SQL 실행계획 관리

(Plan stability)

Author	**이경오**
Creation Date	
Last Updated	
Version	**1.0**

Version	변경일자	변경자(작성자)	주요내용
1	2018-05-08	이경오	Stored Outlines 관련 내용 작성
2	2018-05-15	이경오	SQL Profile 관련 내용 작성
3	2018-08-20	이경오	SPM 관련 내용 작성

Goodus
Breathing Life into Technology

Contents

1. Plan Stability 란?

1.1. 소개

DBMS 의 System 관점과 SQL 관점의 튜닝이 완료되어, 시스템이 안정적인 성능을 내게 되면, 가장 중요한 것이 바로 Plan Stability(실행계획 고정)이 라고 할 수 있다. Plan Stability 는 시스템의 환경 변화와는 상관없이 최적화된 실행계획으로 일정하게 SQL 문이 실행되는 것을 뜻하며, 시스템의 안정성을 위해서 매우 중요한 항목이다.

1.2. Plan Stability를 방해하는 요인

Plan Stability 를 방해하는 요인은 아래와 같다.

changes in optimizer statistics (옵티마이저 통계의 변화)
changes to the optimizer mode settings(옵티마이저 모드의 변화)
changes to parameters affecting the sizes of memory structures(메모리 관련 파라미터 변경)

그밖에 복잡 다난한 요인으로 인해 실행계획은 언제든지 바뀔 수 있다.

1.3. Plan Stability 달성을 위한 Oracle의 노력

Oracle DBMS 는 Plan Stability 를 위해서 많은 기능들을 지속적으로 출시하고 있으며, 대표적으로 Stored Outlines, SQL Profile, SPM(SQL Plan Management)이 있으며, 이번 기술 노트는 이 세가지 기능에 대해서 설명하고자 한다.

2. Stored Outlines

2.1. 소개

Stored Outlines 는 특정 SQL 문의 일정한 실행계획을 유지하도록 하는 힌트의 집합이다. Stored Outlines 로 인하여 시스템 환경 혹은 통계 정보의 변경이 발생해도 일정한 실행계획을 유지할 수 있다.
Plan Stability 는 SQL 문의 성능이 특정 시간 내에 실행되는 것을 보장하는 것을 말하여, Stored Outlines 를 사용함으로써 이것을 달성할 수 있다. Outlines 들은 OL$, OL$HINTS, OL$NODES 테이블에 저장되며 USER_OUTLINE_HINTS, ALL_OUTLINE_HINTS, DBA_OUTLINE_HINTS 뷰로 Outlines 의 정보를 조회할 수 있다.
해당 기술은 9i 부터~10g 버전에서 사용되었으며, 11g 버전부터는 뒤에서 이야기할 SPM(SQL Plan Management)의 등장으로 인해 사용되지 않고 있다. (deprecated)

2.2. Creating Outlines

■ create_stored_outlines 파라미터 설정

Outlines 오라클의 시스템 파라미터 설정으로 자동으로 생성할 수 있다. 즉 create_stored_outlines 파라미터를 TRUE 로 설정하면 자동으로 Outlines 가 생성되게 된다. 설정방법은 아래와 같다.

```
-- Switch on automatic creation of stored outlines.
ALTER SYSTEM SET create_stored_outlines=TRUE;
ALTER SESSION SET create_stored_outlines=TRUE;
```

■ 권한 부여

create_stored_outlines 파라미터가 인스턴스 혹은 세션 레벨로 설정되면 오라클은 실행되는 모든 SQL 문에 대해 세션 인스턴스 혹은 세션 레벨로 Outlines 를 생성한다. Outlines 을 생성하기 위해서는 생성 권한을 부여해야 한다. 아래는 Outlines 생성 권한을 부여하는 예시이다.

```
-- sysdba 로 접속하여 SCOTT 계정에게 권한 부여
GRANT CREATE ANY OUTLINE TO SCOTT;
GRANT EXECUTE_CATALOG_ROLE TO SCOTT;
```

■ CREATE 문을 이용한 수동 생성

Outlines 를 특정 SQL 문에 대해 수동으로 생성이 가능하다. CREATE OUTLINE 문 혹은 DBMS_OUTLN.CREATE_OUTLINE 프로시저를 사용함으로써 특정 SQL 에 대한 Outlines 를 생성할 수 있다. 아래는 Outlines 를 수동으로 생성하는 예시이다.

```
-- SCOTT 계정으로 접속
-- Create an outline for a specific SQL statement.
CREATE OUTLINE emp_dept FOR CATEGORY scott_outlines ON
SELECT
e.empno
, e.ename
, d.dname
FROM
emp e
, dept d
WHERE e.deptno = d.deptno;
```

```
-- Check the outline as been created correctly.
SELECT
name
, category
, sql_text
FROM
user_outlines
WHERE category = 'SCOTT_OUTLINES';

-- List the hints associated with the outline.
SELECT
node
, stage
, join_pos
, hint
FROM
user_outline_hints
WHERE  name  =  'EMP_DEPT';
```

■ **Procedure 를 이용한 수동 생성**

아래는 DBMS_OUTLN.CREATE_OUTLINE procedure 를 사용하여 Outlines 를 create 하는 예제이다. V$SQL 에 존재하는 HAHS_VALUE 및 CHILD_NUMBER 를 기준으로 Outlines 를 생성할 수 있다.

```
-- Run a SQL statement to get it into the shared pool.
SELECT /* goodus_outlie_test */
           e.empno
         , e.ename
         , d.dname
         , e.job
  FROM
         emp e
        , dept d
WHERE e.deptno = d.deptno
    AND d.dname = 'SALES';
```

```
-- Identify the SQL statement in the V$SQL view.
SELECT
            hash_value
          , child_number
          , sql_text
   FROM v$sql
  WHERE sql_text LIKE '%goodus_outlie_test%';

-- Create an outline for the statement.
BEGIN
DBMS_OUTLN.create_outline(
                                    hash_value       => 3689633449,
                                    child_number     => 0,
                                    category         => 'SCOTT_OUTLINES'
                                   );
END;
/

-- Check the outline as been created correctly.
SELECT
            name
          , category
          , sql_text
   FROM user_outlines
WHERE category = 'SCOTT_OUTLINES';

-- List the hints associated with the outline.
SELECT
            node
          , stage
          , join_pos
```

```
          , hint
  FROM user_outline_hints
WHERE name = 'SYS_OUTLINE_18053008493618810';
```

2.3. Using Outlines

지금까지 Outlines 생성에 대해서 알아보았고, 지금부터는 Outlines 을 사용하는 방법에 대해서 알아보도록 한다. Outlines 는 생성 후 바로 사용할 수 있는 것이 아니다.

■ Outlines 사용을 하고 있지 않는 상태

아래의 실습을 통해 Outlines 가 사용되지 않는 것을 확인한다.

```
-- Check if the outlines have been used.
SELECT
          name
        , category
        , used
  FROM user_outlines;

-- Issue both statements again.
SELECT
e.empno
, e.ename
, d.dname
FROM
emp e
, dept d
WHERE e.deptno = d.deptno;

SELECT /* goodus_outlie_test */
          e.empno
        , e.ename
        , d.dname
        , e.job
 FROM
        emp e
       , dept d
```

```
WHERE e.deptno = d.deptno
AND d.dname = 'SALES';

-- Check if the outlines have been used.
SELECT
          name
        , category
        , used
   FROM user_outlines;
```

생성된 Outlines 가 사용되지 않고 있다는 것을 확인 할 수 있다.

■ outlines 사용 설정

해당 Outlines 을 enable 하기 위해서 해당 세션의 query_rewrite_enabled 파라미터를 TRUE 로 설정하고 use_stored_outlines 파라미터의 값을 outline category 의 이름으로 지정해야 한다.

```
-- Enable stored outlines.
ALTER SESSION SET query_rewrite_enabled=TRUE;
ALTER SESSION SET use_stored_outlines=SCOTT_OUTLINES;
```

use_stored_outlines 파라미터는 value 는 category 의 "이름", "false", "true" 의 세가지로 설정할 수 있다. 해당 파라미터를 true 로 설정하면 default outline category 가 사용되게된다.

■ outlines 를 사용한 상태

```
-- Issue both statements again.
SELECT
e.empno
, e.ename
, d.dname
FROM
emp e
, dept d
WHERE e.deptno = d.deptno;

SELECT /* goodus_outlie_test */
          e.empno
        , e.ename
```

```
          , d.dname
          , e.job
 FROM
          emp e
         , dept d
WHERE e.deptno = d.deptno
AND d.dname = 'SALES';

-- Check if the outlines have been used.
SELECT
           name
         , category
         , used
FROM  user_outlines;
```

outlines 에 등록된 SQL 문은 고정된 실행계획으로 실행된다. 이로써 실행 계획의 고정이 되며, Plan Stability 를 달성할 수 있다. 생성한 outlines 를 drop 하기 위해서는 DBMS_OUTLN.drop_by_cat procedure 를 사용하며 사용 예시는 아래와 같다.

```
BEGIN
      DBMS_OUTLN.drop_by_cat (cat => 'SCOTT_OUTLINES');
END;
/
```

3. Stored Outlines을 이용한 실행계획 강제 변경

Stored Outlines 를 이용하여 특정 SQL 의 수정없이 실행계획을 변경 할수 있다. SQL 문장을 수정할 수 없는 경우(솔루션, 긴급상황 등) 해당 방법을 통해 실행계획을 강제로 변경할 수 있다.

3.1. 실행계획 강제 변경 절차

1. 느린 SQL 을 SQ 라고 칭하고, 해당 SQL 에 대해 outline SQO 를 만든다.
2. 빠른 SQL 을 FQ 라고 칭하고, 해당 SQL 에 대해 outline FQO 를 만든다.
3. FQO 를 SQO 으로 overwrite 한다. (지금부턴 SQO 가 빠른 실행계획임)
4. SQ 를 실행 시 SQO 가 수행되도록 한다.
5. 이후 SQ 가 hard parsing 될때 SQO 에 의해 실행계획이 생성된다.
6. SQO 는 실제로는 개선된 FQ 의 outline 인 FQO 를 가지고 있으므로 개선된 실행계획이 생성된다.

