

# ##1116

2020년 11월 16일 월요일    오후 3:47

## 튜닝의 필요성

데이터 분석가들에게 SQL 작성능력은 필수

데이터 분석가들이 데이터 분석을 하는 회사들의 데이터가 대부분 대용량이기 때문에 검색속도가 느리다.

때문에 SQL 튜닝 기술이 분석가들에게 요구된다

SQL튜닝 : data 검색속도를 향상시키는 기술

## ■ 1. index access

※ index access	hint
1. index range scan	index
2. index unique scan	index
3. index skip scan	index_ss
4. index full scan	index_fs
5. index fast full scan	index_ffs
6. index merge scan	and_equal
7. index bitmap merge scan	index_combine
8. index join	index_join

hint : oracle optimizer 가 SQL 을 수행시 실행계획을 만드는데 실행계획을 SQL 사용자가 조정하는 명령어  
optimizer 에게 문법에 맞는 적절한 hint 를 통해 사용자의 요청대로 SQL 을 실행할 수 있다.

※ 실행계획 보는 방법 1

**explain plan for**

**select ename, sal**

**from emp**

**where sal = 1400;**

**select \* from table(dbms\_xplan.display);**

PLAN_TABLE_OUTPUT									
1 Plan hash value: 3956160932									
2									
3									
4   Id   Operation   Name   Rows   Bytes   Cost (%CPU)   Time									
5									
2	6	0   SELECT STATEMENT		1	20	3 (0)	00:00:01		
7	7	* 1   TABLE ACCESS FULL   EMP		1	20	3 (0)	00:00:01		
8									
9									
10 Predicate Information (identified by operation id):									
11									
12									
13 1 - filter("SAL">=1400)									
14									
15 Note									
16 -----									
17 - dynamic statistics used: dynamic sampling (level=2)									

실행계획에 full table scan 으로 나오면 emp 테이블을 처음부터 끝까지 다 스캔했다는 뜻이다  
위 실행계획을 보는 방법은 SQL 의 결과값은 볼 수 없고 실행계획만 확인할 수 있다

※ 실행계획 보는 방법 2 (결과값까지 포함하는 실행계획)

```
select /*+ gather_plan_statistic */ ename, sal
  from emp
 where sal = 1300;
```

SELECT \* FROM TABLE(dbms\_xplan.display\_cursor(null,null,'ALLSTATS LAST'));

```

PLAN_TABLE_OUTPUT
1 SQL_ID 7zc4t63q37hx7, child number 0
2 -----
3 select /*+ gather_plan_statistics */ ename, sal      from emp      where
4 sal = 1300
5
6 Plan hash value: 3956160932
7
8 -----
9 | Id | Operation          | Name | Starts | E-Rows | A-Rows | A-Time | Buffers |
10 -----
11 | 0 | SELECT STATEMENT   |      | 1      |        | 1      | 00:00:00.01 | 7 |
12 | * 1 | TABLE ACCESS FULL| EMP  | 1      | 1      | 1      | 00:00:00.01 | 7 |
13 -----
14
15 Predicate Information (identified by operation id):
16 -----
17
18 1 - filter("SAL">=1300)
19
20 Note
21 -----
22 - dynamic statistics used: dynamic sampling (level=2)
23

```

8행부터 13행 사이에 있는 buffers 개수가 줄어들수록 튜닝이 잘 된 것이다.

ex1) 위 SQL 을 튜닝하시오! (붉은글씨)

1. emp 테이블의 sal 에 index 생성

```
create index emp_sal
  on emp(sal);
```

2. 재 검색

```
select /*+ gather_plan_statistics */ ename, sal
  from emp
 where sal = 1300;
```

SELECT \* FROM TABLE(dbms\_xplan.display\_cursor(null,null,'ALLSTATS LAST'));

8 -----									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10 -----									
11	0	SELECT STATEMENT		1		1	00:00:00.01	2	
12	1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	1	1	1	00:00:00.01	2	
13	* 2	INDEX RANGE SCAN	EMP_SAL	1	1	1	00:00:00.01	1	
14 -----									

실행계획의 읽는 순서는 13-12-11 행으로 읽어주면 된다

위 실행계획을 참고하면 index scan 을 했고, buffer 가 줄어든 것을 확인할 수 있다.

1) 아래의 SQL을 튜닝하시오!

튜닝전 : 실행계획, 버퍼의 개수 비교

```
select empno, ename, sal, job
  from emp
 where empno = 7788;
```

8								
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10								
11	0	SELECT STATEMENT		1		1	00:00:00.01	7
12	* 1	TABLE ACCESS FULL	EMP	1	1	1	00:00:00.01	7
13								

튜닝후 : 실행계획, 버퍼의 개수 비교

>>1 create index

create index emp\_empno

on emp(empno);

>>2 display

select /\*+ gather\_plan\_statistics \*/ empno, ename, sal, job

from emp

where empno=7788;

SELECT \* FROM TABLE(dbms\_xplan.display\_cursor(null,null,'ALLSTATS LAST'));

8								
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10								
11	0	SELECT STATEMENT		1		1	00:00:00.01	2
12	1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	1	1	1	00:00:00.01	2
13	* 2	INDEX RANGE SCAN	EMP_EMPNO	1	1	1	00:00:00.01	1
14								

튜닝전과 비교해 buffer 가 7에서 1로 줄어든 것을 알 수 있다.

table 삭제시 index 도 삭제되었다가 recyclebin 에서 테이블을 복구하면 같이 복구되지만 이름이 이상하게 생성된다.

##22) 오일러 상수 자연상수의 근사값을 SQL로 구하시오!

undefine n

select power( (1+1/&n), &n) as e

from dual;

생각해야할 문제. SQL 알고리즘12번. 오일러 상수 자연상수의 근사값을 SQL 로 구하시오 ~

$$e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n = 2.71828182845904523 \dots$$

SQL로 구현
근사근 단축

# ##1117

2020년 11월 17일 화요일 오전 9:44

SQLD : SQL 개발자 (SQL 수업)

SQLP : SQL 튜너 (SQL 수업 + SQL 튜닝 수업)

ADSP : 데이터 분석가

## ■ SQL 튜닝 목차

1. index SQL 튜닝
2. 조인 문장 튜닝
3. 서브쿼리 문장 튜닝
4. 데이터 분석함수를 이용한 튜닝
5. 자동 SQL 튜닝

index access

※ index access	hint
1. index range scan	index
2. index unique scan	index
3. index skip scan	index_ss
4. index full scan	index_fs
5. index fast full scan	index_ffs
6. index merge scan	and_equal
7. index bitmap merge scan	index_combine
8. index join	index_join

full table scan 과 index scan 의 차이를 실행계획을 통해 경험(어제)

## □ 1. index range scan

인덱스를 range(부분) 하게 먼저 스캔하고 인덱스에서 얻은 rowid 로 테이블의 데이터를 액세스하는 스캔 방법  
ex) 책

- 목차(index) / 책(table) / 페이지번호(rowid)
- 인덱스가 없다면 원하는 데이터를 검색하기 위해서 full table scan 을 할 수 밖에 없지만 인덱스가 있다면 인덱스(목차)를 통해서 먼저 데이터를 검색하고 인덱스의 rowid 를 통해서 테이블을 액세스 할 수 있게 된다.

ex2) 실습

```
1) select ename, sal
   from emp
   where ename = 'BLAKE';
```

↑

create index emp\_enmae

on emp(ename) <<< 인덱스를 걸어 놓으면 검색속도가 매우 향상된다

위 index emp\_ename 생성 후 재 검색하면(힌트포함해서) index range scan 의 결과를 알 수 있다

```
select /*+ gather_plan_statistics index(emp emp_ename) */ ename, sal
```

```
from emp
```

```
where ename='BLAKE';
```

```
SELECT * FROM TABLE(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

↑

패키지

↑

프로시저(인자값1, 인자값2, 인자값3)

2) 실행계획

예상 실행계획 : SQL 을 실행하기 전에 실행계획을 예상한 계획

**explain plan for ~ select \* from table(dbms\_xplan.display);**

실제 실행계획 : SQL 을 직접 실행하면서 바로 나오는 그 실행계획 (보통 주로 이걸 많이 사용함)

**gather\_plan\_statistics hint**

3) hint 해석

**index(emp emp\_ename) :** emp 테이블의 emp\_ename 의 인덱스를 따라 scan 하시오

4) index scan 이 빠른 이유는 rowid 로 정렬된 상태에서 그 위치로 곧장 scan 한 뒤에 마무리하기때문이다.

index 가 걸렸다는 이유는 rowid 가 특정 column 에 대해서 정렬되어있는것이기 때문

5) buffer : SQL 을 처리하기 위해서 읽어들이 메모리 블록의 갯수

★(이 표는 문제를 의미/애매할때만 사용)

2) 아래의 SQL을 튜닝하시오. 튜닝전과 튜닝후의 실행계획을 비교하시오. (buffer 의 개수까지 확인하시오)

>>1 before tuning

select ename, sal, job

from emp

where substr(job,1, 5) = 'SALES';

	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
	0	SELECT STATEMENT		1		4	00:00:00.01	7
*	1	TABLE ACCESS FULL	EMP	1	4	4	00:00:00.01	7

>>2 after tuning

create index emp\_job

on emp(job);

select /\*+ gather\_plan\_statistics \*/ ename, sal, job

from emp

where job = 'SALES%';

select \* from table(dbms\_xplan.display\_cursor(null,null,'allstats last'));

	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
	0	SELECT STATEMENT		1		0	00:00:00.01	1
	1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	1	1	0	00:00:00.01	1
*	2	INDEX RANGE SCAN	EMP_JOB	1	1	0	00:00:00.01	1

만약 이렇게 했는데 fullscan 한다면 hint 부분에 **index(emp emp\_job)** 을 추가하여 index 하도록 옵티마이저에게 지정

3) 아래의 SQL을 튜닝하시오!

create index emp\_sal on emp(sal);

>>1 before tuning

select /\*+ gather\_plan\_statistics \*/ ename, sal

from emp

where sal\*12 = 36000;

	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
	0	SELECT STATEMENT		1		2	00:00:00.01	7
*	1	TABLE ACCESS FULL	EMP	1	2	2	00:00:00.01	7

>>2 after

create index emp\_sal

on emp(sal);

select /\*+ gather\_plan\_statistics \*/ ename, sal

from emp

where sal= 36000/12;

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

10	-----									
11	0	SELECT STATEMENT			1		2	00:00:00.01	2	
12	1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP		1	2	2	00:00:00.01	2	
13	* 2	INDEX RANGE SCAN	EMP_SAL		1	2	2	00:00:00.01	1	
14	-----									

#### 4) 아래의 SQL을 튜닝하시오!

>>1 before

```
select /*+ gather_plan_statistics */ ename, hiredate
  from emp
  where to_char(hiredate, 'RRRR')='1980';
```

8	-----									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers		
10	-----									
11	0	SELECT STATEMENT		1		1	00:00:00.01	7		
12	* 1	TABLE ACCESS FULL	EMP	1	1	1	00:00:00.01	7		
13	-----									

>>2 after

```
create index emp_hiredate on emp(hiredate);
```

```
select /*+ gather_plan_statistics index(hiredate, emp_hiredate) */ ename, hiredate
  from emp
```

**where hiredate between to\_date('1980/01/01', 'RRRR/MM/DD') and to\_date('1980/12/31', 'RRRR/MM/DD');**

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

9	-----									
10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers		
11	-----									
12	0	SELECT STATEMENT		1		1	00:00:00.01	2		
13	* 1	FILTER		1		1	00:00:00.01	2		
14	2	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	1	1	1	00:00:00.01	2		
15	* 3	INDEX RANGE SCAN	EMP_HIREDATE	1	1	1	00:00:00.01	1		
16	-----									

#### 5) 아래의 SQL을 튜닝하시오!

>>1 before

```
select /*+ gather_plan_statistics */ empno, ename, sal
  from emp
  where empno || ename = '7788SCOTT';
```

8	-----									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers		
10	-----									
11	0	SELECT STATEMENT		1		1	00:00:00.01	7		
12	* 1	TABLE ACCESS FULL	EMP	1	1	1	00:00:00.01	7		
13	-----									

>>2 after

```
create index emp_empno on emp(empno);
```

```
select /*+ gather_plan_statistics */ empno, ename, sal
  from emp
  where empno = 7788 and ename='SCOTT';
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	-----									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers		
10	-----									
11	0	SELECT STATEMENT		1		1	00:00:00.01	2		
12	* 1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	1	1	1	00:00:00.01	2		
13	* 2	INDEX RANGE SCAN	EMP_EMPNO	1	1	1	00:00:00.01	1		
14	-----									

#### 6) 아래의 SQL을 튜닝하시오!

```
select /*+ gather_plan_statistics */ ename, sal
from emp
where sal='3000';
```

18	
19	2 - access( 'SAL'=3000)
20	

위 SQL을 실행한 후 계획을 보면 3000 의 양옆에 '(single q.)' 이 없어진 것을 볼 수 있다. 이는 옵티마이저가 암시적으로 문자형으로 작성된 '3000' 을 숫자형인 3000으로 변경하여 select 한 것을 의미. 이는 문자형보다 숫자형이 더 우선순위가 높게 설정되어있어서 이러한 결과가 출력된다. buffer 도 큰 변화가 없다. 이 SQL 은 튜닝해도 buffer에는 큰 변화가 없지만 3000 으로 작성해주는것이 더 좋다.

## 7) 아래의 SQL 을 튜닝하시오!

>>1 before

```
select /*+ gather_plan_statistics */ ename, sal
from emp
where sal like '30%';
```

↓       ↓  
숫자형   문자형

8	-----							
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10	-----							
11	0	SELECT STATEMENT		1		2	00:00:00.01	7
12	* 1	TABLE ACCESS FULL	EMP	1	2	2	00:00:00.01	7
13	-----							

위 SQL 도 옵티마이저가 암시적으로 형변환을 한 결과인데, 실행결과 중 아래를 참고하면

```
1 - filter(TO_CHAR('SAL') LIKE '30%')
```

sal 을 문자형으로 변환해 주었다는 것을 알 수 있다. >> 좌변이 가공된 결과로 출력되었기 때문에 full scan된다. 또한 emp\_sal index 가 존재하고, hint 에 이 조건을 추가해주어도 적용되지 않는다.

**SQL에서 조건절에 like 를 자주 사용하는 쿼리의 테이블의 컬럼은 테이블 생성시부터 숫자형이 아니라 문자형으로 만들어 주어야 한다.**

>>2 after : 함수기반 index 를 생성

```
create index emp_sal_func on emp(to_char(sal));
select /*+ gather_plan_statistics index(emp emp_sal_func) */ ename, sal
from emp
where sal like '30%';
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	-----							
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10	-----							
11	0	SELECT STATEMENT		1		2	00:00:00.01	2
12	1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	1	2	2	00:00:00.01	2
13	* 2	INDEX RANGE SCAN	EMP_SAL_FUNC	1	2	2	00:00:00.01	1
14	-----							

☆##23) 구글입사문제..?

1부터 10000 까지의 숫자 중에서 숫자 8이 총 몇번 나오는가?

(단, 8이 포함되어있는 숫자의 개수를 카운팅하는게 아니라 8이라는 숫자를 모두 카운팅해야한다)

예를 들어 8808 은 3, 8888은 4로 카운팅 해야 한다.

결과 : 4000

hint : 계층형 질의문, regexp(count)

```
select sum(regexp_count(level, 8)) as google
from dual
connect by level <= 10000;
```

### ※ 암시적 형변환 사용시 주의사항

```
select decode( job, 'PRESIDENT', null, sal)
      from emp;
```

#### 9) 위 결과에서 최대값을 출력하시오!

```
select max(decode( job, 'PRESIDENT', null, sal))
      from emp;
```

↑ ↑ ↑

이렇게 SQL 을 하면 실제 최대값인 3000이 아닌 950이 나온다. 왜?

: 암시적 형변환이 발생했기 때문이다.

**decode(job, 'PRESIDENT', null, sal) 을 사용하면 decode 의 세번째 인자값이 null이면 네번째 인자값의 출력이 문자형으로 암시적 형변환이 발생한다.**

```
select to_char(sal) as salary
      from emp
      order by salary desc;
```

이 SQL을 보면 어떤 일이 일어났는지 확인할 수 있다.

#### 10) 아래의 SQL 을 수정해서 원래 예상했던 값이 출력되도록 하시오! (암시적 형변환이 일어나지 않게 하시오!)

```
select max(decode( job, 'PRESIDENT', null, sal))
      from emp;
```

>>수정후

```
select max(decode( job, 'PRESIDENT', 0, sal))
      from emp;
```

### index range scan

non unique 한 인덱스를 액세스하는 스캔 방법

1. unique 인덱스 : 값이 중복되지 않고 unique 해야 생성되는 인덱스
2. non unique 인덱스 : 값이 중복되어도 상관없이 그냥 생성되는 인덱스

ex)

1. unique index

```
create unique index emp_empno on emp(empno); <<< 생성가능
```

```
create unique index emp_deptno on emp(deptno); <<< 생성불가
```

ORA-01452: 중복 키가 있습니다. 유일한 인덱스를 작성할 수 없습니다

01452. 00000 - "cannot CREATE UNIQUE INDEX; duplicate keys found"

emp 테이블의 데이터를 보면 empno 는 중복데이터가 없고 deptno 는 중복데이터가 있다.

#### 11) 사원 이름에 non unique 인덱스를 걸고 이름이 SCOTT인 사원의 이름과 월급을 조회하는 쿼리문의 실제 실행계획을 확인하시오!

