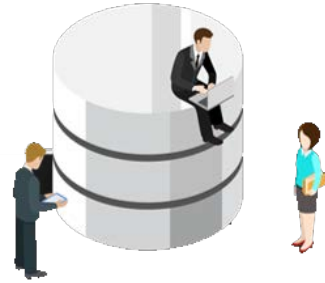


EXEM SEMINAR

exem-academy.com/#seminar



512시간, 2,000명 이상 강의 세미나 진행, 국내 4대 교육기관에 콘텐츠 제휴

엑셈 아카데미 교육 특장점

- 1 실무 사례 기반의 실전 노하우 전수
- 2 전문 기술 도서 저자의 직접 강의
- 3 기술내재화를 위한 사후 온오프 멘토링
- 4 외부소싱이 아닌 전원 엑셈의 스타강사

▼ 2017년 6월, 고객을 찾아가는 엑셈아카데미_원주편 ▼ 엑셈아카데미, 빅데이터/AI/DB/클라우드 세미나 ▼ 2018년 1월 환경관리공단 IT 실무자 빅데이터 분석 및 블록체인 강의 세미나



▼ 2018년 2월 한국산업단지공단 기업사 임원 대상 빅데이터 분석 사례 세미나



▼ 2018년 2월 한국전력공사 인공지능교육



▼ 2018년 3월 에너지밸리기업개발원 주최 광주·전남 지역 대상 클라우드 기반 빅데이터 분석과 활용 사례 발표



▼ 2018년 4월 LS오토모티브 제조 빅데이터 분석 입문·기초·심화 교육



▼ 2018년 1월, 라이나생명 디지털 전략 과정 - 빅데이터 분석 과정



▼ 2018년 1월, 라이나생명 디지털 전략 과정 - 빅데이터 분석 과정



엑셈 온사이트 세미나

4차 산업혁명의 핵심 기술들을 다루는 엑셈 아카데미의 지식 전문 세미나가, 여러분이 있는 곳으로 직접 찾아갑니다. 현재 공공기관, 기업 및 일반, 대학교 등 다양한 사이트에서 엑셈 온사이트 세미나는, 고객 여러분들이 원하는 맞춤형 4차 산업혁명 세미나를 제공합니다. 자세한 문의는 하단의 교육문의 메일을 참조하십시오.

SQL 튜닝 대상 선정방법

1 CHAPTER

 The Start of SQL Tuning Seminar

컨설팅 본부
www.ex-em.com
exem.tistory.com

CONTENTS

STEP. 1

대상 선정의 중요성

- SQL 대상 선정의 중요성
- 잘못된 대상 선정의 예
- 대상 선정 도구

STEP. 2

CASE별 대상 선정 방법

- IO / CPU Top SQL 관련
- Table Full Scan 관련
- Literal SQL 관련
- 배치 프로그램 관련
- 실행계획이 변경 된 SQL 추출
- ASH를 활용한 개선 대상 선정
- Top Table 개념을 활용한 개선 대상 선정

STEP. 1

대상선정의 중요성

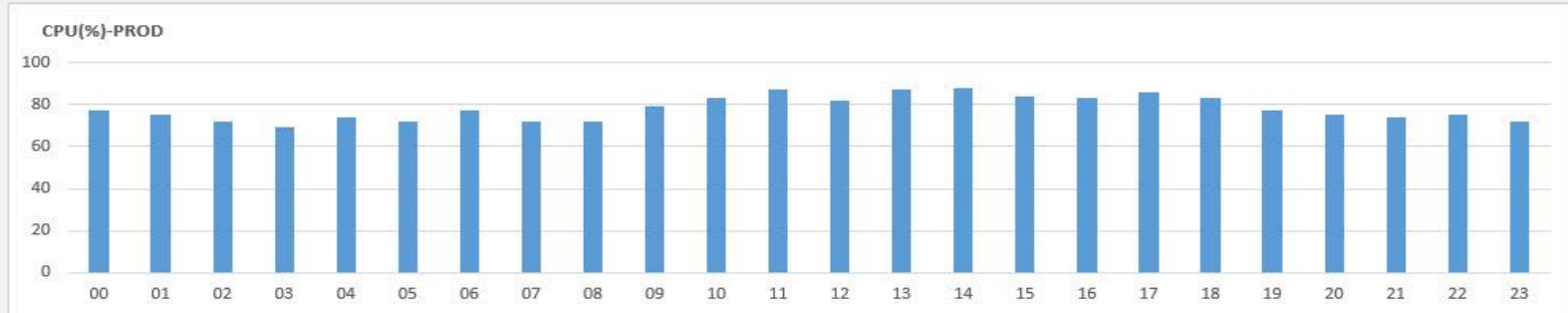


DB 상황에 맞는 대상 선정

- ✓ 특별한 상황에 대처하기 위해 성능 개선 작업을 수행하는 경우 SQL 튜닝 대상 선정은 마땅히 그 요구 사항과 상황에 맞게 선정.
예) CPU, IO, Literal, 특정 업무, 장애 상황 등
- ✓ 일반적인 상황에서의 SQL 튜닝 대상선정 작업은 DB서버 전반의 성능 개선을 목표로 대상선정.

잘못된 SQL 추출 (1)

고객 요구사항 DB 서버의 CPU 사용률이 높다.

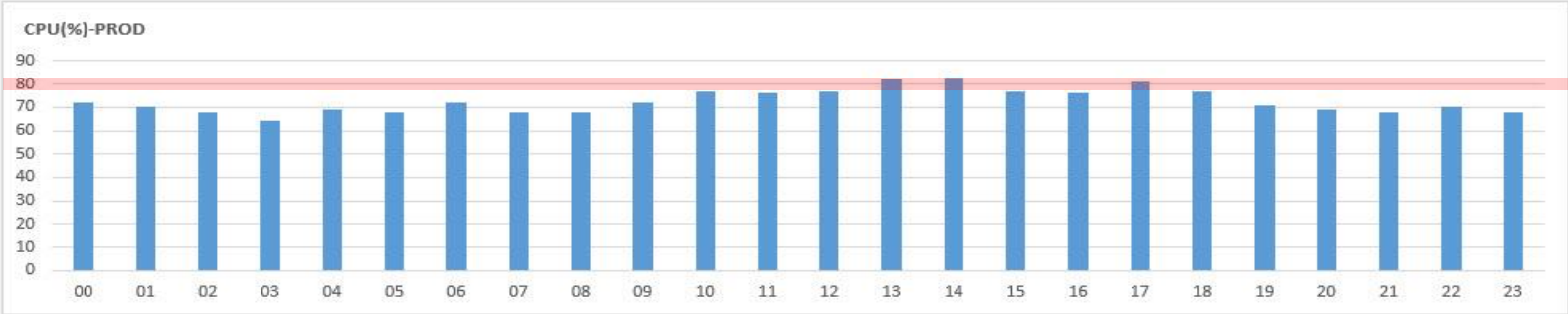


[PROD Database 시간 별 CPU 사용률 추이]

- ✓ Elapsed Time이 높은 SQL 추출?
- ✓ IO가 높은 SQL 추출?
- ✓ TABLE FULL SCAN을 발생 시키는 SQL 추출?
- ✓ DISK IO가 높은 SQL 추출?

잘못된 SQL 추출 (2)

고객 요구사항 DB 서버의 CPU 사용률이 높다.



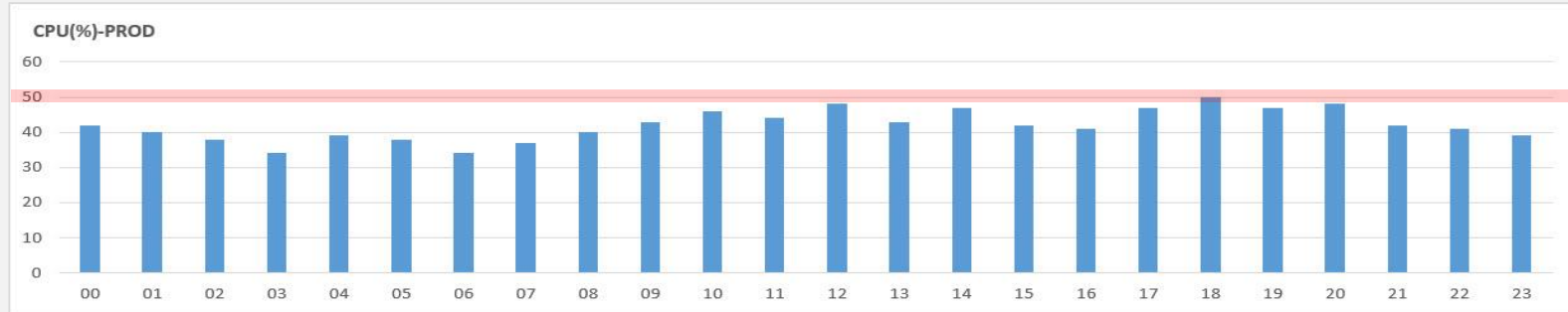
[PROD Database 시간 별 CPU 사용률 추이]

NO	SHEMA NAME	MODULE	SQL ID	EXECUTIONS	BUFFER GETS	DISK I/O	LIO	RATIO CPU(%)
1	MAXMON	PRODUCT_QNT	c6zbza0s68tdj	1	43511	218	43511	0
2	MAXGAUGE	MAXGAUGE@RAC1W253	8vvv6hx92ymmm	1	17	0	17	0
3	MAXGE	MAXGAUGE@RAC1W269	4m8mbu71fgra8	1	52631	381654	52631	2.3
4	MAXGE	SQL*Plus	56qnmkn1xrhnn	1	8298533	8019202	8298533	0.7
5	MAXGE	SQL*Plus	0nt1u6rszy5g3	3	7486704	4371575	2495568	0.6
6	MAXGE	SQL*Plus	1aqwcyb94qk73	2	1114895	1570313	557447.5	0.3
7	MAXGE	OrdersSummary	d7y4tdacc7f3j	2	578881	1116607	289440.5	0.3
8	MAXGE	Customer order summary	0z5myd3av3d56	4	177212	805197	44303	0.4
9	MAXGE	CustomerOrder_rank	7nwf89y3rfbfw	3	560128	1243407	186709.3	0.3
10	MAXGE	SQL*Plus	fc5u7n0r523m	61	6672	9312224	109.4	4

[PROD Database 성능개선 SQL List]

올바른 SQL 추출

고객 요구사항 DB 서버의 CPU 사용률이 높다.



[PROD Database 시간 별 CPU 사용률 추이]

NO	SHEMA NAME	MODULE	SQL ID	EXECUTIONS	BUFFER_GETS	DISK I/O	LIO	RATIO CPU(%)
1	MAXGE	MAIL_E	13t5kwh3tyfhu	17185	1015939	719756	59	15.8
2	MAXGE	SQL*Plus	1gjh8n97gh25w	1	736075624	735365110	736075624	12.7
3	MAXGE	OrdersSummary	g335ufdsfmsb6	5250	719502816	719473435	137048	11.3
4	MAXID		bkjxktgc8afm1	88	436776231	50816941	4963366	10.8
5	MAXGE	OrderInfoCheck	bcv9qynmu1nv9	1844	231157800	38808512	125356	5.3
6	MAXGE	OrderInfoCheck	aztd4470p8vwk	1851	231601037	39885756	125122	5.3
7	MAXGE	OrderInfoCheck	bcg7084jc4um6	1852	231730386	40016138	125124	5.3
8	MAXGE	PROD_UP	ckacc60n500qa	1085	1622145903	145128302	149443	4.7
9	MAXGE	SQL*Plus	fc5u7n0r523m	61	6672	9312224	109.4	4
10	MAXOE		8sr7bd9bp4j9u	258725	52583034	3210480	203	3

[PROD Database 성능개선 SQL List]

■ 튜닝 대상 선정을 위한 정보 및 활용 도구

- 업무 담당자의 요구사항 및 조언을 활용하는 방법
- Oracle 의 Dictionary View를 활용하는 방법
- 성능 관리 솔루션을 활용 하는 방법

STEP.2

SQL튜닝 대상 선정방법

— CASE LIST

1. IO / CPU Top SQL 관련 성능 개선 대상선정
2. Table Full Scan 관련 성능 개선 대상선정
3. Literal SQL 관련 성능 개선 대상선정
4. 배치 프로그램 관련 성능 개선 대상선정
5. 실행계획이 변경 된 SQL 추출
6. ASH를 활용한 성능 개선 대상 선정
7. Top Table 개념을 활용한 성능 개선 대상 선정

IO/ CPU Top SQL 추출

DBA_HIST_SQLSTAT

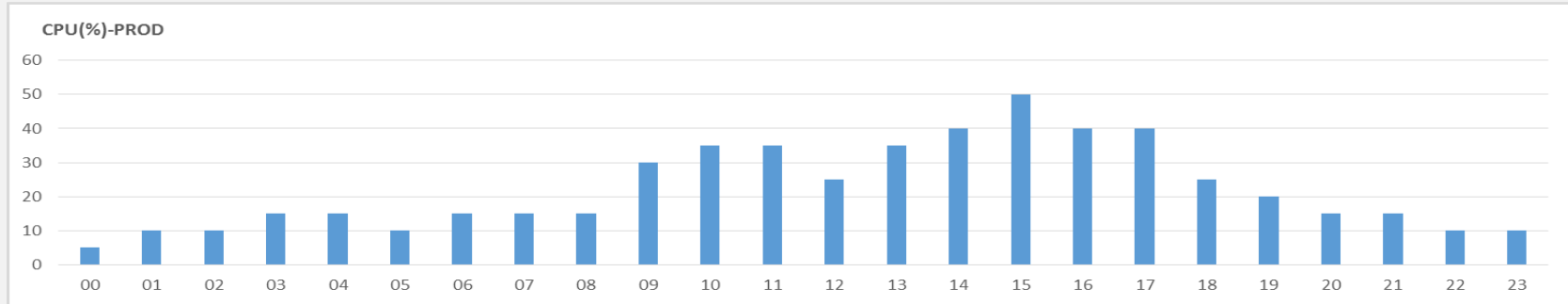
- BUFFER_GETS
- CPU_TIME
- ELAPSED_TIME

[V\$SQLAREA 활용 스크립트]

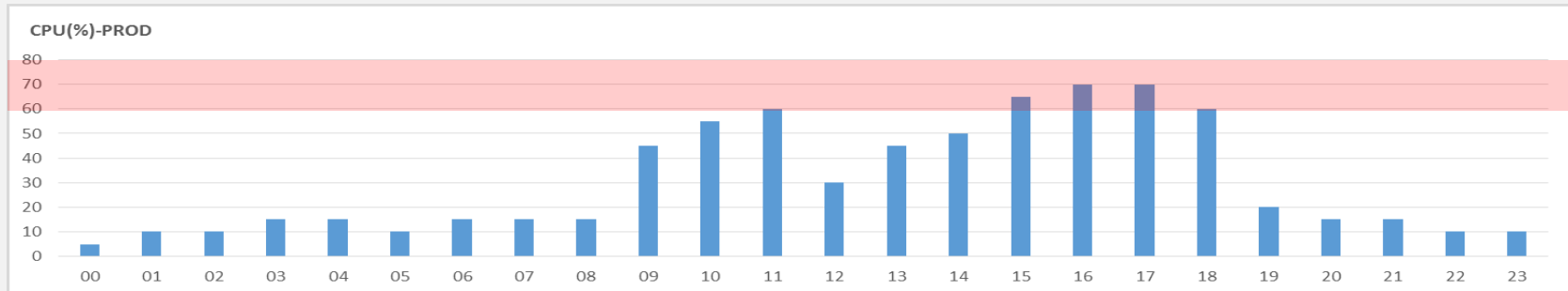
```
SELECT sql_id ,
       schema_name ,
       MODULE ,
       ela_ratio ,
       ela_tot ,
       cpu_ratio ,
       cpu_tot ,
       exec_ratio ,
       exec_tot ,
       lio_ratio ,
       lio_tot ,
       pio_ratio ,
       pio_tot ,
       rows_ratio ,
       rows_tot
FROM (
  SELECT sql_id ,
         parsing_schema_name schema_name ,
         NVL( SUBSTR( b.module , 1 , 15 ) , '-' ) MODULE ,
         ROUND( RATIO_TO_REPORT(SUM(b.elapsed_time_delta)) over() * 100, 1 ) AS ela_ratio ,
         ROUND( SUM( b.elapsed_time_delta ) /1000000 , 0 ) AS ela_tot ,
         ROUND( RATIO_TO_REPORT(SUM(b.cpu_time_delta)) over() * 100, 1 ) AS cpu_ratio ,
         ROUND( SUM( b.cpu_time_delta ) /1000000 , 0 ) AS cpu_tot ,
         ROUND( RATIO_TO_REPORT(SUM(b.executions_delta)) over() * 100, 1 ) AS exec_ratio ,
         SUM( b.executions_delta ) AS exec_tot ,
         ROUND( RATIO_TO_REPORT(SUM(b.buffer_gets_delta)) over() * 100, 1 ) AS lio_ratio ,
         SUM( b.buffer_gets_delta ) AS lio_tot ,
         ROUND( RATIO_TO_REPORT(SUM(b.disk_reads_delta)) over() * 100, 1 ) AS pio_ratio ,
         SUM( b.disk_reads_delta ) AS pio_tot ,
         ROUND( RATIO_TO_REPORT(SUM(b.rows_processed_delta)) over() * 100, 1 ) AS rows_ratio ,
         SUM( b.rows_processed_delta ) AS rows_tot
  FROM dba_hist_snapshot a ,
       dba_hist_sqlstat b
  WHERE a.instance_number=1
        AND a.begin_interval_time >= TO_DATE( :b1 , 'YYYY-MM-DD' )
        AND a.end_interval_time   <= TO_DATE( :b2 , 'YYYY-MM-DD' ) + 0.99999
        AND a.dbid=b.dbid
        AND b.parsing_schema_name NOT IN ( 'SYS' , 'SYSTEM' , 'SYSMAN' )
        AND a.instance_number = b.instance_number
        AND a.snap_id=b.snap_id
  GROUP BY b.sql_id , b.parsing_schema_name , b.module
  ORDER BY cpu_ratio DESC
)
WHERE rownum<=100 ;
```

성능 비교 - 특정 시점

PROD CPU 사용률 시간 별 추이 (2015년 3월 23일)



PROD CPU 사용률 시간 별 추이 (2015년 3월 24일)



성능 비교 - 3월 23일

PROD 시스템의 Top SQL List (2015년 3월 23일)

NO	SQL_ID	SCHEMA_NAME	MODULE	CPU_RATIO	ELAPSED_AVG	BUFFER_AVG	ROWS_AVG
1	33s8sx2usdx3m	DEV50	JDBC Thin Client	6	0.01	427	1
2	0f4549c0ut7d5	DEV50	JDBC Thin Client	7	0	14	1
3	3m9s8nfcn015d	DEV50	JDBC Thin Client	5	0.06	4780	1
4	3d2qaxm2zn1dk	DEV50	JDBC Thin Client	4	0.01	109	1
5	3mmvd78hpddh1	DEV50	JDBC Thin Client	4	0	7	1
6	fj50nrbr4j4dx	DEV50	JDBC Thin Client	3.4	1.75	127891	1
7	fr45uw5bk2u55	DEV50	JDBC Thin Client	3.1	0	13	1
8	b2w67gpzzmj2p	DEV50	JDBC Thin Client	2.4	596.39	7739659	1
9	g26xc6k7n6ux1	DEV50	JDBC Thin Client	2.3	0	19	0
10	7kr63xf4xzrna	DEV50	JDBC Thin Client	2.2	0.01	12	1
11	21rgqvj97x7z1	DEV50	JDBC Thin Client	2.1	0.01	23	1
12	2wv7nm3d7m729	DEV50	JDBC Thin Client	1.9	0	7	1
13	2p3uktbn0c70n	DEV50	JDBC Thin Client	1.2	0.02	13	1
14	70kd44m33xf0h	DEV50	DBMS_SCHEDULER	1.2	13.38	38646	1
15	8v134d96tqnkv	DEV50	JDBC Thin Client	1.2	0	7	1

성능 비교 - 3월 24일

PROD 시스템의 Top SQL List (2015년 3월 24일)

NO	SQL_ID	SCHEMA_NAME	MODULE	CPU_RATIO	ELAPSED_AVG	BUFFER_AVG	ROWS_AVG
1	3abbdrasv4g09	PROD	JDBC Thin Client	23	2.3	127891	35
2	2cfwg24yavw9t	PROD	JDBC Thin Client	17	3.5	127891	22
3	33s8sx2usdx3m	DEV50	JDBC Thin Client	6	0.01	427	1
4	0f4549c0ut7d5	DEV50	JDBC Thin Client	7	0	14	1
5	3m9s8nfcn015d	DEV50	JDBC Thin Client	5	0.06	4780	1
6	3d2qaxm2zn1dk	DEV50	JDBC Thin Client	4	0.01	109	1
7	3mmvd78hpddh1	DEV50	JDBC Thin Client	4	0	7	1
8	fj50rnbr4j4dx	DEV50	JDBC Thin Client	3.4	1.75	127891	1
9	fr45uw5bk2u55	DEV50	JDBC Thin Client	3.1	0	13	0
10	b2w67gpzzmj2p	DEV50	JDBC Thin Client	2.4	596.39	7739659	1

- ✓ 3월 23일에 비해 3월 24일 CPU 사용률이 높은 것은, PROD로 수행된 Top SQL1,2가 과거 시점에는 수행되지 않은 것 확인 가능.
- ✓ 새로 추가된 프로그램의 수행으로 인해 CPU 사용률이 증가하였으며, 해당 SQL을 튜닝대상으로 선정하여 개선 여지가 있는지 점검.

■ TABLE FULL SCAN 발생 SQL 추출 경우

- 신규 테이블에 인덱스 구성이 제대로 되지 않아 발생하는 FTS
- 오픈 시점에는 테이블 사이즈가 작아 성능 문제가 되지 않았지만, 데이터가 많아져 성능 문제를 유발하는 FTS

TABEL FULL SCAN 관련 성능 개선 대상 선정

TABLE FULL SCAN 추출

V\$SQLAREA

V\$SQL_PLAN

- OPERATION
- OPTION

[Full Table Scan으로 수행된 SQL List 추출 스크립트]

```
SELECT ROWNUM cnt , t1.*
FROM (
    SELECT t1.parsing_schema_name ,
           t1.module ,
           t1.sql_id ,
           t1.hash_value ,
           t1.substr_sqltext ,
           t1.executions ,
           t1.buffer_gets ,
           t1.disk_reads ,
           t1.rows_processed ,
           t1.lio ,
           t1.elapsed_sec ,
           t1.cpu_sec ,
           ROUND( t1.cpu_time /t1.cpu_time_total*100 , 1 ) ratio_cpu ,
           ROUND( t1.elapsed_time /elapsed_time_total * 100 , 1 ) ratio_elapsed
    FROM (
        SELECT s.parsing_schema_name ,
               s.module ,
               s.sql_id ,
               s.hash_value ,
               s.address ,
               SUBSTR( s.sql_text , 1 , 100 ) substr_sqltext ,
               s.executions ,
               s.buffer_gets ,
               s.disk_reads ,
               s.rows_processed ,
               s.cpu_time ,
               s.elapsed_time ,
               ROUND( s.buffer_gets / s.executions ) , 1 ) lio ,
               ROUND( s.elapsed_time / s.executions ) /1000000 , 1 ) elapsed_sec ,
               ROUND( s.cpu_time/ s.executions ) /1000000 , 1 ) cpu_sec ,
               SUM( s.cpu_time ) over( ) cpu_time_total ,
               SUM( s.elapsed_time ) over( ) elapsed_time_total
        FROM v$sqlarea s ) t1 ,
    (SELECT DISTINCT hash_value , address
     FROM v$sql_plan
     WHERE operation = 'TABLE ACCESS' AND options = 'FULL' ) x
WHERE t1.executions > 0
AND x.hash_value = t1.hash_value
AND x.address = t1.address
AND t1.parsing_schema_name NOT IN ( 'SYS' , 'SYSTEM' , 'SYSMAN' )
ORDER BY ratio_cpu DESC
) t1
```

TABEL FULL SCAN 관련 성능 개선 대상 선정

성능 비교 – Small Table

Small Table TEST

[데이터 량이 적은 경우]

TABLE NAME: BIG_TAB

COLUMN_NAME	DATA_TYPE	LEN	SCAL	N	DISTINCT	DENSITY	NUM_NULLS	BUCKET	SAMPLE_SIZE	LAST_ANAL
C4	NUMBER	22		Y	5000	0.000200000	0	1	5000	2015-03-19
C5	VARCHAR2	2		Y	26	0.038461538	0	1	5000	2015-03-19
C6	NUMBER	22		Y	5000	0.000200000	0	1	5000	2015-03-19

[SQL 및 실행 계획]

```
SELECT *  
FROM   big_tab  
WHERE  c4 = 100
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		1	00:00:00.01	14
* 1	TABLE ACCESS FULL	BIG_TAB	1	1	1	00:00:00.01	14

Predicate Information (identified by operation id):

1 - filter("C4"=100)

TABEL FULL SCAN 관련 성능 개선 대상 선정

성능 비교 – Big Table

Big Table TEST

[데이터 량이 증가 한 경우]

COLUMN_NAME	DATA_TYPE	LEN	SCAL	N	DISTINCT	DENSITY	NUM_NULLS	BUCKET	SAMPLE_SIZE	LAST_ANAL
C4	NUMBER	22		Y	10000000	0.000000100	0	1	10000000	2015-03-19
C5	VARCHAR2	2		Y	26	0.038461538	0	1	10000000	2015-03-19
C6	NUMBER	22		Y	10000000	0.000000100	0	1	10000000	2015-03-19

[SQL 및 실행 계획]

```
SELECT *  
FROM   big_tab  
WHERE  c4 = 100
```

Id	Operation	Name	Starts	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1	1	00:00:04.42	25986	25983
* 1	TABLE ACCESS FULL	BIG_TAB	1	1	00:00:04.42	25986	25983

Predicate Information (identified by operation id):

1 - filter("C4"=100)

TABEL FULL SCAN 관련 성능 개선 대상 선정

성능 비교 - Index 생성 후

INDEX 생성 후 TEST

[인덱스 생성 후]

COLUMN_NAME	DATA_TYPE	LEN	SCAL	N	DISTINCT	DENSITY	NUM_NULLS	BUCKET	SAMPLE_SIZE	LAST_ANAL
C4	NUMBER	22		Y	10000000	0.000000100	0	1	10000000	2015-03-19
C5	VARCHAR2	2		Y	26	0.038461538	0	1	10000000	2015-03-19
C6	NUMBER	22		Y	10000000	0.000000100	0	1	10000000	2015-03-19

INDEX_NAME	TYPE	U	COLUMN	LIST
BIG_TAB_IDX_01	NORM	N	C4	---> C4 컬럼에 인덱스 생성

[SQL 및 실행 계획]

```
SELECT *
FROM   big_tab
WHERE  c4 = 100
```

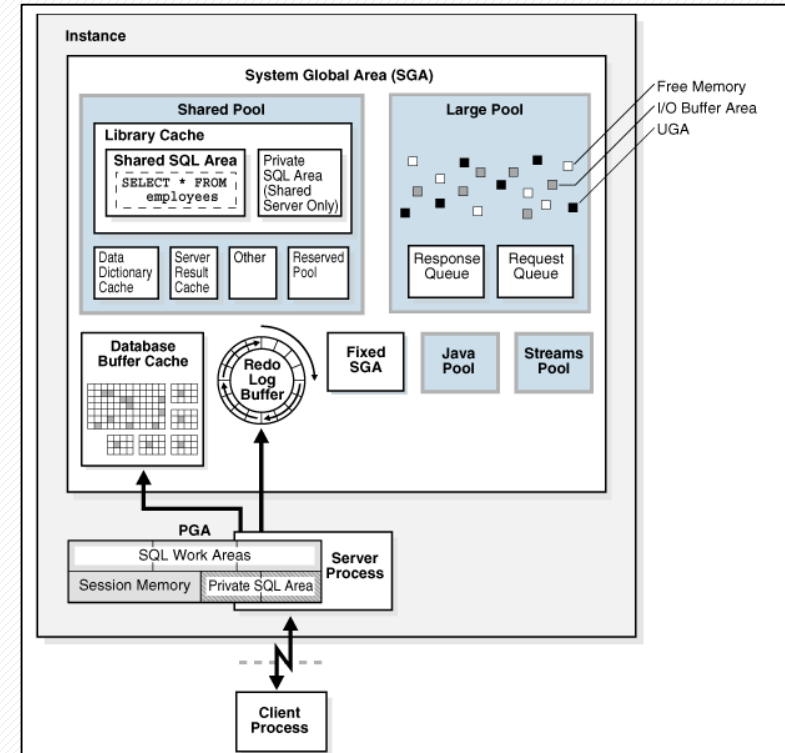
Id	Operation	Name	Starts	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1	1	00:00:00.01	4
1	TABLE ACCESS BY INDEX ROWID	BIG_TAB	1	1	00:00:00.01	4
* 2	INDEX RANGE SCAN	BIG_TAB_IDX_01	1	1	00:00:00.01	3

Predicate Information (identified by operation id):

2 - access("C4"=100)

Literal SQL이란?

- **SELECT * FROM EMP WHERE EMPNO = 1234**
- **SELECT * FROM EMP WHERE EMPNO = 1235**
- **SELECT * FROM EMP WHERE EMPNO = :B1**



Literal SQL 추출

V\$SQLAREA

- FORCE_MATCHING_SIGNATURE
- EXACT_MATCHING_SIGNATURE

[Literal SQL 추출 스크립트]

```
SELECT ROWNUM rno , t1.*
FROM (
    SELECT MAX( substr_sqltext ) sql_text ,
           MAX( parsing_schema_name ) parsing_schema_name ,
           MAX( MODULE ) MODULE ,
           MAX( s.sql_id ) sql_id ,
           COUNT( s.exact_matching_signature ) literal_sql_cnt ,
           ROUND( SUM( buffer_gets ) /sum( s.executions ) , 2 ) buffer_avg ,
           ROUND( SUM( elapsed_time ) /sum( s.executions ) , 2 ) elapsed_avg ,
           ROUND( SUM( rows_processed ) /sum( s.executions ) , 2 ) rows_processed ,
           SUM( s.executions ) executions ,
           ROUND( SUM( cpu_time ) /max( cpu_time_total ) *100 , 2 ) ratio_cpu ,
           ROUND( SUM( elapsed_time ) /max( elapsed_time_total ) *100 , 2 ) elapsed_cpu ,
           COUNT( DISTINCT s.plan_hash_value ) plan_cnt
    FROM (
        SELECT s.parsing_schema_name ,
               s.module ,
               s.sql_id ,
               s.hash_value ,
               s.plan_hash_value ,
               s.address ,
               SUBSTR( s.sql_text , 1 , 100 ) substr_sqltext ,
               s.executions ,
               s.buffer_gets ,
               s.disk_reads ,
               s.rows_processed ,
               s.cpu_time ,
               s.elapsed_time ,
               s.force_matching_signature ,
               s.exact_matching_signature ,
               ROUND( s.buffer_gets / s.executions ) , 1 ) lio ,
               ROUND( s.elapsed_time / s.executions ) /1000000 , 1 ) elapsed_sec ,
               ROUND( s.cpu_time / s.executions ) /1000000 , 1 ) cpu_sec ,
               SUM( s.cpu_time ) over( ) cpu_time_total ,
               SUM( s.elapsed_time ) over( ) elapsed_time_total
        FROM v$sqlarea s ) s
    WHERE s.executions > 0
    AND s.force_matching_signature <> exact_matching_signature
    AND s.parsing_schema_name NOT IN ( 'SYS' , 'SYSTEM' , 'SYSMAN' )
    GROUP BY s.force_matching_signature
    HAVING COUNT( s.exact_matching_signature ) >= 2
    ORDER BY ratio_cpu DESC
) t1
WHERE ROWNUM <= 100
```

SQL LIST 비교 - Before

Literal SQL을 SUM 하지 않은 경우

NO	SCHEMA_NAME	MODULE	SQL_ID	SQL Text	EXECUTIONS	BUFFER_AVG	RATIO_CPU	ELAPSED_AVG
1	DEV50	JDBC Thin Client	2nszajb0qbyvp	select ename, sal from emp where empno = 1	17	1521	0.02	0.3
2	DEV50	JDBC Thin Client	3m9s8nfcn015d	select ename, sal from emp where empno = 2	12	1921	0.02	0.2
3	DEV50	JDBC Thin Client	13t5kwh3tyfhu	select ename, sal from emp where empno = 3	11	1164	0.02	0.2
4	DEV50	JDBC Thin Client	1gjh8n97gh25w	select ename, sal from emp where empno = 4	13	1362	0.02	0.1
5	DEV50	JDBC Thin Client	g335ufdsfmsb6	select ename, sal from emp where empno = 5	14	1269	0.02	0.1
6	DEV50	JDBC Thin Client	bkjxktgc8afm1	select ename, sal from emp where empno = 6	9	2361	0.02	0.1
7	DEV50	JDBC Thin Client	bcv9qynmu1nv9	select ename, sal from emp where empno = 7	8	2351	0.02	0.1
8	DEV50	JDBC Thin Client	aztd4470p8vbk	select ename, sal from emp where empno = 8	7	2457	0.02	0.1
9	DEV50	JDBC Thin Client	bcg7084jc4um6	select ename, sal from emp where empno = 9	9	1122	0.02	0.1
10	DEV50	JDBC Thin Client	ckacc60n500qa	select ename, sal from emp where empno = 10	7	1245	0.01	3.1
11	DEV50	JDBC Thin Client	cuvmy6w7h9kc	SELECT ename ,sal..... 생략 detp.deptno = 1	1	23557	0.01	3.3
12	DEV50	JDBC Thin Client	5ph0ypqguwv5	SELECT ename ,sal..... 생략 detp.deptno = 2	2	21557	0.01	3.1
13	DEV50	JDBC Thin Client	baz69hrswsck9	SELECT ename ,sal..... 생략 detp.deptno = 3	1	24357	0.01	3.4
14	DEV50	JDBC Thin Client	8sr7bd9bp4j9u	SELECT ename ,sal..... 생략 detp.deptno = 4	2	22257	0.01	3.5
15	DEV50	JDBC Thin Client	6tzu043n8njug	SELECT ename ,sal..... 생략 detp.deptno = 5	1	22437	0.01	3.4

SQL LIST 비교 - After

Literal SQL을 SUM 한 경우

NO	SCHEMA_NAME	MODULE	SQL_ID	SQL Text	LITERAL_SQL_CNT	EXECUTIONS	RATIO_CPU	ELAPSED_AVG
1	DEV50	JDBC Thin Client	gm9zparmjwn78	select ename, sal from emp where empno = 2	123	123946	20	0.3
2	DEV50	JDBC Thin Client	96ub5w77tu2ac	SELECT ename ,sal..... 생략 detp.deptno = 1	140	12334	15	3.4
3	DEV50	JDBC Thin Client	c6zbza0s68tdj	SELECT (SELECT COUNT(alert_resou	39	1692	10	12
4	DEV50	JDBC Thin Client	8vvw6hx92ymmm	UPDATE MGMT_CURRENT_METRICS SET 1	2	203	0.02	404.82
5	DEV50	JDBC Thin Client	4m8mbu71fgra8	(SELECT (SELECT instance_name	2	20	0.02	4397.85
6	DEV50	JDBC Thin Client	56qnmkn1xrhnn	SELECT event_id , event_name ,	2	11	0.02	5047.45
7	DEV50	JDBC Thin Client	0nt1u6rszy5g3	SELECT * FROM NLS_SESSION_PARAMETERS	2	751	0.02	125.33
8	DEV50	JDBC Thin Client	1aqwcyb94qk73	select distinct(value) from APM_STRING_DATA	2	23	0	5869.22
9	DEV50	JDBC Thin Client	d7y4tdacc7f3j	select SYS_CONTEXT('USERENV','SESSION_USER	2	187	0	29.98
10	DEV50	JDBC Thin Client	0z5myd3av3d56	UPDATE APM_USER_LIST SET LAST_LOGON_TIME	2	43	0	492.53
11	DEV50	JDBC Thin Client	7nwf89y3rfbfw	/* SELECT os_id, case when unit = " then	2	11	0	901.09
12	DEV50	JDBC Thin Client	fc5u7n0r523m	SELECT NVL(AVG((VALUE*1000.0)/DURATION)	2	5	0	13367.8

- ✓ Literal SQL의 Summary 결과를 보면, 상위 3개 SQL이 CPU 사용률 대부분을 차지하는 것을 확인 가능.
- ✓ Literal SQL이 많이 수행되는 시스템에서는 Literal SQL 에 대한 별도의 처리를 하여 대상 추출 스크립트를 수행해야 정확한 대상이 조회 됨.

특정 프로시저 내에서 수행한 SQL

V\$SQLAREA

- PROGRAM_ID

DBA_OBJECTS

- OBJECT_ID
- OBJECT_NAME

[특정 Program 내에서 수행된 SQL 추출 스크립트]

```
SELECT  o.object_name,
        s.parsing_schema_name AS schema,
        s.module,
        s.sql_id,
        s.hash_value,
        substr(s.sql_text,1,100) as sqltext,
        s.executions,
        s.buffer_gets,
        s.disk_reads,
        round(s.rows_processed/s.executions,1) as "Rows" ,
        round(s.buffer_gets/s.executions,1) lio,
        round(s.elapsed_time/s.executions/1000000,1) elapsed_sec,
        round(s.cpu_time/s.executions/1000000,1) cpu_sec,
        round(s.elapsed_time/1000000,1) as elapsed_time
FROM    ( SELECT object_id, object_name
          FROM    dba_objects
          WHERE   object_name = :Procedure_Name ) o ,
        v$sqlarea s
WHERE   o.object_id = s.program_id
ORDER BY 14 DESC
```


배치 프로그램 테스트 데이터 생성

[테이블 생성하기]

```
DROP TABLE plsql_t1 purge ;

CREATE TABLE plsql_t1 AS
SELECT LEVEL AS c1 ,
       CHR( 65+mod( LEVEL , 26 ) ) AS c2 ,
       level+99999 AS c3
FROM dual
CONNECT BY LEVEL <= 1000000 ;
```

[인덱스 생성 및 통계정보 수집하기]

```
CREATE INDEX plsql_t1_idx_01 ON plsql_t1( c1 ) ;

EXEC dbms_stats.gather_table_stats(ownname=>'exem', tabname=>'plsql_t1', cascade=>true, estimate_percent=>100) ;
```

테스트 프로시저 수행

[프로시저 생성하기]

```
create or replace procedure plsql_batch_1 as
begin
    delete /*+ BatchTest_plsql_batch_1 */ from plsql_t1 ---> SQL에 식별자 부여
where c2 = 'aa';
    commit;
end;
/

create or replace procedure plsql_batch_2 as
begin
    dbms_application_info.set_module('BatchTest',''); ---> Module Name 설정

    insert /*+ BatchTest_plsql_batch_2 */ into plsql_t1 ---> SQL에 식별자 부여
select c1, 'a', c3
    from plsql_t1
    where c2 = 'A';
    commit;

    update /*+ BatchTest_plsql_batch_2 */ plsql_t1 ---> SQL에 식별자 부여
set c2 = 'aa'
    where c2 = 'a';
    commit;

plsql_batch_1; ---> 데이터 delete
end;
/
```

테스트 프로시저 수행 SQL 추출

```
select t1.module, t1.substr_sqltext, t1.executions, t1.buffer_gets
  from (
    select parsing_schema_name Schema,           --> 1
           module,                               --> 2
           sql_id,                               --> 3
           hash_value,                           --> 4
           substr(sql_text,1,37) substr_sqltext, --> 5
           executions,                           --> 6
           buffer_gets,                          --> 7
           disk_reads,                          --> 8
           rows_processed,                      --> 9
           round(buffer_gets/executions,1) lio,   --> 10
           round(elapsed_time/executions/1000000,1) elapsed_sec, --> 11
           round(cpu_time/executions/1000000,1) cpu_sec --> 12
    from v$sqlarea s
   where s.program_id in ( select object_id
                           from dba_objects
                          where object_name in ( 'PLSQL_BATCH_1', 'PLSQL_BATCH_2' ) )

    order by 7 desc
  ) t1
 where rownum <= 50 ;
```

MODULE	SUBSTR_SQLTEXT	EXECUTIONS	BUFFER_GETS
BatchTest	DELETE /*+ BatchTest_plsql_batch_1 */	1	159206
BatchTest	UPDATE /*+ BatchTest_plsql_batch_2 */	1	105014
BatchTest	INSERT /*+ BatchTest_plsql_batch_2 */	1	10901

✓ DBA_OBJECTS, V\$SQLAREA를 활용 하면 손쉽게 프로시저 SQL 수행 내역을 추출할 수 있다.

실행계획이 변경 된 SQL 추출

DBA_HIST_SQLSTAT

- PLAN_HASH_VALUE

[실행계획의 변경이 의심되는 SQL의 변경 이력 추출 스크립트]