3.2. 실습환경 구축

T_OUTLN 테이블을 생성하며, C1 에 대한 단일컬럼 인덱스인 T_OUTLN_I1 와 C2 컬럼에 대한 단일 컬럼 인덱스인 T_OUTLN_I2 인덱스를 생성한다. 또한 C1 컬럼의 값은 선택도가 매우 낮게 데이터를 입력하고, C2 컬럼은 선택도가 높게 데이터를 입력한다. 이렇게 되면 인덱스 스캔시 T_OUTLN_I1 인덱스 스캔은 매우 효율적이고, T_OUTLN_I2 인데스 스캔은 비효율이 발생하게 된다.

```
--SCOTT 계정으로 접속
CREATE TABLE T_OUTLN(C1 INT, C2 INT);
CREATE INDEX T_OUTLN_I1 ON T_OUTLN(C1);
CREATE INDEX T_OUTLN_I2 ON T_OUTLN(C2);

INSERT INTO T_OUTLN
SELECT LEVEL, --C1 이 선택도가 매우 낮음
           MOD(LEVEL, 10) --C2 이 선택도가 높음
  FROM DUAL CONNECT BY LEVEL <= 100000;

COMMIT;

--통계정보생성
EXEC DBMS_STATS.GATHER_TABLE_STATS(USER, 'T_OUTLN', CASCADE=>TRUE, NO_INVALIDATE=>FALS
E);
```

3.3. SQ(Slow Query) SQL실행

현재 C1 컬럼의 선택도가 극히 낮으므로, 아래의 SQL 을 실행 시 T_OUTLN_I1 인덱스 스캔이 매우 유리한 상황이다. 해당 SQL 을 우선 SQ 라고 칭한다.

```
--SQ SQL 실행
SELECT /* SQ */ *
FROM T_OUTLN
WHERE C1 = 1
    AND C2 = 1;
-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |            |      1 |        |      1 |00:00:00.01 |       4 |
```

```
|*  1 |  TABLE ACCESS BY INDEX ROWID| T_OUTLN    |      1 |      1 |      1 |00:00:00.01 |       4 |
|*  2 |   INDEX RANGE SCAN          | T_OUTLN_I1 |      1 |      1 |      1 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------------------

-- T_OUTLN_I1 인덱스를 스캔하여, 성능상 아무런 문제가 없음
```

3.4. SQ(Slow Query) SQL의 outlines 생성

SQL SQL 은 T_OUTLN_I1 인덱스를 스캔하도록 Outlines 를 생성하여 실행계획을 고정하였다.

```
--SCOTT 계정으로 접속
ALTER SESSION SET CREATE_STORED_OUTLINES = TEST_OUTLN;

--sq SQL 실행
SELECT /* SQ */ *
FROM T_OUTLN
WHERE C1 = 1
    AND C2 = 1;

ALTER SESSION SET CREATE_STORED_OUTLINES = FALSE;

--생성된 OUTLINE 확인
SELECT NAME, CATEGORY, USED, ENABLED, SQL_TEXT
FROM USER_OUTLINES;

SELECT NAME, HINT
FROM USER_OUTLINE_HINTS;

--생성된 OUTLINE 을 실행계획으로 고정함
ALTER SESSION SET USE_STORED_OUTLINES = TEST_OUTLN;
```

3.5. SQ(Slow Query) SQL이 느린 SQL이 되도록 데이터 분포 조정

지금부턴 SQ SQL 의 성능이 저하되도록 C1 컬럼의 선택도가 높게 데이터를 새로 입력한다. C1 인덱스를 스캔하도록 실행계획이 고정된 상태이기 때문에 C1 컬럼의 선택도가 높음에도 불구하고 C1 인덱스만 스캔하게 된다. 데이터 재입력은 아래와 같이 진행한다.

```
--데이터 삭제
DELETE
FROM T_OUTLN;
```

```
-- C2 컬럼의 인덱스 스캔이 유리하도록 데이터 분포 조정하여 INSERT
INSERT INTO T_OUTLN
SELECT
        MOD(LEVEL, 10), --C1 이 선택도가 높음
        LEVEL --C2 이 선택도가 매우 낮음
  FROM DUAL CONNECT BY LEVEL <= 100000;

COMMIT;
--통계정보생성
EXEC DBMS_STATS.GATHER_TABLE_STATS(USER, 'T_OUTLN', CASCADE=>TRUE, NO_INVALIDATE=>FALS
E);
```

SQL SQL 은 T_OUTLN_I2 인덱스 스캔이 성능상 유리하게 되었다. 하지만 T_OUTLN_I1 인덱스를 스캔하도록 실행계획이 고정되어 있는 상태이다.

```
--SQ SQL 실행
SELECT /* SQ */ *
 FROM T_OUTLN
WHERE C1 = 1
    AND C2 = 1;


-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |      1 |        |      1 |00:00:00.01 |     257 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| T_OUTLN    |      1 |      1 |      1 |00:00:00.01 |     257 |
|*  2 |   INDEX RANGE SCAN           | T_OUTLN_I1 |      1 |  10066 |  10000 |00:00:00.01 |      28 |
-------------------------------------------------------------------------------------------------
```

3.6. FQ(Fast Query) SQL에 대한 outlines 생성

FQ SQL 을 Outlines 로 등록하여 해당 SQL 은 T_OUTLN_I2 인덱스 스캔으로 고정되도록 한다.

```
--FQ SQL 에 대한 outlines 생성
--SCOTT 계정으로 접속
CREATE OUTLINE TMP_OUTLN FOR CATEGORY TEST_OUTLN ON
 SELECT /* FQ */ /*+ INDEX(T_OUTLN T_OUTLN_I2) */*
   FROM T_OUTLN
  WHERE C1 = 1
      AND C2 = 1;
```

```
--outlines 생성 확인
SELECT OL_NAME,
       SQL_TEXT,
       HINTCOUNT
  FROM OUTLN.OL$
 WHERE CATEGORY = 'TEST_OUTLN';

--FQ SQL 실행
SELECT /* FQ */ /*+ INDEX(T_OUTLN T_OUTLN_I2) */*
    FROM T_OUTLN
   WHERE C1 = 1
     AND C2 = 1
;
-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |      1 |        |      1 |00:00:00.01 |      30 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| T_OUTLN    |      1 |      1 |      1 |00:00:00.01 |      30 |
|*  2 |   INDEX RANGE SCAN           | T_OUTLN_I2 |      1 |      1 |      1 |00:00:00.01 |      29 |
-------------------------------------------------------------------------------------------------------
```

FQ SQL 은 SQ SQL 에 비해서 성능이 훨씬 빠르다.

3.7. FQ의 실행계획을 SQ의 실행계획으로 overwriting

지금부터는 FQ SQL 의 Outlines 를 SQ SQL 의 Outlines 로 overwrite 하도록 한다.
그렇게 함으로써 SQ SQL 의 SQL 변경없이 실행계획이 개선된다.
아래 SQL 을 실행하여(OL_NAME 은 사전에 조회) FQ 의 실행계획을 SQ 의 실행계획으로 overwriting 한다.

```
--SCOTT 계정으로 접속
--SQ 의 HINTCOUNT 값을 FQ 의 HINTCOUNT 로 UPDATE
UPDATE OUTLN.OL$
   SET HINTCOUNT = (SELECT HINTCOUNT
                      FROM OUTLN.OL$
                     WHERE OL_NAME = 'SYS_OUTLINE_18053009171079811') --SQ SQL 의 OL_NAME
 WHERE OL_NAME = 'TMP_OUTLN';
```

```
--SQ 의 힌트정보를 삭제
DELETE
   FROM OUTLN.OL$HINTS
 WHERE OL_NAME = 'SYS_OUTLINE_18053009171079811';

--FQ 의 OL_NAME 을 SQ 의 이름으로 변경
UPDATE OUTLN.OL$HINTS
     SET OL_NAME = 'SYS_OUTLINE_18053009171079811'
 WHERE OL_NAME = 'TMP_OUTLN';

--SQ 의 노드정보 삭제
DELETE
   FROM OUTLN.OL$NODES
 WHERE OL_NAME = 'SYS_OUTLINE_18053009171079811';

--FQ 의 OL_NAME 을 SQ 의 이름으로 변경
UPDATE OUTLN.OL$NODES
     SET OL_NAME = 'SYS_OUTLINE_18053009171079811'
 WHERE OL_NAME = 'TMP_OUTLN';

 COMMIT;
```

SQ SQL 을 실행하여 T_OUTLN_I2 인덱스 스캔을 하는지 확인한다.

```
--SQ SQL 실행
ALTER SESSION SET query_rewrite_enabled=TRUE;
ALTER SESSION SET use_stored_outlines=TMP_OUTLN;

SELECT /* SQ */ *
FROM T_OUTLN
WHERE C1 = 1
    AND C2 = 1;

--------------------------------------------------------------------------------
```

```
| Id  | Operation                    | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |      1 |        |      1 |00:00:00.01 |      30 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| T_OUTLN    |      1 |      1 |      1 |00:00:00.01 |      30 |
|*  2 |   INDEX RANGE SCAN           | T_OUTLN_I2 |      1 |      1 |      1 |00:00:00.01 |      29 |
-------------------------------------------------------------------------------------------------
```

SQ SQL 의 실행계획이 바뀐 것을 확인할 수 있다. Outlines 을 사용하면 SQL 문 자체의 변경없이 특정 SQL 의 속도를 빠르게(실행계획을 개선) 할 수 있다.

4. SQL Profile

4.1. 소개

SQL Profile 은 Oracle 10g 부터 사용할 수 있으며 SQL 문장에 힌트를 넣지 않고도 실행계획을 제어(변경)할 수 있는 방법이다. 프로그램 소스를 찾을 수 없거나, 소스를 수정하기 어려운, 예를 들면 ERP 같은 솔루션 등을 튜닝할 때 유용한 방법이다. 또한, 응급한 상황에서 소스수정/배포를 위한 시간적 여유가 없을 때 요긴하다. 11g 에서 Plan 이 바뀌어져 갑자기 성능 문제가 발생했을 때, 기존 Plan 을 이용하여 Plan 을 고정할 수 있는 방법으로 SQL Plan Baselines 가 있다. 만약 11g 가 아니거나 11g 이긴 하지만 SQL Plan Baselines 를 사용할 수 없는 상황이라면 SQL Profile 은 좋은 대안이 될 수 있다. SQL Profile 은 두가지 방법으로 사용할 수 있다. Tuning Advisor 를 이용한 방법과 직접 Plan 을 제어하는 방법이다.

4.2. 실습환경 구축

SQL Profile 에 대해 자세히 알아보기 위해 아래와 같은 실습환경을 구축한다.

```
--sysdba 로 접속
grant CREATE ANY SQL PROFILE to scott ;
grant ALTER ANY SQL PROFILE to scott ;
grant DROP ANY SQL PROFILE to scott ;

drop table t1;
drop table t2;

--테이블 생성
create table t1 (id number, name varchar2(100));
create table t2 (id number, amt number);

--데이터 입력
insert into t1
```

```
select level, 'popular'
from dual
connect by level <= 100000;

insert into t1
select level, 'rare'
from dual
connect by level <= 10;

insert into t2
select level, 100
from dual
connect by level <= 100000;

commit;

--인덱스 생성
create index t1_name on t1(name);
create index t2_id on t2(id);

--통계정보 생성
exec dbms_stats.gather_table_stats(user,'t1');
exec dbms_stats.gather_table_stats(user,'t2');
```

5. SQL Profile 기반하에 Tuning Advisor를 활용한 실행계획 고정

SQL Profile 기능 기반 하에서, Tuning Advisor 를 활용하여 특정 SQL 을 튜닝 후 해당 SQL 의 실행계획을 고정할 수 있다. 즉 특정 SQL 문을 Tuning Advisor 를 이용하여 SQL 튜닝을 진행하고 튜닝 된 실행계획을 SQL Profile 을 이용하여 실행계획 고정을 한다. 그럼 지금부터 해당 기능의 사용 절차를 자세히 알아본다.