```
create index emp_ename on emp(ename);
select /*+ gather_plan_statistics */ empno, ename, sal
      from emp
      where ename='SCOTT';
select * from table(dbms_xplan.display_cursor(null,null,'allstats last') );
```



8	-----							
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10	-----							
11	0	SELECT STATEMENT		1		1	00:00:00.01	2
12	1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	1	1	1	00:00:00.01	2
13	* 2	INDEX RANGE SCAN	EMP_ENAME	1	1	1	00:00:00.01	1
14	-----							

인덱스를 읽을 때 **인덱스를 2개 이상 읽으면 index range scan**이 된다 **1개만 읽을땐 index unique scan** 이다. 이때, SCOTT 은 데이터가 하나인데 SCOTT 하나만 읽지 않고 SMITH 까지 읽은 이유는 non unique index 이기 때문이다.

**12) 사원 테이블에 직업에 인덱스를 걸고 아래의 SQL 을 수행해서 index range scan 으로 실행계획이 나오는지 확인하시오!**

```
create index emp_job on emp(job);
select /*+ gather_plan_statistics */ ename,job
  from emp
  where job='SALESMAN';
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	-----							
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10	-----							
11	0	SELECT STATEMENT		1		4	00:00:00.01	2
12	1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	1	4	4	00:00:00.01	2
13	* 2	INDEX RANGE SCAN	EMP_JOB	1	4	4	00:00:00.01	1
14	-----							

## □2. index unique scan

unique index 로 테이블을 액세스하는 스캔 방법  
unique index 는 중복된 데이터가 없어야만 인덱스가 걸린다  
ex)  
create unique index emp\_empno on emp(empno);

**13) 사원번호가 7788 번인 사원의 사원번호와 사원이름을 출력하는 쿼리문의 실행계획을 확인하시오!**

```
create unique index emp_empno on emp(empno);
select /*+ gather_plan_statistics */ empno, ename
  from emp
  where empno=7788;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	-----							
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10	-----							
11	0	SELECT STATEMENT		1		1	00:00:00.01	2
12	1	TABLE ACCESS BY INDEX ROWID	EMP	1	1	1	00:00:00.01	2
13	* 2	INDEX UNIQUE SCAN	EMP_EMPNO	1	1	1	00:00:00.01	1
14	-----							

```
select empno, rowid
  from emp
  where empno>=0;
```

EMPNO	ROWID
1	7369 AAAS0rAAHAAAN/AAK
2	7499 AAAS0rAAHAAAN/AAF
3	7521 AAAS0rAAHAAAN/AAI
4	7566 AAAS0rAAHAAAN/AAD
5	7654 AAAS0rAAHAAAN/AAE
6	7698 AAAS0rAAHAAAN/AAB
7	7782 AAAS0rAAHAAAN/AAC
8	7788 AAAS0rAAHAAAN/AAL
9	7839 AAAS0rAAHAAAN/AAA
10	7844 AAAS0rAAHAAAN/AAG
11	7876 AAAS0rAAHAAAN/AAM
12	7900 AAAS0rAAHAAAN/AAH
13	7902 AAAS0rAAHAAAN/AAJ
14	7934 AAAS0rAAHAAAN/AAN

unique index 에서 index 에서 딱 한 개의 데이터만 읽고 끝낸다.(중복이없기때문) non unique index 보다 더 좋음  
primary key 제약이나 unique 제약을 컬럼에 걸면 unique index 가 자동으로 생성된다.

#### ※ 지금까지 만든 인덱스 리스트를 확인하기

```
select index_name, uniqueness
from user_indexes
where table_name = 'EMP';
```

**14) dept 테이블에 deptno 에 primar key 제약을 걸고 dept 테이블에 deptno 에 unique 인덱스가 자동으로 생성되었는지 확인하시오!**

```
alter table dept
add constraint dept_deptno_pk primary key(deptno);
select index_name, uniqueness
from user_indexes
where table_name = 'DEPT'; <<< 테이블명은 대문자로 작성해야함
```

INDEX_NAME	UNIQUENESS
1 DEPT_DEPTNO_PK	UNIQUE

unique 로 생성되고, 제약이름으로 index 가 생성된 것을 확인할 수 있다.

**15) dept 테이블의 loc 에 non unique 인덱스를 생성하시오!**

```
create index dept_loc
on dept(loc);
```

**16) 아래의 SQL을 튜닝하시오! (인덱스를 액세스하고 테이블 액세스 하게 하시오)**

```
select *
from dept
where deptno = 20 and loc = 'DALLAS';
      ↑           ↑
    unique index non unique index
```

8	-----							
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10	-----							
11	0	SELECT STATEMENT		1		1	00:00:00.01	2
12	* 1	TABLE ACCESS BY INDEX ROWID	DEPT	1	1	1	00:00:00.01	2
13	* 2	INDEX UNIQUE SCAN	DEPT_DEPTNO_PK	1	1	1	00:00:00.01	1
14	-----							

옵티마이저가 hint 를 지정해주지 않았지만 unique index 를 스스로 pick 한 것을 관찰할 수 있다.

**17) 아래의 SQL 이 loc 에 걸린 non unique index 를 액세스 하도록 힌트를 주시오!**

```
select /*+ gather_plan_statistics index(dept, dept_loc) */ *
```

```
from dept
where deptno = 20 and loc = 'DALLAS';
```

8									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10									
11	0	SELECT STATEMENT		1		1	00:00:00.01	2	
12	* 1	TABLE ACCESS BY INDEX ROWID BATCHED	DEPT	1	1	1	00:00:00.01	2	
13	* 2	INDEX RANGE SCAN	DEPT_LOC	1	2	1	00:00:00.01	1	
14									

붉은 음영부분을 추가해주면(hint) 아래 실행계획과 같이 dept\_loc 를 액세스한 것을 확인할 수 있다.  
옵티마이저가 hint 지시대로 움직였다는 사실을 알 수 있다.

하지만 **non unique index** 보다 **unique index** 가 훨씬 성능이 우수하다.

### □3. index skip scan : 난이도가 있어서 skip scan 공부 이후 진도 예정

### □4. index full scan

index full scan : 인덱스 전체를 full 로 읽어서 원하는 데이터를 액세스 하는 방법

ex) 사원 테이블에 인원수가 전부 몇명인가?

```
create unique index emp_empno on emp(empno);
```

```
select /*+ gather_plan_statistics */ count(*)
```

```
from emp;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

7									
8	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
9									
10	0	SELECT STATEMENT		1		1	00:00:00.01	1	
11	1	SORT AGGREGATE		1	1	1	00:00:00.01	1	
12	2	INDEX FULL SCAN	EMP_EMPNO	1	14	14	00:00:00.01	1	
13									

count(\*) 로 했을 때 table full scan 이 아닌 index full scan을 하는것이 더 효율적이므로 index : emp\_empno를 액세스하여 스캔하는 것을 볼 수 있다. 만약 full scan 했다면 hint 부분에 **index\_fs(emp emp\_empno)** 로 주면 된다.

### 18) 위 SQL이 full table scan 이 되도록 힌트를 주시오!

```
select /*+ gather_plan_statistics full(emp) */ count(*)
```

```
from emp;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

7									
8	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
9									
10	0	SELECT STATEMENT		1		1	00:00:00.01	7	
11	1	SORT AGGREGATE		1	1	1	00:00:00.01	7	
12	2	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	7	
13									

index full scan 에 비해 full table scan 이 buffers 가 훨씬 많은 것을 관찰할 수 있다.

### ※ hint

**index\_fs(table index\_name)** : index full scan

**full(table)** : full table scan

위 힌트들에서 talbe 은 table 명, index\_name 은 index 명을 입력하면 된다

### 19) 우리반 테이블에 ename 에 인덱스를 걸고 우리반 학생들의 인원수를 출력하는 쿼리문의 성능을 높이시오!

```
create unique index emp12_ename on emp12(ename);
```

```
select /*+ gather_plan_statistics index_fs(emp12 emp12_ename) */ count(ename)
from emp12;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8								
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10								
11	0	SELECT STATEMENT		1		1	00:00:00.01	1
12	1	SORT AGGREGATE		1	1	1	00:00:00.01	1
13	2	INDEX FULL SCAN	EMP12_ENAME	1	30	30	00:00:00.01	1
14								

## 참고용

※ index access	hint
1. index range scan	index
2. index unique scan	index
3. index skip scan	index_ss
4. index full scan	index_fs
5. index fast full scan	index_ffs
6. index merge scan	and_equal
7. index bitmap merge scan	index_combine
8. index join	index_join

## □5. index fast full scan

인덱스를 처음부터 끝까지 스캔하는 방법은 index full scan 과 동일하나 index full scan 과 다른점은 인덱스를 full 로 읽을 때 **multi block i/o** 를 한다는 점이다

### multi block i/o

책 앞 목차가 1페이지~100페이지 라고 할때,

**index full scan** 은 한페이지씩 넘기는것이고,

**index fast full scan** 은 한번에 10페이지씩 넘긴다고 이해하면 된다.

ex) emp table >> job index , 직업, 직업별 인원수를 출력하시오!

```
create index emp_job on emp(job);
```

```
select /*+ gather_plan_statistics index_ffs(emp emp_job)*/ job, count(job)
from emp
group by job;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

```
2 | TABLE ACCESS FULL | EMP
```

위와 같이 index\_ffs 힌트를 주어도 table full scan을 한 이유는 job 컬럼에 not null 제약이 없기 때문이다.

index fast full scan 을 하려면 not null 제약이 있어야 한다.

column 에 null 값이 있으면 hint 를 암만 주고 index 를 만들어도 table full scan 해버린다.

### job에 not null 제약 생성하기

```
alter table emp
```

```
modify job constraint emp_job_nn not null;
```

not null 제약을 생성한 후 위 과정을 반복하면 아래와 같이 index fast full scan 한 것을 볼 수 있다.

```
2 | INDEX FAST FULL SCAN | EMP_JOB | 1 | 14 | 14 |
```

16) 부서번호, 부서번호별 인원수를 출력하는데 index fast full scan 이 될 수 있도록 index 도 걸고 not null 제약도 걸

## 고 힌트도 주어서 실행하시오!

```
create index emp_deptno on emp(deptno);
alter table emp
  modify deptno constraint emp_deptno_nn not null;
select /*+ gather_plan_statistics index_ffs(emp emp_deptno)*/ deptno, count(deptno)
  from emp
  group by deptno;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem
0	SELECT STATEMENT		1		3	00:00:00.01	4		
1	HASH GROUP BY		1	14	3	00:00:00.01	4	1558K	1558K
2	INDEX FAST FULL SCAN	EMP_DEPTNO	1	14	14	00:00:00.01	4		

## ※ index full scan과 ~fast 와의 차이점

1. index fast full scan이 더 빠르다
2. index full scan 이 데이터 정렬이 보장되는 반면 index fast full scan 은 데이터의 정렬이 보장되지 않음

```
select /*+ gather_plan_statistics index_ffs(emp emp_deptno) */ deptno, count(deptno)
  from emp
  group by deptno;
```

DEPTNO	COUNT(DEPTNO)
1	30
2	10
3	20

데이터 정렬X

```
select /*+ gather_plan_statistics index_fs(emp emp_deptno) */ deptno, count(deptno)
  from emp
  group by deptno;
```

DEPTNO	COUNT(DEPTNO)
1	10
2	20
3	30

데이터 정렬O

## 17) 우리반 테이블에서 통신사, 통신사별 인원수를 출력하는데 index fast full scan 이 될 수 있도록 하시오!

```
create index emp12_telecom on emp12(telecom);
alter table emp12
  modify telecom constraint emp12_telecom_nn not null;
select /*+ gather_plan_statistics index_ffs(emp12 emp12_telecom) */ telecom, count(telecom)
  from emp12
  group by telecom;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		3	00:00:00.01	4
1	HASH GROUP BY		1	3	3	00:00:00.01	4
2	INDEX FAST FULL SCAN	EMP12_TELECOM	1	30	30	00:00:00.01	4

**index 이름을 create 한것과 제약조건 emp12\_telecom\_nn 을 혼동할 수 있으니 주의하자**

22 ##24)  $2^{100}$  의 각 자리수의 합은?

예시 :  $2^{15}=32768$  . 각 자리수를 다 더하면  $3+2+7+6+8 = 26$  으로 구해진다.

>> 밑과 지수를 선택할 수 있는 SQL 로 구현(x,y값은 한번씩만 나오면 되기때문에 치환변수는 하나씩 써도됨)

accept x prompt '밑을 입력하세요'

accept y prompt '지수를 입력하세요'

```
select sum(level*regexp_count(power(&x, &y),level)) as g
```

```
from dual  
connect by level<=9;
```

# ##1118

2020년 11월 18일 수요일 오전 9:46

SQL 튜닝목차

1. 인덱스 SQL 튜닝(index access)
2. 조인 문장 튜닝
3. 서브쿼리 문장 튜닝
4. 데이터 분석함수를 이용한 튜닝
5. 자동 SQL 튜닝

## ■ 1. index access (어제 계속)

※ index access	hint
1. index range scan	index
2. index unique scan	index
3. index skip scan	index_ss
4. index full scan	index_fs
5. index fast full scan	index_ffs
6. index merge scan	and_equal
7. index bitmap merge scan	index_combine
8. index join	index_join

## □ 3. index skip scan

1. 단일 컬럼 인덱스 : 컬럼을 하나로 해서 만든 인덱스

ex) create index emp\_deptno  
on emp(deptno);

2. 결합 컬럼 인덱스 : 컬럼을 여러개로 해서 만든 인덱스

ex) create index emp\_deptno\_sal  
on emp(deptno, sal);

1) emp\_deptno\_sal 결합 컬럼 인덱스의 구조  
select deptno, sal, rowid  
from emp  
where deptno >=0;

DEPTNO	SAL	ROWID
10	1300	AAASAFAAHAAAAMmAAAN
10	2450	AAASAFAAHAAAAMmAAC
10	5000	AAASAFAAHAAAAMmAAA
20	800	AAASAFAAHAAAAMmAAK
20	1100	AAASAFAAHAAAAMmAAM
20	2975	AAASAFAAHAAAAMmAAD
20	3000	AAASAFAAHAAAAMmAAJ
20	3000	AAASAFAAHAAAAMmAAL
30	950	AAASAFAAHAAAAMmAAH
30	1250	AAASAFAAHAAAAMmAAE
30	1250	AAASAFAAHAAAAMmAAI
30	1500	AAASAFAAHAAAAMmAAG
30	1600	AAASAFAAHAAAAMmAAF
30	2850	AAASAFAAHAAAAMmAAB

deptno 를 먼저 ascending 하게 정렬하고 그것을 기준으로 sal 을 ascending 하게 정렬되게끔 인덱스를 구성하고 있다.

- 2) 결합 컬럼 인덱스가 필요한 이유 :

인덱스에서 데이터를 읽어오면서 테이블 액세스를 줄일 수 있기때문 (인덱스 : 목차 ,테이블 : 책)

내가 검색하고자 하는 데이터가 목차에 있어서 책을 뒤지지 않고 목차에서만 읽고 끝났다면 빠르게 데이터를 검색한 것이다. 책보다는 목차가 두께가 얇으므로 목차에서 데이터를 찾는게 책에서 하는것보다 훨씬 빠르다.

3) 단일컬럼 인덱스를 통해서 테이블 액세스(비교)

```
select /*+ gather_plan_statistics index(emp emp_deptno) */
      deptno, sal
from emp
where deptno=10;

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10									
11	0	SELECT STATEMENT		1		3	00:00:00.01	2	
12	1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	1	3	3	00:00:00.01	2	
13	* 2	INDEX RANGE SCAN	EMP_DEPTNO	1	3	3	00:00:00.01	1	
14									

위 결과를 보면 12행에 emp\_deptno index 만 읽고 그 다음 table access 를 한 것이 보인다

3-2) 결합컬럼 인덱스(emp\_deptno\_sal) 한 결과

8									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10									
11	0	SELECT STATEMENT		1		3	00:00:00.01	1	
12	* 1	INDEX RANGE SCAN	EMP_DEPTNO_SAL	1	3	3	00:00:00.01	1	
13									

위 결과를 보면 table access 하지 않고 range scan 만 한 것과 buffers 가 줄어든 것을 볼 수 있다

현업에서는 주로 단일 컬럼 인덱스보다는 결합 컬럼 인덱스가 많다

결합컬럼은 2개 이상 많이 만들어서 골비처럼 엮을 수 있지만 그 경우 insert 가 느려지는 부작용이 있다.

**23) 아래의 쿼리가 인덱스를 통해서만 데이터를 가져오고 테이블 액세스를 하지 않도록 결합컬럼 인덱스를 생성하시오!**

```
select /*+ gather_plan_statistics index(emp emp_deptno_sal) */ empno, ename, sal,deptno
from emp
where empno=7788;
```

8									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10									
11	0	SELECT STATEMENT		1		1	00:00:00.01	7	
12	* 1	TABLE ACCESS FULL	EMP	1	1	1	00:00:00.01	7	
13									

>>결합컬럼 인덱스 사용

```
create index emp_index1 on emp(empno, ename, sal, deptno);

select /*+ gather_plan_statistics index(emp emp_index1) */ empno, ename, sal,deptno
from emp
where empno=7788;

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10									
11	0	SELECT STATEMENT		1		1	00:00:00.01	1	
12	* 1	INDEX RANGE SCAN	EMP_INDEX1	1	1	1	00:00:00.01	1	
13									

**24) @demp.sql 수행 후 아래의 SQL 을 튜닝하시오! index fast full scan 이 되도록 수행하시오**

```
>>create index, nn constraint, query, display xplan
create index emp_job on emp(job);
alter table emp
```



```

    modify job constraint emp_job_nn not null;
select /*+ gather_plan_statistics index_ffs(emp emp_job) */ job, count(job)
    from emp
    group by job;
select * from table(dbms_xplan.display_cursor(null,null, 'allstats last'));

```

8												
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem	
10												
11	0	SELECT STATEMENT		1		5	00:00:00.01	4				
12	1	HASH GROUP BY		1	14	5	00:00:00.01	4	1345K	1345K	772K (0)	
13	2	INDEX FAST FULL SCAN	EMP_JOB	1	14	14	00:00:00.01	4				
14												

## 25) 다시 @demo. sql 수행 후 아래의 SQL 이 index fast full scan 이 되도록 튜닝하시오!

```

>>before
select deptno, count(empno)
    from emp
    group by deptno;
>>2 tuning
create index emp_index1 on emp(deptno, empno);
alter table emp
    modify deptno constraint emp_deptno_nn not null;
select /*+ gather_plan_statistics index(emp emp_index1) */ deptno, count(empno)
    from emp
    group by deptno;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));

```

8												
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem	
10												
11	0	SELECT STATEMENT		1		3	00:00:00.01	4				
12	1	HASH GROUP BY		1	14	3	00:00:00.01	4	1558K	1558K	768K (0)	
13	2	INDEX FAST FULL SCAN	EMP_INDEX1	1	14	14	00:00:00.01	4				
14												

not null 제약은 두 컬럼 deptno, empno 둘 중 하나만 걸어도 되는데 select 절에서 앞쪽에 있는 deptno 에 걸어주는것이 더 좋다. 두개 다 걸어도 상관없다. 이때, nn 제약을 걸어주는 이유는 column 을 ffs scan 시 null 값을 누락하고 index 하지 않도록 보장해주는 것이다.

## 26) @demo / 아래의 index 를 생성하고 아래의 SQL 을 튜닝하시오! (단 min 과 group by 를 사용X\_)

```

>>before
select deptno, min(sal)
    from emp
    where deptno = 20
    group by deptno;
>>after
create index emp_deptno_sal on emp(deptno, sal);
select /*+ gather_plan_statistics index_asc(emp emp_deptno_sal) */ sal
    from emp
    where deptno = 20
    fetch first 1 rows only;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));

```

8								
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10								
11	0	SELECT STATEMENT		1		1	00:00:00.01	1
12	* 1	VIEW		1	1	1	00:00:00.01	1
13	* 2	WINDOW NOSORT STOPKEY		1	5	1	00:00:00.01	1
14	* 3	INDEX RANGE SCAN	EMP_DEPTNO_SAL	1	5	1	00:00:00.01	1
15								

fetch first 1 rows only 대신 where 절에 where deptno = 20 and rownum = 1 을 사용해도 결과는 비슷함  
그리고 최소값이기 때문에 hint 에 index\_asc 로 index 를 ascending 정렬로 힌트를 주면 더 확실한 결과값을 볼 수 있다

fetch 대신 rownum 을 사용했을때의 실행계획 ↓ ↓ ↓

8								
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10								
11	0	SELECT STATEMENT		1		1	00:00:00.01	1
12	* 1	COUNT STOPKEY		1		1	00:00:00.01	1
13	* 2	INDEX RANGE SCAN	EMP_DEPTNO_SAL	1	5	1	00:00:00.01	1
14								

\*1 을 보면 count stopkey 라고 적힌 부분을 볼 수 있는데 이는 rownum=1 조건에 의해 1개를 찾고 바로 멈춘것을 의미  
sort : 정렬한다는 의미 : index 사용시 이 문구가 없는 것이 성능이 더 우수

## 27) 아래의 SQL 을 튜닝하시오!

>>before

