	ANALYST	AAAFDqAABAAALLZAAL
3	CLERK	AAAFDqAABAAALLZAAH
4	CLERK	AAAFDqAABAAALLZAAK
5	CLERK	AAAFDqAABAAALLZAAM
6	CLERK	AAAFDqAABAAALLZAAN
7	MANAGEF	AAAFDqAABAAALLZAAB
8	MANAGEF	AAAFDqAABAAALLZAAC
9	MANAGEF	AAAFDqAABAAALLZAAD
10	PRESIDEN	AAAFDqAABAAALLZAAA
11	SALESMA	AAAFDqAABAAALLZAAE
12	SALESMA	AAAFDqAABAAALLZAAF
13	SALESMA	AAAFDqAABAAALLZAAG
14	SALESMA	AAAFDqAABAAALLZAAI

	ROWID	JOB	ENAME
1	AAAFDqAABAAALLZAAA	PRESIDENT	KING
2	AAAFDqAABAAALLZAAB	MANAGER	BLAKE
3	AAAFDqAABAAALLZAAC	MANAGER	CLARK
4	AAAFDqAABAAALLZAAD	MANAGER	JONES
5	AAAFDqAABAAALLZAAE	SALESMAN	MARTIN
6	AAAFDqAABAAALLZAAF	SALESMAN	ALLEN
7	AAAFDqAABAAALLZAAG	SALESMAN	TURNER
8	AAAFDqAABAAALLZAAH	CLERK	JAMES
9	AAAFDqAABAAALLZAAI	SALESMAN	WARD
10	AAAFDqAABAAALLZAAJ	ANALYST	FORD
11	AAAFDqAABAAALLZAAK	CLERK	SMITH
12	AAAFDqAABAAALLZAAL	ANALYST	SCOTT
13	AAAFDqAABAAALLZAAM	CLERK	ADAMS
14	AAAFDqAABAAALLZAAN	CLERK	MILLER

3. Index skip scan

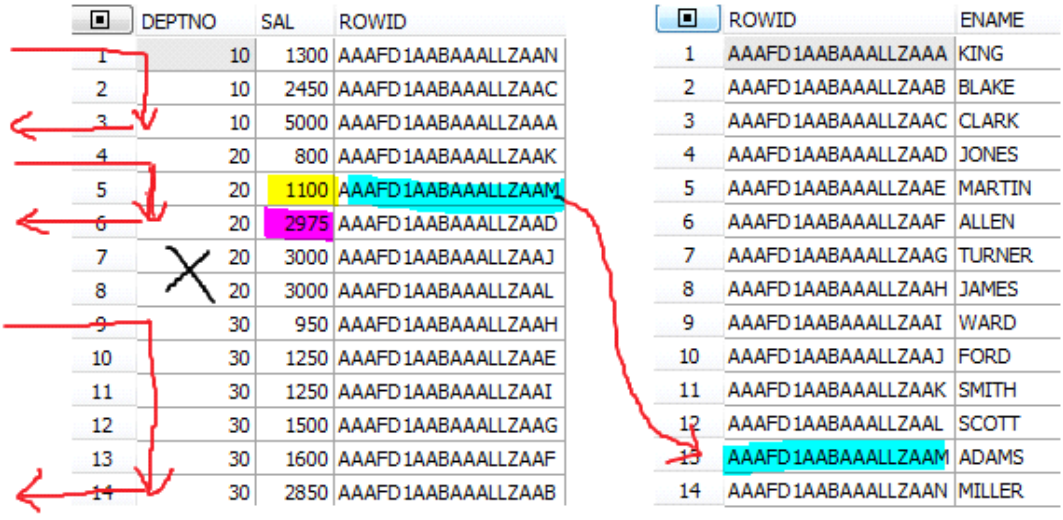
2018년 4월 30일 월요일 오후 8:20

" 인덱스를 전부 스캔하지 않고 스킵 해서 스캔하는 엑세스 방법 "

-- Index skip scan을 이해하려면 결합 컬럼을 이해 해야한다.

3.1 index skip scan 그림

Select ename, sal
From emp
Where sal = 1100;



	DEPTNO	SAL	ROWID
1	10	1300	AAAFD1AABAAALLZAAN
2	10	2450	AAAFD1AABAAALLZAAC
3	10	5000	AAAFD1AABAAALLZAAA
4	20	800	AAAFD1AABAAALLZAAK
5	20	1100	AAAFD1AABAAALLZAAM
6	20	2975	AAAFD1AABAAALLZAAD
7	20	3000	AAAFD1AABAAALLZAAJ
8	20	3000	AAAFD1AABAAALLZAAL
9	30	950	AAAFD1AABAAALLZAAH
10	30	1250	AAAFD1AABAAALLZAAE
11	30	1250	AAAFD1AABAAALLZAAI
12	30	1500	AAAFD1AABAAALLZAAG
13	30	1600	AAAFD1AABAAALLZAAF
14	30	2850	AAAFD1AABAAALLZAAB

	ROWID	ENAME
1	AAAFD1AABAAALLZAAA	KING
2	AAAFD1AABAAALLZAAB	BLAKE
3	AAAFD1AABAAALLZAAC	CLARK
4	AAAFD1AABAAALLZAAD	JONES
5	AAAFD1AABAAALLZAAE	MARTIN
6	AAAFD1AABAAALLZAAF	ALLEN
7	AAAFD1AABAAALLZAAG	TURNER
8	AAAFD1AABAAALLZAAH	JAMES
9	AAAFD1AABAAALLZAAI	WARD
10	AAAFD1AABAAALLZAAJ	FORD
11	AAAFD1AABAAALLZAAK	SMITH
12	AAAFD1AABAAALLZAAL	SCOTT
13	AAAFD1AABAAALLZAAM	ADAMS
14	AAAFD1AABAAALLZAAN	MILLER

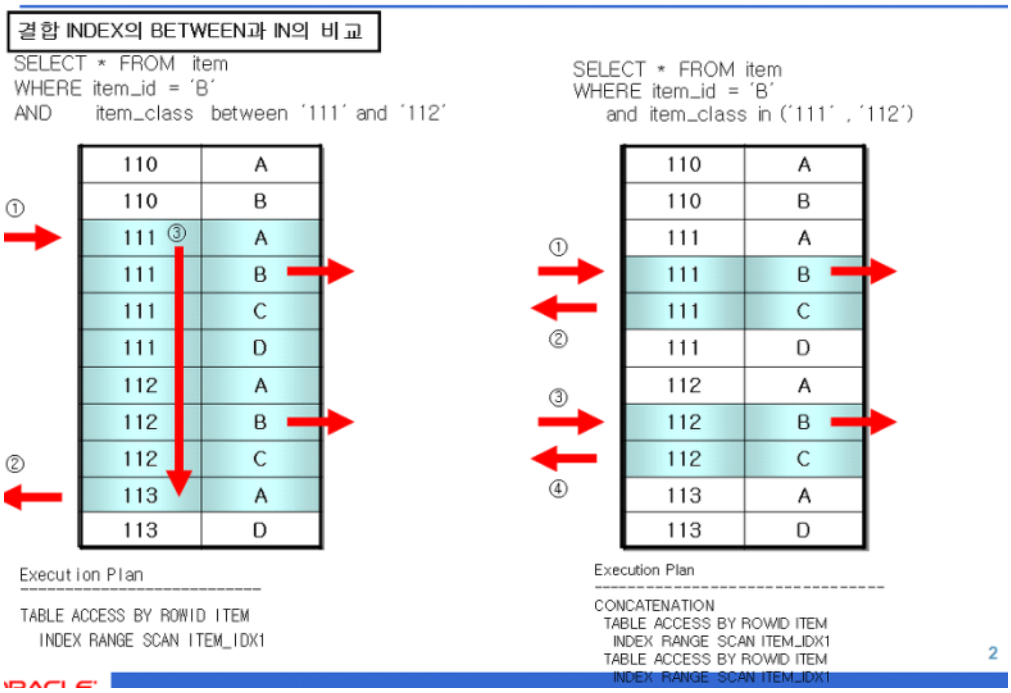
```
SELECT /*+ index_ss(emp emp_Deptno_sal) */ ename, sal
    FROM EMP
      WHERE sal = 1100;
```

*** index skip scan은 결합 컬럼 인덱스의 첫번째 컬럼이 없는데도 index를 사용한다.**

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	-
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP
INDEX SKIP SCAN	SCOTT.EMP_DEPTNO_SAL

3.2 index skip scan 사용할 때

* 결합 컬럼 인덱스의 첫번째 컬럼이 where 절에서 선분조건 (betwwen .. and like) 으로 사용 된다면
인덱스를 넓게 읽을 수 밖에 없기 때문에 성능이 저하된다. 그럴 때는 두가지 방법으로 튜닝한다.
1 index skip scan 사용
2 between .. and 를 in으로 변경

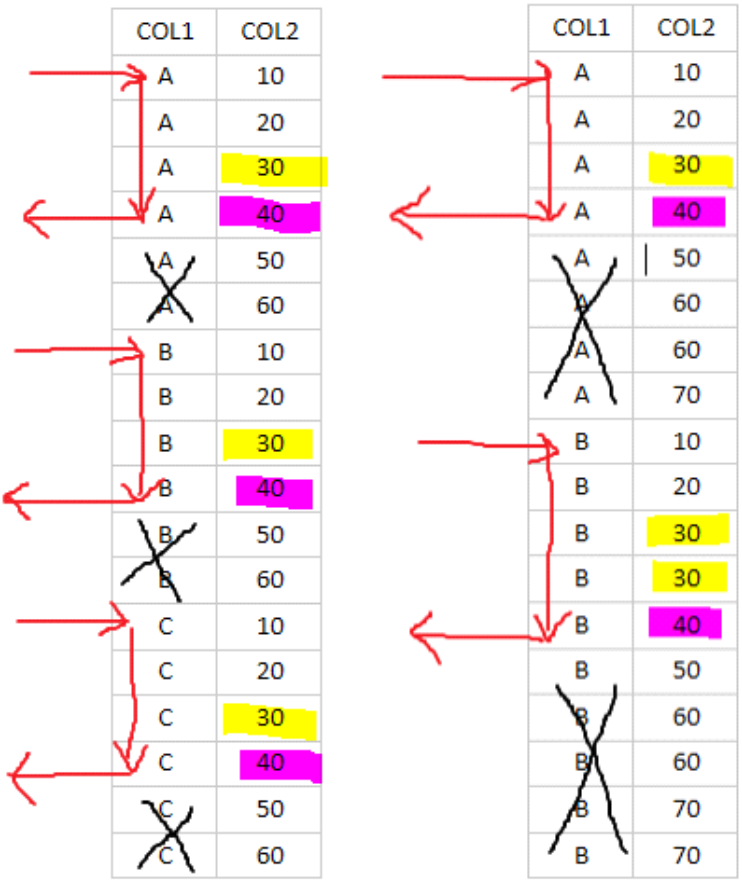


** item_class 값이 111.7도 있을 수 있지 않을까? 111 와 112 사이니까.

3.3 index skip scan 효과를 보기위한 조건

결합 컬럼 인덱스의 첫번째 컬럼의 데이터의 종류가 몇가지 안되어야 효과를 볼 수 있다.

```
SELECT /*+index_ss(emp emp_col1_col2) */ *
       FROM EMP
   WHERE col2 = 30;
```



3.4 예제

문제 14. demobld 스크립트를 수행하고 아래와 같이 결합 컬럼 인덱스를 생성하시오.

```
CREATE INDEX emp_deptno_sal ON EMP(deptno, sal);
```

문제 15. 방금 만든 emp_deptno_sal 의 인덱스 구조가 어떻게 되었는지 확인하시오.

```
SELECT deptno, sal, ROWID
        FROM EMP
          WHERE deptno >= 0;
```

	DEPTNO	SAL	ROWID
1	10	1300	AAAFD1AABAAALLZAAN
2	10	2450	AAAFD1AABAAALLZAAC
3	10	5000	AAAFD1AABAAALLZAAA
4	20	800	AAAFD1AABAAALLZAAK
5	20	1100	AAAFD1AABAAALLZAAM
6	20	2975	AAAFD1AABAAALLZAAD
7	20	3000	AAAFD1AABAAALLZAAJ
8	20	3000	AAAFD1AABAAALLZAAL
9	30	950	AAAFD1AABAAALLZAAH
10	30	1250	AAAFD1AABAAALLZAAE
11	30	1250	AAAFD1AABAAALLZAAI
12	30	1500	AAAFD1AABAAALLZAAG
13	30	1600	AAAFD1AABAAALLZAAF
14	30	2850	AAAFD1AABAAALLZAAB

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
INDEX RANGE SCAN	SCOTT.EMP_DEPTNO_SAL

Table access 할 필요가 없다. Deptno, sal 모두 인덱스에 있기 때문에!

문제 16. 부서번호가 20번인 사원들의 이름, 월급, 부서번호를 출력하는데 emp_deptno_sal 인덱스를 엑세스 할 수 있도록 힌트를 주시오.

```
SELECT /*+ index(emp emp_deptno_sal) */ ename, sal, deptno
      FROM EMP
   WHERE deptno >=0;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	-
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP
INDEX RANGE SCAN	SCOTT.EMP_DEPTNO_SAL

문제 17. 월급이 3000인 사원의 이름, 월급을 출력하는데 실행계획에 emp_deptno_sal 인덱스에 엑세스 하는지 확인 하시오.

```
SELECT ename, sal
      FROM EMP
   WHERE sal = 3000;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
TABLE ACCESS FULL	SCOTT.EMP

**** 왜 sal 조건을 줬는데도 emp_deptno_sal 인덱스에 엑세스 하지 못하고 full table scan을 하는가?**

결합 컬럼 인덱스의 **첫번째 컬럼**이 **where 절에 검색조건에 존재**해야 그 인덱스를 엑세스 할 수 있다.

즉, 결합 컬럼 인덱스를 사용하기 위해선 결합 컬럼 인덱스의 첫번째 컬럼이 where 절 검색조건에 존재 해야 사용가능 하다!

이런 경우 emp_deptno_sal 인덱스를 사용하려면 index skip scan을 사용해야 한다.

■ **index skip scan 그림**

```
Select ename, sal
      From emp
      Where sal = 1100;
```

DEPTNO	SAL	ROWID

ROWID	ENAME

	DEPTNO	SAL	ROWID
1	10	1300	AAAFD1AABAAALLZAAN
2	10	2450	AAAFD1AABAAALLZAAC
3	10	5000	AAAFD1AABAAALLZAAA
4	20	800	AAAFD1AABAAALLZAAK
5	20	1100	AAAFD1AABAAALLZAAM
6	20	2975	AAAFD1AABAAALLZAAD
7	20	3000	AAAFD1AABAAALLZAAJ
8	20	3000	AAAFD1AABAAALLZAAL
9	30	950	AAAFD1AABAAALLZAAH
10	30	1250	AAAFD1AABAAALLZAAE
11	30	1250	AAAFD1AABAAALLZAAI
12	30	1500	AAAFD1AABAAALLZAAG
13	30	1600	AAAFD1AABAAALLZAAF
14	30	2850	AAAFD1AABAAALLZAAB

	ROWID	ENAME
1	AAAFD1AABAAALLZAAA	KING
2	AAAFD1AABAAALLZAAB	BLAKE
3	AAAFD1AABAAALLZAAC	CLARK
4	AAAFD1AABAAALLZAAD	JONES
5	AAAFD1AABAAALLZAAE	MARTIN
6	AAAFD1AABAAALLZAAF	ALLEN
7	AAAFD1AABAAALLZAAG	TURNER
8	AAAFD1AABAAALLZAAH	JAMES
9	AAAFD1AABAAALLZAAI	WARD
10	AAAFD1AABAAALLZAAJ	FORD
11	AAAFD1AABAAALLZAAK	SMITH
12	AAAFD1AABAAALLZAAL	SCOTT
13	AAAFD1AABAAALLZAAM	ADAMS
14	AAAFD1AABAAALLZAAN	MILLER

```
SELECT /*+ index_ss(emp emp_Deptno_sal) */ ename, sal
        FROM EMP
          WHERE sal = 1100;
```

*** index skip scan은 결합 컬럼 인덱스의 첫번째 컬럼이 없는데도 index를 사용한다.**

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	-
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP
INDEX SKIP SCAN	SCOTT.EMP_DEPTNO_SAL

문제 18. 직업 + 월급으로 결합 컬럼 인덱스를 생성하고 아래의 SQL이 index skip scan을 할 수 있도록 힌트를 주시오.

```
CREATE INDEX emp_job_sal ON EMP(job,sal);

SELECT /*+ index_ss(emp emp_job_sal) */ ename, sal, job, hiredate
        FROM EMP
                WHERE sal = 1250;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP
INDEX SKIP SCAN	SCOTT.EMP_JOB_SAL

■ index skip scan 효과를 보기위한 조건

결합 컬럼 인덱스의 첫번째 컬럼의 데이터의 종류가 몇가지 안되어야 효과를 볼 수 있다.

```
SELECT /*+index_ss(emp emp_col1_col2) */ *
        FROM EMP
    WHERE col2 = 30;
```

COL1	COL2
A	10
A	20
A	30
A	40
A	50
A	60
B	10
B	20
B	30
B	40
B	50
B	60
C	10
C	20
C	30
C	40
C	50
C	60

COL1	COL2
A	10
A	20
A	30
A	40
A	50
A	60
A	60
A	70
B	10
B	20
B	30
B	30
B	40
B	50
B	60
B	60
B	70
B	70

**첫번째 컬럼(COL1)의 데이터가 A-Z 까지 있는 경우보다
A-C까지만 있는 경우가 더 성능이 좋다. **(컬럼이 적을수록 스킵 많이 할 수 있어서)**

문제 19. 사원 테이블 직업의 종류가 몇 개인가?

```
SELECT COUNT(DISTINCT job)
    FROM EMP;
```

	COUNT(DISTINCTJOB)
1	5

4. Index full scan

2018년 4월 30일 월요일 오후 8:20

" 인덱스 전체를 처음부터 끝까지 스캔하는 스캔 방법 "

4.1 예제

문제 20. 사원 테이블의 인원수가 몇 명인지 카운트 하시오.

```
SELECT COUNT (*) FROM EMP;
SELECT COUNT (empno) FROM EMP;
SELECT COUNT (1) FROM EMP;
```

	COUNT(*)
1	14

문제 21. 사원 테이블의 사원번호에 인덱스를 생성하고 사원의 인원수가 몇 명인지 카운트 하시오.

```
ALTER table EMP
        ADD CONSTRAINT emp_empno_pk
           PRIMARY KEY (empno);

SELECT COUNT (*) FROM EMP;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
SORT AGGREGATE	
INDEX FULL SCAN	SCOTT.EMP_EMPNO_PK

** index full scan이 table full scan보단 빠르다.

문제 22. 문제 21번을 다시 수행하는데 테이블 table full scan으로 수행되게 하시오.

```
SELECT /*+ full(emp) */ COUNT(empno)
        FROM EMP;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	3

문제 23. 다시 index full scan을 주고 실행한 후에 몇 개의 블락을 읽었는지 확인 하시오.

```
SELECT /*+ index_fs(emp emp_empno_pk) */ COUNT(empno)
        FROM EMP;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	1

5. Index fast full scan

2018년 4월 30일 월요일 오후 8:21

" index full scan 보다 더 성능이 좋은 스캔 방법 "

■ 더 성능이 좋은 이유? " 병렬 처리가 가능하다 "

5.1 예제

문제 24. 문제23번의 index full scan 실행 계획을 index fast full scan이 되도록 힌트를 주시오.

```
SELECT /*+ index_ffs(emp emp_empno_pk) */ COUNT(empno)
        FROM EMP;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
SORT AGGREGATE	
INDEX FAST FULL SCAN	SCOTT.EMP_EMPNO_PK

** index fast full scan이 안되는 경우 empno에 not null 제약을 걸거나 where ename is not null ; 을 써준다.

문제 25. 직업, 직업별 인원수를 출력하는데 아래의 인덱스를 이용해서 수행 하시오.