5.1. 실행계획을 고정할 SQL실행

```
--SCOTT 계정으로 접속
--sqlplus 로접속
alter session set statistics_level = all;

--바인드 변수 선언
var name varchar2(100);
```

```
exec :name := 'popular';

--SQL 실행
select /*+ leading(t1) use_nl(t2) index(t1 t1_name) index(t2 t2_id) test sql profile */
   max(t1.name),
   max(t2.amt)
   from t1,
          t2
 where t1.id = t2.id
    and t1.name = :name;
```

5.2. SQL ID 확인

```
select
          sql_id
       , sql_text
 from v$sql
where sql_text like '%test sql profile%'
;

--결과
3sqjn04a1r84n    select /*+ leading(t1) use_nl(t2) index(t1 t1_name) index(t2 t2_id) test sql profile */
t1.name,           sum(t2.amt)    from t1,           t2   where t1.id = t2.id      and t1.name = :name
group by t1.name
```

5.3. 실행계획 확인

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR('6rrwthsp3b263',NULL,'ALLSTATS LAST, OUTLINE,
ALIAS'));

SQL_ID  6rrwthsp3b263, child number 0
-------------------------------------
select /*+ leading(t1) use_nl(t2) index(t1 t1_name) index(t2 t2_id)
test sql profile */    max(t1.name),    max(t2.amt)    from t1,          t2
where t1.id = t2.id      and t1.name = :name

Plan hash value: 1015865226

-----------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
```

```
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |         |      1 |        |      1 |00:00:00.25 |    3297 |
|   1 |  SORT AGGREGATE                       |         |      1 |      1 |      1 |00:00:00.25 |    3297 |
|   2 |   NESTED LOOPS                        |         |      1 |        |    100K|00:00:00.23 |    3297 |
|   3 |    NESTED LOOPS                       |         |      1 |  97745 |    100K|00:00:00.16 |    2970 |
|   4 |     TABLE ACCESS BY INDEX ROWID BATCHED| T1      |      1 |  97745 |    100K|00:00:00.05 |     514 |
|*  5 |      INDEX RANGE SCAN                 | T1_NAME |      1 |  97745 |    100K|00:00:00.02 |     267 |
|*  6 |     INDEX RANGE SCAN                  | T2_ID   |    100K|      1 |    100K|00:00:00.08 |    2456 |
|   7 |    TABLE ACCESS BY INDEX ROWID        | T2      |    100K|      1 |    100K|00:00:00.04 |     327 |
-------------------------------------------------------------------------------------------------------


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


   1 - SEL$1
   4 - SEL$1 / T1@SEL$1
   5 - SEL$1 / T1@SEL$1
   6 - SEL$1 / T2@SEL$1
   7 - SEL$1 / T2@SEL$1


Outline Data
-------------


  /*+
      BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      OPTIMIZER_FEATURES_ENABLE('12.1.0.1')
      DB_VERSION('12.2.0.1')
      ALL_ROWS
      OUTLINE_LEAF(@"SEL$1")
      INDEX_RS_ASC(@"SEL$1" "T1"@"SEL$1" ("T1"."NAME"))
      BATCH_TABLE_ACCESS_BY_ROWID(@"SEL$1" "T1"@"SEL$1")
      INDEX(@"SEL$1" "T2"@"SEL$1" ("T2"."ID"))
      LEADING(@"SEL$1" "T1"@"SEL$1" "T2"@"SEL$1")
      USE_NL(@"SEL$1" "T2"@"SEL$1")
      NLJ_BATCHING(@"SEL$1" "T2"@"SEL$1")
      END_OUTLINE_DATA
  */
```

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

   5 - access("T1"."NAME"=:NAME)
   6 - access("T1"."ID"="T2"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

위 SQL 은 T1_NAME 인덱스 및 T2_ID 인덱스를 이용하여 Nested Loops 조인을 한 것을 알 수 있다. 그럼 지금부터 해당 SQL 을 Tuning Advisor 를 이용해서 Tuning 을 진행하고 Tuning 된 실행계획을 SQL Profile 을 이용해서 실행계획 고정을 하도록 한다.

5.4. Tuning Advisor 실행

```
--sysdba 로 접속
Grant ADVISOR to scott;
grant select on v_$sqlarea_plan_hash to scott;

--scott 으로 접속
--Tuning Advisor Task 생성(SQL Plus 에서 실행)

--바인드 변수 선언
var v_sql_id varchar2(100);
exec :v_sql_id:= '6rrwthsp3b263';

prompt creating tuning task...
var v_task_id varchar2(100);
begin
  begin
    dbms_sqltune.drop_tuning_task(task_name => 'tuning_task_1');
  exception when others then
    null;
  end;

  :v_task_id := dbms_sqltune.create_tuning_task(
```

```
                        sql_id     => :v_sql_id,
                        task_name => 'tuning_task_1'
                      );
end;
/

prompt task id =
print :v_task_id

--Tuning Advisor 실행
prompt executing tuning task...
exec dbms_sqltune.execute_tuning_task(task_name => 'tuning_task_1');

--생성된 Tuning Advisor 확인
select task_id, task_name, status
from user_advisor_log
where task_name = 'tuning_task_1'
;

--결과
1526      tuning_task_1      COMPLETED

--Tuning Advisor 가 제시하는 실행계획 확인
select  dbms_sqltune.report_tuning_task('tuning_task_1')
from  dual;

--결과
GENERAL INFORMATION SECTION
-------------------------------------------------------------------------------
Tuning Task Name     : tuning_task_1
Tuning Task Owner    : SCOTT
Workload Type        : Single SQL Statement
Scope                : COMPREHENSIVE
Time Limit(seconds): 1800
Completion Status    : COMPLETED
Started at           : 05/30/2018 11:31:09
Completed at         : 05/30/2018 11:31:18
```

```
-------------------------------------------------------------------------------
Schema Name: SCOTT
SQL ID        : 6rrwthsp3b263
SQL Text      : select /*+ leading(t1) use_nl(t2) index(t1 t1_name) index(t2
                  t2_id) test sql profile */
                    max(t1.name),
                    max(t2.amt)
                    from t1,
                           t2
                  where t1.id = t2.id
                    and t1.name = :name
Bind Variables :
 1 -   (VARCHAR2(2000)):popular

-------------------------------------------------------------------------------
FINDINGS SECTION (3 findings)
-------------------------------------------------------------------------------

1- Statistics Finding
---------------------
   Table "SCOTT"."T2" was not analyzed.

   Recommendation
   --------------
   - Consider collecting optimizer statistics for this table.
     execute dbms_stats.gather_table_stats(ownname => 'SCOTT', tabname =>
                'T2', estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE, method_opt
                => 'FOR ALL COLUMNS SIZE AUTO');

   Rationale
   ---------
     The optimizer requires up-to-date statistics for the table in order to
     select a good execution plan.

2- Statistics Finding
---------------------
   Table "SCOTT"."T1" was not analyzed.
```

Recommendation

- Consider collecting optimizer statistics for this table.
execute dbms_stats.gather_table_stats(ownname => 'SCOTT', tabname =>
'T1', estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE, method_opt
=> 'FOR ALL COLUMNS SIZE AUTO');

Rationale

The optimizer requires up-to-date statistics for the table in order to
select a good execution plan.

3- SQL Profile Finding (see explain plans section below)

A potentially better execution plan was found for this statement.

Recommendation (estimated benefit: 86.01%)

- Consider accepting the recommended SQL profile.
execute dbms_sqltune.accept_sql_profile(task_name => 'tuning_task_1',
task_owner => 'SCOTT', replace => TRUE);

Validation results

The SQL profile was tested by executing both its plan and the original plan
and measuring their respective execution statistics. A plan may have been
only partially executed if the other could be run to completion in less time.

	Original Plan	With SQL Profile	% Improved
Completion Status:	COMPLETE	COMPLETE	
Elapsed Time (s):	.221247	.195695	11.54 %
CPU Time (s):	.220969	.042027	80.98 %
User I/O Time (s):	0	0	
Buffer Gets:	3297	460	86.04 %
Physical Read Requests:	0	0	
Physical Write Requests:	0	0	
Physical Read Bytes:	0	0	

```
  Physical Write Bytes:                0                    0
  Rows Processed:                      1                    1
  Fetches:                             1                    1
  Executions:                          1                    1

  Notes
  -----
  1. Statistics for the original plan were averaged over 5 executions.
  2. Statistics for the SQL profile plan were averaged over 6 executions.

------------------------------------------------------------------------------
EXPLAIN PLANS SECTION
------------------------------------------------------------------------------

1- Original With Adjusted Cost
------------------------------
Plan hash value: 1015865226

-----------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |          |     1 |    17 |   100K  (1)| 00:00:04 |
|   1 |  SORT AGGREGATE                          |          |     1 |    17 |            |          |
|   2 |   NESTED LOOPS                           |          | 50005 |   830K|   100K  (1)| 00:00:04 |
|   3 |    NESTED LOOPS                          |          | 50005 |   830K|   100K  (1)| 00:00:04 |
|   4 |     TABLE ACCESS BY INDEX ROWID BATCHED| T1       | 50005 |   537K|   258   (1)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN                    | T1_NAME  | 50005 |       |   133   (0)| 00:00:01 |
|*  6 |     INDEX RANGE SCAN                     | T2_ID    |     1 |       |     1   (0)| 00:00:01 |
|   7 |    TABLE ACCESS BY INDEX ROWID           | T2       |     1 |     6 |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   5 - access("T1"."NAME"=:NAME)
   6 - access("T1"."ID"="T2"."ID")
```

2- Using SQL Profile

```
-------------------
Plan hash value: 906334482

-------------------------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |     1 |    17 |       |   312   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE     |      |     1 |    17 |       |            |          |
|*  2 |   HASH JOIN         |      | 50005 |   830K|  1128K|   312   (1)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL| T1   | 50005 |   537K|       |   103   (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| T2   |   100K|   585K|       |    68   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   2 - access("T1"."ID"="T2"."ID")
   3 - filter("T1"."NAME"=:NAME)