```
WITH sql_temp AS (
    SELECT sql_id ,
           a.plan_hash_value ,
           a.snap_id ,
           SUM( executions_delta ) over( PARTITION BY sql_id ,plan_hash_value ) sum_exec ,
           SUM( buffer_gets_delta ) over( PARTITION BY sql_id ,plan_hash_value ) sum_io ,
           SUM( elapsed_time_delta ) over( PARTITION BY sql_id ,plan_hash_value ) sum_elapsed ,
           MIN( a.snap_id ) over( PARTITION BY sql_id ,plan_hash_value ) first_exec_time ,
           MAX( a.snap_id ) over( PARTITION BY sql_id ,plan_hash_value ) end_exec_time ,
           COUNT( DISTINCT sql_id||plan_hash_value ) over( PARTITION BY sql_id ) plan_count
    FROM dba_hist_sqlstat a ,
         dba_hist_snapshot b
    WHERE a.instance_number=1
          AND a.snap_id=b.snap_id
          AND a.dbid=b.dbid
          AND a.instance_number=b.instance_number
          AND b.begin_interval_time BETWEEN TO_DATE( :from_time , 'yyyy-mm-dd' )
          AND TO_DATE( :to_time , 'yyyy-mm-dd' )
          AND plan_hash_value <> 0
    ORDER BY sql_id ,
           snap_id DESC
)
SELECT sql_id ,
       plan_hash_value ,
       MAX( end_exec_time ) AS end_exec_time ,
       MIN( first_exec_time ) AS first_exec_time ,
       ROUND( MAX( sum_elapsed /1000000 ) /decode( MAX( sum_exec ) , 0 , 1 , MAX( sum_exec ) ) , 3 )
       avg_elapsed ,
       ROUND( MAX( sum_io ) /decode( MAX( sum_exec ) , 0 , 1 , MAX( sum_exec ) ) , 3 ) avg_io ,
       MAX( sum_exec ) total_sql_exec ,
       MAX( sum_io ) total_io ,
       ROUND( MAX( sum_elapsed /1000000 ) , 3 ) total_elapsed ,
       MAX( plan_count ) AS plan_cont
FROM sql_temp
WHERE plan_count > 1
GROUP BY sql_id ,
         plan_hash_value
ORDER BY sql_id ,
         end_exec_time DESC
```

실행계획이 변경 된 SQL TEST

NO	SQL_ID	END_EXEC_TIME(SNAP_ID)	FIRST_EXEC_TIME(SNAP_ID)	AVG_ELAPSE	AVG_IO	TOTAL_SQL_EXEC	TOTAL_IO	PLAN_COUNT
1	01d5n1nm17r2h	2302	2301	64.03	793,565	450	357,104,250	2
	01d5n1nm17r2h	2301	2214	0.14	87	84,885,737	7,385,059,119	
2	179n4hvx27979	2302	2301	112.03	3,748,073	60	224,884,397	2
	179n4hvx27979	2301	2233	0.007	16	8595846	137.533.536	

- ✓ _optim_peek_user_binds, _optimizer_use_feedback등 SQL Plan 변경에 영향을 미칠 수 있는 파라미터 OFF 고려.
- ✓ 통계정보 생성시 SQL Plan 변경으로 인한 장애예방 방안 마련 필요.

■ ASH(Active Session History) 유용성

- 매 초마다 Active Session을 샘플링하여 MMON이 30분 마다 AWR(DBA_HIST_ACTIVE_SESS_HISTORY)에 저장
- 세션 Level의 실시간 모니터링 가능
- 과거 시점 조회 가능, 특정 시간대의 발생한 장애 및 성능 저하 원인 분석 가능
- BLOCKING_SESSION 컬럼을 활용하여 TX,TM LOCK 이외의 HOLD SESSION 분석 가능

ASH를 활용한 스크립트

V\$ACTIVE_SESSION_HISTORY

V\$EVENT_NAME

[최근 가장 많이 수행 된 SQL과 수행 점유율(수행 횟수)]

```
SELECT sql_id ,
       COUNT( * ) ,
       COUNT( * ) *100/sum( COUNT( * ) ) over( ) ratio
FROM   v$active_session_history
WHERE  sample_time >= to_date(:from_time,'yyyymmdd hh24miss')
AND    sample_time <  to_date(:to_time,'yyyymmdd hh24miss')
GROUP  BY sql_id
ORDER  BY COUNT( * ) DESC ;
```

[특정 Session이 가장 많이 수행 한 SQL과 수행 점유율(수행 횟수)]

```
SELECT sql_id ,
       COUNT( * ) ,
       COUNT( * ) *100/sum( COUNT( * ) ) over( ) ratio
FROM   v$active_session_history
WHERE  sample_time >= to_date(:from_time,'yyyymmdd hh24miss')
AND    sample_time <  to_date(:to_time,'yyyymmdd hh24miss')
AND    session_id = :b1
GROUP  BY sql_id
ORDER  BY COUNT( * ) DESC ;
```

[특정 구간 이벤트 별 대기 시간]

```
SELECT NVL(a.event, 'ON CPU') AS event,
       COUNT(*) AS total_wait_time
FROM   v$active_session_history a
WHERE  sample_time >= to_date(:from_time,'yyyymmdd hh24miss')
AND    sample_time <  to_date(:to_time,'yyyymmdd hh24miss')
GROUP  BY a.event
ORDER  BY total_wait_time DESC;
```

[특정 구간 CPU 점유율 순 - TOP SQL]

```
SELECT ash.sql_id ,
       SUM( decode( ash.session_state , 'ON CPU' , 1 , 0 ) ) "CPU" ,
       SUM( decode( ash.session_state , 'WAITING' , 1 , 0 ) ) - SUM( decode( ash.session_state ,
'WAITING' , decode( en.wait_class , 'User I/O' , 1 , 0 ) , 0 ) ) "WAIT" ,
       SUM( decode( ash.session_state , 'WAITING' , decode( en.wait_class , 'User I/O' , 1 , 0 ) , 0 ) )
"IO" ,
       SUM( decode( ash.session_state , 'ON CPU' , 1 , 1 ) ) "TOTAL"
FROM   v$active_session_history ash ,
       v$event_name en
WHERE  sql_id IS NOT NULL
AND    en.event#=ash.event#
AND    ash.sample_time >= to_date(:from_time,'yyyymmdd hh24miss')
AND    ash.sample_time <  to_date(:to_time,'yyyymmdd hh24miss')
GROUP  BY sql_id
ORDER  BY SUM( decode( session_state , 'ON CPU' , 1 , 1 ) ) DESC;
```


ASH를 활용한 스크립트

V\$ACTIVE_SESSION_HISTORY

V\$EVENT_NAME

[특정 구간 CPU 점유율 순 - TOP Session]

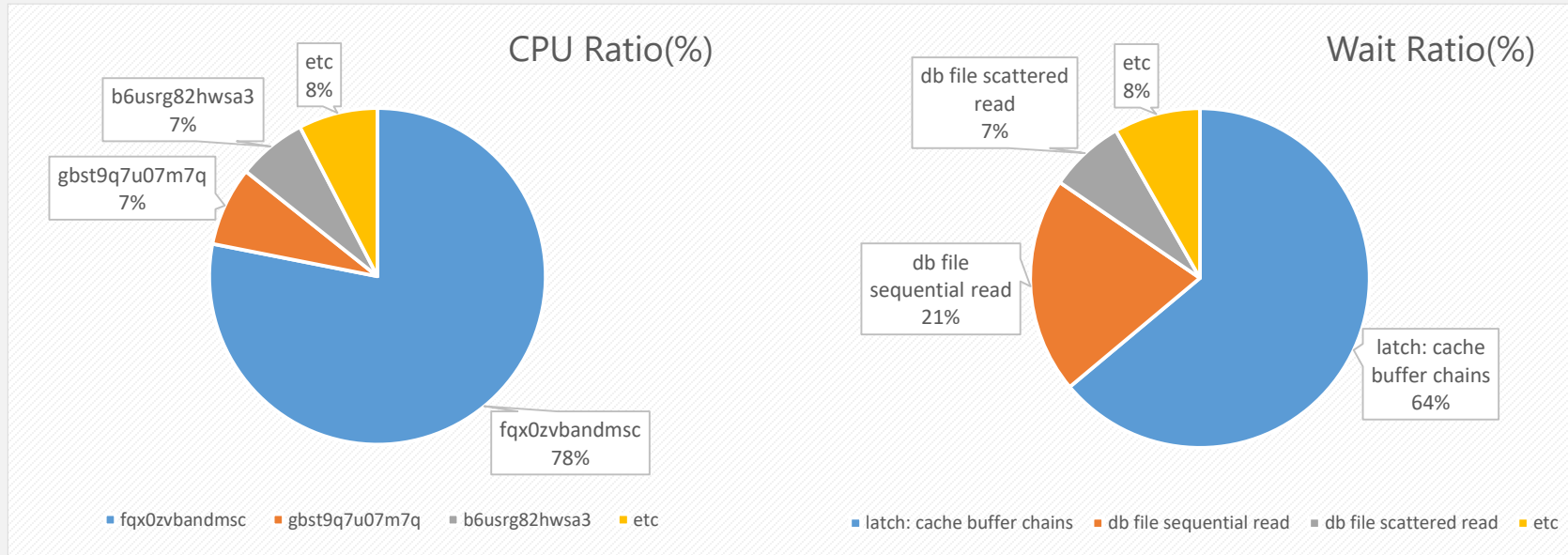
```
SELECT ash.session_id ,
       ash.session_serial# ,
       ash.user_id ,
       ash.program ,
       SUM( decode( ash.session_state , 'ON CPU' , 1 , 0 ) ) "CPU" ,
       SUM( decode( ash.session_state , 'WAITING' , 1 , 0 ) ) - SUM( decode( ash.session_state ,
       'WAITING' , decode( en.wait_class , 'User I/O' , 1 , 0 ) , 0 ) ) "WAITING" ,
       SUM( decode( ash.session_state , 'WAITING' , decode( en.wait_class , 'User I/O' , 1 , 0 ) , 0 ) )
       "IO" ,
       SUM( decode( session_state , 'ON CPU' , 1 , 1 ) ) "TOTAL"
FROM   v$active_session_history ash ,
       v$event_name en
WHERE  en.event# = ash.event#
AND    ash.sample_time >= to_date(:from_time,'yyyymmdd hh24miss')
AND    ash.sample_time <  to_date(:to_time,'yyyymmdd hh24miss')
GROUP BY session_id ,
        user_id ,
        session_serial# ,
        program
ORDER BY SUM( decode( session_state , 'ON CPU' , 1 , 1 ) ) DESC;
```

[특정 구간 수행 이력]

```
SELECT ash.sample_time TIME ,
       ash.session_id ,
       ash.session_serial# ,
       ash.user_id ,
       ash.program ,
       ash.module ,
       ash.client_id ,
       ash.machine ,
       ash.session_state ,
       ash.event ,
       ash.sql_id ,
       ash.blocking_session ,
       ash.current_obj# ,
       ash.current_file# ,
       ash.pga_allocated ,
       ash.temp_space_allocated
FROM   v$active_session_history ash
WHERE  ash.sample_time >= TO_DATE( :from_time , 'yyyymmdd hh24miss' )
AND    ash.sample_time <  TO_DATE( :to_time , 'yyyymmdd hh24miss' )
ORDER BY ash.sample_time DESC;
```

특정 구간의 SQL에 대한 분석 가능

SQL별 CPU(%), Wait Time(%)



- ✓ 최근 시점의 SQL 성능 정보를 파악가능
- ✓ 해당 구간의 성능 저하 SQL파악 가능
- ✓ 최근 시점의 성능 문제 원인이 되는 Wait Event 파악 가능

TOP Table 활용의 유용성

- 업무적인 성격을 파악.
- ACCESS가 많은 테이블 즉, 활용도가 높은 테이블 기반으로 관련 정보를 파악 가능.
- 업무적 중요도 측면에서 우선시 되는 테이블을 인지하여 성능 개선 작업을 진행 가능.
- 성능 개선 대상 테이블의 관련 정보와 인덱스 개수, 통계정보 생성 일자등을 한눈에 파악 가능.

TOP Table을 활용한 정보 추출

DBA_SEGMENTS

DBA_TABLES

DBA_INDEXES

DBA_HIST_SEG_STAT

DBA_TAB_COMMENTS

DBA_PART_TABLES

[Top Table 추출 스크립트]

```
with segs as (
  select /*+ inline */ owner, segment_name, sum(bytes) bytes
  from dba_segments
  where segment_name not like 'bin%'
  and owner not in ('sys', 'system', 'xdb', 'dbsnmp', 'outln', 'orddata')
  and owner not like '%sys%'
  and owner not like 'apex%'
  group by owner, segment_name
), base as (
  select s.owner, t.table_name, round(sum(s.bytes)/1024/1024,1) mb, max(last_analyzed) last_analyzed
  from segs s, dba_tables t
  where s.owner = t.owner
  and s.segment_name = t.table_name
  group by s.owner, t.table_name
), ibase as (
  select /* materialize */ s.owner, i.table_name, count(distinct i.index_name) cnt, round(sum(bytes)/1024/1024,1) mb
  from segs s, dba_indexes i
  where i.index_name = s.segment_name
  and i.owner = s.owner
  group by s.owner, i.table_name
), seg_stat as (
  select /*+ no merge */
    owner, object_name,
    sum(logical_reads_delta) logical_reads_total,
    sum(physical_reads_delta) physical_reads_total,
    max(created) created,
    dba_hist_seg_stat a, dba_hist_snapshot b, dba_objects o
  from
    where 1=1
    and a.snap_id = b.snap_id and a.instance_number = b.instance_number
    and b.begin_interval_time between to_date('20150120' || ' 00:00:00', 'yyyymmdd hh24:mi:ss')
    and to_date('20150206' || ' 23:59:59', 'yyyymmdd hh24:mi:ss')
    and a.obj# = o.object_id
  group by owner, object_name
)
select /*+ leading(s) use_nl(p,c) use_hash(i) */
  s.table_name, s.owner,
  , case when s.mb >= 1024 then round(s.mb/1024,1)||'g'
    when s.mb <= 1024 then s.mb ||'m'
  end tab_size
, ss.logical_reads_total lread
, ss.physical_reads_total pread
, round(( decode( physical_reads_total , 0 , 1 , physical_reads_total )
/decode( logical_reads_total , 0 , 1 , logical_reads_total ) ) *100 , 1 ) lr_pr_rate
, p.partitioning_type part_type
, (select column_name
  from dba_part_key_columns kc
  where kc.name = p.table_name
  and rownum <= 1 ) part_col
, p.partition count part_cnt
, p.subpartitioning_type sub_type
, (select column_name
  from dba_subpart_key_columns skc
  where skc.name = p.table_name
  and rownum <= 1 ) sub_col
, (select count(1) from dba_tab_subpartitions dts
  where dts.table_name = p.table_name
  and dts.table_owner = p.owner) sub_cnt
, i.cnt idx_cnt
, case when i.mb >= 1024 then round(i.mb/1024,1)||'g'
  when i.mb <= 1024 then i.mb ||'m'
end idx_size
, s.last_analyzed, comments
from base s, dba_part_tables p, all_tab_comments c, ibase i, seg_stat ss
where 1=1
and s.mb > 0
and s.table_name = p.table_name(+) and s.owner = p.owner(+)
and s.table_name = c.table_name(+) and s.owner = c.owner(+)
and s.table_name = i.table_name(+) and s.owner = i.owner(+)
and s.owner = ss.owner(+) and s.table_name = ss.object_name(+)
order by to_number(s.mb) desc;
```

TOP Table 개념을 활용한 성능 개선 대상 선정

TOP Table 추출 List

NO	테이블 명	테이블 크기(MB)	Logical reads	Physical reads	LR_PR_RATE(%)	파티션 타입	파티션 키	파티션 개수	인덱스 개수	인덱스 크기(MB)	COMMENTS
1	ET_MBR_RSUS_HIST	50,445	278,364	180,411	64.81	RANGE	OCCUR_DTIME	112	2	8,443	상품평 이력
2	OP_ORD_DTL_INFO	22,578	10,583,721	1,205,739	11.39	RANGE	ORD_DTIME	3,338	5	20,163	주문 내역 정보
3	PR_GOODS_BASE	12,409	14,478,233	215,092	1.49	LIST	ENTR_CNTR_CD	28	7	20,095	상품 기본 정보
4	MST_PR_REQUEST	7,765	2,457,661	70,925	2.89	-	-	-	4	2,022	상품 요청 마스터
5	ET_MBR_BASE	5,553	892,337	12,443	1.39	-	-	-	4	1,392	고객 분석
6	CS_CCN_INFO	5,109	379,760	18,229	4.80	-	-	-	3	842	고객 상담 정보
7	CC_PROM_INFO	4,322	553,320	12,288	2.22	-	-	-	5	1,908	상품 프로모션 정보
8	ET_MBR_MILG_RSUS	3,812	1,964,206	189,052	9.62	-	-	-	4	961	일반 상품평
9	CS_TELC_INFO	3,011	221,098	5,012	2.27	-	-	-	4	703	통화 내역 정보
10	PR_ITEM_BASE	2,852	389,551	7,634	1.96	-	-	-	3	420	단품 기본 정보

ET_MBR_RSUS_HIST 분석

- ✓ ET_MBR_RSUS_HIST 테이블 Comments 컬럼 정보가 『상품평 이력』인 것으로 보아 어떠한 정보를 저장해 두는 이력 테이블임을 유추 가능
- ✓ 대부분의 이력 테이블의 성격은 Online 서비스에서는 자주 Access되지는 않지만 해당 정보를 저장해 둔다는 의미가 크며, 주로 Batch 작업에 의해서 Access 되는 경우가 많으며 위표 내용 중 해당 테이블의 Access 빈도를 유추할 수 있는 Logical reads의 결과 값으로 유추 가능
- ✓ Logical reads 대비 Physical reads의 양이 차지하는 비율이 60% 이상이고 해당 테이블이 Range-List 파티션 테이블임을 감안했을 때, Batch 작업에 의한 Table Full Scan 등의 작업으로 Physical reads의 양이 상대적으로 비중이 큰 것으로 짐작 가능
- ✓ ET_MBR_RSUS_HIST 테이블의 인덱스 개수는 2개에 불과한 것으로 보아 Online 서비스에서 제공해야 하는 데이터의 성격 유추 가능

SQL 튜닝을 위한 정보 분석방법

2
CHAPTER

 The Start of SQL Tuning Seminar

컨설팅 본부
www.ex-em.com
exem.tistory.com

CONTENTS

STEP. 1

SQL의 이해

- SQL Tuning 접근 방법

STEP. 2

SQL Graph

- SQL Graph란?
- SQL Graph의 접근 순서

STEP. 3

다양한 SQL Trace

- 10046 EVENT
- DBMS_XPLAN
- DBMS_MONITOR
- DBMS_SQLTUNE

STEP. 4

Object 정보 분석 및 활용

- TABLE 구성정보 확인
- INDEX 구성정보 확인

STEP. 5

업무정보 분석 및 활용

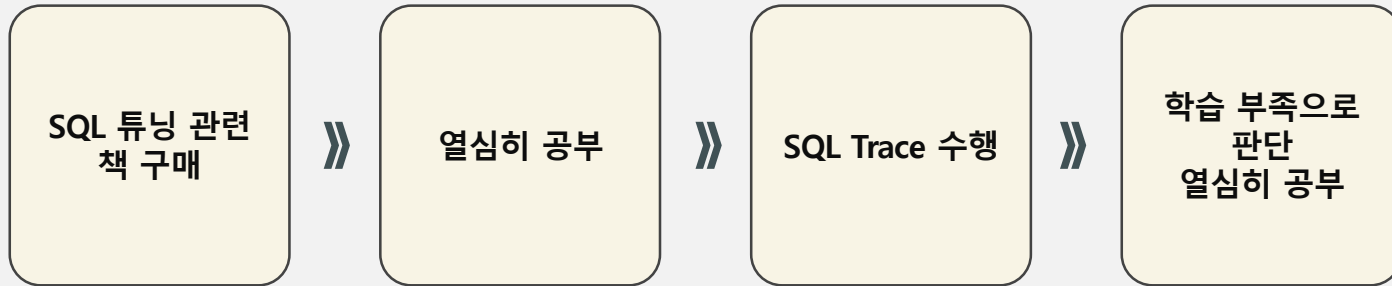
- Top Table
- Access Pattern

STEP. 1

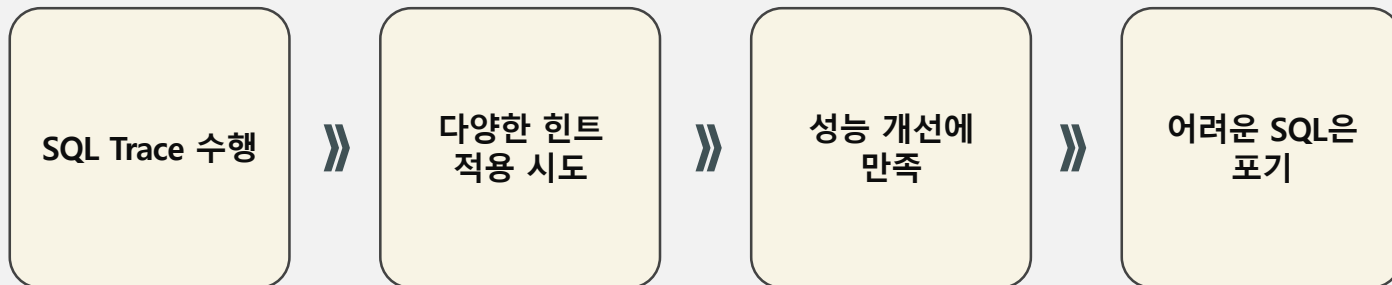
SQL의 이해

초심자의 SQL 튜닝 학습 과정

CASE 1



CASE 2



SQL 튜닝을 위한 핵심 요소

SQL Trace 이해

SQL 이해

나쁜 습관

- SQL을 보면 SQL Trace를 먼저 수행하고 SQL Trace 에서 비효율을 찾으려 한다.
- 이것저것 힌트를 적용해본 후 성능을 개선한다.
- 복잡한 SQL 은 능력 밖이라 생각한다.

좋은 습관

- SQL 자체를 이해하려고 노력한다.
- 어떤 업무에 사용되는 SQL 인지 이해하려고 노력한다.
- 집합적인 사고를 가지고 해당 SQL의 건수(집합)가 얼마나 될지, 어떻게 흘러갈지 고민해본다.
- 오라클은 어떻게 풀었는지 비판적인 생각을 가지고 접근한다.
- 이해가 안가는 부분에서는 항상 WHY 한다.

이해 예제

DEPT		EMP		
DEPTNO	DNAME	DEPTNP	DEPTNO	ENAME
10	ACCOUNTS	7782	10	CLARK
20	RESEARCH	7934	10	MILLER
30	SALES	7876	20	ADAMS
		7902	20	FORD
		7900	30	JAMES

```
select e.ename
      , e.empno
      , d.deptno
from emp e, dept d
where e.deptno =
d.deptno
and e.salary >= 100;
```



월급이 100 이상인 사람의 이름, 사번, 부서번호를 추출하는 SQL



아하...
월급이 100 이상인 사람이 많으면 Hash Join 이 유리하고
100 이상인 사람이 적으면
NL Join 이 유리하겠네

Trace와 SQL의 관계

Oracle

PLAN_TABLE_OUTPUT

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		14	238	2 (0)	00:00:01
* 1	TABLE ACCESS BY INDEX ROWID	EMP	14	238	2 (0)	00:00:01
* 2	INDEX FULL SCAN	TEST_IDX	14		1 (0)	00:00:01

Predicate Information (identified by)

- 1 - filter("E"."SAL">=100)
- 2 - filter("E"."DEPTNO" IS NOT NU

Trace는 SQL 에 종속적이지
절대 우선하지 않습니다.

PostgreSQL

```
"Aggregate (cost=2213.43..2213.44 rows=1 loops=1)"
" Buffers: shared hit=247 read=42 dirtied=0"
"  -> Bitmap Heap Scan on pg_proc (cost=77.21..2205.37 rows=3224 width=0) (actual time=0.375..132.549 rows=3224 loops=1)"
"    Heap Blocks: exact=275"
"    Buffers: shared hit=247 read=42 dirtied=54"
"  -> Bitmap Index Scan on pg_proc_oid_index (cost=0.00..76.40 rows=3224 width=0) (actual time=0.340..0.340 rows=3408 loops=1)"
"    Buffers: shared hit=14"
"Planning time: 0.493 ms"
"Execution time: 132.940 ms"
```

— SQL을 이해하고 집합적인 분석이 끝난 후 알아야 하는 전문지식

[인덱스 생성]

```
Create index idx1_emp on emp(sal);
```

[힌트 적용]

```
select /*+ leading(e) use_nl(e d) index(e idx1_emp)*/  
      e.ename  
    , e.empno  
    , d.deptno  
from emp e, dept d  
where e.deptno = d.deptno  
and e.salary >= 100;
```

※ JPPD, OPT_PARAM 등의 지식은 어려운 20% 미만의 SQL 성능을 개선하기 위해서 필요

STEP. 2

SQL Graph

■ SQL Graph란?

- SQL을 이해하는 가장 좋은 방법
- 정해진 방법은 없음
- 짧고 단순한 SQL 을 많이 하는것 보다, 길고 어려운 SQL 을 하루 이상 고민 추천
- SQL 은 결국 집합의 크기를 생각하고 조인하는 것

Quiz : 오라클에서 사용하는 조인 방법은 몇가지가 있을까요?

1. Nested Loop

2. Hash Join

3. Merge Join

4. Cartition Join

```
SELECT name, inverse, version  
from v$sql_hint  
where class = 'JOIN'
```

NAME	INVERSE	VERSION
USE_MERGE_CARTESIAN		11.1.0.6
USE_MERGE	NO_USE_MERGE	8.1.0
USE_HASH	NO_USE_HASH	8.1.0
USE_NL	NO_USE_NL	8.1.0

■ 작성 순서 (권고)

- #1 SQL 을 그림으로 표현
 - 연결관계, Alias, Predicate 등
- #2 각 집합의 조인 순서와 방법 고찰
 - 예측 조인 순서 및 방법을 #1에 추가
- #3 SQL Trace 결과 확인
 - Oracle Optimizer 가 선택한 조인 순서 및 방법을 #2에 추가

SQL

```

FROM ghub.idmtcord500 a inner join t34mcar0 c ON a.prjtno = c.reqno
AND c.companyid = a.companyid
AND c.todate = '9999999999999999'
AND c.purcpt IN ( SUBSTR( :b3 , 1 , 4 ) , :b2 )
AND c.purcpt||c.purim LIKE :b1 || '%' inner join gsrn.itvcord100 b ON b.prjtno = a.prjtno
AND b.ser = a.ser
AND b.seq = a.seq
AND b.islast = 'Y'
AND NVL( b.updcod , ' ' ) != 'D' left join t34mcar5 d ON d.reqno = a.prjtno
AND d.cartype = SUBSTR( a.ser , 3 , 2 )
AND d.companyid = a.companyid
AND d.todate = '9999999999999999' left join vw_emp e ON
a.vndcod = f.gre_code
AND f.companyid = :b4 left join gsrn.itvknd100 f1 ON
AND f1.vndcod = a.vndcod left join gsrn.tqcod300 g ON
AND g.codeid = '21016'
AND g.bizcode = :b4
AND g.lang = :b5 left join gsrn.tqcod300 h ON a.REGION = h.codevalue
AND h.codeid = '21016'
AND h.bizcode = :b4
AND h.lang = :b5 left join gsrn.tqcod300 i ON a.sregion
AND i.codeid = '21016'
AND i.bizcode = :b4
AND i.lang = :b5 left join gsrn.tqcod300 j ON a.REGION = j.codevalue||'00'
AND j.codeid = '21016'
AND j.bizcode = :b4
AND j.lang = :b5 left join gsrn.tqcod300 k ON k.codevalue = a.ordtyp
AND k.codeid = '21150'
AND k.bizcode = :b4
AND k.lang = :b5 left join gsrn.tqcod300 l ON l.codevalue = SUBSTR( a.ser , 3 , 2 )
AND l.codeid = '21013'
AND l.bizcode = :b4
AND l.lang = :b5 left join gsrn.tqcod300 ll ON ll.codevalue
AND ll.codeid = '21E17'
AND ll.bizcode = :b4
AND ll.lang = :b5 left join gsrn.tqcod300 n ON n.codevalue
AND n.codeid = '21E17'
AND n.bizcode = :b4
AND n.lang = :b5
WHERE a.status = 'A1'
AND a.islast = 'Y'
AND a.reqbn IN ( 'U' , 'D' , 'B' )
AND a.prjtno LIKE :b9 || '%'
AND a.recdte BETWEEN :b8
AND :b7
AND NOT EXISTS (
SELECT 'X'
FROM t34mcar5_sub s
WHERE s.reqno = a.prjtno
AND s.companyid = :b4
AND s.todate = '9999999999999999'
AND s.cartype != 'TT'
AND ROWNUM = 1
)
AND a.companyid = :b4
AND a.domainid = :b6

```

Trace

id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (CPU)	E-Time	A-Rows	A-Time	Buffers	Miss	Hit	Used-Mem
0	SELECT STATEMENT		1	1		82 (100)		18	00:00:06.19	809K			
1	1 SORT AGGREGATE		14	1	22			14	00:00:00.01	32			
2	2 TABLE ACCESS BY INDEX ROWID	T34MAGE	14	1	22	4 (0)	00:00:01	4	00:00:00.01	32			
3	3 INDEX RANGE SCAN	PK_T34MAGE	14	1		3 (0)	00:00:01	14	00:00:00.01	16			
4	4 SORT AGGREGATE		0	1	17			0	00:00:00.01	0			
5	5 TABLE ACCESS BY INDEX ROWID	T34MCARD	0	2	34	114 (0)	00:00:02	0	00:00:00.01	0			
6	6 INDEX RANG SCAN	SYS_C0007692	0	2		113 (0)	00:00:02	0	00:00:00.01	0			
7	7 SORT GROUP BY		18	1	36			18	00:00:00.01	40	1024	1024	
8	8 TABLE ACCESS BY INDEX ROWID	T34MCARD	18	1	36	4 (0)	00:00:01	28	00:00:00.01	40			
9	9 INDEX RANGE SCAN	SYS_C0007692	18	1		3 (0)	00:00:01	28	00:00:00.01	25			
10	10 SORT GROUP BY		0	1	36			0	00:00:00.01	0	73728	73728	
11	11 TABLE ACCESS BY INDEX ROWID	T34MCARD	0	1	36	4 (0)	00:00:01	0	00:00:00.01	0			
12	12 SORT AGGREGATE		0	1	36			0	00:00:00.01	0			
13	13 TABLE ACCESS BY INDEX ROWID	T34MCARD	1	1	2000	82 (3)	00:00:01	18	00:00:00.15	809K	172K	153K (0)	
14	14 SORT AGGREGATE		1	1				28	00:00:00.15	809K			
15	15 TABLE ACCESS BY INDEX ROWID	T34MCARD	1	1	2000	78 (2)	00:00:01	28	00:00:00.15	809K			
16	16 INDEX RANGE SCAN	PK_T34MCARD	1	1	1960	74 (2)	00:00:01	28	00:00:00.15	809K			
17	17 TABLE ACCESS BY INDEX ROWID	T34MCARD	1	1	1004	72 (2)	00:00:01	28	00:00:00.15	809K			
18	18 INDEX RANGE SCAN	PK_T34MCARD	1	1	1778	70 (2)	00:00:01	18	00:00:00.15	809K			
19	19 TABLE ACCESS BY INDEX ROWID	T34MCARD	1	1	1760	69 (2)	00:00:01	18	00:00:00.15	809K			
20	20 NESTED LOOPS OUTER		1	1	1650	67 (2)	00:00:01	18	00:00:00.15	809K			
21	21 NESTED LOOPS OUTER		1	1	1511	65 (2)	00:00:01	18	00:00:00.15	809K			
22	22 NESTED LOOPS OUTER		1	1	1434	63 (2)	00:00:01	18	00:00:00.15	809K			
23	23 NESTED LOOPS OUTER		1	1	890	55 (2)	00:00:01	18	00:00:00.01	775K			
24	24 NESTED LOOPS OUTER		1	1	871	53 (2)	00:00:01	18	00:00:00.01	758K			
25	25 NESTED LOOPS OUTER		1	1	752	51 (2)	00:00:01	18	00:00:00.38	459K			
26	26 NESTED LOOPS OUTER		1	1	633	49 (3)	00:00:01	18	00:00:01.50	227K			
27	27 NESTED LOOPS OUTER		1	1	514	47 (3)	00:00:01	18	00:00:00.41	444K			
28	28 NESTED LOOPS OUTER		1	1	385	45 (3)	00:00:01	18	00:00:00.37	361K			
29	29 NESTED LOOPS OUTER		1	1	276	43 (3)	00:00:01	18	00:00:00.08	126K			
30	30 NESTED LOOPS OUTER		1	1				18	00:00:00.07	69K			
31	31 NESTED LOOPS OUTER		1	1				1	00:00:00.07	69K			
32	32 BITMAP AND		1	1				3	00:00:00.02	319			
33	33 BITMAP CONVERSION FROM ROWIDS	IX_10MTCDRESGO_STATUS	1	1	5947	14 (0)	00:00:01	74717	00:00:00.02	319			
34	34 INDEX RANGE SCAN	IX_10MTCDRESGO_STATUS	1	1				1	00:00:00.04	349			
35	35 SORT ORDER BY		1	1				14085	00:00:00.03	349	619K	472K	550K (0)
36	36 INDEX RANGE SCAN	IX_10MTCDRESGO_STATUS	1	1	5947	26 (0)	00:00:01	14085	00:00:00.02	349			
37	37 INDEX RANGE SCAN	T000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:00.25	296K			
38	38 INDEX RANGE SCAN	T000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:00.04	530K			
39	39 INDEX RANGE SCAN	T000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:00.48	182K			
40	40 INDEX RANGE SCAN	T000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:01.48	182K			
41	41 INDEX RANGE SCAN	T000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:01.32	182K			
42	42 INDEX RANGE SCAN	T000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:01.31	182K			
43	43 INDEX RANGE SCAN	T000000_PK	1828	1	444	1 (0)	00:00:01	1611	00:00:00.14	247K			
44	44 VIEW PEEKED INDEX DATE	V170M000	1828	1	426	8 (0)	00:00:01	1611	00:00:00.14	247K			
45	45 FILTER		1828	1	300	7 (0)	00:00:01	1611	00:00:00.12	214K			
46	46 NESTED LOOPS OUTER		1828	1	362	7 (0)	00:00:01	1611	00:00:00.11	205K			
47	47 NESTED LOOPS OUTER		1828	1	359	7 (0)	00:00:01	1611	00:00:00.10	204K			
48	48 NESTED LOOPS OUTER		1828	1	323	6 (0)	00:00:01	1611	00:00:00.09	179K			
49	49 NESTED LOOPS OUTER		1828	1	277	5 (0)	00:00:01	1611	00:00:00.08	151K			
50	50 NESTED LOOPS OUTER		1828	1	231	4 (0)	00:00:01	1611	00:00:00.07	155K			
51	51 NESTED LOOPS OUTER		1828	1	165	3 (0)	00:00:01	1611	00:00:00.06	156K			
52	52 NESTED LOOPS OUTER		1828	1	139	2 (0)	00:00:01	1611	00:00:00.05	173K			
53	53 NESTED LOOPS OUTER		1828	1	127	2 (0)	00:00:01	1611	00:00:00.04	173K			
54	54 TABLE ACCESS BY INDEX ROWID	T000100	1828	1	111	2 (0)	00:00:01	1611	00:00:00.03	158K			
55	55 INDEX RANGE SCAN	PK_T000100	1828	1	95	2 (0)	00:00:01	1611	00:00:00.03	461K			
56	56 TABLE ACCESS BY INDEX ROWID	T000100	1828	1	79	2 (0)	00:00:01	1611	00:00:00.02	304K			
57	57 INDEX RANGE SCAN	PK_T000100	1828	1	1	1 (0)	00:00:01	1611	00:00:00.01	161K			
58	58 INDEX RANGE SCAN	PK_T000100	1828	1	16	0 (0)		124	00:00:00.01	137K			
59	59 INDEX RANGE SCAN	PK_T000100	1828	1	16	0 (0)		124	00:00:00.01	137K			
60	60 INDEX RANGE SCAN	PK_T000100	1828	1	16	0 (0)		124	00:00:00.01	137K			
61	61 INDEX RANGE SCAN	PK_T000100	1828	1	12	0 (0)		460	00:00:00.01	9			
62	62 INDEX RANGE SCAN	PK_T000100	1828	1	46	1 (0)	00:00:01	1611	00:00:00.01	261K			
63	63 INDEX RANGE SCAN	PK_T000100	1828	1	46	1 (0)	00:00:01	1611	00:00:00.01	261K			
64	64 INDEX RANGE SCAN	PK_T000100	1828	1	46	1 (0)	00:00:01	1611	00:00:00.01	261K			
65	65 INDEX RANGE SCAN	PK_T000100	1828	1	46	1 (0)	00:00:01	1611	00:00:00.01	261K			
66	66 INDEX RANGE SCAN	PK_T000100	1828	1	46	1 (0)	00:00:01	1611	00:00:00.01	261K			
67	67 INDEX RANGE SCAN	PK_T000100	1828	1	46	1 (0)	00:00:01	1611	00:00:00.01	261K			
68	68 INDEX RANGE SCAN	PK_T000100	1828	1	46	1 (0)	00:00:01	1611	00:00:00.01	261K			
69	69 INDEX RANGE SCAN	PK_T000100	1828	1	46	1 (0)	00:00:01	1611	00:00:00.01	261K			
70	70 INDEX RANGE SCAN	PK_T000100	1828	1	46	1 (0)	00:00:01	1611	00:00:00.01	261K			
71	71 INDEX RANGE SCAN	PK_T000100	1828	1	46	1 (0)	00:00:01	1611	00:00:00.01	261K			
72	72 TABLE ACCESS BY INDEX ROWID	T34MCARD	1828	1	77	2 (0)	00:00:01	18	00:00:00.03	444K			
73	73 INDEX RANGE SCAN	SYS_C0007692	1828	1	1	1 (0)	00:00:01	18	00:00:00.01	322K			
74	74 INDEX RANGE SCAN	T000000_PK	18	1	119	2 (0)	00:00:01	1611	00:00:00.01	51			
75	75 INDEX RANGE SCAN	T000000_PK	18	1	119	2 (0)	00:00:01	1611	00:00:00.01	49			
76	76 TABLE ACCESS BY INDEX ROWID	T000000_PK	18	1	119	2 (0)	00:00:01	1611	00:00:00.01	27			
77	77 INDEX RANGE SCAN	PK_T000100	18	1	1	1 (0)	00:00:01	18	00:00:00.01	9			
78	78 TABLE ACCESS BY INDEX ROWID	T34MCARD	18	1	56	3 (0)	00:00:01	28	00:00:00.01	46			
79	79 INDEX RANGE SCAN	SYS_C0007692	18	1	1	1 (0)	00:00:01	28	00:00:00.01	11			
80	80 TABLE ACCESS BY INDEX ROWID	T000000_PK	28	1	100	1 (0)	00:00:01	28	00:00:00.01	37			
81	81 INDEX RANGE SCAN	T000000_PK	28	1	1	1 (0)	00:00:01	28	00:00:00.01	9			
82	82 TABLE ACCESS BY INDEX ROWID	T000100	28	1	60	4 (0)	00:00:01	28	00:00:00.01	64			
83	83 INDEX RANGE SCAN	PK_T000100	28	1	1	3 (0)	00:00:01	28	00:00:00.01	50			
84	84 COUNT STOPKEY		14	1				0	00:00:00.01	24			
85	85 INDEX RANGE SCAN	T34MCARD_SUB_PK	14	1	34	3 (0)	00:00:01	28	00:00:00.01	24			

쿼리가 길다..!

트레이스 결과가 복잡하다..!

어렵다...

이건.. 내가 할 수 있는게 아니야..!



SQL