```
SELECT /*+ index_ffs(emp emp_job_ename) */ job, COUNT(*)
        FROM EMP
          GROUP BY job;
```

문제 26. 부서번호, 부서번호 별 인원수를 출력하는데 가장 빠르게 출력될 수 있도록 적절한 인덱스를 생성하고 힌트를 주고 수행 하시오.

```
SELECT /*+ index_ffs(emp emp_deptno_empno) */ deptno, COUNT(empno)
        FROM EMP
          GROUP BY deptno;
```

SELECT STATEMENT Optimizer Mode=ALL_ROWS	
HASH GROUP BY	
INDEX FAST FULL SCAN	SCOTT.EMP_DEPTNO_EMPNO

문제 27. 문제 26번의 SQL을 병렬처리 할 수 있도록 힌트를 주고 수행 하시오.

```
SELECT /*+ index_ffs(emp emp_deptno_empno) parallel_index(emp, emp_deptno_empno,4) */
        deptno, COUNT(empno)
        FROM EMP
       GROUP BY deptno;
```

테이블 index 병렬도

현재 쓰고있는 버전에서는 flashback, parallel_index를 사용 못함

Optimizer 모드 : Default ☑ 실행 계획 개체 보기(O)

트리 뷰 | 텍스트 | 순서도

Operation	Object Name	Rows	Bytes	Cost	Object Node	In/Out	PStart	PStop
SELECT STATEMENT Optimizer Mode=ALL_ROWS		14	182	6				
PX COORDINATOR								
PX SEND QC (RANDOM)	SYS.:TQ10001	14	182	6	:Q1001	P->S	QC (RANDOM)	
HASH GROUP BY		14	182	6	:Q1001	PCWP		
PX RECEIVE		14	182	6	:Q1001	PCWP		
PX SEND HASH	SYS.:TQ10000	14	182	6	:Q1000	P->P	HASH	
HASH GROUP BY		14	182	6	:Q1000	PCWP		
PX BLOCK ITERATOR		14	182	2	:Q1000	PCWC		
INDEX FAST FULL SCAN	SCOTT.EMP_DEPTNO_EMPNO	14	182	2	:Q1000	PCWP		

문제 28. 이름에 EN 또는 IN을 포함하고 있는 사원들의 이름, 월급, 직업을 출력 하시오.

```
SELECT /*+ index(emp emp_ename)*/ ename, sal
        FROM EMP
       WHERE ename LIKE '%EN%' or ename LIKE '%IN%';
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	40

문제 29. 아래의 SQL을 튜닝 하시오.

```
SELECT /*+ index(emp emp_ename)*/ ename, sal
        FROM EMP
       WHERE ename LIKE '%EN%' or ename LIKE '%IN%';
```

방법1. 이름에 EN 또는 IN이 포함된 사원들의 데이터의 rowid를 출력하시오.

```
SELECT /*+ INDEX_ffs(emp emp_ename) */ ROWID
        FROM EMP
       WHERE ename LIKE '%EN%' OR ENAME LIKE '%IN%';
```

SELECT STATEMENT Optimizer Mode=ALL_ROWS
INDEX FAST FULL SCAN SCOTT.EMP_ENAME

	ROWID
1	AAAFD1AABAAALLZAAF
2	AAAFD1AABAAALLZAAA
3	AAAFD1AABAAALLZAAE

방법2. 위에서 구한 rowid 3건을 가지고 해당 rowid의 테이블의 이름, 월급, 직업을 출력 하시오.

```
SELECT ename, sal, job
        FROM EMP
    WHERE ROWID IN (SELECT /*+ INDEX_ffs(emp emp_ename) */ ROWID
                                    FROM EMP
                                  WHERE ename LIKE '%EN%' OR ENAME LIKE '%IN%');
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	41

```
SELECT STATEMENT Optimizer Mode=ALL_ROWS
    TABLE ACCESS FULL                    SCOTT.EMP
```

** 옵티마이저가 자동으로 아래와 같이 바꿔서 수행한다.

```
SELECT ename, sal, job
        FROM EMP
           WHERE ename LIKE '%EN%' OR ename LIKE '%IN%';
```

- Index fast full scan 을 쓰려면 ..? (뒤에서 배우니 보기만 하자)

```
SELECT /*+ leading(e3 e1) use_n1(e1) rowid(e1) */ e1.ename, e1.sal, e1.job
        FROM EMP e1, (SELECT /*+ index_ffs (e2 emp_ename) no_merge */ ROWID rr
                                FROM EMP e2
                                        WHERE ename LIKE '%EN%'
                                                OR ename LIKE '%in%' ) e3
           WHERE E1.ROWID = e3.rr;
```

```
SELECT STATEMENT Optimizer Mode=ALL_ROWS                    1    50    3
  NESTED LOOPS                                              1    50    3
    VIEW                                                    1    12    2
      INDEX FAST FULL SCAN          SCOTT.EMP_ENAME         1    19    2
    TABLE ACCESS BY USER ROWID      SCOTT.EMP               1    38    1
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	4

문제 30. (오늘 마지막 문제) 이름에 EN 또는 IN을 포함하는 사원들의 이름, 월급, 직업을 출력하는데 정규식을 사용하면 성능이 더 좋아지는지 확인해 보시오.

```
SELECT /*+ index(emp emp_ename)*/ ename, sal, job
        FROM EMP
           WHERE regexp_like(ename,'(EN|IN)');
```

```
SELECT STATEMENT Optimizer Mode=ALL_ROWS
    TABLE ACCESS FULL                    SCOTT.EMP
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	41

6. Index merge scan

2018년 4월 30일 월요일 오후 8:21

" where에 조건이 여러 개가 있고 그 조건에 사용되는 컬럼이 **다 단일 컬럼 인덱스로 구성**되어 있을 때
하나의 인덱스를 사용하는게 아니라 **여러 개의 인덱스를 동시에 사용**해서 시너지 효과를 보는 스캔 방법 "

6.1 예제

문제 31. demobld.sql을 돌리고 직업이 SALESMAN이고 부서번호가 30번인 사원들의 이름, 월급,
직업, 부서번호를 출력하시오.

```
SELECT ENAME, sal, job, deptno
    FROM EMP
      WHERE job = 'SALESMAN' AND deptno = 30;
```

```
SELECT STATEMENT Optimizer Mode=ALL_ROWS
  TABLE ACCESS FULL                    SCOTT.EMP
```

	ENAME	SAL	JOB	DEPTNO
1	MARTIN	1250	SALESMAN	30
2	ALLEN	1600	SALESMAN	30
3	TURNER	1500	SALESMAN	30
4	WARD	1250	SALESMAN	30

문제 32. 직업, 부서번호에 각각 단일 컬럼 인덱스를 생성하고 문제 31번 쿼리의 실행계획을 보면 직업, 부서번호 중 어느 컬럼에 인덱스를 엑세스 하는가?

```
SELECT ENAME, sal, job, deptno
    FROM EMP
      WHERE job = 'SALESMAN' AND deptno = 30;
```

```
SELECT STATEMENT Optimizer Mode=ALL_ROWS                4    156    1
  TABLE ACCESS BY INDEX ROWID        SCOTT.EMP          4    156    1
    INDEX RANGE SCAN                 SCOTT.EMP_JOB      1           1
```

문제 33. 그렇다면 왜 deptno 인덱스를 타지 않고 job의 인덱스를 탔을까?

Emp_deptno 와 emp_job 두개의 인덱스 중에서 index range scan을 더 짧게 하는 인덱

스를 선택하기 때문!

SELECT COUNT(*) FROM EMP WHERE deptno = 30;

	COUNT(*)
1	6

SELECT COUNT(*) FROM EMP WHERE job ='SALESMAN';

	COUNT(*)
1	4

JOB	ROWID
ANALYST	AAAFFMAABAAALLZAAJ
ANALYST	AAAFFMAABAAALLZAAL
CLERK	AAAFFMAABAAALLZAAH
CLERK	AAAFFMAABAAALLZAAK
CLERK	AAAFFMAABAAALLZAAM
CLERK	AAAFFMAABAAALLZAAN
MANAGEF	AAAFFMAABAAALLZAAB
MANAGEF	AAAFFMAABAAALLZAAC
MANAGEF	AAAFFMAABAAALLZAAD
PRESIDEI	AAAFFMAABAAALLZAAA
SALESMA	AAAFFMAABAAALLZAAE
SALESMA	AAAFFMAABAAALLZAAF
SALESMA	AAAFFMAABAAALLZAAG
SALESMA	AAAFFMAABAAALLZAAI

DEPTNO	ROWID
10	AAAFFMAABAAALLZAAA
10	AAAFFMAABAAALLZAAC
10	AAAFFMAABAAALLZAAN
20	AAAFFMAABAAALLZAAD
20	AAAFFMAABAAALLZAAJ
20	AAAFFMAABAAALLZAAK
20	AAAFFMAABAAALLZAAL
20	AAAFFMAABAAALLZAAM
30	AAAFFMAABAAALLZAAB
30	AAAFFMAABAAALLZAAE
30	AAAFFMAABAAALLZAAF
30	AAAFFMAABAAALLZAAG
30	AAAFFMAABAAALLZAAH
30	AAAFFMAABAAALLZAAI

1	recursive calls	0
2	db block gets	0
3	consistent gets	40

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP
INDEX RANGE SCAN	SCOTT.EMP_JOB

문제 34. 그러면 emp_deptno 인덱스를 엑세스 하게끔 힌트를 주고 실행 해보시오.

```
SELECT /*+ index(emp emp_Deptno) */ ename, sal, job
        FROM EMP
    WHERE deptno = 30 AND job = 'SALESMAN';
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	40

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP
INDEX RANGE SCAN	SCOTT.EMP_DEPTNO

-----> 여러 개의 인덱스를 동시에 사용해서 시너지 효과를 주자 !! = **index merge scan**

문제 35. 아래의 SQL이 index merge scan이 될 수 있도록 힌트를 주시오.

```
SELECT /*+ and_equal(emp emp_job emp_deptno)*/ ename, sal, job, deptno
        FROM EMP
        WHERE job = 'SALESMAN' AND deptno = 30;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	43

오히려 40 ---> 43으로 증가했다.

	Operation	Object Name	Rows
5	SELECT STATEMENT Optimizer Mode=ALL_ROWS	-	4
4	TABLE ACCESS BY INDEX ROWID	SCOTT.EMP	4
3	AND-EQUAL		
1	INDEX RANGE SCAN	SCOTT.EMP_DEPTNO	1
2	INDEX RANGE SCAN	SCOTT.EMP_JOB	1

오히려 역효과가 났다.. -> 요즘은 index bitmap merge scan을 사용한다

7. Index bitmap merge scan

2018년 4월 30일 월요일 오후 8:22

" index merge scan이 진화된 기술로 여러 개의 인덱스를 각각 bitmap으로 변환해서 하나로 합쳐서 구한 rowid로 테이블을 엑세스 하는 스캔 방법 "

7.1 예제

Ex) SELECT **/*+ index_combine(emp)*/** ename, sal, job, deptno
FROM EMP
WHERE job = 'SALESMAN' AND deptno = 30;

문제 36. 우리반 테이블에 전공과 통신사에 각각 단일 컬럼 인덱스를 걸고, 전공이 경제학이고 통신사가 sk인 학생들의 이름, 전공, 통신사를 출력하는 쿼리의 실행 계획이 index bitmap merge scan이 되도록 하시오.

```
SELECT /*+ index_combine(emp2) */ ename, major, telecom
FROM EMP2
WHERE telecom = 'sk' AND major='경제학';
```

	ENAME	MAJOR	TELECOM
1	윤진민	경제학	sk

-- 현재 버전에선 안통함 ..

8. Index join

2018년 4월 30일 월요일 오후 8:23

" **테이블을 엑세스 하지 않으면서** 인덱스만 가지고 조인을 해서 결과를 보여주는 스캔 방법 "

8.1 예제

예제1. demobld 파일을 돌리고 사원번호, 이름에 각각 단일 컬럼 인덱스를 생성하고 아래의 sql 실행계획을 확인해본다.

```
SELECT empno, ename
    FROM EMP
  WHERE empno = 7788;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP
INDEX RANGE SCAN	SCOTT.EMP_EMPNO

문제 37. 아래의 SQL을 index join 엑세스 방법으로 힌트를 줘서 테이블 엑세스를 하지 말게 하시오.

```
SELECT /*+ index_join(emp) */empno, ename
    FROM EMP
  WHERE empno = 7788;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP
INDEX RANGE SCAN	SCOTT.EMP_EMPNO

---> 여전히 TABLE ACCESS 한다.. **(컬럼 두개 모두 not null 제약을 걸어 줘야된다)**

문제 38. empno, ename에 각각 not null 제약을 걸고 실행계획을 확인하시오.

```
ALTER table EMP
    MODIFY empno CONSTRAINT emp_empno_nn NOT NULL;  -- 이미 not null 걸려있다.

ALTER table EMP
    MODIFY ename CONSTRAINT emp_ename_nn NOT NULL;

SELECT /*+ index_join(emp) */empno, ename
    FROM EMP
  WHERE empno = 7788;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
VIEW	SCOTT.index$_join$_001

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
VIEW	SCOTT.index$_join$_001
HASH JOIN	
INDEX RANGE SCAN	SCOTT.EMP_EMPNO
INDEX FAST FULL SCAN	SCOTT.EMP_ENAEM

** table에 access 하지 않는다!!

문제 39. 전공이 통계학이고 나이가 20대인 학생의 학생번호, 이름, 전공, 나이를 출력하는 쿼리를 출력하는데 쿼리의 성능이 높아지도록 적절하게 인덱스를 생성하고 인덱스를 잘 탈 수 있도록 힌트를 사용해서 SQL을 작성 하시오.

```
SELECT empno, ename, major, age
      FROM EMP2
   WHERE major = '통계학' AND age LIKE '2%';
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
INDEX FULL SCAN	SCOTT.EM2_EMPNO_ENAME_MAJOR_AGE

1	recursive calls	0
2	db block gets	0
3	consistent gets	59

```
SELECT e.empno, e.ename, e.major, e.age
      FROM EMP2 e
   WHERE major = '통계학' AND age LIKE '2%';
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	-
INDEX FULL SCAN	SCOTT.EM2_EMPNO_ENAME_MAJOR_AGE

recursive calls	0
db block gets	0
consistent gets	1

** 테이블 별칭을 줘서 컬럼마다 테이블 별칭을 사용하면 아주 미세하게 빨라진다 (실제 데이터에선 별 의미 없다)

-튜닝후-

```
SELECT empno, ename, major, age
       FROM EMP2
   WHERE major = '통계학' AND age BETWEEN 20 AND 29;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	-
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP2
INDEX RANGE SCAN	SCOTT.EMP2_MAJOR

1	recursive calls	0
2	db block gets	0
3	consistent gets	60

문제 40. 아래의 SQL을 튜닝 하시오.

```
CREATE INDEX emp2_ename ON EMP2(ename);
```

-튜닝 전-

```
SELECT ename, age, major FROM EMP2 WHERE SUBSTR(ename,1,1) = '김';
```

SELECT STATEMENT Optimizer Mode=ALL_ROWS	
TABLE ACCESS FULL	SCOTT.EMP2

-튜닝 후-

```
SELECT ename, age, major FROM EMP2 WHERE ename like '김%';
```

SELECT STATEMENT Optimizer Mode=ALL_ROWS	
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP2
INDEX RANGE SCAN	SCOTT.EMP2_ENAME

문제 41. 아래의 SQL을 튜닝 하시오.

```
CREATE INDEX emp2_birth
        ON EMP2(birth);
```

-튜닝 전-

```
SELECT ename, age, major
        FROM EMP2
    WHERE TO_CHAR(birth, 'RR/MM/DD') = '92/05/25';
```

SELECT STATEMENT Optimizer Mode=ALL_ROWS	
TABLE ACCESS FULL	SCOTT.EMP2

-튜닝 후-

```
SELECT ename , age, major
            FROM EMP2
        WHERE birth = TO_DATE('92/05/25','RR/MM/DD');
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP2
INDEX RANGE SCAN	SCOTT.EMP2_BIRTH

9. join 튜닝

2018년 5월 1일 화요일 오후 2:18

-- 현업에서는 INDEX 튜닝보다 JOIN 튜닝할 일이 많다..

9.1 조인 SQL 문법의 종류

1. 오라클 조인 문법
   (1) equi join
   (2) non equi join
   (3) outer join
   (4) Self join

2.1999 ANSI 문법
   (1) on 절을 사용한 조인
   (2) using 절을 사용한 조인
   (3) left / right / full outer 조인
   (4) natural 조인
   (5) cross 조인

9.2 조인을 실행하는 3가지 방법

1.nested loop join : 조인되는 건수가 적을 때 유리함

2. hash join : 조인되는 건수가 많을 때 사용 (대용량의 데이터)

3.sort merge join : hash 조인으로 처리가 안되는 non-equi 조인 조건일 때 사용

9.3 예제

문제 43. 이름, 월급, 부서위치를 출력 하시오.

```
SELECT ename, sal, LOC
      FROM EMP e, DEPT d
        WHERE e.DEPTNO = d.DEPTNO;
```

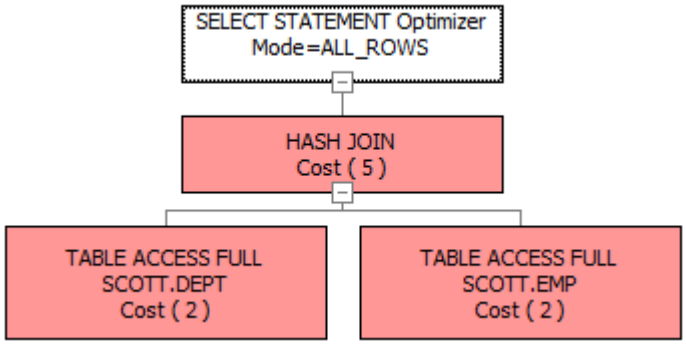
	ENAME	SAL	LOC
1	KING	5000	NEW YORK
2	BLAKE	2850	CHICAGO
3	CLARK	2450	NEW YORK
4	JONES	2975	DALLAS
5	MARTIN	1250	CHICAGO
6	ALLEN	1600	CHICAGO
7	TURNER	1500	CHICAGO
8	JAMES	950	CHICAGO
9	WARD	1250	CHICAGO
10	FORD	3000	DALLAS
11	SMITH	800	DALLAS
12	SCOTT	3000	DALLAS
13	ADAMS	1100	DALLAS
14	MILLER	1300	NEW YORK

문제 44. 위의 SQL의 테이블 중 어떤 테이블을 먼저 읽어올까?

1. emp -----> dept ?
2. dept -----> emp ? 어떤 테이블을 먼저 읽을까??? --> 실행계획으로 확인해보자.

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
HASH JOIN	
1 TABLE ACCESS FULL	SCOTT.DEPT
2 TABLE ACCESS FULL	SCOTT.EMP

** dept 테이블을 먼저 읽는다. (DEPT ----> EMP)



	ENAME	SAL	JOB	DEPTNO
1	KING	5000	PRESIDENT	10
2	BLAKE	2850	MANAGER	30
3	CLARK	2450	MANAGER	10
4	JONES	2975	MANAGER	20
5	MARTIN	1250	SALESMAN	30
6	ALLEN	1600	SALESMAN	30
7	TURNER	1500	SALESMAN	30
8	JAMES	950	CLERK	30
9	WARD	1250	SALESMAN	30
10	FORD	3000	ANALYST	20
11	SMITH	800	CLERK	20
12	SCOTT	3000	ANALYST	20
13	ADAMS	1100	CLERK	20
14	MILLER	1300	CLERK	10

	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON

EMP ----> DEPT = 조인 시도를 14번 했다.