---------------------------------------------------------------------------
```

특정 SQL 에 대해서 Tuning Advisor 를 통해서 SQL 튜닝을 진행하였고, Tuning Advisor 가 제시하는 실행계획까지 확인하였다. 그럼 지금부터 해당 Tuning Advisor 를 SQL Profile 에 등록하여 SQL 의 실행계획을 고정하도록 한다. 단, Tuning Advisor 가 제시한 실행계획이 훨씬 더 성능이 좋다고 가정한다.

5.5. Tuning Advisor가 생성한 실행계획을 SQL Profile에 등록

```
--scott 계정으로 접속
--기존 tuning_task_1 삭제, 만약 미존재지 에러남(무시할것)
BEGIN
   DBMS_SQLTUNE.DROP_SQL_PROFILE (
      name => 'tuning_task_1'
);
END;
/

execute dbms_sqltune.accept_sql_profile(task_name => 'tuning_task_1', name=>'tuning_task_1',
task_owner => 'SCOTT', replace => TRUE);
```

```
--등록된 SQL Profile 확인
select name, sql_text, type, status
from dba_sql_profiles
where name = 'tuning_task_1'
;
--실행계획을 고정한 '6rrwthsp3b263' SQL 을 실행하여 고정된 실행계획 확인
alter session set statistics_level = all;

--바인드 변수 선언
var name varchar2(100);
exec :name := 'popular';

--SQL 실행 해당 SQL 의 힌트는 무시되고 고정된 실행계획으로 수행됨
select /*+ leading(t1) use_nl(t2) index(t1 t1_name) index(t2 t2_id) test sql profile */
   max(t1.name),
   max(t2.amt)
   from t1,
          t2
 where t1.id = t2.id
    and t1.name = :name;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST '));

SQL_ID  6rrwthsp3b263, child number 0
-------------------------------------
select /*+ leading(t1) use_nl(t2) index(t1 t1_name) index(t2 t2_id)
test sql profile */    max(t1.name),    max(t2.amt)    from t1,           t2
where t1.id = t2.id      and t1.name = :name

Plan hash value: 906334482

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



PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
```

```
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buf
fers |  OMem |  1Mem | Used-Mem |

--------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |      1 |        |      1 |00:00:00.28 | 460 |       |       |          |
|   1 |  SORT AGGREGATE     |      |      1 |      1 |      1 |00:00:00.28 | 460 |       |       |          |
|*  2 |   HASH JOIN         |      |      1 |  50005 |    100K|00:00:00.26 | 460 |  6967K|  2145K| 8850K (0)|
|*  3 |    TABLE ACCESS FULL| T1   |      1 |  50005 |    100K|00:00:00.01 | 269 |       |       |          |
|   4 |    TABLE ACCESS FULL| T2   |      1 |    100K|    100K|00:00:00.01 | 191 |       |       |          |
--------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------



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

   2 - access("T1"."ID"="T2"."ID")
   3 - filter("T1"."NAME"=:NAME)

Note
-----

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
   - SQL profile tuning_task_1 used for this statement
   - this is an adaptive plan
```

Tuning Advisor 가 제시한 실행계획대로 SQL 이 실행될 수 있도록 실행계획이 고정된 것을 확인할수 있다.

6. SQL Profile을 이용한 실행계획 제어 및 고정

이번에는 SQL Profile 을 이용해서 특정 SQL 의 실행계획을 변경하는 것을 진행해보도록 한다.

해당 작업의 원활한 진행을 위해 아래와 같이 SQL Profile 권한을 scott 계정 에게 준다.

■ 권한주기

```
--sysdba 로 접속
GRANT SELECT ON DBA_SQL_PROFILES TO SCOTT;
```

또한 작업을 원활하게 처리하기 위해서 plan2.sql, set_sqlprof.sql, drop_sqlprof.sql 스크립트를 생성하도록 한다.

■ plan2.sql

```
REM        ====================================
rem set serveroutput on size unlimited
set serveroutput on

var h_sql_id varchar2(13)
exec :h_sql_id := '&1'

declare
   v_session_user varchar2(20) := null;
   v_parsing_schema_name   varchar2(20) := null;
   v_module varchar2(100) := null;
   v_sql_id varchar(13);
   v_sql_fulltext clob; --11g feature
   --v_sql_fulltext varchar2(32767);

begin
   dbms_output.enable(buffer_size => null) ; --mean buffer size unlimited

   select parsing_schema_name, module, sql_id, sql_fulltext
   into v_parsing_schema_name, v_module, v_sql_id, v_sql_fulltext
   from v$sql
   where sql_id = :h_sql_id
   and     rownum = 1
   ;

   execute immediate 'alter session set current_schema = ' || v_parsing_schema_name ;
   execute immediate 'explain plan set statement_id = ' || ''''||v_sql_id ||''''|| ' for ' || chr(10) ||
v_sql_fulltext ;

   dbms_output.put_line
('================================================================');
```

```
   dbms_output.put_line ('|      Do Explain plan of specified SQL_ID and Display result ');
   dbms_output.put_line
('================================================================');
   dbms_output.put_line ('==> Parsing_user : ' || v_parsing_schema_name || chr(10) ||
                         '==> Module          : ' || v_module || chr(10) ||
                         '==> SQL_ID          : ' || v_sql_id || chr(10) ||
                         '----------------------------------------------------------' || chr(10) ||
                          v_sql_fulltext);

   select sys_context('userenv','session_user')
   into v_session_user
   from dual;

   execute immediate 'alter session set current_schema = ' || v_session_user ;

end;
/

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY(NULL,:h_sql_id,'OUTLINE, ALIAS'));
set serveroutput off
```

■ set_sqlprof.sql

```
REM        ====================================

accept h_sql_id          prompt " ==> sql_id : "
accept h_profile_name prompt " ==> sql_profile_name : "
prompt /*------------------------------------------------------*/
prompt /* ==> now enter the profile hints in following 7 lines */
prompt /*------------------------------------------------------*/
accept h_sqlprof_attr_1 prompt " ==> profile hint (1/7) : "
accept h_sqlprof_attr_2 prompt " ==> profile hint (2/7) : "
accept h_sqlprof_attr_3 prompt " ==> profile hint (3/7) : "
accept h_sqlprof_attr_4 prompt " ==> profile hint (4/7) : "
accept h_sqlprof_attr_5 prompt " ==> profile hint (5/7) : "
accept h_sqlprof_attr_6 prompt " ==> profile hint (6/7) : "
accept h_sqlprof_attr_7 prompt " ==> profile hint (7/7) : "

var h_sql_id varchar2(13)
```

```
var h_profile_name varchar2(30)
var h_sqlprof_attr varchar2(500)

declare
   v_sql_fulltext clob := null;

begin
   :h_sql_id := '&h_sql_id' ;
   :h_profile_name := '&h_profile_name' ;
   :h_sqlprof_attr := '&h_sqlprof_attr_1' ||' '||
                      '&h_sqlprof_attr_2' ||' '||
                      '&h_sqlprof_attr_3' ||' '||
                      '&h_sqlprof_attr_4' ||' '||
                      '&h_sqlprof_attr_5' ||' '||
                      '&h_sqlprof_attr_6' ||' '||
                      '&h_sqlprof_attr_7'
                      ; --max 500 byte 까지 가능

   select sql_fulltext
   into v_sql_fulltext
   from v$sql
   where sql_id = :h_sql_id
   and     rownum = 1
   ;

   dbms_sqltune.import_sql_profile(
      name => :h_profile_name,
      sql_text => v_sql_fulltext,
      profile => sqlprof_attr(:h_sqlprof_attr)

   );
end;
/

select name, sql_text, type, status
from dba_sql_profiles
where name = :h_profile_name
;
```

```
prompt /*-------------------------------*/
prompt /*   sql profile 적용후 plan 확인   */
prompt /*-------------------------------*/
prompt
```

■ **drop_sqlprof.sql**

```
REM        ====================================

accept h_profile_name prompt " ==> sql_profile_name : "

prompt dropping existing profile
exec dbms_sqltune.drop_sql_profile('&h_profile_name', true);

select name, sql_text, type, status, created
from dba_sql_profiles
where name = '&h_profile_name'
;
```

6.1. 특정 SQL 실행

```
--scott 계정으로 접속
alter session set statistics_level = all;
--특정 SQL 실행
var name varchar2(100);
exec :name := 'popular';

select /*+ full(t1) full(t2) use_hash(t1 t2) test2 sql profile */
   max(t1.name),
   max(t2.amt)
   from t1,
          t2
 where t1.id = t2.id
    and t1.name = :name;

select
          sql_id
       , sql_text
 from v$sql
```

```
where sql_text like '%test2 sql profile%'
;
--SQL ID 확인
--97mm7wfa2x0w1

--실행계획확인
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR('97mm7wfa2x0w1',NULL,'ALLSTATS LAST,
OUTLINE, ALIAS'));

SQL_ID  97mm7wfa2x0w1, child number 0
-------------------------------------
select /*+ full(t1) full(t2) use_hash(t1 t2) test2 sql profile */
max(t1.name),    max(t2.amt)    from t1,          t2   where t1.id = t2.id
 and t1.name = :name

Plan hash value: 4274056747

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



PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      1 |00:00:00.10 | 487 |       |       |          |
|   1 |  SORT AGGREGATE    |      |      1 |      1 |      1 |00:00:00.10 | 487 |       |       |          |
|*  2 |   HASH JOIN        |      |      1 |  97745 |    100K|00:00:00.10 | 487 |  6642K|  2659K| 6921K (0)|
|   3 |    TABLE ACCESS FULL| T2   |      1 |    111K|    100K|00:00:00.01 | 191 |       |       |          |
|*  4 |    TABLE ACCESS FULL| T1   |      1 |  97745 |    100K|00:00:00.01 | 269 |       |       |          |
-----------------------------------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------



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

```
   1 - SEL$1
   3 - SEL$1 / T2@SEL$1
   4 - SEL$1 / T1@SEL$1

Outline Data

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

  /*+
      BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      OPTIMIZER_FEATURES_ENABLE('12.1.0.1')
      DB_VERSION('12.2.0.1')
      ALL_ROWS
      OUTLINE_LEAF(@"SEL$1")
      FULL(@"SEL$1" "T1"@"SEL$1")
      FULL(@"SEL$1" "T2"@"SEL$1")

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
      LEADING(@"SEL$1" "T1"@"SEL$1" "T2"@"SEL$1")
      USE_HASH(@"SEL$1" "T2"@"SEL$1")
      SWAP_JOIN_INPUTS(@"SEL$1" "T2"@"SEL$1")
      END_OUTLINE_DATA
  */

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

   2 - access("T1"."ID"="T2"."ID")
   4 - filter("T1"."NAME"=:NAME)

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

Note
-----
```

```
    - dynamic statistics used: dynamic sampling (level=2)
```

SQL_ID "97mm7wfa2x0w1" SQL 은 T1, T2 테이블을 Full Table Scan 하며, Hash Join 으로 실행 되는것을 알 수 있다. 그럼 지금부터 해당 SQL 을 SQL Profile 을 이용하여 실행계획을 변경 및 고정하도록 하겠다.