```

FROM ghub.idmtcord500 a inner join t34mcar0 c ON a.prjtno = c.reqno
AND c.companyid = a.companyid
AND c.todate = '9999999999999999'
AND c.purcpt IN ( SUBSTR( :b3 , 1 , 4 ) , :b2 )
AND c.purcpt||c.purim LIKE :b1 || '%' inner join gsrn.itvcord100 b ON b.prjtno = a.prjtno
AND b.ser = a.ser
AND b.seq = a.seq
AND b.islast = 'Y'
AND NVL( b.updcod , ' ' ) != 'D' left join t34mcar5 d ON d.reqno = a.prjtno
AND d.cartype = SUBSTR( a.ser , 3 , 2 )
AND d.companyid = a.companyid
AND d.todate = '9999999999999999' left join vw_emp e ON a.dwgim = e.emp_no left join imt_vnd f ON
a.vndcod = f.gre_code
AND f.companyid = :b4 left join gsrn.itvknd100 f1 ON f1.companyid = :b4
AND f1.vndcod = a.vndcod left join gsrn.tqcod300 g ON a.sregion = g.codevalue
AND g.codeid = '21016'
AND g.bizcode = :b4
AND g.lang = :b5 left join gsrn.tqcod300 h ON a.sregion = h.codevalue
AND h.codeid = '21016'
AND h.bizcode = :b4
AND h.lang = :b5 left join gsrn.tqcod300 i ON a.sregion = i.codevalue||'00'
AND i.codeid = '21016'
AND i.bizcode = :b4
AND i.lang = :b5 left join gsrn.tqcod300 j ON a.sregion = j.codevalue||'00'
AND j.codeid = '21016'
AND j.bizcode = :b4
AND j.lang = :b5 left join gsrn.tqcod300 k ON k.codevalue = a.ordtyp
AND k.codeid = '21150'
AND k.bizcode = :b4
AND k.lang = :b5 left join gsrn.tqcod300 l ON l.codevalue = SUBSTR( a.ser , 3 , 2 )
AND l.codeid = '21013'
AND l.bizcode = :b4
AND l.lang = :b5 left join gsrn.tqcod300 ll ON ll.codevalue = c.unitgbn
AND ll.codeid = '21E17'
AND ll.bizcode = :b4
AND ll.lang = :b5 left join gsrn.tqcod300 n ON n.codevalue = c.unitgbn
AND n.codeid = '21E17'
AND n.bizcode = :b4
AND n.lang = :b5
WHERE a.status = 'A1'
AND a.islast = 'Y'
AND a.reqgbn IN ( 'U' , 'D' , 'B' )
AND a.prjtno LIKE :b9 || '%'
AND a.recdte BETWEEN :b8
AND :b7
AND NOT EXISTS (
SELECT 'X'
FROM t34mcar5_sub s
WHERE s.reqno = a.prjtno
AND s.companyid = :b4
AND s.todate = '9999999999999999'
AND s.cartype != 'TT'
AND ROWNUM = 1
)
AND a.companyid = :b4
AND a.domainid = :b6

```

Trace

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (CPU)	E-Time	A-Rows	A-Time	Buffers	Miss	Hit	Used-Mem
0	SELECT STATEMENT		1	1		82 (100)		18	00:00:06.19	809K			
1	SQL*Aggregate		14	1	22			14	00:00:00.01	32			
2	TABLE ACCESS BY INDEX ROWID	T34MAGE	14	1	22	4 (0)	00:00:01	4	00:00:00.01	32			
3	INDEX RANGE SCAN	PK_T34MAGE	14	1		3 (0)	00:00:01	14	00:00:00.01	16			
4	SQL*Aggregate		0	1	17			0	00:00:00.01	0			
5	TABLE ACCESS BY INDEX ROWID	T34MCAR5	0	2	34	114 (0)	00:00:02	0	00:00:00.01	0			
6	INDEX SKIP SCAN	SYS_I0000762	0	2	113	(0)	00:00:02	0	00:00:00.01	0			
7	SQL*Group By		18	1	36			18	00:00:00.01	40	1024	1024	
8	TABLE ACCESS BY INDEX ROWID	T34MCAR5	18	1	36	4 (0)	00:00:01	28	00:00:00.01	40			
9	INDEX RANGE SCAN	SYS_I0000762	18	1	3	(0)	00:00:01	28	00:00:00.01	25			
10	SQL*Group By		0	1	36			0	00:00:00.01	0	73728	73728	
11	TABLE ACCESS BY INDEX ROWID	T34MCAR5	0	1	36	4 (0)	00:00:01	0	00:00:00.01	0			
12	INDEX RANGE SCAN	SYS_I0000762	0	1	3	(0)	00:00:01	0	00:00:00.01	0			
13	SQL*Group By		1	1	2000	82 (3)	00:00:01	18	00:00:06.15	809K	179K	153K (0)	
14	FILTER		1	1				28	00:00:06.15	809K			
15	NESTED LOOPS		1	1	2000	78	(2)	28	00:00:06.15	809K			
16	NESTED LOOPS OUTER		1	1	1960	74	(2)	28	00:00:06.15	809K			
17	NESTED LOOPS OUTER		1	1	1004	72	(2)	28	00:00:06.15	809K			
18	NESTED LOOPS OUTER		1	1	1778	70	(2)	18	00:00:06.15	809K			
19	NESTED LOOPS OUTER		1	1	1778	69	(2)	18	00:00:06.15	809K			
20	NESTED LOOPS OUTER		1	1	1511	67	(2)	18	00:00:06.15	809K			
21	NESTED LOOPS		1	1	1511	65	(2)	18	00:00:06.15	809K			
22	NESTED LOOPS OUTER		1	1	1424	63	(2)	18	00:00:06.15	809K			
23	NESTED LOOPS OUTER		1	1	890	59	(2)	18	00:00:06.15	772K			
24	NESTED LOOPS OUTER		1	1	871	53	(2)	18	00:00:06.15	589K			
25	NESTED LOOPS OUTER		1	1	752	51	(2)	18	00:00:06.15	459K			
26	NESTED LOOPS OUTER		1	1	833	49	(3)	18	00:00:06.15	227K			
27	NESTED LOOPS OUTER		1	1	514	47	(3)	18	00:00:06.15	442K			
28	NESTED LOOPS OUTER		1	1	424	45	(3)	18	00:00:06.15	442K			
29	NESTED LOOPS OUTER		1	1	276	43	(3)	18	00:00:06.15	120			
30	TABLE ACCESS BY INDEX ROWID	I00T000000	1	1	276	43	(3)	18	00:00:06.15	120			
31	BITMAP CONVERSION TO ROWIDS		1	1				18	00:00:06.15	608			
32	BITMAP AND		1	1				1	00:00:00.07	608			
33	BITMAP CONVERSION FROM ROWIDS		1	1				1	00:00:00.02	319			
34	INDEX RANGE SCAN	I_00T0000000_STATUS	1	1	5947	14 (0)	00:00:01	74717	00:00:02.31	319			
35	INDEX RANGE SCAN		1	1				1	00:00:00.04	349			
36	INDEX RANGE SCAN	I_00T0000000_STATUS	1	1				1	00:00:00.00	349	619K	472K	550K (0)
37	INDEX RANGE SCAN	I_00T0000000_STATUS	1	1	5947	26 (0)	00:00:01	14265	00:00:02.31	349			
38	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
39	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
40	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
41	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
42	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
43	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
44	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
45	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
46	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
47	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
48	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
49	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
50	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
51	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
52	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
53	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
54	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
55	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
56	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
57	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
58	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
59	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
60	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
61	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
62	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
63	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
64	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
65	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
66	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
67	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
68	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
69	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
70	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
71	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
72	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
73	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
74	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
75	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
76	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
77	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
78	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
79	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
80	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
81	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
82	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
83	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
84	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			
85	INDEX RANGE SCAN	I0000000_PK	1828	1	119	2 (0)	00:00:01	1611	00:00:02.25	319K			

SQL

```

Select a.c1, a.c2, a.c3
From cust_detail a
    , mng_agent b
    , sap_agent c
    , cust d
Where a.cust_no = b.cno
And a.cust_no = d.cno
And b.agent = c.agent
And c.org4 = :org4
And a.cust_name = :cnm
Group by a.c1, a.c2, a.c3;
    
```

Trace

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	2	0.39	15.86	3013	14348	0	2
total	4	0.39	15.86	3013	14348	0	2

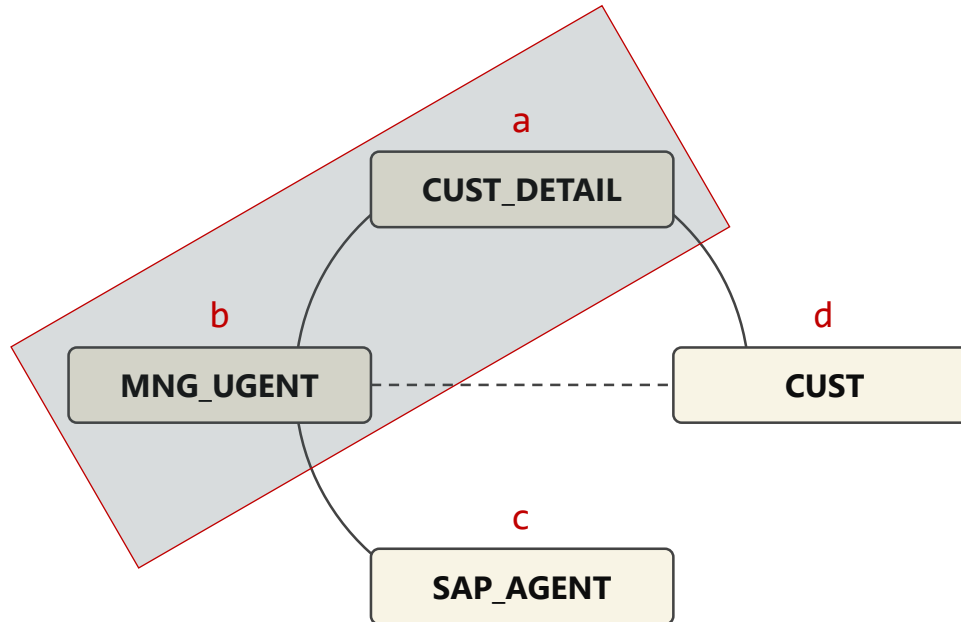
```

Rows      Row Source Operation
-----
      2  SORT GROUP BY (cr=14235 r=2993 w=0 time=15605897 us)
      6  NESTED LOOPS (cr=14235 r=2993 w=0 time=15605783 us)
      6    NESTED LOOPS (cr=14215 r=2993 w=0 time=15605669 us)
     2852    NESTED LOOPS (cr=8510 r=2985 w=0 time=15502113 us)
     2030      TABLE ACCESS BY INDEX ROWID TBL_OTC_CUST_DETAIL (cr=1772 r=1430 w=0 time=6266821 us)
     2030        INDEX RANGE SCAN IDX_TBL_OTC_CUST_DETAIL_X0 (cr=12 r=10 w=0 time=84509 us)
     2852      TABLE ACCESS BY INDEX ROWID TBL_OTC_MNG_AGENT (cr=6738 r=1555 w=0 time=9220612 us)
     2852        INDEX RANGE SCAN IDX_TBL_OTC_MNG_AGENT_PK (cr=4075 r=463 w=0 time=2863287 us)
      6      TABLE ACCESS BY INDEX ROWID WC_BP_SAP_AGENT (cr=5705 r=8 w=0 time=93932 us)
     2851        INDEX UNIQUE SCAN PKWC_BP_SAP_AGENT (cr=2854 r=0 w=0 time=23478 us)(object id 17313)
      6      TABLE ACCESS BY INDEX ROWID TBL_OTC_CUST (cr=20 r=0 w=0 time=81 us)
      6      INDEX UNIQUE SCAN IDX_TBL_OTC_CUST_PK (cr=14 r=0 w=0 time=48 us)(object id 18373)
    
```

SQL

```
Select a.c1, a.c2, a.c3
From cust_detail a
     , mng_agent b
     , sap_agent c
     , cust d
Where a.cust_no = b.cno
And a.cust_no = d.cno
And b.agent = c.agent
And c.org4 = :org4
And a.cust_name = :cnm
Group by a.c1, a.c2, a.c3;
```

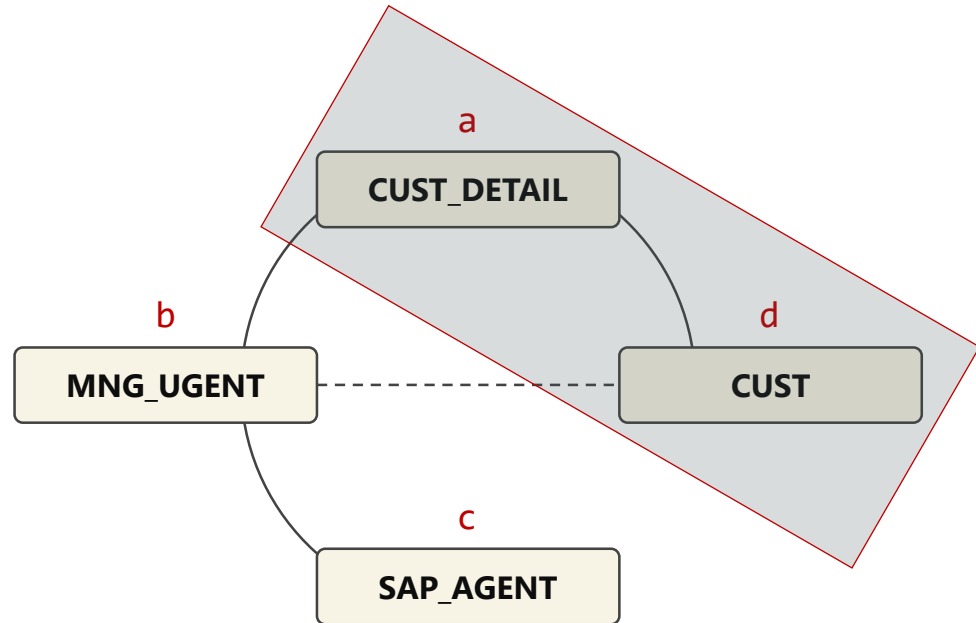
Graph



SQL

```
Select a.c1, a.c2, a.c3
From cust_detail a
     , mng_agent b
     , sap_agent c
     , cust d
Where a.cust_no = b.cno
And a.cust_no = d.cno
And b.agent = c.agent
And c.org4 = :org4
And a.cust_name = :cnm
Group by a.c1, a.c2, a.c3;
```

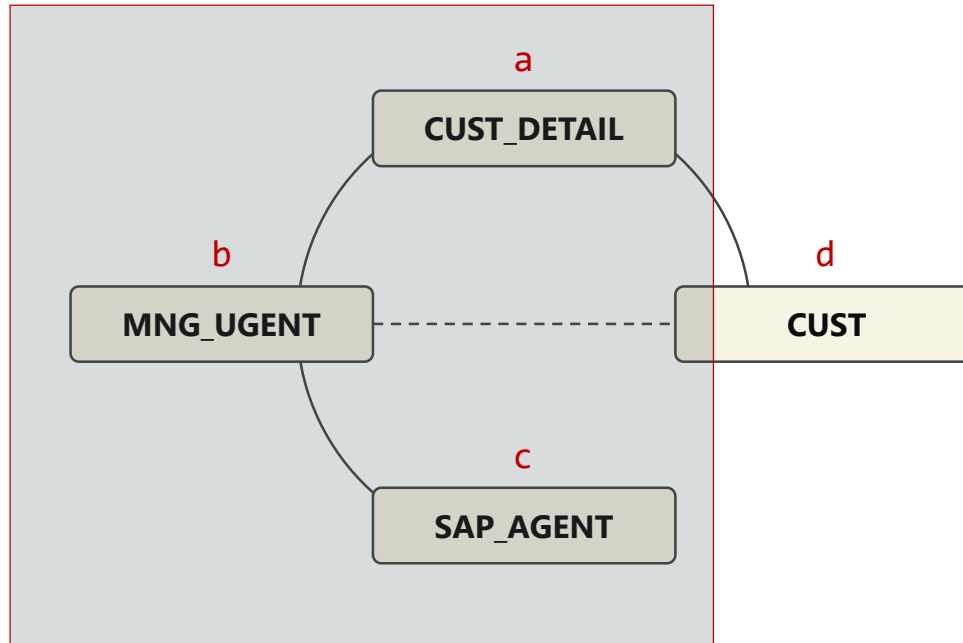
Graph



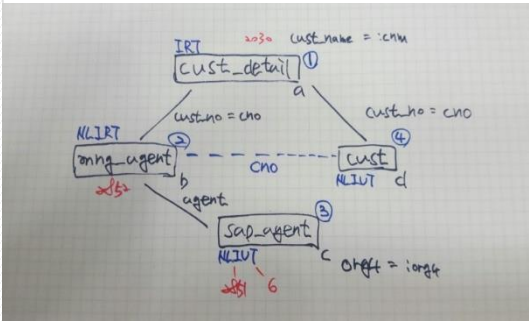
SQL

```
Select a.c1, a.c2, a.c3
From cust_detail a
     , mng_agent b
     , sap_agent c
     , cust d
Where a.cust_no = b.cno
And a.cust_no = d.cno
And b.agent = c.agent
And c.org4 = :org4
And a.cust_name = :cnm
Group by a.c1, a.c2, a.c3;
```

Graph



Graph



Trace

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	2	0.39	15.86	3013	14348	0	2
total	4	0.39	15.86	3013	14348	0	2

Rows	Row Source Operation
2	SORT GROUP BY (cr=14235 r=2993 w=0 time=15605897 us)
6	NESTED LOOPS (cr=14235 r=2993 w=0 time=15605783 us)
6	NESTED LOOPS (cr=14215 r=2993 w=0 time=15605669 us)
2852	NESTED LOOPS (cr=8510 r=2985 w=0 time=15502113 us)
2030	TABLE ACCESS BY INDEX ROWID TBL_OTC_CUST_DETAIL (cr=1772 r=1430 w=0 time=6266821 us)
2030	INDEX RANGE SCAN IDX_TBL_OTC_CUST_DETAIL_X0 (cr=12 r=10 w=0 time=84509 us)
2852	TABLE ACCESS BY INDEX ROWID TBL_OTC_MNG_AGENT (cr=6738 r=1555 w=0 time=9220612 us)
2852	INDEX RANGE SCAN IDX_TBL_OTC_MNG_AGENT_PK (cr=4075 r=463 w=0 time=2863287 us)
6	TABLE ACCESS BY INDEX ROWID WC_BP_SAP_AGENT (cr=5705 r=8 w=0 time=93932 us)
2851	INDEX UNIQUE SCAN PKWC_BP_SAP_AGENT (cr=2854 r=0 w=0 time=23478 us)(object id 17313)
6	TABLE ACCESS BY INDEX ROWID TBL_OTC_CUST (cr=20 r=0 w=0 time=81 us)
6	INDEX UNIQUE SCAN IDX_TBL_OTC_CUST_PK (cr=14 r=0 w=0 time=48 us)(object id 18373)

STEP. 3

다양한 SQL Trace

10046 Trace

DBMS_XPLAN

DBMS_MONITOR

DBMS_SQLTUNE

10046 Trace?

- 10046 Trace 는 과거에 머물러 있지 않다.

10046 Trace

10046 Trace는 과거형일까?

LEVEL	DESCRIPTION	VERSION
1	기본 정보	ALL
4	기본 정보 + Bind 정보	ALL
8	기본 정보 + Wait Event 정보	ALL
16	기본 정보 + 매 실행마다 실행계획 STAT 정보 저장	11.1 이상
32	기본 정보 + 실행계획을 제외하고 저장	11.1 이상
64	기본 정보 + 최초 수행 이후 60초 주기로 수행	11.2.0.2 이상

10046 Trace

DBMS_XPLAN

DBMS_MONITOR

DBMS_SQLTUNE

DBMS_XPLAN?

- STEP1. 수행 원리
- STEP2. 출력 기준 / 내용 이해
- STEP3. 활용
- STEP4. Optimizer 연결고리

DBMS_XPLAN

DISPLAY

```
EXPLAIN PLAN
SET      STATEMENT_ID = 'imsi'
INTO     xtool_plan_table
FOR
SELECT /*+ leading(t) use_nl(d) */
        COUNT( * )
FROM     target t ,
        dummy d
WHERE    t.tn1 = d.dn2
AND      t.tn2 > 5
AND      t.tc1 > 5
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	61	3 (0)	00:00:01
1	SORT AGGREGATE		1	61		
2	NESTED LOOPS		2	122	3 (0)	00:00:01
* 3	TABLE ACCESS BY INDEX ROWID	TARGET	1	48	2 (0)	00:00:01
* 4	INDEX RANGE SCAN	TARGET_IDX1	3		1 (0)	00:00:01
* 5	INDEX RANGE SCAN	DUMMY_IDX1	2	26	1 (0)	00:00:01

Predicate Information (identified by operation id):

```
3 - filter(TO_NUMBER("T"."TC1")>5)
4 - access("T"."TN2">5 AND "T"."TN2" IS NOT NULL)
5 - access("T"."TN1"="D"."DN2")
```

DISPLAY_CURSOR

```
SELECT /*+ gather_plan_statistics
        leading(t) use_nl(d) */
        COUNT( * )
FROM     target t ,
        dummy d
WHERE    t.tn1 = d.dn2
AND      t.tn2 > 5
AND      t.tc1 > 5
```

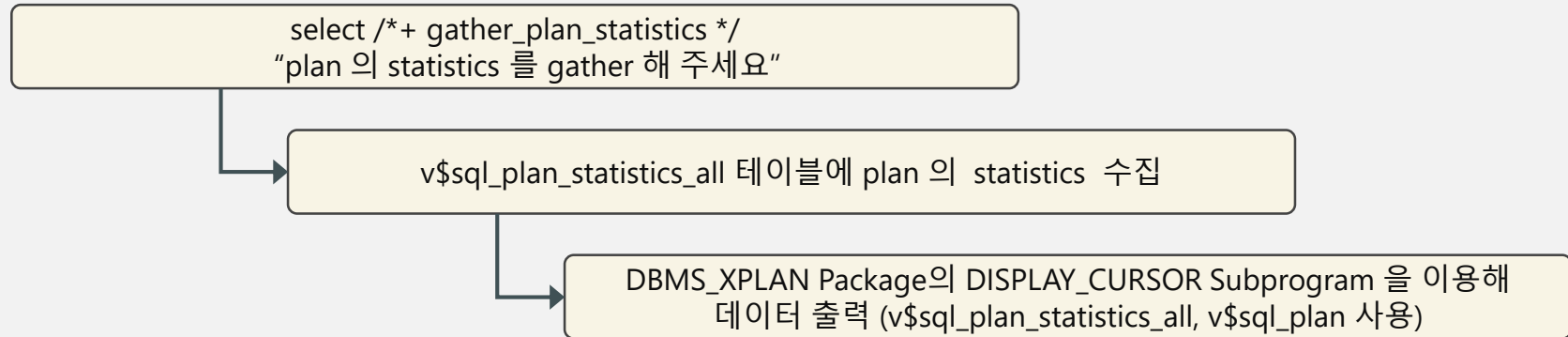
Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-Time	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1			3 (100)		1	00:00:05.21	141K	69
1	SORT AGGREGATE		1	1	61			1	00:00:05.21	141K	69
2	NESTED LOOPS		1	2	122	3 (0)	00:00:01	20M	00:00:03.90	141K	69
* 3	TABLE ACCESS BY INDEX ROWID	TARGET	1	1	48	2 (0)	00:00:01	40960	00:00:00.03	1432	69
* 4	INDEX RANGE SCAN	TARGET_IDX1	1	3		1 (0)	00:00:01	40960	00:00:00.01	196	69
* 5	INDEX RANGE SCAN	DUMMY_IDX1	40960	2	26	1 (0)	00:00:01	20M	00:00:01.79	140K	0

Predicate Information (identified by operation id):

```
3 - filter(TO_NUMBER("T"."TC1")>5)
4 - access("T"."TN2">5 AND "T"."TN2" IS NOT NULL)
5 - access("T"."TN1"="D"."DN2")
```

DBMS_XPLAN

STEP1. DISPLAY_CURSOR 수행 원리



GATHER_PLAN_STATISTICS 힌트가 STATISTICS 의 추가 수집을 의미하기 때문에
아래 두가지 방법으로도 Plan 의 Statistics 를 수집할 수 있습니다.

- STATISTICS LEVEL = ALL (Parameter)
- _ROWSOURCE_EXECUTION_STATISTICS = true (Hidden Parameter)

DBMS_XPLAN

STEP2. DISPLAY_CURSOR 출력 기준 / 내용 이해 (1)

```

--- function display_cursor(sql_id          varchar2 default null,
---                          cursor_child_no integer default 0,
---                          format         varchar2 default 'TYPICAL')
---
--- - sql_id:
---   specifies the sql_id value for a specific SQL statement, as
---   shown in V$SQL.SQL_ID, V$SESSION.SQL_ID, or
---   V$SESSION.PREV_SQL_ID. If no sql_id is specified, the last
---   executed statement of the current session is shown.
---
--- - cursor_child_no:
---   specifies the child number for a specific sql cursor, as shown in
---   V$SQL.CHILD_NUMBER or in V$SESSION.SQL_CHILD_NUMBER,
---   V$SESSION.PREV_CHILD_NUMBER. This input parameter is only
---   considered when sql_id is set.
---
---   If not specified, all child cursors for the specified sql_id are
---   displayed.

```

함께 사용하면 좋은 쿼리

- select * from table(dbms_xplan.display_cursor(null, null, 'typical allstats last'))
- select * from v\$active_session_history where session_id = userenv('sid')
- alter session set current_schema=maxgauge
- alter session set statistics_level=all;

DBMS_XPLAN

STEP2. DISPLAY_CURSOR 출력 기준 / 내용 이해 (2)

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT				18 (100)						
1	PX COORDINATOR										
2	PX SEND QC (RANDOM)	:TQ10001	16	992	18 (6)	00:00:01			Q1,01	P->S	QC (RAND)
3	HASH GROUP BY		16	992	18 (6)	00:00:01			Q1,01	PCWP	
4	PX RECEIVE		16	992	18 (6)	00:00:01			Q1,01	PCWP	
5	PX SEND HASH	:TQ10000	16	992	18 (6)	00:00:01			Q1,00	P->P	HASH
6	HASH GROUP BY		16	992	18 (6)	00:00:01			Q1,00	PCWP	
7	NESTED LOOPS		16	992	17 (0)	00:00:01			Q1,00	PCWP	
8	NESTED LOOPS		20	992	17 (0)	00:00:01			Q1,00	PCWP	
9	PX BLOCK ITERATOR		2	114	15 (0)	00:00:01	1	5	Q1,00	PCWC	
* 10	TABLE ACCESS FULL	TARGET	2	114	15 (0)	00:00:01	1	5	Q1,00	PCWP	
* 11	INDEX RANGE SCAN	DUMMY_IDX1	10		1 (0)	00:00:01			Q1,00	PCWP	
12	TABLE ACCESS BY INDEX ROWID	DUMMY	10	50	1 (0)	00:00:01			Q1,00	PCWP	

Plan Format option

Basic	ID, OPERATION, NAME의 결과를 출력.
Typical	ID, OPERATION, NAME, ROWS, BYTES, COST를 출력. + Predicate Information 정보 출력
All	TYPICAL 의 PLAN 정보와 동일 + Query Block Name, Column Projection 추가로 출력.
Advanced	ALL의 정보와 동일 + Outline 정보, Peeked Binds, Note 추가로 출력.
Allstats	ID, OPERATION, NAME, STARTS, E-ROWS, A-ROWS, BUFFERS 출력 + Predicate Information 정보 출력 (Allstats Format은 Plan Statistics 의 누적정보를 보여준다)
Allstats last	Allstats Format과 동일 (가장 최근에 실행된 Plan Statistics 만 출력)

DBMS_XPLAN

STEP2. DISPLAY_CURSOR 출력 기준 / 내용 이해 (3)

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

```
1 - SEL$1
10 - SEL$1 / T@SEL$1
11 - SEL$1 / D@SEL$1
12 - SEL$1 / D@SEL$1
```

Advanced
All

Outline Data

```
/*+
  BEGIN_OUTLINE_DATA
  IGNORE_OPTIM_EMBEDDED_HINTS
  OPTIMIZER_FEATURES_ENABLE('11.2.0.4')
  DB_VERSION('11.2.0.4')
  ALL_ROWS
  SHARED(2)
  OUTLINE_LEAF(@"SEL$1")
  FULL(@"SEL$1" "T"@"SEL$1")
  INDEX(@"SEL$1" "D"@"SEL$1" ("DUMMY"."DN2" "DUMMY"."DDT"))
  LEADING(@"SEL$1" "T"@"SEL$1" "D"@"SEL$1")
  USE_NL(@"SEL$1" "D"@"SEL$1")
  NLJ_BATCHING(@"SEL$1" "D"@"SEL$1")
  PQ_DISTRIBUTE(@"SEL$1" "D"@"SEL$1" NONE BROADCAST)
  GBY_PUSHDOWN(@"SEL$1")
  USE_HASH_AGGREGATION(@"SEL$1")
  END_OUTLINE_DATA
*/
```

Peeked Binds (identified by position):

```
1 - (VARCHAR2(30), CSID=846): '1'
2 - (VARCHAR2(30), CSID=846): '1'
```

Predicate Information (identified by operation id):

```
10 - access(:Z>=:Z AND :Z<=:Z)
      filter(("T"."TC1"=:B1 AND ("D"."TN2"=:TO_NUMBER(:B1)))
11 - access("T"."TN1"=:D"."DN2")
      filter("D"."DN2" IS NOT NULL)
```

Advanced
All
Typical
Basic

Column Projection Information (identified by operation id):

```
1 - "T"."TDT"[DATE,7], SUM()[22]
2 - (#keys=0) "T"."TDT"[DATE,7], SUM()[22]
3 - "T"."TDT"[DATE,7], SUM()[22]
4 - "T"."TDT"[DATE,7], SUM()[22]
5 - (#keys=1) "T"."TDT"[DATE,7], SYS_OP_MSR()[25]
6 - "T"."TDT"[DATE,7], SYS_OP_MSR()[25]
7 - "T"."TDT"[DATE,7], "D"."DN1"[NUMBER,22]
8 - "T"."TDT"[DATE,7], "D".ROWID[ROWID,10]
9 - "T"."TN1"[NUMBER,22], "T"."TDT"[DATE,7]
10 - "T"."TN1"[NUMBER,22], "T"."TDT"[DATE,7]
11 - "D".ROWID[ROWID,10]
12 - "D"."DN1"[NUMBER,22]
```

Advanced
All
Typical
Basic

Note

- dynamic sampling used for this statement (level=2)
- Degree of Parallelism is 2 because of hint

Advanced
All
Typical
Basic

DBMS_XPLAN

STEP2. DISPLAY_CURSOR 출력 기준 / 내용 이해 (3)

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

Plan row option	
outline	Outline 정보 출력을 제어Oracle에서 해당 Query를 수행하기 위해 사용된 Hint 들이 출력된다. (사용자에게 의한 Hint + Optimizer가 부여한 힌트 포함)
predicate	Predicate Information 정보 출력을 제어 (각 Operation의 Access & Filter 정보를 출력)
alias	Query Block Name / Object Alias 정보 출력 제어
projection	projection information 정보 출력을 제어한다. (각 Operation의 Select Column 정보를 출력, PARTITION TABLE 조회시 해당 DATA 출력)
remote	remote 정보 출력을 제어. (DB LINK 사용시 REMOTE Query의 정보 출력)
peeked_binds	Bind 변수 사용정보 출력을 제어 (_optim_peek_user_binds = TRUE 인경우 출력)
note	특이사항에 대한 정보 출력 (Cardinality Feed Back 적용여부, Dynamic sampling, plan table의 old Version인지 등 특이사항에 대한 정보 출력)

DBMS_XPLAN

STEP2. DISPLAY_CURSOR 출력 기준 / 내용 이해 (4)

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

Id		Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-Time	Pstart	Pstop	TQ	IN-OUT	PQ Distrib	A-Rows	A-Time	Buffers	CMem	IMem	Used-Mem
0		SELECT STATEMENT		1			18 (100)							1	00:00:00.02	15			
1		PX COORDINATOR		1										1	00:00:00.02	15			
2		PX SEND QC (RANDOM)	:TQ10001	0	16	992	18 (6)	00:00:01			Q1,01	P->S	QC (RAND)	0	00:00:00.01	0			
3		HASH GROUP BY		0	16	992	18 (6)	00:00:01			Q1,01	POW		0	00:00:00.01	0	928K	928K	
4		PX RECEIVE		0	16	992	18 (6)	00:00:01			Q1,01	POW		0	00:00:00.01	0			
5		PX SEND HASH	:TQ10000	0	16	992	18 (6)	00:00:01			Q1,01	P->P	HASH	0	00:00:00.01	0			
6		HASH GROUP BY		0	16	992	18 (6)	00:00:01			Q1,00	POW		0	00:00:00.01	0	928K	928K	
7		NESTED LOOPS		0	16	992	17 (0)	00:00:01			Q1,00	POW		0	00:00:00.01	0			
8		NESTED LOOPS		0	20	992	17 (0)	00:00:01			Q1,00	POW		0	00:00:00.01	0			
9		PX BLOCK ITERATOR		0	2	114	15 (0)	00:00:01	1	5	Q1,00	POW		0	00:00:00.01	0			
* 10		TABLE ACCESS FULL	TARGET	0	2	114	15 (0)	00:00:01	1	5	Q1,00	POW		0	00:00:00.01	0			
* 11		INDEX RANGE SCAN	DUMMY_IDX1	0	10		1 (0)	00:00:01			Q1,00	POW		0	00:00:00.01	0			
12		TABLE ACCESS BY INDEX ROWID	DUMMY	0	10	50	1 (0)	00:00:01			Q1,00	POW		0	00:00:00.01	0			

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

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT				18 (100)						
1	PX COORDINATOR										
2	PX SEND QC (RANDOM)	:TQ10001	16	992	18 (6)	00:00:01			Q1,01	P->S	QC (RAND)
3	HASH GROUP BY		16	992	18 (6)	00:00:01			Q1,01	PQWP	
4	PX RECEIVE		16	992	18 (6)	00:00:01			Q1,01	PQWP	
5	PX SEND HASH	:TQ10000	16	992	18 (6)	00:00:01			Q1,00	P->P	HASH
6	HASH GROUP BY		16	992	18 (6)	00:00:01			Q1,00	PQWP	
7	NESTED LOOPS		16	992	17 (0)	00:00:01			Q1,00	PQWP	
8	NESTED LOOPS		20	992	17 (0)	00:00:01			Q1,00	PQWP	
9	PX BLOCK ITERATOR		2	114	15 (0)	00:00:01		1	5	Q1,00	PQWC
* 10	TABLE ACCESS FULL	TARGET	2	114	15 (0)	00:00:01	1	5	Q1,00	PQWP	
* 11	INDEX RANGE SCAN	DUMMY_IDX1	10		1 (0)	00:00:01			Q1,00	PQWP	
* 12	TABLE ACCESS BY INDEX ROWID	DUMMY	10	50	1 (0)	00:00:01			Q1,00	PQWP	

■ 미 출력 정보

Starts	: 조인 횟수	ABuffers	: 메모리에서 읽은 Block 수 메모리 관련 정보
A -Rows	: 실제 추출 건수	OMem, 1Mem, Used-Mem	: 메모리 관련 정보
A -Time	: 실제 수행 시간		

- Allstats 옵션을 적용하지 않을 경우 가장 중요한 정보 출력되지 않기때문에 사실상 필수 옵션이라 할 수 있다.

DBMS_XPLAN

STEP2. DISPLAY_CURSOR 출력 기준 / 내용 이해 (4)

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-Time	Pstart	Pstop	TQ	IN-OUT	PQ Distrib	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1			18 (100)							1	00:00:00.02	15			
1	PX COORDINATOR		1										1	00:00:00.02	15			
2	PX SEND QC (RANDOM)	:TQ10001	0	16	992	18 (6)	00:00:01			Q1,01	P->S	QC (RAND)	0	00:00:00.01	0			
3	HASH GROUP BY		0	16	992	18 (6)	00:00:01			Q1,01	PQMP		0	00:00:00.01	0	920K	920K	
4	PX RECEIVE		0	16	992	18 (6)	00:00:01			Q1,01	PQMP		0	00:00:00.01	0			
5	PX SEND HASH	:TQ10000	0	16	992	18 (6)	00:00:01			Q1,00	P->P	HASH	0	00:00:00.01	0			
6	HASH GROUP BY		0	16	992	18 (6)	00:00:01			Q1,00	PQMP		0	00:00:00.01	0	920K	920K	
7	NESTED LOOPS		0	16	992	17 (0)	00:00:01			Q1,00	PQMP		0	00:00:00.01	0			
8	NESTED LOOPS		0	20	992	17 (0)	00:00:01			Q1,00	PQMP		0	00:00:00.01	0			
9	PX BLOCK ITERATOR		0	2	114	15 (0)	00:00:01	1	5	Q1,00	PQWC		0	00:00:00.01	0			
* 10	TABLE ACCESS FULL	TARGET	0	2	114	15 (0)	00:00:01	1	5	Q1,00	PQMP		0	00:00:00.01	0			
* 11	INDEX RANGE SCAN	DUMMY_IDX1	0	10		1 (0)	00:00:01			Q1,00	PQMP		0	00:00:00.01	0			
12	TABLE ACCESS BY INDEX ROWID	DUMMY	0	10	50	1 (0)	00:00:01			Q1,00	PQMP		0	00:00:00.01	0			

Plan 정보 영역

ID	각 Operation의 ID Number
Operation	Row단위 Access 정보
Name	Access하는 Object 정보
Starts	해당 Operation 수행 횟수 (Nestead Loop 조인의 경우 Looping 횟수)
Cost (%CPU)	각 Operation의 Cost이며 Child Operation 들의 누계
E-Rows	해당 Operation을 통해 추출되는 Row 수 (예상)
A-Rows	해당 Operation을 통해 추출되는 Row 수 (실측)
A-Time	해당 Operation단계에서 수행된 시간. Second. (실측)

Plan 정보 영역

Pstart	Access Object가 Partition Table 인 경우 표기, 상수 값인 경우 Start Partition Key Number (변수 값인 경우 KEY로 표기)
Pstop	Access Object가 Partition Table 인 경우 표기, 상수 값인 경우 End Partition Key Number (변수 값인 경우 KEY로 표기)
Buffer	해당 Operation단계에서 실제 Buffer Cache를 통해 읽은 Block 량 (Logical Reads)
Read	해당 Operation단계에서 실제 Disk로부터 읽은 Block량 (Physical Reads)
OMem	Optimal 연산에 필요한 Memory Size (예상)
1Mem	One Pass 연산에 필요한 Memory Size (예상)
Used-Mem	실제로 사용된 Memory Size. (0:Optimal / 1:One Pass / 2이상:Multi Pass)

DBMS_XPLAN

STEP2. DISPLAY_CURSOR 출력 기준 / 내용 이해 (3)

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

Plan Column option	
cost	PLAN Column 중 COST(%CPU)정보 출력 제어.
rows	PLAN Column 중 E-Rows 정보 출력을 제어.
bytes	PLAN Column 중 E-Bytes 정보 출력을 제어.
partition	PLAN Column 중 Partition Start & Pstop 정보 출력을 제어 (PARTITION TABLE 조회시 해당 DATA 출력)
parallel	PLAN Column 중 TQ, IN-UT, PQ Distrib 정보 출력을 제어. (Parallel Query 수행시 해당 DATA 출력)

DBMS_XPLAN

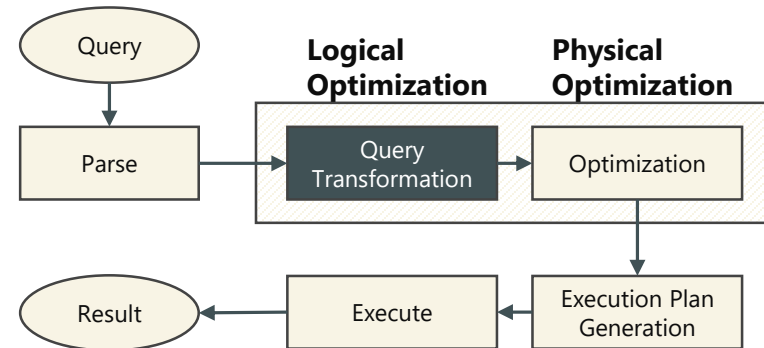
STEP3. 활용 (Query Block Name)

```
select /*+ gather_plan_statistics */
      *
from (select * from target where tn1 = :b1 ) a
     ,(select /*+ no_merge */ * from target where tn2 = :b2 ) b
where a.tn1 = b.tn2
```



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

```
1 - SEL$F5BB74E1
2 - SEL$F5BB74E1 / TARGET@SEL$2
3 - SEL$3        / B@SEL$1
4 - SEL$3
5 - SEL$3        / TARGET@SEL$3
6 - SEL$3        / TARGET@SEL$3
```



```
select *
from target a
     ,(select /*+ no_merge */ * from target where tn2 = :b2 ) b
where a.tn1 = b.tn2
and   a.tn1 = :b1
```

DBMS_XPLAN

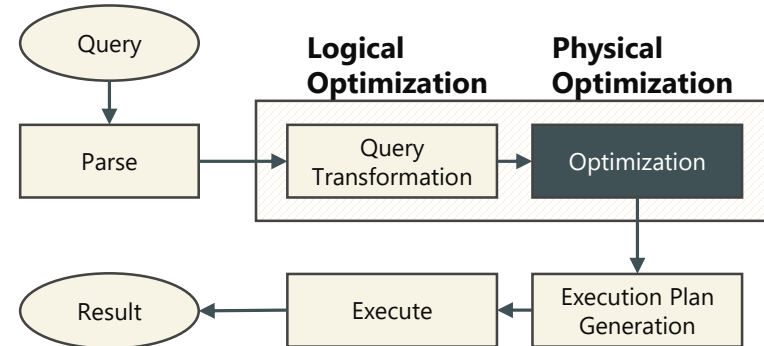
STEP3. 활용 (Peeked Binds)

```
select /*+ gather_plan_statistics */
      *
from (select * from target where tn1 = :b1 ) a
     ,(select /*+ no_merge */ * from target where tn2 = :b2 ) b
where a.tn1 = b.tn2
```



Peeked Binds (identified by position):

-
- 1 - (VARCHAR2(30), CSID=846): '5'
 - 2 - (VARCHAR2(30), CSID=846): '5'



Oracle Optimizer는 Peeked Bind 를 사용 하면
실행계획 생성과정에서 Histogram 통계가 존재 경우 Histogram 통계를 사용 합니다.