LOC	DNAME	DEPTNO
NEW YORK	ACCOUNTING	10
DALLAS	RESEARCH	20
CHICAGO	SALES	30
BOSTON	OPERATIONS	40

Dept -----> EMP = 조인 4번 실행

	DEPTNO	ENAME	SAL	JOB
1	10	KING	5000	PRESIDENT
2	30	BLAKE	2850	MANAGER
3	10	CLARK	2450	MANAGER
4	20	JONES	2975	MANAGER
5	30	MARTIN	1250	SALESMAN
6	30	ALLEN	1600	SALESMAN
7	30	TURNER	1500	SALESMAN
8	30	JAMES	950	CLERK
9	30	WARD	1250	SALESMAN
10	20	FORD	3000	ANALYST
11	20	SMITH	800	CLERK
12	20	SCOTT	3000	ANALYST
13	20	ADAMS	1100	CLERK
14	10	MILLER	1300	CLERK

★ 조인 튜닝시에 가장 중요한 2가지

1.조인순서

-**ordered** : from 절에서 기술한 테이블 순서대로 조인

-**Leading** : leading 힌트 안에 쓴 테이블 순서대로 조인

2.조인방법

-**use_nl** : nested loop 조인으로 조인해라

-**use_hash** : hash 조인으로 조인해라

-**use_merge** : sort merge 조인으로 조인해라

10. nested loop join

2018년 5월 1일 화요일 오후 2:18

" 두개의 테이블을 조인할 때 하나 하나 순차적으로 반복해서 조인하는 조인 방법 "

----> 조인되는 건수가 적을 때 유리한 방법이다 !!

10.1 예제

Ex)

LOC	DNAME	DEPTNO
NEW YORK	ACCOUNTING	10
DALLAS	RESEARCH	20
CHICAGO	SALES	30
BOSTON	OPERATIONS	40

Dept -----> EMP = 조인 4번 실행

	DEPTNO	ENAME	SAL	JOB
1	10	KING	5000	PRESIDENT
2	30	BLAKE	2850	MANAGER
3	10	CLARK	2450	MANAGER
4	20	JONES	2975	MANAGER
5	30	MARTIN	1250	SALESMAN
6	30	ALLEN	1600	SALESMAN
7	30	TURNER	1500	SALESMAN
8	30	JAMES	950	CLERK
9	30	WARD	1250	SALESMAN
10	20	FORD	3000	ANALYST
11	20	SMITH	800	CLERK
12	20	SCOTT	3000	ANALYST
13	20	ADAMS	1100	CLERK
14	10	MILLER	1300	CLERK

문제 45. ordered 힌트를 이용해서 아래 SQL의 조인 순서를 EMP---->DEPT 순으로 조인되게 하시오.

```
SELECT /*+ ordered */ e.ename, e.sal, d.loc
        FROM EMP e, DEPT d
           WHERE e.deptno = d.DEPTNO;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
HASH JOIN	
TABLE ACCESS FULL	1 SCOTT.EMP
TABLE ACCESS FULL	2 SCOTT.DEPT

문제 46. 아래의 SQL을 다음 방법으로 조인 하시오.

조인순서 : EMP -----> DEPT
조인방법 : nested loop join

```
SELECT /*+ ordered use_nl(d) */ e.ename, e.sal, d.loc
        FROM EMP e, DEPT d
    WHERE e.deptno = d.DEPTNO;
```

1	recursive calls	0

1	recursive calls	0
2	db block gets	0
3	consistent gets	45

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
NESTED LOOPS	
TABLE ACCESS FULL	SCOTT.EMP
TABLE ACCESS FULL	SCOTT.DEPT

문제 47. 아래의 JOIN SQL을 튜닝 하시오.

-튜닝 전-

```
SELECT /*+ ordered use_nl(d) */ e.ename, e.sal, d.loc
        FROM EMP e, DEPT d
         WHERE e.deptno = d.DEPTNO;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
NESTED LOOPS	
TABLE ACCESS FULL	SCOTT.EMP
TABLE ACCESS FULL	SCOTT.DEPT

1	recursive calls	0
2	db block gets	0
3	consistent gets	45

-튜닝 후-

```
SELECT /*+ ordered use_nl( e ) */ e.ename, e.sal, d.loc
        FROM DEPT d, EMP e        -- ordered : from절의 테이블 순서대로 조인
    WHERE e.deptno = d.DEPTNO;
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		14
NESTED LOOPS		
NESTED LOOPS		14
TABLE ACCESS FULL	SCOTT.DEPT	4
INDEX RANGE SCAN	SCOTT.EMP_DEPTNO	5
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP	4

1	recursive calls	0
2	db block gets	0
3	consistent gets	6

문제 48. 아래의 조인 SQL에 적절한 힌트를 주시오.

```
SELECT /*+ use_nl(e) */e.ename, e.sal, d.loc
        FROM EMP e, DEPT d
        WHERE e.DEPTNO = d.DEPTNO AND e.job = 'SALESMAN'   -- 4건
        AND d.loc = 'CHICAGO';  -- 1건


SELECT COUNT(*) FROM EMP WHERE job = 'SALESMAN';  --- 4건
```

SELECT COUNT(*) FROM DEPT WHERE loc = 'CHICAGO'; --- 1건 조인하므로 DEPT 테이블 부터 조인하는게 좋다

SELECT **/*+ leading(d e) use_nl(e) */** e.ename, e.sal, d.loc -- leading 을 쓰면 from 절은 안건들여도 된다.
FROM EMP e, DEPT d
WHERE e.DEPTNO = d.DEPTNO AND e.job = 'SALESMAN' -- 4건
AND d.loc = 'CHICAGO'; -- 1건

문제 49. sk텔레콤 통신사인 학생들의 이름, 주소, 나이, 통신사, 통신사 월 정액을 출력 하시오.

SELECT /*+ leading(t,e) use_nl(e) */e.ename, e.address,e.telecom, t.t_price
FROM EMP2 e, TELECOM_PRICE t
WHERE e.TELECOM = t.TELECOM AND **t.telecom = 'sk'**;

★ t.telecom = 'sk' 을 e.telecom = 'sk' 로 쓸 수 있지만 그러면 조인하는 수가 많아져서 성능 저하..

	TELECOM	T_PRICE	T_PROFIT
1	sk	60000	0
2	lg	65000	3
3	kt	64000	2

	TELECOM	ENAME
1	sk	김원섭
2	sk	이유진
3	sk	윤진민
4	sk	김대경
5	sk	김동윤
6	sk	윤동환
7	sk	김건태
8	sk	한지윤

문제 50. price 테이블과 market_code와 조인해서 a_name, a_price, m_type_name 을 출력 하시오.

-비교-

(1) hash join

SELECT p.a_name, p.a_price, m.m_type_name
FROM PRICE p, market_code m
WHERE p.M_TYPE_CODE = m.M_TYPE_CODE;

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
HASH JOIN	
TABLE ACCESS FULL	SCOTT.MARKET_CODE
TABLE ACCESS FULL	SCOTT.PRICE

1	recursive calls	0
2	db block gets	0
3	consistent gets	10

(2) nested join

SELECT /*+ leading(m p) use_nl(p) */p.a_name, p.a_price, m.m_type_name
FROM PRICE p, market_code m
WHERE p.M_TYPE_CODE = m.M_TYPE_CODE;

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
NESTED LOOPS	
TABLE ACCESS FULL	SCOTT.MARKET_CODE
TABLE ACCESS FULL	SCOTT.PRICE

1	recursive calls	0
2	db block gets	0
3	consistent gets	10

(3) 조인순서를 반대로 바꿨을 때 (price ----> market_code)

```
SELECT /*+ leading(p m ) use_nl(m)  */p.a_name, p.a_price, m.m_type_name
        FROM PRICE p, market_code m
          WHERE p.M_TYPE_CODE = m.M_TYPE_CODE;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
NESTED LOOPS	
TABLE ACCESS FULL	SCOTT.PRICE
TABLE ACCESS FULL	SCOTT.MARKET_CODE

1	recursive calls	0
2	db block gets	0
3	consistent gets	234

문제 51. 이름이 SCOTT인 사원의 이름, 월급, 부서위치를 출력하는데 적절한 조인순서와 조인방법을 힌트를 주어서 작성 하시오.

```
SELECT /*+ leading (e d) use_nl(d) */e.ename, e.sal, d.loc
      FROM EMP e, DEPT d
    WHERE e.DEPTNO = d.DEPTNO AND e.ename = 'SCOTT';
```

* e테이블 부터 조인 해야된다 ---> e는 scott 하나만 비교하면 되지만 d부터 하면 4개를 비교 해야된다.

SELECT STATEMENT Optimizer Mode=ALL_ROWS	
NESTED LOOPS	
TABLE ACCESS FULL	SCOTT.EMP
TABLE ACCESS FULL	SCOTT.DEPT

문제 52. price 테이블과 gu_code와 조인을 해서 a_name, a_price, m_gu_name을 출력 하시오.

	P_SEQ	M_SEQ	M_NAME	A_SEQ	A_NAME	A_UNIT	A_PRICE	P_YEAR_MONTH	ADD_COL	M_TYPE_CODE	M_GU_CODE
1	421105	11	남대문시장	170	배(신고),중급(대)	1개 580g	2000	2013-03	안성배	001	140000
2	421106	11	남대문시장	307	배추(2.5~3kg)	1포기`	6000	2013-03	해남	001	140000
3	421107	11	남대문시장	282	무(세척무)	1개 1.4kg	1000	2013-03	제주	001	140000
4	421108	11	남대문시장	309	양파(1.5kg망)	1망	4500	2013-03	전남 무안	001	140000
5	421109	11	남대문시장	310	상추(100g)	1봉지	500	2013-03	이천용곡농장	001	140000
6	421110	11	남대문시장	311	오이(다다기)	1개	700	2013-03	미금농협	001	140000
7	421111	11	남대문시장	118	호박(인큐베이터)	1 ㅐ	1500	2013-03	논개애호박	001	140000
8	421112	11	남대문시장	98	쇠고기(한우2등급	1근	18000	2013-03	한우 앞다리	001	140000
9	421113	11	남대문시장	99	돼지고기(생삼겹	1근	9000	2013-03	축협음성공판	001	140000
10	421114	11	남대문시장	283	닭고기(육계)	1마리 1kg	4500	2013-03	동두천	001	140000

[↑ price 테이블]

	M_GU_CODE	M_GU_NAME
1	590000	동작구
2	380000	은평구
3	620000	관악구
4	680000	강남구
5	170000	용산구
6	305000	강북구
7	320000	도봉구
8	200000	성동구
9	545000	금천구
10	500000	강서구
11	290000	성북구

[↑ gu_code]

```
SELECT /*+ leading( g p ) use_nl(p) */p.a_name, p.a_price, g.m_gu_name
        FROM PRICE p , GU_CODE g
    WHERE p.M_GU_CODE = g.M_GU_CODE;
```

	A_NAME	A_PRICE	M_GU_NAME
1	사과(부사, 300g)	1500	동작구
2	배(신고, 600g)	3000	동작구
3	배추(국산)	4150	동작구
4	무	1650	동작구
5	양파(작은망)	6980	동작구
6	상추	1250	동작구
7	오이(다다기)	750	동작구
8	애호박	1750	동작구
9	쇠고기(한우,불고	13200	동작구
10	돼지고기(생삼겹	9300	동작구
11	닭고기(육계)	6300	동작구

***조건이 따로 없기 때문에 로우의 수가 적은(조인되는 건수가 적은) 테이블 부터 엑세스한다.**

```
SELECT STATEMENT Optimizer Mode=ALL_ROWS
  NESTED LOOPS
    TABLE ACCESS FULL        SCOTT.GU_CODE
    TABLE ACCESS FULL        SCOTT.PRICE
```

문제 53. price 테이블과 gu_code와 조인을 해서 이마트 역삼점의 a_name, a_price, m_gu_name을 출력 하시오.

```
SELECT /*+ leading( g p ) use_nl(p) */ p.a_name, p.a_price, g.m_gu_name
        FROM PRICE p, GU_CODE g
    WHERE p.M_GU_CODE = g.M_GU_CODE AND p.M_NAME = '이마트 역삼점';
```

	A_NAME	A_PRICE	M_GU_NAME
1	사과(부사, 300g)	1725	강남구
2	배(신고, 600g)	4600	강남구
3	배추(2.5~3kg)	1980	강남구
4	무(1kg)	1480	강남구
5	양파(작은망)	5380	강남구
6	상추(100g)	990	강남구
7	오이(다다기)	696	강남구
8	호박(인큐베이터	1980	강남구
9	쇠고기(한우,불고	16800	강남구
10	돼지고기(생삼겹	6000	강남구
11	닭고기(육계)	6980	강남구

문제 54. 이름, 월급, 부서위치, 급여등급을 출력 하시오. (dept - emp - salgrade)

```
SELECT /*+ leading(d  s e ) use_nl(e) use_nl(s)   */ e.ename, e.sal, d.loc, s.grade
      FROM EMP e, DEPT d, SALGRADE s
    WHERE e.DEPTNO = d.DEPTNO AND e.sal BETWEEN s.losal AND s.HISAL;
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
NESTED LOOPS		1
NESTED LOOPS		20
TABLE ACCESS FULL	SCOTT.DEPT	4
TABLE ACCESS FULL	SCOTT.SALGRADE	5
TABLE ACCESS FULL	SCOTT.EMP	1

문제 55. price 테이블, gu_code, market_code 3개 테이블을 조인해서 a_name, a_price, m_gu_name, m_type_name을 출력하는데 적절한 조인순서와 힌트를 주고 실행 하시오.

```
SELECT /*+ leading( m p g ) use_nl(p) use_nl(g) */p.a_name, p.a_price, g.m_gu_name, m.m_type_name
      FROM PRICE p, GU_CODE g, MARKET_CODE m
    WHERE p.M_TYPE_CODE = m.M_TYPE_CODE AND p.M_GU_CODE = g.M_GU_CODE;
```

M (2) ------ p (9516) -------- g (4)

	A_NAME	A_PRICE	M_GU_NAME	M_TYPE_NAME
1	사과(부사, 300g)	2300	강동구	대형마트
2	배(신고, 600g)	3933	강동구	대형마트
3	배추(2.5~3kg)	2000	강동구	대형마트
4	무(1kg)	1000	강동구	대형마트
5	양파	4360	강동구	대형마트
6	상추(100g)	980	강동구	대형마트
7	오이(다다기)	750	강동구	대형마트
8	애호박	1500	강동구	대형마트
9	쇠고기(한우,불고	16800	강동구	대형마트
10	돼지고기(생삼겹	4740	강동구	대형마트
11	닭고기(육계)	6580	강동구	대형마트
12	달걀(왕란)	2300	강동구	대형마트
13	조기(국산,냉동)	1000	강동구	대형마트

Operation	Object Name	Rows
6 SELECT STATEMENT Optimizer Mode=ALL_ROWS	-	9 K
5 NESTED LOOPS		9 K
3 NESTED LOOPS		9 K
1 TABLE ACCESS FULL	SCOTT.MARKET_CODE	2
2 TABLE ACCESS FULL	SCOTT.PRICE	4 K
4 TABLE ACCESS FULL	SCOTT.GU_CODE	1

문제 56. 급여등급이 2등급인 사원들의 이름, 월급 ,부서위치, 급여등급을 출력하는데, 적절한 조인순서와 조인방법을 써서 구현 하시오.

```
SELECT /*+ leading(s e d) use_nl(e) use_nl(d) */ e.ename, e.sal, d.loc, s.grade
      FROM EMP e, DEPT d, SALGRADE s
```

```
WHERE e.deptno =d.DEPTNO
      AND e.sal BETWEEN s.LOSAL AND s.HISAL AND s.GRADE = 2;
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
NESTED LOOPS		1
NESTED LOOPS		1
TABLE ACCESS FULL	SCOTT.SALGRADE	1
TABLE ACCESS FULL	SCOTT.EMP	1
TABLE ACCESS FULL	SCOTT.DEPT	1

	ENAME	SAL	LOC	GRADE
1	MARTIN	1250	CHICAGO	2
2	WARD	1250	CHICAGO	2
3	MILLER	1300	NEW YORK	2

문제 57. 아래 SQL을 튜닝 하시오.

-튜닝 전-

```
SELECT /*+ leading( s t ) use_nl(t)  */ t.calendar_year, SUM(s.amount_sold)
      FROM sales200 s, times200 t
        WHERE s.time_id = t.time_id
              AND t.week_ending_day_id = 1581
       GROUP BY t.CALENDAR_YEAR;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	50540806

-튜닝 후-

```
SELECT /*+ leading( t s ) use_nl( s )  */ t.calendar_year, SUM(s.amount_sold)
      FROM sales200 s, times200 t
        WHERE s.time_id = t.time_id
         AND t.week_ending_day_id = 1581
        GROUP BY t.CALENDAR_YEAR;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	31142

문제 58. 아래의 SQL을 튜닝 하시오.

튜닝전 :

```
SELECT /*+ leading (s c) use_nl(c) */ COUNT(*)
        FROM sales200 s, customers200 c
      WHERE s.cust_id = c.cust_id AND c.country_id = 52790
            AND s.time_id BETWEEN TO_DATE('1999/01/01','RRRR/MM/DD')
        AND TO_DATE ('1999/12/31','RRRR/MM/DD');
```

Select	1 / 1	1107 초	60 : 13	삽입	행	ANSI-DOS	자동 추적	(0 x 0)	가로 그리드	1 개 행이 선택되었습니다.

1	recursive calls	92
2	db block gets	0
3	consistent gets	361508396

```
SELECT COUNT(*)
        FROM customers200
      WHERE COUNTRY_ID = 52790;
```

	COUNT(*)
1	18520

```
SELECT COUNT(*)
        FROM sales200
      WHERE time_id BETWEEN TO_DATE('1999/01/01','RRRR/MM/DD') AND TO_DATE
('1999/12/31','RRRR/MM/DD');
```

	COUNT(*)
1	247945

```
SELECT /*+ leading (c s) use_hash (s) */ COUNT(*)
      FROM sales200 s, customers200 c
    WHERE s.cust_id = c.cust_id AND c.country_id = 52790
        AND s.time_id BETWEEN TO_DATE('1999/01/01','RRRR/MM/DD') AND TO_DATE
('1999/12/31','RRRR/MM/DD');
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	5899

문제 59. 아래의 SQL을 nested loop 조인으로 수행하면서 좋은 성능을 보이게끔 인덱스를 생성 하시오.

-튜닝 전-

```
SELECT /*+ leading (c s) use_nl (s) */ COUNT(*)
      FROM sales200 s, customers200 c
    WHERE s.cust_id = c.cust_id AND c.country_id = 52790
        AND s.time_id BETWEEN TO_DATE('1999/01/01','RRRR/MM/DD') AND TO_DATE
('1999/12/31','RRRR/MM/DD');
```

-튜닝 후-

```
CREATE INDEX sales200_cust_id
      ON sales200(cust_id);

CREATE INDEX customers200_cust_id
    ON customers200(cust_id);                    --- 인덱스를 걸어준다.