```
select /*+ gather_plan_statistics */ deptno, max(sal)
  from emp
  where deptno = 20
  group by deptno;
```

>>after

```
create index emp_deptno_sal on emp(deptno, sal);
select /*+ gather_plan_statistics index_desc(emp emp_deptno_sal) */ sal
  from emp
  where deptno = 20 and rownum = 1;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

## 28) 아래의 SQL 을 튜닝하시오!

>>before

```
select job, max(hiredate)
  from emp
  where job='SALESMAN'
  group by job;
```

>>after

```
create index emp_job_hiredate on emp(job, hiredate);
select /*+ gather_plan_statistics index_desc(emp emp_job_hiredate) */ hiredate
  from emp
  where job='SALESMAN' and rownum=1;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

rownum은 between 함수 이용해서 더 다양하게 활용 가능하다

----- index skip scan -----

인덱스를 full scan, fast full scan 으로 전체 스캔하는 것이 아니라 중간중간 skip 해서 원하는 데이터를 검색하는 방법

ex) emp\_deptno\_sal index의 구조

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

```
select deptno, sal, rowid
from emp
where deptno >= 0;
```

	DEPTNO	SAL	ROWID
1	10	1300	AAASCxAAHAAA02AAN
2	10	2450	AAASCxAAHAAA02AAC
3	10	5000	AAASCxAAHAAA02AAA
4	20	800	AAASCxAAHAAA02AAK
5	20	1100	AAASCxAAHAAA02AAM
6	20	2975	AAASCxAAHAAA02AAD
7	20	3000	AAASCxAAHAAA02AAJ
8	20	3000	AAASCxAAHAAA02AAL
9	30	950	AAASCxAAHAAA02AAH
10	30	1250	AAASCxAAHAAA02AAE
11	30	1250	AAASCxAAHAAA02AAI
12	30	1500	AAASCxAAHAAA02AAG
13	30	1600	AAASCxAAHAAA02AAF
14	30	2850	AAASCxAAHAAA02AAB

deptno asc, sal asc가 적용된 모습 ↑↑

쿼리문에서 **결합 컬럼 인덱스**를 사용하려면 쿼리문의 **where 절에 결합 컬럼 인덱스의 첫 번째 컬럼이 where 절에 반드시 있어야 한다 (index emp\_empno\_sal)**

- 1) EMP\_DEPTNO\_SAL 결합 컬럼 인덱스의 첫 번째 컬럼인 deptno 가 where 절에 있는 경우  
**index range scan 한다**

```
select /*+ gather_plan_statistics index(emp emp_empno_sal) */ deptno, sal, rowid
from emp
where deptno = 20;
```

3	-----
3	Id   Operation   Name
3	-----
1	0   SELECT STATEMENT
2	* 1   INDEX RANGE SCAN   EMP_EMPNO_SAL
3	-----

- 2)EMP\_DEPTNO\_SAL 결합컬럼 인덱스의 두번째 컬럼인 sal이 where절에 있는 경우  
**index full scan / table full sscan 한다**

```
select /*+ gather_plan_statistics index(emp emp_empno_sal) */ deptno, sal
from emp
where sal = 3000;
```

8	-----
9	Id   Operation   Name
10	-----
11	0   SELECT STATEMENT
12	* 1   INDEX FULL SCAN   EMP_EMPNO_SAL
13	-----

- 3) 위 와 같이 index range scan 으로 출력되지 않고 index full scan 으로 출력되는 상황을 극복하기 위해  
index skip scan 을 사용한다

### ###통계 코드구현(이항분포1)

한 개의 주사위를 360 번 던져서 3의 배수의 눈이 나올 확률을 구하시오!

```
undefine n
```

```
with dice as ( select round( dbms_random.value(0.5,6.5)) as d1
from dual
connect by level <= &n)
```

```
select count(*)/&n as p
```

```
from dice
where mod(d1, 3) = 0;
```

### ※ index skip scan 의 원리

```
select deptno, sal, rowid
from emp
where deptno>=0;
```

	DEPTNO	SAL	ROWID
1	10	1300	AAASCxAAHAAAA02AAN
2	10	2450	AAASCxAAHAAAA02AAC
3	10	5000	AAASCxAAHAAAA02AAA
4	20	800	AAASCxAAHAAAA02AAK
5	20	1100	AAASCxAAHAAAA02AAM
6	20	2975	AAASCxAAHAAAA02AAD
7	20	3000	AAASCxAAHAAAA02AAJ
8	20	3000	AAASCxAAHAAAA02AAL
9	30	950	AAASCxAAHAAAA02AAH
10	30	1250	AAASCxAAHAAAA02AAE
11	30	1250	AAASCxAAHAAAA02AAI
12	30	1500	AAASCxAAHAAAA02AAG
13	30	1600	AAASCxAAHAAAA02AAF
14	30	2850	AAASCxAAHAAAA02AAB

```
select /*+ gather_plan_statistics index(emp emp_empno_sal) */ deptno, sal
from emp
where sal = 2975;
```

deptno 와는 달리 sal 은 emp\_deptno\_sal 인덱스에서 deptno 처럼 정렬되어 있지 않기 때문에 월급이 2975 를 조회하기 위해 인덱스를 처음부터 끝까지 다 full 로 불필요 데이터까지 scan 하는 일이 벌어진다

그렇다면 결합 컬럼 인덱스를 index range scan 할 수 있는 방법은 무엇인가?

1) deptno 를 선두 컬럼이 아닌 sal을 선두 컬럼으로 두고 인덱스를 다시 설정하면 된다

**drop index emp\_deptno\_sal;**

**create index emp\_sal\_deptno on emp(sal, deptno);**

```
select /*+ gather_plan_statistics index(emp emp_deptno_sal) */ deptno, sal
from emp
where sal = 2975;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

deptno 와 sal 의 중요도에 따라 index 를 생성하면 된다. 이 때 둘 다 중요하면 둘 다 만들수 있다. 하지만 이는 index의 과부하를 초래할 수 있고 기존의 index hint 가 엉키는 상황이 올 수 있다.

### 2) index skip☆☆

index skip scan 은 emp\_deptno\_sal 인덱스를 처음부터 끝까지 읽는것은 index full scan 과 같지만 중간중간 검색을 skip 할 수 있다. 만약 10번부터 조회해서 월급이 2975가 있는지 찾아보고있으면 거기까지만 읽고 skip 한 후 다음 deptno column 인 20번을 조회하고 찾으면 skip 하고 넘어간다.

DEPTNO	SAL	ROWID
1	10	1300 AAASCOAAHAAAAAANOAN
2	10	2450 AAASCOAAHAAAAAANOAC
3	10	5000 AAASCOAAHAAAAAANOAA
4	20	800 AAASCOAAHAAAAAANOAAK
5	20	1100 AAASCOAAHAAAAAANOAM
6	20	2975 AAASCOAAHAAAAAANOAD
7	20	3000 AAASCOAAHAAAAAANOAJ
8	20	3000 AAASCOAAHAAAAAANOAL
9	30	950 AAASCOAAHAAAAAANOAH
10	30	1250 AAASCOAAHAAAAAANOAE
11	30	1250 AAASCOAAHAAAAAANOAI
12	30	1500 AAASCOAAHAAAAAANOAG
13	30	1600 AAASCOAAHAAAAAANOAF
14	30	2850 AAASCOAAHAAAAAANOAB

그러면 이제 예시를 보자

```
select /*+ gather_plan_statistics index_ss(emp emp_deptno_sal) */ deptno, sal
  from emp
  where sal = 2975;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		1	00:00:00.01	1
* 1	INDEX SKIP SCAN	EMP_DEPTNO_SAL	1	1	1	00:00:00.01	1

skip hint 는 **index\_ss** 로 작성

## 29) @demo / 아래의 SQL 이 index skip scan 이 되도록 튜닝하시오!

>>before

```
select /*+ gather_plan_statistics */ empno, ename, job, sal
  from emp
  where sal = 1250;
```

>>after

```
create index emp_job_sal on emp(job, sal);
```

```
select /*+ gather_plan_statistics index_ss(emp emp_job_sal) */ empno, ename, job, sal
  from emp
  where sal = 1250;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

Id	Operation	Name
0	SELECT STATEMENT	
1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP
* 2	INDEX SKIP SCAN	EMP_JOB_SAL

**index skip scan** 이 효과를 보려면 결합 컬럼 인덱스의 첫번째 컬럼의 데이터의 종류가 몇 가지 안 되어야 효과가 있다. 위 SQL (29번 위쪽) 예를 들면 deptno 가 10 20 30 만 있을때와 10 20 30 40 50 60 70 80 ... 이 있다고 하면 전자가 skip 비율이 더 높다. 따라서 그 선두컬럼의 data 가 몇 가지 종류로 되어 있는지 query 를 작성하여 확인해볼 필요가 있다. 아래 문제 30번을 참고해 보자

## 30) 직업의 종류가 몇 가지 인지 작성하시오!

```
select count(distinct(job) )
  from emp;
```

## ※ index\_asc 와 index\_desc hint 를 사용한 튜닝

order by 절을 남발하면 성능이 떨어지며 database 의 규모가 커질수록 그 하락폭이 커진다

↓↓↓↓↓

### 31) @demo / 아래의 index 를 생성한 후 아래의 SQL 을 튜닝하시오! (단 order by 절을 사용하지 마시오!)

```
create index emp_sal on emp(sal);
```

>>before

```
select /*+ gather_plan_statistics */ ename, sal
```

```
from emp
```

```
order by sal desc;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10									
11	0	SELECT STATEMENT		1		14	00:00:00.01	7	
12	1	SORT ORDER BY		1	14	14	00:00:00.01	7	
13	2	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	7	
14									

>>after

```
select /*+ gather_plan_statistics index_desc(emp emp_sal) */ ename, sal
```

```
from emp
```

```
where sal >= 0;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10									
11	0	SELECT STATEMENT		1		14	00:00:00.01	2	
12	1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	1	14	14	00:00:00.01	2	
13	* 2	INDEX RANGE SCAN DESCENDING	EMP_SAL	1	14	14	00:00:00.01	1	
14									

index\_desc hint 가 제대로 동작하려면 where 절에 index 컬럼이 검색조건으로 있어야 한다

### 32) 아래의 SQL 을 튜닝하시오! (실행계획에 sort 가 있지 않게 하시오)

>>before

```
select /*+ gather_plan_statistics */ max(sal)
```

```
from emp;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

7									
8	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
9									
10	0	SELECT STATEMENT		1		1	00:00:00.01	1	
11	1	SORT AGGREGATE		1	1	1	00:00:00.01	1	
12	2	INDEX FULL SCAN (MIN/MAX)	EMP_SAL	1	1	1	00:00:00.01	1	
13									

>>after (no group funtion)

```
select /*+ gather_plan_statistics index_desc(emp emp_sal) */ sal
```

```
from emp
```

```
where sal >= 0 and rownum =1 ;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10									
11	0	SELECT STATEMENT		1		1	00:00:00.01	1	
12	* 1	COUNT STOPKEY		1		1	00:00:00.01	1	
13	* 2	INDEX RANGE SCAN DESCENDING	EMP_SAL	1	14	1	00:00:00.01	1	
14									

**33) 아래의 SQL 을 튜닝하시오! (emp 테이블을 한번만 액세스해서 결과가 나오게 하시오!)**

```
>>before
select /*+ gather_plan_statistics */ ename, sal
  from emp
 where sal = ( select max(sal)
               from emp);

>>after
select /*+ gather_plan_statistics index_desc(emp emp_sal) */ ename, sal
  from emp
 where sal >=0 and rownum=1;
```

**34) @demo / 아래의 SQL 을 튜닝하시오! (order by 절 안쓰고 출력!)**

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

>>before
select empno, deptno, sal
  from emp
 where deptno = 20
 order by sal desc;

>>after
select /*+ gather_plan_statistics index_desc(emp emp_deptno_sal) */ empno, deptno, sal
  from emp
 where deptno = 20;
```

**35) @demo / 아래의 SQL 을 튜닝하시오!**

```
>>before
select /*+ gather_plan_statistics */ ename, hiredate
  from emp
 where to_char(hiredate, 'RRRR') = '1981'
 order by hiredate desc;

>>after
create index emp_hiredate on emp(hiredate);
select /*+ gather_plan_statistics index_desc(emp emp_hiredate) */ ename, hiredate
  from emp
 where hiredate between to_date('1981/01/01', 'RRRR/MM/DD') and to_date('1981/12/31', 'RRRR/MM/DD');
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

10	-----		
11	Id	Operation	Name
12	-----		
13	0	SELECT STATEMENT	
14	* 1	FILTER	
15	2	TABLE ACCESS BY INDEX ROWID BATCHED	EMP
16	* 3	INDEX RANGE SCAN DESCENDING	EMP_HIREDATE
17	-----		

**36) @demo / 아래의 SQL을 튜닝하시오!**

```
>>before
select /*+ gather_plan_statistics */ ename, job, sal
  from emp
 where substr(job,1,5) = 'SALES'
 order by sal desc;

>>after
```

```
create index emp_job_sal on emp(job, sal);
select /*+ gather_plan_statistics index_desc(emp emp_job_sal) */ ename, job, sal
  from emp
  where job like 'SALES%';
```

## □6. index merge scan

※ index access	hint
6. index merge scan	and_equal
7. index bitmap merge scan	index_combine
8. index join	index_join

index merge scan : 두 개의 인덱스를 동시에 사용하여 하나의 인덱스만 사용했을 때보다 더 좋은 성능을 보이는 스캔

**ex) @demo / 아래 SQL의 실행계획을 확인하시오!**

```
create index emp_deptno on emp(deptno);
create index emp_job on emp(job);
select /*+ gather_plan_statistics */ ename, deptno, job
  from emp
  where job = 'SALESMAN' and deptno = 30;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	-----			
9	Id	Operation	Name	
10	-----			
11	0	SELECT STATEMENT		
12	* 1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	
13	* 2	INDEX RANGE SCAN	EMP_JOB	
14	-----			
..				

옵티마이저가 스스로 emp\_job 으로 index를 탄 모습이다

**37) 위 SQL 이 deptno 에 걸린 인덱스를 타도록 힌트를 주고 실행하시오!**

```
select /*+ gather_plan_statistics index(emp emp_deptno) */ ename, deptno, job
  from emp
  where job = 'SALESMAN' and deptno = 30;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	-----			
9	Id	Operation	Name	
10	-----			
11	0	SELECT STATEMENT		
12	* 1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	
13	* 2	INDEX RANGE SCAN	EMP_DEPTNO	
14	-----			

**38) 위 SQL을 job 에 걸린 인덱스를 타도록 힌트를 주고 실행하시오!**

```
select /*+ gather_plan_statistics index(emp emp_job) */ ename, deptno, job
  from emp
  where job = 'SALESMAN' and deptno = 30;
```

job 과 deptno 단일 컬럼 인덱스를 각각 건 상태에서 각각 두 개의 index를 사용해서 데이터를 검색하였다  
그러면 2개 중 어떤 컬럼의 인덱스를 사용하는 것이 더 성능이 좋을까?

목차를 길게 읽는것보다 목차를 짧게 읽는게 더 좋은 목차(index) 이다

아래 두 SQL 을 보자

```
select count(*) from emp where job = 'SALESMAN'; >>> 4개
```



```
select count(*) from emp where deptno = 30; >>> 6개
```

위 SQL 을 보면 출력값이 4개로 나온 것이 더 좋은 성능(적은 검색량)을 보여줄 수 있기 때문에 emp\_job index 를 쓰는 것이 좋은 판단이다.

그런데 이때, 2개의 index 를 모두 사용해서 더 큰 시너지를 낼 수 있게 해주는 것이 **index merge scan** 이다

```
select /*+ gather_plan_statistics and_equal(emp emp_job emp_deptno) */ ename, deptno, job
  from emp
  where job = 'SALESMAN' and deptno = 30;

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		4	00:00:00.01	5
* 1	TABLE ACCESS BY INDEX ROWID	EMP	1	4	4	00:00:00.01	5
2	AND-EQUAL		1		4	00:00:00.01	4
* 3	INDEX RANGE SCAN	EMP_JOB	1	4	4	00:00:00.01	3
* 4	INDEX RANGE SCAN	EMP_DEPTNO	1	6	4	00:00:00.01	1

그런데 위 스캔 방식은 옛날 방법이다. 이보다 더 좋은 스캔방법이 나왔다. 그것이 index bitmap merge scan 이다

## □7. index bitmap merge scan ~ index\_combine

두 개의 인덱스를 같이 스캔해서 시너지 효과를 보는 방법은 index merge scan 과 동일하다. 하지만 bitmap 을 이용해서 인덱스의 사이즈를 아주 많이 줄인다는게 둘의 차이점이며 해당 방법의 스캔시간은 짧다

ex) 책의 내용을 검색하기 위해서 두 개의 목차를 먼저 읽고 요약해서 목차를 A4 한페이지로 만들어 버리면 목차를 빠르게 검색할 수 있기에 테이블의 데이터를 찾기가 쉬워진다(짧아진다)

hint : index\_combine

```
select /*+ gather_plan_statistics index_combine(emp) */ ename,deptno, job
  from emp
  where job= 'SALESMAN' and deptno =30;

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

Id	Operation	Name	Starts	E-Rows	A-Rows
0	SELECT STATEMENT		1		4
1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	1	4	4
2	BITMAP CONVERSION TO ROWIDS		1		4
3	BITMAP AND		1		1
4	BITMAP CONVERSION FROM ROWIDS		1		1
* 5	INDEX RANGE SCAN	EMP_DEPTNO	1		6
6	BITMAP CONVERSION FROM ROWIDS		1		1
* 7	INDEX RANGE SCAN	EMP_JOB	1		4

## □8. index join ~ /\*+ index\_join(emp emp\_

```
create index emp_deptno on emp(deptno);
```

```
create index emp_job on emp(job);
```

```
select /*+ gather_plan_statistics index_join(emp emp_deptno emp_job) */ deptno, job
  from emp
  where job='SALESMAN' and deptno =30;

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

☆##통계 코드구현(이항분포1-2)

**평균 : select avg(sal) from emp;**

**분산 : select variance(sal) from emp; (데이터의 퍼짐정도)**

**루트 : select sqrt(4) from dual;**

**39) 이항 분포 공식을 이용해서 다음 이항 분포의 평균값과 분산값과 표준편차를 구하시오!**

한개의 주사위를 36000번 던져서 3의 배수의 눈이 나올 횟수를 확률변수 x라고 하자. 이때 확률변수 x의 평균값과 분산값과 표준편차를 구하시오! (손으로)

>>평균

$36000 \times \frac{1}{3} = 12000$

>>분산

$12000 \times (1 - \frac{1}{3}) = 8000$

>>표준편차

40 route 5

한개의 주사위를 36000번 던져서 3의 배수의 눈이 나오는 횟수를 X 라고 했을때 3의 배수의 눈이 나오는 횟수의 평균값은 무엇이고 분산값은 무엇이고 표준편차는 무엇인가 ? ( 직접 주사위를 360번 SQL 로 던져서 알아내세요)

>>수학공식이용

undefine n

with dice as ( select round( dbms\_random.value(0.5,6.5)) as d1

from dual

connect by level <= &n)

select count(\*) as avg, round ( count(\*) \* (1-count(\*)/&n), 3) as var,

round( sqrt( count(\*) \* (1-count(\*)/&n) ), 3) as standard\_var

from dice

where mod(d1, 3) = 0;

# ##1119

2020년 11월 19일 목요일    오후 12:10

SQL 튜닝목차

1. 인덱스 SQL 튜닝(index access)
2. 조인 문장 튜닝
3. 서브쿼리 문장 튜닝
4. 데이터 분석함수를 이용한 튜닝
5. 자동 SQL 튜닝

## ■2. 조인 문장 튜닝

1. nested loop 조인
2. hash 조인
3. sort merge 조인
4. 조인 순서의 중요성
5. outer 조인
6. 스칼라 서브쿼리를 이용한 조인
7. 조인을 내포한 DML 문 튜닝
8. 고급 조인 문장 튜닝

### □1. nested loop join

SQL 시간에 배운 조인 문법

1. 1999 ANSI 조인 문법
2. 오라클 조인 문법
  - equi join
  - non equi join
  - outer join
  - self join

SQL 튜닝시간에 배울 조인 방법 : 일종의 실행 계획(더 빠른 조인을 위해)

1. nested loop join
2. hash join
3. sort merge join

**40) 사원 이름, 월급, 부서위치를 출력하는데 조인 순서가 어떤 것이 더 성능이 좋겠는가?**

1. emp → dept
2. dept → emp

emp table >> 14개    dept >> 4개

작은 테이블을 먼저 읽고 큰 테이블이랑 조인하는것이 성능이 더 좋으므로 dept table 을 먼저 읽는 것이 더 좋다

**41) 위 조인문장의 실행계획을 확인해서 어느 테이블을 먼저 읽고 조인했는지 확인하시오!**