DBMS_XPLAN

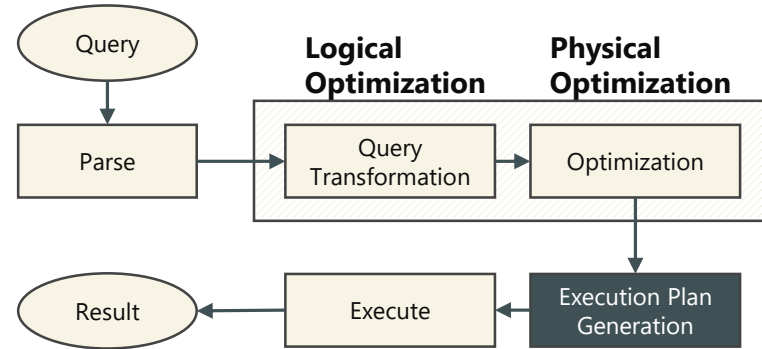
STEP3. 활용 (Outline Data)

```
select /*+ gather_plan_statistics */
*
from (select * from target where tn1 = :b1 ) a
, (select /*+ no_merge */ * from target where tn2 = :b2 ) b
where a.tn1 = b.tn2
```



Outline Data

```
/*+
BEGIN_OUTLINE_DATA
IGNORE_OPTIM_EMBEDDED_HINTS
OPTIMIZER_FEATURES_ENABLE('11.2.0.3')
DB_VERSION('11.2.0.3')
ALL_ROWS
OUTLINE_LEAF(@"SEL$1")
INDEX(@"SEL$1" "C"@"SEL$1" ("CHANNELS"."CHANNEL_ID"))
FULL(@"SEL$1" "S"@"SEL$1")
LEADING(@"SEL$1" "C"@"SEL$1" "S"@"SEL$1")
USE_HASH(@"SEL$1" "S"@"SEL$1")
END_OUTLINE_DATA
*/
```



Plan hash value: 1715085390

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		10	00:00:00.01	5
* 1	HASH JOIN		1	10	10	00:00:00.01	5
* 2	TABLE ACCESS FULL	TARGET	1	1	1	00:00:00.01	3
3	VIEW		1	10	10	00:00:00.01	2
* 4	FILTER		1		10	00:00:00.01	2
5	TABLE ACCESS BY INDEX ROWID	TARGET	1	10	10	00:00:00.01	2
* 6	INDEX RANGE SCAN	TARGET_IDX1	1	1	10	00:00:00.01	1

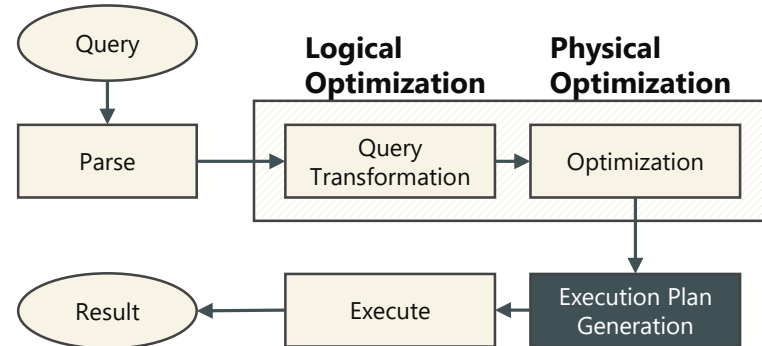
DBMS_XPLAN

STEP3. 활용 (Predicate Information)

```
select /*+ gather_plan_statistics */
      *
from (select * from target where tn1 = :b1 ) a
     ,(select /*+ no_merge */ * from target where tn2 = :b2 ) b
where a.tn1 = b.tn2
```

Predicate Information (identified by operation id):

- 1 - access("TARGET"."TN1"="B"."TN2")
- 2 - filter("TN1"=TO_NUMBER(:B1))
- 4 - filter(TO_NUMBER(:B1)=TO_NUMBER(:B2))
- 6 - access("TARGET"."TN2"=TO_NUMBER(:B1))
- filter("TN2"=TO_NUMBER(:B2))



Plan hash value: 1715085390

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		10	00:00:00.01	5
* 1	HASH JOIN		1	10	10	00:00:00.01	5
* 2	TABLE ACCESS FULL	TARGET	1	1	1	00:00:00.01	3
3	VIEW		1	10	10	00:00:00.01	2
* 4	FILTER		1		10	00:00:00.01	2
5	TABLE ACCESS BY INDEX ROWID	TARGET	1	10	10	00:00:00.01	2
* 6	INDEX RANGE SCAN	TARGET_IDX1	1	1	10	00:00:00.01	1

DBMS_XPLAN

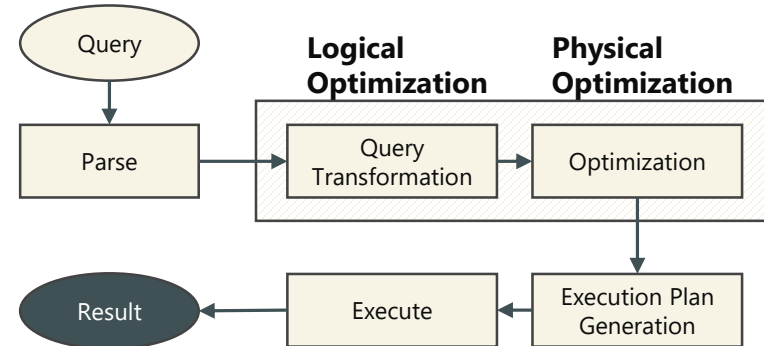
STEP3. 활용 (Column Projection)

```
select /*+ gather_plan_statistics */
      *
from (select * from target where tn1 = :b1 ) a
      ,(select /*+ no_merge */ * from target where tn2 = :b2 ) b
where a.tn1 = b.tn2
```



Column Projection Information (identified by operation id):

- 1 - (#keys=1) "TARGET"."TN1"[NUMBER,22], "B"."TN2"[NUMBER,22],
"TARGET"."TDT"[DATE,7], "TARGET"."TN2"[NUMBER,22], "TARGET"."TC1"[VARCHAR2,40],
"B"."TN1"[NUMBER,22], "B"."TDT"[DATE,7], "B"."TC1"[VARCHAR2,40]
- 2 - "TN1"[NUMBER,22], "TARGET"."TN2"[NUMBER,22], "TARGET"."TC1"[VARCHAR2,40],
"TARGET"."TDT"[DATE,7]
- 3 - "B"."TN1"[NUMBER,22], "B"."TN2"[NUMBER,22], "B"."TC1"[VARCHAR2,40],
"B"."TDT"[DATE,7]
- 4 - "TARGET"."TN1"[NUMBER,22], "TN2"[NUMBER,22], "TARGET"."TC1"[VARCHAR2,40],
"TARGET"."TDT"[DATE,7]
- 5 - "TARGET"."TN1"[NUMBER,22], "TN2"[NUMBER,22], "TARGET"."TC1"[VARCHAR2,40],
"TARGET"."TDT"[DATE,7]
- 6 - "TARGET".ROWID[ROWID,10], "TN2"[NUMBER,22], "TARGET"."TDT"[DATE,7]



TN1	TN2	TC1	TDT	TN1_1	TN2_1	TC1_1	TDT_1
5	5	5	2017-03-23 오후 2:20:29	5	5	5	2017-03-23 오후 2:20:29
5	5	5	2017-03-23 오후 2:20:29	15	5	15	2017-03-23 오후 2:20:29
5	5	5	2017-03-23 오후 2:20:29	25	5	25	2017-03-23 오후 2:20:29
5	5	5	2017-03-23 오후 2:20:29	35	5	35	2017-03-23 오후 2:20:29
5	5	5	2017-03-23 오후 2:20:29	45	5	45	2017-03-23 오후 2:20:29
5	5	5	2017-03-23 오후 2:20:29	55	5	55	2017-03-23 오후 2:20:29

DBMS_XPLAN

STEP3. 활용 (Parallel Query)

SQL_ID 5av1c6wms8dbs, child number 0

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-Time	Pstart	Pstop	TQ	IN-OUT	PQ Distrib	A-Rows	A-Time	Buffers	OMem	IMem	Used-Mem
0	SELECT STATEMENT		1			18 (100)							1	00:00:00.01	15			
1	PX COORDINATOR		1										1	00:00:00.01	15			
2	PX SEND QC (RANDOM)	:TQ10001	0	16	992	18 (6)	00:00:01			Q1,01	P->S	QC (RAND)	0	00:00:00.01	0			
3	HASH GROUP BY		0	16	992	18 (6)	00:00:01			Q1,01	POWP		0	00:00:00.01	0	920K	920K	
4	PX RECEIVE		0	16	992	18 (6)	00:00:01			Q1,01	POWP		0	00:00:00.01	0			
5	PX SEND HASH	:TQ10000	0	16	992	18 (6)	00:00:01			Q1,00	P->P	HASH	0	00:00:00.01	0			
6	HASH GROUP BY		0	16	992	18 (6)	00:00:01			Q1,00	POWP		0	00:00:00.01	0	920K	920K	
7	NESTED LOOPS		0	16	992	17 (0)	00:00:01			Q1,00	POWP		0	00:00:00.01	0			
8	NESTED LOOPS		0	20	992	17 (0)	00:00:01			Q1,00	POWP		0	00:00:00.01	0			
9	PX BLOCK ITERATOR		0	2	114	15 (0)	00:00:01	1	5	Q1,00	POWC		0	00:00:00.01	0			
* 10	TABLE ACCESS FULL	TARGET	0	2	114	15 (0)	00:00:01	1	5	Q1,00	POWP		0	00:00:00.01	0			
* 11	INDEX RANGE SCAN	DUMMY_IDX1	0	10		1 (0)	00:00:01			Q1,00	POWP		0	00:00:00.01	0			
12	TABLE ACCESS BY INDEX ROWID	DUMMY	0	10	50	1 (0)	00:00:01			Q1,00	POWP		0	00:00:00.01	0			

SQL_ID 5av1c6wms8dbs, child number 1

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-Time	Pstart	Pstop	TQ	IN-OUT	PQ Distrib	A-Rows	A-Time	Buffers	OMem	IMem	Used-Mem
0	SELECT STATEMENT		0			18 (100)							0	00:00:00.01	0			
1	PX COORDINATOR		0										0	00:00:00.01	0			
2	PX SEND QC (RANDOM)	:TQ10001	0	16	992	18 (6)	00:00:01			Q1,01	P->S	QC (RAND)	0	00:00:00.01	0			
3	HASH GROUP BY		0	16	992	18 (6)	00:00:01			Q1,01	POWP		0	00:00:00.01	0	1148K	1148K	467K (0)
4	PX RECEIVE		0	16	992	18 (6)	00:00:01			Q1,01	POWP		0	00:00:00.01	0			
5	PX SEND HASH	:TQ10000	0	16	992	18 (6)	00:00:01			Q1,00	P->P	HASH	0	00:00:00.01	0			
6	HASH GROUP BY		1	16	992	18 (6)	00:00:01			Q1,00	POWP		1	00:00:00.01	11	1148K	1148K	466K (0)
7	NESTED LOOPS		1	16	992	17 (0)	00:00:01			Q1,00	POWP		10	00:00:00.01	11			
8	NESTED LOOPS		1	20	992	17 (0)	00:00:01			Q1,00	POWP		10	00:00:00.01	10			
9	PX BLOCK ITERATOR		1	2	114	15 (0)	00:00:01	1	5	Q1,00	POWC		1	00:00:00.01	9			
* 10	TABLE ACCESS FULL	TARGET	3	2	114	15 (0)	00:00:01	1	5	Q1,00	POWP		1	00:00:00.01	9			
* 11	INDEX RANGE SCAN	DUMMY_IDX1	1	10		1 (0)	00:00:01			Q1,00	POWP		10	00:00:00.01	1			
12	TABLE ACCESS BY INDEX ROWID	DUMMY	10	10	50	1 (0)	00:00:01			Q1,00	POWP		10	00:00:00.01	1			

DBMS_XPLAN

STEP3. 활용 (추가 정보 확인)

DISPLAY_CURSOR + @

ID	OPERATION	NAME	STARTS	E-Rows	COST	TIME	A-Rows	A-Time	BUFFERS	WRITES	MB	BP	COMMENT	CREATED	LAST_AN
0	SELECT STATEMENT		1		3		1	5.210011	141803	0					
1	SORT AGGREGATE		1	1			1	5.210011	141803	0					
2	NESTED LOOPS		1	2	3	1	20971520	3.896045	141803	0					
*3	TABLE ACCESS BY INDEX ROWID	TARGET	1	1	2	1	40960	.027272	1432	0	.1	DEFAULT	타겟	20170320	
*4	INDEX RANGE SCAN	TARGET_IDX1	1	3	1	1	40960	.007324	196	0	.1	DEFAULT		20170320	20170320
*5	INDEX RANGE SCAN	DUMMY_IDX1	40960	2	1	1	20971520	1.787066	140371	0	.1	DEFAULT		20170320	20170320

Predicate Information

3 - filter(TO_NUMBER("T"."TC1")>5)
 4 - access("T"."TN2">5 AND "T"."TN2" IS NOT NULL)
 5 - access("T"."TN1"="D","DN2")

DBMS_XPLAN

STEP4. Optimizer 연결고리

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-Time	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1			3 (100)		1	00:00:05.21	141K	69
1	SORT AGGREGATE		1	1	61			1	00:00:05.21	141K	69
2	NESTED LOOPS		1	2	122	3 (0)	00:00:01	20M	00:00:03.90	141K	69
* 3	TABLE ACCESS BY INDEX ROWID	TARGET	1	1	48	2 (0)	00:00:01	40960	00:00:00.03	1432	69
* 4	INDEX RANGE SCAN	TARGET_IDX1	1	3		1 (0)	00:00:01	40960	00:00:00.01	196	69
* 5	INDEX RANGE SCAN	DUMMY_IDX1	40960	2	26	1 (0)	00:00:01	20M	00:00:01.79	140K	0



10053 EVENT

10046 Trace

DBMS_XPLAN

DBMS_MONITOR

DBMS_SQLTUNE

■ DBMS_MONITOR?

- **STEP1.** Format 이해
- **STEP2.** SubProgram 이해

DBMS_XPLAN

STEP1. Format 이해

```
PROCEDURE session_trace_enable(  
  session_id IN BINARY_INTEGER DEFAULT NULL,  
  serial_num IN BINARY_INTEGER DEFAULT NULL,  
  waits IN BOOLEAN DEFAULT TRUE,  
  binds IN BOOLEAN DEFAULT FALSE,  
  plan_stat IN VARCHAR2 DEFAULT NULL);
```

session_id

V\$SESSION.SID 값을 설정한다.

serial_num

V\$SESSION.SERIAL# 값을 설정한다.

waits

Wait 값 Trace 여부를 설정한다.

binds

Bind 값 Trace 여부를 설정한다.

plan_stat

first_execution, all_executions, never 3가지로 설정 가능하다.

LEVEL 1	→ plan_stat	파라미터 first_execution	설정
LEVEL 4	→ bind	파라미터 TURE	설정
LEVEL 8	→ waits	파라미터 TRUE	설정
LEVEL 16	→ plan_stat	파라미터 all_executions	설정
LEVEL 32	→ plan_stat	파라미터 never	설정

DBMS_XPLAN

STEP2. SubProgram 이해

PROCEDURE	DESCRIPTION	VERSION
client_id_stat_enable	client identifier 를 이용해 statistics 값의 수집을 활성화	10.1 이상
client_id_stat_disable	client identifier 를 이용해 statistics 값의 수집을 비활성화	10.1 이상
serv_mod_act_stat_enable	Service 명, Module 명, Action 명의 조합을 이용해 statistics 값의 수집을 활성화	10.1 이상
serv_mod_act_stat_disable	Service 명, Module 명, Action 명의 조합을 이용해 statistics 값의 수집을 비활성화	10.1 이상
client_id_trace_enable	client identifier 를 이용한 Trace 활성화	10.1 이상
client_id_trace_disable	client identifier 를 이용한 Trace 비활성화	10.1 이상
serv_mod_act_trace_enable	Service 명, Module 명, Action 명의 조합을 이용한 Trace 활성화	10.1 이상
serv_mod_act_trace_disable	Service 명, Module 명, Action 명의 조합을 이용한 Trace 비활성화	10.1 이상
session_trace_enable	sid,serial# 을 이용해 Trace 활성화	10.1 이상
session_trace_disable	sid,serial# 을 이용해 Trace 비활성화	10.1 이상
database_trace_enable	Database Trace 활성화	10.2 이상
database_trace_disable	Database Trace 비활성화	10.2 이상

10046 Trace

DBMS_XPLAN

DBMS_MONITOR

DBMS_SQLTUNE

■ DBMS_SQLTUNE?

- **STEP1.** Format 이해
- **STEP2.** 환경설정
- **STEP3.** Script 작성
- **STEP4.** Script 활용

DBMS_XPLAN

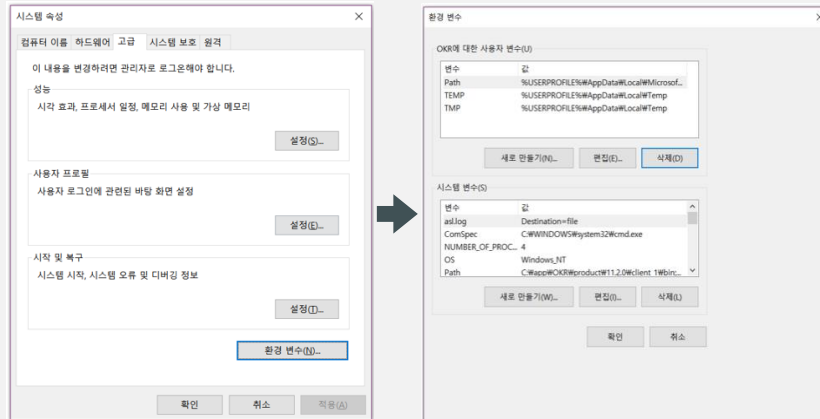
STEP1. SubProgram 이해

```

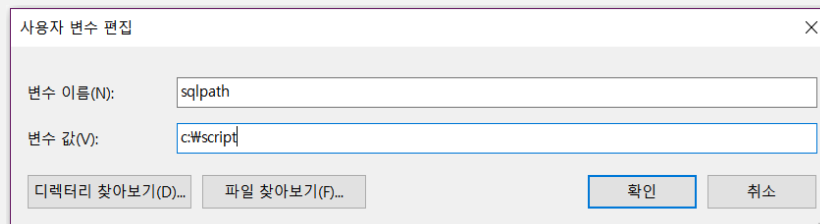
FUNCTION report_sql_monitor(
    sql_id          in varchar2 default NULL,
    session_id      in number   default NULL,
    session_serial  in number   default NULL,
    sql_exec_start  in date     default NULL,
    sql_exec_id     in number   default NULL,
    inst_id        in number   default NULL,
    start_time_filter in date     default NULL,
    end_time_filter  in date     default NULL,
    instance_id_filter in number default NULL,
    parallel_filter  in varchar2 default NULL,
    plan_line_filter in number   default NULL,
    event_detail     in varchar2 default 'yes',
    bucket_max_count in number   default 128,
    bucket_interval  in number   default NULL,
    base_path        in varchar2 default NULL,
    last_refresh_time in date     default NULL,
    report_level     in varchar2 default 'TYPICAL',
    type            in varchar2 default 'TEXT',
    sql_plan_hash_value in number default NULL)
RETURN clob;
    
```

DBMS_XPLAN

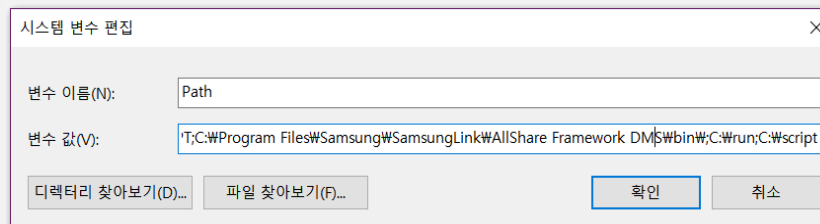
STEP2. 환경설정



사용자 변수에 SQLPATH 를 새로 만들기로 추가한다.



SQLPATH 변수의 변수 값은 .sql 스크립트 파일이 위치할 곳이다.



이제 생성한 SQLPATH 의 경로(변수 값)를 시스템 변수에 추가한다.
환경구성은 끝났다. 이제 어느 곳에서나 SQLPATH 경로 안에 있는 SQL 파일을 SQLPlus 에서 실행할 수 있게 되었다.

DBMS_SQLTUNE

STEP3. Script 작성

SM_SQL.SQL
SM_HASH.SQL
SM_SESS.SQL
SM_ACTIVE.SQL

```
SELECT sql_id AS sqlid
FROM   v$session
WHERE  sid = &1 ;
```

```
SELECT decode( UPPER( '&2' ) , 'T' , 'TEXT' , 'H' , 'HTML' , 'A' , 'ACTIVE' , 'TEXT' ) rtype
FROM   dual ;
```

```
SELECT TO_CHAR( SYSDATE , 'yyyymmdd' ) || '_' || '&1' ||
decode( UPPER( '&2' ) , 'T' , '_T.out' , 'H' , '_H.html' , 'A' , '_A.html' , '.out' ) file_str
FROM   dual ;
```

```
SELECT dbms_sqltune.report_sql_monitor( session_id => '&1' , report_level=>'ALL' , type=> '&rtype' )
FROM   dual ;
```

DBMS_SQLTUNE

STEP4. Script 활용

```

-----
Status          : EXECUTING
Instance ID     : 1
Session        : SH (132:277)
SQL ID         : c0y4tsbab59f4
SQL Execution ID : 16777216
Execution Started : 07/20/2015 12:41:07
First Refresh Time : 07/20/2015 12:41:07
Last Refresh Time : 07/20/2015 12:41:18
Duration       : 13s
Module/Action  : SH@ORCL1:132/Trial Until 2015-12-31
Service       : orcl.168.1.58
Program       : LitePlus.exe
    
```

Global Stats

Elapsed Time(s)	Cpu Time(s)	IO Waits(s)	Concurrency Waits(s)	Other Waits(s)	Buffer Gets	Read Reqs	Read Bytes
117	83	0.14	0.05	33	96030	859	750MB

Parallel Execution Details (DOP=4 , Servers Allocated=4)

Name	Type	Server#	Elapsed Time(s)	Cpu Time(s)	IO Waits(s)	Concurrency Waits(s)	Other Waits(s)	Buffer Gets	Read Bytes	Wait Events (sample #)
PX Coordinator	QC		0.18	0.11	0.01	0.05	0.01	381	1MB	
p000	Set 1	1	29	17	0.01		12	22354	175MB	
p001	Set 1	2	29	23	0.03		5.67	26744	210MB	
p002	Set 1	3	29	21	0.01		7.99	21342	168MB	
p003	Set 1	4	29	21	0.07		8.12	25209	197MB	

DBMS_SQLTUNE

STEP4. Script 활용 (TEXT TYPE)

SQL Plan Monitoring Details (Plan Hash Value=1103069905)

Id	Operation	Name	Execs	Rows (Actual)	Read Bytes	Activity (%)	Activity Detail (# samples)	Progress
0	SELECT STATEMENT		1					
1	SORT AGGREGATE		1					
2	PX COORDINATOR		5		472KB			
3	PX SEND QC (RANDOM)	:TQ10001	4					
-> 4	SORT AGGREGATE		4	0		69.23	Cpu (36)	
-> 5	HASH JOIN		4	158M		13.46	Cpu (7)	
6								
8	PX RECEIVE		4	328K				
9	PX SEND BROADCAST	:TQ10000	1	328K				
... 중간생략								
-> 13	PX BLOCK ITERATOR		4	3M		3.85	Cpu (2)	
-> 14	TABLE ACCESS FULL	PART_TEST2	17	3M	749MB	13.46	Cpu (7)	78%

DBMS_SQLTUNE

STEP4. Script 활용 (HTML TYPE)

SQL Text

SELECT /*+ monitor parallel(1, 4) parallel(c, 4)*/ MAX(c.time_id) FROM partition_test2 t, costs c WHERE t.time_id = c.time_id

Global Information: DONE (ALL ROWS)

Instance ID : 1
Session : SH (132, 277)
SQL ID : c0y4taba5994
SQL Execution ID : 16777216
Execution Started : 07/20/2015 12:41:07
First Refresh Time : 07/20/2015 12:41:07
Last Refresh Time : 07/20/2015 12:43:29
Duration : 142s
Module/Action : SH@ORCL1:132/Trial Unbl 2015-12-31
Service : orcl168.1.58
Program : LitePlus.exe
Fetch Calls : 1

Buffer Gets	IO Requests	Database Time	Wait Activity
416K	3417	553s	

Parallel Execution Details (DOP=4 , Servers Allocated=4)

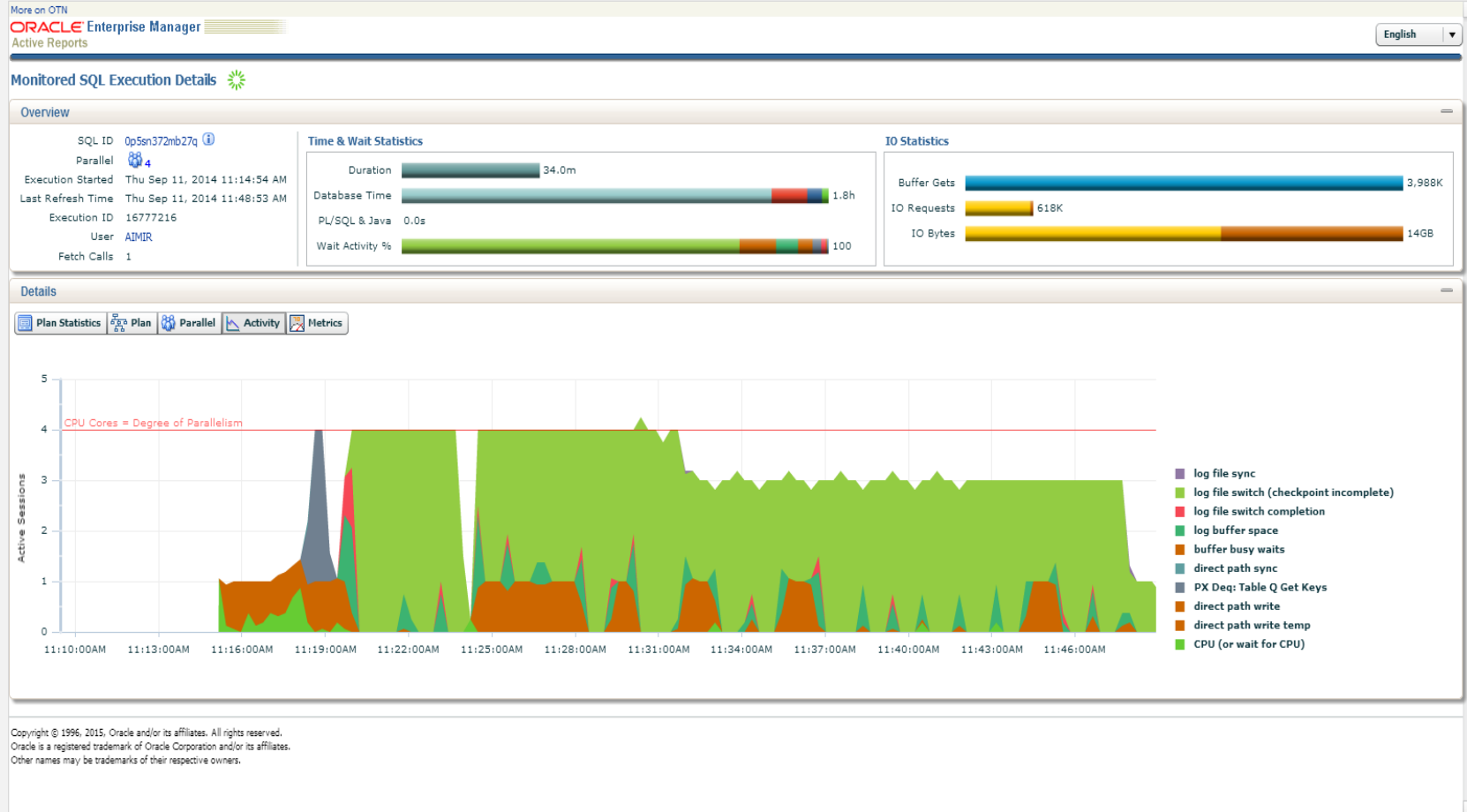
PX Processes Drill-Down					
Name	Type	Server	Buffer Gets	IO Requests	Database Time
PX Coordinator	QC	DB1 (<0.1%)		91 (2.7%)	0s (<0.1%)
p000	Sel 1	1	95503 (22%)	765 (22%)	136s (24%)
p001	Sel 1	2	116K (27%)	913 (26%)	135s (24%)
p002	Sel 1	3	112K (26%)	897 (26%)	142s (25%)
p003	Sel 1	4	93755 (22%)	751 (21%)	140s (25%)

SQL Plan Monitoring Details (Plan Hash Value=1103069905)

Id	Operation	Name	Estimated Rows	Cost	Active Period (142s)	ExeCs	Rows	Memory (Max)	Temp (Max)	IO Requests	CPU Activity	Wait Activity
0	SELECT STATEMENT					1	1					
1	SORT AGGREGATE					1	1					
2	PX COORDINATOR					5	4			99 (1.7%)		
3	PX SEND QC (RANDOM)	TQ10001				4	4					
4	SORT AGGREGATE					4	4				73%	
5	HASH JOIN		821K	4085.2		4	821K	28.6MB			16%	
6	BUFFER SORT					4	328K	12.7MB				
7	PART JOIN FILTER CREATE	BP0000	821K	31		4	328K					
8	PX RECEIVE		821K	31		4	328K					
9	PX SEND BROADCAST	TQ10000	821K	31		1	328K					
10	PARTITION RANGE ALL		821K	31		1	821K					
11	BITMAP CONVERSION TO ROWIDS		821K	31		20	821K					
12	BITMAP INDEX FAST FULL SCAN	COSTS_TIME_BIX				20	1480			92 (.9%)		
13	PX BLOCK ITERATOR		18K	2971.8		4	15K				1.1%	
14	TABLE ACCESS FULL	PARTITION_TEST2	18K	2971.8		60	15K			3326 (97%)	9.0%	

DBMS_SQLTUNE

STEP4. Script 활용 (ACTIVE TYPE)



STEP.4

Object 정보 분석 및 활용

SQL Tuning 고려 사항

“ 단위 SQL 성능 개선은 생각보다 쉽습니다.
하지만
해당 SQL을 실무에 적용하는것은 **생각보다 어렵습니다.** ”

- ✓ 얼마나 자주 사용하는 SQL인가
- ✓ 인덱스는 꼭 만들어야 하는가
- ✓ 업무적으로 중요도가 높은가
- ✓ 인덱스를 만들 공간은 충분한가
- ✓ 컬럼의 데이터 분포는 인덱스를 생성하기에 적합한가
- ✓ 바인드 변수 조회 패턴은 어떤가
- ✓ 인덱스를 생성했을 때, 다른 SQL의 실행계획에 영향을 주지는 않는가

SQL 튜닝 결과를 적용하기 위해서는 판단(SQL 튜닝)에 대한
수치화된 명확한 근거를 제시할 수 있어야 한다.

TABLE 구성정보 확인

NDV
Not Null
Density
Histogram
Column Comment
PK
Partition Key
Segment Size
Num Rows
Constraint
Index
etc ..



DBA_COL_COMMENTS
DBA_TAB_COLUMNS
DBA_INDEXES
DBA_CONSTRAINTS
DBA_IND_COLUMNS
DBA_TABLES
DBA_SEGMENTS
DBA_PART_KEY_COLUMNS
DBA_SUBPART_KEY_COLUMNS
DBA_TAB_SUBPARTITIONS
DBA_PART_TABLES
DBA_TAB_COLUMNS
DBA_CONSTRAINTS
DBA_INDEXES
DBA_PART_KEY_COLUMNS
DBA_OBJECTS
DBA_PART_INDEXES
DBA_IND_COLUMNS



@

TABLE 구성정보 확인

## TABLE INFO										
COLUMN_NAME	IS DATA_TYPE	DL N	NN	NUM_DISTINCT	DENSITY	ACL	NBK	SAMPLE_SIZE	Gram	COMMENT
CUST_ID	PK NUMBER	22 N	0	7,059	.0001417	5	1	918843	NONE	FK to the customers dimension table
AMOUNT_SOLD	NUMBER	22 N	0	3,586	.0002789	5	1	918843	NONE	invoiced amount to the customer
TIME_ID	DATE	7 N	0	1,460	.0006849	8	1	918843	NONE	FK to the times dimension table
PROD_ID	NUMBER	22 N	0	72	.0000005	4	72	918843	FREQU	FK to the products dimension table
CHANNEL_ID	NUMBER	22 N	0	4	.2500000	3	1	918843	NONE	FK to the channels dimension table
PROMO_ID	NUMBER	22 N	0	4	.2500000	4	1	918843	NONE	promotion identifier, without FK constraint (intentionally) to show outer join optimization
QUANTITY_SOLD	NUMBER	22 N	0	1	1.0000000	3	1	918843	NONE	product quantity sold with the transaction
TABLE_NAME	TABLESPACE_NAME		NUM_ROWS	BLOCKS	AVG_ROW_LEN	PARTITIONED	LAST_ANA	COMMENT		
SALES(SH)			918843	1907	29	YES	13/08/24	facts table, without a primary key ; all rows are uniquely identified by the combination of all foreign keys		
SEGMENT_NAME	PARTITION_NAME		SEGMENT_TYPE		TABLESPACE_NAME			MB BUFFER_POOL		
SALES	SALES_Q4_2001		TABLE PARTITION		EXAMPLE			8 DEFAULT		
SALES	SALES_Q4_2000		TABLE PARTITION		EXAMPLE			8 DEFAULT		
SALES	SALES_Q4_1999		TABLE PARTITION		EXAMPLE			8 DEFAULT		
SALES	SALES_Q4_1998		TABLE PARTITION		EXAMPLE			8 DEFAULT		
SALES	SALES_Q3_2001		TABLE PARTITION		EXAMPLE			8 DEFAULT		
SALES	SALES_Q3_2000		TABLE PARTITION		EXAMPLE			8 DEFAULT		
SALES	SALES_Q3_1999		TABLE PARTITION		EXAMPLE			8 DEFAULT		
SALES	SALES_Q3_1998		TABLE PARTITION		EXAMPLE			8 DEFAULT		
SALES	SALES_Q2_2001		TABLE PARTITION		EXAMPLE			8 DEFAULT		
SALES	SALES_Q2_2000		TABLE PARTITION		EXAMPLE			8 DEFAULT		
SALES	SALES_Q2_1999		TABLE PARTITION		EXAMPLE			8 DEFAULT		
SALES	SALES_Q2_1998		TABLE PARTITION		EXAMPLE			8 DEFAULT		

TABLE 구성정보 확인

```
=====
## PARTITION INFO
```

```
=====
TABLE_NAME      PART_TYPE  PART_COL      PART_CNT  SUB_TYPE  SUB_COL      SUB_CNT  DATA_DEFAULT
-----
SALES            RANGE      TIME_ID       28  NONE      0            0
```

```
=====
## CONSTRAINT INFO
```

```
=====
CONST_NAME      TYPE  CONDITION      R_OWNER      R_CONSTRAINT_NAME  R_TABLE      S  LAST_CHA
-----
SALES_CHANNEL_FK  R      "PROD_ID" IS NOT NULL  SH      CHANNELS_PK  CHANNELS      E  14/08/12
SALES_TIME_FK     R      "CUST_ID" IS NOT NULL  SH      TIMES_PK     TIMES         E  14/08/12
SALES_PROMO_FK    R      "TIME_ID" IS NOT NULL  SH      PROMO_PK     PROMOTIONS    E  14/08/12
SALES_PRODUCT_FK  R      "CHANNEL_ID" IS NOT NULL  SH      PRODUCTS_PK  PRODUCTS      E  14/08/12
SALES_CUSTOMER_FK R      "PROMO_ID" IS NOT NULL  SH      CUSTOMERS_PK  CUSTOMERS     E  14/08/12
SYS_C0011118      C      "PROD_ID" IS NOT NULL  SH      CUSTOMERS_PK  CUSTOMERS     E  14/08/12
SYS_C0011119      C      "CUST_ID" IS NOT NULL  SH      TIMES_PK     TIMES         E  14/08/12
SYS_C0011120      C      "TIME_ID" IS NOT NULL  SH      PROMO_PK     PROMOTIONS    E  14/08/12
SYS_C0011121      C      "CHANNEL_ID" IS NOT NULL  SH      PRODUCTS_PK  PRODUCTS      E  14/08/12
SYS_C0011122      C      "PROMO_ID" IS NOT NULL  SH      CUSTOMERS_PK  CUSTOMERS     E  14/08/12
```

```
=====
## INDEX INFO
```

```
=====
INDEX_NAME      TYPE  U  C  TABLESPACE_NAME      B      LB  DISTINCT_KEYS      CF  S      NUM_ROWS  LAST_ANA  PARTITIONED  T  G  BUFFER_POOL
-----
SALES_TIME_BIX  NEW  N  D      1      57      1460      1460  N      1460  13/08/24  YES      N  N  DEFAULT
SALES_PROMO_BIX NEW  N  D      1      30      4      54  N      54  13/08/24  YES      N  N  DEFAULT
SALES_CHANNEL_BIX NEW  N  D      1      47      4      92  N      92  13/08/24  YES      N  N  DEFAULT
SALES_CUST_BIX  NEW  N  D      1      452     7059     35808  N      35808  13/08/24  YES      N  N  DEFAULT
SALES_PROD_BIX  NEW  N  D      1      32      72      1074  N      1074  13/08/24  YES      N  N  DEFAULT
```

```
=====
CREATED  ISUNIQUE  LOCALITY  ALIGNMENT  PART_TYPE  SUB_T  PART_KEY_NAME      INDEX_NAME      COLUMN_NAME
-----
20140812 NONUNIQUE LOCAL  PREFIXED  RANGE      NONE  TIME_ID      SALES_TIME_BIX  TIME_ID
20140812 NONUNIQUE LOCAL  NON_PREFIXED RANGE      NONE  TIME_ID      SALES_PROMO_BIX  PROMO_ID
20140812 NONUNIQUE LOCAL  NON_PREFIXED RANGE      NONE  TIME_ID      SALES_CHANNEL_BIX  CHANNEL_ID
20140812 NONUNIQUE LOCAL  NON_PREFIXED RANGE      NONE  TIME_ID      SALES_CUST_BIX  CUST_ID
20140812 NONUNIQUE LOCAL  NON_PREFIXED RANGE      NONE  TIME_ID      SALES_PROD_BIX  PROD_ID
```

STEP. 5

업무정보 분석 및 활용

■ 최고의 튜닝

튜너 :

- 쿼리 변환, 인덱스 생성, 로직변경 등.. 난.. 역시... 훌륭해..
- 10초 -> 1초

업무 담당자 :

- [REDACTED]
- 10초 -> 0.01 초

튜닝이 어려운 쿼리는 ?

- 1) KEY 조건이 두개 이상의 테이블에 분리되어있는 경우
- 2) 테이블 자체가 Filter 성격의 집합으로 구성된 경우

※ 액세스 범위가 넓거나 추출 데이터가 많은 쿼리는 튜닝이 어려운 것이 아니라 업무적으로 조율이 필요한 것이다.

업무 설명 해주세요



튜너

...



업무 담당자

TOP TABLE

3

	TABLE_NAME	OWNER	TAB_SIZE	LR_READ	PR_READ	LR_PR_RATE	PART_TYPE	PART_COL	PART_CNT	SUB_TYPE	SUB_COL	SUB_CNT	IDX_CNT	IDX_SIZE	LAST_ANALYZED	COMMENTS
1	HDDPI_CGO_WTL	HDDSM	11.8G	583952256	542663525	92.9	RANGE	PCW_DSK_YMD	12	NONE		0	6	27.8G	2013-03-28 오후	화물작업내역
2	HDDPI_WBL_MTR#	HDDSM	6.8G	16365056	2428810	14.8	RANGE	PWM_DSK_YMD	12	NONE		0	25	16.1G	2013-04-21 오후	운송장
	HDDPI_WBL_LOG	HDDSM	4G									0	2	4G	2013-04-21 오후	운송장화물작업로그
	HDDOR_SMS_SUM	HDDSM	4G	112624	17872	15.9						0	5	20G	2013-04-24 오전	
	HDDUS_PRN_TMP2#	HDDSM	2.9G	3466224	3403165	98.2						0	3	447M	2013-12-23 오전	
	HDDPI_IBL_ETC	HDDSM	2.6G	51901376	6195039	11.9						0	18	4G	2013-03-29 오전	국제운송장 기타정보
	HDDUS_PRN_OKT#	HDDSM	2.5G	2965776	2950930	99.5						0	2	360M	2013-03-29 오전	
	HDDPI_CHG_HST	HDDSM	2.3G	15296	338	2.2						0	2	1.6G	2013-03-29 오전	운송장 수정 HISTORY
	HDDPI_OBS_DAT#	HDDSM	1.9G	1210144272	518608260	42.9						0	9	617M	2013-04-27 오후	
4	HDDOR_ADD_CVS	HDDSM	1.5G									0	7	2G	2013-03-29 오전	도로명(주소) 변환
	HDDUS_SSH_OTH#	HDDSM	1.3G	55984	9186	16.4						0	2	372M	2013-03-29 오전	
	HDDOR_ADD_CVS_TMP4	HDDSM	1.2G									0	10	1.9G		도로명(주소) 변환
	HDDPI_INV_PRN	HDDSM	1.2G	59552	6627	11.1						0	1	824M	2013-03-29 오전	Commercial INVOICE출력
	HDDOR_ADD_CVS_TMP3	HDDSM	1.1G									0	8	1.6G		도로명(주소) 변환
	HDDOR_RCV_MTR	HDDSM	1.1G									0	1	808M	2013-03-29 오전	EDI 접수통보/결과통보
	HDDUS_PRN_TMP#	HDDSM	1002M	392304	384089	97.9						0	1	66M	2013-03-29 오전	
	HDDPI_OPW_TMP	HDDSM	968M									0	11	19.1G	2013-04-26 오전	운송장 운영 실적
	HDDUS_PRN_TMP2_BAK	HDDSM	964M									0	1	50M	2013-03-29 오전	
	HDDWM_RAK_MTR	HDDSM	938M	136688	14353	10.5						0	1	818M	2013-03-29 오전	
	HDDUS_INV_TMP#	HDDSM	826M	635664	632793	99.5						0	2	150M	2013-03-29 오전	
	HDDRF_BLK_MTR	HDDSM	784M	10528868368	313460823	3						0	3	1.4G	2013-04-25 오후	

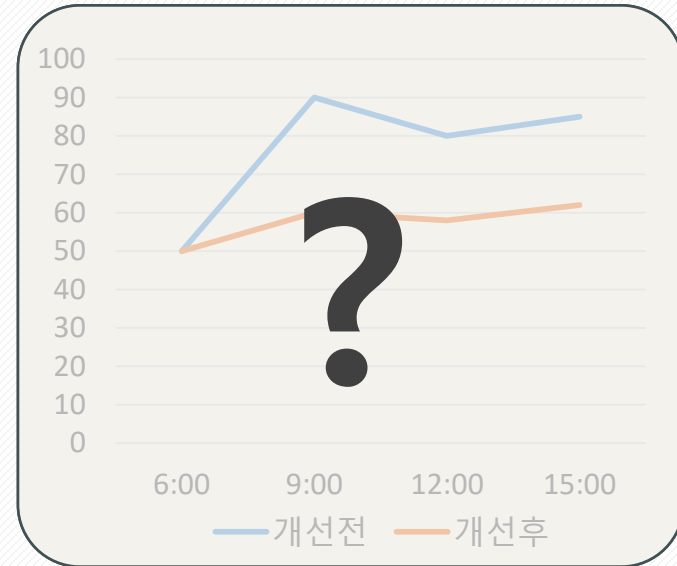
- 가장 큰 테이블이 11.8G 면 크지 않네.. 화물 작업 내역.. 이력 테이블이구나 일별 파티션 되어있고 Part Cnt 가 12 인걸 보니 일년치만 보관하는가? Logical Reads Physical Reads 비율이 높은걸로 봐서 주로 배치성 쿼리가 수행되겠네
- 인덱스가 25개 인걸 보면 온라인 쿼리가 주로 수행되겠군. 운송장이라.. Physical Reads 비율이 인덱스 개수에 비해서 좀 높은데? 일별 파티션 되어있는걸 보면 배치에서도 사용될 수 있겠다.
- 통계정보 정책은 따로 없는것 같고..
- 일량 정보가 없는 테이블이 있네.. 이름에 TMP가 있는건 그렇다고 치고.. 없는데 사용되지 않는건 뭘까..? TMP 인덱스는 많네..