SELECT /*+ leading (c s) use_nl(s) */ COUNT(*)
      FROM sales200 s, customers200 c
      WHERE s.cust_id = c.cust_id AND c.country_id = 52790
        AND s.time_id BETWEEN TO_DATE('1999/01/01','RRRR/MM/DD') AND TO_DATE
('1999/12/31','RRRR/MM/DD');
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	456698

Operation	Object Name	Rows
8 SELECT STATEMENT Optimizer Mode=ALL_ROWS	-	1
7 SORT AGGREGATE		1
6 FILTER		
5 NESTED LOOPS		
3 NESTED LOOPS		236 K
1 TABLE ACCESS FULL	SCOTT.CUSTOMERS200	18 K
2 INDEX RANGE SCAN	SCOTT.SALES200_CUST_ID	162
4 TABLE ACCESS BY INDEX ROWID	SCOTT.SALES200	13

++ customers200_country_id 인덱스를 추가하면

```
CREATE INDEX customers200_country_id ON customers200(country_id);

SELECT /*+ leading (c s) use_nl(s) */ COUNT(*)
        FROM sales200 s, customers200 c
        WHERE s.cust_id = c.cust_id AND c.country_id = 52790
            AND s.time_id BETWEEN TO_DATE('1999/01/01','RRRR/MM/DD') AND TO_DATE
('1999/12/31','RRRR/MM/DD');
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS	-	1
SORT AGGREGATE		1
FILTER		
NESTED LOOPS		
NESTED LOOPS		236 K
VIEW	SCOTT.index$_join$_002	18 K
HASH JOIN		
INDEX RANGE SCAN	SCOTT.CUSTOMERS200_COUNTRY_ID	18 K
INDEX FAST FULL SCAN	SCOTT.CUSTOMERS200_CUST_ID	18 K
INDEX RANGE SCAN	SCOTT.SALES200_CUST_ID	162
TABLE ACCESS BY INDEX ROWID	SCOTT.SALES200	13

1	recursive calls	0
2	db block gets	0
3	consistent gets	428646

--- 인덱스를 추가하니 더 빨라졌다.

문제 60. 아래 SQL을 nested loop 조인으로 수행하면서 좋은 성능을 보이게끔 인덱스를 생성 하시오.

-튜닝 전-

```
SELECT /*+ leading (t s) use_nl(s) */t.CALENDAR_YEAR, sum(s.amount_sold)
        FROM sales200 s, times200 t
        WHERE s.time_id = t.TIME_id AND t.WEEK_ENDING_DAY_ID = 1581
        GROUP BY t.CALENDAR_year;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	31142

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		11 K
HASH GROUP BY		11 K
NESTED LOOPS		11 K
TABLE ACCESS FULL	SCOTT.TIMES200	7
TABLE ACCESS FULL	SCOTT.SALES200	1 K

-튜닝 후-

```
CREATE INDEX times200_time_id ON times200(time_id);
        CREATE INDEX sales_time_id ON sales200(time_id);
        CREATE INDEX times200_week_ending_day_id ON times200(week_ending_day_id);
```

--- 인덱스를 3개 추가해준다.

```
SELECT /*+ leading (t s) use_nl(s) */t.CALENDAR_YEAR, sum(s.amount_sold)
        FROM sales200 s, times200 t
        WHERE s.time_id = t.TIME_id AND t.WEEK_ENDING_DAY_ID = 1581
        GROUP BY t.CALENDAR_year;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	286

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		11 K
HASH GROUP BY		11 K
NESTED LOOPS		
NESTED LOOPS		11 K
TABLE ACCESS BY INDEX ROWID	SCOTT.TIMES200	7
INDEX RANGE SCAN	SCOTT.TIMES200_WEEK_ENDING_DAY_ID	7
INDEX RANGE SCAN	SCOTT.SALES_TIME_ID	1 K
TABLE ACCESS BY INDEX ROWID	SCOTT.SALES200	1 K

문제 61. 아래의 SQL을 튜닝 하시오.

-튜닝 전-

```
SELECT /*+ leading(s t p) use_nl(t) use_nl(p) */p.prod_name, t.calendar_year, SUM(s.amount_sold)
            FROM sales200 s, times200 t, products200 p
        WHERE s.TIME_ID = t.TIME_ID
                AND s.PROD_ID = p.prod_id
            AND t.CALENDAR_YEAR IN (2000, 2001)
            AND p.prod_name LIKE 'Deluxe%'
      GROUP BY p.prod_name, t.calendar_year;
```

1	recursive calls	10
2	db block gets	0
3	consistent gets	1996977

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		10 K
HASH GROUP BY		10 K
NESTED LOOPS		10 K
NESTED LOOPS		785 K
TABLE ACCESS FULL	SCOTT.SALES200	785 K
TABLE ACCESS BY INDEX ROWID	SCOTT.TIMES200	1
INDEX RANGE SCAN	SCOTT.TIMES200_TIME_ID	1
TABLE ACCESS FULL	SCOTT.PRODUCTS200	1

-튜닝 후-

```
CREATE INDEX products200_prod_id ON products200(prod_id);
        CREATE INDEX products200_prod_name ON products200(prod_name);
        CREATE INDEX times200_calendar_year ON times200(calendar_year);
       CREATE INDEX sales200_prod_id ON sales200(prod_id);
       CREATE INDEX sales200_time_id ON sales200(time_id);
       CREATE INDEX times200_time_id ON times200(time_id);
```

-- 인덱스를 다 걸어준다.

```
SELECT /*+ leading(p s t ) use_nl(s) use_nl(t) */p.prod_name, t.calendar_year, SUM(s.amount_sold)
        FROM sales200 s, times200 t, products200 p
        WHERE s.TIME_ID = t.TIME_ID
                AND s.PROD_ID = p.prod_id
            AND t.CALENDAR_YEAR IN (2000, 2001)
            AND p.prod_name LIKE 'Deluxe%'
      GROUP BY p.prod_name, t.calendar_year;
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		10 K
HASH GROUP BY		10 K
NESTED LOOPS		
NESTED LOOPS		10 K
NESTED LOOPS		10 K
TABLE ACCESS BY INDEX ROWID	SCOTT.PRODUCTS200	1
INDEX RANGE SCAN	SCOTT.PRODUCTS200_PROD_NAME	1
TABLE ACCESS BY INDEX ROWID	SCOTT.SALES200	10 K
INDEX RANGE SCAN	SCOTT.SALES200_PROD_ID	68
INDEX RANGE SCAN	SCOTT.TIMES200_TIME_ID	1
TABLE ACCESS BY INDEX ROWID	SCOTT.TIMES200	1

1	recursive calls	0
2	db block gets	0
3	consistent gets	439

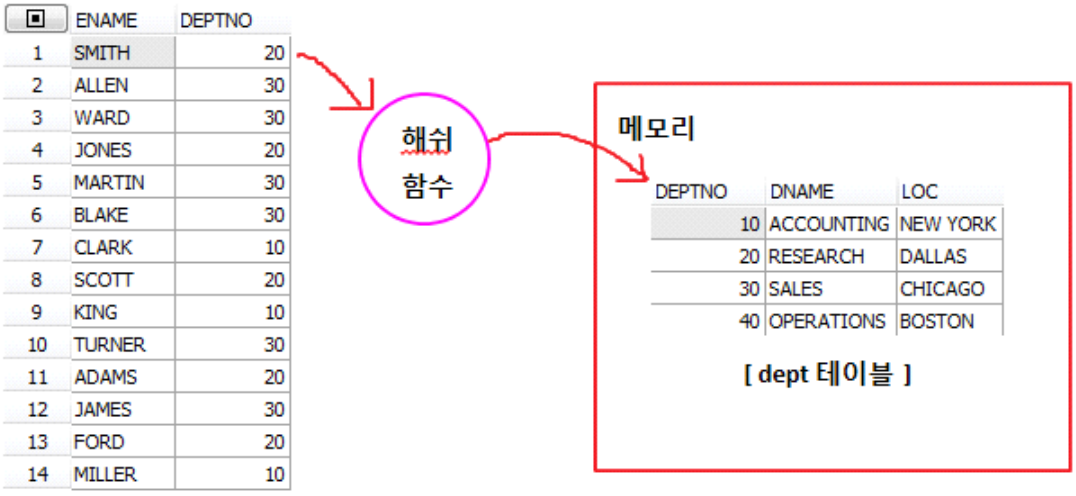
11. hash join

2018년 5월 1일 화요일 오후 2:19

11.1 해쉬조인의 원리

```
ex) SELECT /*+ leading(d e) use_hash(e) */ e.ename, d.loc
          FROM EMP e, DEPT d
        WHERE e.DEPTNO = d.DEPTNO;
```

***해쉬조인**을 하게 되면 **dept 테이블을 메모리(작은 테이블 = 먼저 엑세스하는 테이블)**에 올려놓고 **메모리에 있는 dept 테이블과 emp와 조인**을 시도한다.



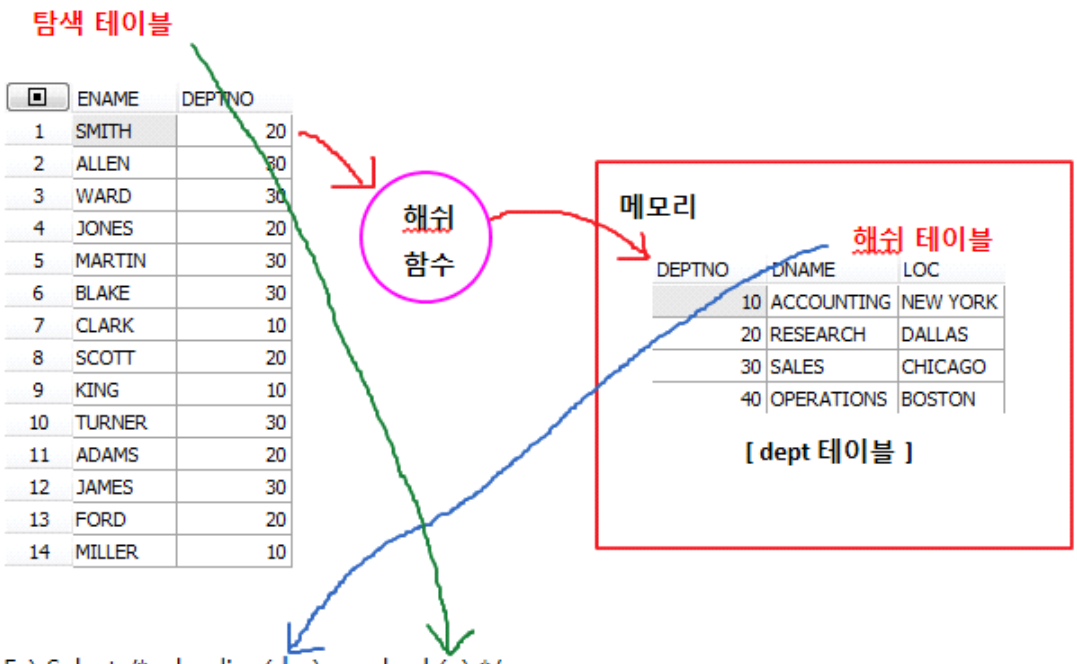
***테이블 크기가 작거나** where **조건에서 검색되는 결과가 적은 테이블**을 메모리에 올린다.
*해쉬테이블 = 메모리에 올리는 테이블

11.2 해쉬조인 사용시 힌트

1.use_hash (테이블명) : 해쉬조인해라 ~
2.swap_join_inputs : **해쉬 테이블**을 선정할 때 사용하는 힌트
3.no_swap_join_inputs : **prob 테이블**을 선정할 때 사용하는 힌트

* 해쉬 테이블과 prob 테이블

- 해쉬 테이블 : 메모리에 올라가는 테이블
- 탐색 테이블 : 디스크에서 메모리에 있는 해쉬 테이블과 조인하는 테이블



	ENAME	DEPTNO
1	SMITH	20
2	ALLEN	30
3	WARD	30
4	JONES	20
5	MARTIN	30
6	BLAKE	30
7	CLARK	10
8	SCOTT	20
9	KING	10
10	TURNER	30
11	ADAMS	20
12	JAMES	30
13	FORD	20
14	MILLER	10

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
Ex) Select /*+ leading(d e) use_hash(e) */
      e.ename, d.loc
      From emp e, dept d
      Where e.deptno = d.deptno;
```

---> 위와 같이 테이블이 2개면 leading 힌트 만으로도 해쉬 테이블과 탐색 테이블을 선정할 수 있다.

---> 테이블이 3개면 leading 힌트만으로 해쉬 테이블과 탐색 테이블을 선정하기 어렵다.
그래서 필요한 힌트가 swap_join_inputs와 no_swap_join_inputs이다.

```
SELECT /*+ leading( d e b ) use_hash(e) use_hash(b) */ e.ename, d.loc, b.BONUS
      FROM EMP e, DEPT d, BONUS b
        WHERE e.empno = b.EMPNO AND e.DEPTNO = d.DEPTNO;
```

SELECT STATEMENT Optimizer Mode=ALL_ROWS		14
HASH JOIN		14
HASH JOIN		14
TABLE ACCESS FULL	SCOTT.DEPT	4
TABLE ACCESS FULL	SCOTT.EMP	14
TABLE ACCESS FULL	SCOTT.BONUS	14

11.3 해쉬 조인시 병렬작업 하는 방법

*Hash join 할 때는 **table full 스캔이 index 스캔보다 빠르다.**

예제 1. 이름과 부서위치를 출력하는데 해쉬 조인으로 수행되게 하시오.

```
SELECT /*+ leading( d e ) use_hash(e)*/ e.ename, d.loc
      FROM EMP e, DEPT d
        WHERE e.DEPTNO = d.DEPTNO;
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS	-	14
HASH JOIN		14
TABLE ACCESS FULL	SCOTT.DEPT	4
TABLE ACCESS FULL	SCOTT.EMP	14

1	recursive calls	0
2	db block gets	0
3	consistent gets	14

11.4 예제

문제 63. 아래의 SQL을 해쉬조인으로 수행 하시오.

```
SELECT /*+ leading(p s t ) use_nl(s) use_nl(t) */p.prod_name, t.calendar_year, SUM(s.amount_sold)
        FROM sales200 s, times200 t, products200 p
        WHERE s.TIME_ID = t.TIME_ID
                AND s.PROD_ID = p.prod_id
            AND t.CALENDAR_YEAR IN (2000, 2001)
            AND p.prod_name LIKE 'Deluxe%'
      GROUP BY p.prod_name, t.calendar_year;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	439

```
SELECT /*+ leading(p s t ) use_hash(s) use_hash(t) */p.prod_name, t.calendar_year, SUM(s.amount_sold)
            FROM sales200 s, times200 t, products200 p
        WHERE s.TIME_ID = t.TIME_ID
            AND s.PROD_ID = p.prod_id
            AND t.CALENDAR_YEAR IN (2000, 2001)
            AND p.prod_name LIKE 'Deluxe%'
                    GROUP BY p.prod_name, t.calendar_year;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	4471

*해쉬조인을 쓴다고 무조건 빨라지는건 아니다.

문제 64. 아래 SQL을 hash 조인으로 수행하는데 times 테이블을 해쉬 테이블로 구성하는 것과 sales 테이블을 해쉬 테이블로 구성하는 것의 성능 차이를 확인 하시오.

-SQL-

```
SELECT /*+ leading (t s) use_nl(s) */t.CALENDAR_YEAR, sum(s.amount_sold)
        FROM sales200 s, times200 t
        WHERE s.time_id = t.TIME_id AND t.WEEK_ENDING_DAY_ID = 1581
        GROUP BY t.CALENDAR_year;
```

```
SELECT /*+ leading (t s) use_hash(s) */ t.CALENDAR_YEAR, sum(s.amount_sold)
        FROM sales200 s, times200 t
        WHERE s.time_id = t.TIME_id AND t.WEEK_ENDING_DAY_ID = 1581
        GROUP BY t.CALENDAR_year;
```

문제 65. 아래의 SQL을 조인 순서와 조인방법을 아래와 같이 설정하시오.

조인순서 : times -> sales -> products
조인방법 :　　해쉬조인　해쉬조인

```
SELECT /*+ leading(p s t ) use_hash(s) use_hash(t) */p.prod_name, t.calendar_year, SUM(s.amount_sold)
            FROM sales200 s, times200 t, products200 p
        WHERE s.TIME_ID = t.TIME_ID
            AND s.PROD_ID = p.prod_id
          AND t.CALENDAR_YEAR IN (2000, 2001)
          AND p.prod_name LIKE 'Deluxe%'
                GROUP BY p.prod_name, t.calendar_year;
```

```
SELECT /*+ leading (t s p) use_hash(s) use_hash(p) */ p.prod_name, t.calendar_year, SUM(s.amount_sold)
                FROM sales200 s, times200 t, products200 p
            WHERE s.TIME_ID = t.TIME_ID
                AND s.PROD_ID = p.prod_id
              AND t.CALENDAR_YEAR IN (2000, 2001)
              AND p.prod_name LIKE 'Deluxe%'
            GROUP BY p.prod_name, t.calendar_year;
```

Operation	Object Name	Rows
7 SELECT STATEMENT Optimizer Mode=ALL_ROWS		10 K
6 HASH GROUP BY		10 K
5 HASH JOIN		10 K
4 TABLE ACCESS FULL	SCOTT.PRODUCTS200	1
3 HASH JOIN		785 K
1 TABLE ACCESS FULL	SCOTT.TIMES200	731
2 TABLE ACCESS FULL	SCOTT.SALES200	785 K

* 메모리 올라가는 순서

문제 66. 아래와 같은 실행계획이 나오게 하시오.

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		10 K
HASH GROUP BY		10 K
HASH JOIN		10 K
HASH JOIN		785 K
TABLE ACCESS FULL	SCOTT.TIMES200	731
TABLE ACCESS FULL	SCOTT.SALES200	785 K
TABLE ACCESS FULL	SCOTT.PRODUCTS200	1

```
SELECT /*+ leading (t s p ) use_hash(s) use_hash(p) no_swap_join_inputs(p) */
```

```
p.prod_name, t.calendar_year, SUM(s.amount_sold)
                FROM sales200 s, times200 t, products200 p
            WHERE s.TIME_ID = t.TIME_ID
                AND s.PROD_ID = p.prod_id
              AND t.CALENDAR_YEAR IN (2000, 2001)
              AND p.prod_name LIKE 'Deluxe%'
          GROUP BY p.prod_name, t.calendar_year;
```

문제 67. 아래의 SQL을 다음과 같이 실행 계획이 나오게 하시오.

```
SELECT  e.ename, d.loc, b.BONUS
      FROM EMP e, DEPT d, BONUS b
        WHERE e.empno = b.EMPNO AND e.DEPTNO = d.DEPTNO;
```

Operation	Object Name	Rows
6 SELECT STATEMENT Optimizer Mode=ALL_ROWS		14
5 HASH JOIN		14
4 TABLE ACCESS FULL 해쉬	SCOTT.BONUS	14
3 HASH JOIN		14
TABLE ACCESS FULL 해쉬	1 SCOTT.DEPT	4
TABLE ACCESS FULL	2 SCOTT.EMP	14

*** 위에 있는 테이블이 해쉬 테이블이다.**

swap_join_inputs 힌트를 써서 테이블을 위로 올림 **(해쉬 테이블을 선정**함 **)**

```
SELECT /*+ leading ( d e b ) swap_join_inputs(b) */ e.ename, d.loc, b.BONUS
                FROM EMP e, DEPT d, BONUS b
        WHERE e.empno = b.EMPNO AND e.DEPTNO = d.DEPTNO;
```

문제 68. 아래와 같이 실행계획이 나오게 하시오.

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		14
HASH JOIN		14
HASH JOIN		14
TABLE ACCESS FULL	SCOTT.BONUS	14
TABLE ACCESS FULL	SCOTT.EMP	14
TABLE ACCESS FULL	SCOTT.DEPT	4

```
SELECT /*+ leading ( b e d )  */ e.ename, d.loc, b.BONUS
      FROM EMP e, DEPT d, BONUS b
            WHERE e.empno = b.EMPNO AND e.DEPTNO = d.DEPTNO;
```

■ **왜 힌트를 써야하나?** ----> 더 좋은 실행계획을 세우기 위해 (계속 테이블의 크기가 바껴서 빨라졌다 느려졌다 함)

```
-- 테이블이 언제 분석작업이 수행됐는지 조회
      SELECT table_name, num_rows, last_analyzed
            FROM user_tables;
```

-- 테이블 emp에 대해 수동으로 분석작업을 수행

ANALYZE TABLE EMP COMPUTE statistics;

문제 69. 아래와 같이 실행계획이 나오게 하시오.

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		14
HASH JOIN		14
NESTED LOOPS		14
TABLE ACCESS FULL	SCOTT.BONUS	14
TABLE ACCESS FULL	SCOTT.EMP	1
TABLE ACCESS FULL	SCOTT.DEPT	4

SELECT /*+ leading (b e d) **use_nl**(e) **use_hash**(d) */ e.ename, d.loc, b.BONUS

FROM EMP e, DEPT d, BONUS b

WHERE e.empno = b.EMPNO AND e.DEPTNO = d.DEPTNO;

문제 70. emp와 salgrade 를 조인해서 이름, 월급, 급여등급을 출력하는데 조인방법이 해쉬 조인이 되게 하시오.