```
select /*+ gather_plan_statistics */ e.ename, e.sal, d.loc
  from emp e, dept d
 where e.deptno=d.deptno;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	-----								
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10	-----								
11	0	SELECT STATEMENT		1		14	00:00:00.01	15	
12	* 1	HASH JOIN		1	14	14	00:00:00.01	15	
13	2	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	7	
14	3	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	7	
15	-----								

실행계획에서 table access 순서는 hash join 밑의 dept - emp 순으로 조인하였다(위에서 아래순으로 조인)

※ 조인튜닝시 중요한 튜닝방법 2가지

1. 조인 순서를 교정 : hint : **orderd** **힌트**는 from 절에서 기술한 테이블 순서대로 조인해라  
**leading** 힌트는 힌트 안에 사용한 테이블 순서대로 조인하시오
2. 조인 방법을 조정

**42) 문제 41번의 조인문장의 조인순서를 emp >>> dept 순으로 조인되게 조인순서를 조정하시오!**

```
select /*+ gather_plan_statistics ordered */ e.ename, e.sal, d.loc
  from emp e, dept d
 where e.deptno=d.deptno;

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	-----								
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10	-----								
11	0	SELECT STATEMENT		1		14	00:00:00.01	15	
12	* 1	HASH JOIN		1	14	14	00:00:00.01	15	
13	2	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	7	
14	3	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	7	
15	-----								

실행결과를 보면 ordered hint 에 의해 emp - dept 테이블 순으로 조인한 것을 알 수 있다

즉, 조인순서가 41번에 비해 달라진 것을 알 수 있다. 버퍼의 개수는 비효율적인 조인을 할수록 증가한다. 단 현재의 예시들에서는 emp 와 dept 테이블의 규모가 크지 않아서 변화량을 보이지 않았다

**43) 다시 위의 SQL 조인 순서를 dept >> emp 순이 되도록 hint 를 주고 SQL 을 작성하시오!**

```
select /*+ gather_plan_statistics ordered */ e.ename, e.sal, d.loc
  from dept d, emp e
 where e.deptno=d.deptno;

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

hint를 주고 from 절의 테이블 순서를 조정해주어 조인 순서를 fix 하였다

※ 튜닝방법tip : 기존 튜닝 전 SQL 에 힌트가 있으면 힌트를 먼저 빼고 수행해본다 (optimizer에게 기대)

**44) emp 와 salgrade 테이블을 조인해서 이름과 월급과 급여등급(grade)을 출력하시오! (성능이 좋게 SQL 입력하시오)**

```
select /*+ gather_plan_statistics orderd */ e.ename, e.sal, s.grade
  from salgrade s, emp e
 where e.sal between s.losal and s.hisal;

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	-----								
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10	-----								
11	0	SELECT STATEMENT		1		14	00:00:00.01		13
12	1	MERGE JOIN		1	1	14	00:00:00.01		13
13	2	SORT JOIN		1	5	5	00:00:00.01		6
14	3	TABLE ACCESS FULL	SALGRADE	1	5	5	00:00:00.01		6
15	* 4	FILTER		5		14	00:00:00.01		7
16	* 5	SORT JOIN		5	14	40	00:00:00.01		7
17	6	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01		7
18	-----								

>>2 leading hint를 사용하시오 (emp 테이블부터 조인하도록 조정하시오)

```
select /*+ gather_plan_statistics leading(e s) */ e.ename, e.sal, s.grade
      from salgrade s, emp e
      where e.sal between s.losal and s.hisal;
```

leading 힌트의 형태는 leading(컬럼별칭, 컬럼별칭) 이다  
ordered 보다 leading 이 더 사용하기 편리하다

**45) emp 와 dept 테이블을 조인해서 이름과 월급과 직업과 부서위치를 출력하시오! 단, leading hint 를 이용해서 조인순서를 정하시오!**

```
select /*+ gather_plan_statistics leading(e d) */ e.ename, e.sal, e.job, d.loc
      from emp e ,dept d
      where e.deptno=d.deptno;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

### ※ 조인튜닝시 중요튜닝

1. 조인순서 : hint : **ordered, leading**
2. 조인방법 : leading 힌트 안에 쓴 테이블(별칭) 순서대로 조인해라
  - nested loop join : 적은량의 데이터를 조인할 때 유리  
**hint : use\_nl**
  - hash join : 대용량 데이터 조인시 유리  
**hint : use\_hash**
  - sort merge join : 대용량 데이터 조인시 유리 (hash join으로 수행되지 못하는 SQL에 사용)  
**hint : use\_merge**

작은 테이블을 조인시 hash join 을 하면 쓸데없이 메모리를 과다하게 사용하게 된다 ( hash / merge 사용)
3. 위 두 방법은 조인순서>>조인방법 순으로 힌트에 기술하여 작성하여 준다  
ex) /\*+ gather\_plan\_statistics leading(e d) use\_nl(e d) \*/

**46) 문제 45번의 쿼리의 실행계획이 nested loop 조인이 되게 힌트를 주시오!**

```
select /*+ gather_plan_statistics leading(e d) use_nl(e d) */ e.ename, e.sal, e.job, d.loc
      from emp e ,dept d
      where e.deptno=d.deptno;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	-----								
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10	-----								
11	0	SELECT STATEMENT		1		14	00:00:00.01		105
12	1	NESTED LOOPS		1	14	14	00:00:00.01		105
13	2	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01		7
14	* 3	TABLE ACCESS FULL	DEPT	14	1	14	00:00:00.01		98
15	-----								

해쉬조인 때와는 달리 버퍼의 개수가 늘어나는 것을 알 수 있다. 하지만 emp와 dept 테이블이 서로 조인되는 데이터의 양이 적으므로 nested loop 조인을 사용하는것이 더 바람직함

47) 46번 쿼리의 조인 문장의 실행계획이 nested loop 조인이 되도록 하되 조인순서는 dept >> emp 순으로 하시오!

```
select /*+ gather_plan_statistics leading(d e) use_nl(d e) */ e.ename, e.sal, e.job, d.loc
  from emp e ,dept d
  where e.deptno=d.deptno;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10								
11	0	SELECT STATEMENT		1		14	00:00:00.01	35
12	1	NESTED LOOPS		1	14	14	00:00:00.01	35
13	2	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	7
14	* 3	TABLE ACCESS FULL	EMP	4	4	14	00:00:00.01	28
15								

조인순서를 변경하고 버퍼가 줄어든 것을 관찰할 수 있다

48) emp 와 salgrade 테이블을 조인해서 이름과 월급과 급여등급( grade) 을 출력하는 조인문장의 조인 순서와 조인 방법을 아래와 같이 되게 하시오!

조인순서 : salgrade >> emp

조인방법 : nested loop join

```
select /*+ gather_plan_statistics leading(s e) use_nl(s e) */ e.ename, e.sal, s.grade
  from emp e, salgrade s
  where e.sal between s.losal and s.hisal;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10								
11								
12	0	SELECT STATEMENT		1		14	00:00:00.01	41
13	1	NESTED LOOPS		1	1	14	00:00:00.01	41
14	2	TABLE ACCESS FULL	SALGRADE	1	5	5	00:00:00.01	6
15	* 3	TABLE ACCESS FULL	EMP	5	1	14	00:00:00.01	35
16								

49) 이름이 SCOTT 인 사원의 이름과 월급과 부서위치를 출력하는 조인 문장을 작성하시오!

```
select /*+ gather_plan_statistics */ e.ename, e.sal, d.loc
  from emp e, dept d
  where e.deptno=d.deptno and e.ename='SCOTT';
```

50) 위 SQL 은 조인순서가 아래의 2개중 어떤것이 더 좋은 순서인가??

1. emp >> dept

2. dept >> emp

위 두 가지 경우 중 where 조건에 e.ename='SCOTT' 으로 인해 emp 가 1건이 되기 때문에 emp >> dept 순서가 더 좋다

emp 테이블에서 ename 이 SCOTT 인 사원을 읽고 이 사원의 loc 만 알면 되기 때문에 조인횟수가 1회이다.

※ 조인횟수

```
select /*+ gather_plan_statistics */ e.ename, e.sal, d.loc
  from emp e, dept d
  where e.deptno=d.deptno;
```

위 49번 SQL에서 emp>>dept 순이라면 14번의 조인, dept>>emp 순서이면 4번의 조인을 시도한다

emp	dept
-----	------

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DEPTNO	DNAME	LOC
1	7839 KING	PRESIDENT	(null)	81/11/17	5000	(null)	10	1	10 ACCOUNTING	NEW YORK
2	7698 BLAKE	MANAGER	7839 81/05/01	2850	(null)		30	2	20 RESEARCH	DALLAS
3	7782 CLARK	MANAGER	7839 81/05/09	2450	(null)		10	3	30 SALES	CHICAGO
4	7566 JONES	MANAGER	7839 81/04/01	2975	(null)		20	4	40 OPERATIONS	BOSTON
5	7654 MARTIN	SALESMAN	7698 81/09/10	1250	1400		30			
6	7499 ALLEN	SALESMAN	7698 81/02/11	1600	300		30			
7	7844 TURNER	SALESMAN	7698 81/08/21	1500	0		30			
8	7900 JAMES	CLERK	7698 81/12/11	950	(null)		30			
9	7521 WARD	SALESMAN	7698 81/02/23	1250	500		30			
10	7902 FORD	ANALYST	7566 81/12/11	3000	(null)		20			
11	7369 SMITH	CLERK	7902 80/12/09	800	(null)		20			
12	7788 SCOTT	ANALYST	7566 82/12/22	3000	(null)		20			
13	7876 ADAMS	CLERK	7788 83/01/15	1100	(null)		20			
14	7934 MILLER	CLERK	7782 82/01/11	1300	(null)		10			

51) 아래의 조인문장에 조인순서는 emp>>dept 로 조인하고 조인방법은 nested loop 조인이 되도록 힌트를 주고 튜닝하시오!

>>before

```
select /*+ gather_plan_statistics */ e.ename, e.sal, d.loc
  from emp e, dept d
  where e.deptno=d.deptno and e.ename='SCOTT';
```

>>after

```
select /*+ gather_plan_statistics leading(e d) use_nl(e d) */ e.ename, e.sal, d.loc
  from emp e, dept d
  where e.deptno=d.deptno and e.ename='SCOTT';
```

9	-----								
10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
11	-----								
12	0	SELECT STATEMENT		1		1	00:00:00.01		14
13	1	NESTED LOOPS		1	1	1	00:00:00.01		14
14	* 2	TABLE ACCESS FULL	EMP	1	1	1	00:00:00.01		7
15	* 3	TABLE ACCESS FULL	DEPT	1	1	1	00:00:00.01		7
16	-----								

※ 조인문장에 조인(연결) 조건 말고 검색조건이 따로 있으면 검색조건으로 검색되는 건수를 확인하고 그 건수가 조인 대상이 되는 상대 테이블보다 작다면 작은 건수를 검색하는 테이블을 먼저 읽어야 성능향상에 도움이 된다

52) 아래의 SQL의 조인 방법은 무조건 nested loop 조인으로 하되 순서는 직접 정하시오!

```
select /*+ gather_plan_statistics */ e.ename, e.sal, d.loc
  from emp e, dept d
  where e.deptno = d.deptno and e.job='SALESMAN' and d.loc='CHICAGO';
>>
select /*+ gather_plan_statistics leading(d e) use_nl(d 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 * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

9								
10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
11								
12	0	SELECT STATEMENT		1		4	00:00:00.01	14
13	1	NESTED LOOPS		1	1	4	00:00:00.01	14
14	* 2	TABLE ACCESS FULL	DEPT	1	1	1	00:00:00.01	7
15	* 3	TABLE ACCESS FULL	EMP	1	1	4	00:00:00.01	7
16								

위 SQL에서 -- 4건 이 부분은 주석으로 SQL 실행시 영향을 주지 않는다

☆53) 아래의 SQL이 적절한 실행계획이 나올 수 있도록 튜닝하시오!

>>before

```
select e.ename, e.sal, s.grade
  from emp e, salgrade s
 where e.sal between s.losal and s.hisal
        and s.grade=1;
```

>>after

```
select /*+ gather_plan_statistics leading(s e) use_nl(s e) */ e.ename, e.sal, s.grade
  from emp e, salgrade s
 where e.sal between s.losal and s.hisal
        and s.grade=1; -- 1건
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		3	00:00:00.01	13
1	NESTED LOOPS		1	1	3	00:00:00.01	13
* 2	TABLE ACCESS FULL	SALGRADE	1	1	1	00:00:00.01	6
* 3	TABLE ACCESS FULL	EMP	1	1	3	00:00:00.01	7

한편, 아래는 emp>>salgrade 조인 시 실행계획이다

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		3	00:00:00.01	13
1	MERGE JOIN		1	1	3	00:00:00.01	13
2	SORT JOIN		1	14	14	00:00:00.01	7
3	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	7
* 4	FILTER		14		3	00:00:00.01	6
* 5	SORT JOIN		14	1	14	00:00:00.01	6
* 6	TABLE ACCESS FULL	SALGRADE	1	1	1	00:00:00.01	6

버퍼의 개수가 salgrade >> emp 순서 조인이 더 적다

※ 3개 이상 조인

leading 힌트에 사용했던 테이블 별칭 중 두 번째 별칭을 use\_nl 에 기술하면 된다

```
select /*+ gather_plan_statistics leading(s e) use_nl(e) */ e.ename, e.sal, s.grade
  from emp e, salgrade s
 where e.sal between s.losal and s.hisal
        and s.grade=1; -- 1건
```

**leading(s e) use\_nl(e) 의 뜻은 salgrade 와 emp 순으로 조인하는데 emp 와 조인할 때 nested loop 조인하시오**

54) emp, dept, salgrade 를 조인해서 이름, 월급, 부서위치, 급여등급을 출력하시오!

```
select /*+ gather_plan_statistics */ 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;
```

55) 위 SQL 의 조인 순서와 방법을 아래와 같이 되도록 하시오!

**dept --> emp --> salgrade <----- 연결고리가 있는 table 끼리 조인되도록 고려한 테이블 조인 순서**

만약, dept --> salgrade -->emp 순서로 한다면 dept 와 salgrade 의 조인조건을 줄 수 없다

```
select /*+ gather_plan_statistics leading(d e s) 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;  
 select \* from table(dbms\_xplan.display\_cursor(null,null,'allstats last'));

9									
10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
11									
12	0	SELECT STATEMENT		1		14	00:00:00.01	119	
13	1	NESTED LOOPS		1	1	14	00:00:00.01	119	
14	2	NESTED LOOPS		1	14	14	00:00:00.01	35	
15	3	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	7	
16	* 4	TABLE ACCESS FULL	EMP	4	4	14	00:00:00.01	28	
17	* 5	TABLE ACCESS FULL	SALGRADE	14	1	14	00:00:00.01	84	
18									

위 SQL 중 hint leading(d e s) use\_nl(e) use\_nl(s) 이 부분의 의미는 아래와 같다

**"dept --> emp --> salgrade 순으로 조인하면서 dept 와 emp 가 조인할 때 nested loop 방법으로하고 dept와 emp 가 조인한 결과와 salgrade 와 조인할 때 nested loop 조인하시오"**

## 56) 아래와 같이 조인되도록 하시오!

salgrade --> emp --> dept

↑        ↑  
 nested loop    nested loop

select /\*+ gather\_plan\_statistics 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;

55번 문제에서 힌트만 수정해주면 join 순서가 달라진 것을 알 수 있다

## 57) 56번의 실행계획을 분석하시오!

9									
10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
11									
12	0	6 SELECT STATEMENT		1		14	00:00:00.01	139	
13	1	5 NESTED LOOPS		1	1	14	00:00:00.01	139	
14	2	3 NESTED LOOPS		1	1	14	00:00:00.01	41	
15	3	1 TABLE ACCESS FULL	SALGRADE	1	5	5	00:00:00.01	6	
16	* 4	2 TABLE ACCESS FULL	EMP	5	1	14	00:00:00.01	35	
17	* 5	4 TABLE ACCESS FULL	DEPT	14	1	14	00:00:00.01	98	
18									

위 실행계획의 붉은 숫자는 SQL 이 실행되는 순서이다. 우선 15,16행이 같은 라인에 있으므로 그 부분은 위에서 아래의 순서로 실행된다. 그 부분은 salgrade 와 emp 가 서로 nested loop 조인하는 부분이다. 그리고 14행과 17행이 같은 라인에 있으므로 마찬가지로 위에서 아래로 실행된다. 그 의미는 salgrade 와 emp 가 서로 nested loop 조인한 결과와 dept 테이블과 조인하는데 그 방법은 13행에 적힌 nested loop 의 방법으로 실행되며 끝으로 select 하는 부분까지 관찰할 수 있다. 이 부분에 대해서는 스스로 분석할 수 있어야 tuning 에서의 유의미한 발전을 이룰 수 있다

## 58) 이름이 KING 인 사원의 이름, 월급, 부서위치, 급여등급을 출력하시오!

select /\*+ gather\_plan\_statistics \*/ 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 -- 조인조건

and s.hisal and ename='KING'; -- 검색조건 (1건)

## 59) 58번의 SQL 의 조인순서를 아래와 같이 하고 조인방법은 전부 nested loop 조인으로 하시오!

조인순서 : emp --> dept --> salgrade

emp 테이블에서 ename 이 KING 인 데이터가 1건이므로 1건만 읽어서 dept 테이블과 조인하여 부서위치를 가져오고 마지막으로 emp와 dept 테이블과 조인한 결과가 salgrade 테이블과 조인한다

select /\*+ gather\_plan\_statistics leading(e d s) use\_nl(d) 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 and ename='KING';
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));

```

11	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
13	0	SELECT STATEMENT		1		1	00:00:00.01	20
14	1	NESTED LOOPS		1	1	1	00:00:00.01	20
15	2	NESTED LOOPS		1	1	1	00:00:00.01	14
16	* 3	TABLE ACCESS FULL	EMP	1	1	1	00:00:00.01	7
17	* 4	TABLE ACCESS FULL	DEPT	1	1	1	00:00:00.01	7
18	* 5	TABLE ACCESS FULL	SALGRADE	1	1	1	00:00:00.01	6

## 60) CHICAGO 에서 근무하는 직원들의 이름, 부서위치, 월급, 급여등급을 출력하시오!