TOP1 SQL

개선전 : 2.0초

개선후 : 0.1초

개선방법 : **인덱스 생성**



ACCESS PATTERN

SQL_ID	PHV	PSN	EXEC_T	BG	BG_1	DR_1	ROW_1	ELAPST_1	CPUT_1
dxghzm1u4qaay	1352726093	HDDSM	25285	3196527842	126420	88657.67	1	22.787	20.15
d4qd4t6hxbth5	605192182	HDDSM	495	184218906	372159	13539.43	8889	10.311	2.21
5x9jxkqysy848	960436637	HDDSM	95490	59869528	4	0.02	1	0	0
argy7y4j3x4nt	1015217574	HDDSM	50553	40645259	7	0.01	1	0	0
4w4kbsp0p730f	2904388702	HDDSM	114	373260	3274	608.46	0	0.51	0.062
6sr8dt1jf5r2c	2593896058	HDDSM	84	354763	4223	78.7	1	0.028	0.025
c5bsbzp4b4mdh	3452167378	HDDSM	1	353188	353188	33152	1	24.408	4.69
5f2dtgxzf7md9	3195384279	HDDSM	179	347548	1942	326.02	6	0.255	0.053
38rk2ucwu0cyk	489609065	HDDSM	1	217605	217605	18807	0	14.947	2.83

CAM_WBL_NUM	CAM_ARR_YMD	CAM_DPT_YMD	CAM_DLV_YMD	CAM_ARR_AIR	CAM_ACL_CLS	CAM_ACL_CNT	CAM_DLV_CNT	CAM_FNS_YN	CAM_DLV_CEN
{HA}=/{A}=	{A}<=/{F}<=/{F}<>/{A}=/{A}>=/{F}>=	{A}<=/{A}=/{F}=/{A}>=/{A}S NULL/{F}S NULL	{A}=	{A}=/{F}=	{F}=/{F}N	{F}<>/{F}=/{F}S NULL	{F}=/{F}>=/{F}N/{F}S NULL	{F}=/{F}S NULL	
{A}=								{F}S NULL	
{A}=								{F}S NULL	
{A}=								{F}S NULL	
		{A}<=/{A}>=	{F}S NULL						{HA}=
{A}=	{A}<=/{F}<=/{F}<>/{A}=/{A}>=/{F}>=	{A}<=/{A}=/{F}=/{A}>=/{A}S NULL/{F}S NULL	{A}=	{A}=/{F}=	{F}=/{F}N	{F}<>/{F}=/{F}S NULL	{F}=/{F}>=/{F}N/{F}S NULL	{F}=/{F}S NULL	
{A}=	{A}<=/{F}<=/{A}>=/{F}>=			{A}=/{F}=		{F}S NULL		{F}S NULL	
	{A}<=/{A}>=			{A}=					
	{A}<=/{A}>=			{A}=		{F}S NULL		{F}S NULL	

- 1 TOP1, TOP2 는 I/O 가 많네..
- 2 TOP3, TOP4 는 사용하는 조건이 CAM_WBL_NUM 하나인걸 보면.. 매우 심플한 쿼리겠군
- 3 인덱스에 추가하려는 컬럼은 TOP1 말고도 여러곳에서 사용되고, CAM_ARR_YMD, CAM_ARR_AIR 과 함께 사용되니 인덱스를 변경해도 되겠군
- 4 TOP4의 경우는 인덱스 CAL_FNS_YN 이 인덱스 선두 컬럼이 아니니까 상관 없고..
- 5 TOP5 부터는 우선순위가 좀 밀리니까.. 접어두자

SQL Tuning과 HINT의 관계

3
CHAPTER

 The Start of SQL Tuning Seminar

컨설팅 본부
www.ex-em.com
exem.tistory.com

CONTENTS

STEP. 1

HINT?

- HINT란?
- HINT의 사용 규칙
- HINT의 사용 목적

STEP. 2

HINT의 종류와 사용방법

- 조인 방법 관련
- Data Access 관련
- View 제어
- Subquery 제어
- Expansion 제어
- WITH절 제어
- PARALLEL 제어
- PARAMETER 제어

STEP. 3

HINT 적용 및 주의사항

- HINT 사용 시 주의사항
- Dynamic SQL의 HINT
- 적용의 중요성

STEP. 1

HINT?

HINT란 무엇인가?

- ✓ SQL 실행계획을 변경하는 방법 중 하나.
- ✓ 실행계획을 변경하는 방법 중 가장 능동적이며 강력한 방법.
- ✓ HINT만을 적용한다면, SQL에 변경을 주는 것이 아니므로 데이터 무결성의 오류가 발생하지 않음.

SQL 첫 번째 Command 이후에 기재

SQL 블록 첫 키워드

SELECT
INSERT
UPDATE
DELETE
MERGE

```
/*+ LEADING(A) USE_NL(A B) INDEX(A IDX_01) */
```

힌트 구문

테이블 명 혹은 테이블 별칭(Alias) 사용

테이블 명으로 적용한 힌트 예제

```
SELECT /*+ LEADING(a) INDEX(a IDX01) */  
       a.c3 ,  
       b.c3  
FROM   t1 a, t2 b  
WHERE  a.c1 = :b1  
AND    a.c2 = b.c2
```

테이블 별칭으로 적용한 힌트 예제

```
SELECT /*+ LEADING(T1) INDEX(T1 IDX01) */  
       t1.c3 ,  
       t2.c3  
FROM   t1, t2  
WHERE  t1.c1 = :b1  
AND    t1.c2 = t2.c2
```

바람직한 HINT 사용 vs 잘못된 HINT 사용

— 잘못 적용 된 HINT

```
SELECT /*+ SQL 설명 */  
      /*+ 힌트 구문 */  
      t1.c1 ,  
      t2.c1  
FROM   t1 ,  
      t2  
WHERE  t1.c1 = :b1  
AND    t1.c2 = t2.c2
```

— 잘 적용 된 HINT

```
SELECT /*+ 힌트 구문 */  
      /*+ SQL 설명 */  
      t1.c1 ,  
      t2.c1  
FROM   t1 ,  
      t2  
WHERE  t1.c1 = :b1  
AND    t1.c2 = t2.c2
```

```
SELECT /* SQL 설명 */  
      /*+ 힌트 구문 */  
      t1.c1 ,  
      t2.c1  
FROM   t1 ,  
      t2  
WHERE  t1.c1 = :b1  
AND    t1.c2 = t2.c2
```

통계정보에 의한 실행계획 이상

Bind Peeking에 의한 실행계획 이상

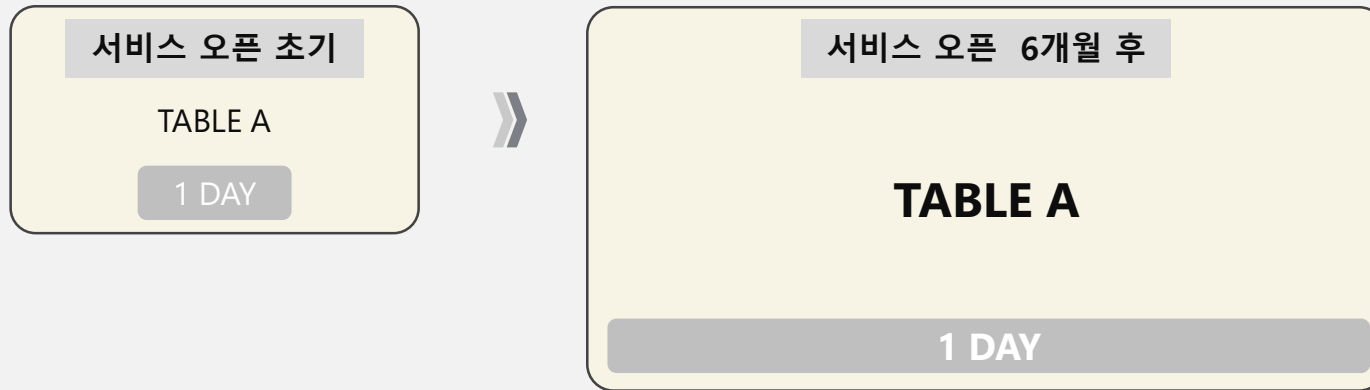
인덱스 구성 변경에 의한 실행계획 이상

Optimizer가 비효율 실행계획을 수립

- Optimizing 시 참조되는 데이터가 변경되거나 실제 테이블과 맞지 않게 수집된 경우, 비효율적인 실행계획을 수립하는데 악영향.
- 통계정보의 갱신이 필요함.
- 이미 수립되어 있는 수많은 SQL Plan의 변경이 불가피.
- 이 과정에서 효율적으로 수행되던 SQL의 성능에 좋지 않은 영향을 미칠 수 있음.
- 즉, 성능상 Side Effect가 발생할 수 있음.
- 이 경우, 통계정보 갱신 이전에 문제가 되는 SQL을 선별하여 HINT로 문제를 해결할 수 있음.

통계정보에 의한 실행계획 이상

실제 사례 (A 홈쇼핑 인터넷 쇼핑 업무)



- 통계정보는 서비스 오픈 초기에 생성하고 이후에 모든 테이블의 통계정보는 Lock!
- 서비스 오픈 6개월 후 하루 동안 쌓이는 데이터의 Size 측면에서 큰 차이가 발생함.
- Optimizer는 이런 차이를 감지하지 못하므로 실행계획의 변경은 없음.
- 서비스 오픈 초기 이후 점진적으로 I/O 발생량, CPU 사용량이 증가함.

통계정보에 의한 실행계획 이상

통계정보의 갱신이 어려운 경우, 특정 SQL에 HINT를 적용하여 비효율 해소

```
/* location: */
SELECT /*+ INDEX(A IMSI03)*/
COUNT(CCN_NO) AS CNT
FROM CUSTOM_INFO A
WHERE 1 = 1
AND MBR_NO = :1
AND ACCP_TP_CD IN
('21','22','31','90','91','94',
'96','97')
/* 이메일 */
AND CCN_PRGS_STAT_CD != '02'
/* 미완료 조건 */
AND ACCP_DTIME >= TRUNC
(ADD_MONTHS(SYSDATE, -1))
/* 신규 30일 조건 */
```

Execution Plan
SELECT STATEMENT CHOOSE-Cost: 3
SORT AGGREGATE
TABLE ACCESS BY INDEX ROWID EC_MGR.CS_CCN_INFO(1) Analyzed: 20130709 ("MBR_NO"=:1 AND "CCN_PRGS_STAT_CD" <> '02' AND ("ACCP_TP_CD"='21' OR "ACCP_TP_CD"='22' OR "ACCP_TP_CD"='31' OR "ACCP_TP_CD"='90' OR "ACCP_TP_CD"='91' OR "ACCP_TP_CD"='94' OR "ACCP_TP_CD"='96' OR "ACCP_TP_CD"='97'))
INDEX RANGE SCAN EC_MGR.CS_CCN_INFO_IDX06 (ACCP_DTIME) Analyzed: 20130709 ("ACCP_DTIME" >= TRUNC(ADD_MONTHS(SYSDATE@!, -1)))

통계정보 갱신 전 : ACCP_DTIME 기간 조건으로 INDEX RANGE SCAN을 수행함.

SESSION LOGICAL READS : 약 45만 blocks

Execution Plan
SELECT STATEMENT CHOOSE-Cost: 7
SORT AGGREGATE
TABLE ACCESS BY INDEX ROWID MAXGAUGE.XM_CS_CCN_INFO(1) Analyzed: 20131101 ("CCN_PRGS_STAT_CD" <> '02' AND "ACCP_DTIME" >= TRUNC(ADD_MONTHS(SYSDATE@!, -1)) AND "MBR_NO"=:1)
INDEX RANGE SCAN MAXGAUGE.IDX_IMSI03 (MBR_NO) Analyzed: 20131101 ("MBR_NO"=:1)

통계정보 갱신 후 : MBR_NO Equal 조건으로 INDEX RANGE SCAN을 수행함.

SESSION LOGICAL READS : 6 blocks

HINT의 사용 목적

통계정보에 의한 실행계획 이상

Bind Peeking에 의한 실행계획 이상

인덱스 구성 변경에 의한 실행계획 이상

Optimizer가 비효율 실행계획을 수립

- Bind Peeking 이란?
- 본연의 기능 자체는 보다 합리적인 실행계획에 큰 기여.
- 그러나, 성능상의 큰 변수가 존재.

Bind Peeking에 의한 실행계획 이상

Bind Peeking 기능의 함정 - CASE #1 수행 빈도 99%!

CASE #1

:B1 = '20170201'

:B2 = '20170202'

```
SELECT *  
FROM T_CUST  
WHERE JOIN_DATE BETWEEN :B1 AND :B2;
```

T_CUST

Total rows : 약 1억 건

INDEX SCAN이 성능상 유리 !!

Bind Peeking에 의한 실행계획 이상

Bind Peeking 기능의 함정 - CASE #2 수행 빈도 1% 이하

CASE #2

:B1 = '20160101'

:B2 = '20161231'

```
SELECT *  
FROM T_CUST  
WHERE JOIN_DATE BETWEEN :B1 AND :B2;
```

T_CUST

Total rows : 약 1억 건

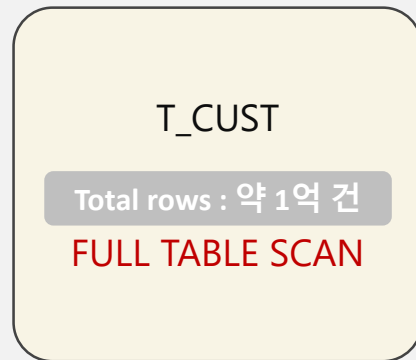
INDEX SCAN이 성능상 유리 ??



Bind Peeking에 의한 실행계획 이상

Bind Peeking 기능의 함정

- Hard Parsing시 입력된 Bind 값이 CASE #2와 같다면?
- T_CUST 테이블을 Table Full Scan 하는 실행계획이 수립될 것이고,
- 이후 1일에 해당하는 기간의 Bind 값이 입력된 SQL 수행 시에도 Table Full Scan으로 수행할 것이므로 큰 비효율 발생!



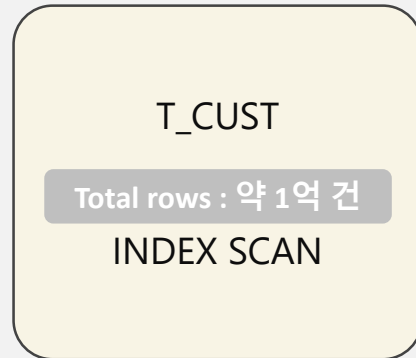
Access → TABLE FULL SCAN 수행 !!

Bind Peeking에 의한 실행계획 이상

Bind Peeking 기능의 함정

- 이 경우 힌트를 통한 적극적 개입을 통하여 비효율을 개선할 수 있음.

```
SELECT /*+ INDEX(A) */ *  
FROM T_CUST  
WHERE JOIN_DATE BETWEEN :B1 AND :B2;
```



Access → INDEX SCAN 수행 !!

HINT의 사용 목적

통계정보에 의한 실행계획 이상

Bind Peeking에 의한 실행계획 이상

인덱스 구성 변경에 의한 실행계획 이상

Optimizer가 비효율 실행계획을 수립

- 인덱스 구성 정보가 변경되면?
- 실행계획의 변경에 따른 비효율 실행계획이 수립될 여지가 있음.
- 힌트 적용이 필요.

HINT의 사용 목적

통계정보에 의한 실행계획 이상

Bind Peeking에 의한 실행계획 이상

인덱스 구성 변경에 의한 실행계획 이상

Optimizer가 비효율 실행계획을 수립

- Optimizer의 발전은 인정하지만, 완벽하지는 않음.
- Optimizer가 반영하지 못하는 부분이 존재.
- 힌트 적용이 필요.

STEP. 2

HINT의 종류와 사용방법

HINT의 종류와 사용방법 - 조인 순서 관련 HINT

ORDERED

LEADING

- FROM절에 나열된 테이블 순서대로 조인을 수행

ORDERED Hint 적용 전

```
SELECT t1.* ,
       t2.* ,
       t3.*
FROM   hint_t1 t1 ,
       hint_t2 t2 ,
       hint_t3 t3
WHERE  t1.cust_name = :b1
AND    t3.orddate BETWEEN :b2 AND :b3
AND    t1.cust_no = t2.cust_no
AND    t2.cust_no = t3.cust_no
```

Id	Operation	Name	Rows	Bytes	Time
0	SELECT STATEMENT				
* 1	FILTER				
* 2	HASH JOIN		4	276	00:00:33
* 3	HASH JOIN		96	4320	00:00:32
4	TABLE ACCESS FULL	HINT_T2	300K	5859K	00:00:03
5	TABLE ACCESS BY INDEX ROWID	HINT_T3	32	800	00:00:01
* 6	INDEX RANGE SCAN	IDX03_HINT_T3	32		00:00:01
* 7	TABLE ACCESS FULL	HINT_T1	3846	92304	00:00:02

HINT의 종류와 사용방법 - 조인 순서 관련 HINT

ORDERED

LEADING

- FROM절에 나열된 테이블 순서대로 조인을 수행

ORDERED Hint 적용 후 (EX 1)

```
SELECT /*+ ordered */
      t1.* ,
      t2.* ,
      t3.*
FROM   hint_t1 t1 ,
      hint_t2 t2 ,
      hint_t3 t3
WHERE  t1.cust_name = :b1
AND    t3.orddate BETWEEN :b2 AND :b3
AND    t1.cust_no = t2.cust_no
AND    t2.cust_no = t3.cust_no ;
```

Id	Operation	Name	Rows	Bytes	Time
0	SELECT STATEMENT				
* 1	FILTER				
* 2	TABLE ACCESS BY INDEX ROWID	HINT_T3	1	25	00:00:01
3	NESTED LOOPS		11538	777K	03:11:20
4	NESTED LOOPS		11538	495K	03:04:24
* 5	TABLE ACCESS FULL	HINT_T1	3846	92304	00:00:02
* 6	TABLE ACCESS FULL	HINT_T2	3	60	00:00:03
* 7	INDEX RANGE SCAN	IDX03_HINT_T3	32		00:00:01

ORDERED

LEADING

- FROM절에 나열된 테이블 순서대로 조인을 수행

ORDERED Hint 적용 후 (EX 2)

```
SELECT /*+ ordered */
      t1.* ,
      t2.* ,
      t3.*
FROM   hint_t3 t3 ,
      hint_t1 t1 ,
      hint_t2 t2
WHERE  t1.cust_name = :b1
AND    t3.orddate BETWEEN :b2 AND :b3
AND    t1.cust_no = t2.cust_no
AND    t2.cust_no = t3.cust_no ;
```

Id	Operation	Name	Rows	Bytes	Time
0	SELECT STATEMENT				
* 1	FILTER				
2	MERGE JOIN		11538	777K	00:02:09
3	SORT JOIN		123K	5889K	00:01:14
4	MERGE JOIN CARTESIAN		123K	5889K	00:00:37
5	TABLE ACCESS BY INDEX ROWID	HINT_T3	32	800	00:00:01
* 6	INDEX RANGE SCAN	IDX03_HINT_T3	32		00:00:01
7	BUFFER SORT		3846	92304	00:00:37
* 8	TABLE ACCESS FULL	HINT_T1	3846	92304	00:00:02
* 9	SORT JOIN		300K	5859K	00:00:55
10	TABLE ACCESS FULL	HINT_T2	300K	5859K	00:00:03

HINT의 종류와 사용방법 - 조인 순서 관련 HINT

ORDERED

LEADING

- 조인순서를 명시적으로 지정할 수 있음.
- 10g 이후 Driving Table만을 지정하던 것이 모든 테이블의 순서의 지정이 가능.

LEADING Hint 후 (EX 1)

```
SELECT /*+ leading(t1 t3 t2) */
       t1.* ,
       t2.* ,
       t3.*
FROM   hint_t1 t1 ,
       hint_t2 t2 ,
       hint_t3 t3
WHERE  t1.cust_name = :b1
AND    t3.orddate BETWEEN :b2 AND :b3
AND    t1.cust_no = t2.cust_no
AND    t2.cust_no = t3.cust_no ;
```

Id	Operation	Name	Rows	Bytes	Time
0	SELECT STATEMENT				
* 1	FILTER				
2	TABLE ACCESS BY INDEX ROWID	HINT_T2	1	20	00:00:01
3	NESTED LOOPS		11538	777K	01:16:16
4	NESTED LOOPS		123K	5889K	00:02:20
* 5	TABLE ACCESS FULL	HINT_T1	3846	92304	00:00:02
6	TABLE ACCESS BY INDEX ROWID	HINT_T3	32	800	00:00:01
* 7	INDEX RANGE SCAN	IDX03_HINT_T3	32		00:00:01
* 8	INDEX RANGE SCAN	IDX02_HINT_T2	1		00:00:01

— ORDERED vs. LEADING

- 10g 이후의 LEADING Hint는 전체 테이블의 조인 순서를 조정하는 것이 보다 간편.
- ORDERED Hint도 가능하지만 조인하려는 순서에 맞게 FROM절 테이블 순서를 조정해야 하는 불편함.
- 전체 테이블 조인 순서를 조정할 때에는 Hint의 의미가 직관적인 LEADING Hint를 사용하는 편이 더 효과적.

HINT의 종류와 사용방법 - 조인 방법 관련 HINT

NESTED LOOP JOIN

HASH JOIN

MERGE JOIN

SWAP JOIN INPUTS

- 일반적으로 조인 대상 건수가 적을 때 유리한 조인 방식.
- 조인 연결 컬럼에 반드시 인덱스가 존재하여야 함.
- **USE_NL**로 표현하며, 테이블 명이나 Alias를 적지하여야 함.

Nested Loop Join 유도 힌트 적용

```
SELECT /*+ leading(t2 t3) use_nl(t3) */ -- 조인순서 결정 힌트와 같이 사용
      t2.* ,
      t3.*
FROM   hint_t2 t2 ,
      hint_t3 t3
WHERE  t2.cust_addr=:b1
AND    t3.cust_no = t2.cust_no
AND    t3.orddate BETWEEN :b2
AND    :b3 ;
```

Id	Operation	Name	Rows	Bytes	Time
0	SELECT STATEMENT				
* 1	FILTER				
* 2	TABLE ACCESS BY INDEX ROWID	HINT_T3	1	25	00:00:01
3	NESTED LOOPS		37	1665	00:25:49
* 4	TABLE ACCESS FULL	HINT_T2	42857	837K	00:00:03
* 5	INDEX RANGE SCAN	IDX03_HINT_T3	32		00:00:01

HINT의 종류와 사용방법 - 조인 방법 관련 HINT

NESTED LOOP JOIN

HASH JOIN

MERGE JOIN

SWAP JOIN INPUTS

- 선행 테이블에서 많은 건수가 추출되고, 후행 테이블에 Access량이 많은 경우 성능상 유리함.
- USE_HASH로 표현하며, 테이블 명이나 Alias를 적지하여야 함.

Hash Join 유도 힌트 적용

```
SELECT /*+ leading(t2 t3) use_hash(t3) */
       t2.* ,
       t3.*
FROM   hint_t2 t2 ,
       hint_t3 t3
WHERE  t2.cust_addr=:b1
AND    t3.cust_no = t2.cust_no
AND    t3.orddate BETWEEN :b2
AND    :b3 ;
```

Id	Operation	Name	Rows	Bytes	Time
0	SELECT STATEMENT				
* 1	FILTER				
* 2	HASH JOIN		37	1665	00:00:08
* 3	TABLE ACCESS FULL	HINT_T2	42857	837K	00:00:03
4	TABLE ACCESS BY INDEX ROWID	HINT_T3	32	800	00:00:01
* 5	INDEX RANGE SCAN	IDX03_HINT_T3	32		00:00:01

HINT의 종류와 사용방법 - 조인 방법 관련 HINT

NESTED LOOP JOIN

HASH JOIN

MERGE JOIN

SWAP JOIN INPUTS

- 조인 방식의 특성상 추가적인 Sorting이 발생하여 거의 사용되지 않는 추세.
(Hash Join 방식이 성능상 유리함)
- **USE_MERGE**로 표현하며, 테이블 명이나 Alias를 적지하여야 함.

— Merge Join 유도 힌트 적용

```
SELECT /*+ leading(t2 t3) use_merge(t3) */
       t2.* ,
       t3.*
FROM   hint_t2 t2 ,
       hint_t3 t3
WHERE  t2.cust_addr=:b1
AND    t3.cust_no = t2.cust_no
AND    t3.orddate BETWEEN :b2
AND    :b3 ;
```

Id	Operation	Name	Rows	Bytes	Time
0	SELECT STATEMENT				
* 1	FILTER				
2	MERGE JOIN		37	1665	00:00:09
3	SORT JOIN		42857	837K	00:00:09
* 4	TABLE ACCESS FULL	HINT_T2	42857	837K	00:00:03
* 5	SORT JOIN		32	800	00:00:01
6	TABLE ACCESS BY INDEX ROWID	HINT_T3	32	800	00:00:01
* 7	INDEX RANGE SCAN	IDX03_HINT_T3	32		00:00:01

HINT의 종류와 사용방법 - 조인 방법 관련 HINT

NESTED LOOP JOIN

HASH JOIN

MERGE JOIN

SWAP JOIN INPUTS

- 일반적으로 조인의 순서가 결정된 Outer Join에서 조인 순서를 변경하는 힌트.
- 단, Hash Outer Join인 경우에만 조인 순서의 변경이 가능함.

Swap Join 힌트 적용 전

```
SELECT t1.* ,  
       t2.*  
FROM   hint_t1 t1 ,  
       hint_t2 t2  
WHERE  t1.cust_no = t2.cust_no( + ) ;
```

Id	Operation	Name	E-Rows	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT			232K	00:00:05.15	9342	1386
* 1	HASH JOIN OUTER		300K	232K	00:00:05.15	9342	1386
2	TABLE ACCESS FULL	HINT_T1	100K	100K	00:00:00.26	419	417
3	TABLE ACCESS FULL	HINT_T2	300K	300K	00:00:00.87	8923	969

HINT의 종류와 사용방법 - 조인 방법 관련 HINT

NESTED LOOP JOIN

HASH JOIN

MERGE JOIN

SWAP JOIN INPUTS

- 일반적으로 조인의 순서가 결정된 Outer Join에서 조인 순서를 변경하는 힌트.
- 단, Hash Outer Join인 경우에만 조인 순서의 변경이 가능함.

Swap Join 힌트 적용 후

```
SELECT /*+ SWAP_JOIN_INPUTS(T2) */
       t1.* ,
       t2.*
FROM   hint_t1 t1 ,
       hint_t2 t2
WHERE  t1.cust_no = t2.cust_no( + ) ;
```

Id	Operation	Name	E-Rows	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT			232K	00:00:04.98	10666	1384
* 1	HASH JOIN RIGHT OUTER		300K	232K	00:00:04.98	10666	1384
2	TABLE ACCESS FULL	HINT_T2	300K	300K	00:00:00.76	971	968
3	TABLE ACCESS FULL	HINT_T1	100K	100K	00:00:00.34	9695	416

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

TABLE FULL SCAN이 사용되는 경우

- 조인컬럼에 인덱스가 없고, 상수 조건도 없는 Hash Join으로 수행하는 경우
- Index Scan이 비효율이 클 경우
- Parallel Query로 Index Fast Full Scan으로 수행이 불가능한 경우
- Exadata의 Smart Scan으로 수행할 경우

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

■ 성능상 TABLE FULL SCAN을 사용하지 말아야 하는 경우

- SELECT COLUMN절에 있는 스칼라 서브쿼리
- Nested Loops Join으로 수행하는 후행 테이블
- Filter 방식으로 수행되는 서브쿼리
- 반복 수행하는 LOOP 구문내의 SQL

HINT의 종류와 사용방법 – DATA ACCESS 관련 HINT

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

- 힌트에 FULL과 함께 대상 테이블의 이름 혹은 Alias를 기재해야 함.

Table Full Scan 힌트 적용

```
SELECT /*+ FULL(T1) */  
      COUNT( * )  
FROM   hint_t1 t1  
WHERE  cust_id = 'ID_1' ;
```

Id	Operation	Name	Starts	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1	1	00:00:00.06	419	416
1	SORT AGGREGATE		1	1	00:00:00.06	419	416
* 2	TABLE ACCESS FULL	HINT_T1	1	10000	00:00:00.04	419	416

HINT의 종류와 사용방법 – DATA ACCESS 관련 HINT

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

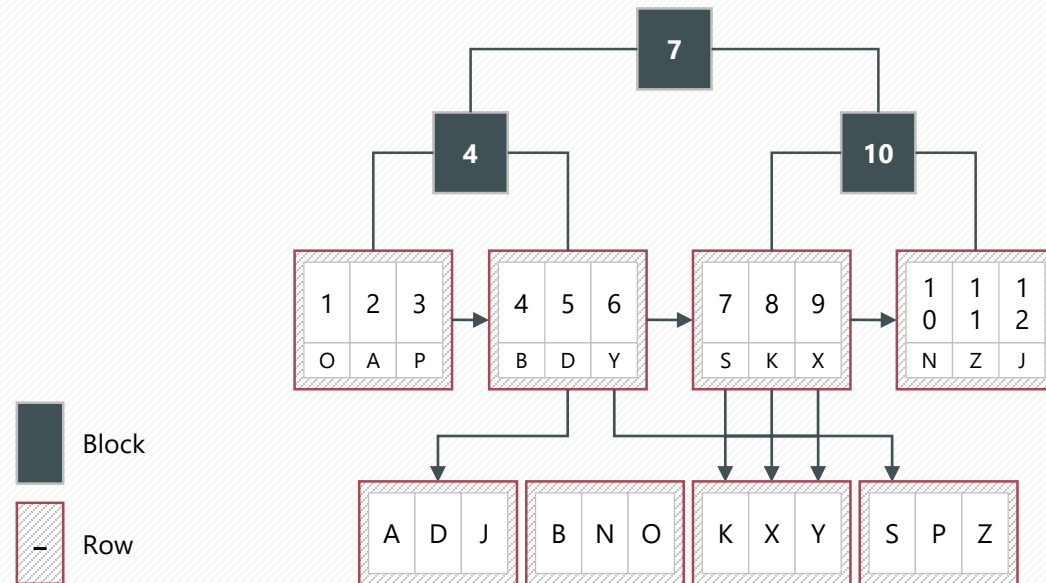
B*TREE INDEX COMBINATION

INDEX JOIN

INDEX RANGE SCAN

```
SELECT c2  
FROM t  
WHERE c1 >= 5  
AND c1 <= 9 ;
```

TABLE ACCESS BY INDEX ROWID
INDEX RANGE SCAN



HINT의 종류와 사용방법 – DATA ACCESS 관련 HINT

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

- Index를 경유하는 Scan의 대표적인 방법.
- INDEX(T1(COL1 COL2)) 힌트는 테이블에 COL1+COL2 순서로 결합 인덱스를 사용하도록 유도.
- 향후, 인덱스 명이 변경될 가능성이 있는 경우에 사용하면 효과적.

Index Range Scan 사용 예 (EX 1)

```
SELECT /*+ INDEX(T1 IDX03_HINT_T3) */  
-- OR /*+ INDEX_RS(T1 IDX03_HINT_T3) */  
  DISTINCT goods_no  
FROM   hint_t3 t1  
WHERE  t1.orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )  
AND    TO_DATE( '2013-01-30' , 'YYYY-MM-DD' ) ;
```

Id	Operation	Name	A-Time	Buffers	Reads
0	SELECT STATEMENT		00:00:00.01	4	4
1	HASH UNIQUE		00:00:00.01	4	4
2	TABLE ACCESS BY INDEX ROWID	HINT_T3	00:00:00.01	4	4
* 3	INDEX RANGE SCAN	IDX03_HINT_T3	00:00:00.01	3	3

HINT의 종류와 사용방법 – DATA ACCESS 관련 HINT

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

- Index를 경유하는 Scan의 대표적인 방법.
- INDEX(T1(COL1 COL2)) 힌트는 테이블에 COL1+COL2 순서로 결합 인덱스를 사용하도록 유도.
- 향후, 인덱스 명이 변경될 가능성이 있는 경우에 사용하면 효과적.

Index Range Scan 사용 예 (EX 2)

```
SELECT /*+ INDEX(T1(ORD_NO GOODS_NO)) */  
      *  
FROM   hint_t3 t1  
WHERE  ord_no = 10  
AND    t1.goods_no = '11000' ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1	00:00:00.01	4
1	TABLE ACCESS BY INDEX ROWID	HINT_T3	1	00:00:00.01	4
* 2	INDEX RANGE SCAN	IDX01_HINT_T3	1	00:00:00.01	3

HINT의 종류와 사용방법 – DATA ACCESS 관련 HINT

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

- 힌트에 기재된 Index를 내림차순 정렬을 처리하여 수행하도록 유도하는 힌트.
- 인덱스 스캔 방식은 기본적으로 정렬을 보장하므로 사용 가능.
- INDEX_DESC로 표현하며 대개, Paging처리나, MIN/MAX 값을 구하는 SQL에서 주로 사용됨.

Index Range Scan Descending 사용 예

```
SELECT *
FROM (
    SELECT /*+ INDEX_DESC(T1 IDX03_HINT_T3) */
        *
    FROM   hint_t3 t1
    WHERE  orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )
    AND    TO_DATE( '2013-01-31' , 'YYYY-MM-DD' )
    ORDER  BY orddate DESC
)
WHERE ROWNUM <= 10 ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		10	00:00:00.01	4
* 1	COUNT STOPKEY		10	00:00:00.01	4
2	TABLE ACCESS BY INDEX ROWID	HINT_T3	10	00:00:00.01	4
* 3	INDEX RANGE SCAN DESCENDING	IDX03_HINT_T3	10	00:00:00.01	3

HINT의 종류와 사용방법 – DATA ACCESS 관련 HINT

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

- Index Segment 전체를 읽는데 사용하는 가장 빠른 방법.
- INDEX_FFS 로 표현되며, Parallel 방식 수행도 가능.
- 정렬을 보장하지 않으며, SQL에서 사용되는 모든 컬럼이 인덱스 구성 컬럼이어야 함.

Index Fast Full Scan 사용 예

```
SELECT /*+ INDEX_FFS(T1 IDX01_HINT_T3) */  
-- Parallel 수행은 /*+ PARALLEL_INDEX(T1 IDX01_HINT_T3) */로 수행 가능  
  ord_no ,  
  goods_no  
FROM   hint_t3 t1  
WHERE  orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )  
AND    TO_DATE( '2014-12-31' , 'YYYY-MM-DD' )  
ORDER  BY ord_no ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		730	00:00:00.25	9851	9828
1	SORT ORDER BY		730	00:00:00.25	9851	9828
* 2	INDEX FAST FULL SCAN	IDX01_HINT_T3	730	00:00:00.24	9851	9828

HINT의 종류와 사용방법 – DATA ACCESS 관련 HINT

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

- 조건절에 인덱스 선두 컬럼이 사용되지 않더라도 Index Scan을 가능하게 해주는 방법. (Oracle 9i ~)
- 인덱스 선두 컬럼의 NDV(Number of Distinct Value)가 낮아야 유리함.
- INDEX_SS 로 표현되며, INDEX SKIP SCAN으로 유도하는 힌트

Index Skip Scan의 성능적 관점에서 유용성 TEST

```
CREATE TABLE sales_sum AS
SELECT ROWNUM custno ,
       '2011' ||lpad( CEIL( rownum/100000 ) , 2 , '0' ) yyyymm ,
       decode( MOD( ROWNUM , 12 ) , 1 , 'A' , 'B' ) gubun ,
       ROUND( dbms_random.value( 1000 , 100000 ) , - 2 ) price
FROM   dual
CONNECT BY LEVEL <= 100000 ;
```

- 2011년에 월 단위 매출을 산정하기 위해서 생성되었다고 가정.
- GUBUN 컬럼의 경우 MOD 함수를 사용하여 데이터를 생성하였는데, 'A' 값일 확률은 1/12로 'B'값에 비해 월등히 적은 데이터.
- 인덱스는 SALES_SUM_IDX1(yyyymm, gubun)가 존재.

HINT의 종류와 사용방법 – DATA ACCESS 관련 HINT

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

- 조건절에 인덱스 선두 컬럼이 사용되지 않더라도 Index Scan을 가능하게 해주는 방법. (Oracle 9i ~)
- 인덱스 선두 컬럼의 NDV(Number of Distinct Value)가 낮아야 유리함.
- INDEX_SS 로 표현되며, INDEX SKIP SCAN으로 유도하는 힌트

Index Skip Scan의 성능적 관점에서 유용성 TEST

```
SELECT /*+ Index_rs(s sales_sum_idx1) CASE #1 */
      COUNT( * )
FROM   sales_sum s
WHERE  gubun = 'A'
AND    yyyymm BETWEEN '201101'
AND    '201112' ;
```

Id	Operation	Name	Starts	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1	1	00:00:00.15	280	279
1	SORT AGGREGATE		1	1	00:00:00.15	280	279
* 2	INDEX RANGE SCAN	SALES_SUM_IDX1	1	8334	00:00:00.14	280	279

Predicate Information (identified by operation id):

```
2 - access("YYYMM">='201101' AND "GUBUN"='A' AND "YYYMM"<='201112')
    filter("GUBUN"='A')
```

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

— B*TREE INDEX COMBINATION 동작 방식

- STEP1. Index Scan을 통해 B*Tree Index의 Key와 Rowid 값을 읽는다.
- STEP2. Key 값과 Rowid 값을 Bitmap 값으로 변환한다.
- STEP3. Bit 연산을 통해 원하는 결과(Bitmap 값)를 도출한다.
- STEP4. 연산된 Bit 값을 다시 Rowid로 변환해서 원하는 데이터를 추출한다.

HINT의 종류와 사용방법 – DATA ACCESS 관련 HINT

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

- 하나의 테이블에 여러 인덱스의 사용이 가능.
- Bitmap Index를 흉내낸 기법.

B*Tree Index Combination 사용 예

```
SELECT *  
FROM   cust  
WHERE  성별 = '남'  
AND    지역 = '서울' ;
```

CUST

성별 = '남' : 10만 건

지역 = '서울' : 10만 건

성별 AND 지역 : 100만 건

SQL 설명 (가정)

- 성별 컬럼과 지역 컬럼에 각각 인덱스가 생성되어 있다.
- 성별 = '남'으로 추출되는 데이터는 10만건, 지역 = '서울'로 추출되는 데이터는 10만 건이며 두 조건을 만족하는 데이터는 100건이다.

HINT의 종류와 사용방법 – DATA ACCESS 관련 HINT

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

- 하나의 테이블에 여러 인덱스의 사용이 가능.
- Bitmap Index를 흉내낸 기법.

B*Tree Index Combination 사용 예

IDX02_HINT_T3 컬럼 : cust_no
IDX03_HINT_T3 컬럼 : ORDDATE

```
SELECT /*+ INDEX_COMBINE(T3 IDX02_HINT_T3 IDX03_HINT_T3) */  
      *  
FROM   hint_t3 t3  
WHERE  orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )  
AND    TO_DATE( '2013-06-30' , 'YYYY-MM-DD' )  
AND    cust_no BETWEEN '1'  
AND    '10' ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1	00:00:00.01	7	3
1	TABLE ACCESS BY INDEX ROWID	HINT_T3	1	00:00:00.01	7	3
2	BITMAP CONVERSION TO ROWIDS		1	00:00:00.01	6	3
3	BITMAP AND		1	00:00:00.01	6	3
4	BITMAP CONVERSION FROM ROWIDS		1	00:00:00.01	3	3
5	SORT ORDER BY		50	00:00:00.01	3	3
* 6	INDEX RANGE SCAN	IDX02_HINT_T3	50	00:00:00.01	3	3
7	BITMAP CONVERSION FROM ROWIDS		1	00:00:00.01	3	0
8	SORT ORDER BY		181	00:00:00.01	3	0
* 9	INDEX RANGE SCAN	IDX03_HINT_T3	181	00:00:00.01	3	0

HINT의 종류와 사용방법 – DATA ACCESS 관련 HINT

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

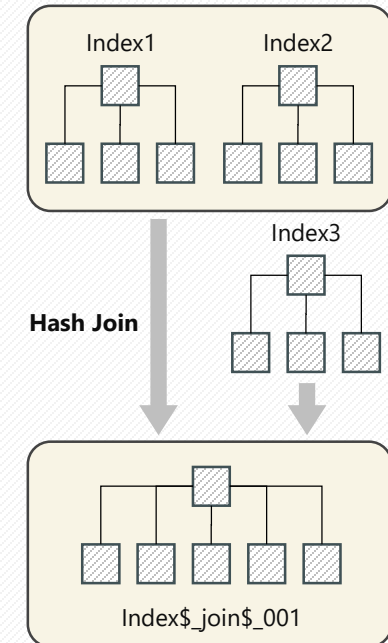
B*TREE INDEX COMBINATION

INDEX JOIN

INDEX JOIN

```
SELECT /*+ Index_join(t1 t1(c2) t1(c3)) */  
      COUNT( c1 )  
FROM   t1  
WHERE  t1.c2 BETWEEN 1 AND 100  
AND    t1.c3 BETWEEN 1 AND 10
```

SORT AGGREGATE VIEW	INDEX\$_join\$_001
HASH JOIN	
HASH JOIN	
INDEX RANGE SCAN	T1_I2
INDEX RANGE SCAN	T1_I3
INDEX FAST FULL SCAN	T1_I1



- T1_I2인덱스와 T1_I3인덱스를 Index Range Scan으로 각각 읽어서 Hash Join을 이용하여 Join.
- 위에서 Join한 결과를 T1_I1인덱스를 Index Fast Full Scan으로 읽으면서 Hash Join을 수행한다. 그 결과는 Index\$_Join\$_001이라는 Temporary Object에 저장.
- 제약 사항 : Table Random Access가 필요한 경우에는 INDEX JOIN을 수행할 수 없음.