6.2. SQL Profile 에 등록

set_sqlprof 스크립트를 호출하여 특정 SQL 에 SQL Hint 를 Import 시킨다.

```
@set_sqlprof
==> sql_id : 97mm7wfa2x0w1
 ==> sql_profile_name : 97mm7wfa2x0w1_prof
/*-------------------------------------------------------*/
/* ==> now enter the profile hints in following 7 lines */
/*-------------------------------------------------------*/
 ==> profile hint (1/7) : LEADING(@"SEL$1" "T1"@"SEL$1" "T2"@"SEL$1")
 ==> profile hint (2/7) : USE_NL(@"SEL$1" "T2"@"SEL$1")
 ==> profile hint (3/7) : INDEX_RS_ASC(@"SEL$1" "T1"@"SEL$1" ("T1"."NAME"))
 ==> profile hint (4/7) : INDEX(@"SEL$1" "T2"@"SEL$1" ("T2"."ID"))
 ==> profile hint (5/7) :
 ==> profile hint (6/7) :
 ==> profile hint (7/7) :
구    5:    :h_sql_id := '&h_sql_id' ;
신    5:    :h_sql_id := '97mm7wfa2x0w1' ;
구    6:    :h_profile_name := '&h_profile_name' ;
신    6:    :h_profile_name := '97mm7wfa2x0w1_prof' ;
구    7:    :h_sqlprof_attr := '&h_sqlprof_attr_1' ||' '||
신    7:    :h_sqlprof_attr := 'LEADING(@"SEL$1" "T1"@"SEL$1" "T2"@"SEL$1")' ||' '||
구    8:                                '&h_sqlprof_attr_2' ||' '||
신    8:                                'USE_NL(@"SEL$1" "T2"@"SEL$1")' ||' '||
구    9:                                '&h_sqlprof_attr_3' ||' '||
신    9:                                'INDEX_RS_ASC(@"SEL$1" "T1"@"SEL$1" ("T1"."NAME"))' ||' '||
구   10:                                '&h_sqlprof_attr_4' ||' '||
신   10:                                'INDEX(@"SEL$1" "T2"@"SEL$1" ("T2"."ID"))' ||' '||
구   11:                                '&h_sqlprof_attr_5' ||' '||
신   11:                                '' ||' '||
구   12:                                '&h_sqlprof_attr_6' ||' '||
```

```
신   12:                          '' ||' '||
구   13:                          '&h_sqlprof_attr_7'
신   13:                          ''

PL/SQL 처리가 정상적으로 완료되었습니다.



NAME
--------------------------------------------------------------------------------
SQL_TEXT
--------------------------------------------------------------------------------
TYPE                        STATUS
-------------------- ------------------------
97mm7wfa2x0w1_prof
select /*+ full(t1) full(t2) use_hash(t1 t2) test2 sql profile */
   max(t1.name)
MANUAL                      ENABLED



/*--------------------------------*/
/*   sql profile 적용후 plan 확인   */
/*--------------------------------*/
```

6.3. 변경 및 고정된 실행계획 확인

Import 시킨 힌트가 정상적으로 실행 계획에 반영 및 고정되었는지 확인한다.

```
alter session set statistics_level = all;

var name varchar2(100);
exec :name := 'popular';

select /*+ full(t1) full(t2) use_hash(t1 t2) test2 sql profile */
   max(t1.name),
   max(t2.amt)
   from t1,
        t2
 where t1.id = t2.id
```

```
   and t1.name = :name;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST '));

SQL_ID  97mm7wfa2x0w1, child number 0
-------------------------------------
select /*+ full(t1) full(t2) use_hash(t1 t2) test2 sql profile */
max(t1.name),    max(t2.amt)    from t1,          t2   where t1.id = t2.id
 and t1.name = :name

Plan hash value: 1015865226

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


PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |         |      1 |        |      1 |00:00:00.28 |    3297 |
|   1 |  SORT AGGREGATE                     |         |      1 |      1 |      1 |00:00:00.28 |    3297 |
|   2 |   NESTED LOOPS                      |         |      1 |        |   100K|00:00:00.27 |    3297 |
|   3 |    NESTED LOOPS                     |         |      1 |  97745 |   100K|00:00:00.18 |    2970 |
|   4 |     TABLE ACCESS BY INDEX ROWID BATCHED| T1      |      1 |  97745 |   100K|00:00:00.05 |     514 |
|*  5 |      INDEX RANGE SCAN               | T1_NAME |      1 |  97745 |   100K|00:00:00.02 |     267 |
|*  6 |     INDEX RANGE SCAN                | T2_ID   |    100K|      1 |   100K|00:00:00.09 |    2456 |
|   7 |    TABLE ACCESS BY INDEX ROWID      | T2      |    100K|      1 |   100K|00:00:00.05 |     327 |
-----------------------------------------------------------------------------------------------------------


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

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

   5 - access("T1"."NAME"=:NAME)
   6 - access("T1"."ID"="T2"."ID")
```

```
Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - SQL profile 97mm7wfa2x0w1_prof used for this statement
```

Import 시킨 Hint(Hash Join -> Nested Loops Join)가 적용되어 실행계획이 변경 및 고정된것을 알수있다.

7. SQL Plan Management(SPM)이란?

7.1. 소개

SQL Plan Management(이하 SPM)은 Oracle 11g 에서 추가된 기능이다. 기존의 Plan Stability 를 위한 기능으로는 Stored Outlines 및 SQL Profile 이 있었다. SPM 은 실행계획의 변경 및 고정 뿐만 아니라 실행계획을 안정시키고, 성능을 높일 수 있는 다양한 기능을 제공한다. 아래는 SPM 의 용도 및 효과이다.

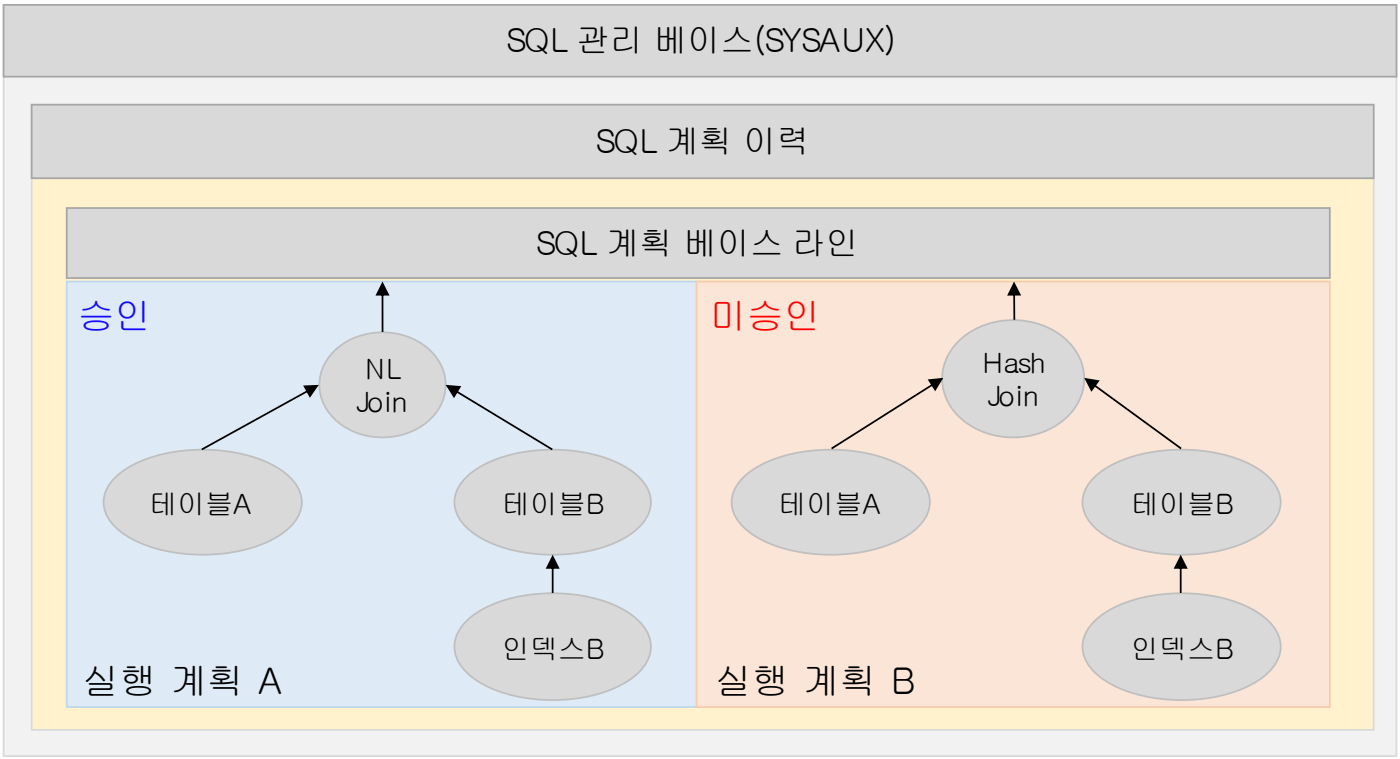
SPM 용도	SPM 결과
성능 안정화(Proactive)	- 실행계획 변화에 따른 갑작스런 성능 저하를 예방 - DBMS 업그레이드 이전의 실행계획을 업그레이드 이후 다시 활용함으로써 옵티마이저 버전 차이에 의한 SQL 성능저하를 방지
성능 개선(Reactive)	- 어플리케이션을 따로 수정하지 않고, 옵티마이저 힌트 효과를 얻을 수 있음(SQL 을 변경하지 않고 실행계획을 변경하고 싶을 때) - 성능이 갑자기 떨어 졌을 때 이전의 실행계획으로 되돌릴 수 있음(SQL 의 실행계획을 이전 것으로 돌리고 싶을 때)

7.2. SPM의 구조

실행 계획의 변화로 SQL 성능 저하가 발생할 경우에 대비해 SPM 은 미리 실행계획(성능이 좋을 것이라 판단한)을 저장해 둔다. 이러한 실행계획들은 SQL Plan Baselines 라고 한다. 실행계획은 SQL 단위로 저장할 수도 있고 여러 실행계획을 한꺼번에 저장할 수도 있다. 또한 SQL Management Baselines 은 SYSAUX 테이블 스페이스에 저장된다. SQL Management Baselines 내에 SQL Plan Baselines 와 SQL Plan History 가 존재한다.

in SQL Management Baselines	설명
SQL Plan Baselines (SQL 계획 베이스라인)	승인되어 사용할 수 있는 실행계획을 뜻함, 하나 또는 여러 개의 실행계획이 존재하며 SQL 실행때는 SQL Plan Baselines 에 저장된 실행 계획만 사용
SQL Plan History (SQL 계획이력)	특정 SQL 과 관련해 저장된 모든 실행계획을 뜻함, SQL Plan History 는 SQL Plan Baselines 이외에 미 승인 실행계획을 포함, 자동 추출된 실행계획은 승인전까지는 쓰이지 않음

SQL Plan Baselines 에 저장한 실행계획으로 SQL 구문을 실행해 성능을 안정화할 수 있다. 또한 실행계획이 갑작스레 바뀌어 성능에 영향을 미치는 것을 미연에 방지한다. 또한 SPM 으로 관리하지 않는 SQL 의 실행계획이 갑자기 바뀌어 성능에 악영향을 줄 경우 이전의 실행계획을 SQL Plan History 에서 찾아서 실행계획을 원래대로 되돌려 성능을 개선할 수 있다. 지금까지 설명한 SPM 의 구조는 아래의 그림과 같다.