```

select /*+ gather_plan_statistics */ e.ename, d.loc, e.sal, s.grade
from emp e, dept d, salgrade s
where e.deptno=d.deptno and e.sal between s.losal and s.hisal and d.loc='CHICAGO';

```

## 61) 60번의 SQL 의 조인 방법은 전부 nested loop 조인으로 하되 조인순서를 지정하여 튜닝하시오!

```

select /*+ gather_plan_statistics leading(d e s) use_nl(e) use_nl(s)*/ e.ename, d.loc, e.sal, s.grade
from emp e, dept d, salgrade s
where e.deptno=d.deptno and e.sal between s.losal and s.hisal and d.loc='CHICAGO';
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));

```

11	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
13	0	SELECT STATEMENT		1		6	00:00:00.01	50
14	1	NESTED LOOPS		1	1	6	00:00:00.01	50
15	2	NESTED LOOPS		1	5	6	00:00:00.01	14
16	* 3	TABLE ACCESS FULL	DEPT	1	1	1	00:00:00.01	7
17	* 4	TABLE ACCESS FULL	EMP	1	5	6	00:00:00.01	7
18	* 5	TABLE ACCESS FULL	SALGRADE	6	1	6	00:00:00.01	36

## 62) 부서위치, 부서위치별 토탈월급을 출력하시오! (단, 세로로 출력)

```

select /*+ gather_plan_statistics */ d.loc, sum(e.sal)
from emp e, dept d
where e.deptno=d.deptno
group by d.loc;

```

## 63) 62번의 조인순서와 조인방법을 기술하고 수행하시오!

```

select /*+ gather_plan_statistics 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;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));

```

## □2. hash loop join

nested loop 조인 : 순차적 반복 조인으로 한 레코드씩 순차적으로 조인	
--	--

```
select e.ename, d.loc
from emp e, dept d
where e.deptno=d.deptno and e.job='SALESMAN';
```

	ENAME	LOC
1	MARTIN	CHI CAGO
2	ALLEN	CHI CAGO
3	TURNER	CHI CAGO
4	WARD	CHI CAGO

위와 같이 하나씩, MARTIN, ALLEN, TU..순으로 조인하면  
대량의 data의 경우 검색속도가 매우 느려진다

대량 data 시  
비효율적

### hash join : 해쉬 알고리즘을 이용해서 데이터를 메모리에 올려놓고 메모리에서 조인하는 방식

메모리에서 다 조인하면 좋지만 디스크에 비해 매우 크기가 작기 때문에 그것까지는 불가능하다

메모리는 크기가 작으므로 서로 메모리에 데이터를 올리고 조인하려고 하면 경합이 걸려 다같이 속도가 느려진다

대용량 데이터를 조인할 때 nested loop 조인을 하면 검색속도가 매우 느리므로 hash loop 를 사용함

### ex)hash loop join

```
select /*+ gather_plan_statistics leading(d e) use_hash(e) */ e.ename, d.loc
from emp e, dept d
where e.deptno = d.deptno;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

hash loop join									nested loop join	
8										
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	Buffers	
10										
11	0	SELECT STATEMENT		1		14	00:00:00.01	15	35	
12	* 1	HASH JOIN		1	14	14	00:00:00.01	15	35	
13	2	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	7	7	
14	3	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	7	28	
15										

nested loop join 보다 버퍼의 개수가 월등히 적은 것을 관찰할 수 있다

dept 테이블과 emp 테이블이 둘 다 메모리로 올라갈 때 메모리에 둘다 한면에 다 올라가서 조인하면 좋지만 이것은 불가능하고 둘 중에 하나만 먼저 메모리로 올리고 다른 테이블은 데이터의 일부를 조금씩 메모리를 올리면서 조인함

ROWID	LOC	DNAME	DEPTNO
1 AAASSdAAHAAAAM8AAA	NEW YORK	ACCOUNTING	10
2 AAASSdAAHAAAAM8AAB	DALLAS	RESEARCH	20
3 AAASSdAAHAAAAM8AAC	CHI CAGO	SALES	30
4 AAASSdAAHAAAAM8AAD	BOSTON	OPERATIONS	40

rowid : 행의 물리적 주소 (어느 파일에 어느 블록에 있다 라는 물리적 주소 정보)

dept 테이블이 디스크에 있을때는 이 정보가 유용

dept 테이블이 통채로 메모리로 올라가면 rowid주소가 필요없어지고 새로운 주소가 필요한데 그것이 메모리 주소이다  
아래 설명을 참고 ↓ ↓

### HASH JOIN

TABLE ACCESS FULL DEPT ---> hash table (메모리로 올라간 테이블)

TABLE ACCESS FULL EMP ---> probe table (디스크에 있는 테이블)

메모리가 공간이 작기 때문에 emp 와 dept 테이블 중 작은 테이블인 dept 를 올려야 한다

### ※ 해쉬 조인시 주의사항

1. 작은 테이블이 메모리로 올라가도록 힌트를 부여하면 된다

### 64) 아래의 SQL을 튜닝하는데 해쉬조인되게 하고 emp 테이블이 메모리에 올라가도록 튜닝하시오!

```
select /*+ gather_plan_statistics leading(e d) use_hash(d) */ e.ename, d.loc
from emp e, dept d
where e.deptno = d.deptno;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	-----								
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10	-----								
11	0	SELECT STATEMENT		1		14	00:00:00.01	15	
12	* 1	HASH JOIN		1	14	14	00:00:00.01	15	
13	2	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	7	
14	3	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	7	
15	-----								

## 65) 작업이 SALESMAN 인 직원들의 이름, 월급, comm2 를 출력하시오!

```
create table bouns
```

```
as
```

```
select empno, sal*1.2 as comm2
```

```
from emp;
```

```
select /*+ gather_plan_statistics leading(e b) use_hash(b) */ e.ename, e.sal, b.comm2
```

```
from emp e, bonus b
```

```
where e.empno = b.empno and e.job='SALESMAN';
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

9	-----									
10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers		
11	-----									
12	0	SELECT STATEMENT		1		4	00:00:00.01	10		
13	* 1	HASH JOIN		1	4	4	00:00:00.01	10		
14	* 2	TABLE ACCESS FULL	EMP	1	4	4	00:00:00.01	7		
15	3	TABLE ACCESS FULL	BONUS	1	14	14	00:00:00.01	2		
16	-----									

위와같이 emp 와 bonus 둘 다 14 건으로 둘중에 작은 테이블이 없다면 조건에 의해 걸러지는 데이터가 작은것을 메모리로 올리면 된다. 작업이 SALESMAN 인 직원들의 데이터가 4건이므로 4건만 메모리로 올리고 bonus 와 조인

## 67) 아래의 SQL 의 조인순서와 방법을 직접 결정하시오!

```
select e.ename, d.loc
```

```
from emp e, dept d
```

```
where e.deptno=d.deptno and e.job='SALESMAN' -- 4건
```

```
and d.loc='CHICAGO'; -- 1건
```

```
>> dept-->emp , hash join
```

```
select /*+ gather_plan_statistics leading(d e) use_hash(e) */ e.ename, d.loc
```

```
from emp e, dept d
```

```
where e.deptno=d.deptno and e.job='SALESMAN' -- 4건
```

```
and d.loc='CHICAGO'; -- 1건
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

9	-----											
10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem	
11	-----											
12	0	SELECT STATEMENT		1		4	00:00:00.01	15				
13	* 1	HASH JOIN		1	1	4	00:00:00.01	15	1744K	1744K	593K (0)	
14	* 2	TABLE ACCESS FULL	DEPT	1	1	1	00:00:00.01	7				
15	* 3	TABLE ACCESS FULL	EMP	1	4	4	00:00:00.01	7				
16	-----											

이때, join 의 순서를 emp-->dept 순서로 하게 되면 아래와 같다

10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
11											
12	0	SELECT STATEMENT		1		4	00:00:00.01	15			
13	* 1	HASH JOIN		1	1	4	00:00:00.01	15	1856K	1856K	631K (0)
14	* 2	TABLE ACCESS FULL	EMP	1	4	4	00:00:00.01	7			
15	* 3	TABLE ACCESS FULL	DEPT	1	1	1	00:00:00.01	7			
16											

전자의 후자의 실행결과의 버퍼는 동일하지만(저용량이라 그런것) 메모리의 사용량은 후자가 더 큰 것을 볼 수 있다

### 68) emp와 dept와 bonus 를 조인해서 이름, 부서위치, comm2를 출력하시오!

```
select e.ename, d.loc, b.comm2
  from emp e, dept d, bonus b
 where e.deptno=d.deptno and e.empno=b.empno;
```

### 69) 위의 SQL을 조인순서와 조인방법을 아래와 같이 하시오!

조인순서 : dept --> emp --> bonus

조인방법 : hash / hash

```
select /*+ gather_plan_statistics leading(d e b) use_hash(e) use_hash(b) */ e.ename, d.loc, b.comm2
  from emp e, dept d, bonus b
 where e.deptno=d.deptno and e.empno=b.empno;
```

select \* from table(dbms\_xplan.display\_cursor(null,null,'allstats last'));

10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
11											
12	0	SELECT STATEMENT		1		14	00:00:00.01	18			
13	* 1	HASH JOIN		1	14	14	00:00:00.01	18	1476K	1476K	1492K (0)
14	* 2	HASH JOIN		1	14	14	00:00:00.01	15	1797K	1797K	980K (0)
15	3	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	7			
16	4	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	7			
17	5	TABLE ACCESS FULL	BONUS	1	14	14	00:00:00.01	2			
18											

dept, emp 를 hash join ---> 이 결과와 bonus 와 hash join

Id	Operation	Name
* 1	HASH JOIN	
* 2	HASH JOIN	
3	TABLE ACCESS FULL	DEPT ---> hash table
4	TABLE ACCESS FULL	EMP ---> probe table ---> (3~4) : hash table
5	TABLE ACCESS FULL	BONUS ---> probe table

### 70) 위 SQL 의 실행계획을 아래와 같이 나오게 하시오!

0	SELECT STATEMENT		1		14	00:00:00.01				
* 1	HASH JOIN		1	14	14	00:00:00.01				
2	TABLE ACCESS FULL	BONUS → hash table	1	14	14	00:00:00.01				
* 3	HASH JOIN		1	14	14	00:00:00.01				
4	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01				
5	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01				

Predicate Information (identified by operation id):

```
select /*+ gather_plan_statistics leading(d e b) use_hash(e) use_hash(b) swap_join_inputs(b) */
  e.ename, d.loc, b.comm2
  from emp e, dept d, bonus b
 where e.deptno=d.deptno and e.empno=b.empno;
```

select \* from table(dbms\_xplan.display\_cursor(null,null,'allstats last'));

만약 bonus table 이 크기가 작다면 이 실행계획이 좋은 실행계획이다 --> 순서를 편집할 수 있는게 장점!

※ hint

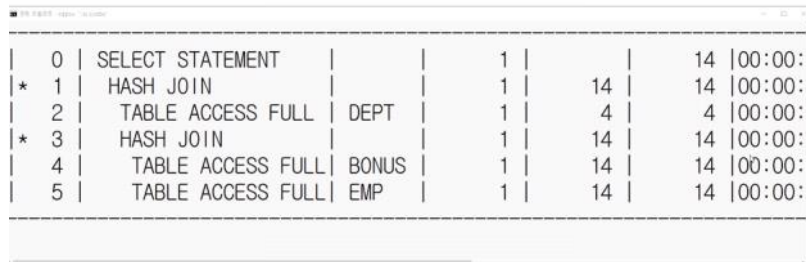
**swap\_join\_inputs : hash table 을 결정하는 힌트**

**no\_swap\_join\_inputs : probe table 을 결정하는 힌트**

2개의 테이블을 조인할 때 hash table 을 결정하는 것은 leading 힌트로 컨트롤 가능하지만

3개 이상의 테이블을 조인할 때는 hash table 을 결정하려면 leading 으로는 제어가 안되고 swap join inputs 사용

**71) 아래와 같이 실행계획이 나오게 하시오! 위 70번 SQL 활용**



0	SELECT STATEMENT		1	14	00:00:
* 1	HASH JOIN		1	14	00:00:
2	TABLE ACCESS FULL	DEPT	1	4	00:00:
* 3	HASH JOIN		1	14	00:00:
4	TABLE ACCESS FULL	BONUS	1	14	00:00:
5	TABLE ACCESS FULL	EMP	1	14	00:00:

```
select /*+ gather_plan_statistics leading(b e d) use_hash(e) use_hash(d) swap_join_inputs(d) */  
       e.ename, d.loc, b.comm2  
from emp e, dept d, bonus b  
where e.deptno=d.deptno and e.empno=b.empno;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

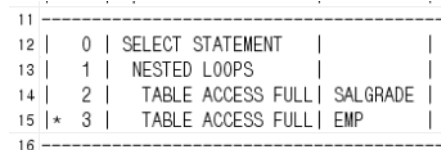
bonus ----hash join---- emp 한 뒤 이 결과와 dept 와 hash join을 한다. 이 때 swap\_join\_inputs 를 사용해서 dept 를 메모리에 올려서 위 join 을 수행한 결과로 이해할 수 있다

swap 은 해쉬 조인에서만 사용가능하며 nested loop 에서는 사용불가

**72) emp 와 salgrade 테이블을 조인해서 이름, 월급, 급여등급을 출력하고 hash join으로 유도하시오!**

```
select /*+ gather_plan_statistics 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;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```



11	-----
12	0   SELECT STATEMENT
13	1   NESTED LOOPS
14	2   TABLE ACCESS FULL   SALGRADE
15	* 3   TABLE ACCESS FULL   EMP
16	-----

위 SQL은 실행계획이 hash join 으로 풀리지 않는다

hash join 이 되려면 where 절의 조인조건이 이퀄(=) 이 되어야 한다

non equi join 은 해쉬 조인으로 수행되지 않는다

그렇다면 위와 같이 SQL 의 두 테이블이 둘 다 대용량인데 해쉬조인을 못쓰면 어떻게 튜닝을 해야하는가?

이때 사용하는 것이 sort merge join 이다

### □3. sort merge join

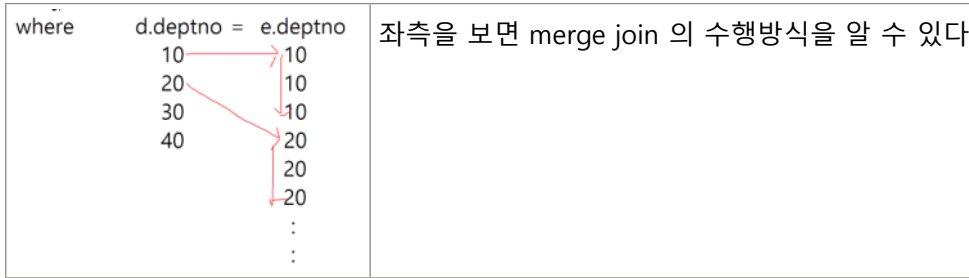
**sort merge join : 조인하려는 데이터의 양이 많으면서 조인조건이 이퀄이 아닐때, 검색성능을 높이기 위한 조인**

**sort merge join을 수행하면 연결고리가 되는 키 컬럼을 오라클이 알아서 정렬하고 조인한다**

ex)

```
select /*+ gather_plan_statistics leading(d e) use_merge(e) */ e.ename, d.loc, e.deptno
  from emp e, dept d
  where e.deptno=d.deptno;
```

위 SQL 의 결과를 보면 order by 절을 사용하지 않았음에도 deptno가 정렬된 것을 확인할 수 있는데, 이는 sort merge join 이 deptno 를 정렬해놓고 조인을 수행했다는 것을 의미한다



### 73) 아래의 SQL의 실행계획이 sort merge join 이 되게 하시오!

```
select /*+ gather_plan_statistics 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;

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

9	-----							
10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
11	-----							
12	0	SELECT STATEMENT		1		14	00:00:00.01	13
13	1	MERGE JOIN		1	1	14	00:00:00.01	13
14	2	SORT JOIN		1	5	5	00:00:00.01	6
15	3	TABLE ACCESS FULL	SALGRADE	1	5	5	00:00:00.01	6
16	* 4	FILTER		5		14	00:00:00.01	7
17	* 5	SORT JOIN		5	14	40	00:00:00.01	7
18	6	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	7
19	-----							

### ☆아래 그림 참고(이항분포 연습문제)

아래의 문제를 SQL 로 구현하여 평균과 분산과 표준편차를 구하세요!