HINT의 종류와 사용방법 – DATA ACCESS 관련 HINT

TABLE FULL SCAN

INDEX RANGE SCAN

INDEX RANGE SCAN DESCENDING

INDEX FAST FULL SCAN

INDEX SKIP SCAN

B*TREE INDEX COMBINATION

INDEX JOIN

- 하나의 테이블에 여러 인덱스를 사용할 수 있는 또 다른 방법.
- INDEX_JOIN으로 표현하며, 테이블 Alias와 인덱스 명을 동시에 기재.

Index Join 사용 예

```
SELECT /*+ INDEX_JOIN(T1 IDX02_HINT_T3 IDX03_HINT_T3) */
      COUNT( * )
FROM   hint_t3 t1
WHERE  orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )
AND    TO_DATE( '2013-06-30' , 'YYYY-MM-DD' )
AND    cust_no BETWEEN '1' AND '10' ;
```

Id	Operation	Name	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT			1	00:00:00.01	6
1	SORT AGGREGATE		1	1	00:00:00.01	6
* 2	VIEW	INDEX\$_join\$_001	1	1	00:00:00.01	6
* 3	HASH JOIN			1	00:00:00.01	6
* 4	INDEX RANGE SCAN	IDX02_HINT_T3	1	50	00:00:00.01	3
* 5	INDEX RANGE SCAN	IDX03_HINT_T3	1	181	00:00:00.01	3

VIEW 제어 힌트

- View란 FROM절의 또 다른 SQL Block(In-Line View)과 사용자에게 의해 생성된 View를 모두 포함함.
- View가 어떻게 수행되느냐에 따라 SQL의 성능에 영향을 줌.
- MERGE, NO_MERGE로 제어할 수 있음.

HINT의 종류와 사용방법 - VIEW 제어 HINT

MERGE

NO_MERGE

- In-Line View 또는 View를 View 외부 테이블과 Merging 되지 않고, 독립적으로 수행하고자 할 때 사용되는 힌트.
- 일반적으로 실행계획 상에 VIEW라는 Operation을 목격할 수 있음.

NO_MERGE 힌트가 필요한 상황

```
SELECT t1.* , t2.cost
FROM   t1 ,
      (
        SELECT ---> 추출 건수 : 100건 (그룹핑 후)
              id , SUM( cost ) AS cost
        FROM   t2
        GROUP  BY id
      ) t2
WHERE  t1.id = t2.id ;
```

- 조인순서는 T2 -> T1, 조인방법은 Nested Loops Join으로 수행.
- 인라인 뷰 T2는 Group By를 수행 후 100건 추출하고, 이 후 T1 테이블은 T2.ID 컬럼에 생성된 인덱스로 100회 수행 후 최종 데이터를 추출해야 성능상 문제가 없을 것으로 보임.
- 만약 인라인 뷰 T2가 T1과 VIEW MERGING이 되면, T2 구문내의 GROUP BY절이 T1 테이블과 조인을 수행 후 최종 데이터 추출 시점에 처리되어 성능문제를 유발.
- 이 경우에 NO_MERGE 힌트가 필요.

HINT의 종류와 사용방법 – VIEW 제어 HINT

MERGE

NO_MERGE

- In-Line View 또는 View를 View 외부 테이블과 Merging 되지 않고, 독립적으로 수행하고자 할 때 사용되는 힌트.
- 일반적으로 실행계획 상에 VIEW라는 Operation을 목격할 수 있음.

NO_MERGE 힌트 사용 예

```
SELECT t1.*
FROM   hint_t3 t1 ,
      (
        SELECT /*+ NO_MERGE */
              *
        FROM   hint_t2
        WHERE  addr_div = 0
      ) t2
WHERE  t1.cust_no = t2.cust_no
AND    t1.orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )
AND    TO_DATE( '2013-01-31' , 'YYYY-MM-DD' ) ;
```

Id	Operation	Name	A-Time	Buffers	Reads
0	SELECT STATEMENT		00:00:02.29	976	968
* 1	HASH JOIN		00:00:02.29	976	968
2	TABLE ACCESS BY INDEX ROWID	HINT_T3	00:00:00.01	4	0
* 3	INDEX RANGE SCAN	IDX03_HINT_T3	00:00:00.01	3	0
4	VIEW		00:00:01.16	972	968
* 5	TABLE ACCESS FULL	HINT_T2	00:00:00.41	972	968

■ SUBQUERY 제어 힌트

- Where절에 존재하는 또 다른 SQL Block을 Sub-Query라 함.
- Optimizer는 Sub-Query가 많을수록 Cost 계산이 복잡해짐.
- Filter 동작방식 vs. Join 동작방식
- 어떤 방식이 유리한지 각 상황에 맞게 Hint를 적용해야 함.

HINT의 종류와 사용방법 – SUBQUERY 제어 HINT

UNNEST

NO_UNNEST

HASH_SJ

HASH_AJ

- Join 동작방식(Semi or Anti Join)을 유도하는 힌트.

UNNEST 힌트 사용 예

```
SELECT t1.*
FROM   hint_t3 t1
WHERE  t1.orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )
AND    TO_DATE( '2013-01-31' , 'YYYY-MM-DD' )
AND    EXISTS (
        SELECT /*+ UNNEST */
              'X'
        FROM   hint_t2 t2
        WHERE  addr_div = 0
        AND    t1.cust_no = t2.cust_no
      ) ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		12	00:00:00.01	116
1	NESTED LOOPS SEMI		12	00:00:00.01	116
2	TABLE ACCESS BY INDEX ROWID	HINT_T3	31	00:00:00.01	4
* 3	INDEX RANGE SCAN	IDX03_HINT_T3	31	00:00:00.01	3
* 4	TABLE ACCESS BY INDEX ROWID	HINT_T2	12	00:00:00.01	112
* 5	INDEX RANGE SCAN	IDX02_HINT_T2	60	00:00:00.01	64

HINT의 종류와 사용방법 – SUBQUERY 제어 HINT

UNNEST

NO_UNNEST

HASH_SJ

HASH_AJ

- Filter 동작방식을 유도하는 힌트.

NO_UNNEST 힌트 사용 예

```
SELECT t1.*
FROM   hint_t3 t1
WHERE  t1.orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )
AND    TO_DATE( '2013-01-31' , 'YYYY-MM-DD' )
AND    EXISTS (
        SELECT /*+ NO_UNNEST */
              'X'
        FROM   hint_t2 t2
        WHERE  addr_div = 0
        AND    t1.cust_no = t2.cust_no
      ) ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		12	00:00:00.01	157
* 1	FILTER		12	00:00:00.01	157
2	TABLE ACCESS BY INDEX ROWID	HINT_T3	31	00:00:00.01	4
* 3	INDEX RANGE SCAN	IDX03_HINT_T3	31	00:00:00.01	3
* 4	TABLE ACCESS BY INDEX ROWID	HINT_T2	12	00:00:00.01	153
* 5	INDEX RANGE SCAN	IDX02_HINT_T2	60	00:00:00.01	93

HINT의 종류와 사용방법 – SUBQUERY 제어 HINT

UNNEST

NO_UNNEST

HASH_SJ

HASH_AJ

- EXISTS나 IN 조건을 사용한 경우 서브쿼리에 UNNEST와 함께 HASH_SJ 힌트를 부여하면 HASH JOIN SEMI로 처리.

— HASH_SJ 힌트 사용 예

```
SELECT t1.*
FROM   hint_t3 t1
WHERE  t1.orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )
AND    TO_DATE( '2013-01-31' , 'YYYY-MM-DD' )
AND    EXISTS (
        SELECT /*+ UNNEST HASH_SJ */
              'X'
        FROM   hint_t2 t2
        WHERE  addr_div = 0
        AND    t1.cust_no = t2.cust_no
      ) ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		12	00:00:01.54	975	968
* 1	HASH JOIN SEMI		12	00:00:01.54	975	968
2	TABLE ACCESS BY INDEX ROWID	HINT_T3	31	00:00:00.01	4	0
* 3	INDEX RANGE SCAN	IDX03_HINT_T3	31	00:00:00.01	3	0
* 4	TABLE ACCESS FULL	HINT_T2	150K	00:00:00.39	971	968

HINT의 종류와 사용방법 – SUBQUERY 제어 HINT

UNNEST

NO_UNNEST

HASH_SJ

HASH_AJ

- NOT EXISTS나 NOT IN 조건을 사용한 경우 서브쿼리에 UNNEST와 함께 HASH_AJ힌트를 사용하면 HASH JOIN ANTI로 처리하도록 제어.

HASH_AJ 힌트 사용 예

```
SELECT t1.*
FROM   hint_t3 t1
WHERE  t1.orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )
AND    TO_DATE( '2013-01-31' , 'YYYY-MM-DD' )
AND    NOT EXISTS (
        SELECT /*+ UNNEST HASH_AJ */
              'X'
        FROM   hint_t2 t2
        WHERE  addr_div = 0
        AND    t1.cust_no = t2.cust_no
      ) ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		19	00:00:01.51	975	968
* 1	HASH JOIN ANTI		19	00:00:01.51	975	968
2	TABLE ACCESS BY INDEX ROWID	HINT_T3	31	00:00:00.01	4	0
* 3	INDEX RANGE SCAN	IDX03_HINT_T3	31	00:00:00.01	3	0
* 4	TABLE ACCESS FULL	HINT_T2	150K	00:00:00.39	971	968

EXPANSION 제어 힌트

- Where절에 사용된 OR 조건은 SQL을 분기하여 실행하도록 하는 실행계획을 수립하는 힌트.
- “조건1 or 조건2” 형태의 조건과 각각의 컬럼에 효율적인 인덱스가 생성되어 있는 경우.
- 분기된 실행계획이 비효율이 많은 경우라면, 반대로 CONCATENATION이 되지 않게 SQL을 수행.
- USE_CONCAT와 NO_EXPAND의 힌트로 상황에 맞게 적용해야 함.

USE_CONCAT

NO_EXPAND

- OR 조건에 맞는 각 인덱스를 사용하여 CONCATENATION한 실행계획 수립하는데 사용하는 힌트.

USE_CONCAT 힌트 사용 예

```
SELECT /*+ USE_CONCAT(T3) */
      COUNT( * )
FROM   hint_t3 t3
WHERE  ( ord_no BETWEEN 1
        AND      100
        OR       orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )
        AND      TO_DATE( '2013-01-31' , 'YYYY-MM-DD' ) ) ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1	00:00:00.01	7
1	SORT AGGREGATE		1	00:00:00.01	7
2	CONCATENATION		100	00:00:00.01	7
3	TABLE ACCESS BY INDEX ROWID	HINT_T3	31	00:00:00.01	4
* 4	INDEX RANGE SCAN	IDX03_HINT_T3	31	00:00:00.01	3
* 5	INDEX RANGE SCAN	IDX01_HINT_T3	69	00:00:00.01	3

HINT의 종류와 사용방법 – EXPANSION 제어 HINT

USE_CONCAT

NO_EXPAND

- SQL 실행계획 분기가 불가하도록 유도.

— NO_EXPAND 힌트 사용 예

```
SELECT /*+ NO_EXPAND */
      COUNT( * )
FROM   hint_t3 t3
WHERE  ( ord_no BETWEEN 1
        AND      100
        OR       orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )
        AND      TO_DATE( '2013-01-31' , 'YYYY-MM-DD' ) ) ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1	00:00:00.01	6
1	SORT AGGREGATE		1	00:00:00.01	6
2	BITMAP CONVERSION COUNT		1	00:00:00.01	6
3	BITMAP OR		1	00:00:00.01	6
4	BITMAP CONVERSION FROM ROWIDS		1	00:00:00.01	3
5	SORT ORDER BY		100	00:00:00.01	3
* 6	INDEX RANGE SCAN	IDX01_HINT_T3	100	00:00:00.01	3
7	BITMAP CONVERSION FROM ROWIDS		1	00:00:00.01	3
8	SORT ORDER BY		31	00:00:00.01	3
* 9	INDEX RANGE SCAN	IDX03_HINT_T3	31	00:00:00.01	3

WITH절 제어 힌트

- With절은 동일한 데이터를 반복적으로 읽어 처리 하는 SQL의 성능을 개선하기 위해 주로 사용됨.
- Global Temporary Table (Materialize)을 사용하는 방식과 Inline 방식 두 가지 방식으로 사용됨.
- 결국 상황에 따라 Materialize방식과 Inline 방식을 선택하여 사용해야 함.

HINT의 종류와 사용방법 - WITH절 제어 HINT

MATERIALIZE

INLINE

- WITH절에서 추출된 데이터를 Global Temporary Table 저장 후 사용.

— MATERIALIZE 힌트 사용 예

```
WITH tmp AS (  
    SELECT /*+ MATERIALIZE */  
        -- 해당 QUERY BLOCK의 데이터를 Global Temp Table에 저장  
        cust_no  
    FROM    hint_t3  
    WHERE   orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )  
        AND   TO_DATE( '2013-01-31' , 'YYYY-MM-DD' )  
)  
SELECT t1.cust_no  
FROM tmp ,  
     hint_t1 t1  
WHERE tmp.cust_no = t1.cust_no  
AND   t1.cust_name = 'CUST_A'  
UNION ALL  
SELECT TO_NUMBER( t2.cust_no )  
FROM tmp ,  
     hint_t2 t2  
WHERE tmp.cust_no = t2.cust_no  
AND   t2.addr_div = 0 ;
```

HINT의 종류와 사용방법 - WITH절 제어 HINT

MATERIALIZE

INLINE

- WITH절에서 추출된 데이터를 Global Temporary Table 저장 후 사용.

MATERIALIZE 힌트 사용 예

Id	Operation	Name	A-Time	Buffers
0	SELECT STATEMENT		00:00:00.01	226
1	TEMP TABLE TRANSFORMATION		00:00:00.01	226
2	LOAD AS SELECT		00:00:00.01	8
3	TABLE ACCESS BY INDEX ROWID	HINT_T3	00:00:00.01	4
* 4	INDEX RANGE SCAN	IDX03_HINT_T3	00:00:00.01	3
5	UNION-ALL		00:00:00.01	215
6	NESTED LOOPS		00:00:00.01	61
7	NESTED LOOPS		00:00:00.01	33
8	VIEW		00:00:00.01	6
9	TABLE ACCESS FULL	SYS_TEMP_0FD9D662E_428F34	00:00:00.01	6
* 10	INDEX UNIQUE SCAN	IDX01_HINT_T1	00:00:00.01	27
* 11	TABLE ACCESS BY INDEX ROWID	HINT_T1	00:00:00.01	28
12	NESTED LOOPS		00:00:00.01	154
13	NESTED LOOPS		00:00:00.01	70
14	VIEW		00:00:00.01	4
15	TABLE ACCESS FULL	SYS_TEMP_0FD9D662E_428F34	00:00:00.01	4
* 16	INDEX RANGE SCAN	IDX02_HINT_T2	00:00:00.01	66
* 17	TABLE ACCESS BY INDEX ROWID	HINT_T2	00:00:00.01	84

HINT의 종류와 사용방법 - WITH절 제어 HINT

MATERIALIZE

INLINE

- WITH절을 INLINE VIEW로 실행되도록 유도하는 힌트.

— INLINE 힌트 사용 예

```
WITH tmp AS (  
    SELECT /*+ INLINE */  
           cust_no  
    FROM   hint_t3  
    WHERE  orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )  
    AND    TO_DATE( '2013-01-31' , 'YYYY-MM-DD' )  
)  
SELECT t1.cust_no  
FROM   tmp ,  
       hint_t1 t1  
WHERE  tmp.cust_no = t1.cust_no  
AND    t1.cust_name = 'CUST_A'  
UNION ALL  
SELECT TO_NUMBER( t2.cust_no )  
FROM   tmp ,  
       hint_t2 t2  
WHERE  tmp.cust_no = t2.cust_no  
AND    t2.addr_div = 0 ;
```

HINT의 종류와 사용방법 - WITH절 제어 HINT

MATERIALIZE

INLINE

- WITH절을 INLINE VIEW로 실행되도록 유도하는 힌트.

— INLINE 힌트 사용 예

Id	Operation	Name	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		37	00:00:00.01	215
1	UNION-ALL		37	00:00:00.01	215
2	NESTED LOOPS		1	00:00:00.01	59
3	NESTED LOOPS		28	00:00:00.01	31
4	TABLE ACCESS BY INDEX ROWID	HINT_T3	31	00:00:00.01	4
* 5	INDEX RANGE SCAN	IDX03_HINT_T3	31	00:00:00.01	3
* 6	INDEX UNIQUE SCAN	IDX01_HINT_T1	28	00:00:00.01	27
* 7	TABLE ACCESS BY INDEX ROWID	HINT_T1	1	00:00:00.01	28
8	NESTED LOOPS		36	00:00:00.01	156
9	NESTED LOOPS		84	00:00:00.01	72
10	TABLE ACCESS BY INDEX ROWID	HINT_T3	31	00:00:00.01	6
* 11	INDEX RANGE SCAN	IDX03_HINT_T3	31	00:00:00.01	4
* 12	INDEX RANGE SCAN	IDX02_HINT_T2	84	00:00:00.01	66
* 13	TABLE ACCESS BY INDEX ROWID	HINT_T2	36	00:00:00.01	84

■ PARALLEL 제어 힌트

- 병렬 처리 (Parallel Processin)로 데이터를 처리할 때 사용하는 힌트.
- Hint로 Degree(Slave Session) 수를 지정하고, 해당 수의 Slave Session들이 병렬 작업을 수행함.
- Multi Block I/O만을 수행하며 그 대상은 테이블 뿐만 아니라 인덱스 또한 가능하며, DML 작업도 가능.

HINT의 종류와 사용방법 - PARALLEL 제어 HINT

PARALLEL

PARALLEL_INDEX

- 테이블을 병렬처리 하도록 유도.

PARALLEL 힌트 사용 예

```
SELECT /*+ FULL(T3) PARALLEL(T3 4) */  
      COUNT( * )  
FROM   hint_t3 t3  
WHERE  goods_no IS NOT NULL ;
```

Id	Operation	Name	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT			1	00:00:12.98	5
1	SORT AGGREGATE		1	1	00:00:12.98	5
2	PX COORDINATOR			4	00:00:12.98	5
3	PX SEND QC (RANDOM)	:TQ10000	1	0	00:00:00.01	0
4	SORT AGGREGATE		1	0	00:00:00.01	0
5	PX BLOCK ITERATOR		2500K	0	00:00:00.01	0
* 6	TABLE ACCESS FULL	HINT_T3	2500K	0	00:00:00.01	0

HINT의 종류와 사용방법 – PARALLEL 제어 HINT

PARALLEL

PARALLEL_INDEX

- 인덱스를 병렬처리 하도록 유도. (Index Fast Full Scan)

PARALLEL_INDEX 힌트 사용 예

```
SELECT /*+ PARALLEL_INDEX(T3 IDX01_HINT_T3 4) */  
      COUNT( * )  
FROM   hint_t3 t3  
WHERE  goods_no IS NOT NULL ;
```

Id	Operation	Name	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT			1	00:00:12.07	5
1	SORT AGGREGATE		1	1	00:00:12.07	5
2	PX COORDINATOR			4	00:00:12.07	5
3	PX SEND QC (RANDOM)	:TQ10000	1	0	00:00:00.01	0
4	SORT AGGREGATE		1	0	00:00:00.01	0
5	PX BLOCK ITERATOR		2500K	0	00:00:00.01	0
* 6	INDEX FAST FULL SCAN	IDX01_HINT_T3	2500K	0	00:00:00.01	0

■ PARAMETER 제어 힌트

- Optimizer 관련 Parameter는 SQL 실행계획에 영향을 미치는 주요 요소 중 하나.
- 따라서, SQL의 실행계획 변경에 Optimizer 관련 Parameter를 변경하는 것도 방법.
- Parameter 설정 값을 변경하는 것은 DB 서버 전반에 영향을 미치므로 위험함.
- ORACLE 10.2.0.1 버전부터 SQL Level로 Optimizer 관련 Parameter 변경이 가능해짐.
- OPT_PARAM 힌트가 그 역할을 수행함.

ORA_PARAM

- SQL Level에서 Optimizer 관련 Parameter 변경을 가능케 해주는 힌트.

— OPT_PARAM 힌트 사용 예

```
SELECT /*+ INDEX_COMBINE(T1 IDX02_HINT_T3 IDX03_HINT_T3) */
      *
FROM   hint_t3 t1
WHERE  orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )
AND    TO_DATE( '2015-12-31' , 'YYYY-MM-DD' )
AND    cust_no BETWEEN '1'
AND    '10' ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1	00:00:00.01	9
* 1	TABLE ACCESS BY INDEX ROWID	HINT_T3	1	00:00:00.01	9
* 2	INDEX RANGE SCAN	IDX03_HINT_T3	1095	00:00:00.01	5

- Hint를 통해 B*Tree Index Combination을 수행하려 하였으나 Index Rang Scan이 수행됨.

ORA_PARAM

- SQL Level에서 Optimizer 관련 Parameter 변경을 가능케 해주는 힌트.

— OPT_PARAM 힌트 사용 예

```
SELECT a.ksppinm ,  
       b.ksppstvl  
FROM   x$ksppi a ,  
       x$ksppsv b  
WHERE  a.indx=b.indx  
AND    a.ksppinm LIKE '_b_tree_bitmap_plans' ;
```

KSPPINM	KSPPSTVL

_b_tree_bitmap_plans	FALSE

- 해당 Parameter 설정 값을 조회해 본 결과 기능이 False로 설정되어 있음을 확인.
- Parameter 설정 값을 변경하면 다른 SQL에 영향을 미치므로 해당 SQL Level에서만 B*Tree Index Combination이 동작하도록 힌트 사용.

ORA_PARAM

- SQL Level에서 Optimizer 관련 Parameter 변경을 가능케 해주는 힌트.

OPT_PARAM 힌트 사용 예

```
SELECT /*+ OPT_PARAM('_b_tree_bitmap_plans','TRUE') INDEX_COMBINE(T1
IDX02_HINT_T3 IDX03_HINT_T3) */
*
FROM   hint_t3 t1
WHERE  orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )
AND    TO_DATE( '2015-12-31' , 'YYYY-MM-DD' )
AND    cust_no BETWEEN '1' AND '10' ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1	00:00:00.01	9
1	TABLE ACCESS BY INDEX ROWID	HINT_T3	1	00:00:00.01	9
2	BITMAP CONVERSION TO ROWIDS		1	00:00:00.01	8
3	BITMAP AND		1	00:00:00.01	8
4	BITMAP CONVERSION FROM ROWIDS		1	00:00:00.01	3
5	SORT ORDER BY		50	00:00:00.01	3
* 6	INDEX RANGE SCAN	IDX02_HINT_T3	50	00:00:00.01	3
7	BITMAP CONVERSION FROM ROWIDS		1	00:00:00.01	5
8	SORT ORDER BY		1095	00:00:00.01	5
* 9	INDEX RANGE SCAN	IDX03_HINT_T3	1095	00:00:00.01	5

ORA_PARAM

- SQL Level에서 Optimizer 관련 Parameter 변경을 가능케 해주는 힌트.

OPT_PARAM 힌트 사용 예

```
SELECT t1.* ,
       t2.* ,
       t3.*
FROM   hint_t1 t1 ,
       hint_t2 t2 ,
       hint_t3 t3
WHERE  t1.cust_name = 'CUST_A'
AND    t3.orddate BETWEEN TO_DATE(:b1, 'YYYY-MM-DD')
AND    TO_DATE(:b2, 'YYYY-MM-DD')
AND    t1.cust_no = t2.cust_no
AND    t2.cust_no = t3.cust_no ;
```

:B1 := '2013-01-01'
:B2 := '2013-01-31'

- 조인순서
- 조인 방법
- 데이터 Access 방법

Id	Operation	Name	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		3	00:00:03.43	1395
* 1	FILTER		3	00:00:03.43	1395
* 2	HASH JOIN		3	00:00:03.43	1395
3	TABLE ACCESS BY INDEX ROWID	HINT_T3	31	00:00:00.01	4
* 4	INDEX RANGE SCAN	IDX03_HINT_T3	31	00:00:00.01	3
* 5	HASH JOIN		6810	00:00:03.37	1391
* 6	TABLE ACCESS FULL	HINT_T1	3846	00:00:00.03	420
7	TABLE ACCESS FULL	HINT_T2	300K	00:00:00.82	971

ORA_PARAM

- SQL Level에서 Optimizer 관련 Parameter 변경을 가능케 해주는 힌트.

OPT_PARAM 힌트 사용 예

```
SELECT /*+ LEADING(T3 T2 T1) USE_NL(T1 T2) INDEX(T1 IDX01_HINT_T1) INDEX(T2
  IDX02_HINT_T2) */
      t1.* , t2.* , t3.*
FROM   hint_t1 t1 ,
      hint_t2 t2 ,
      hint_t3 t3
WHERE  t1.cust_name = 'CUST_A'
AND    t3.orddate BETWEEN TO_DATE( :b1 , 'YYYY-MM-DD' )
AND    TO_DATE( :b2 , 'YYYY-MM-DD' )
AND    t1.cust_no = t2.cust_no
AND    t2.cust_no = t3.cust_no ;
```

Id	Operation	Name	Starts	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1	3	00:00:00.01	245
* 1	FILTER		1	3	00:00:00.01	245
2	NESTED LOOPS		1	3	00:00:00.01	245
3	NESTED LOOPS		1	84	00:00:00.01	161
4	NESTED LOOPS		1	84	00:00:00.01	152
5	TABLE ACCESS BY INDEX ROWID	HINT_T3	1	31	00:00:00.01	4
* 6	INDEX RANGE SCAN	IDX03_HINT_T	1	31	00:00:00.01	3
7	TABLE ACCESS BY INDEX ROWID	HINT_T2	31	84	00:00:00.01	148
* 8	INDEX RANGE SCAN	IDX02_HINT_T2	31	84	00:00:00.01	64
* 9	INDEX UNIQUE SCAN	IDX01_HINT_T1	84	84	00:00:00.01	9
* 10	TABLE ACCESS BY INDEX ROWID	HINT_T1	84	3	00:00:00.01	84

STEP. 3

HINT 적용 및 주의사항

■ HINT 사용 시 주의사항

- 의도와 다르게 실행계획이 변경되지 않아야 한다.
- Index Naming Rule.
- 데이터 변경이 심한 경우 힌트 사용에 주의해야 한다.
- 힌트가 적용된 프로그램 유지보수에 각별한 주의가 필요하다.
- Dynamic SQL인 경우, 힌트 사용을 보다 효율적으로 적용해야 한다.

Dynamic SQL의 힌트 적용의 중요성

Dynamic SQL 힌트 적용 예 – SQL 성능 점검

- 이미 /*+ FULL(T3) */ 힌트가 적용되어 있음.
- 테이블 전체를 읽는 것보다 조건절의 한달 가량의 데이터만을 Access 하는 것을 효율적이라 판단.
- 따라서, 기존 힌트 대신에 Index Scan이 가능한 힌트로 대체.

```
SELECT /*+ FULL(T3) */  
      COUNT( * )  
FROM   hint_t3 t3  
WHERE  1 = 1  
AND    goods_no BETWEEN '1'  
AND    '100'  
AND    orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )  
AND    TO_DATE( '2013-01-31' , 'YYYY-MM-DD' ) ;
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1		1	00:00:00.14	10511	10507
1	SORT AGGREGATE		1	1	1	00:00:00.14	10511	10507
* 2	TABLE ACCESS FULL	HINT_T3	1	1	1	00:00:00.14	10511	10507

Dynamic SQL의 힌트 적용의 중요성

Dynamic SQL 힌트 적용 예 – 새로운 힌트 적용 후

- 기존 Table Full Scan 방식에 비해 I/O 발생량, 응답시간 측면에서 모두 개선된 결과를 보임.
- 단, 해당 SQL이 Dynamic SQL이고, 개선자가 이 사실을 몰랐다면?

```
SELECT /*+ INDEX(T3 IDX03_HINT_T3) */  
--- 수행방식을 Table Full Scan에서 Index로 변경  
    COUNT( * )  
FROM   hint_t3 t3  
WHERE  1 = 1  
AND    goods_no BETWEEN '1'  
AND    '100'  
AND    orddate BETWEEN TO_DATE( '2013-01-01' , 'YYYY-MM-DD' )  
AND    TO_DATE( '2013-01-31' , 'YYYY-MM-DD' ) ;
```

Id	Operation	Name	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT			1	00:00:00.01	4
1	SORT AGGREGATE		1	1	00:00:00.01	4
* 2	TABLE ACCESS BY INDEX ROWID	HINT_T3	1	1	00:00:00.01	4
* 3	INDEX RANGE SCAN	IDX03_HINT_T3	31	31	00:00:00.01	3

Dynamic SQL의 힌트 적용의 중요성

Dynamic SQL 힌트 적용 예 – 새로운 힌트 적용 후

- 변경된 조건으로 인하여 기존보다 비효율이 크게 증가!
- 기존과 같이 Table Full Scan으로 계속 수행되었다면, 오히려 어느 정도의 성능은 보장되었을 것!
- 상황에 맞게 힌트를 적용하지 않는다면 이와 같은 낭패를 볼 수 있음!

```
SELECT /*+ INDEX(T3 IDX03_HINT_T3) */  
      COUNT( * )  
FROM   hint_t3 t3  
WHERE  1 =1  
AND    goods_no BETWEEN '1'  
AND    '100'  
AND    cust_no BETWEEN '1'  
AND    '100'  
AND    orddate >= TO_DATE( '2013-01-01' , 'YYYY-MM-DD' ) ;
```

Id	Operation	Name	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1	00:00:13.64	17143	17139
1	SORT AGGREGATE		1	00:00:13.64	17143	17139
* 2	TABLE ACCESS BY INDEX ROWID	HINT_T3	50	00:00:13.64	17143	17139
* 3	INDEX RANGE SCAN	IDX03_HINT_T3	2500K	00:00:06.71	6636	6633

Dynamic SQL 튜닝 방법

4

CHAPTER

CONTENTS

STEP. 1

Dynamic SQL이란?

- SQL 작성 방법의 변화

STEP. 2

Dynamic SQL 튜닝 진행방식

- SQL 조회 조건 파악
- SQL 조회 조건 분석
- 조회 조건 별 효율성 체크
- XML 소스 수정 및 개선안 적용
- 개선안 적용 후 모니터링

STEP. 3

Dynamic SQL 튜닝 적용시 주의사항

- 적용방법의 중요성

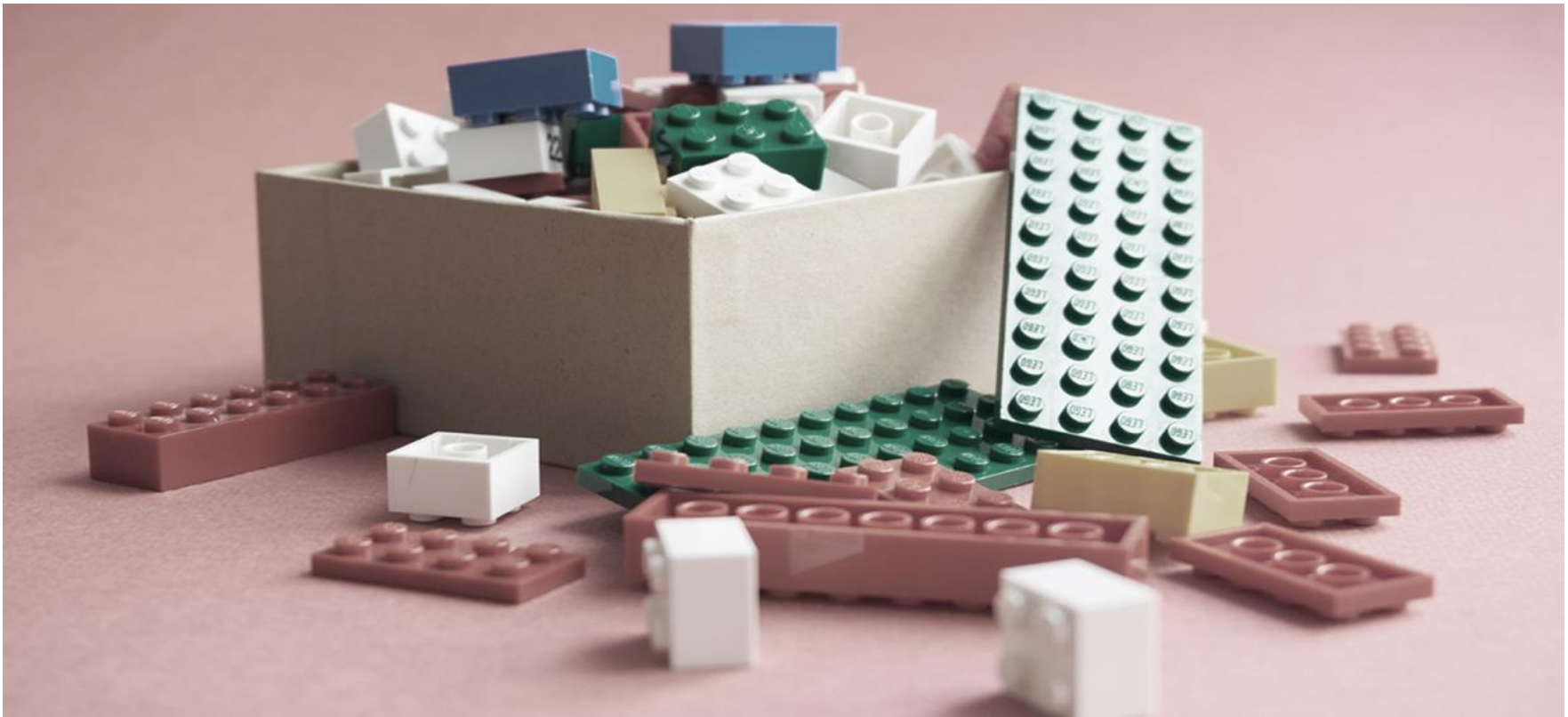
STEP. 4

Dynamic SQL 힌트 적용하기

- 가장 효율적인 조회 조건 순서

STEP. 1

Dynamic SQL이란?



Dynamic SQL이란?

- ✓ SQL 특정 검색 조건에 따라 SQL이 유연하게 변경되는 방식
- ✓ 쉽게 말해서 화면의 조회 조건에 따라 SQL에 조건을 추가하는 방식
- ✓ 보통 웹 프로그램에서 XML을 활용하여 SQL에 Dynamic 하게 검색 조건을 추가하는 방식

SQL 작성 방법의 변화

Dynamic SQL 사용 화면 예

[illegible]