SELECT /*+ leading(s e) **use_hash(e)** */ e.ename, e.sal, s.grade

FROM EMP e, SALGRADE s

WHERE **e.sal BETWEEN s.LOSAL AND s.HISAL**; -- non equi 조인이라서 해쉬 조인이 안된다.

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
NESTED LOOPS		1
TABLE ACCESS FULL	SCOTT.SALGRADE	5
TABLE ACCESS FULL	SCOTT.EMP	1

*hash 조인이 가능 하려면 조인의 연결고리가 이퀄 조건(=)이여야 한다.

---> hash 조인이 불가능한 이퀄 조건은 **sort merge join**을 사용한다.

문제 81. 아래의 해쉬 조인 문장의 full table scan이 병렬로 처리되게 하시오.

SELECT **/*+ leading(d e) use_hash(e) full(e) full(d) parallel(e,4) parallel(d,4)*/**

e.ename, d.loc

FROM EMP e, DEPT d

WHERE e.DEPTNO = d.DEPTNO;

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		14
PX COORDINATOR		
PX SEND QC (RANDOM)	SYS.:TQ10002	14
HASH JOIN BUFFERED		14
PX RECEIVE		4
PX SEND HASH	SYS.:TQ10000	4
PX BLOCK ITERATOR		4
TABLE ACCESS FULL	SCOTT.DEPT	4
PX RECEIVE		14
PX SEND HASH	SYS.:TQ10001	14
PX BLOCK ITERATOR		14
TABLE ACCESS FULL	SCOTT.EMP	14

***full (테이블명) parallel (테이블명, 병렬도) :** 병렬도가 높을 수록 속도 빨라지지만 무작정 높게 주진 말자.
-- Dba한테 물어보고 쓸 것.

12. sort merge join

2018년 5월 1일 화요일 오후 2:20

" 조인의 연결고리가 **이퀄 조건(=)이 아니면서 대용량 테이블**의 조인이어서 해쉬 조인을 사용할 수 없을 때
유리한 조인 방법 "

■ **사용힌트 : use_merge**

12.1 sort merge join 원리

조인하는 **키 컬럼을 미리 정렬 해놓고 조인**한다.

Ex)

	ENAME	DEPTNO
1	CLARK	10
2	KING	10
3	MILLER	10
4	JONES	20
5	FORD	20
6	ADAMS	20
7	SMITH	20
8	SCOTT	20
9	WARD	30
10	TURNER	30
11	ALLEN	30
12	JAMES	30
13	BLAKE	30
14	MARTIN	30

	DEPTNO	LOC
1	10	NEW YORK
2	20	DALLAS
3	30	CHICAGO
4	40	BOSTON

12.2 예제

예제 1.

```
SELECT /*+ leading(d e ) use_merge(e)  */  e.ename, d.loc, e.deptno
                FROM EMP e, DEPT d
            WHERE e.DEPTNO = d.DEPTNO;
```

	ENAME	LOC	DEPTNO
1	CLARK	NEW YORK	10
2	KING	NEW YORK	10
3	MILLER	NEW YORK	10
4	JONES	DALLAS	20
5	FORD	DALLAS	20
6	ADAMS	DALLAS	20
7	SMITH	DALLAS	20
8	SCOTT	DALLAS	20
9	WARD	CHICAGO	30
10	TURNER	CHICAGO	30
11	ALLEN	CHICAGO	30
12	JAMES	CHICAGO	30
13	BLAKE	CHICAGO	30
14	MARTIN	CHICAGO	30

-- order by 를 쓰지 않았는데 부서번호로 정렬 되어있다.

문제 71. emp와 salgrade를 조인해서 이름, 월급, 급여등급을 출력하는데 조인 방법이 sort merge join이 되게 하시오.

```
SELECT /*+  leading(s e) use_merge(e) */e.ename, e.sal, s.grade
        FROM EMP e , SALGRADE s
      WHERE e.sal BETWEEN s.LOSAL AND s.HISAL;
```

	ENAME	SAL	GRADE
1	SMITH	800	1
2	JAMES	950	1
3	ADAMS	1100	1
4	WARD	1250	2
5	MARTIN	1250	2
6	MILLER	1300	2
7	TURNER	1500	3
8	ALLEN	1600	3
9	CLARK	2450	4
10	BLAKE	2850	4
11	JONES	2975	4
12	SCOTT	3000	4
13	FORD	3000	4
14	KING	5000	5

-- sal (키 컬럼) 순으로 정렬 되어있다.

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
MERGE JOIN		1
SORT JOIN → 키 컬럼 정렬		5
TABLE ACCESS FULL	SCOTT.SALGRADE	5
FILTER		
SORT JOIN → 키 컬럼 정렬		14
TABLE ACCESS FULL	SCOTT.EMP	14

문제 72. 부서위치, 부서위치별 토탈월급을 출력하는데 적절한 조인순서와 조인방법을 명시해서 쿼리를 작성 하시오.

```
SELECT /*+ leading(d e) use_nl(e) */ d.loc, SUM(e.sal)
      FROM EMP e, DEPT d
     WHERE e.DEPTNO = d.DEPTNO
     GROUP BY d.loc;
```

	LOC	SUM(E.SAL)
1	NEW YORK	8750
2	CHICAGO	9400
3	DALLAS	10875

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		14
HASH GROUP BY		14
NESTED LOOPS		14
TABLE ACCESS FULL	SCOTT.DEPT	4
TABLE ACCESS FULL	SCOTT.EMP	4

문제 73. (점심시간 문제) 이름, 월급, 부서위치, 급여등급을 출력하는데 아래와 같이 실행 계획이 출력되게 하시오.

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
HASH JOIN		1
NESTED LOOPS		1
TABLE ACCESS FULL	SCOTT.SALGRADE	5
TABLE ACCESS FULL	SCOTT.EMP	1
TABLE ACCESS FULL	SCOTT.DEPT	4

```
SELECT /*+  leading(s e d) use_nl(e) use_hash(d) */e.ename, e.sal, d.loc, s.grade
          FROM EMP e , SALGRADE s, DEPT d
        WHERE e.sal BETWEEN s.LOSAL AND s.HISAL AND e.DEPTNO = d.DEPTNO;
```

13. outer join

2018년 5월 1일 화요일 오후 2:21

13.1 예제

예제 1. 이름과 부서위치를 출력하는데 outer join 사인을 사용해서 BOSTON도 출력되게 하시오.

```
SELECT e.ename, d.loc
    FROM EMP e, DEPT d
  WHERE e.DEPTNO (+) = d.DEPTNO;
```

	ENAME	LOC
2	ALLEN	CHICAGO
3	WARD	CHICAGO
4	JONES	DALLAS
5	MARTIN	CHICAGO
6	BLAKE	CHICAGO
7	CLARK	NEW YORI
8	SCOTT	DALLAS
9	KING	NEW YORI
10	TURNER	CHICAGO
11	ADAMS	DALLAS
12	JAMES	CHICAGO
13	FORD	DALLAS
14	MILLER	NEW YORI
15	(null)	BOSTON

-- nested loop join으로 출력해 보시오.

```
SELECT /*+  leading ( d e ) use_nl(e) */e.ename, d.loc
      FROM EMP e, DEPT d
        WHERE e.DEPTNO (+) = d.DEPTNO;
```

문제 74. 이름, 부서위치를 출력하는 outer join 문장을 아래와 같이 작성하는데 지금 입력한 JACK도 출력되게 하시오!

```
INSERT INTO emp(empno, ename, sal, deptno)
      VALUES ( 1092 , 'JACK' , 3400 , 70 );

SELECT e.ename, d.loc
      FROM EMP e, DEPT d
        WHERE e.DEPTNO  = d.DEPTNO (+);
```

	ENAME	LOC
2	KING	NEW YORK
3	CLARK	NEW YORK
4	FORD	DALLAS
5	ADAMS	DALLAS
6	SCOTT	DALLAS
7	JONES	DALLAS
8	SMITH	DALLAS
9	JAMES	CHICAGO
10	TURNER	CHICAGO
11	BLAKE	CHICAGO
12	MARTIN	CHICAGO
13	WARD	CHICAGO
14	ALLEN	CHICAGO
15	JACK	(null)

문제 75. 문제 74번의 조인방법을 nested loop 조인이 되게 하고 조인순서를 dept -> emp 순으로 조인되게 하시오.

```
SELECT /*+ leading (d e) use_nl(e) */ e.ename, d.loc
        FROM EMP e, DEPT d
          WHERE e.DEPTNO  = d.DEPTNO (+);
```

*leading 힌트를 써도 emp 테이블을 먼저 엑세스 한다. (outer join을 사용했기 때문에)

---> 조인할 때 outer join 사인을 사용하게 되면 조인 순서가 무조건 outer join 사인이 없는 쪽에서
outer join 사인이 있는 쪽으로 조인을 한다.

```
WHERE e.DEPTNO  = d.DEPTNO (+);
```

```
|Operation                                  | Object Name | Rows
-------------------------------------------------------------------
| SELECT STATEMENT Optimizer Mode=ALL_ROWS  |             |   14
|  HASH JOIN OUTER                          |             |   14
|   TABLE ACCESS FULL                       |SCOTT.EMP    |   14
|   TABLE ACCESS FULL                       |SCOTT.DEPT   |    4
```

문제 76. 문제 74번의 조인방법을 해쉬 조인이 되게 하고 조인 순서를 dept -> emp 순으로 조인되게 하시오.

```
SELECT /*+ leading ( d e ) use_hash(e) swap_join_inputs(d) */e.eNAME, d.loc
        FROM EMP e, DEPT d
          WHERE e.DEPTNO = d.DEPTNO (+);
```

```
Operation                                   | Object Name | Rows |
-----------------------------------------------------------------
 SELECT STATEMENT Optimizer Mode=ALL_ROWS   |             |   14 |
  HASH JOIN RIGHT OUTER                     |             |   14 |
   TABLE ACCESS FULL                        |SCOTT.DEPT   |    4 |
   TABLE ACCESS FULL                        |SCOTT.EMP    |   14 |
```

문제 77. 아래의 SQL을 튜닝 하시오.

```
SELECT e.ename, d.loc, e.job
      FROM EMP e, DEPT d
         WHERE e.deptno = d.DEPTNO (+)
            AND d.loc(+) = 'DALLAS';
```

e d

```
WHERE e.deptno = d.DEPTNO (+)
```

	ENAME	LOC	JOB
1	FORD	DALLAS	ANALYST
2	ADAMS	DALLAS	CLERK
3	SCOTT	DALLAS	ANALYST
4	JONES	DALLAS	MANAGEF
5	SMITH	DALLAS	CLERK
6	MILLER	(null)	CLERK
7	KING	(null)	PRESIDEN
8	CLARK	(null)	MANAGEF
9	JAMES	(null)	CLERK
10	TURNER	(null)	SALESMA
11	BLAKE	(null)	MANAGEF
12	MARTIN	(null)	SALESMA
13	WARD	(null)	SALESMA
14	ALLEN	(null)	SALESMA

```
SELECT /*+ swap_join_inputs(d) */ e.ename, d.loc, e.job
      FROM EMP e, DEPT d
        WHERE e.deptno = d.DEPTNO (+)
        AND d.loc(+) = 'DALLAS';
```

```
|Operation                                   | Object Name | Rows |
-----------------------------------------------------------------
| SELECT STATEMENT Optimizer Mode=ALL_ROWS   |             |   14 |
|  HASH JOIN RIGHT OUTER                     |             |   14 |
|   TABLE ACCESS FULL                        |SCOTT.DEPT   |    1 |
|   TABLE ACCESS FULL                        |SCOTT.EMP    |   14 |
```

문제 78. 아래의 full outer join을 튜닝 하시오.

```
SELECT e.ename, d.loc
         FROM EMP e FULL outer JOIN DEPT d
```

ON (e.DEPTNO = d.DEPTNO);

```
|Operation                                    | Object Name   | Rows |
-----------------------------------------------------------------------
| SELECT STATEMENT Optimizer Mode=ALL_ROWS    |               |   15 |
|  VIEW                                       |SYS.VW_FOJ_0   |   15 |
|   HASH JOIN FULL OUTER                      |               |   15 |
|    TABLE ACCESS FULL                        |SCOTT.DEPT     |    4 |
|    TABLE ACCESS FULL                        |SCOTT.EMP      |   14 |
```

SELECT **/*+ opt_param('_optimizer_native_full_outer_join','off') */** e.ename, d.loc
FROM EMP e FULL outer JOIN DEPT d
ON (e.DEPTNO = d.DEPTNO);

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		18
VIEW		18
UNION-ALL		
HASH JOIN OUTER		14
TABLE ACCESS FULL	SCOTT.EMP	14
TABLE ACCESS FULL	SCOTT.DEPT	4
HASH JOIN ANTI		4
TABLE ACCESS FULL	SCOTT.DEPT	4
TABLE ACCESS FULL	SCOTT.EMP	14

*** opt_param('_optimizer_native_full_outer_join','off') : 성능이 더 좋은 full outer join 실행계획을 사용하지 않겠다.**

*** opt_param('_optimizer_native_full_outer_join','force') : 성능이 더 좋은 full outer join 실행계획을 사용한다.**

문제 79. 아래의 SQL을 튜닝 하시오. (where 절에서 와일드 카드를 앞에 사용하는 경우)

SELECT ename, sal, job
FROM EMP
WHERE ename LIKE '%EN%' OR ename LIKE '%IN%'; -- 와일드 카드가 앞에 있다.

1	recursive calls	0
2	db block gets	0
3	consistent gets	57

SELECT /*+**leading(e3 e1) use_nl(e1)**/e1.ename, e1.sal, e1.job
FROM EMP e1, (SELECT /*+ **index_ffs** (e2 emp_ename) **no_merge** */ ROWID rr
FROM EMP e2
WHERE ename LIKE '%EN%' OR ename LIKE '%IN%') e3
WHERE E1.ROWID = e3.rr;

1	recursive calls	0
2	db block gets	0
3	consistent gets	8

***no_merge 힌트 :** in line view를 해체하지 말아라. 해체되면

```
SELECT ename, sal, job
        FROM EMP
      WHERE ename LIKE '%EN%' OR ename LIKE '%IN%';
```

이렇게 되버림.

문제 80. 우리반 테이블에서 주소가 송파구와 동작구에 사는 학생들의 이름, 주소, 나이를 출력하시오.

-튜닝 전-

```
SELECT ename, address, age
    FROM EMP2
  WHERE address LIKE '%송파구%' OR address LIKE '%동작구%';
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	46

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		2
TABLE ACCESS FULL	SCOTT.EMP2	2

-튜닝 후-

```
SELECT /*+ leading (e3 e1) use_nl(e1) */ e1.ename, e1.address, e1.age
    FROM EMP2 e1, (SELECT /*+ index_ffs(e2 emp_ename) no_merge */ROWID ri
      FROM EMP2 e2
      WHERE address LIKE '%송파구%' OR address LIKE '%동작구%') e3
  WHERE e1.ROWID = e3.ri;
```

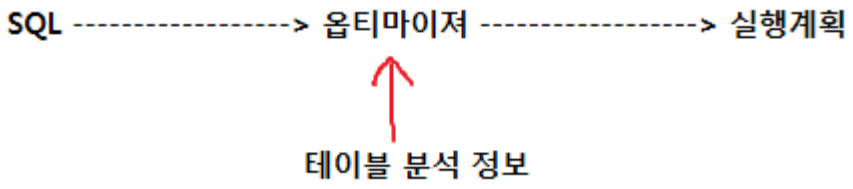
1	recursive calls	0
2	db block gets	0
3	consistent gets	2

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		2
NESTED LOOPS		2
VIEW		2
INDEX FULL SCAN	SCOTT.EMP2_ADDRESS	2
TABLE ACCESS BY USER ROWID	SCOTT.EMP2	1

14. 옵티마이져

2018년 5월 1일 화요일 오후 2:22

" SQL을 가장 효율적이고 빠르게 수행할 수 있는 최적의 처리경로를 선택해 주는 DBMS의 핵심 엔진 "



15. 쿼리변환

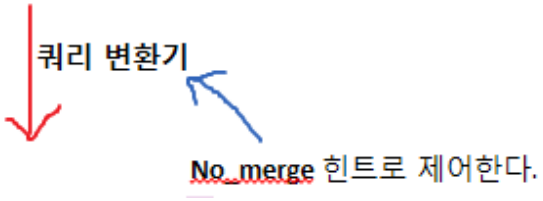
2018년 5월 1일 화요일 오후 2:23

" 옵티마이저가 실행계획을 생성하기 전에 사용자가 작성한 SQL보다 결과는 똑같은데 비용이 더 적게 발생하는 SQL이 있다면 그 SQL로 알아서 쿼리를 변경한다. "

15.1 예제

예제 1. 이름이 EN 또는 IN을 포함하고 있는 사원들의 이름, 월급, 직업을 출력 하시오.

```
SELECT /*+leading( e3 e1 ) use_nl(e1)*/e1.ename, e1.sal, e1.job
        FROM EMP e1, (SELECT /*+ index_ffs(e2 emp_ename) */ ROWID rr        -- no_merge를 지웠음
                                FROM EMP e2
                                        WHERE ename LIKE '%EN%' OR ename LIKE '%IN%' ) e3
         WHERE E1.ROWID = e3.rr;
```




```
SELECT ename, sal, job
        FROM EMP
      WHERE ename LIKE '%EN%' OR ename LIKE '%IN%';
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
TABLE ACCESS FULL	SCOTT.EMP	1

문제 82. DALLAS에 있는 부서번호에서 근무하는 사원들의 이름,월급을 출력하는데 조인을 사용하지 말고 서브쿼리문으로 작성 하시오.

```
SELECT ename, sal
        FROM EMP
        WHERE deptno = (SELECT deptno
                                FROM DEPT
                                        WHERE loc = 'DALLAS' );
```

	ENAME	SAL
1	SMITH	800
2	JONES	2975
3	SCOTT	3000
4	ADAMS	1100
5	FORD	3000

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
TABLE ACCESS FULL	SCOTT.EMP
TABLE ACCESS FULL	SCOTT.DEPT

```
SELECT ename, sal
        FROM EMP
        WHERE deptno in (SELECT deptno
                         FROM DEPT
                         WHERE loc = 'DALLAS' );
```

서브 쿼리를 사용했는데 조인을 하고 있다.

Join ?

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		5
HASH JOIN SEMI		5
TABLE ACCESS FULL	SCOTT.EMP	14
TABLE ACCESS FULL	SCOTT.DEPT	1

*SEMI 조인 : 절반의 조인, **메인 쿼리부터 수행되는 것으로 고정된다.**

--->swap_join_inputs 힌트를 통해 변경가능

문제 84. 해쉬 세미조인하지 말고 서브쿼리로 수행하라고 힌트를 주시오.

***일반 조인은 쿼리순서를 조절할 수 있지만 세미조인은 메인 쿼리부터 수행되는 것으로 고정된다.**

강하게 감싸라!

```
SELECT ename, sal
    FROM EMP
    WHERE deptno in (SELECT /*+ no_unnest */ deptno
                     FROM DEPT
                     WHERE loc = 'DALLAS' );
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		5
FILTER		
TABLE ACCESS FULL	SCOTT.EMP	14
TABLE ACCESS FULL	SCOTT.DEPT	1

***no_unnest : 하지마라 + 감싸지마라 = 부정 + 부정 => 강한긍정!! = 강하게 감싸라!!**

문제 85. 아래의 SQL의 실행계획을 dept가 먼저 드라이빙 되면서 해쉬 세미 조인되게 하시오.