**문제 02** 두 개의 동전을 300번 던져서 둘 다 앞면이 나오는 횟수를 확률변수  $X$ 라고 할 때,  $X$ 의 평균과 분산 및 표준편차를 구하여라.

**풀이** 두 개의 동전을 한 번 던져서 둘 다 앞면이 나올 확률은  $\frac{1}{4}$ 이고, 매회 시행은 독립시행이다. 따라서 확률변수  $X$ 는 이항분포  $B\left(300, \frac{1}{4}\right)$ 을 따르므로

$$E(X) = 300 \times \frac{1}{4} = 75, V(X) = 300 \times \frac{1}{4} \times \frac{3}{4} = \frac{225}{4}$$

$$\sigma(X) = \sqrt{V(X)} = \sqrt{\frac{225}{4}} = \frac{15}{2}$$

☞  $E(X) = 75, V(X) = \frac{225}{4}, \sigma(X) = \frac{15}{2}$

```
undefine n
with coin as ( select round( dbms_random.value(0, 1)) as c1,
                    round( dbms_random.value(0, 1)) as c2
  from dual
  connect by level <= &n)
select count(*) as avg, -- count(*) = n * (count(*)/n)
       round ( count(*) * (1-count(*)/&n), 3) as var,
       round( sqrt( count(*) * (1-count(*)/&n) ), 3) as standard_var
```

```
from coin
where c1 = 1 and c2 = 1;
```



# ##1120

2020년 11월 20일 금요일 오전 9:44

## SQL 목차로 복습

SQL 튜닝목차

1. 인덱스 SQL 튜닝(index access)
2. 조인 문장 튜닝
3. 서브쿼리 문장 튜닝
4. 데이터 분석함수를 이용한 튜닝
5. 자동 SQL 튜닝

데이터 분석가(데이터 사이언티스트) : SQL, 파이썬, R 코딩을 통해서 데이터 분석  
대용량 데이터 --> 빅데이터 시대 --> SQL + SQL 튜닝기술

조인문장 튜닝시 주요 기술

1. 조인순서 : leading, ordered
2. 조인방법

method	hint	content
nested loop join	use_nl	적은량의 데이터
hash join	use_hash	대용량 데이터
sort merge join	use_merge	대용량 데이터 (3건이상)

### ※ 서버별 SQL 사용 예시

서버명	설명	크기	SQL
OLTP 서버	실시간 조회 처리를 위한 데이터 저장소	작은정보	nested loop join
DW 서버	데이터 저장소(big) 건강보험 심사평가원	큰 정보	hash, merge join 데이터 분석함수 파티션 테이블, 병렬처리

## ■2. join 소목차

1. nested loop 조인
2. hash 조인
3. sort merge 조인
4. 조인 순서의 중요성
5. outer 조인
6. 스칼라 서브쿼리를 이용한 조인
7. 조인을 내포한 DML 문 튜닝
8. 고급 조인 문장 튜닝

오늘 진도 --> outer join 부터

## □4. 조인 순서의 중요성

조인순서에 따라 수행 성능이 달라질 수 있다

조인순서는 조인하려는 시도횟수가 적은 테이블을 driving 테이블로 선정해서 조인순서를 고려하는게 중요

선행 테이블 : driving table

후행 테이블 : driven table

#### 1. nested loop join

nested loop

table access full dept ---> 선행 테이블, driving table

table access full emp ---> 후행 테이블, driven table

#### 2. hash loop join

hash join

table access full dept ---> hash table : 메모리로 올라가는 테이블

table access full emp ---> probe table : 디스크에 있는 테이블

**nested loop join, hash loop join 둘 다 먼저 읽는 테이블은 크기가 작거나 검색조건절에 의해서 액세스되는 건수가 작은 것을 먼저 읽는 것이 조인순서에 있어 중요함**

### ※ 조인문장을 튜닝하는 순서

```
select e.ename, d.loc
```

```
from emp e, dept d
```

```
where e.deptno = d.deptno
```

```
and e.job='SALESMAN'
```

```
and d.loc='CHICAGO';
```

>>tuning...

```
select /*+ gather_plan_statistics leading(d e) use_nl(e) */ e.ename, d.loc
```

```
from emp e, dept d -- emp 14건, dept 4건
```

```
where e.deptno = d.deptno
```

```
and e.job='SALESMAN' -- 4건
```

```
and d.loc='CHICAGO'; -- 1건
```

#### 1. 조인순서를 정한다 : from 절의 테이블의 건수를 다 확인한다

검색조건의 건수도 일일이 다 확인한다

#### 2. 조인방법을 정한다 : 접속한 서버의 종류를 확인한다(OLTP / DW)

위 예시 SQL 과 같이 데이터건수를 확인해서 대용량이 아니면 nested loop join 을 사용

use\_nl 을 날려서 너무 느리면 대용량 데이터로 판단(정확한 데이터 개수로 정의되진 않음)

use\_hash 날렸을 때 갑자기 빨라지면 그대로 쓰면 된다

### 74) 아래의 조인문장의 조인순서와 조인방법을 결정하시오!

```
select e.ename, d.loc, e.sal
```

```
from emp e, dept d
```

```
where e.deptno = d.deptno
```

```
and e.ename = 'KING';
```

>>

```
select /*+ gather_plan_statistics leading(e d) use_nl(d) */ e.ename, d.loc, e.sal
```

```
from emp e, dept d
```

```
where e.deptno = d.deptno
```

```
and e.ename = 'KING'; -- 1건
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

75) emp와 dept가 대용량 테이블이라고 가정하고 아래의 SQL의 조인순서와 조인방법을 결정하시오!

(emp 는 12000 만 건, dept 는 5400만 건)

```
select /*+ gather_plan_statistics leading(d e) use_hash(e) */ e.ename, d.loc
  from emp e, dept d
  where e.deptno = d.deptno;
```

select \* from table(dbms\_xplan.display\_cursor(null,null,'allstats last'));

sort merge 조인은 내부적으로 정렬을 일으키는 단점이 있어서 deptno 를 정렬해서 볼 필요가 없으면 hash join을 사용

76) 아래의 SQL의 조인순서와 조인방법을 결정하시오! (단, emp 와 salgrade 가 대용량 테이블이라고 가정(DW서버) )

```
select /*+ gather_plan_statistics 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;
```

select \* from table(dbms\_xplan.display\_cursor(null,null,'allstats last'));

10	Id	Operation	Name	ξ
11				
12	0	SELECT STATEMENT		
13	1	MERGE JOIN		
14	2	SORT JOIN		
15	3	TABLE ACCESS FULL	SALGRADE	
16	* 4	FILTER		
17	* 5	SORT JOIN		
18	6	TABLE ACCESS FULL	EMP	
19				

위 경우 where 절에 = 연산자가없어서 hash join 을 하게 되면 자동으로 nested loop join 으로 옵티마이저가 돌린다

## □5. outer 조인의 튜닝방법

outer 조인의 조인순서는 outer 조인 sign 이 없는쪽에서 있는쪽으로 고정 이 된다. 이 때문에 조인 순서의 변경이 어려워 튜닝이 힘든데 이를 개선할 수 있는 힌트가 있다

ex)

```
insert into emp(empno, ename, sal, deptno) values(2921, 'JACK', 4500, 70);
```

```
select /*+ gather_plan_statistics */ e.ename, d.loc
  from emp e, dept d
  where e.deptno = d.deptno(+);
```

결과에 JACK 이 출력되면서 부서위치가 비어있는것을 확인가능 --> JACK 이 어느 부서에도 배치되지 않음

이때, JACK 이 결과로 출력되게 하려면 equi join 으로는 안되고 outer join 을 사용해야 한다

조인순서를 확인해보자 ↓ ↓

8									
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10									
11	0	SELECT STATEMENT		1		16	00:00:00.01	14	
12	* 1	HASH JOIN OUTER		1	14	16	00:00:00.01	14	
13	2	TABLE ACCESS FULL	EMP	1	14	16	00:00:00.01	7	
14	3	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	7	
15									

optimizer 가 emp 테이블을 driving 으로 읽은 것은 outer join sign 이 없는 쪽에서 있는 쪽으로 조인하기 때문이다

즉, emp table 에서 JACK 은 deptno=70 이기 때문에 dept 테이블에는 없는 번호이기 때문에 있는 쪽에서 먼저 읽어야 조인할 수 있기 때문에 이렇게 된 것이다

77) 아래의 조인순서를 dept ---> emp 순이 되게 하시오

```
select /*+ gather_plan_statistics leading(d e) use_hash(e) */ e.ename, d.loc
```

```
from emp e, dept d
```

```
where e.deptno = d.deptno(+);
```

조인순서를 변경하기 위해 leading(d e) use\_hash(e) 힌트를 주어도 조인순서는 변경되지 않는다

outer join 은 조인순서가 outer join sign 이 없는쪽(emp)에서 있는쪽 (deptno(+))으로 고정되기 때문

### 78) 위 outer join 의 순서를 dept --> emp 순이 되게 하시오!

```
select /*+ gather_plan_statistics leading(d e) use_hash(e)
```

```
swap_join_inputs(d) */ e.ename, d.loc
```

```
from emp e, dept d
```

```
where e.deptno = d.deptno(+);
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

9	-----								
10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
11	-----								
12	0	SELECT STATEMENT		1		16	00:00:00.01	14	
13	* 1	HASH JOIN RIGHT OUTER		1	16	16	00:00:00.01	14	
14	2	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	7	
15	3	TABLE ACCESS FULL	EMP	1	16	16	00:00:00.01	7	
16	-----								

**swap\_join\_inputs** 를 사용해서 dept 테이블을 hash 테이블로 구성해서 dept 테이블을 먼저 driving 할 수 있도록 변경  
outer join 튜닝은 hash join 으로만 가능한데 이는 swap\_join\_inputs hint 가 hash join 에서만 가능하기 때문  
결과도 JACK 이 잘 출력되는 것도 확인된다

### 79) emp 와 bonus 를 조인해서 이름, comm2 를 출력하는데 outer join 을 이용해서 JACK 도 출력되게 하시오!

```
create table bonus
```

```
as
```

```
select empno, sal*1.5 as comm2
```

```
from emp;
```

```
>>
```

```
select e.ename, b.comm2
```

```
from emp e, bonus b
```

```
where e.empno = b.empno(+);
```

### 80) 위 조인순서가 bonus --> emp 순이 되도록 하시오!

```
select /*+ gather_plan_statistics leading(b e) use_hash(e)
```

```
swap_join_inputs(b) */ e.ename, b.comm2
```

```
from emp e, bonus b
```

```
where e.empno = b.empno(+);
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

9	-----								
10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
11	-----								
12	0	SELECT STATEMENT		1		15	00:00:00.01	9	
13	* 1	HASH JOIN RIGHT OUTER		1	16	15	00:00:00.01	9	
14	2	TABLE ACCESS FULL	BONUS	1	14	14	00:00:00.01	2	
15	3	TABLE ACCESS FULL	EMP	1	16	15	00:00:00.01	7	
16	-----								

swap\_join\_inputs 사용 안하면 leading 으로 순서를 지정해줘도 join 순서는 emp-->bonus 로 고정된다

※ SQL 실무 연습 관련

Programers (프로그래머스) 참고

## □6. 스칼라 서브쿼리를 이용한 조인

scarlar subquery : select 절의 subquery

select 문에서 subquery 를 쓸 수 있는 절 (파란색:가능 / 붉은색:불가)

select	scalar suq.q
from	in line view
where	
group by	
having	
order by	in line view

81) 이름, 월급, 사원 테이블에서의 최대월급을 출력하시오! ( 단, scalar sub.q.사용)

```
select /*+ gather_plan_statistics */ ename, sal,(select max(sal) from emp) as max
from emp;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10	11	0	SELECT STATEMENT		1		15	00:00:00.01	7
12	1		SORT AGGREGATE		1	1	1	00:00:00.01	7
13	2		TABLE ACCESS FULL	EMP	1	16	15	00:00:00.01	7
14	3		TABLE ACCESS FULL	EMP	1	16	15	00:00:00.01	7
15									

>>tuning...

```
select /*+ gather_plan_statistics */
ename, sal,max(sal) over() as max
from emp;
```

8	9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10	11	0	SELECT STATEMENT		1		15	00:00:00.01	7
12	1		WINDOW BUFFER		1	16	15	00:00:00.01	7
13	2		TABLE ACCESS FULL	EMP	1	16	15	00:00:00.01	7
14									

튜닝 전에는 emp 테이블을 두번 액세스했는데 튜닝 후에는 emp 테이블을 한번만 액세스해도 같은 결과가 출력된다

82) 아래의 SQL을 튜닝하시오!

```
select ename, sal, (select max(sal) from emp) as max_sal,
(select min(sal) from emp) as min_sal
from emp;
```

9	10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
11	12	0	SELECT STATEMENT		1		15	00:00:00.01	7
13	1		SORT AGGREGATE		1	1	1	00:00:00.01	7
14	2		TABLE ACCESS FULL	EMP	1	16	15	00:00:00.01	7
15	3		SORT AGGREGATE		1	1	1	00:00:00.01	7
16	4		TABLE ACCESS FULL	EMP	1	16	15	00:00:00.01	7
17	5		TABLE ACCESS FULL	EMP	1	16	15	00:00:00.01	7
18									

>>tuning...

```
select ename, sal, max(sal) over() as max_sal,
min(sal) over() as min_sal
from emp;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

9	-----								
10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
11	-----								
12	0	SELECT STATEMENT		1		15	00:00:00.01	7	
13	1	WINDOW BUFFER		1	16	15	00:00:00.01	7	
14	2	TABLE ACCESS FULL	EMP	1	16	15	00:00:00.01	7	
15	-----								
16									

### 83) 아래의 SQL을 튜닝하시오!

```
select e.deptno, e.ename, e.sal, v.d_avg
  from emp e, ( select deptno, avg(sal) d_avg
                from emp
                group by deptno) v
 where e.deptno = v.deptno and e.sal > v.d_avg;
>>tuning... (emp table access once)
```

```
select /*+ gather_plan_statistics */ *
  from( select deptno, ename, sal,
               avg(sal) over(partition by deptno) as avg_d
        from emp)
 where sal > avg_d;
```

9	-----								
10	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
11	-----								
12	0	SELECT STATEMENT		1		6	00:00:00.01	7	
13	* 1	VIEW		1	16	6	00:00:00.01	7	
14	2	WINDOW SORT		1	16	15	00:00:00.01	7	
15	3	TABLE ACCESS FULL	EMP	1	16	15	00:00:00.01	7	
16	-----								

### 84) 부서번호, 이름, 월급, 순위를 출력하는데 순위가 부서번호별로 각각 월급이 높은 순서대로 순위를 부여하고 순위가 1등인 직원들만 출력하시오!

```
with test as ( select deptno, ename, sal,
                     dense_rank() over( partition by job order by sal desc) as rank
               from emp)
select *
  from test
 where rank = 1;
```

## □7. 조인을 내포한 DML 문장 튜닝

DML 문을 이용한 서브쿼리문의 튜닝

ex) ALLEN의 월급을 KING의 월급으로 변경하시오!

```
update emp
  set sal = (select sal from emp where ename='KING')
 where ename = 'ALLEN';
```

### 85) 직원 테이블에 loc 컬럼을 추가하고 해당 직원이 속한 부서위치로 값을 갱신하시오!

```
alter table emp
```

```
add loc varchar2(20);
```

```
update emp e
```

```
set loc = (select loc
           from dept d
           where e.deptno = d.deptno);
```

emp 테이블의 컬럼이 서브쿼리 안으로 들어가면 서브쿼리부터 수행되는것이 아니라 메인 쿼리(update)문부터 실행된다  
update 문을 수행하는데 제일 먼저 맨 위에 있는 KING 의 부서위치를 갱신하기 위해서 KING 의 부서번호 10번을 서브쿼리 안으로 넣어서 서브쿼리에서 10번 부서번호의 부서위치를 출력하고 KING의 부서위치를 NEW YORK 으로 갱신  
그 다음행의 BLAKE 도 같은 방식으로 갱신하며 emp 테이블의 14명 직원들의 데이터도 같은 방법으로 갱신하는데, 이때 update 가 14번 실행된다. 대용량일수록 그 횟수가 늘어나므로 지독한 성능저하를 맛볼수 있다.

위 85번의 SQL은 튜닝 가능하다

```
alter table emp
```

```
add loc varchar2(20);
```

```
merge into emp e
```

```
using dept d
```

```
on (e.deptno = d.deptno)
```

```
when matched then
```

```
update set e.loc = d.loc
```

merge 는 부모키가 없어도 수행이 가능하다. 많이들 사용함

86) 사원 테이블에 sal2 라는 컬럼을 추가하시오!

```
alter table emp
```

```
add sal2 number(10);
```

87) 지금 추가한 sal2 에 해당 사원의 월급으로 값을 갱신하시오!

```
update emp
```

```
set sal2 = sal;
```

88) 아래의 복합뷰를 생성하고 view를 쿼리하시오!

```
create view emp_dept
```

```
as
```

```
select e.ename, e.loc as emp_loc, d.loc as dept_loc
```

```
from emp e, dept d
```

```
where e.deptno=d.deptno;
```

```
select * from emp_dept;
```

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

### 89) emp\_dept 뷰를 수정해서 emp\_loc 의 값을 dept\_loc 의 값으로 갱신하시오!

update emp\_dept

set emp\_loc = dept\_loc;

SQL 오류: ORA-01779: 키-보존된것이 아닌 테이블로 대응한 열을 수정할 수 없습니다  
01779. 00000 - "cannot modify a column which maps to a non key-preserved table"  
\*Cause: An attempt was made to insert or update columns of a join view which  
map to a non-key-preserved table.  
\*Action: Modify the underlying base tables directly.

### 90) 위 update 문이 수행되게 설정하시오!

위 복합뷰를 갱신하려면 dept 테이블에 deptno 에 primary key 제약을 걸어주면 갱신할 수 있다

alter table dept

add constraint dept\_deptno\_pk primary key(deptno);

update emp\_dept

set emp\_loc = dept\_loc;

select \* from emp\_dept;

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

또한 view 를 수정하면 view 생성시 연결된 table 도 같이 수정된다(위 경우에는 emp table도 같이 수정된다)

### 91) 만약 dba 가 view 를 승인해주지 않아 만들 수 없다면 어떻게 해야 하는가?

update 문에서 서브쿼리를 쓸 수 있는 절

update <--- 서브쿼리 사용가능

set <--- 서브쿼리 사용가능

where <--- 서브쿼리 사용가능

update ( select e.ename, e.loc as emp\_loc, d.loc as dept\_loc

from emp e, dept d

where e.deptno=d.deptno )

set emp\_loc = dept\_loc;

뷰를 만들어주지 않을때는 update 에 서브쿼리를 사용해서 뷰의 쿼리문을 직접 작성하고 set 절에서 update 하면된다

위 SQL 이 작동하기 위해서는 emp table 에 loc 컬럼을 추가하고 dept table의 deptno에 pk 를 설정해주어야 한다

### 92)@demo / emp 테이블에 dname 컬럼을 추가하고 해당 사원의 부서명으로 값을 갱신하시오!

>>튜닝전

alter table emp add dname varchar2(10);

update emp e

set dname = (select dname

from dept d

where e.dname=d.dname);

>>tuning... without view



```

alter table emp
  add dname varchar2(10);
alter table dept
  add constraint dept_dname_pk primary key(deptno);
update ( select e.dname as emp_dname, d.dname as dept_dname
        from emp e, dept d
        where e.deptno=d.deptno)
set emp_dname = dept_dname;
>>using merge
alter table emp
  add dname varchar2(10);
merge into emp e
using dept d
on (e.deptno=d.deptno)
when matched then
update set e.dname=d.dname;

```

---

### ※primary key 와 update 의 관계

key 설정시에 조인조건이 e.deptno=d.deptno 에서 부모컬럼이 dept table 의 deptno 이므로 deptno 에 pk를 걸어주어야 한다. 이는 dept table 의 deptno 에 null, 중복값을 허용하지 않겠다는 보증이므로 신뢰도가 올라간다.

아래 예시를 보자

```
select * from dept;
```

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

위 출력값에서 deptno 가 10번, 20번 30번 ~ 외 다양한 중복값, null 값이 있을 때 e.deptno=d.deptno 조인조건으로 조인을 할 경우 그 값은 신뢰할 수 없기 때문에 pk 없는 상태의 dept.deptno 와 emp.deptno 를 조인하여 update 를 할 수 없는 것이다.

## □8. 고급 조인문장 튜닝

" 뷰 안의 조인문장의 순서를 변경하고자 할 때 사용하는 튜닝방법"

ex) @demo

```
create view emp_dept
```

```
as
```

```
select e.empno, e.ename, e.sal, e.deptno, d.loc
```

```
from emp e, dept d
```

```
where e.deptno = d.deptno;
```

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

e.empno 의 컬럼명이 empno 로 e. 을 제외하고 들어간 것을 볼 수 있다

### 93) emp\_dept view 와 salgrade 테이블을 서로 조인해서 이름, 월급, 부서위치, 급여등급을 출력하시오!