```

# if (!$vo.payFrYyyyymm.equals("") && $vo.searchGubn.equals("0"))
/* 지급연월 */
    AND gf42.pay_yyyyymm BETWEEN :vo.payFrYyyyymm AND :vo.payToYyyyymm
# end
# if (!$vo.payFrYyyyymm.equals("") && $vo.searchGubn.equals("1"))
/* 발채연월 */
    AND gf42.search_yyyyymm BETWEEN :vo.payFrYyyyymm AND :vo.payToYyyyymm
# end
# if (!$vo.cntlBrchCd.equals("")) /* 지사조건 */
    AND ub12.mgmt_brch_cd = :vo.cntlBrchCd
    AND ub12.mgmt_brch_pstn_type = :vo.cntlPstnType
# end

```

STEP.2

튜닝 진행방식

■ 튜닝작업의 진행절차

- **STEP1.** SQL 조회 조건 파악
- **STEP2.** SQL 조회 조건 분석 데이터 수집
- **STEP3.** 조회 조건 별 효율성 체크 (개선안 도출)
- **STEP4.** XML 소스 수정 및 적용
- **STEP5.** 개선안 적용 후 모니터링

SQL 조회 조건 파악

SQL 조회 조건 분석

조회 조건 별 효율성 체크

XML 소스 수정 및 적용

개선안 적용 후 모니터링

- 성능개선 시 먼저 수행해야 하는 작업은 SQL 조회 조건을 상세히 파악하는 것이다.
- 조회 조건 컬럼에 따라 SQL의 Access Path가 결정되기 때문이다.
- 효율적인 조회 조건 컬럼을 가진 테이블이 먼저 수행되게 하고, 선행 테이블에서 추출된 데이터와 조인 조건 컬럼을 파악 후 조인방법과 데이터 액세스 방법이 결정되기 때문이다.

테이블 별 검색 조건 정리

XML File : nygf_WellInvst.sql.xml

SQL	화면 조회조건	필수 여부	Condition	Value
From절	TBF42	지급연월	!\$vo.payFrYyyyymm.equals("") && \$vo.searchGubn.equals("0")	AND gf42.pay_yyyyymm BETWEEN :vo.payFrYyyyymm AND :vo.payToYyyyymm
		발행연월	!\$vo.payFrYyyyymm.equals("") && \$vo.searchGubn.equals("1")	AND gf42.search_yyyyymm BETWEEN :vo.payFrYyyyymm AND :vo.payToYyyyymm
	TBB12	지사조건	!\$vo.cntlBrchCd.equals("")	AND ub12.mgmt_brch_cd = :vo.cntlBrchCd AND ub12.mgmt_brch_pstn_type = :vo.cntlPstnType
Where절	TBF42	처리상태	!\$vo.procYn.equals("")	AND DECODE(gf42.proc_yn, 'Y', 'Y', 'I', 'I', 'N') = :vo.procYn
		수급자주민번호	!\$vo.sgjJuminNo.equals("")	AND gf42.sgj_jumin_no = :vo.sgjJuminNo AND gf42.sgj_jumin_no_seq = :vo.sgjJuminNoSeq
		수급자명	!\$vo.sgjNm.equals("")	AND gf42.sgj_nm = :vo.sgjNm
	TBB12	담당자사번	!\$vo.pChrgEmpNo.equals("")	AND ub12.p_chrg_emp_no = :vo.pChrgEmpNo
	ETC	등록월	!\$vo.regFrYyyyymm.equals("")	AND SUBSTR(fst_in_dthms, 1, 6) BETWEEN :vo.regFrYyyyymm AND :vo.regToYyyyymm
		불일치내역조회	\$vo.chkBad.equals("1")	AND (gf43_a.ans_contn = '2' OR gf43_b.ans_contn = '2' OR gf43_c.ans_contn = '2' OR gf43_d.ans_contn = '2')

SQL 조회 조건 파악

SQL 조회 조건 분석

조회 조건 별 효율성 체크

XML 소스 수정 및 적용

개선안 적용 후 모니터링

- 조회 조건 및 조인 컬럼의 효율성 확인을 위해 필요한 데이터를 조사한다.
- 각 조회 조건의 효율성 확인을 위해서, 인덱스 존재여부와 해당 인덱스 및 검색 조건, 조인 컬럼의 Cardinality나 Distinct Value등을 확인해야 한다.
- 조회 조건의 효율성을 확인할 수 있는 정보들을 수집해야 Dynamic SQL에 대한 성능개선안을 도출할 수 있기 때문이다.

조회 조건 분석을 위한 데이터 수집

Table Alias	Table Name	Column 설명	Column	Table_rows	Num Distinct	Num_Nulls	Rows(=평균)
gf42	TBF42	지급연월	pay_yyyymm	2,037,827	65	0	31,351
		발채연월	search_yyyymm	2,037,827	65	0	31,351
		처리상태	proc_yn	2,037,827	3	0	679,276
		수급자주민번호	sgj_jumin_no	2,037,827	325,933	0	6
			sgj_jumin_no_seq	2,037,827	1	0	2,037,827
		수급자명	sgj_nm	2,037,827	40,250	0	51
ub12	TBB12	지사조건	mgmt_brch_cd	3,276,363	175	0	18,722
			mgmt_brch_pstn_type	3,276,363	3	0	1,092,121
		담당자사번	p_chrg_emp_no	3,276,363	2,243	37,768	1,444

SQL 조회 조건 파악

SQL 조회 조건 분석

조회 조건 별 효율성 체크

XML 소스 수정 및 적용

개선안 적용 후 모니터링

- 데이터를 이용하여 각 조회 조건 별 효율성을 분석하고 개선안을 도출하는 작업을 수행한다.
- 가장 중요한 부분은 모든 조회 조건 컬럼들의 효율 순서를 정하는 것이다.
- 도출된 순서가 바로 조인순서가 된다.

Dynamic SQL의 개선안 도출

Table Alias	Table Name	Column 설명	Column	Rows(=평균)	Index 사용 시 효율성	Index 존재 여부	Index Name	Driving	Check Order	비고
gf42	TBF42	지급연월	pay_yyyymm	31,351	X	O	IX_TBF42_04	N	4	지급연월 + 지사조건 조회 시 기존 Inline View Z 사용 시 효율적임.
		발체연월	search_yyyymm	31,351	X	O	IX_TBF42_01	N	4	발체연월 + 지사조건 조회 시 기존 Inline View Z 사용 시 효율적임.
		처리상태	proc_yn	679,276	X	X		N		
		수급자주민번호	sgj_jumin_no	6	O	O	IX_TBF42_03	Y	1	평균 6건 이내로 효율적임.
			sgj_jumin_no_seq	2,037,827						
		수급자명	sgj_nm	51	O	X		Y	2	조회 시 효율적이거나, 인덱스가 존재하지 않음.
ub12	TBB12	지사조건	mgmt_brch_cd	18,722	X	O	IX_TBB12_07	N	4	지급연월 + 지사조건 조회 시 기존 Inline View Z 사용 시 효율적임.
			mgmt_brch_pstn_type	1,092,121	X	O				
		담당자사번	p_chrg_emp_no	1,444	O	O	IX_TBB12_08	Y	3	평균 1444건 추출되므로 수급자 주민번호, 수급자명으로 조회되지 않는 경우 체크 해야 함.

조회 조건 별 효율성 체크

— 추가 인덱스 생성 고려

- 모든 정보를 토대로 효율적인 조회가 가능한 컬럼 중 인덱스가 존재하지 않은 컬럼을 선별한다.
- 수급자명 조건인 TBF42 테이블의 `sgj_nm` 컬럼에 인덱스가 생성되어 있지 않다.
- 해당 컬럼은 =조회 시 평균 51건만 추출하여, 인덱스 생성 시 효율적인 조회가 될 수 있음을 알 수 있으므로, 추가적인 개선안으로 해당 컬럼에 인덱스 생성을 고려한다.

수급자주민번호



수급자명



담당자 사번



지급연월+지사조건 or 발체연월+지사조건

Table Alias	Table Name	Column 설명	Column	Table_rows	Num Distinct	Num_Nulls	Rows(=평균)	Index 사용 시	Index 존재여부	Index Name	Driving	Check Order	비고
gf42	TBF42	지급연월	pay_yyyymm	2,037,827	65	0	31,351	X	O	IX_TBF42_04	N	4	지급연월 + 지사조건 조회 시 기존 Inline View Z 사용 시 효율적임.
		발체연월	search_yyyymm	2,037,827	65	0	31,351	X	O	IX_TBF42_01	N	4	발체연월 + 지사조건 조회 시 기존 Inline View Z 사용 시 효율적임.
		처리상태	proc_yn	2,037,827	3	0	679,276	X	X		N		
		수급자주민번호	sgj_jumin_no	2,037,827	325,933	0	6	O	O	IX_TBF42_03	Y	1	평균 6건 이내로 효율적임.
			sgj_jumin_no_seq	2,037,827	1	0	2,037,827						
ub12	TBB12	수급자명	sgj_nm	2,037,827	40,250	0	51	O	X		Y	2	조회 시 효율적이나, 인덱스가 존재하지 않음.
		지사조건	mgmt_brch_cd	3,276,363	175	0	18,722	X	O	IX_TBB12_07	N	4	지급연월 + 지사조건 조회 시 기존 Inline View Z 사용 시 효율적임.
			mgmt_brch_pstn_type	3,276,363	3	0	1,092,121	X	O				
		담당자사번	p_chrg_emp_no	3,276,363	2,243	37,768	1,444	O	O	IX_TBB12_08	Y	3	평균 1444건 추출되므로 수급자주민번호, 수급자명으로 조회되지 않는 경우 체크 해야함.

SQL 조회 조건 파악

SQL 조회 조건 분석

조회 조건 별 효율성 체크

XML 소스 수정 및 적용

개선안 적용 후 모니터링

수급자 주민번호 조회여부 판단(1)

```
#if(!$vo.sgjJuminNo.equals(""))
```

⇒ 수급자주민번호로 조회 시 튜닝내용 적용

```
#end
```

수급자명 조회여부를 판단(2)

```
#if($vo.sgjJuminNo.equals("") && !$vo.sgjNm.equals(""))
```

수급자주민번호로 조회되지 않는 경우

⇒ 수급자명으로 조회 시 튜닝내용 적용

```
#end
```

SQL 조회 조건 파악

SQL 조회 조건 분석

조회 조건 별 효율성 체크

XML 소스 수정 및 적용

개선안 적용 후 모니터링

■ 담당자 사번 조회여부 판단(3)

```
##($vo.sgjJuminNo.equals("") && $vo.sgjNm.equals("") && !$vo.pChrgEmpNo.equals(""))  
수급자주민번호와 수급자명으로 조회되지 않는 경우
```

⇒ 담당자 사번으로 조회 시 튜닝내용 적용
#end

■ 적용순서1,2,3이 모두 조회되지 않는 경우에 지급연월+지사조건에 해당하는 개선안을 적용(4)

```
##( $vo.sgjJuminNo.equals("") && $vo.sgjNm.equals("") && $vo.pChrgEmpNo.equals("")  
&& !$vo.payFrYyyymm.equals("") && $vo.searchGubn.equals("0") && !$vo.cntlBrchCd.equals(""))  
수급자주민번호, 수급자명, 담당자사번으로 조회되지 않는 경우
```

⇒ 지급연월+지사조건으로 조회 시 튜닝내용 적용
#end

SQL 조회 조건 파악

SQL 조회 조건 분석

조회 조건 별 효율성 체크

XML 소스 수정 및 적용

개선안 적용 후 모니터링

■ 모니터링 이유

- Dynamic SQL의 경우, 개선안을 도출하는 작업도 중요하지만 개선안을 제대로 반영하고 해당 프로그램이 문제없이 수행되는 것을 확인.
- 개선안을 도출하는 작업 시에 미처 발견하지 못한 또 다른 조회 패턴 추출.
- 개선할 여지가 남아있어 추가로 개선작업을 적용해야 하는 경우가 존재.

대량의 데이터 처리 성능개선 방법

5
CHAPTER

 The Start of SQL Tuning Seminar

컨설팅 본부
www.ex-em.com
exem.tistory.com

CONTENTS

STEP. 1

인식의 전환

- 대용량 처리의 성능 이슈
- 인식의 전환 필요

STEP. 2

성능 이슈 해결의 전략들

- 적극적 Partitioning 전략
- Parallel Processing 전략
- Memory Caching 전략
- Summary 전략

STEP. 1

인식의 전환



대용량 데이터 처리의 성능 이슈

- ✓ OLTP/OLAP 업무 구분의 모호함.
- ✓ 대량의 데이터를 대상으로 하는 업무들의 비중이 늘어나고 있음.
- ✓ 해당 이슈에 대한 해결을 위해서는 인식의 전환이 필요함.

Parallel Processing

- ✓ Parallel Processing은 오로지 Batch 업무에 국한되어야 하는가?
- ✓ CPU, Memory 등의 Resource 측면에서 과부하를 유발시키는 핵심인가?
- ✓ 상식적으로 Parallel Processing은 가급적 사용하지 않는 편이 좋은가?

Table Partitioning

- ✓ Partition Table은 단지, 관리적 이점만이 존재하는가?
- ✓ Partition Table로의 전환이 가능하다면 효율적인 전환 방법이 존재하는가?
- ✓ Partition Table로의 전환이 불가능하다면 대안은?

Memory 사용의 극대화

- ✓ Memory 사용을 효율적으로 사용하고 있는가?
- ✓ Multi-Buffer Pool 공약을 세울 것!
- ✓ 정형화 된 업무의 Result Caching 사용!

Summary Segment의 필요성

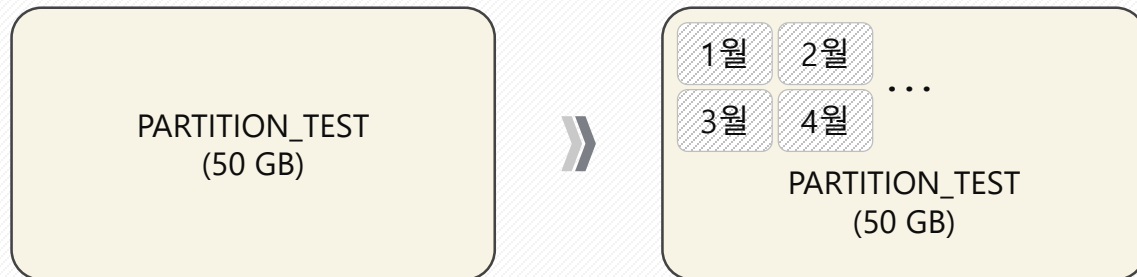
- ✓ Summary Segment의 필요성은 절대적임.
- ✓ M-View 혹은 Summary Table의 필요성은 물론이며, 다중화 구조 고려.

STEP.2

성능 이슈 해결의 전략들

Partitioning의 당위성

- Index Scan의 Cost가 높아지는 순간 Table Full Scan 발생.
- 반복되는 대용량 데이터 처리 업무 시마다 50GB에 해당하는 Segment를 모두 읽을 것인가?
- Partitioning이 해법이 될 수 있다.
- 모든 경우에 Partition Table로의 전환이 가능한 것은 아니다.



Non-Partition Table 전환

Composite Partition 전략

Partition Split 활용

Partition Table 전환이 불가능한 경우

[테이블 생성 DDL]

```
CREATE TABLE partition_test (  
    TIME_ID DATE ,  
        DAY_NAME VARCHAR2(9),  
    ...중략...  
        END_OF_CAL_YEAR DATE ,  
        END_OF_FIS_YEAR DATE )  
PARTITION BY RANGE( time_id )(  
    PARTITION partition_test_p1 VALUES less than (to_date('1998-01-01','YYYY-MM-DD')),  
    PARTITION partition_test_p2 VALUES less than (to_date('1998-02-01','YYYY-MM-DD')),  
    ...중략...  
    PARTITION partition_test_p71 VALUES less than (to_date('2003-11-01','YYYY-MM-DD')),  
    PARTITION partition_test_p72 VALUES less than (to_date('2003-12-01','YYYY-MM-DD')) );
```

[테이블 생성 DDL]

```
CREATE TABLE partition_test2 (  
    TIME_ID DATE,  
        DAY_NAME VARCHAR2(9),  
    ...중략...  
        END_OF_CAL_YEAR DATE,  
        END_OF_FIS_YEAR DATE );
```

[Partition_test2 인덱스 생성 스크립트]

```
CREATE INDEX partition_test_idx_01 ON partition_test2 (time_id) ;
```

Non-Partition Table 전환

Composite Partition 전략

Partition Split 활용

Partition Table 전환이 불가능한 경우

[데이터 입력(두 테이블 동일)]

```
INSERT /*+ append */
INTO partition_test
SELECT a.*
FROM sh.times a ,
(
  SELECT LEVEL
  FROM dual
  CONNECT BY LEVEL <= 10000
) b;
```

COMMIT;

```
INSERT /*+ append */
INTO partition_test2
SELECT a.*
FROM sh.times a ,
(
  SELECT LEVEL
  FROM dual
  CONNECT BY LEVEL <= 10000
) b;
```

COMMIT;

성능 비교 - Before

전체 데이터를 읽는 경우 - Non Partition

```
:FROM_DATE = '1998-01-01'
:TO_DATE    = '1998-02-01'

SELECT day_number_in_month ,
       day_number_in_week ,
       day_name ,
       COUNT( * )
FROM   partition_test2
WHERE  time_id >= TO_DATE( :from_date , 'yyyy-mm-dd' )
AND    time_id < TO_DATE( :to_date , 'yyyy-mm-dd' )
GROUP  BY day_number_in_month ,
         day_number_in_week ,
         day_name;
```

Id	Operation	Name	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		31	00:00:08.39	518K	518K
1	HASH GROUP BY		31	00:00:08.39	518K	518K
* 2	FILTER		310K	00:00:08.30	518K	518K
* 3	TABLE ACCESS FULL	PARTITION_TEST2	310K	00:00:08.24	518K	518K

Predicate Information (identified by operation id):

- 2 - filter(TO_DATE(:FROM_DATE,'yyyy-mm-dd')<TO_DATE(:TO_DATE,'yyyy-mm-dd'))
- 3 - filter(("TIME_ID">=TO_DATE(:FROM_DATE,'yyyy-mm-dd') AND "TIME_ID"<TO_DATE(:TO_DATE,'yyyy-mm-dd')))

성능 비교 - Before

인덱스 사용 - Non Partition

```

:FROM_DATE = '1998-01-01'
:TO_DATE    = '1998-02-01'

SELECT day_number_in_month ,
       day_number_in_week ,
       day_name ,
       COUNT( * )
FROM   partition_test2
WHERE  time_id >= TO_DATE( :from_date , 'yyyy-mm-dd' )
AND    time_id < TO_DATE( :to_date , 'yyyy-mm-dd' )
GROUP  BY day_number_in_month ,
         day_number_in_week ,
         day_name
    
```

Id	Operation	Name	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		31	00:00:05.75	310K	34915
1	HASH GROUP BY		31	00:00:05.75	310K	34915
* 2	FILTER		310K	00:00:05.62	310K	34915
3	TABLE ACCESS BY INDEX ROWID	PARTITION_TEST2	310K	00:00:05.53	310K	34915
* 4	INDEX RANGE SCAN	PARTITION_TEST_IDX_01	310K	00:00:00.14	823	805

Predicate Information (identified by operation id):

- ```

2 - filter(TO_DATE(:FROM_DATE,'yyyy-mm-dd')<TO_DATE(:TO_DATE,'yyyy-mm-dd'))
4 - access("TIME_ID">=TO_DATE(:FROM_DATE,'yyyy-mm-dd') AND "TIME_ID"<TO_DATE(:TO_DATE,'yyyy-mm-dd'))

```

## 성능 비교 - After

### 필요한 파티션만 읽는 경우 - Partition Range Iterator

```

:FROM_DATE = '1998-01-01'
:TO_DATE = '1998-02-01'

SELECT day_number_in_month ,
 day_number_in_week ,
 day_name ,
 COUNT(*)
FROM partition_test
WHERE time_id >= TO_DATE(:from_date , 'yyyy-mm-dd')
AND time_id < TO_DATE(:to_date , 'yyyy-mm-dd')
GROUP BY day_number_in_month ,
 day_number_in_week ,
 day_name

```

| Id  | Operation                | Name           | A-Rows | A-Time      | Buffers | Reads |
|-----|--------------------------|----------------|--------|-------------|---------|-------|
| 0   | SELECT STATEMENT         |                | 31     | 00:00:00.29 | 9216    | 9216  |
| 1   | HASH GROUP BY            |                | 31     | 00:00:00.29 | 9216    | 9216  |
| * 2 | FILTER                   |                | 310K   | 00:00:00.21 | 9216    | 9216  |
| 3   | PARTITION RANGE ITERATOR |                | 310K   | 00:00:00.14 | 9216    | 9216  |
| * 4 | TABLE ACCESS FULL        | PARTITION_TEST | 310K   | 00:00:00.08 | 9216    | 9216  |

Predicate Information (identified by operation id):

- ```

2 - filter(TO_DATE(:FROM_DATE,'yyyy-mm-dd')<TO_DATE(:TO_DATE,'yyyy-mm-dd'))
4 - filter(("TIME_ID">=TO_DATE(:FROM_DATE,'yyyy-mm-dd') AND "TIME_ID"<TO_DATE(:TO_DATE,'yyyy-mm-dd'))
    
```


성능 비교 – Partition Table Size

Size 비교 SQL 수행

```
SELECT partition_name ,  
       blocks ,  
       bytes/1024/1024 MB  
FROM   dba_segments  
WHERE  partition_name = UPPER( 'PARTITION_TEST_P2')  
AND    segment_name   = UPPER( 'PARTITION_TEST' );
```

PARTITION_NAME	BLOCKS	MB
-----	-----	-----
PARTITION_TEST_P2	9216	72

Non-Partition Table 전환

Composite Partition 전략

Partition Split 활용

Partition Table 전환이 불가능한 경우

- Partition Table의 성능적 관점을 이해했다면?
- 이제 더 나누어 쪼갤 여지가 있는지를 따져보게 됨.
- 업무 성격에 따라 가/불가가 정해지겠지만, 가능하다면 Composite Partition 으로의 전환까지 고려하게 됨.
- 다음 테스트는 동일 테이블에서 요일로 또 다시 나누어 본 결과임.

성능 비교 - 단일 Partition

단일 Partition - SQL 수행 및 XPLAN 결과

```
:FROM_DATE = '1998-01-01'
:TO_DATE    = '1998-02-01'
:DAY_NAME   = 'Monday'
```

```
SELECT day_number_in_month ,
       day_number_in_week ,
       day_name ,
       COUNT( * )
FROM   partition_test
WHERE  time_id >= TO_DATE( :from_date , 'yyyy-mm-dd' )
AND    time_id < TO_DATE( :to_date , 'yyyy-mm-dd' )
AND    day_name = :day_name
GROUP BY day_number_in_month , day_number_in_week , day_name
```

Id	Operation	Name	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		4	00:00:00.27	8860	8858
1	HASH GROUP BY		4	00:00:00.27	8860	8858
* 2	FILTER		40000	00:00:00.26	8860	8858
3	PARTITION RANGE ITERATOR		40000	00:00:00.25	8860	8858
* 4	TABLE ACCESS FULL	PARTITION_TEST	40000	00:00:00.24	8860	8858

Predicate Information (identified by operation id):

- ```
2 - filter(TO_DATE(:FROM_DATE,'yyyy-mm-dd')<TO_DATE(:TO_DATE,'yyyy-mm-dd'))
4 - filter(("DAY_NAME"=:DAY_NAME AND "TIME_ID"<TO_DATE(:TO_DATE,'yyyy-mm-dd') AND "TIME_ID">=TO_DATE(:FROM_DATE,'yyyy-mm-dd')))
```

## 성능 비교 – Composite Partition

### Composite Partition 구성 스크립트

```
CREATE TABLE composite_partition (
 TIME_ID DATE , DAY_NAME VARCHAR2(9),
 ...중략...
 END_OF_FIS_YEAR DATE)
PARTITION BY RANGE (time_id)
SUBPARTITION BY LIST (day_name)
(
 PARTITION composite_partition01 VALUES less than (to_date('1998-01-01','YYYY-MM-DD'))
 (
 SUBPARTITION partition01_monday VALUES ('Monday'),
 SUBPARTITION partition01_tuesday VALUES ('Tuesday'),
 SUBPARTITION partition01_wednesday VALUES ('Wednesday'),
 SUBPARTITION partition01_thursday VALUES ('Thursday'),
 SUBPARTITION partition01_friday VALUES ('Friday') ,
 SUBPARTITION partition01_saturday VALUES ('Saturday') ,
 SUBPARTITION partition01_sunday VALUES ('Sunday')
),
 ...중략...
 PARTITION composite_partition72 VALUES less than (to_date('2003-12-01','YYYY-MM-DD'))
 (
 SUBPARTITION partition72_monday VALUES ('Monday'),
 SUBPARTITION partition72_tuesday VALUES ('Tuesday'),
 SUBPARTITION partition72_wednesday VALUES ('Wednesday'),
 SUBPARTITION partition72_thursday VALUES ('Thursday'),
 SUBPARTITION partition72_friday VALUES ('Friday') ,
 SUBPARTITION partition72_saturday VALUES ('Saturday') ,
 SUBPARTITION partition72_sunday VALUES ('Sunday')
)
);
```

## 성능 비교 – Composite Partition

### Composite Partition - SQL 수행 및 XPLAN 결과

```

:FROM_DATE = '1998-01-01'
:TO_DATE = '1998-02-01'
:DAY_NAME = 'Monday'

SELECT day_number_in_month ,
 day_number_in_week ,
 day_name ,
 COUNT(*)
FROM composite_partition
WHERE time_id >= TO_DATE(:from_date , 'yyyy-mm-dd')
AND time_id < TO_DATE(:to_date , 'yyyy-mm-dd')
AND day_name = :day_name
GROUP BY day_number_in_month , day_number_in_week , day_name;

```

| Id  | Operation                | Name                | A-Rows | A-Time      | Buffers | Reads |
|-----|--------------------------|---------------------|--------|-------------|---------|-------|
| 0   | SELECT STATEMENT         |                     | 4      | 00:00:00.07 | 1149    | 1143  |
| 1   | HASH GROUP BY            |                     | 4      | 00:00:00.07 | 1149    | 1143  |
| * 2 | FILTER                   |                     | 40000  | 00:00:00.06 | 1149    | 1143  |
| 3   | PARTITION RANGE ITERATOR |                     | 40000  | 00:00:00.05 | 1149    | 1143  |
| 4   | PARTITION LIST SINGLE    |                     | 40000  | 00:00:00.04 | 1149    | 1143  |
| * 5 | TABLE ACCESS FULL        | COMPOSITE_PARTITION | 40000  | 00:00:00.04 | 1149    | 1143  |

Predicate Information (identified by operation id):

- ```

2 - filter(TO_DATE(:FROM_DATE,'yyyy-mm-dd')<TO_DATE(:TO_DATE,'yyyy-mm-dd'))
5 - filter(("TIME_ID">=TO_DATE(:FROM_DATE,'yyyy-mm-dd') AND "TIME_ID"<TO_DATE(:TO_DATE,'yyyy-mm-dd'))
    
```

Non-Partition Table 전환

Composite Partition 전략

Partition Split 활용

Partition Table 전환이 불가능한 경우

- Partition Table의 성능 이점을 이해하여 적극 활용.
- Partition 적용만큼 중요한 것은 Partition Table의 관리.
- 관리 소홀로 인한 부하 발생 사례가 생각보다 많음.
- 주요 대상은 Default Partition Table!
- 관리 포인트는 미리 충분한 Partition Table을 생성해 놓거나, Procedure 등을 통한 Partition 자동 관리 기법을 도입해야 함.

Non-Partition Table 전환

Composite Partition 전략

Partition Split 활용

Partition Table 전환이 불가능한 경우

[테이블 생성 DDL]

```
CREATE TABLE partition_test3 (TIME_ID DATE ,
                              DAY_NAME VARCHAR2(9),
                              ...중략...
                              END_OF_FIS_YEAR DATE )
PARTITION BY RANGE( time_id )(
    PARTITION partition_test01 VALUES less than (to_date('1998-01-01','YYYY-MM-DD')),
    PARTITION partition_test02 VALUES less than (to_date('1998-02-01','YYYY-MM-DD')),
    ...중략...
    PARTITION partition_test25 VALUES less than (to_date('2000-01-01','YYYY-MM-DD')),
    PARTITION partition_test_default VALUES less THAN (maxvalue)
);
```

[데이터 입력]

```
INSERT /*+ append */ INTO partition_test3
SELECT a.*
FROM   sh.times a ,
      (
        SELECT LEVEL
        FROM   dual
        CONNECT BY LEVEL <= 10000
      ) b;

COMMIT;
```

Default Partition의 관리가 필요한 사례

Partition 테이블 별 Size 비교

```
SELECT partition_name ,  
       blocks ,  
       bytes/1024/1024 mb  
FROM   dba_segments  
WHERE  segment_name = UPPER( 'partition_test3' );
```

PARTITION_NAME	BLOCKS	MB
PARTITION_TEST_DEFAULT	316928	2476
PARTITION_TEST25	9600	75
PARTITION_TEST24	9600	75
PARTITION_TEST23	9984	78
PARTITION_TEST22	9728	76
PARTITION_TEST21	9984	78
PARTITION_TEST20	9472	74
PARTITION_TEST19	9088	71
PARTITION_TEST18	9472	74
...중략...		
PARTITION_TEST05	9088	71
PARTITION_TEST04	9984	78
PARTITION_TEST03	8576	67
PARTITION_TEST02	9984	78

해당 Partition Table의 Default Partition Segment는
관리 소홀로 2001년 1월 이후의 데이터를 조회하는
경우, 불필요한 I/O가 발생하게 됨.

Default Partition의 관리가 필요한 사례

Partition Split 수행

```
ALTER TABLE partition_test3 split PARTITION partition_test_default AT( TO_DATE( '2000-02-01' , 'YYYY-MM-DD' ) )
INTO( PARTITION partition_test26 , PARTITION partition_test_default ) ;
ALTER TABLE partition_test3 split PARTITION partition_test_default AT( TO_DATE( '2000-03-01' , 'YYYY-MM-DD' ) )
INTO( PARTITION partition_test27 , PARTITION partition_test_default ) ;
ALTER TABLE partition_test3 split PARTITION partition_test_default AT( TO_DATE( '2000-04-01' , 'YYYY-MM-DD' ) )
INTO( PARTITION partition_test28 , PARTITION partition_test_default ) ;
ALTER TABLE partition_test3 split PARTITION partition_test_default AT( TO_DATE( '2000-05-01' , 'YYYY-MM-DD' ) )
INTO( PARTITION partition_test29 , PARTITION partition_test_default ) ;
ALTER TABLE partition_test3 split PARTITION partition_test_default AT( TO_DATE( '2000-06-01' , 'YYYY-MM-DD' ) )
INTO( PARTITION partition_test30 , PARTITION partition_test_default ) ;
ALTER TABLE partition_test3 split PARTITION partition_test_default AT( TO_DATE( '2000-07-01' , 'YYYY-MM-DD' ) )
INTO( PARTITION partition_test31 , PARTITION partition_test_default ) ;
..중략..
ALTER TABLE partition_test3 split PARTITION partition_test_default AT( TO_DATE( '2003-12-01' , 'YYYY-MM-DD' ) )
INTO( PARTITION partition_test72 , PARTITION partition_test_default ) ;
```

Default Partition의 관리가 필요한 사례

Size 변화 확인

PARTITION_NAME	BLOCKS	BYTES/1024/1024
-----	-----	-----
PARTITION_TEST72	10240	80
PARTITION_TEST71	9216	72
PARTITION_TEST70	9216	72
..중략..		
PARTITION_TEST39	9216	72
PARTITION_TEST38	9216	72
PARTITION_TEST37	9216	72
PARTITION_TEST36	9216	72
PARTITION_TEST35	9216	72
PARTITION_TEST34	9216	72
PARTITION_TEST33	9216	72
PARTITION_TEST32	9216	72
PARTITION_TEST31	9216	72
PARTITION_TEST30	9216	72
PARTITION_TEST29	9216	72
PARTITION_TEST28	9216	72
PARTITION_TEST27	9216	72
PARTITION_TEST26	9216	72

적극적 Partitioning 전략

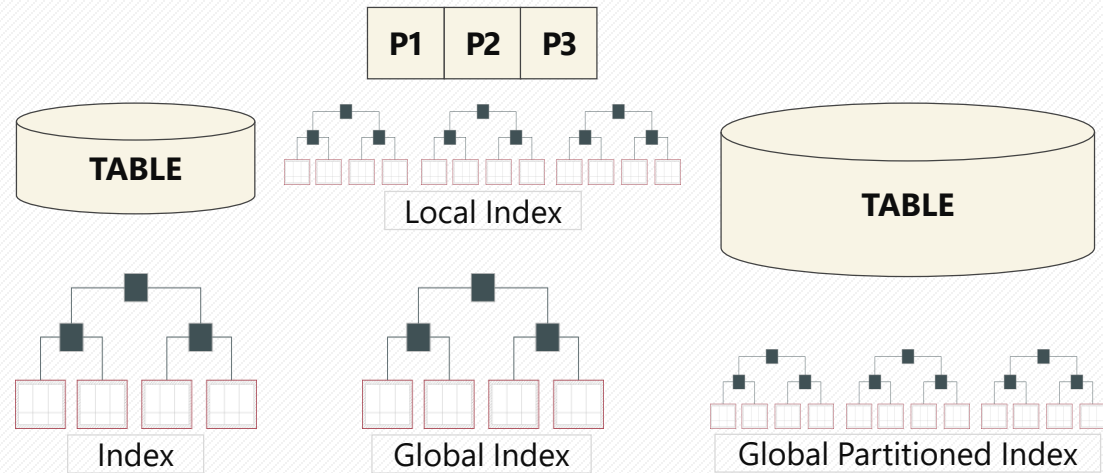
Non-Partition Table 전환

Composite Partition 전략

Partition Split 활용

Partition Table 전환이 불가한 경우

- Partition Key를 항상 조건 절에 입력할 수 없는 경우.
- Table Size가 크거나 Table의 업무의 중요도로 작업에 보수적인 경우.
- 기타 등등의 이유로 전환이 불가할 때, 방안은?
- 경우에 따라 Partition Table을 도입한 것과 같은 효과를 얻을 수 있다!
- Global Partitioned Index!



Non-Partition Table 전환

Composite Partition 전략

Partition Split 활용

Partition Table 전환이 불가능한 경우

[테이블 생성 DDL (생성 전문은 Appendix 참조)]

```
CREATE TABLE global_partition_idx_test (  
  TIME_ID DATE ,  
          DAY_NAME VARCHAR2(9),  
          DAY_NUMBER_IN_WEEK NUMBER(1,0) ,  
  ...중략...  
          END_OF_CAL_YEAR DATE ,  
          END_OF_FIS_YEAR DATE );
```

[데이터 입력]

```
INSERT /*+ append */  
INTO global_partition_idx_test  
SELECT a.*  
FROM sh.times a ,  
      (  
        SELECT LEVEL  
        FROM dual  
        CONNECT BY LEVEL <= 10000  
      ) b;  
  
COMMIT;
```

Global Partitioned Index

일반 테이블 - SQL 수행 및 XPLAN 결과

```
:FROM_DATE = '2000-01-01'  
:TO_DATE    = '2000-02-01'
```

```
SELECT day_number_in_month ,  
       day_number_in_week ,  
       day_name ,  
       COUNT( * )  
FROM   global_partition_idx_test a  
WHERE  time_id >= TO_DATE( :from_date , 'yyyy-mm-dd' )  
AND    time_id <  TO_DATE( :to_date  , 'yyyy-mm-dd' )  
GROUP  BY day_number_in_month ,  
         day_number_in_week ,  
         day_name
```

Id	Operation	Name	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		31	00:00:08.39	518K	518K
1	HASH GROUP BY		31	00:00:08.39	518K	518K
* 2	FILTER		310K	00:00:08.30	518K	518K
* 3	TABLE ACCESS FULL	GLOBAL_PARTITION_IDX_TEST	310K	00:00:08.24	518K	518K

Predicate Information (identified by operation id):

- 2 - filter(TO_DATE(:FROM_DATE,'yyyy-mm-dd')<TO_DATE(:TO_DATE,'yyyy-mm-dd'))
- 3 - filter(("TIME_ID">=TO_DATE(:FROM_DATE,'yyyy-mm-dd') AND "TIME_ID"<TO_DATE(:TO_DATE,'yyyy-mm-dd')))

Global Partitioned Index

Global Partitioned Index 생성 스크립트

```
CREATE INDEX global_part_idx01 ON global_partition_idx_test
(time_id, day_number_in_month ,day_number_in_week ,day_name )
GLOBAL PARTITION BY RANGE(time_id)
(
    PARTITION global_part_idx_test001 VALUES less than (to_date('1998-01-01','YYYY-MM-DD')),
    PARTITION global_part_idx_test002 VALUES less than (to_date('1998-02-01','YYYY-MM-DD')),
    PARTITION global_part_idx_test003 VALUES less than (to_date('1998-03-01','YYYY-MM-DD')),
    PARTITION global_part_idx_test004 VALUES less than (to_date('1998-04-01','YYYY-MM-DD')),
    PARTITION global_part_idx_test005 VALUES less than (to_date('1998-05-01','YYYY-MM-DD')),
    PARTITION global_part_idx_test006 VALUES less than (to_date('1998-06-01','YYYY-MM-DD')),
    ...중략...
    PARTITION global_part_idx_test0066 VALUES less than (to_date('2003-06-01','YYYY-MM-DD')),
    PARTITION global_part_idx_test0067 VALUES less than (to_date('2003-07-01','YYYY-MM-DD')),
    PARTITION global_part_idx_test0068 VALUES less than (to_date('2003-08-01','YYYY-MM-DD')),
    PARTITION global_part_idx_test0069 VALUES less than (to_date('2003-09-01','YYYY-MM-DD')),
    PARTITION global_part_idx_test0070 VALUES less than (to_date('2003-10-01','YYYY-MM-DD')),
    PARTITION global_part_idx_test0071 VALUES less than (to_date('2003-11-01','YYYY-MM-DD')),
    PARTITION global_part_idx_test0072 VALUES less than (to_date('2003-12-01','YYYY-MM-DD')),
    PARTITION global_part_idx_max VALUES less than (maxvalue)
);
```

Global Partitioned Index

Global Partitioned Index - SQL 수행 및 XPLAN 결과

```

:FROM_DATE = '2000-01-01'
:TO_DATE    = '2000-02-01'

SELECT /*+ index_ffs(a) */
  day_number_in_month ,
    day_number_in_week ,
    day_name ,
    COUNT( * )
FROM   global_partition_idx_test a
WHERE  time_id >= TO_DATE( :from_date , 'yyyy-mm-dd' )
AND    time_id <  TO_DATE( :to_date  , 'yyyy-mm-dd' )
GROUP  BY day_number_in_month ,
          day_number_in_week ,
          day_name

```

Id	Operation	Name	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		31	00:00:00.33	1433	1420
1	HASH GROUP BY		31	00:00:00.33	1433	1420
* 2	FILTER		310K	00:00:00.27	1433	1420
3	PARTITION RANGE ITERATOR		310K	00:00:00.20	1433	1420
* 4	INDEX FAST FULL SCAN	GLOBAL_PART_IDX01	310K	00:00:00.14	1433	1420

Predicate Information (identified by operation id):

- ```

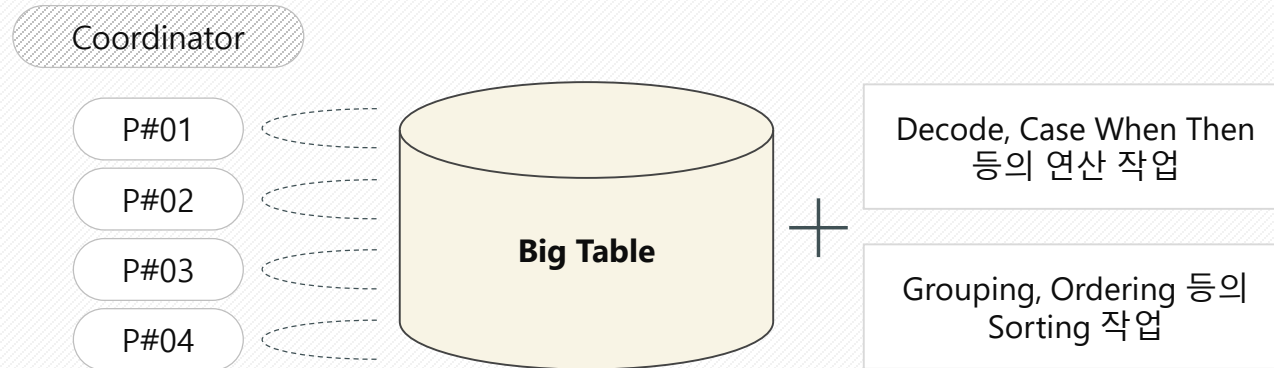
2 - filter(TO_DATE(:FROM_DATE,'yyyy-mm-dd')<TO_DATE(:TO_DATE,'yyyy-mm-dd'))
4 - access("TIME_ID">=TO_DATE(:FROM_DATE,'yyyy-mm-dd') AND "TIME_ID"<TO_DATE(:TO_DATE,'yyyy-mm-dd'))

```

### Parallel Processing 이란?

- 동일한 업무를 여러 세션에서 동시에 나누어 처리하는 것.