```
SELECT ename, sal
       FROM EMP
         WHERE deptno IN (SELECT /*+ swap_join_inputs(dept) */ DEPTNO
                          FROM DEPT
                                WHERE loc = 'DALLAS');
```

```
SELECT STATEMENT Optimizer Mode=ALL_ROWS                    5
  HASH JOIN RIGHT SEMI                                      5
    TABLE ACCESS FULL                 SCOTT.DEPT            1
    TABLE ACCESS FULL                 SCOTT.EMP            14
```

16. subquery unnesting

2018년 5월 1일 화요일 오후 2:23

" 서브쿼리를 실행할 때 서브쿼리문으로 실행할지 아니면 조인 형태로 변경해서 실행할지를 결정하게끔 쿼리변환기를 제어하는 기술. 서브쿼리를 감싸지않겠다. (unnesting) "

-힌트-

1.unnest (감싸지 마라) : 서브쿼리로 수행하지 말고 서브쿼리를 풀어 해쳐서 수행해라

2.no_unnest (강하게 감싸라) : 반드시 **서브쿼리로 실행**해라.

***No_unnest와 짝궁 인 힌트**

- **Push_subq :** 서브 쿼리부터 수행해라 ~
- **No_push_subq :** 메인 쿼리부터 수행해라 ~

16.1 예제

예제 1. 아래의 SQL의 실행계획이 서브쿼리 실행되는지 세미조인 형태로 실행되는지 확인 하시오.

```
SELECT ename, sal
       FROM EMP
   WHERE deptno IN (SELECT DEPTNO
                    FROM DEPT
              WHERE loc = 'DALLAS');
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		5
HASH JOIN SEMI		5
TABLE ACCESS FULL	SCOTT.EMP	14
TABLE ACCESS FULL	SCOTT.DEPT	1

문제 86. 아래의 SQL이 세미조인으로 수행되지 않고 서브 쿼리문으로 수행되게끔 힌트를 주시오.

```
SELECT ename, sal
       FROM EMP
         WHERE deptno IN (SELECT /*+ no_unnest */  DEPTNO
                          FROM DEPT
                                WHERE loc = 'DALLAS');
```

Operation	Object Name
4 SELECT STATEMENT Optimizer Mode=ALL_ROWS	
3 FILTER	
1 TABLE ACCESS FULL	SCOTT.EMP
2 TABLE ACCESS FULL	SCOTT.DEPT

문제 87. 문제 86번에서 실행계획이 서브쿼리부터 수행될 수 있도록 힌트를 주시오.

```
SELECT ename, sal
    FROM EMP
      WHERE deptno IN (SELECT /*+ no_unnest push_subq */  DEPTNO
                        FROM DEPT
                                WHERE loc = 'DALLAS');
```

Operation	Object Name	Rows
3 SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
2 TABLE ACCESS FULL	SCOTT.EMP	1
1 TABLE ACCESS FULL	SCOTT.DEPT	1

- **/*+ No_unnest + [Push_subq] */** : 서브쿼리로 수행하고 서브쿼리 부터 수행해라

문제 88. 아래의 SQL을 QB_NAME 힌트를 이용해서 다시 작성 하시오.

*QB_NAME : Query block에 이름을 지어주는 힌트

```
Select /*+ QB_NAME(main) */ ename, sal
    From emp
    Where deptno in ( select /*+ QB_NAME(sub) */ deptno
                      From dept
                      Where 'DAL
```

문제 89. 관리자인 사원들의 이름을 출력 하시오. (자기 밑에 직속부하가 한명이라도 있는 사원들)

```
SELECT ename
    FROM EMP
  WHERE empno IN (SELECT mgr FROM EMP );
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		6
HASH JOIN SEMI		6
TABLE ACCESS FULL	SCOTT.EMP	14
TABLE ACCESS FULL	SCOTT.EMP	13

문제 90. 아래의 SQL의 실행계획이 해쉬 세미조인 되지 않고 서브쿼리로 수행되게 하시오.

```
SELECT ename
    FROM EMP
  WHERE empno IN (SELECT/*+ NO_UNNEST */ mgr FROM EMP );
```

```
SELECT STATEMENT Optimizer Mode=ALL_ROWS                    1
  FILTER
    TABLE ACCESS FULL                 SCOTT.EMP             14
    TABLE ACCESS FULL                 SCOTT.EMP              2
```

문제 91. 아래의 SQL의 실행계획이 해쉬세미조인 되지 않고 서브쿼리로 수행되게 하는데 서브쿼리 부터 수행되게 하시오.

```
SELECT ename
      FROM EMP
   WHERE empno IN (SELECT/*+ NO_UNNEST PUSH_SUBQ */ mgr FROM EMP );
```

```
Operation                                  Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS
  TABLE ACCESS FULL                        SCOTT.EMP
    TABLE ACCESS FULL                      SCOTT.EMP
```

*어떤 EMP가 메인쿼리인지 서브쿼리인지 구별이 안됨 ----> QB_NAME() 힌트를 사용한다.

문제 92. 문제91번의 실행계획을 보면 둘다 EMP 테이블이라 메인쿼리부터 수행했는지 서브쿼리부터 수행했는지 확인하기 어려우니 확인 될 수 있도록 힌트를 주시오.

```
SELECT /*+ QB_NAME(main) */ ename
      FROM EMP
   WHERE empno IN (SELECT/*+ NO_UNNEST PUSH_SUBQ  qb_name(sub) */ mgr FROM EMP );
```

*format을 아래와 같이 바꿔줘야함.

```
Format : ADVANCED ALLSTATS LAST

-----------------------------------------------------------------------------
| Id  | Operation           | Name | E-Rows |E-Bytes| Cost (%CPU)| E-Time   |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |        |       |     6 (100)|          |
|*  1 |  TABLE ACCESS FULL  | EMP  |      1 |     8 |     3   (0)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| EMP  |      2 |     6 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------

   1 - MAIN / EMP@MAIN
   2 - SUB  / EMP@SUB
```

문제 93. 관리자가 아닌 사원들의 이름을 출력 하시오.

```
SELECT ename
        FROM EMP
    WHERE empno NOT IN (SELECT mgr
                                FROM EMP
                 WHERE mgr IS NOT null);
```

	ENAME
1	TURNER
2	WARD
3	MARTIN
4	ALLEN
5	MILLER
6	SMITH
7	ADAMS
8	JAMES

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		8
HASH JOIN ANTI SNA		8
TABLE ACCESS FULL	SCOTT.EMP	14
TABLE ACCESS FULL	SCOTT.EMP	13

문제 94. 문제 93번의 실행계획이 hash right anti 조인이 되게 하시오.

```
SELECT ename
        FROM EMP
 WHERE empno NOT IN (SELECT /*+swap_join_inputs(emp)*/ mgr
                                FROM EMP
                                 WHERE mgr IS NOT null);
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		14
HASH JOIN RIGHT ANTI		14
TABLE ACCESS FULL	SCOTT.EMP	13
TABLE ACCESS FULL	SCOTT.EMP	14

17. view merging

2018년 5월 1일 화요일 오후 2:24

" view나 in line view를 해체할지 말지 결정하게끔 쿼리변환기를 제어하는 방법 "

■ **힌트종류**

1.merge() : 뷰나 인라인뷰를 해체해라 ~
2.no_merge() : 뷰나 인라인뷰를 해체하지 말아라 ~

17.1 예제

문제 95. 직업이 SALESMAN인 사원들의 사원번호, 이름, 직업, 관리자번호, 입사일, 월급, 커미션, 부서번호를 출력하는 VIEW를 생성 하시오. (emp_salesman 으로 만드시오)

```
CREATE VIEW emp_salesman
AS
SELECT *
        FROM EMP
    WHERE job = 'SALESMAN';
```

문제 96. emp_salesman 뷰와 dept와 조인해서 이름, 직업, 관리자번호, 월급, 부서위치를 출력 하시오.

```
SELECT e.ename, e.mgr, e.sal, d.loc
        FROM EMP_salesman e, DEPT d
          WHERE e.deptno = d.DEPTNO;
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		3
HASH JOIN		3
TABLE ACCESS FULL	SCOTT.EMP	3
TABLE ACCESS FULL	SCOTT.DEPT	4

View?

* 옵티마이져가 view를 해체하면 비용이 적게 들것이라 예상해서 view를 해체했다.

문제 97. 문제 96번을 다시 수행하는데 view를 해체하지 않도록 힌트를 주고 실행 하시오.

```
SELECT /*+ no_merge(e) */ e.ename, e.mgr, e.sal, d.loc
        FROM EMP_salesman e, DEPT d
          WHERE e.deptno = d.DEPTNO;
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		3
HASH JOIN		3
VIEW	SCOTT.EMP_SALESMAN	3
TABLE ACCESS FULL	SCOTT.EMP	3
TABLE ACCESS FULL	SCOTT.DEPT	4

*no_merge를 쓰면 view를 해체하지 않으므로 조인 순서를 제어할 수 있다.

문제 98. 문제 97번의 view와 dept 테이블과의 조인순서와 조인방법이 아래와 같이 수행되게 하시오.

조인순서 : emp_salesman --> dept
조인방법 : nested loop join

```
SELECT /*+ no_merge(e) leading (e d) use_nl(d) */ e.ename, e.mgr, e.sal, d.loc
        FROM EMP_salesman e, DEPT d
           WHERE e.deptno = d.DEPTNO;
```

Operation	Object Name	Rows
5 SELECT STATEMENT Optimizer Mode=ALL_ROWS		3
4 NESTED LOOPS		3
2 VIEW	SCOTT.EMP_SALESMAN	3
1 TABLE ACCESS FULL	SCOTT.EMP	3
3 TABLE ACCESS FULL	SCOTT.DEPT	1

*뷰를 해체하지 않는 경우 ..?

대용량의 경우 모든 테이블의 내용(1억건)을 조인 시켜야 되지만 뷰는 salesman 인경우의 건수(5건)만 조인을 하기 때문에 뷰를 해체하지 않는 것이 더 효율적이다.

문제 99. 사원번호, 이름, 부서위치, 부서번호를 출력하는 뷰를 생성 하시오. (뷰 이름 : emp9000)

```
CREATE view emp9000
AS
SELECT e.ename, d.loc, d.deptno
        FROM EMP e, DEPT d
    WHERE e.DEPTNO = d.DEPTNO;
```

문제 100. 문제 99번에서 만든 emp9000과 보너스테이블을 조인해서 이름, 부서위치, 보너스를 출력 하시오.

```
SELECT e.ename, e.loc, b.BONUS
        FROM emp9000 e, BONUS b
    WHERE e.empno = b.EMPNO;
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		14
HASH JOIN		14
HASH JOIN		14
TABLE ACCESS FULL	SCOTT.DEPT	4
TABLE ACCESS FULL	SCOTT.EMP	14
TABLE ACCESS FULL	SCOTT.BONUS	14

*뷰를 해체했다.

문제 101. 문제 100번의 view를 해체하지 말고 bonus 테이블과 조인되게 하시오.

```
SELECT /*+ no_merge(e) */ e.ename, e.loc, b.BONUS
        FROM emp9000 e, BONUS b
    WHERE e.empno = b.EMPNO;
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		14
HASH JOIN		14
VIEW	SCOTT.EMP9000	14
HASH JOIN		14
TABLE ACCESS FULL	SCOTT.DEPT	4
TABLE ACCESS FULL	SCOTT.EMP	14
TABLE ACCESS FULL	SCOTT.BONUS	14

문제 102. 아래와 같이 실행계획이 나오게 하시오.

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		14
HASH JOIN		14
VIEW	SCOTT.EMP9000	14
NESTED LOOPS		14
TABLE ACCESS FULL	SCOTT.EMP	14
TABLE ACCESS FULL	SCOTT.DEPT	1
TABLE ACCESS FULL	SCOTT.BONUS	14

```
CREATE view emp9000
AS
SELECT e.empno, e.ename, d.loc, d.deptno
        FROM EMP e, DEPT d
    WHERE e.DEPTNO = d.DEPTNO;

SELECT /*+  leading(ee.e ee.d) use_nl(ee.d) no_merge(ee)  */ ee.ename, ee.loc, b.BONUS
        FROM emp9000 ee, BONUS b
        WHERE ee.empno = b.EMPNO;
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
HASH JOIN	
VIEW	SCOTT.EMP9000
NESTED LOOPS	
TABLE ACCESS FULL	SCOTT.EMP
TABLE ACCESS FULL	SCOTT.DEPT
TABLE ACCESS FULL	SCOTT.BONUS

뷰 만에

■ 서브쿼리 사용시 힌트 총정리

1.서브쿼리 형태로 수행해라 : No_unnest 힌트

-서브 쿼리부터 수행해라 : push_subq 힌트

-메인 쿼리부터 수행해라 : no_push_subq 힌트

2.세미/안티 조인으로 변경해서 수행해라 ~ Unnest 힌트

-세미 조인 (in 또는 exist 연산자를 사용할 때)

1.nested loop semi 조인 : nl_sj

2.hash semi 조인 : hash_sj -- 주로 사용

3.sort merge semi 조인 : merge_sj

-안티 조인 (not in 연산자를 사용할 때)

1.nested loop anti 조인 : nl_aj

2.hash anti 조인 : hash_aj

3.sort merge anti 조인 : merge_aj

문제 103. 관리자인 사원들의 이름을 출력하는데 실행계획이 nested loop semi join이 나오게 하시오.

```
SELECT ename
    FROM EMP
      WHERE empno in (SELECT /*+ unnest nl_sj */ mgr FROM emp);
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	162

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
NESTED LOOPS SEMI	
TABLE ACCESS FULL	SCOTT.EMP
TABLE ACCESS FULL	SCOTT.EMP

문제 104. 관리자인 사원들의 이름을 출력하는데 실행계획이 hash semi가 나오게 하시오.

```
SELECT ename
  FROM EMP
  WHERE empno in (SELECT /*+ unnest hash_sj */ mgr FROM emp);
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	64

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
HASH JOIN SEMI	
TABLE ACCESS FULL	SCOTT.EMP
TABLE ACCESS FULL	SCOTT.EMP

문제 105. 관리자가 아닌 사원들의 이름을 출력하는데 nested loop anti 조인이 되게 하시오.

```
SELECT ename
FROM EMP
  WHERE empno NOT in (SELECT /*+ unnest nl_aj */ mgr
                      FROM EMP WHERE mgr IS NOT null);
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
HASH JOIN ANTI SNA	
TABLE ACCESS FULL	SCOTT.EMP
TABLE ACCESS FULL	SCOTT.EMP

1	recursive calls	0
2	db block gets	0
3	consistent gets	64

*nested loop anti join 을 하기 위해선 where 절에서 비교하는 양 옆에 not null 제약이 있어야한다.
서브쿼리에서는 is not null 를 사용해서 null을 처리하므로 없어도 되지만 empno에는 not null 제약이 없으므로 nested loop anti 조인을 하지 못한다.

```
SELECT ename
  FROM EMP
    WHERE empno IS NOT NULL          -- not null 제약을 대신할 수 있다.
      AND  empno NOT in (SELECT /*+ unnest nl_aj */ mgr FROM EMP WHERE mgr IS NOT null);
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		8
NESTED LOOPS ANTI		8
TABLE ACCESS FULL	SCOTT.EMP	14
TABLE ACCESS FULL	SCOTT.EMP	6

1	recursive calls	0
2	db block gets	0
3	consistent gets	162

문제 106. 관리자가 아닌 사원들의 이름을 출력하는데 hash anti 조인으로 수행되게 하시오.

```
SELECT ename
        FROM EMP
         WHERE empno IS NOT NULL          -- not null 제약을 대신할 수 있다.
           AND  empno NOT in (SELECT /*+ unnest hash_aj */ mgr FROM EMP WHERE mgr IS NOT null);
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	64

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
HASH JOIN ANTI	
TABLE ACCESS FULL	SCOTT.EMP
TABLE ACCESS FULL	SCOTT.EMP

문제 107. exists 문을 이용해서 부서 테이블에서 부서위치를 출력하는데 사원 테이블에 존재하는 부서번호에 대한 부서위치만 출력 하시오.

```
SELECT loc
       FROM DEPT d
       WHERE EXISTs (SELECT 'x'
                     FROM EMP e
                     WHERE d.DEPTNO = e.deptno);
```

Operation	Object Name
SELECT STATEMENT Optimizer Mode=ALL_ROWS	
HASH JOIN SEMI	
TABLE ACCESS FULL	SCOTT.DEPT
TABLE ACCESS FULL	SCOTT.EMP

문제 108. not exists 문을 이용해서 부서 테이블에서 부서위치를 출력하는데 사원 테이블에 존재하지 않는 부서번호에 대한 부서위치만 출력 하시오.

```
SELECT loc
  FROM DEPT d
 WHERE NOT EXISTS (SELECT 'x' FROM EMP e WHERE d.DEPTNO = e.DEPTNO);
```

SELECT STATEMENT Optimizer Mode=ALL_ROWS	-	4
HASH JOIN ANTI		4
TABLE ACCESS FULL	SCOTT.DEPT	4
TABLE ACCESS FULL	SCOTT.EMP	14

* not exists 는 is not null 을 사용하지 않아도 hash anti join 이 잘 나옴.

18. push_pred, no_push_pred

2018년 5월 1일 화요일 오후 2:24

" 뷰 또는 인라인 뷰 바깥의 조인 조건을 안으로 집어넣거나 집어넣지 않는 힌트 "

■ 쿼리 변환기를 제어하는 view 관련 힌트

1. push_pred() : 뷰나 인라인 뷰 바깥에 있는 조인 조건이 뷰나 인라인 뷰 안으로 집어넣는 힌트

2.no_push_pred() : 뷰나 인라인 뷰 바깥에 있는 조인 조건을 뷰나 인라인 뷰 안으로 집어넣지 않게 하는 힌트

*push_pred : 뷰 밖의 조건을 뷰 안으로 집어넣는 작업

18.1 예제

문제 109. 부서번호, 부서번호별 평균월급을 출력하는 view를 dept_avgsal이라는 이름으로 생성 하시오.

```
CREATE VIEW dept_avgsal
as
SELECT deptno, avg(sal) 평균
        FROM EMP
    GROUP BY deptno;

SELECT *
        FROM dept_avgsal;
```

	DEPTNO	평균
1	30	1566.6666666666666666666666666666666667
2	70	3400
3	20	2175
4	10	2916.6666666666666666666666666666666667

문제 110. 문제 109번에서 만든 dept_avgsal 뷰와 dept 테이블과 조인해서 부서위치가 CHICAGO의 부서번호, 부서위치, 그 부서의 평균 월급이 출력되게 하시오.

```
SELECT d.deptno, d.loc, a.평균
        FROM DEPT d, dept_avgsal a
    WHERE d.DEPTNO = a.DEPTNO AND d.loc = 'CHICAGO';
```

	DEPTNO	LOC	평균
1	30	CHICAGO	1566.6666666666666666666666666666666667

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
HASH GROUP BY		1
HASH JOIN		5
TABLE ACCESS FULL	SCOTT.DEPT	1
TABLE ACCESS FULL	SCOTT.EMP	14

문제 111. 문제 110번을 다시 수행하는데 뷰를 해체하지 말고 수행하게 하시오.

```
SELECT /*+ no_merge(a) */ d.deptno, d.loc, a.평균
        FROM DEPT d, dept_avgsal a
    WHERE d.DEPTNO = a.DEPTNO AND d.loc = 'CHICAGO';
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
HASH JOIN		1
JOIN FILTER CREATE	SYS.:BF0000	1
TABLE ACCESS FULL	SCOTT.DEPT	1
VIEW	SCOTT.DEPT_AVGSAL	3
HASH GROUP BY		3
JOIN FILTER USE	SYS.:BF0000	14
TABLE ACCESS FULL	SCOTT.EMP	14

문제 112. 아래의 인덱스 2개를 각각 수행하고 문제 111번의 실행계획을 확인 하시오.