7.3. SQL Plan Baselines 확인 방법

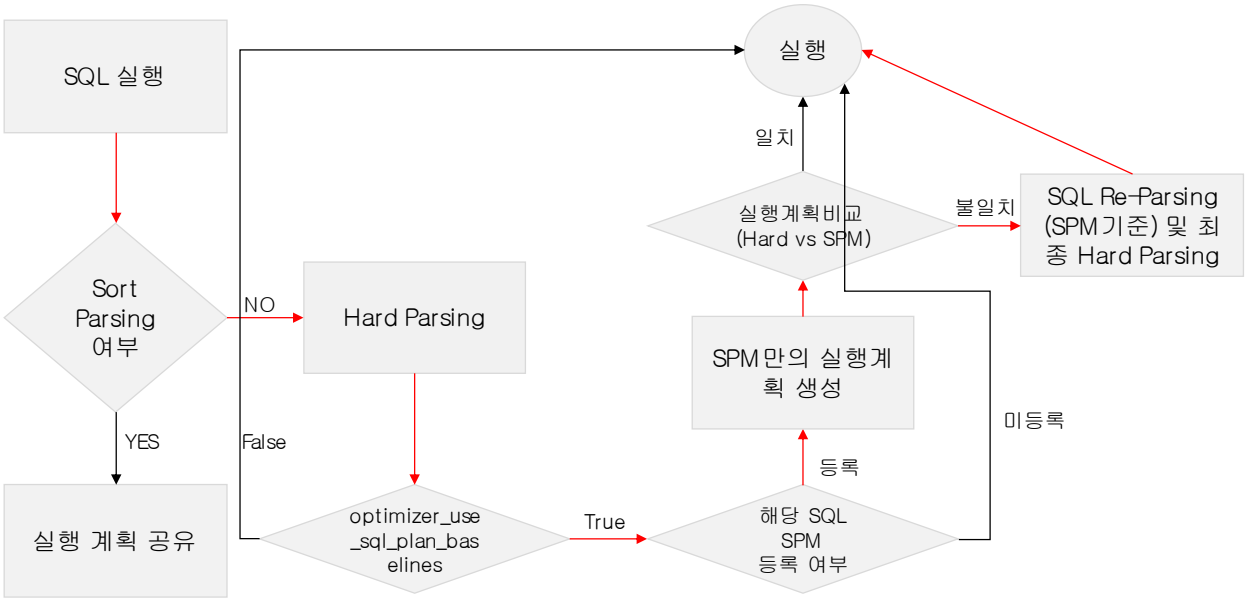
생성된 SQL Plan Baselines 는 DBA_SQL_PLAN_BASELINES 뷰에서 확인이 가능하다. 확인 방법은 아래와 같다.

```
SELECT
        SQL_HANDLE --KEY
    , PLAN_NAME -- 실행계획명명
```

```
    , TO_CHAR(CREATED, 'YYYY/MM/DD HH24:MI:SS') CREATED --생성일시
    , ACCEPTED --승인여부
    , ENABLED --사용가능여부
    , SQL_TEXT --SQL 문
FROM
        DBA_SQL_PLAN_BASELINES
ORDER BY CREATED;
```

7.4. 옵티마이저와 SPM

SQL 이 실행되고, Soft Parsing 이 불가능한 경우 Hard Parsing 은 반드시 수행한다. 이 시점에 옵티마이저는 실행계획을 생성한다. optimizer_use_sql_plan_baselines 파라미터가 TRUE 이고 해당 SQL 이 SPM 에 등록되어 있는 SQL 일 경우 SPM 만의 실행계획을 생성하고, 그 후 Hard Parsing 으로 생성된 실행 계획과 SPM 으로 생성된 실행계획을 비교하여 일치하면 바로 SQL 을 실행하고, 일치하지 않으면 SPM 을 이용해(SPM 을 기준으로) SQL 을 Re-Parsing 한다. 그후 등록된 Plan Baselines 을 적용해서 최종 Hard Parsing 을 수행하고 SQL 을 실행하게 된다. 그후 동일 SQL 이다시 실행되면 Soft Parsing 으로 실행되게 된다. 여기서 잘 알아둬야 할 점은 SPM 으로 만들어지는 실행계획은 Hint 의 집합일 뿐 실행 계획이라고 볼 수 없으며, 해당 Hint 의 집합으로 새로운 실행계획을 만들어 내는 것이다. 아래 그림은 SQL 실행부터 옵티마이저와 SPM 이 어떤 식으로 작동하는지에 대한 순서도이다.



이러한 원리로 인해 SPM 은 SQL 의 과거 실행계획으로 되돌릴 수 있는데 그렇게 하지 못하는 경우도 존재한다. 원인은 아래와 같다.

- 인덱스를 삭제하는 등의 객체 변경이 있을 경우

- 배치 처리 시간 때 트리거를 비활성화 했을 때
- 데이터 이관을 고속화 하려고 키 제약을 비활성화 했을 때

8. SPM에 실행계획 등록

8.1. SPM 등록 유형

SPM 의 등록방법에는 수동 등록 및 자동 등록이 있으며 상세한 내용은 아래와 같다.

자동/수동	등록방법
자동 등록	- DBMS 파라미터를 설정하여 실행되는 모든 SQL 의 실행계획을 SPM 에 저장 및 사용 - optimizer_capture_sql_plan_baselines(저장), optimizer_use_sql_plan_baselines(사용)를 TRUE 로 설정
수동 등록	- AWR 스냅샷을 기반으로 실행계획 로드 하기 - 공유 영역 기반으로 실행계획 로드 하기 - 타 DBMS 에서 SPM 가져오기 - Stored Outlines 의 실행계획을 SPM 으로 가져오기 - SQL 튜닝 어드바이저의 추천 실행계획을 SPM 으로 가져오기

8.2. SPM 등록

우선 SPM 의 원활한 테스트를 위해 아래의 SQL 을 실행하여 실습환경을 구축한다.

```
--scott 계정으로 접속
--테이블 생성
CREATE TABLE SPM_TEST
(
        SPM_TEST_ID VARCHAR2(10)
      , SPM_TEST_NAME VARCHAR2(150)
      , INST_DTM DATE
      , INST_ID VARCHAR2(50)
      , CONSTRAINT SPM_TEST_PK PRIMARY KEY(SPM_TEST_ID)
);

--데이터 입력
INSERT INTO SPM_TEST
SELECT
          LPAD(TO_CHAR(ROWNUM), 10, '0') AS SPM_TEST_ID
       , DBMS_RANDOM.STRING('X', 150)   AS SPM_TEST_NAME
       , SYSDATE AS INST_DTM
```

```
      , DBMS_RANDOM.STRING('X', 50)   AS INST_ID
  FROM
       DUAL CONNECT BY LEVEL <= 100000
;

COMMIT;

--인덱스 생성
CREATE INDEX SPM_TEST_IX01 ON SPM_TEST(SPM_TEST_NAME);

--통계정보수집
EXEC DBMS_STATS.GATHER_TABLE_STATS(USER, 'SPM_TEST', CASCADE=>TRUE,
NO_INVALIDATE=>FALSE);
```

SPM_TEST 테이블을 신규로 생성했으며, SPM_TEST_NAME 컬럼에 대해서 인덱스를 생성하였다. 또한 정확한 실행계획 도출을 위해 통계 정보도 수집하였다. 아래의 파라미터 설정을 하여 해당 세션에서 실행되는 SQL 문의 실행계획을 SPM 에 등록한다.

```
--scott 계정으로 접속
ALTER SESSION SET OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES = TRUE; --SPM 에 등록
ALTER SESSION SET OPTIMIZER_USE_SQL_PLAN_BASELINES = FALSE; --사용은 하지 않음
```

아래의 SQL 을 실행후 실행계획을 확인한다. (해당 SQL 을 2 번째 실행할 때 SPM 에 등록)

```
alter session set statistics_level = all;

--SQL 실행
SELECT /* SPM_TEST3 */ /*+ INDEX(SPM_TEST SPM_TEST_IX01) */
          *
FROM SPM_TEST
WHERE SPM_TEST_NAME LIKE 'AB%';

--실행계획확인
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST NOTE'));

SQL_ID   c4ww0xr8wk22n, child number 0
-------------------------------------
SELECT /* SPM_TEST3 */ /*+ INDEX(SPM_TEST SPM_TEST_IX01) */             *
FROM SPM_TEST   WHERE SPM_TEST_NAME LIKE 'AB%'
```

```
Plan hash value: 3475086095

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |      1 |        |     79 |00:00:00.01 |      84 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| SPM_TEST      |      1 |     10 |     79 |00:00:00.01 |      84 |
|*  2 |   INDEX RANGE SCAN                  | SPM_TEST_IX01 |      1 |     10 |     79 |00:00:00.01 |       5 |
-------------------------------------------------------------------------------------------------------------

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

   2 - access("SPM_TEST_NAME" LIKE 'AB%')
       filter("SPM_TEST_NAME" LIKE 'AB%')
```

아래 SQL 을 실행하여 SPM 등록 내역을 확인한다.

```
SELECT
        SQL_HANDLE --KEY
      , PLAN_NAME -- 실행계획명명
      , TO_CHAR(CREATED, 'YYYY/MM/DD HH24:MI:SS') CREATED --생성일시
      , ACCEPTED --승인여부
      , ENABLED --사용가능여부
      , SQL_TEXT --SQL 문
      , FIXED --사용여부
  FROM
        DBA_SQL_PLAN_BASELINES A
 WHERE SQL_TEXT LIKE '%SPM_TEST3%'
 ORDER BY A.CREATED
;

--결과
SQL_eb79dee31ba9dbb6  SQL_PLAN_fqyfywcdumqxqe84e475a          2018/05/30 13:27:09        YES
          YES       SELECT /* SPM_TEST3 */ /*+ INDEX(SPM_TEST SPM_TEST_IX01) */
           *
 FROM SPM_TEST
WHERE SPM_TEST_NAME LIKE 'AB%'          NO
```