```
/*+ full(a) parallel(a 4) */
```





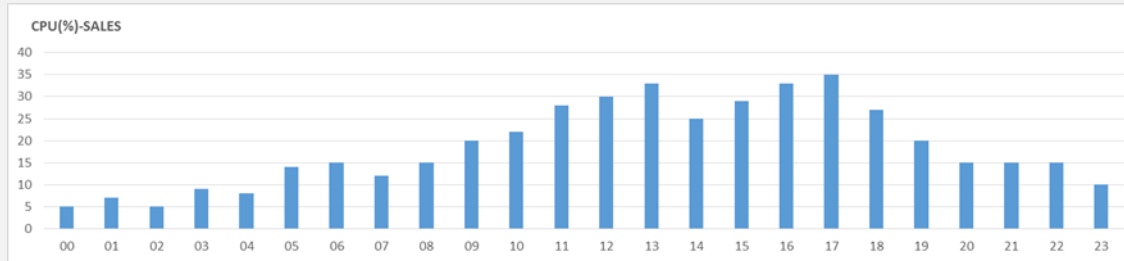
### Parallel Processing과 오해

- Parallel Processing은 오로지 Batch Program 작업에만 국한되어 사용되어야 하는가?
- CPU, Memory 등의 Resource 측면에서 과부하를 유발시키는가?
- 상식적으로 Parallel Processing은 가급적 사용하지 않는 것이 좋은가?

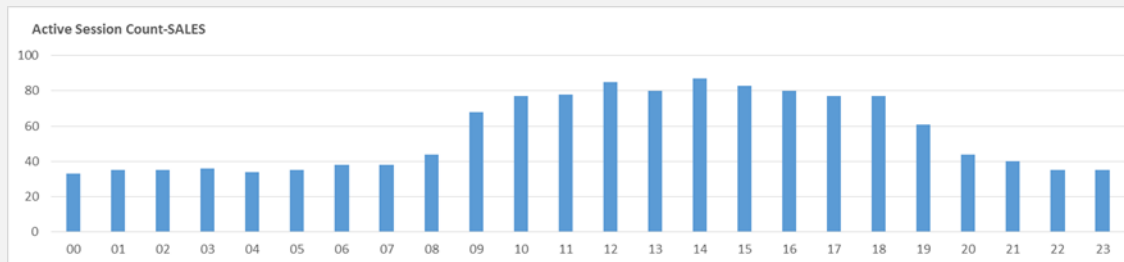


## Parallel Processing과 오해

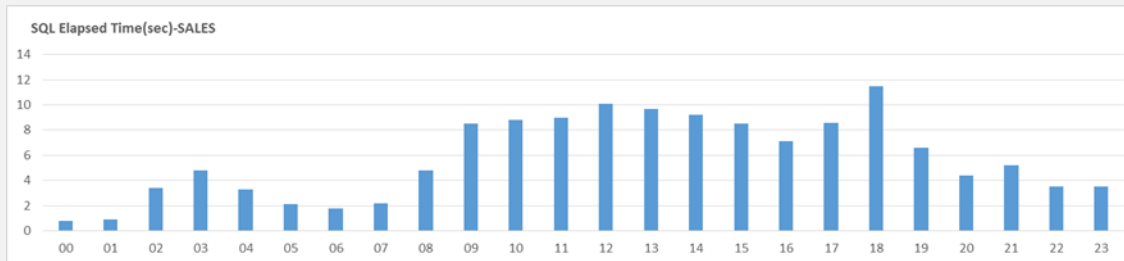
사례를 통해 본 Parallel Processing의 오해



[ SALES DB CPU 사용률 ]



[ SALES DB Active Session ]



[ SALES DB SQL Elapsed Time ]

## Parallel Processing과 오해

### 성능문제 SQL - One Processing

```
SELECT * FROM (
 WITH DATA AS (
 SELECT SHR.ORGLEVEL1,
 ...중략...
 FROM RPT_SEASON SHR
 , (SELECT * FROM RPT_WEEK WHERE WEEK BETWEEN '201324' AND '201339') WEEK
 , RPT_SITE REQSITE
 , RPT_SITE MFGSITE
 , (
 SELECT DECODE(INT,1, 'REQ_QTY',2, 'ONTIME_QTY',3, 'SHORT_QTY') CATEGORYNAME,
 DECODE(INT,1, 'C01',2, 'C02',3, 'C03') CATEGORYID
 FROM RPT_COPYTABLE WHERE INT <= 3
) G
 WHERE SHR.TOSITEID = REQSITE.SITEID
 AND SHR.FROMSITEID = MFGSITE.SITEID(+)
 AND SHR.PLANID = '201324_M'
 AND SHR.SHORTNAME NOT IN ('CONSENSUS_ERROR', 'SELLER_PREALLOC')
 AND SHR.PATHID = '1'
 AND SHR.PROBLEMID = '1'
 AND TO_CHAR(SHR.DUEDATE,'IYYYIW') = WEEK.WEEK -- 추가
)
 SELECT GROUPING(ORGLEVEL1) SORT1, GROUPING(APPARELGROUP) SORT2, GROUPING(CATEGORYNAME) SORT3
 ...중략...
 FROM DATA
 GROUP BY GROUPING SETS((),(CATEGORYNAME, CATEGORYID),(ORGLEVEL1,APPARELGROUP, CATEGORYNAME, CATEGORYID))
 HAVING GROUPING(ORGLEVEL1)||GROUPING(APPARELGROUP)||GROUPING(CATEGORYNAME) NOT IN ('111')
)
 ORDER BY SORT1 DESC, ORGLEVEL1, SORT2 DESC, APPARELGROUP , MEASURE , MEASURENAME
```

## Parallel Processing과 오해

### 성능문제 SQL - One Processing

| Id   | Operation                         | Name             | A-Rows | A-Time      | Buffers |
|------|-----------------------------------|------------------|--------|-------------|---------|
| 0    | SELECT STATEMENT                  |                  | 24     | 00:00:15.87 | 1352K   |
| 1    | SORT AGGREGATE                    |                  | 1      | 00:00:00.01 | 2       |
| 2    | FIRST ROW                         |                  | 0      | 00:00:00.01 | 2       |
| * 3  | INDEX RANGE SCAN (MIN/MAX)        | IDX_RPT_SALES    | 0      | 00:00:00.01 | 2       |
| 4    | SORT ORDER BY                     |                  | 24     | 00:00:15.87 | 1352K   |
| 5    | VIEW                              |                  | 24     | 00:00:15.87 | 1352K   |
| * 6  | FILTER                            |                  | 24     | 00:00:15.87 | 1352K   |
| 7    | SORT GROUP BY ROLLUP              |                  | 25     | 00:00:15.87 | 1352K   |
| 8    | MERGE JOIN CARTESIAN              |                  | 1020K  | 00:00:12.47 | 1352K   |
| 9    | NESTED LOOPS OUTER                |                  | 340K   | 00:00:11.12 | 1352K   |
| 10   | NESTED LOOPS                      |                  | 340K   | 00:00:10.14 | 1338K   |
| 11   | NESTED LOOPS                      |                  | 340K   | 00:00:08.92 | 1042K   |
| 12   | PARTITION LIST SINGLE             |                  | 340K   | 00:00:07.82 | 1042K   |
| * 13 | TABLE ACCESS BY LOCAL INDEX ROWID | RPT_SEASON       | 340K   | 00:00:07.55 | 1042K   |
| * 14 | INDEX RANGE SCAN                  | PK_UI_RPT_SEASON | 1046K  | 00:00:01.00 | 10160   |
| * 15 | INDEX UNIQUE SCAN                 | PK_RPT_WEEK      | 340K   | 00:00:00.81 | 4       |
| * 16 | INDEX UNIQUE SCAN                 | PK_RPT_SITE      | 340K   | 00:00:00.94 | 296K    |
| * 17 | INDEX UNIQUE SCAN                 | PK_RPT_SITE      | 339K   | 00:00:00.55 | 13526   |
| 18   | BUFFER SORT                       |                  | 1020K  | 00:00:00.58 | 1       |
| * 19 | INDEX RANGE SCAN                  | PK_RPT_COPYTABLE | 3      | 00:00:00.01 | 1       |

## Parallel Processing과 오해

### 성능문제 SQL - Parallel Processing

```
SELECT * FROM (
 WITH DATA AS (
 SELECT /*+ full(shr) parallel(shr 8) use_hash(reqsite mfgsite week g) */ -- PQ 힌트적용
 ...중략...
 FROM RPT_SEASON SHR
 , (SELECT * FROM RPT_WEEK WHERE WEEK BETWEEN '201324' AND '201339') WEEK
 , RPT_SITE REQSITE
 , RPT_SITE MFGSITE
 , (
 SELECT DECODE(INT,1, 'REQ_QTY',2, 'ONTIME_QTY',3, 'SHORT_QTY') CATEGORYNAME,
 DECODE(INT,1, 'C01',2, 'C02',3, 'C03') CATEGORYID
 FROM RPT_COPYTABLE WHERE INT <= 3
) G
 WHERE SHR.TOSITEID = REQSITE.SITEID
 AND SHR.FROMSITEID = MFGSITE.SITEID(+)
 AND SHR.PLANID = '201324_M'
 AND SHR.SHORTNAME NOT IN ('CONSENSUS_ERROR', 'SELLER_PREALLOC')
 AND SHR.PATHID = '1'
 AND SHR.PROBLEMID = '1'
 AND TO_CHAR(SHR.DUEDATE,'IYYYIW') = WEEK.WEEK -- 추가
)
 SELECT GROUPING(ORGLLEVEL1) SORT1, GROUPING(APPARELGROUP) SORT2, GROUPING(CATEGORYNAME) SORT3
 ...중략...
 FROM DATA
 GROUP BY GROUPING SETS((),(CATEGORYNAME, CATEGORYID),(ORGLLEVEL1,APPARELGROUP, CATEGORYNAME, CATEGORYID))
 HAVING GROUPING(ORGLLEVEL1)||GROUPING(APPARELGROUP)||GROUPING(CATEGORYNAME) NOT IN ('111')
)
 ORDER BY SORT1 DESC, ORGLLEVEL1, SORT2 DESC, APPARELGROUP , MEASURE , MEASURENAME
```

## Parallel Processing과 오해

### 성능문제 SQL - Parallel Processing

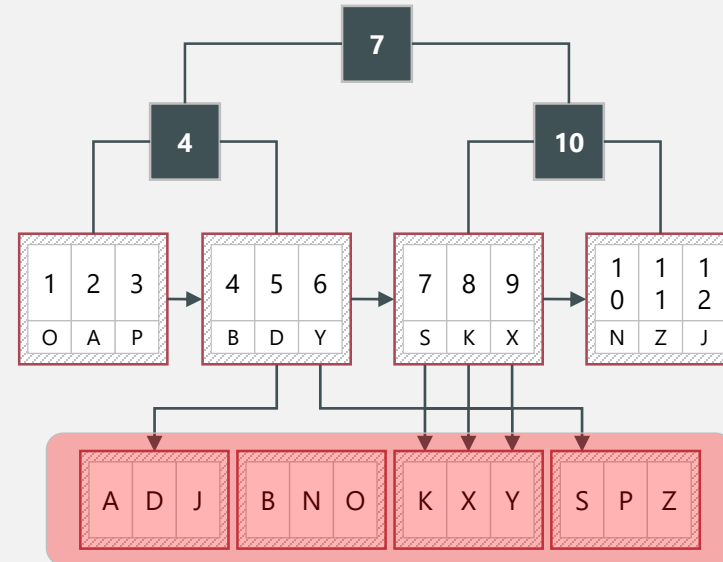
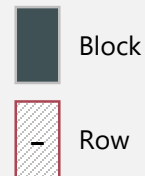
| Id       | Operation                  | Name               | A-Rows | A-Time      | Buffers |
|----------|----------------------------|--------------------|--------|-------------|---------|
| 0        | SELECT STATEMENT           |                    | 24     | 00:00:02.62 | 401     |
| 1        | SORT AGGREGATE             |                    | 0      | 00:00:00.01 | 0       |
| 2        | FIRST ROW                  |                    | 0      | 00:00:00.01 | 0       |
| * 3      | INDEX RANGE SCAN (MIN/MAX) | IDX_RPT_SALES      | 0      | 00:00:00.01 | 0       |
| 4        | PX COORDINATOR             |                    | 24     | 00:00:02.62 | 401     |
| 5        | PX SEND QC (ORDER)         | :TQ10008           | 0      | 00:00:00.01 | 0       |
| 6        | SORT ORDER BY              |                    | 0      | 00:00:00.01 | 0       |
| 7        | PX RECEIVE                 |                    | 0      | 00:00:00.01 | 0       |
| 8        | PX SEND RANGE              | :TQ10007           | 0      | 00:00:00.01 | 0       |
| 9        | VIEW                       |                    | 0      | 00:00:00.01 | 0       |
| * 10     | FILTER                     |                    | 0      | 00:00:00.01 | 0       |
| 11       | SORT GROUP BY              |                    | 0      | 00:00:00.01 | 0       |
| ...중략... |                            |                    |        |             |         |
| 29       | PX RECEIVE                 |                    | 0      | 00:00:00.01 | 0       |
| 30       | PX SEND HASH               | :TQ10003           | 0      | 00:00:00.01 | 0       |
| 31       | PX BLOCK ITERATOR          |                    | 0      | 00:00:00.01 | 0       |
| * 32     | TABLE ACCESS FULL          | RPT_SHORTREASONSMM | 0      | 00:00:00.01 | 0       |
| 33       | BUFFER SORT                |                    | 0      | 00:00:00.01 | 0       |
| 34       | PX RECEIVE                 |                    | 0      | 00:00:00.01 | 0       |
| 35       | PX SEND HASH               | :TQ10001           | 0      | 00:00:00.01 | 0       |
| 36       | INDEX FULL SCAN            | PK_RPT_SITE        | 3971   | 00:00:00.01 | 9       |
| 37       | BUFFER SORT                |                    | 0      | 00:00:00.01 | 0       |
| 38       | PX RECEIVE                 |                    | 0      | 00:00:00.01 | 0       |
| 39       | PX SEND HASH               | :TQ10002           | 0      | 00:00:00.01 | 0       |
| 40       | INDEX FULL SCAN            | PK_RPT_SITE        | 3971   | 00:00:00.01 | 9       |

## Covering Index와 Parallel Processing

### 일반 B-Tree Index

```
SELECT c2 FROM t
WHERE C1 >= 5 and
C1 <= 9;
```

TABLE ACCESS BY INDEX ROWID  
INDEX RANGE SCAN



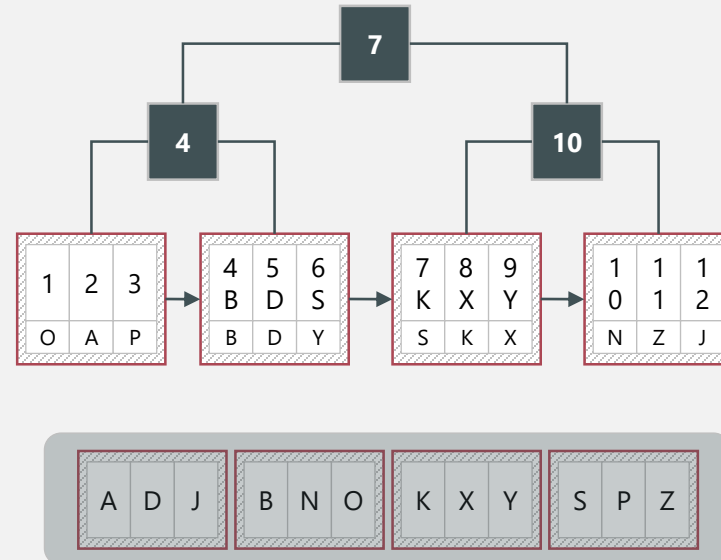
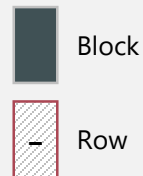
## Covering Index와 Parallel Processing

### Covering Index!

```
CREATE INDEX _ ON t(c1, c2)
```

```
SELECT c2 FROM t
WHERE C1 >= 5 and
C1 <= 9;
```

**INDEX RANGE SCAN**





## Covering Index와 Parallel Processing

### 개선 전 SQL/XPLAN

```
SELECT ROWNUM no ,
 sales_sum ,
 cust_id
FROM (
 SELECT SUM(s.amount_sold) sales_sum ,
 s.cust_id cust_id
 FROM sales_info s
 GROUP BY s.cust_id
 ORDER BY 1. DESC
)
WHERE ROWNUM <= 10 ;
```

| Id  | Operation             | Name       | Starts | A-Rows | A-Time      | Buffers | Reads |
|-----|-----------------------|------------|--------|--------|-------------|---------|-------|
| 0   | SELECT STATEMENT      |            | 1      | 10     | 00:00:46.30 | 443K    | 443K  |
| * 1 | COUNT STOPKEY         |            | 1      | 10     | 00:00:46.30 | 443K    | 443K  |
| 2   | VIEW                  |            | 1      | 10     | 00:00:46.30 | 443K    | 443K  |
| * 3 | SORT ORDER BY STOPKEY |            | 1      | 10     | 00:00:46.30 | 443K    | 443K  |
| 4   | HASH GROUP BY         |            | 1      | 7059   | 00:00:46.30 | 443K    | 443K  |
| 5   | TABLE ACCESS FULL     | SALES_INFO | 1      | 91M    | 00:00:15.89 | 443K    | 443K  |

Predicate Information (identified by operation id):

- 1 - filter(ROWNUM<=10)
- 3 - filter(ROWNUM<=10)

### Default Partition의 관리가 필요한 사례

#### — 개선을 위한 정보조사 및 Covering Index 생성

##### [ SALES\_INFO 테이블 사이즈 체크 ]

```
SELECT SUM(blocks) blks ,
 SUM(bytes/1024/1024) mb
FROM dba_segments
WHERE segment_name =upper('sales_info');
```

```
BLKS MB

450560 3520
```

##### [ 개선을 위한 Covering Index 생성 ]

```
CREATE INDEX sales_info_ix_01 ON sales_info(amount_sold , cust_id) ;
```

##### [ 생성된 Covering Index 사이즈 체크 ]

```
SELECT SUM(blocks) blks ,
 SUM(bytes/1024/1024) mb
FROM dba_segments
WHERE segment_name =upper('sales_info_ix_01')
```

```
BLKS MB

253952 1984
```

## Default Partition의 관리가 필요한 사례

### — Covering Index 생성 후 Parallel Index Fast Full Scan 수행 - 개선 후 SQL/XPLAN

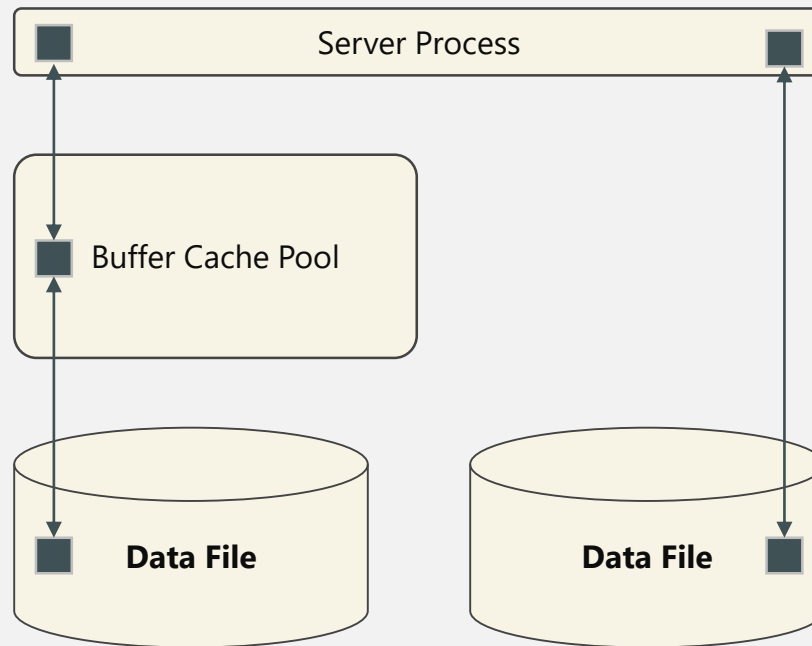
```
SELECT ROWNUM no , sales_sum , cust_id
FROM (
 SELECT /*+ parallel_index(s, 4) index_ffs(s) */
 SUM(s.amount_sold) sales_sum , s.cust_id cust_id
 FROM sales_info s
 GROUP BY s.cust_id
 ORDER BY 1 DESC
)
WHERE ROWNUM <= 10 ;
```

| Id   | Operation             | Name             | Starts | A-Rows | A-Time      | Buffers |
|------|-----------------------|------------------|--------|--------|-------------|---------|
| 0    | SELECT STATEMENT      |                  | 1      | 10     | 00:00:07.80 | 5       |
| * 1  | COUNT STOPKEY         |                  | 1      | 10     | 00:00:07.80 | 5       |
| 2    | PX COORDINATOR        |                  | 1      | 10     | 00:00:07.76 | 5       |
| 3    | PX SEND QC (ORDER)    | :TQ10002         | 0      | 0      | 00:00:00.01 | 0       |
| 4    | VIEW                  |                  | 0      | 0      | 00:00:00.01 | 0       |
| * 5  | SORT ORDER BY STOPKEY |                  | 0      | 0      | 00:00:00.01 | 0       |
| 6    | PX RECEIVE            |                  | 0      | 0      | 00:00:00.01 | 0       |
| 7    | PX SEND RANGE         | :TQ10001         | 0      | 0      | 00:00:00.01 | 0       |
| * 8  | SORT ORDER BY STOPKEY |                  | 0      | 0      | 00:00:00.01 | 0       |
| 9    | HASH GROUP BY         |                  | 0      | 0      | 00:00:00.01 | 0       |
| 10   | PX RECEIVE            |                  | 0      | 0      | 00:00:00.01 | 0       |
| 11   | PX SEND HASH          | :TQ10000         | 0      | 0      | 00:00:00.01 | 0       |
| 12   | HASH GROUP BY         |                  | 0      | 0      | 00:00:00.01 | 0       |
| 13   | PX BLOCK ITERATOR     |                  | 0      | 0      | 00:00:00.01 | 0       |
| * 14 | INDEX FAST FULL SCAN  | SALES_INFO_IX_01 | 0      | 0      | 00:00:00.01 | 0       |

### Default Partition의 관리가 필요한 사례

#### I/O Pattern

- Buffer Cache 혹은 Keep Buffer Pool 등 Memory I/O가 성능상 유리!



## In-Memory Parallel Processing

### In-Memory PQ Test

[ 테스트를 위한 테스트 테이블 생성(약 350MB) ]

```
CREATE TABLE in_memory_pq_test AS
SELECT rownum num,b.*
FROM sales,
 (SELECT LEVEL FROM dual CONNECT BY LEVEL <= 26) b ;
```

[ Direct Path Mode PQ 수행 ]

```
SELECT /*+ parallel(i 4) */ COUNT(*)
FROM in_memory_pq_test i;
```

| Id | Operation           | Name              | Rows | Cost (%CPU) | Time     | TQ    | IN-OUT |
|----|---------------------|-------------------|------|-------------|----------|-------|--------|
| 0  | SELECT STATEMENT    |                   | 1    | 3545 (2)    | 00:05:53 |       |        |
| 1  | SORT AGGREGATE      |                   | 1    |             |          |       |        |
| 2  | PX COORDINATOR      |                   |      |             |          |       |        |
| 3  | PX SEND QC (RANDOM) | :TQ10000          | 1    |             |          | Q1,00 | P->S   |
| 4  | SORT AGGREGATE      |                   | 1    |             |          | Q1,00 | PCWP   |
| 5  | PX BLOCK ITERATOR   |                   | 28M  | 3545 (2)    | 00:05:53 | Q1,00 | PCWC   |
| 6  | TABLE ACCESS FULL   | IN_MEMORY_PQ_TEST | 28M  | 3545 (2)    | 00:05:53 | Q1,00 | PCWP   |

## In-Memory Parallel Processing

### In-Memory PQ Test

#### [ Statistics Info ]

##### Statistics

```

 49 recursive calls
 0 db block gets
 46419 consistent gets
 46363 physical reads
 0 redo size
 542 bytes sent via SQL*Net to client
 520 bytes received via SQL*Net from client
 2 SQL*Net roundtrips to/from client
 6 sorts (memory)
 0 sorts (disk)
 1 rows processed
```

## In-Memory Parallel Processing

### In-Memory PQ Test

#### [ 대상 세그먼트를 Keep Buffer에 Caching ]

```
ALTER TABLE IN_MEMORY_PQ_TEST STORAGE (BUFFER_POOL KEEP);
```

```
SELECT COUNT(*) FROM IN_MEMORY_PQ_TEST;
```

#### [ Caching 결과 확인 ]

```
SELECT COUNT(*)
FROM v$bh
WHERE objd= (
 SELECT data_object_id
 FROM user_objects
 WHERE object_name='IN_MEMORY_PQ_TEST'
)
AND status='xcur' ;
```

```
 COUNT(*)

 46073
```

## In-Memory Parallel Processing

### In-Memory PQ Test

```
ALTER SESSION SET "_PARALLEL_CLUSTER_CACHE_POLICY"=CACHED;
```

```
SELECT /*+ parallel(i 4) */ COUNT(*)
FROM in_memory_pq_test i;
```

| Id | Operation           | Name              | Rows | Cost (%CPU) | Time     | TQ    | IN-OUT |
|----|---------------------|-------------------|------|-------------|----------|-------|--------|
| 0  | SELECT STATEMENT    |                   | 1    | 3545 (2)    | 00:00:43 |       |        |
| 1  | SORT AGGREGATE      |                   | 1    |             |          |       |        |
| 2  | PX COORDINATOR      |                   |      |             |          |       |        |
| 3  | PX SEND QC (RANDOM) | :TQ10000          | 1    |             |          | Q1,00 | P->S   |
| 4  | SORT AGGREGATE      |                   | 1    |             |          | Q1,00 | PCWP   |
| 5  | PX BLOCK ITERATOR   |                   | 28M  | 3545 (2)    | 00:00:43 | Q1,00 | PCWC   |
| 6  | TABLE ACCESS FULL   | IN_MEMORY_PQ_TEST | 28M  | 3545 (2)    | 00:00:43 | Q1,00 | PCWP   |

### Statistics

```

12 recursive calls
0 db block gets
46805 consistent gets
0 physical reads
0 redo size
542 bytes sent via SQL*Net to client
520 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1 rows processed

```



### Keep Buffer Pool?

- Oracle에서 지원하는 Multi Buffer Pool의 한 종류.
- 보통 Segment 전체를 I/O 하는 등의 업무를 위해 Memory 내에 Caching 하기 위해 존재함.
- 핵심은 Disk I/O를 최소화하여 빠른 응답시간을 유도하는 것이 핵심!

### ■ Keep Buffer Pool 활용 순서

- Keep 대상 Segment 선정
- OS Memory & SGA 여유 공간 확인
- Keep Buffer Pool 설정
- 대상 Segment 속성 변경
- 대상 Segment Caching
- Keep 효율성 체크
- Keep 대상 선정 기준

## Keep Buffer Pool 설정 순서

### Keep 대상 Segment 선정

- 선정 기준 1 : 업무의 중요도
- 선정 기준 2 : Segment의 크기
- 선정 기준 3 : Table Full Scan / Index Fast Full Scan, Disk I/O



Table Full Scan/  
Index FFS



Disk I/O

### Keep Buffer Pool 설정 순서

#### Segment 속성 변경

##### [ 일반 테이블/인덱스 속성 변경 ]

```
SQL> ALTER TABLE T1 STORAGE (BUFFER_POOL KEEP);
```

Table altered.

```
SQL> ALTER INDEX T1_PK STORAGE (BUFFER_POOL KEEP);
```

Index altered.

##### [ Partition 테이블/인덱스 속성 변경 ]

```
SQL> ALTER TABLE P1 MODIFY PARTITION P1_1 STORAGE (BUFFER_POOL KEEP);
```

Table altered.

```
SQL> ALTER INDEX P1_ID1 MODIFY PARTITION P1_ID1_1 STORAGE (BUFFER_POOL KEEP);
```

Index altered.

※ 참고 : Composite Partition 테이블의 경우, Main Partition Table 단위로만 Keep 설정 가능

### Keep Buffer Pool 설정 순서

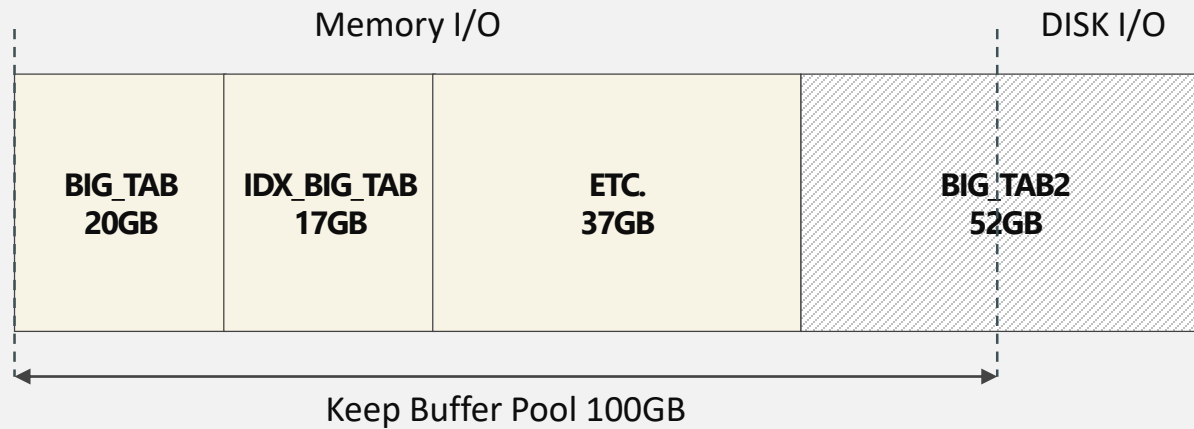
#### Segment Caching

- 대상 Segment의 속성 변경
- 대상 Segment Full Scan 수행!
- 대상 Segment를 Access 하면, Disk I/O 발생하지 않음. 예외 사항이 아니면...
- 위의 예외 사항은 『Keep Buffer 유지보수 및 상주 전략』을 참조

## Keep Buffer Pool 설정 순서

### Segment Caching

- Keep Buffer Pool 설정하여 사용하다 보면, 필연적으로 위와 같은 상황 발생.
- Keep Buffer Pool에 경합이 발생하고 Disk I/O가 다시 발생할 수 있음.
- 이 때, 필요한 개념이 Keep Buffer Pool 유지 보수 전략임.



## Keep Buffer Pool 설정 순서

### Keep Buffer Pool 유지보수 전략

- 업무 중요도에 따른 Caching 대상 세그먼트 선정
- 해당 세그먼트의 적극적 Partitioning 방안 마련
  1. Partition 테이블/인덱스 : 업무 성격 및 Access Pattern을 감안한 선별적 Keeping
  2. Non Partition 테이블/인덱스 : 업무 중요도 파악 및 해당 세그먼트의 주기적 Size 증가 량 체크
- 신규 생성 대상 세그먼트의 경우, Caching 대상인지 결정
- Keep Buffer Pool 효율성 Report
- 모든 사안을 관리하는 자동화 프로시저 작성 및 수행

## Keep Buffer Pool 설정 순서

### Keep Buffer Pool 상주 전략

- 업무 중요도 파악
- 업무 중요도에 상응하는 세그먼트 목록 정리
- 해당 세그먼트의 주기적 Access
- 모든 사안을 관리하는 자동화 프로시저 작성 및 수행

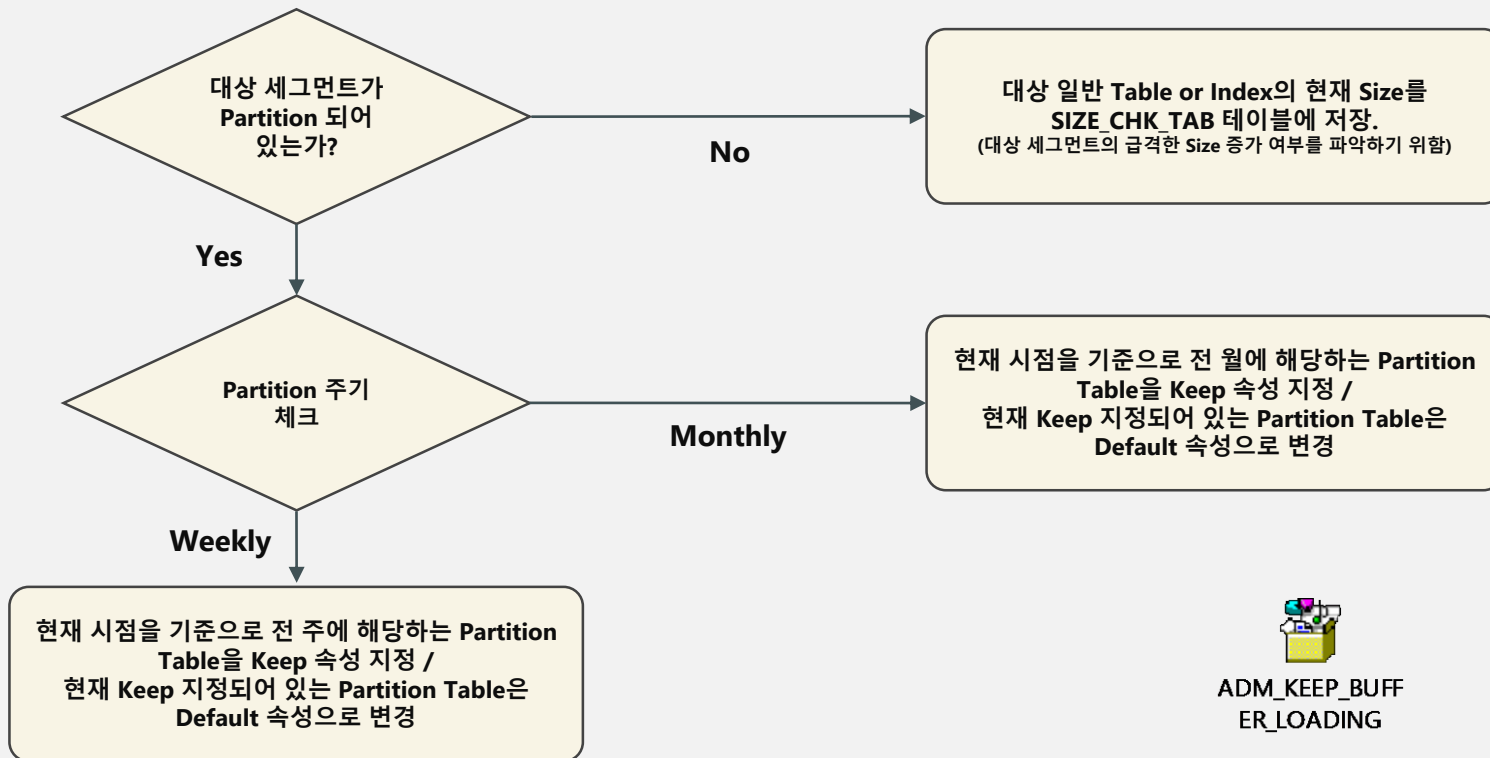
### 특정 상황을 가정한 유지보수 전략 및 상주 전략 프로시저 작성 예

- Partition 테이블은 월 단위, 주 단위 Partition 테이블만 존재
- 월 단위 Partition 테이블의 조회조건은 항상 전 달의 데이터가 대상이 됨
- 주 단위 Partition 테이블의 조회조건은 항상 전 주의 데이터가 대상이 됨



## Keep Buffer Pool 설정 순서

### Keep Buffer Pool 효율성 유지 방안



ADM\_KEEP\_BUFF  
ER\_LOADING

## Result Cache 활용

### Result Cache 동작 원리 – 최초 실행

- Result Cache SQL이 실행되면 가장 먼저 Shared Pool 영역의 Result Cache 메모리에서 Object의 Caching 여부를 확인한다.
- Object가 Caching되어 있지 않다면 Buffer Cache에서 Block을 찾는다.
- Buffer Cache에 원하는 Block이 존재하지 않으면 Disk에서 Block을 읽어 Buffer Cache에 전송한다.
- 그 결과 값을 Query한 세션에 전송한다.
- 마지막으로 Result Cache 영역에 SQL 결과 값을 저장한다.

### Result Cache 동작 원리 – 반복 실행

- Result Cache SQL이 실행되면 가장 먼저 Shared Pool 영역의 Result Cache 메모리에서 Object의 Caching 여부를 확인한다.
- Caching된 정보가 존재 하면 I/O를 발생시키지 않고 Caching된 결과 값을 요청한 세션에게 전송한다.

## Result Cache 활용

### Result Cache 기능 활용한 성능 개선 테스트

#### [ 테이블 생성 스크립트 ]

```
CREATE TABLE tb_rc_test_yyyymmdd AS
SELECT TO_CHAR(ADD_MONTHS(SYSDATE , - 1) , 'YYYYMM') ||
 TRIM(TO_CHAR(ROUND(dbms_random.value(1 , 31)) , '09')) ||
 TRIM(TO_CHAR(ROUND(dbms_random.value(0 , 23)) , '09')) ||
 TRIM(TO_CHAR(ROUND(dbms_random.value(00 , 59)) , '09')) sdate ,
 dbms_random.string('U' , 2) prod ,
 ROUND(dbms_random.value(100 , 100000)) amt1 ,
 ROUND(dbms_random.value(10 , 10000)) amt2 ,
 ROUND(dbms_random.value(1000 , 100)) amt3
FROM dual
CONNECT BY LEVEL <= 10000000 ;
```

## Result Cache 활용

### Result Cache 기능 활용한 성능 개선 테스트

#### [ Result Cache 기능을 사용하지 않은 경우 ]

```
SELECT SUBSTR(SDATE,1,6) SDATE,
PROD,
SUM(AMT1) AMT1,
SUM(AMT2) AMT2,
SUM(AMT3) AMT3
FROM TB_RC_TEST_YYYYMMDD
GROUP BY SUBSTR(SDATE,1,6), PROD
```

일 별로 판매실적을 조회하는 SQL이 반복 수행됨.

| call    | count | cpu   | elapsed | disk  | QUERY | current | rows |
|---------|-------|-------|---------|-------|-------|---------|------|
| Parse   | 1     | 0.00  | 0.00    | 0     | 0     | 0       | 0    |
| Execute | 1     | 0.00  | 0.00    | 0     | 0     | 0       | 0    |
| Fetch   | 47    | 15.62 | 15.80   | 46944 | 46948 | 0       | 676  |
| total   | 49    | 15.62 | 15.80   | 46944 | 46948 | 0       | 676  |

Rows (1st) Row Source OPERATION

```

676 SORT GROUP BY (cr=46948 pr=46944 pw=0 time=15803245)
10000000 TABLE ACCESS FULL TB_RC_TEST_YYYYMMDD (cr=46948 pr=46944 pw=0 time=3144528)
```

## Result Cache 활용

### Result Cache 기능 활용한 성능 개선 테스트

#### [ Result Cache 기능을 사용하는 경우(최초 수행) ]

```
SELECT /*+ RESULT_CACHE */
SUBSTR(SDATE,1,6) SDATE,
PROD,
SUM(AMT1) AMT1,
SUM(AMT2) AMT2,
SUM(AMT3) AMT3
FROM TB_RC_TEST_YYYYMMDD
GROUP BY SUBSTR(SDATE,1,6), PROD
```

최초 수행 시에는 Result Cache의 성능 효과를 볼 수 없음.

| call    | count | cpu   | elapsed | disk  | QUERY | current | rows |
|---------|-------|-------|---------|-------|-------|---------|------|
| Parse   | 1     | 0.00  | 0.00    | 0     | 0     | 0       | 0    |
| Execute | 1     | 0.00  | 0.00    | 0     | 0     | 0       | 0    |
| Fetch   | 47    | 15.62 | 15.80   | 46944 | 46948 | 0       | 676  |
| total   | 49    | 15.62 | 15.80   | 46944 | 46948 | 0       | 676  |

Rows (1st) Row Source OPERATION

```

676 SORT GROUP BY (cr=46948 pr=46944 pw=0 time=15803245)
10000000 TABLE ACCESS FULL TB_RC_TEST_YYYYMMDD (cr=46948 pr=46944 pw=0 time=3144528)
```

Result Cache 활용

Result Cache 기능 활용한 성능 개선 테스트

[ Result Caching 여부 확인 ]

```
SELECT id ,
 TYPE ,
 status ,
 bucket_no ,
 hash ,
 name
FROM v$result_cache_object ;
```

| ID | TYPE       |           | STATUS    | BUCKET_NO | HASH   | NAME                                                                                                                                                                                    |
|----|------------|-----------|-----------|-----------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Dependency | Published | 660       | 319061    |        | DBAADM.TB_RC_TEST_YYYYMMDD                                                                                                                                                              |
| 1  | RESULT     |           | Published | 2011      | 159411 | SELECT /*+ RESULT_CACHE */<br>SUBSTR(SDATE,1,6) SDATE,<br>PROD,<br>SUM(AMT1) AMT1,<br>SUM(amt2) AMT2,<br>SUM(AMT3) AMT3<br>FROM TB_RC_TEST_YYYYMMDD<br>GROUP BY SUBSTR(SDATE,1,6), PROD |

## Result Cache 활용

### Result Cache 기능 활용한 성능 개선 테스트

#### [ Result Cache 기능을 사용하는 경우(반복 수행) ]

```
SELECT /*+ RESULT_CACHE */
 SUBSTR(sdate , 1 , 6) sdate ,
 prod ,
 SUM(amt1) amt1 ,
 SUM(amt2) amt2 ,
 SUM(amt3) amt3
FROM tb_rc_test_yyyymmdd
GROUP BY SUBSTR(sdate , 1 , 6) , prod
```

| call    | count | cpu  | elapsed | disk | QUERY | current | rows |
|---------|-------|------|---------|------|-------|---------|------|
| Parse   | 1     | 0.00 | 0.00    | 0    | 0     | 0       | 0    |
| Execute | 1     | 0.00 | 0.00    | 0    | 0     | 0       | 0    |
| Fetch   | 47    | 0.00 | 0.00    | 0    | 0     | 0       | 676  |
| total   | 49    | 0.00 | 0.00    | 0    | 0     | 0       | 676  |

Rows (1st) Row Source OPERATION

```

676 RESULT CACHE gnxxqgpxppdavj80b6rddg4qsqj (cr=0 pr=0 pw=0 time=20 us)
 0 SORT GROUP BY (cr=0 pr=0 pw=0 time=0)
 0 TABLE ACCESS FULL TB_RC_TEST_YYYYMMDD (cr=0 pr=0 pw=0 time=0)
```

## Result Cache 활용

### Result Cache 기능 활용한 성능 개선 테스트

[ Result Cache 기능을 사용하는 경우(동일한 쿼리 블록에 적용) ]

```
SELECT SDATE, SUM(SUM_AMT)
FROM (SELECT SDATE,
NVL(AMT1,0) + NVL(AMT2,0) + NVL(AMT3,0) SUM_AMT
 FROM (SELECT /*+ RESULT_CACHE */
 SUBSTR(SDATE,1,6) SDATE,
 PROD,
 SUM(AMT1) AMT1,
 SUM(AMT2) AMT2,
 SUM(AMT3) AMT3
 FROM TB_RC_TEST_YYYYMMDD
 GROUP BY SUBSTR(SDATE,1,6), PROD)
 UNION ALL
 SELECT SUBSTR(SDATE,1,6) SDATE,
 SUM(AMT1) + SUM(AMT2) + SUM(AMT3) SUM_AMT
 FROM TB_RC_TEST_SYSDATE
 WHERE SDATE < :SDATE
 GROUP BY SUBSTR(SDATE,1,6), PROD
)
GROUP BY SDATE;
```

동일한 쿼리 블록을 사용하는 다른 SQL에서도  
Result Cache 효과가 적용된다.



## Result Cache 활용

### Result Cache 기능 활용한 성능 개선 테스트

[ Result Cache 기능을 사용하는 경우(동일한 쿼리 블록에 적용) ]

| call    | count | cpu  | elapsed | disk | QUERY | current | rows |
|---------|-------|------|---------|------|-------|---------|------|
| Parse   | 1     | 0.00 | 0.00    | 0    | 0     | 0       | 0    |
| Execute | 1     | 0.00 | 0.00    | 0    | 0     | 0       | 0    |
| Fetch   | 2     | 0.18 | 0.18    | 0    | 1413  | 0       | 2    |
| total   | 4     | 0.18 | 0.19    | 0    | 1413  | 0       | 2    |

Rows (1st) Row Source OPERATION

```

 2 SORT GROUP BY (cr=1413 pr=0 pw=0 time=188846)
1352 VIEW (cr=1413 pr=0 pw=0 time=2355)
1352 UNION-ALL (cr=1413 pr=0 pw=0 time=2230 us)
 676 VIEW (cr=0 pr=0 pw=0 time=578)
 676 RESULT CACHE gnqxgpxppdavj80b6rddg4qsqj (cr=0 pr=0 pw=0 time=126 us)
 0 SORT GROUP BY (cr=0 pr=0 pw=0 time=0)
 0 TABLE ACCESS FULL TB_RC_TEST_YYYYMMDD (cr=0 pr=0 pw=0 time=0)
 676 SORT GROUP BY (cr=1413 pr=0 pw=0 time=187432)
104850 TABLE ACCESS FULL TB_RC_TEST_SYSDATE (cr=1413 pr=0 pw=0 time=57449)

```

## Result Cache 활용

### Result Cache 사용시 고려사항

- DML과 Result Cache의 관계.
- Bind 변수와 입력되는 값의 상관관계.
- 특정 Object 사용시 Caching되지 않음.
  - 임시 테이블 또는 Dictionary Object (DBA\_\*, V\$\_\*, GV\$\_\* 등) 참조 시
  - 시퀀스 CURRVAL 및 NEXTVAL Column 호출 시
  - Query에서 SQL 함수를 사용 할 경우

 THANK YOU 