```
CREATE INDEX emp_deptno ON EMP(deptno);

CREATE INDEX dept_loc ON DEPT(loc);

SELECT /*+ no_merge(a) */ d.deptno, d.loc, a.평균
        FROM DEPT d, dept_avgsal a
        WHERE d.DEPTNO = a.DEPTNO AND d.loc = 'CHICAGO';
```

*아래의 push predicate 라는 실행계획은 뷰 밖의 조건을 뷰 안으로 집어넣어서 group by 되는 건수를 줄이는 부분 -- **WHERE d.DEPTNO = a.DEPTNO AND d.loc = 'CHICAGO' 를 뷰 안으로 집어넘..**

View에 조건을 넣는 작업 : push predicate

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
NESTED LOOPS		1
TABLE ACCESS BY INDEX ROWID	SCOTT.DEPT	1
INDEX RANGE SCAN	SCOTT.DEPT_LOC	1
VIEW PUSHED PREDICATE	SCOTT.DEPT_AVGSAL	1
FILTER		
SORT AGGREGATE		1
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP	5
INDEX RANGE SCAN	SCOTT.EMP_DEPTNO	5

*부서번호의 평균 월급만 있으면 되기때문에 group by 안함..

문제 113. 아래의 뷰 밖의 조건이 뷰 안으로 들어가지 못하도록 힌트를 주시오.

= push pred 사용하지 않도록

```
SELECT /*+ no_merge(a) no_push_pred(a) */ d.deptno, d.loc, a.평균
        FROM DEPT d, dept_avgsal a
          WHERE d.DEPTNO = a.DEPTNO AND d.loc = 'CHICAGO';
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
HASH JOIN		1
JOIN FILTER CREATE	SYS.:BF0000	1
TABLE ACCESS BY INDEX ROWID	SCOTT.DEPT	1
INDEX RANGE SCAN	SCOTT.DEPT_LOC	1
VIEW	SCOTT.DEPT_AVGSAL	3
HASH GROUP BY		3
JOIN FILTER USE	SYS.:BF0000	14
TABLE ACCESS FULL	SCOTT.EMP	14

***뷰에는 loc이란 컬럼이 없는데 어떻게 뷰 안의 조건으로 들어갈까??**

--> loc 컬럼으로 비교하지 않고 CHICAGO의 deptno 조건(deptno = 30)으로 CHICAGO를 찾는다.

문제 114. 아래의 SQL의 실행계획을 인라인 뷰 바깥에 있는 조인 조건을 인라인뷰 안쪽으로 집어끔 하시오.

```
CREATE INDEX emp_deptno_job ON EMP(deptno, job);
```

```
SELECT d.dname, e.*
    FROM DEPT d,
        (SELECT deptno, empno, ename, job, sal, sal * 1.1 AS sal2, hiredate
            FROM EMP WHERE job = 'CLERK'
         UNION all
         SELECT deptno, empno, ename, job, sal, sal*1.1 AS sal2, hiredate
            FROM EMP
         WHERE job='SALESMAN') e
    WHERE e.deptno = d.DEPTNO AND d.loc = 'CHICAGO';
```

loc 없음

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1
NESTED LOOPS		1
TABLE ACCESS FULL	SCOTT.DEPT	1
VIEW		1
UNION ALL PUSHED PREDICATE		
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP	1
INDEX RANGE SCAN	SCOTT.EMP_DEPTNO_JOB	2
TABLE ACCESS BY INDEX ROWID	SCOTT.EMP	1
INDEX RANGE SCAN	SCOTT.EMP_DEPTNO_JOB	2

1	recursive calls	0
2	db block gets	0
3	consistent gets	11

Push pred 작업이 수행 되서 아래의 코드를 수행

```
SELECT d.dname, e.*
        FROM DEPT d,
          (SELECT deptno, empno, ename, job, sal, sal * 1.1 AS sal2, hiredate
                FROM EMP
           WHERE job = 'CLERK' and deptno = 30
          UNION all
          SELECT deptno, empno, ename, job, sal, sal*1.1 AS sal2, hiredate
               FROM EMP
          WHERE job='SALESMAN'  and deptno = 30
        ) e
    WHERE e.deptno = d.DEPTNO
        AND d.loc = 'CHICAGO';
```

++ 아래의 SQL의 실행계획을 인라인 뷰 바깥에 있는 조인 조건을 인라인뷰 안쪽으로 집어 넣지 않게끔 하시오.

```
SELECT /*+ no_push_pred(e) */d.dname, e.*
    FROM DEPT d,
        (SELECT deptno, empno, ename, job, sal, sal * 1.1 AS sal2, hiredate
            FROM EMP WHERE job = 'CLERK'
     UNION all
     SELECT deptno, empno, ename, job, sal, sal*1.1 AS sal2, hiredate
            FROM EMP
```

```
        WHERE job='SALESMAN') e
    WHERE e.deptno = d.DEPTNO AND d.loc = 'CHICAGO';
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		2
HASH JOIN		2
TABLE ACCESS FULL	SCOTT.DEPT	1
VIEW		8
UNION-ALL		
TABLE ACCESS FULL	SCOTT.EMP	4
TABLE ACCESS FULL	SCOTT.EMP	4

1	recursive calls	0
2	db block gets	0
3	consistent gets	21

-----> **보통 push pred가 사용되길 권장된다.**

19. 스칼라 서브쿼리문을 이용한 튜닝

2018년 5월 1일 화요일 오후 2:26

Select문에서 서브쿼리를 쓸 수 있는 절 :

1.select -- **스칼라 서브쿼리**
2.from -- 인라인 뷰
3.where -- 서브쿼리
4.group by -- 사용못함 x
5.having -- 서브쿼리
6.order by -- **스칼라 서브쿼리 (거의 안씀)**

19.1 예제

문제 115. 이름, 월급, 사원 테이블의 최대월급, 사원 테이블의 최소월급, 사원 테이블의 평균 월급을 출력 하시오.

```
SELECT ename, sal , MAX(Sal) OVER () 최대, MIN(sal) OVER () 최소, AVG(sal) OVER () 평균
        FROM EMP;
```

	ENAME	SAL	최대	최소	평균
1	KING	5000	5000	800	2073.2142857142857142857142857142857142 86
2	BLAKE	2850	5000	800	2073.2142857142857142857142857142857142 86
3	CLARK	2450	5000	800	2073.2142857142857142857142857142857142 86
4	JONES	2975	5000	800	2073.2142857142857142857142857142857142 86
5	MARTIN	1250	5000	800	2073.2142857142857142857142857142857142 86
6	ALLEN	1600	5000	800	2073.2142857142857142857142857142857142 86
7	TURNER	1500	5000	800	2073.2142857142857142857142857142857142 86
8	JAMES	950	5000	800	2073.2142857142857142857142857142857142 86
9	WARD	1250	5000	800	2073.2142857142857142857142857142857142 86
10	FORD	3000	5000	800	2073.2142857142857142857142857142857142 86
11	SMITH	800	5000	800	2073.2142857142857142857142857142857142 86
12	SCOTT	3000	5000	800	2073.2142857142857142857142857142857142 86
13	ADAMS	1100	5000	800	2073.2142857142857142857142857142857142 86
14	MILLER	1300	5000	800	2073.2142857142857142857142857142857142 86

문제 116. 이름, 월급, 사원 테이블의 최대 월급, 사원 테이블의 최소 월급, 사원 테이블의 평균 월급을 출력 하시오.
(분석 함수를 이용하지 말고 수행 하시오.)

```
SELECT ename, sal, substr(salary,1,10) 최대,
```

```
                SUBSTR(salary,11,10) 최소,
                SUBSTR(salary,21,10) 평균
        FROM (
                SELECT ename, sal, (SELECT RPAD(MAX(sal),10,' ') ||
                                    RPAD(MIN(sal),10,' ') ||
                                    ROUND(RPAD(AVG(sal),10,' ') )
                                    FROM EMP
                                    ) AS salary
                FROM EMP
        );
```

	ENAME	SAL	최대	최소	평균
1	KING	5000	5000	800	2073
2	BLAKE	2850	5000	800	2073
3	CLARK	2450	5000	800	2073
4	JONES	2975	5000	800	2073
5	MARTIN	1250	5000	800	2073
6	ALLEN	1600	5000	800	2073
7	TURNER	1500	5000	800	2073
8	JAMES	950	5000	800	2073
9	WARD	1250	5000	800	2073
10	FORD	3000	5000	800	2073
11	SMITH	800	5000	800	2073
12	SCOTT	3000	5000	800	2073
13	ADAMS	1100	5000	800	2073
14	MILLER	1300	5000	800	2073

1	recursive calls	0
2	db block gets	0
3	consistent gets	14

문제 117. 문제 116번을 튜닝 전 SQL로 작성 하시오.

```
SELECT ename, sal, (SELECT MAX(sal) FROM emp) 최대, (SELECT MIN(sal) FROM emp)
최소 ,
        (SELECT AVG(sal) FROM emp) 평균
    FROM EMP;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	67

문제 118. 아래의 SQL을 튜닝 하시오.

```
SELECT E.ename, e.sal, ee.*              --9
    FROM (SELECT deptno ,MAX(sal), MIN(sal), AVG(sal)
```

```
                    FROM EMP
            GROUP BY deptno) ee, EMP e
   WHERE E.DEPTNO = EE.deptno;


SELECT /*+ push_pred(ee) */E.ename, e.sal, ee.*   -- 11
    FROM (SELECT deptno ,MAX(sal), MIN(sal), AVG(sal)
                    FROM EMP
            GROUP BY deptno) ee, EMP e
WHERE E.DEPTNO = EE.deptno;
```

20. 수정가능 조인 뷰

2018년 5월 1일 화요일 오후 2:26

" 테이블의 데이터를 갱신할 때 뷰를 이용해서 갱신하는 방법으로 SQL 튜닝을 위해서 사용하는 기술 "

20.1 예제

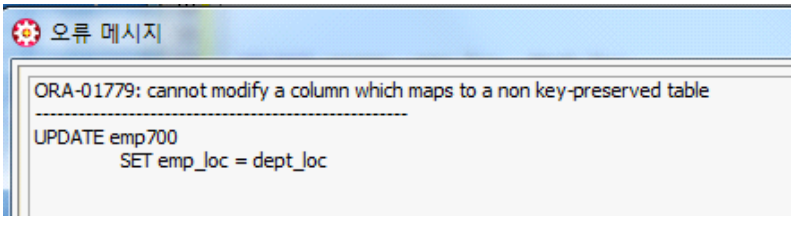
예제 1. 사원 테이블에 loc 컬럼을 추가하고 아래의 view를 만드시오.

```
ALTER TABLE EMP
        ADD loc VARCHAR2(20);

CREATE VIEW emp700
AS
SELECT e.ename, e.loc AS emp_loc, d.loc AS dept_loc
        FROM EMP e, DEPT d
    WHERE e.deptno = d.deptno;  -- 조인을 통해 만든 복합 뷰

SELECT ename, emp_loc, dept_loc
        FROM emp700;
```

	ENAME	EMP_LOC	DEPT_LOC
1	KING	(null)	NEW YORK
2	CLARK	(null)	NEW YORK
3	MILLER	(null)	NEW YORK
4	JONES	(null)	DALLAS
5	FORD	(null)	DALLAS
6	SMITH	(null)	DALLAS
7	SCOTT	(null)	DALLAS
8	ADAMS	(null)	DALLAS
9	BLAKE	(null)	CHICAGO
10	MARTIN	(null)	CHICAGO
11	ALLEN	(null)	CHICAGO
12	TURNER	(null)	CHICAGO
13	JAMES	(null)	CHICAGO
14	WARD	(null)	CHICAGO

문제 119. emp_loc 컬럼의 데이터를 dept_loc 컬럼의 데이터로 변경 하시오.

```
ALTER TABLE DEPT
        ADD CONSTRAINT dept_deptno_pk PRIMARY KEY(Deptno);
```



Primary key 제약을 걸지 않으면 -오류발생

```
UPDATE emp700
    SET emp_loc = dept_loc;
```

	ENAME	EMP_LOC	DEPT_LOC
1	KING	NEW YORK	NEW YORK
2	CLARK	NEW YORK	NEW YORK
3	MILLER	NEW YORK	NEW YORK
4	JONES	DALLAS	DALLAS
5	FORD	DALLAS	DALLAS
6	SMITH	DALLAS	DALLAS
7	SCOTT	DALLAS	DALLAS
8	ADAMS	DALLAS	DALLAS
9	BLAKE	CHICAGO	CHICAGO
10	MARTIN	CHICAGO	CHICAGO
11	ALLEN	CHICAGO	CHICAGO
12	TURNER	CHICAGO	CHICAGO
13	JAMES	CHICAGO	CHICAGO
14	WARD	CHICAGO	CHICAGO

■ emp 테이블에 추가된 loc 컬럼을 dept 테이블의 loc 컬럼의 데이터로 변경하는 3가지 방법

1.상호관련 서브쿼리를 이용해서 변경 (악성 SQL , 가장 많이 씀)
2.수정가능한 조인 뷰를 이용해서 변경 (view를 만들고 primary key 제약을 걸어야 함, 번거롭다)
3.merge 문을 이용해서 변경 (수행 속도 빠르고 , 바로 할 수 있음)

문제 120. 다시 rollback을 해서 emp 테이블의 loc 컬럼의 데이터를 dept 테이블의 loc컬럼의 데이터로 변경 하는데
merge 문을 이용해서 수행 하시오.

*Merge를 사용하면 primary key 제약을 걸지 않아도 된다

```
MERGE INTO EMP e
    USING dept d
  ON (e.deptno = d.deptno)
  WHEN matched THEN
        UPDATE SET e.loc = d.loc;

SELECT ename, emp_loc, dept_loc
    FROM emp700;
```

	ENAME	EMP_LOC	DEPT_LOC
1	KING	NEW YORK	NEW YORK
2	BLAKE	CHICAGO	CHICAGO
3	CLARK	NEW YORK	NEW YORK
4	JONES	DALLAS	DALLAS
5	MARTIN	CHICAGO	CHICAGO
6	ALLEN	CHICAGO	CHICAGO
7	TURNER	CHICAGO	CHICAGO
8	JAMES	CHICAGO	CHICAGO
9	WARD	CHICAGO	CHICAGO
10	FORD	DALLAS	DALLAS
11	SMITH	DALLAS	DALLAS
12	SCOTT	DALLAS	DALLAS
13	ADAMS	DALLAS	DALLAS
14	MILLER	NEW YORK	NEW YORK

문제 121. emp 테이블에 loc 컬럼을 dept 테이블에 loc 컬럼의 데이터로 변경하는데 상호관련 서브쿼리를 이용해서 변경 하시오.

```
UPDATE EMP e
    SET e.loc = (SELECT loc
                 FROM DEPT d
                     WHERE e.deptno = d.DEPTNO);
```

	ENAME	LOC	DEPTNO
1	KING	NEW YORK	10
2	BLAKE	CHICAGO	30
3	CLARK	NEW YORK	10
4	JONES	DALLAS	20
5	MARTIN	CHICAGO	30
6	ALLEN	CHICAGO	30
7	TURNER	CHICAGO	30
8	JAMES	CHICAGO	30
9	WARD	CHICAGO	30
10	FORD	DALLAS	20
11	SMITH	DALLAS	20
12	SCOTT	DALLAS	20
13	ADAMS	DALLAS	20
14	MILLER	NEW YORK	10

***왜 악성 SQL 일까?**

Where 절의 d.deptno 에 (10,20,30,40)을 넣어보면서 e.deptno 와 비교하고 e.loc에 d.loc 값을 저장하는
작업을 14번 (emp 로우 수) 만큼 수행하므로 악성 SQL이다.

■ merge 문 실습을 위한 스크립트 생성

```
CREATE TABLE sales300
AS
SELECT *
    FROM sh.SALES;

CREATE TABLE sales400
```

```
AS
SELECT ROWNUM rn, prod_id, cust_id, time_id, channel_id, promo_id, quantity_sold,
amount_sold
        FROM sh.SALES;

ALTER TABLE sales400
        ADD date_id DATE;

CREATE TABLE time2
        (rn number(10),
     date_id DATE );

   SELECT * FROM time2;
    SELECT *
                FROM time2
                WHERE ROWNUM < 10;
```

	RN	DATE_ID
1	1	1961-01-22 오전 12:00:00
2	2	1961-01-23 오전 12:00:00
3	3	1961-01-24 오전 12:00:00
4	4	1961-01-25 오전 12:00:00
5	5	1961-01-26 오전 12:00:00
6	6	1961-01-27 오전 12:00:00
7	7	1961-01-28 오전 12:00:00
8	8	1961-01-29 오전 12:00:00
9	9	1961-01-30 오전 12:00:00

```
    SELECT *
            FROM sales400
        WHERE ROWNUM < 10;
```

	RN	PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD	DATE_ID
1	1	13	987	1998-01-10 오전 12:00:00	3	999	1	1232.16	(null)
2	2	13	1660	1998-01-10 오전 12:00:00	3	999	1	1232.16	(null)
3	3	13	1762	1998-01-10 오전 12:00:00	3	999	1	1232.16	(null)
4	4	13	1843	1998-01-10 오전 12:00:00	3	999	1	1232.16	(null)
5	5	13	1948	1998-01-10 오전 12:00:00	3	999	1	1232.16	(null)
6	6	13	2273	1998-01-10 오전 12:00:00	3	999	1	1232.16	(null)
7	7	13	2380	1998-01-10 오전 12:00:00	3	999	1	1232.16	(null)
8	8	13	2683	1998-01-10 오전 12:00:00	3	999	1	1232.16	(null)
9	9	13	2865	1998-01-10 오전 12:00:00	3	999	1	1232.16	(null)

문제 122. sales400의 data_id 컬럼을 time2의 data_id 컬럼의 데이터로 변경하는데 상호 관련 서브쿼리문으로 수행 하시오.

```
UPDATE sales400 s
        SET s.DATE_ID = ( SELECT t.date_id
                                FROM time2 t
                                WHERE t.rn = s.rn);
```

Update	0 / 0	968 초	71 : 69	삽입	행	ANSI-DOS				가로 그리드	사용자가 작업을 취소했습니다

문제 123. sales400의 data_id를 time2의 date_id로 변경하는데, 수정가능한 조인 뷰를 이용해서 변경 하시오.

1. primary key 제약을 건다.

```
ALTER TABLE time2
        ADD CONSTRAINT time2_rn_pk PRIMARY KEY(rn);
```

2. 뷰를 만든다.

```
CREATE VIEW sales400_view
AS
SELECT s.rn, s.date_id AS s_date_id, t.date_id AS t_date_id
        FROM sales400 s, time2 t
    WHERE s.rn = t.rn;
```

	RN	S_DATE_ID	T_DATE_ID
1	1	(null)	1961-01-22 오전 12:00:00
2	2	(null)	1961-01-23 오전 12:00:00
3	3	(null)	1961-01-24 오전 12:00:00
4	4	(null)	1961-01-25 오전 12:00:00
5	5	(null)	1961-01-26 오전 12:00:00
6	6	(null)	1961-01-27 오전 12:00:00
7	7	(null)	1961-01-28 오전 12:00:00
8	8	(null)	1961-01-29 오전 12:00:00
9	9	(null)	1961-01-30 오전 12:00:00

3.뷰를 update 한다.

```
UPDATE sales400_view
        SET s_Date_id = t_date_id;
```

Update	918843	15 초	88 : 24	삽입	행	ANSI-DOS			가로 그리드	918843 행이 갱신되었습니다.

***상호관련 서브쿼리를 사용하는 것보다 훨씬 빠르다.**

```
CREATE VIEW sales400_view
AS
SELECT s.rn, s.date_id AS s_date_id, t.date_id AS t_date_id
        FROM sales400 s, time2 t
    WHERE s.rn = t.rn;
```

위의 select 문을 뷰의 자리에 써준다.

*** 뷰를 생성하지 않을 때 -- 수정가능한 조인 뷰 (상당히 유용하다)**

```
ALTER TABLE sales400
        DROP COLUMN date_id;      -- 이전 데이터 삭제

ALTER TABLE sales400
        add date_id DATE;

UPDATE (
```

```
            SELECT s.rn, s.date_id AS s_date_id, t.date_id AS t_date_id
             FROM sales400 s, time2 t
                WHERE s.rn = t.rn)
        SET s_Date_id = t_date_id;
```

Update	918843	22 초	87 : 1	삽입	행	ANSI-DOS				가로 그리드

문제 124. sales400의 data_id를 time2의 date_id로 변경하는데, merge문을 이용해서 변경 하시오.

```
MERGE INTO sales400 s
      USING time2 t
   ON (s.rn = t.rn)
   WHEN matched THEN
         UPDATE SET s.date_id = t.date_id;
```

***뷰, primary 제약 조건을 추가할 필요가 없다.**

Merge	918843	16 초	103 : 20	삽입	행	ANSI-DOS				가로 그리드	918843 행이 병합되었습니다.