ALTER_SQL_PLAN_BASELINE 를 실행하여 해당 실행내역을 고정한다.

```
---SQPLUS 로 접속
 VAR PBSTS VARCHAR2(30);
EXEC :PBSTS := DBMS_SPM.ALTER_SQL_PLAN_BASELINE('SQL_eb79dee31ba9dbb6',
'SQL_PLAN_fqyfywcdumqxqe84e475a', 'FIXED', 'YES');
```

SPM 등록 내역을 확인하여 실행계획 고정이 완료된 상태를 조회 및 확인한다.

```
SELECT
        SQL_HANDLE --KEY
      , PLAN_NAME -- 실행계획명명
      , TO_CHAR(CREATED, 'YYYY/MM/DD HH24:MI:SS') CREATED --생성일시
      , ACCEPTED --승인여부
      , ENABLED --사용가능여부
      , SQL_TEXT --SQL 문
      , FIXED
  FROM
        DBA_SQL_PLAN_BASELINES A
WHERE SQL_TEXT LIKE '%SPM_TEST3%'
AND FIXED = 'YES'
ORDER BY A.CREATED;

--결과
SQL_eb79dee31ba9dbb6  SQL_PLAN_fqyfywcdumqxqe84e475a          2018/05/30 13:27:09        YES
YES       SELECT /* SPM_TEST3 */ /*+ INDEX(SPM_TEST SPM_TEST_IX01) */
          *
FROM SPM_TEST
WHERE SPM_TEST_NAME LIKE 'AB%'          YES
```

해당 SQL 문의 SPM 등록 및 FIXED 설정이 완료되었다. 그럼 현재 세션에서 해당 SQL 을 실행 시킬때 SPM 을 사용하여 일관된 실행계획으로 실행되는 것을 실습하도록 한다.

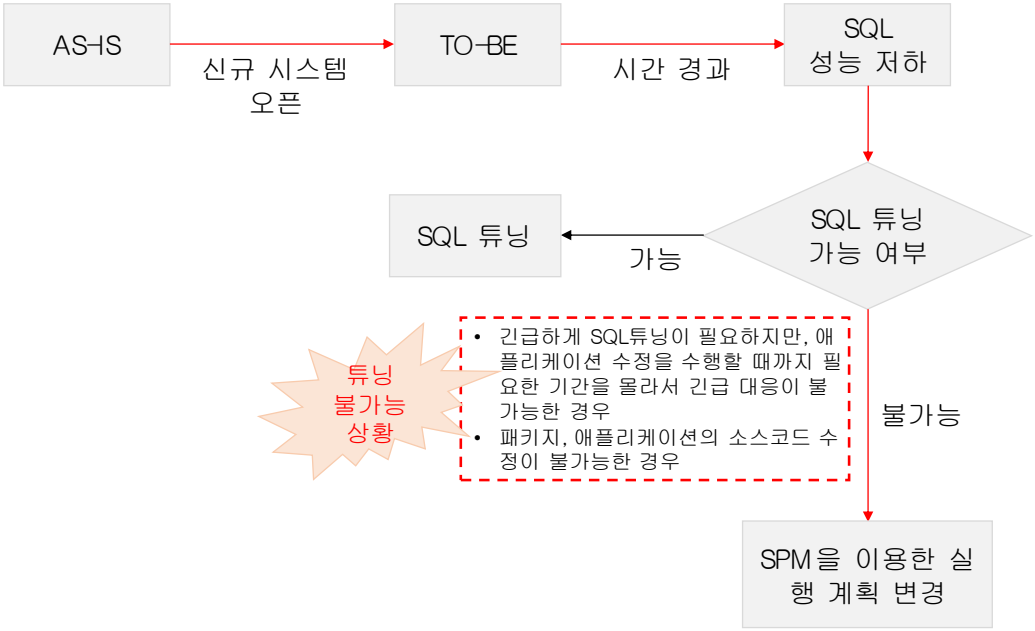
```
--현재 세션에서 SPM 에 등록된 실행계획을 사용하도록 한다.
ALTER SESSION SET OPTIMIZER_USE_SQL_PLAN_BASELINES = TRUE;

--SQL 실행
alter session set statistics_level = all;

 SELECT /* SPM_TEST3 */ /*+ INDEX(SPM_TEST SPM_TEST_IX01) */
          *
```

```
FROM SPM_TEST
WHERE SPM_TEST_NAME LIKE 'AB%';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST NOTE'));

--NOTE 의 내용을 보면 SPM 에 등록된 실행계획대로 SQL 이 실행된것을 확인할수있다.

SQL_ID  c4ww0xr8wk22n, child number 1
-------------------------------------
SELECT /* SPM_TEST3 */ /*+ INDEX(SPM_TEST SPM_TEST_IX01) */             *
FROM SPM_TEST   WHERE SPM_TEST_NAME LIKE 'AB%'

Plan hash value: 3475086095

-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |      1 |        |     79 |00:00:00.01 |      84 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| SPM_TEST      |      1 |     10 |     79 |00:00:00.01 |      84 |
|*  2 |   INDEX RANGE SCAN                  | SPM_TEST_IX01 |      1 |     10 |     79 |00:00:00.01 |       5 |
-----------------------------------------------------------------------------------------------------------------

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

   2 - access("SPM_TEST_NAME" LIKE 'AB%')
       filter("SPM_TEST_NAME" LIKE 'AB%')

Note
-----
   - SQL plan baseline SQL_PLAN_fqyfywcdumqxqe84e475a used for this statement
```

9. SPM을 이용한 실행계획 변경

9.1. 소개

신규 시스템이 오픈 후 시간이 지날수록 초기에 성능이 좋았던 SQL 문의 성능이 저하되는 경우가 있다. 이럴 경우 해결책으로 SQL 튜닝이 있으며 SQL 튜닝 기법중 하나가 SQL Hint 를 이용한 실행계획 변경이다. SQL Hint 를 사용하려면 SQL 문의 변경이 필요하고 SQL 문의 변경은 Application 의 변경을 뜻한다. 하지만 아래의 상황에서 프로그램의 변경이 불가능 혹은 여의치 않을 수 있다.



- 긴급하게 SQL 튜닝이 필요하지만, 애플리케이션 수정을 수행할 때까지 필요한 기간을 몰라서 긴급 대응이 불가능한 경우
- 패키지, 애플리케이션의 소스코드 수정이 불가능한 경우

이러한 경우 SPM 을 이용하여 애플리케이션의 수정없이 실행계획을 수정할 수 있다.

9.2. 작업 절차

SPM 을 이용한 실행계획 변경은 아래의 절차를 따른다.

- 성능이 나쁜 SQL 을 현재 실행계획(SP 라고함)을 SQL Plan Baselines 로 등록함
- SQL 에 힌트를 추가한 성능이 좋은 실행계획(FP 라고함)을 생성함
- SP 를 FP 로 교체함(기존 SP 는 SQL Plan Baselines 에서 삭제)

그럼 지금부터 위 작업순서대로 실행계획을 변경하도록 한다.

9.3. 성능이 나쁜 SP SQL문을 실행

우선 성능이 나쁜 SP SQL 문을 애플리케이션 계정에서 실행한다. (scott 계정을 애플리케이션 계정이라고 가정)

```
--scott 계정으로 로그인
--SQL 실행
--해당 SQL 은 애플리케이션 내에서 실행중인 SQL 임
SELECT /* SPM_HINT_TEST1 */
          E.EMPNO
        , E.ENAME
        , D.DNAME
    FROM
          EMP E
        , DEPT D
WHERE E.DEPTNO = D.DEPTNO;
```

방금 실행한 SP SQL 문의 실행 내역을 확인한다. 아래 내용을 참고하여 SP SQL 의 SQL ID 및 Plan Hash Value 을 확인한다.

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST '));


SQL_ID  g3qs2wsuhjunq, child number 0
-------------------------------------
SELECT /* SPM_HINT_TEST1 */          E.EMPNO        , E.ENAME        ,
D.DNAME      FROM            EMP E          , DEPT D   WHERE E.DEPTNO = D.DEPTNO

Plan hash value: 844388907


-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |      1 |        |     12 |00:00:00.01 |      11 |       |       |          |
|   1 |  MERGE JOIN                  |         |      1 |     12 |     12 |00:00:00.01 |      11 |       |       |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |      4 |      4 |00:00:00.01 |       4 |       |       |          |
|   3 |    INDEX FULL SCAN           | PK_DEPT |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|*  4 |   SORT JOIN                  |         |      4 |     12 |     12 |00:00:00.01 |       7 |  2048 |  2048 | 2048  (0)|
|   5 |    TABLE ACCESS FULL         | EMP     |      1 |     12 |     12 |00:00:00.01 |       7 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------
```

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

   4 - access("E"."DEPTNO"="D"."DEPTNO")
       filter("E"."DEPTNO"="D"."DEPTNO")
```

지금까지 애플리케이션 계정으로 로그인하여 프로그램에서 사용하는 SP SQL 문을 실행하였고 해당 SP SQL 의 SQL ID 및 Plan Hash Value 를 알아내었다.

9.4. SP SQL의 실행계획을 SQL Plan Baselines에 등록

위에서 추출한 SQL_ID 와 PLAN_HASH_VALE 정보를 바탕으로 SP SQL 의 실행계획을 SQL Plan Baselines 에 등록한다.

```
set serveroutput on
declare ret number;
begin
        ret := dbms_spm.load_plans_from_cursor_cache(
                        sql_id => 'g3qs2wsuhjunq',
                        plan_hash_value => '844388907'
                        );
        dbms_output.put_line('LOAD PLANS : ' || ret);
end;
/
```

위 스크립트를 수행 후 DBA_SQL_PLAN_BASELINES 뷰를 조회하여 정상적으로 등록되었는지 확인한다. 또한 해당 SP SQL 의 SQL Plan Baselines 의 SQL_HANDLE 값을 조회한다.

```
SELECT
        SQL_HANDLE --KEY
      , PLAN_NAME -- 실행계획명명
      , TO_CHAR(CREATED, 'YYYY/MM/DD HH24:MI:SS') CREATED --생성일시
      , ACCEPTED --승인여부
      , ENABLED --사용가능여부
      , SQL_TEXT --SQL 문
      , FIXED
  FROM
        DBA_SQL_PLAN_BASELINES A
WHERE SQL_TEXT LIKE '%SPM_HINT_TEST1%'
ORDER BY A.CREATED;
```

위 SQL 의 SQL_HANDLE 값은 "SQL_b3d411d007dbd632" , PLAN_NAME 의 값은 "SQL_PLAN_b7p0ju03xrpjk5ac47e2d" 이 나왔으며 해당 값은 SP SQL 의 실행계획을 FP SQL 의 실행계획으로 바꿔 치기 할 때 사용한다.