```
select /*+ gather_plan_statistics leading(s v) use_nl(v) */
       v.ename, v.sal, v.loc, s.grade
from emp_dept v, salgrade s
where v.sal between s.losal and s.hisal;
```

### 94) 93번의 실행계획을 보시오

9	-----				
10	Id	Operation	Name	Starts	E
11	-----				
12	0	SELECT STATEMENT		1	
13	* 1	HASH JOIN		1	
14	2	MERGE JOIN		1	
15	3	SORT JOIN		1	
16	4	TABLE ACCESS FULL	SALGRADE	1	
17	* 5	FILTER		5	
18	* 6	SORT JOIN		5	
19	7	TABLE ACCESS FULL	EMP	1	
20	8	TABLE ACCESS FULL	DEPT	1	
21	-----				

93번의 hint 에서는 salgrade --> emp\_deptn 순으로 nested join 을 하라고 지시했지만 실제로는 다르게 실행되었다.

힌트가 무시된 이유는 view 를 해체해 버려서 이다. view 가 해체되지 않았다면 실행계획에 view 가 보인다.

이를 방지하기 위해서 **view** 를 해체하지 못하게 하는 hint 인 **no\_merge** 를 주어야 한다.

93번의 SQL을 다시 수정해서 view가 해체되지 못하게 실습해보자

```
select /*+ gather_plan_statistics no_merge(v) leading(s v) use_nl(v) */
       v.ename, v.sal, v.loc, s.grade
from emp_dept v, salgrade s
where v.sal between s.losal and s.hisal;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

9	-----			
10	Id	Operation	Name	Starts
11	-----			
12	0	4SELECT STATEMENT		1
13	1	3NESTED LOOPS		1
14	2	1TABLE ACCESS FULL	SALGRADE	1
15	* 3	2VIEW	EMP_DEPT	5
16	* 4	HASH JOIN		5
17	5	TABLE ACCESS FULL	DEPT	5
18	6	TABLE ACCESS FULL	EMP	5
19	-----			

좌측 박스는 view 로 묶인 개념으로 이해

hint : no\_merge(v) ---> "view를 해체하지 말것 " 이라는 힌트

merge(v) ---> "view 를 해체해라" 라는 힌트

위 실행계획의 해석 : salgrade 를 full scan 하고 view 를 스캔하면서 nested loop join 을 수행

※ view 내부 의 테이블의 조인순서 변경 ☆☆

```
select /*+ gather_plan_statistics no_merge(v) leading(s v) use_nl(v) */
       v.ename, v.sal, v.loc, s.grade
from emp_dept v, salgrade s
where v.sal between s.losal and s.hisal;
```

현재 위 SQL 의 view 안의 조인순서는 dept --> emp 순이다. (바로 위 실행계획 그림 참고)

이를 emp --> dept 순으로 변경이 되는가? 됩니다.

>>add hint : leading(v.e v.d) use\_hash(v.d)

```
select /*+ gather_plan_statistics no_merge(v) leading(s v) use_nl(v)
       leading(v.e v.d) use_hash(v.d) */ v.ename, v.sal, v.loc, s.grade
from emp_dept v, salgrade s
where v.sal between s.losal and s.hisal;
```

10	-----			
11	Id	Operation	Name	
12	-----			
13	0	SELECT STATEMENT		
14	1	NESTED LOOPS		
15	2	TABLE ACCESS FULL	SALGRADE	
16	* 3	VIEW	EMP_DEPT	
17	* 4	HASH JOIN		
18	5	TABLE ACCESS FULL	EMP	
19	6	TABLE ACCESS FULL	DEPT	
20	-----			

위 힌트에서 leading(v.e v.d) use\_hash(v.d) 에서 v. 을 붙이는 이유는 join 시 view 테이블 약칭을 v 로 주었기 때문

95) @demo / 위의 실행계획이 아래와 같이 나오게 하시오!

9	-----			
10	Id	Operation	Name	
11	-----			
12	0	SELECT STATEMENT		
13	1	NESTED LOOPS		
14	2	TABLE ACCESS FULL	SALGRADE	
15	* 3	VIEW	EMP_DEPT	
16	4	NESTED LOOPS		
17	5	TABLE ACCESS FULL	EMP	
18	* 6	TABLE ACCESS FULL	DEPT	
19	-----			

```
select /*+ gather_plan_statistics no_merge(v) leading(s v) use_nl(v)
       leading(v.e v.d) use_nl(v.d) */ v.ename, v.sal, v.loc, s.grade
from emp_dept v, salgrade s
where v.sal between s.losal and s.hisal;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

96) 위 실행계획을 아래와 같이 출력되게 하시오!

9	-----			
10	Id	Operation	Name	
11	-----			
12	0	SELECT STATEMENT		
13	1	NESTED LOOPS		
14	2	VIEW	EMP_DEPT	
15	3	NESTED LOOPS		
16	4	TABLE ACCESS FULL	EMP	
17	* 5	TABLE ACCESS FULL	DEPT	
18	* 6	TABLE ACCESS FULL	SALGRADE	
19	-----			

```
select /*+ gather_plan_statistics no_merge(v) leading(v s) use_nl(s)
        leading(v.e v.d) use_nl(v.d) */ v.ename, v.sal, v.loc, s.grade
from emp_dept v, salgrade s
where v.sal between s.losal and s.hisal;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

※ 버퍼의 개수는 가장 위에 있는것 (0) 번에 있는 select statement row 값이 총합이다. (누적으로 합산됨)

97) 이번에는 아래와 같이 실행계획이 나오게 하시오! ( 이 시리즈에서 가장 효율적인 튜닝방법)

10	-----			
11	Id	Operation	Name	S
12	-----			
13	0	SELECT STATEMENT		
14	1	NESTED LOOPS		
15	2	TABLE ACCESS FULL	SALGRADE	
16	* 3	VIEW	EMP_DEPT	
17	* 4	HASH JOIN		
18	5	TABLE ACCESS FULL	DEPT	
19	6	TABLE ACCESS FULL	EMP	
20	-----			

```
select /*+ gather_plan_statistics no_merge(v) leading(s v) use_nl(v)
        leading(v.d v.e) use_hash(v.e) */ v.ename, v.sal, v.loc, s.grade
from emp_dept v, salgrade s
where v.sal between s.losal and s.hisal;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

view 를 사용하여 조인하는 어떤 SQL 의 성능이 느리다면 위와 같은 방법으로 튜닝을 하면 효과적으로 튜닝할 수 있다.

1)no\_merge(v) 2)leading(v.e v.d) use\_nl(v.d)

### ■3. 서브쿼리문 튜닝

```
select ename, sal
from emp
where sal > ( select sal
              from emp
              where ename = 'JONES');
```

서브쿼리 문장 튜닝의 기술은 크게 2가지로 나누어 진다

서브쿼리 튜닝 방법	hint
1. 순수하게 서브쿼리문으로 수행하는 방법	no_unnest
2. 서브쿼리를 조인으로 변경하여 수행하는 방법	unnest

nest : 감싸다

unnest : 감싸지 않고 풀어헤치다

no\_unnest : 감싸다\*2 (부정\*부정=강한긍정)

☆98 ##24) 1과 10 사이의 숫자 중 소수만 출력하시오! (단, 소수는 1과 자기 자신의 수로만 나눌 수 있는 수이다)

소수 : 2, 3, 5, 7

**>>1 solo column group & self join**

```
with num as (select level x
              from dual
              connect by level <= 10)
```

```
select a.x
       from num a, num b
       where mod(a.x,b.x)=0
       group by a.x
       having count(*)=2;
```

**>>2 exist**

```
select 소수
       from (select level as 소수
              from dual
              where level > 1
              connect by level <= 1000)
where not exists (select level
                  from dual
                  where mod (소수, level) = 0 and level > 1
                  connect by level <= 소수 -1);
```

**>>3 count(\*) over(...)**

```
with test as( select x, count(*) over(partition by x) z
               from ( select level X   from dual
                       connect by level <= 10 ) ,
               ( select level Y   from dual
                 connect by level <= 10 )
               where mod(x,y) = 0
               )
select distinct(x)
from test
where z = 2;
```

**>>4**

```
with num1 as (select level x
              from dual
              connect by level <= 10),
num2 as (select level y
         from dual
         connect by level <= 10)
select n1.x
       from num1 n1, num2 n2
       where mod(n1.x,n2.y)=0
       group by n1.x
       having count(*)=2;
```



# ##1123

2020년 11월 21일 토요일 오전 12:50

## ■ SQL 튜닝 목차

### 1. index SQL 튜닝

※ index access	hint
1. index range scan	index
2. index unique scan	index
3. index skip scan	index_ss
4. index full scan	index_fs
5. index fast full scan	index_ffs
6. index merge scan	and_equal
7. index bitmap merge scan	index_combine
8. index join	index_join

### 2. 조인 문장 튜닝

method	hint	content
nested loop join	use_nl	적은량의 데이터
hash join	use_hash	대용량 데이터
sort merge join	use_merge	대용량 데이터 (3건이상)

### 3. 서브쿼리 문장 튜닝 --> 오늘 진도

#### 4. 데이터 분석함수를 이용한 튜닝

#### 5. 자동 SQL 튜닝

## ■ 3. 서브쿼리문 튜닝

ex) DALLAS 에서 근무하는 직원들의 이름과 월급을 출력하시오!

```
select ename, sal
  from emp
 where deptno in ( select deptno
                   from dept
                   where loc = 'DALLAS');
```

99) 위의 서브쿼리문을 조인으로 수행하여 같은 결과를 출력하시오!

```
select e.ename, e.sal
  from emp e, dept d
 where e.deptno=d.deptno and d.loc = 'DALLAS';
```

아래 SQL의 실행계획을 보시오

```
select /*+ gather_plan_statistics */ ename, sal
```

```

from emp
where deptno in ( select deptno
                  from dept
                  where loc = 'DALLAS');

```

```

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));

```

실행계획에 filter 가 보이면 서브쿼리로 실행한 것이고, hash join 이 보이면 join 으로 변경하여 수행한 것이다  
 위 SQL 은 hash join 으로 서브쿼리를 변경하여 수행하였다  
 이때 속도는 조인이 빠를때도 있고 서브쿼리문이 빠를 때도 있다.

#### ※ 서브쿼리의 실행계획 2가지

서브쿼리 튜닝 방법	hint
1. 순수하게 서브쿼리문으로 수행하는 방법	<b>no_unnest</b>
2. 서브쿼리를 조인으로 변경하여 수행하는 방법	<b>unnest</b>

nest : 감싸다

unnest : 감싸지 않고 풀어헤치다

no\_unnest : 감싸다\*2 (부정\*부정=강한긍정)

#### 100) 아래의 SQL의 실행계획이 조인으로 풀리게 하시오!

```

select /*+ gather_plan_statistics */ ename, sal
  from emp
  where deptno in ( select /*+ unnest */ deptno
                   from dept
                   where loc = 'DALLAS');

```

```

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));

```

unnest 힌트가 유리한 경우는 서브쿼리문의 실행계획이 filter 가 나오면서 성능이 너무 느릴때 조인의 방법 중 가장 강력한 hash 조인으로 수행되게 하면서 성능을 높이고 싶을 때 유용함

#### 101) 아래의 실행계획이 조인이 아닌 filter 가 실행계획에 출력되는 순수한 서브쿼리문으로 실행되게 하시오!

```

select /*+ gather_plan_statistics */ ename, sal
  from emp
  where deptno in ( select /*+ no_unnest */ deptno
                   from dept
                   where loc = 'DALLAS');

```

```

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));

```

no\_unnest를 사용하여 순수하게 서브쿼리문으로 수행됨. 하지만 hash join semi 로 수행되었을때의 버퍼보다 약 2배이상 버퍼가 증가된 것을 볼 수 있다. 위 서브쿼리의 테이블 2개(dept, emp)가 대용량인 경우는 unnest 를 써서 해쉬조인으로 수행되는것이 유리하고, 저용량인 경우 굳이 메모리를 사용하는 hash join 으로 유도하지 않고 서브쿼리문의 실행계획으로(filter) 수행되는것이 유리하다.

#### 102) 아래의 SQL의 실행계획을 확인하고 조인으로 변경되어서 수행되게 하시오!

```

select /*+ gather_plan_statistics */ ename, sal
  from emp
  where deptno = ( select deptno
                  from dept
                  where deptno=10);

```



```

10 | Id | Operation | Name |
11 |-----|-----|
12 | 0 | SELECT STATEMENT | |
13 |* 1 | TABLE ACCESS FULL | EMP |
14 |* 2 | TABLE ACCESS FULL | DEPT |
15 |-----|-----|

```

위 SQL 은 순수하게 서브쿼리문으로 수행한 SQL인데 서브쿼리, 메인쿼리 순으로 수행하였다

그런데 위 SQL에서 unnest 힌트를 주어도 실행계획은 그대로이며 **where deptno in ( select /\*+ unnest \*/ deptno ~**으로 변경해야 unnest hint 가 작동한다. 그 이유는 서브쿼리에서 메인쿼리로 1건만 리턴(=) 되면 unnest 힌트를 주어도 옵티마이저가 해쉬조인으로 풀지 않는다. 이 상황에서 해쉬조인으로 풀고 싶다면 = 대신 in 을 사용해주면 된다.

서브쿼리 튜닝 방법 box	hint	
1. 순수하게 서브쿼리문으로 수행하는 방법	<b>no_unnest</b>	1, 1-2, 1-3 같이 사용
1-2 서브쿼리부터 수행	<b>push_subq</b>	
1-3 메인쿼리부터 수행	<b>no_push_subq</b>	
2. 서브쿼리를 조인으로 변경하여 수행하는 방법	<b>unnest</b>	
2-1 semi join		
- nested loop semi join	<b>nl_sj</b>	
- hash semi join	<b>hash_sj</b>	
- merge semi join	<b>merge_sj</b>	
2-2 anti join		
- nested loop anti join	<b>nl_aj</b>	
- hash anti join	<b>hash_aj</b>	
- merge anti join	<b>merge_aj</b>	

**103) 아래의 SQL 의 실행계획이 조인으로 풀리지 말고 서브쿼리문으로 수행되게 하는데 서브쿼리문부터 수행되게 하시오!**

```

select /*+ gather_plan_statistics */ ename, sal
  from emp
  where deptno = ( select deptno
                   from dept
                   where deptno=10);

```

>>

```

select /*+ gather_plan_statistics */ ename, sal
  from emp
  where deptno = ( select/*+ no_unnest push_subq */ deptno
                   from dept
                   where deptno=10);

```

**104) 위 SQL 의 실행계획이 메인쿼리부터 수행되게 하시오! (순수하게 서브쿼리로 수행되면서 메인쿼리부터 수행)**

```

select /*+ gather_plan_statistics */ ename, sal
  from emp
  where deptno = ( select/*+ no_unnest no_push_subq */ deptno
                   from dept
                   where deptno=10);

```

9					
10	Id	Operation	Name	S	
11					
12	0	SELECT STATEMENT			
13	* 1	<b>FILTER</b>			
14	2	TABLE ACCESS FULL	EMP		
15	* 3	TABLE ACCESS FULL	DEPT		
16					

no\_unnest 와 no\_push\_subq 는 서로 짝궁 힌트이다. no\_unnest 는 조인으로 풀지 말고 서브쿼리문으로 수행해라~ 라는 힌트이고 이 힌트를 먼저 써야 조인으로 풀지 않고 서브쿼리문으로 수행되고, 그 이후에 no\_push\_subq 힌트가 동작한다. 대체로 push\_subq 힌트가 서브쿼리문으로 수행되었을때 더 유리한 힌트이다. 왜냐하면 서브쿼리문부터 수행하면서 데이터를 검색해 메인쿼리로 넘겨주기만 하면 되기 때문이다. 만약 메인쿼리부터 수행된다면 메인 쿼리에 있는 부서번호중에 서브쿼리에 있는 부서번호를 찾기 위해 일일이 스캔하면서 찾는작업(filter) 을 해야 하기 때문에 대용량 테이블의 경우 성능이 많이 느려진다.

#### 105) 아래 SQL이 순수하게 서브쿼리문으로 수행되게 하고 서브쿼리부터 수행되게 하시오!

```
select /*+ gather_plan_statistics */ ename, sal
  from emp
  where deptno in ( select /*+ no_unnest push_subq */ deptno
                    from dept);
```

#### 106) 위 SQL 의 실행계획이 메인쿼리부터 수행되게 하시오!

```
select /*+ gather_plan_statistics */ ename, sal
  from emp
  where deptno in ( select /*+ no_unnest no_push_subq */ deptno
                    from dept);
```

#### 107) 위 SQL 이 조인으로 수행되게 하시오! ( hash semi join 으로 수행되게 하시오!)

```
select /*+ gather_plan_statistics */ ename, sal
  from emp
  where deptno in ( select /*+ unnest hash_sj */ deptno
                    from dept);

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

10	Id	Operation	Name	
11				
12	0	SELECT STATEMENT		
13	* 1	<b>HASH JOIN SEMI</b>		
14	2	TABLE ACCESS FULL	EMP	
15	3	TABLE ACCESS FULL	DEPT	
16				

위 실행계획을 보면 emp가 hash table 이 되었다(큰 테이블이 메모리로 올라감).

#### ※ semi : 절반

join 을 했는데 완전한 조인이 아닌 절반의 조인을 한 것. 절반의 조인을 하는 이유는 위 SQL이 조인문장이 아닌 서브쿼리 문장이기 때문이다.

0		SELECT STATEMENT			
* 1		HASH JOIN SEMI			
2		TABLE ACCESS FULL		EMP	
3		VIEW		VW_NSO_1	
4		TABLE ACCESS FULL		DEPT	

선생님 화면처럼 view 로 묶여진 것이 실행계획에 보일수도 있다 (xe sys 인 경우)

### 108) 위 SQL의 실행계획이 dept 가 메모리로 올라가게 하시오! ( dept 가 hash table 이 되게 하시오!)

```
select /*+ gather_plan_statistics */ ename, sal
  from emp
  where deptno in ( select /*+ unnest hash_sj swap_join_inputs(dept) */ deptno
                    from dept);
```

emp 와 dept 가 대용량 테이블이고 위와 같은 SQL 이면 hash semi join 으로 수행하되 작은 테이블이 메모리로 올라가게 힌트를 준 위의 힌트가 가장 모범적인 힌트이다.

### 109) 위 실행계획이 순수하게 nested loop semi join 으로 하시오!

```
select /*+ gather_plan_statistics */ ename, sal
  from emp
  where deptno in ( select /*+ unnest nl_sj */ deptno
                    from dept);
```

hash semi 조인때는 버퍼의 개수가 9개였는데 nested loop semi 조인때는 버퍼의 개수가 15개로 늘어났다. hash semi 조인을 사용하는게 성능상 유리하다

### 110) 관리자인 사원들의 이름을 출력하시오!

```
select /*+ gather_plan_statistics */ ename
  from emp
  where empno in ( select mgr
                  from emp
                  where mgr is not null);
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

실행계획을 보면 둘다 emp 여서 메인쿼리부터 수행했는지 서브쿼리부터 수행했는지 확실히 알기가 어렵다.

### 111) 위 110을 확실히 알 수 있도록 qb\_name 힌트를 써서 다시 실행하시오!

실행계획에서 서브쿼리의 emp 와 메인쿼리의 emp 중 어떤것을 먼저 읽었는지 확인하기 위한 방법

```
select /*+ gather_plan_statistics qb_name(mainquery) */ ename
  from emp
  where empno in ( select /*+ qb_name(subquery) */ mgr
                  from emp);
```

```
select * from table(dbms_xplan.display_cursor(format => 'advanced' ));
```

9	-----						
10	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
11	-----						
12	0	SELECT STATEMENT				6 (100)	
13	* 1	HASH JOIN SEMI		6	84	6 (0)	00:00:01
14	2	TABLE ACCESS FULL	EMP	14	140	3 (0)	00:00:01
15	* 3	TABLE ACCESS FULL	EMP	13	52	3 (0)	00:00:01
16	-----						
17							
18	Query Block Name / Object Alias (identified by operation id):						
19	-----						
20							
21	1	- SEL\$05692069					
22	2	- SEL\$05692069 / EMP@MAINQUERY					
23	3	- SEL\$05692069 / EMP@SUBQUERY					

qb\_name(...) 으로 쿼리 명을 지정해줄 수 있고, format=>'advanced' 힌트로 그 상세내역을 알 수 있다

### ☆ 112) 주사위 한개를 288회 던져서 5 이상의 눈이 나올 확률을 구하시오!