21. 분석함수를 이용한 SQL 튜닝

2018년 5월 1일 화요일 오후 2:28

" 데이터 분석함수를 이용하면 복잡한 쿼리를 간단히 수행할 수 있다. 코딩도 간단해지고 성능도 좋아진다. "

21.1 예제

튜닝 전 :

```
SELECT deptno, SUM(sal)
    FROM EMP
 GROUP BY deptno
UNION
SELECT NULL, sum(sal)
    From EMP;
```

	속성	값
1	recursive calls	0
2	db block gets	0
3	consistent gets	55

튜닝 후 :

```
SELECT deptno, SUM(sal)
    FROM EMP
      GROUP BY ROLLup(deptno);
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	48

문제 125. 아래 SQL을 튜닝 하시오.

-튜닝 전-

```
SELECT deptno, NULL AS JOB, SUM(sal)
    FROM EMP
      GROUP BY deptno
UNION
SELECT NULL AS DEPTNO , job, sum(sal)
    From EMP
     GROUP BY job;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	55

-튜닝 후-

```
SELECT deptno , job, SUM(sal)
        FROM EMP
    GROUP BY GROUPING sets(deptno,job);
```

1	recursive calls	4
2	db block gets	24
3	consistent gets	62

--> grouping sets를 쓰니까 더 느려짐

```
SELECT /*+ expand_gset_to_union */deptno , job, SUM(sal)
        FROM EMP
        GROUP BY GROUPING sets(deptno,job);
```

expand_gset_to_union : 쿼리변환기에게 grouping sets를 다시 union으로 수행하라고 명령하는 힌트

1	recursive calls	0
2	db block gets	0
3	consistent gets	55

문제 126.(점심시간 문제) 아래의 SQL을 튜닝 하시오.

-튜닝 전-

```
SELECT DEPTNO, ENAME, SUM(SAL)
        FROM EMP
    GROUP BY DEPTNO, ENAME
UNION ALL
SELECT NULL AS DEPTNO, NULL AS ENAME,SUM(SAL)
        FROM EMP
    ORDER BY DEPTNO;
```

	DEPTNO	ENAME	SUM(SAL)
1	10	MILLER	1300
2	10	KING	5000
3	10	CLARK	2450
4	20	ADAMS	1100
5	20	SMITH	800
6	20	FORD	3000
7	20	JONES	2975
8	20	SCOTT	3000
9	30	MARTIN	1250
10	30	BLAKE	2850
11	30	JAMES	950
12	30	WARD	1250
13	30	ALLEN	1600
14	30	TURNER	1500
15	(null)	(null)	29025

1	recursive calls	0
2	db block gets	0
3	consistent gets	55

```
SELECT DEPTNO,ENAME, SUM(SAL)
        FROM EMP
    GROUP BY GROUPING sets((DEPTNO,ENAME),());
```

	DEPTNO	ENAME	SUM(SAL)
1	10	KING	5000
2	10	CLARK	2450
3	10	MILLER	1300
4	20	FORD	3000
5	20	ADAMS	1100
6	20	JONES	2975
7	20	SCOTT	3000
8	20	SMITH	800
9	30	WARD	1250
10	30	ALLEN	1600
11	30	BLAKE	2850
12	30	JAMES	950
13	30	MARTIN	1250
14	30	TURNER	1500
15	(null)	(null)	29025

1	recursive calls	0
2	db block gets	0
3	consistent gets	48

문제 127. 아래의 SQL을 분석함수를 써서 튜닝 하시오.

-튜닝 전-

```
SELECT e.ename, e.sal, e.deptno, d.avgsal
        FROM EMP e, (SELECT deptno, AVG(sal) avgsal
                                FROM EMP
                                                GROUP BY deptno ) d
        WHERE e.deptno = d.deptno;
```

1	WARD	1250	30	1566.6666666666666666666666666666666667
2	JAMES	950	30	1566.6666666666666666666666666666666667
3	TURNER	1500	30	1566.6666666666666666666666666666666667
4	ALLEN	1600	30	1566.6666666666666666666666666666666667
5	MARTIN	1250	30	1566.6666666666666666666666666666666667
6	BLAKE	2850	30	1566.6666666666666666666666666666666667
7	ADAMS	1100	20	2175
8	SCOTT	3000	20	2175
9	SMITH	800	20	2175
10	FORD	3000	20	2175
11	JONES	2975	20	2175
12	MILLER	1300	10	2916.6666666666666666666666666666666667
13	CLARK	2450	10	2916.6666666666666666666666666666666667
14	KING	5000	10	2916.6666666666666666666666666666666667

1	recursive calls	0
2	db block gets	0
3	consistent gets	14

-튜닝 후-

```
SELECT ename, sal, deptno, AVG(sal) OVER (PARTITION BY deptno) avgsal
```

```
        FROM EMP;
```

문제 128. 아래의 SQL을 튜닝 하시오.

-튜닝 전-

```
SELECT e.empno, e.ename, e2.job, e2.max_sal
        FROM EMP e, (SELECT job, max(sal) AS max_sal
                                FROM EMP
                    GROUP BY job) e2
    WHERE e.job = e2.job
            AND e.sal = e2.max_sal;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	14

	EMPNO	ENAME	JOB	MAX_SAL
1	7499	ALLEN	SALESMAN	1600
2	7934	MILLER	CLERK	1300
3	7839	KING	PRESIDENT	5000
4	7566	JONES	MANAGER	2975
5	7788	SCOTT	ANALYST	3000
6	7902	FORD	ANALYST	3000

```
SELECT *
        FROM (SELECT empno, ename, job, sal,rank() over (PARTITION BY job ORDER BY sal desc) 순위
        FROM EMP)
    WHERE 순위 = 1;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	7

문제 129. 아래의 SQL을 튜닝 하시오.

```
SELECT empno, ename, sal, (SELECT SUM(sal) FROM EMP e WHERE e.empno <= b.empno) sumsal
        FROM EMP b
    ORDER BY empno;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	144

	EMPNO	ENAME	SAL	SUMSAL
1	7369	SMITH	800	800
2	7499	ALLEN	1600	2400
3	7521	WARD	1250	3650
4	7566	JONES	2975	6625
5	7654	MARTIN	1250	7875
6	7698	BLAKE	2850	10725
7	7782	CLARK	2450	13175
8	7788	SCOTT	3000	16175
9	7839	KING	5000	21175
10	7844	TURNER	1500	22675
11	7876	ADAMS	1100	23775
12	7900	JAMES	950	24725
13	7902	FORD	3000	27725
14	7934	MILLER	1300	29025

```
SELECT empno, ename, sal,
        SUM(sal) OVER (ORDER BY empno ROWS BETWEEN unbounded preceding AND CURRENT row) sumsal
                FROM EMP b;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	46

*unbounded preceding = 첫 번째 행
Unbounded following = 마지막 행
current row = 현재 행

Ex) **ROWS BETWEEN unbounded preceding AND CURRENT row : 처음부터 현재 로우까지의 합**
ROWS BETWEEN unbounded preceding AND unbounded following : 처음부터 마지막 로우까지의 총합

문제 130. 아래의 SQL을 튜닝 하시오.

```
SELECT deptno, empno, ename, sal ,
        (SELECT SUM(sal) FROM EMP e WHERE  e.empno <= b.empno AND e.deptno = b.deptno ) sumsal
        FROM EMP b
    ORDER BY deptno, empno;
```

	DEPTNO	EMPNO	ENAME	SAL	SUMSAL
1	10	7782	CLARK	2450	2450
2	10	7839	KING	5000	7450
3	10	7934	MILLER	1300	8750
4	20	7369	SMITH	800	800
5	20	7566	JONES	2975	3775
6	20	7788	SCOTT	3000	6775
7	20	7876	ADAMS	1100	7875
8	20	7902	FORD	3000	10875
9	30	7499	ALLEN	1600	1600
10	30	7521	WARD	1250	2850
11	30	7654	MARTIN	1250	4100
12	30	7698	BLAKE	2850	6950
13	30	7844	TURNER	1500	8450
14	30	7900	JAMES	950	9400

1	recursive calls	0
2	db block gets	0
3	consistent gets	144

SELECT deptno, empno, ename, sal, **SUM(sal) OVER (PARTITION BY deptno ORDER BY empno)** sumsal
FROM EMP ;

1	recursive calls	4
2	db block gets	0
3	consistent gets	54

문제 131. 아래의 SQL을 튜닝 하시오.

```
SELECT a.deptno, a.empno, a.ename, a.sal, b.sal
      FROM ( SELECT ROWNUM no1, deptno, empno, SAL, ename
                    FROM ( SELECT deptno, empno, ename, SAL
                        FROM EMP
                  ORDER BY deptno, empno)) a,
         (SELECT rownum+1 no2, deptno, empno, ename, sal
                  FROM (SELECT deptno, empno, ename, SAL
                              FROM EMP
                    ORDER BY deptno, empno))b
   WHERE a.no1= b.no2 (+)
   ORDER BY no1;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	14

	DEPTNO	EMPNO	ENAME	SAL	SAL_1
1	10	7782	CLARK	2450	(null)
2	10	7839	KING	5000	2450
3	10	7934	MILLER	1300	5000
4	20	7369	SMITH	800	1300
5	20	7566	JONES	2975	800
6	20	7788	SCOTT	3000	2975
7	20	7876	ADAMS	1100	3000
8	20	7902	FORD	3000	1100
9	30	7499	ALLEN	1600	3000
10	30	7521	WARD	1250	1600
11	30	7654	MARTIN	1250	1250
12	30	7698	BLAKE	2850	1250
13	30	7844	TURNER	1500	2850
14	30	7900	JAMES	950	1500

SELECT deptno, empno, ename, sal, LAG(sal,1) OVER (ORDER BY deptno, empno) sal_1
FROM EMP;

1	recursive calls	0
2	db block gets	0
3	consistent gets	46

문제 132. 아래의 SQL을 튜닝 하시오.

```
SELECT DECODE(NO,1,deptno,2,null) AS deptno, SUM(sal)
      FROM EMP e, (SELECT ROWNUM NO FROM dual CONNECT BY LEVEL <= 2) d
   GROUP BY DECODE(NO,1,e.deptno,2,null)
   ORDER BY DECODE(NO,1,e.deptno,2,null);
```

	DEPTNO	SUM(SAL)
1	10	8750
2	20	10875
3	30	9400
4	(null)	29025

```
SELECT deptno, SUM(sal)
      FROM EMP
   GROUP BY ROLLUP(deptno);
```

문제 133. 아래의 SQL을 튜닝 하시오.

```
SELECT ename, job
      FROM EMP e1
   WHERE 4<= (SELECT COUNT(*)
                          FROM EMP e2
                WHERE  e2.job = e1.job);
```

	ENAME	JOB
1	MARTIN	SALESMAN
2	ALLEN	SALESMAN
3	TURNER	SALESMAN
4	JAMES	CLERK
5	WARD	SALESMAN
6	SMITH	CLERK
7	ADAMS	CLERK
8	MILLER	CLERK

1	recursive calls	0
2	db block gets	0
3	consistent gets	81

```
SELECT *
      FROM (SELECT ename, job, COUNT(*) OVER (PARTITION BY job) cnt
                FROM EMP)
   WHERE cnt >= 4;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	7

문제 134. 아래의 SQL을 튜닝 하시오.

```
SELECT *
        FROM (SELECT t.*, COUNT(*) OVER (PARTITION BY s.time_id) cc
                FROM sales100 s, TIMES100 t
                    WHERE s.time_id = t.TIME_Id)
         WHERE cc > 20;
```

1	recursive calls	224
2	db block gets	10
3	consistent gets	4496

	TIME_ID	DAY_NAME	DAY_NUMBER_IN_WEEK	DAY_NUMBER_IN_MONTH	CALENDAR_WEEK_NUMBER	FISCAL_WEEK_NUMBER	WEEK_ENDING_DAY	WEEK_ENDING_DAY_ID	CALENDAR_MONTH_NUMBER	FISC
1	1998-01-01 오전 12:00:00	Thursday	4	1	1	1	1998-01-04 오전 12:00:00	1484	1	
2	1998-01-02 오전 12:00:00	Friday	5	2	1	1	1998-01-04 오전 12:00:00	1484	1	
3	1998-01-03 오전 12:00:00	Saturday	6	3	1	1	1998-01-04 오전 12:00:00	1484	1	
4	1998-01-04 오전 12:00:00	Sunday	7	4	1	1	1998-01-04 오전 12:00:00	1484	1	
5	1998-01-05 오전 12:00:00	Monday	1	5	2	2	1998-01-11 오전 12:00:00	1533	1	
6	1998-01-06 오전 12:00:00	Tuesday	2	6	2	2	1998-01-11 오전 12:00:00	1533	1	
7	1998-01-07 오전 12:00:00	Wednesday	3	7	2	2	1998-01-11 오전 12:00:00	1533	1	
8	1998-01-08 오전 12:00:00	Thursday	4	8	2	2	1998-01-11 오전 12:00:00	1533	1	
9	1998-01-09 오전 12:00:00	Friday	5	9	2	2	1998-01-11 오전 12:00:00	1533	1	
10	1998-01-10 오전 12:00:00	Saturday	6	10	2	2	1998-01-11 오전 12:00:00	1533	1	
11	1998-01-11 오전 12:00:00	Sunday	7	11	2	2	1998-01-11 오전 12:00:00	1533	1	
12	1998-01-12 오전 12:00:00	Monday	1	12	3	3	1998-01-18 오전 12:00:00	1581	1	
13	1998-01-13 오전 12:00:00	Tuesday	2	13	3	3	1998-01-18 오전 12:00:00	1581	1	
14	1998-01-14 오전 12:00:00	Wednesday	3	14	3	3	1998-01-18 오전 12:00:00	1581	1	

```
select *
        from
        (select time_id,count(*) cnt
                from sales100 group by time_id) s, times100 t
                where t.time_id=s.time_id and cnt >20;
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	4595

문제 135. 튜닝 전 SQL결과 데이터와 튜닝 후 SQL 결과 데이터가 서로 같은지 확인 하시오. (데이터 검증)

```
select t.*
        from
                (select time_id,count(*) cnt
                from sales100 group by time_id) s, times100 t
                where t.time_id=s.time_id and cnt >20
minus
select *
        from times100 t
        where 20< (select  count(*) from sales100 s
                where s.time_id=t.time_id);
```

TIME_ID	DAY_NAME	DAY_NUMBER_IN_WEEK	DAY_NUMBER_IN_MONTH	CALENDAR_WEEK_NUMBER	FISCAL_WEEK_NUMBER	WEEK_ENDING_DAY	WEEK_ENDING_DAY_ID	CA

표시할 데이터가 없습니다.

22. with절을 이용한 튜닝

2018년 5월 1일 화요일 오후 2:28

" 비슷한 패턴의 SQL이 하나의 SQL에서 여러 개로 사용되면서 성능이 느릴 때 유용한 튜닝 방법 "

■ **with절 튜닝시 힌트**

1. /*+ materialize */ : temp 테이블을 사용하겠다.
2. /*+ inline */ : temp 테이블 사용안하고 서브쿼리로 수행하겠다.

22.1 예제

예제 1. 직업, 직업별 토탈 월급을 출력 하시오.

```
SELECT job, SUM(sal)
      FROM EMP
      GROUP BY job;
```

	JOB	SUM(SAL)
1	SALESMAN	5600
2	CLERK	4150
3	PRESIDENT	5000
4	MANAGER	8275
5	ANALYST	6000

문제 136. 직업별 토탈 월급들의 평균값을 출력 하시오.

```
SELECT AVG(SUM(sal))
      FROM EMP
    GROUP BY job;
```

	AVG(SUM(SAL))
1	5805

문제 137. 직업, 직업별 토탈 월급을 출력하는데 직업별 토탈 월급들에 대한 평균값보다 더 큰 것만 출력 하시오.

```
SELECT job, SUM(sal)
      FROM EMP
        GROUP BY job
         HAVING SUM(sal) > (SELECT AVG(SUM(sal))
                            FROM EMP
                            GROUP BY job);
```

	JOB	SUM(SAL)
1	MANAGER	8275
2	ANALYST	6000

문제 138. 위의 SQL을 with 절로 수행 하시오.

```
WITH job_sum AS (SELECT job, SUM(sal) sumsal
                 FROM EMP
                      GROUP BY job )
SELECT job, sumsal
    FROM job_sum
      WHERE sumsal > (SELECT AVG(sumsal)
                      FROM job_sum);
```

	JOB	SUMSAL
1	MANAGER	8275
2	ANALYST	6000

1	recursive calls	2
2	db block gets	8
3	consistent gets	54

SELECT STATEMENT Optimizer Mode=ALL_ROWS	-	14
TEMP TABLE TRANSFORMATION		
LOAD AS SELECT	.SYS_TEMP_0FD9D6617_2EBB555	
HASH GROUP BY		14
TABLE ACCESS FULL	SCOTT.EMP	14
VIEW		14
TABLE ACCESS FULL	SYS.SYS_TEMP_0FD9D6617_2EBB555	14
SORT AGGREGATE		1
VIEW		14
TABLE ACCESS FULL	SYS.SYS_TEMP_0FD9D6617_2EBB555	14

***with 절은 임시 테이블을 만들고 with 절이 끝나면 사라진다.**

문제 139. 위의 with 절이 temp 테이블을 사용하지 않도록 힌트를 주고 실행 하시오.

```
WITH job_sum AS (SELECT /*+ inline */ job, SUM(sal) sumsal
                 FROM EMP
                      GROUP BY job )
SELECT job, sumsal
    FROM job_sum
      WHERE sumsal > (SELECT  AVG(sumsal)
                      FROM job_sum);
```

1	recursive calls	0
2	db block gets	0
3	consistent gets	53

문제 140. 아래의 SQL을 튜닝 하시오.

```
WITH V AS (SELECT /*+ materialize */ deptno, SUM(sal) AS sumsal
                  FROM EMP
                  WHERE job = 'SALESMAN'
                        GROUP BY deptno )
SELECT d.deptno, d.dname, v.sumsal
    FROM DEPT d, v
      WHERE d.deptno = v.deptno
       AND v.sumsal = (SELECT MAX(v.sumsal) FROM v);
```

	DEPTNO	DNAME	SUMSAL
1	30	SALES	5600

1	recursive calls	2
2	db block gets	8
3	consistent gets	61

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		4
TEMP TABLE TRANSFORMATION		
LOAD AS SELECT	.SYS_TEMP_0FD9D661D_2EBB555	
HASH GROUP BY		4
TABLE ACCESS FULL	SCOTT.EMP	4
HASH JOIN		4
TABLE ACCESS FULL	SCOTT.DEPT	4
VIEW		4
TABLE ACCESS FULL	SYS.SYS_TEMP_0FD9D661D_2EBB555	4
SORT AGGREGATE		1
VIEW		4
TABLE ACCESS FULL	SYS.SYS_TEMP_0FD9D661D_2EBB555	4

```
WITH V AS (SELECT /*+ inline */ deptno, SUM(sal) AS sumsal
                  FROM EMP
                  WHERE job = 'SALESMAN'
                        GROUP BY deptno )
SELECT d.deptno, d.dname, v.sumsal
    FROM DEPT d, v
      WHERE d.deptno = v.deptno
       AND v.sumsal = (SELECT MAX(v.sumsal) FROM v);
```

Operation	Object Name	Rows
SELECT STATEMENT Optimizer Mode=ALL_ROWS		4
HASH JOIN		4
TABLE ACCESS FULL	SCOTT.DEPT	4
VIEW		4
WINDOW BUFFER		4
HASH GROUP BY		4
TABLE ACCESS FULL	SCOTT.EMP	4

1	recursive calls	0
2	db block gets	0
3	consistent gets	53