```

with dice as ( select round(dbms_random.value(0.5,6.5)) as d1
               from dual
               connect by level <= 288)
select round(count(*)/288, 2)
       from dice
       where d1 = 5 or d1 = 6;

```

### 113) 아래의 쿼리의 실행계획이 해시세미조인으로 수행되게 하시오!

```

select /*+ gather_plan_statistics */ ename
       from emp
       where empno in ( select mgr
                        from emp);
>>
select /*+ gather_plan_statistics */ ename
       from emp
       where empno in ( select /*+ unnest hash_sj */ mgr
                        from emp);
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));

```

8	-----			
9	Id	Operation	Name	Cost
10	-----			
11	0	SELECT STATEMENT		
12	* 1	HASH JOIN SEMI		
13	2	TABLE ACCESS FULL	EMP	
14	* 3	TABLE ACCESS FULL	EMP	
15	-----			

### 114) 위 SQL의 실행계획을 보면 메인쿼리부터 수행했는지 서브쿼리부터 수행했는지를 확인하기 위해 qb\_name 힌트를 써서 수행하시오!

```

select /*+ qb_name(main) gather_plan_statistics */ ename
       from emp
       where empno in ( select /*+ qb_name(sub) unnest hash_sj */ mgr
                        from emp);
select * from table(dbms_xplan.display_cursor(format=>'advanced'));

```

9	-----							
10	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	
11	-----							
12	0	SELECT STATEMENT				6 (100)		
13	* 1	HASH JOIN SEMI		6	84	6 (0)	00:00:01	
14	2	TABLE ACCESS FULL	EMP	14	140	3 (0)	00:00:01	
15	* 3	TABLE ACCESS FULL	EMP	13	52	3 (0)	00:00:01	
16	-----							
17								
18	Query Block Name / Object Alias (identified by operation id):							
19	-----							
20								
21	1 -	SEL\$526A7031						
22	2 -	SEL\$526A7031 / EMP@MAIN						
23	3 -	SEL\$526A7031 / EMP@SUB						

세미조인은 무조건 메인쿼리부터 수행된다. 그런데 swap\_join\_inputs 를 사용하면 서브쿼리부터 수행되게 할 수 있다.

### 115) 위 SQL이 서브쿼리부터 수행되는 hash semi join 이 되게 하시오!

```

select /*+ qb_name(main) gather_plan_statistics */ ename
       from emp
       where empno in ( select /*+ qb_name(sub) unnest hash_sj swap_join_inputs(emp@sub) */ mgr

```

```

from emp);

select * from table(dbms_xplan.display_cursor(format=>'advanced'));

```

9							
10	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
11							
12	0	SELECT STATEMENT				6 (100)	
13	* 1	HASH JOIN RIGHT SEMI		6	84	6 (0)	00:00:01
14	* 2	TABLE ACCESS FULL	EMP	13	52	3 (0)	00:00:01
15	3	TABLE ACCESS FULL	EMP	14	140	3 (0)	00:00:01
16							
17							
18	Query Block Name / Object Alias (identified by operation id):						
19							
20							
21	1	SEL\$526A7031					
22	2	SEL\$526A7031 / EMP@SUB					
23	3	SEL\$526A7031 / EMP@MAIN					

메인쿼리의 테이블과 서브쿼리의 테이블이 서로 같을때 해쉬 세미조인을 한 경우 서브쿼리의 테이블부터 수행되게 할 경우 qb\_name(sub) hint 를 이용해서 쿼리블럭의 이름을 **sub** 로 주고 **swap\_join\_inpuops (emp@sub)** 을 사용하면 된다. 이 경우에 실행계획이 서브쿼리의 테이블부터 수행되면서 **hash right semi join** 으로 수행된다

### 116) 관리자가 아닌 직원들의 이름을 출력하시오!

```

select /*+ gather_plan_statistics */ ename
from emp
where empno not in ( select mgr
                     from emp
                     where mgr is not null);

```

### 117) 위 실행계획이 서브쿼리로 풀리지 않고 조인으로 풀리게 하시오! (서브쿼리의 테이블과 메인쿼리의 테이블이 서로 대용량이면 hash join 으로 풀리는게 훨씬 성능이 좋다)

```

select /*+ qb_name(main) gather_plan_statistics */ ename
from emp
where empno not in ( select /*+ qb_name(sub) gather_plan_statistics unnest hash_aj */ mgr
                     from emp
                     where mgr is not null);

```

```

select * from table(dbms_xplan.display_cursor(format=>'advanced'));

```

not in 을 사용한 서브쿼리 문장의 성능 발달을 위해서 hash anto join 을 사용하면 된다. 힌트는 unnest hash\_aj 이다

### 118) 위 해쉬조인 실행계획의 조인순서가 서브쿼리부터 수행되도록 하시오!

```

select /*+ qb_name(main) gather_plan_statistics */ ename
from emp
where empno not in ( select /*+ qb_name(sub) gather_plan_statistics unnest hash_aj
                           swap_join_inpuops (emp@sub) */ mgr
                     from emp
                     where mgr is not null);

```

```

select * from table(dbms_xplan.display_cursor(format=>'advanced'));

```

### ※ 정리

서브쿼리 튜닝 방법 box	hint	
1. 순수하게 서브쿼리문으로 수행하는 방법	<b>no_unnest</b>	기본형
1-2 서브쿼리부터 수행	<b>push_subq</b>	1, 1-2, 1-3 같이 사용

1-3 메인쿼리부터 수행	<b>no_push_subq</b>	
2. 서브쿼리를 조인으로 변경하여 수행하는 방법	<b>unnest</b>	기본형
2-1 semi join		조건절 in
- nested loop semi join	<b>nl_sj</b>	
- hash semi join	<b>hash_sj</b>	
- merge semi join	<b>merge_sj</b>	
2-2 anti join		조건절 not in
- nested loop anti join	<b>nl_aj</b>	
- hash anti join	<b>hash_aj</b>	
- merge anti join	<b>merge_aj</b>	
3. qb_name(table) hint		

## ■ 4. 데이터 분석함수를 이용한 튜닝

이전까지는 SQL 튜일할 때 힌트를 이용했지만 지금부터는 완전히 다른 SQL 로 변경

### 119) 아래의 SQL 을 튜이하시오!

```
select deptno, sum(sal)
  from emp
  group by deptno
 union all
 select null as deptno, sum(sal)
  from emp;
>>tuning...
select /*+ gather_plan_statistics */ deptno, sum(sal)
  from emp
  group by rollup(deptno);
```

### 120) 아래의 SQL 을 튜닝하시오!

```
select deptno, null as job, sum(sal)
  from emp
  group by deptno
 union all
 select null as deptno, job, sum(sal)
  from emp
  group by job
  order by deptno asc, job asc;
>>tuning..
select deptno, job, sum(sal)
  from emp
  group by grouping sets( deptno, job )
  order by deptno asc, job asc;
```

버퍼의 개수가 줄어든다

### 121) 아래의 SQL 을 튜닝하시오!

```
select deptno, null as job, sum(sal)
  from emp
  group by deptno
union all
select null as deptno, job, sum(sal)
  from emp
  group by job
union all
select null as deptno, null as job, sum(sal)
  from emp
order by deptno asc, job asc;
버퍼 18개
>>tuning..
select /*+ gather_plan_statistics */ deptno, job, sum(sal)
  from emp
  group by grouping sets( deptno, job, () )
  order by deptno asc, job asc;
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
버퍼 9개로 성능향상
```

### 122) 아래의 SQL 을 튜닝하시오!

```
select /*+ gather_plan_statistics */ empno, ename, sal, (select sum(sal)
                                                             from emp s
                                                             where s.empno < m.empno) as 누적치
  from emp m
  order by empno asc;
>>tuning..
select /*+ gather_plan_statistics */ empno, ename, sal, sum(sal) over(order by empno asc) 누적치
  from emp
튜닝전은 emp 테이블을 2번 액세스했지만 튜닝후는 한번만 액세스하였다.
```

### 123) 아래의 SQL을 튜닝하시오!

```
select /*+ gather_plan_statistics */ deptno, empno, ename, sal, (select sum(sal)
                                                                    from emp s
                                                                    where s.empno <= m.empno
                                                                    and s.deptno=m.deptno ) 누적치
  from emp m
  order by deptno asc, empno asc;
>>tuning..
select /*+ gather_plan_statistics */ deptno, empno, ename, sal,
      sum(sal) over(partition by deptno
                    order by empno asc) 누적치
  from emp;
```

**124) 아래의 SQL 을 튜닝하시오! SQLP 주관식 단골문제 3문제 중 하나**

```
create index emp_ename on emp(ename);
```

```
select /*+ gather_plan_statistics */ ename, sal, job
```

```
from emp
```

```
where ename like '%EN%' or ename like '%IN%';
```

ename 에 아무리 인덱스가 있더라도 like 연산자 사용시 와일드카드(%) 가 앞에 있으면 인덱스를 액세스 하지 못하고 full table scan 한다. 버퍼 7개

```
>>tuning..
```

```
>>1 먼저 이름에 EN 또는 IN 이 포함되어져있는 사원의 rowid 를 emp_ename 인덱스를 통해 알아낸다.
```

```
alter table emp modify ename constraint emp_ename_nn not null;
```

```
select /*+ gather_plan_statistics index_ffs(emp emp_ename) */ rowid
```

```
from emp
```

```
where ename like '%EN%' or ename like '%IN%';
```

```
>>2 알아낸 rowid 를 통해서 테이블에서 해당 데이터를 검색하는데 nested loop join 으로 검색한다
```

```
select /*+ leading(v e) use_nl(e) */ e.ename, e.sal, e.job
```

```
from emp e,
```

```
(select /*+ gather_plan_statistics index_ffs(emp emp_ename) */ rowid as rn
```

```
from emp
```

```
where ename like '%EN%' or ename like '%IN%') v
```

```
where e.rowid = v.rn;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

10	-----								
11	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
12	-----								
13	0	SELECT STATEMENT		1		3	00:00:00.01	7	
14	* 1	TABLE ACCESS FULL	EMP	1	3	3	00:00:00.01	7	
15	-----								

from 절의 서브쿼리인 인라인뷰에서 emp\_ename 의 인덱스를 빠르게 스캔해서 rowid 3개를 알아낸 다음, emp table과 조인하는 SQL 이다. 그런데 실행계획을 보면 full table emp 로 풀리면서 from 절의 서브쿼리인 in line view 를 해체해버렸다. 이를 인라인뷰를 해체하지 못하도록 no\_merge 힌트를 사용해보자.

```
>>최종 튜닝
```

```
select /*+ leading(v e) use_nl(e) no_merge(v) */ e.ename, e.sal, e.job
```

```
from emp e,
```

```
(select /*+ gather_plan_statistics index_ffs(emp emp_ename) */ rowid as rn
```

```
from emp
```

```
where ename like '%EN%' or ename like '%IN%') v
```

```
where e.rowid = v.rn;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

10	-----								
11	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
12	-----								
13	0	SELECT STATEMENT		1		3	00:00:00.01	5	
14	1	NESTED LOOPS		1	3	3	00:00:00.01	5	
15	2	VIEW		1	3	3	00:00:00.01	4	
16	* 3	INDEX FAST FULL SCAN	EMP_ENAME	1	3	3	00:00:00.01	4	
17	4	TABLE ACCESS BY USER ROWID	EMP	3	1	3	00:00:00.01	1	
18	-----								

튜닝전보다 버퍼의 개수가 더 줄었다.



125) 우리반 테이블에서 성과 이름에 '정', '준' 자를 포함하고 있는 학생들의 이름, 나이를 출력하시오!

```
select ename, age
```

```
from emp12
```

```
where ename like '%정%' or ename like '%준%';
```

8	-----							
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
10	-----							
11	0	SELECT STATEMENT		1			9  00:00:00.01	6
12	* 1	TABLE ACCESS FULL	EMP12	1	3		9  00:00:00.01	6
13	-----							

```
>>tuning...
```

```
>>1 사전준비
```

```
create index emp12_ename on emp12(ename);
```

```
alter table emp12 modify ename constraint emp12_ename_nn not null;
```

```
select /*+ gather_plan_statistics ffs(emp emp_ename) */ rowid
```

```
from emp12
```

```
where ename like '%정%' or ename like '%준%';
```

```
>>2 합치기
```

```
select /*+ gather_plan_statistics no_merge(v) leading(v e) use_nl(e) */ e.ename, e.age
```

```
from emp12 e,
```

```
( select /*+ gather_plan_statistics ffs(emp12 emp12_ename) */ rowid as rn
```

```
from emp12
```

```
where ename like '%정%' or ename like '%준%') v
```

```
where e.rowid = v.rn;
```

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

11	-----							
12	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
13	-----							
14	0	SELECT STATEMENT		1			9  00:00:00.01	2
15	1	NESTED LOOPS		1	3		9  00:00:00.01	2
16	2	VIEW		1	3		9  00:00:00.01	1
17	* 3	INDEX FULL SCAN	EMP12_ENAME	1	3		9  00:00:00.01	1
18	4	TABLE ACCESS BY USER ROWID	EMP12	9	1		9  00:00:00.01	1

## ■5. 자동 SQL 튜닝

SQL 튜닝 어드바이저(advisor) ---> 점점 인공지능화 되어가고 있다(참고용)

advisor 에게 악성 SQL 튜닝을 물어보는 방법

카페 sql tuning 게시판을 참고하자

<http://cafe.daum.net/oracleoracle/SfWN/142>

```
-- scott으로 test 테이블 생성
```

```
SQL> connect scott/tiger
```

```
SQL> create table test (n number );
```

```
SQL> declare
```

```
begin
```

```
for i in 1 .. 10000 loop
```

```
insert into test values(i);
```

```
commit;
```

```
end loop;
```

```
end;
```

```
/
```

```
-- 인덱스 생성
SQL> create index test_idx on test(n);
```

```
-- test 테이블 통계정보 분석
SQL> analyze table test estimate statistics;
```

테스트 테이블에 대한 정보를 분석해서 오라클에게 알려주는 명령어

1. 테이블의 건수
2. 테이블의 크기
3. index 는 어디어디에 있는지 등

```
-- NO_INDEX 힌트를 주어 풀테이블 스캔으로 수행되는 SQL확인
```

```
SQL> set autot traceonly explain ---> 건너뛰고
```

```
SQL> select /*+ gather_plan_statistics NO_INDEX(test test_idx) */ * from test where n = 1 ;
```

test 테이블에 n컬럼의 데이터가 1인 데이터를 검색하는데 no\_index hint 를 이용해서 test 테이블의 test\_idx 인덱스를 타지 말고 full table scan 해라 하고 실행한다. 악성 SQL 생성을 위한 hint

```
select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));
```

8	-----								
9	Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	
10	-----								
11	0	SELECT STATEMENT		1		1	00:00:00.01	22	
12	* 1	TABLE ACCESS FULL	TEST	1	1	1	00:00:00.01	22	
13	-----								

악성 SQL 생성확인(table full scan)

```
-- 튜닝 TASK생성 후, SQL Tuning Advisor를 실행
```

```
SQL> connect / as sysdba
```

SQL 튜닝 어드바이저에게 악성 SQL 을 주면서 튜닝해 달라고 부탁하는 명령어

```
SQL> declare
```

```
my_task_name VARCHAR2(30);
```

```
my_sqltext CLOB;
```

```
begin
```

```
my_sqltext := 'select /*+ no_index(test test_idx) */ * -- tuning 할 sql script 붙이는 곳
from test where n = 1';
```

```
my_task_name := DBMS_SQLTUNE.CREATE_TUNING_TASK (
sql_text => my_sqltext,
```

```
user_name => 'SCOTT', -- 유저 이름 변경
```

```
scope => 'COMPREHENSIVE',
```

```
time_limit => 60, -- 시간 제한
```

```
task_name => 'my_sql_tuning_task_1', -- 튜닝 작업 이름
```

```
description => 'Task to tune a query on a specified table' );
```

```
end;
```

```
/
```

위에서 만든 튜닝 작업 'my\_sql\_tuning\_task\_1' 을 실행해라~ ↓ ↓

```
SQL> begin
```

```
DBMS_SQLTUNE.EXECUTE_TUNING_TASK (
```

```
task_name => 'my_sql_tuning_task_1' );
```

```
end;
```

```
/
```

```
-- SQL Tuning Advisor를 통해 얻은 결과(튜닝 레포트)를 확인하고 SQL Profile을 적용
```

SQL튜닝 분석결과를 확인할 수 있다.

```
SQL> SET LONG 70000
```

```
SQL> SET LONGCHUNKSIZE 1000
```

```
SQL> SET LINESIZE 100
```

위 3개의 SQL은 편집기이고 아래는 레포트보기 SQL 이다

```
SQL> select DBMS_SQLTUNE.REPORT_TUNING_TASK( 'my_sql_tuning_task_1') from DUAL;
```

위 SQL을 실행하면 일부만 나오는데 결과를 복사해서 메모장에 붙여넣으면 다 볼 수 있다

위 악성 SQL 을 그대로 둔 상태에서 그냥 실행계획만 인덱스 스캔하는 실행계획으로 바꿔치기하려면 튜닝 레포트에 SQL profile 적용하는 문장을 수행하면 된다. 혹은 아래 SQL을 적용해 주고 원 SQL 실행계획을 보면 index 로 바뀐것을 볼 수 있다.

```
SQL> DECLARE
my_sqlprofile_name VARCHAR2(30);
BEGIN
my_sqlprofile_name := DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
task_name => 'my_sql_tuning_task_1',
name => 'my_sql_profile' );
END;
/
```

-- 위에서 실행한 악성 SQL의 실행계획을 확인

```
SQL> conn scott/tiger
SQL> set autot traceonly explain
SQL> select /*+ NO_INDEX(test test_idx) */ * from test where n = 1;
```

\* sql profile drop 방법  
exec dbms\_sqltune.drop\_sql\_profile('test\_prof', true)

\* sql tuning task drop 방법  
select task\_name  
from user\_ADVISOR\_TASKS;

exec dbms\_sqltune.drop\_tuning\_task('ts45');

\* sql tuning task drop 프로시저  
set serveroutput on  
declare  
not\_drop EXCEPTION;  
PRAGMA EXCEPTION\_INIT(not\_drop, -27365);

```
cursor c1 is
select task_name
from user_ADVISOR_TASKS;
begin
for empref in c1 loop
dbms_sqltune.drop_tuning_task(task_name=>empref.task_name);
end loop;
commit;
exception
when not_drop then
dbms_output.put_line('not not not');
end;
/
```

☆126) 숫자를 물어보게 하고 숫자를 입력하면 해당 숫자가 정수인지 합성수인지가 출력되게 하시오!

>>1 my answer

undefine p

```

accept p prompt '숫자를 입력하세요~'
with num1 as (select level x
               from dual
               connect by level <= &p),
num2 as (select level y
         from dual
         connect by level <= &p)
select decode(n1.x, &p, '소수', '합성수') as 출력값
  from num1 n1, num2 n2
 where mod(n1.x,n2.y)=0
 group by n1.x
 having count(*)=2
 order by n1.x desc
 fetch first 1 rows only;

```

>>2

```

undefine p_num
accept p_num prompt '숫자를 입력하세요 ~'
with ntable as ( select count( decode ( mod(&p_num,level), 0, 0) ) as num
                 from dual
                 connect by level <=&p_num
                 )
select case when num=2 then '소수입니다' else '합성수 입니다' end
  from ntable;

```