9.5. 성능이 좋은 FP SQL문을 SQL Plan Baselines에 등록

지금부터 성능이 좋은 FP SQL 문을 SQL Plan Baselines 에 등록하도록 한다.

```
--scott 계정으로 로그인
--SQL 실행
--해당 SQL 은 애플리케이션 내에서 실행중인 SQL 임
SELECT /* SPM_HINT_TEST2 */ /*+ FULL(E) FULL(D) LEADING(D) USE_HASH(E) */
          E.EMPNO
        , E.ENAME
        , D.DNAME
   FROM
          EMP E
        , DEPT D
WHERE E.DEPTNO = D.DEPTNO;
```

방금 실행한 FP SQL 문의 실행 내역을 확인한다. 아래 내용을 참고하여 FP SQL 의 SQL ID 및 Plan Hash Value 를 확인한다.

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST '));

SQL_ID  0smrjcs7pxmn6, child number 0
-------------------------------------
SELECT /* SPM_HINT_TEST2 */ /*+ FULL(E) FULL(D) LEADING(D) USE_HASH(E)
*/          E.EMPNO        , E.ENAME        , D.DNAME      FROM            EMP E
     , DEPT D   WHERE E.DEPTNO = D.DEPTNO

Plan hash value: 615168685

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     12 |00:00:00.01 |      18 |       |       |          |
|*  1 |  HASH JOIN         |      |      1 |     12 |     12 |00:00:00.01 |      18 |  1695K|  1695K| 1046K (0)|
|   2 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       7 |       |       |          |
|   3 |   TABLE ACCESS FULL| EMP  |      1 |     12 |     12 |00:00:00.01 |       8 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------
```

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

   1 - access("E"."DEPTNO"="D"."DEPTNO")
```

지금까지 성능이 좋은 FP SQL 을 실행하였고 SQL_ID 및 PLAN HASH VALUE 를 알아내었다.

9.6. SP SQL의 실행계획을 FP SQL의 실행계획으로 바꿔 치기

SP SQL 의 실행계획을 FP SQL 의 실행계획으로 바꿔 치기 하도록 한다.

```
set serveroutput on
declare ret number;
begin
        ret := dbms_spm.load_plans_from_cursor_cache(
                        sql_id => '0smrjcs7pxmn6',
                        plan_hash_value => '615168685',
                        sql_handle => 'SQL_b3d411d007dbd632'
                        );
        dbms_output.put_line('LOAD PLANS : ' || ret);
end;
/
```

9.7. 기존 SP SQL의 SQL Plan Baselines 제거

여기까지 완료하였으면 기존의 SP SQL 의 SQL Plan Baselines 를 제거한다.

```
set serveroutput on
declare ret number;
begin
        ret := dbms_spm.drop_sql_plan_baseline(
                        sql_handle => 'SQL_b3d411d007dbd632',
                        plan_name => 'SQL_PLAN_b7p0ju03xrpjk5ac47e2d'
                        );
        dbms_output.put_line('LOAD PLANS : ' || ret);
end;
/
```

해당 작업까지 완료되면 기존의 힌트가 존재하지 않는 SP SQL 문은 힌트를 적용한 FP SQL 문의 실행계획으로 실행되며, 아래와 같이 확인이 가능하다.

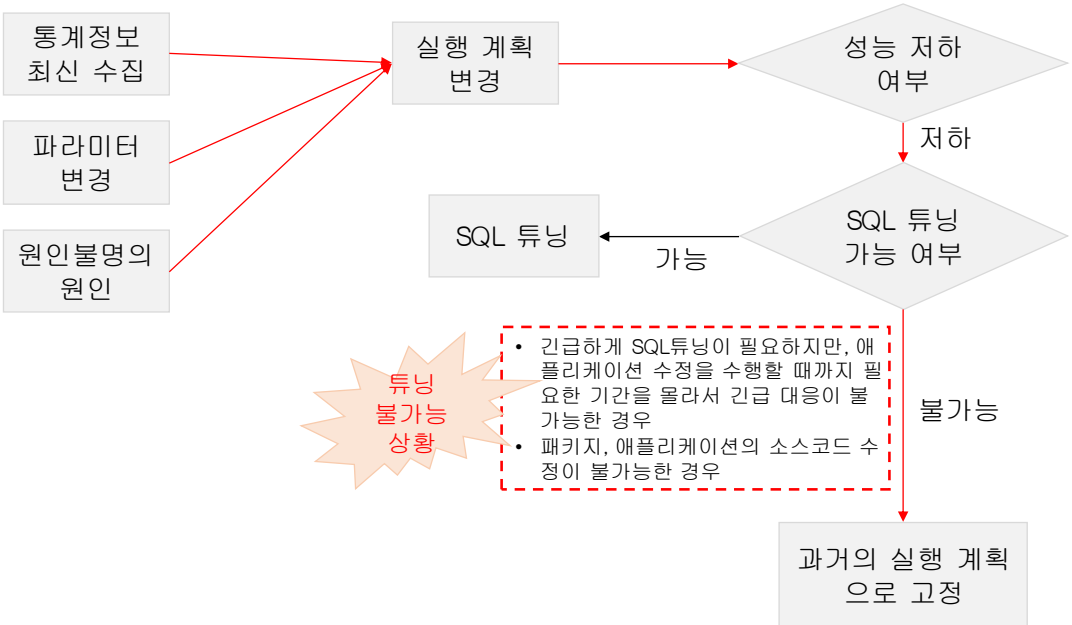
```
ALTER SESSION SET OPTIMIZER_USE_SQL_PLAN_BASELINES = TRUE;
--기존의 SP SQL 실행
SELECT /* SPM_HINT_TEST1 */
          E.EMPNO
        , E.ENAME
        , D.DNAME
   FROM
          EMP E
        , DEPT D
WHERE E.DEPTNO = D.DEPTNO;
--FP SQL 의 실행계획으로 실행되는지 확인
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL, NULL, 'ALLSTATS LAST '));

SQL_ID  g3qs2wsuhjunq, child number 3
-------------------------------------
SELECT /* SPM_HINT_TEST1 */           E.EMPNO         , E.ENAME         ,
D.DNAME      FROM             EMP E          , DEPT D   WHERE E.DEPTNO = D.DEPTNO

Plan hash value: 615168685

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     12 |00:00:00.01 |      18 |       |       |          |
|*  1 |  HASH JOIN         |      |      1 |     12 |     12 |00:00:00.01 |      18 |  1695K|  1695K| 1099K (0)|
|   2 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       7 |       |       |          |
|   3 |   TABLE ACCESS FULL| EMP  |      1 |     12 |     12 |00:00:00.01 |       8 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------

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

   1 - access("E"."DEPTNO"="D"."DEPTNO")

Note
-----
   - SQL plan baseline SQL_PLAN_b7p0ju03xrpjk8447c07a used for this statement
```

SP SQL 이 FP SQL 의 실행계획으로 실행된 것을 확인 할 수 있다. 위와 같은 방법으로 SQL 의 수정(프로그램 수정)이 불가능한 경우 SPM 을 이용해 강제로 실행계획을 변경할 수 있다.

10. SPM을 이용하여 과거의 실행계획으로 SQL실행하기

10.1. 소개

옵티마이저는 통계 정보를 기반으로 비용이 가장 적은 실행계획을 채택한다. 보통 통계정보를 수집한 시점의 데이터 양을 기반으로 최적의 실행계획을 선택하므로 통계정보를 항상 최신으로 유지하며 운영하는게 좋지만, 가끔 최신 통계정보를 수집하고 나서 나쁜 실행계획을 선택하는 경우가 있다. 이런 경우 갑작스런 성능 저하 현상이 발생된다. 이런 상황에서 가장 좋은 튜닝 방법은 과거에 사용했던 실행계획을 사용하도록 실행계획을 고정시키는 것이다. 단 조건이 존재하는데 AWR 및 SQL Tuning Set 에 대한 라이센스가 있어야 하고, 과거에 사용했던 실행계획 자체가 AWR 에 존재해야 한다.



10.2. 작업 절차

SPM 을 이용하여 과거의 실행계획으로 SQL 을 실행하기 위해서는 아래의 절차를 따른다.

- 성능이 좋았던 과거의 실행계획이 AWR 에 존재하는지 확인한다.
- 과거의 실행계획을 AWR 에서 찾아서 SQL Tuning Set(이하 STS)에 등록한다.
- STS 에 SQL Plan Base Lines 를 작성한다.

10.3. 실습환경 구축

원활한 실습을 진행하기 위해서 동일한 특정 SQL 문에서 힌트절을 추가하여 SQL 의 성능이 좋게 만든다. 해당 SQL 이 BEFORE SQL 로써 기존에 성능이 좋았던 실행계획의 SQL 이다. BEFORE SQL 의 힌트 절은 없다고 가정하며, 해당 SQL(힌트절이 없는)의 성능이 저하되어 과거의 실행계획으로 고정 시켜야 하는 상황이 발생한다고 가정한다. 아래는 BEFORE SQL 이다.

```
--SCOTT 계정으로 접속
--BEFORE SQL
--해당 SQL 은 HASH 조인으로 풀리면서 성능이 좋음
SELECT /* BEFORE_PLAN */ /*+ FULL(D) FULL(E) USE_HASH(D E) */
          E.EMPNO
        , E.ENAME
        , D.DNAME
   FROM EMP E
        , DEPT D
  WHERE E.DEPTNO = D.DEPTNO ;
--원할한 테스트를 위해 AWR 스냅샷을 바로 생성한다.
execute dbms_workload_repository.create_snapshot;
```

10.4. 과거 성능이 좋았던 SQL문의 SQL_ID를 추출

그럼 지금부터 과거 성능이 좋았던 BEFORE SQL 문의 SQL_ID 를 추출한다. 추출 SQL 문은 아래와 같다.

```
SELECT SQL_ID, SQL_TEXT
FROM DBA_HIST_SQLTEXT
WHERE SQL_TEXT LIKE '%BEFORE_PLAN%';
--결과
1yg9gfupwxvq2
```

10.5. 추출한 SQL_ID를 기준으로 PLAN_HASH_VALUE 및 SNAP_ID 추출

과거 성능이 좋았던 SQL_ID 를 기준으로 PLAN_HASH_VALUE 및 SNAP_ID 를 추출한다. 추출 SQL 문은 아래와 같다.

```
SELECT A.SNAP_ID
       , TO_CHAR(B.BEGIN_INTERVAL_TIME, 'YYYY-MM-DD HH24:MI:SS') AS BEGIN_INTERVAL_TIME
       , A.SQL_ID
       , A.PLAN_HASH_VALUE
  FROM DBA_HIST_SQLSTAT A
```

```
      , DBA_HIST_SNAPSHOT B
 WHERE A.DBID = B.DBID
   AND A.SNAP_ID = B.SNAP_ID
   AND A.SQL_ID = '1yg9gfupwxvq2'
 ORDER BY A.SNAP_ID;
--결과
1335     2018-05-29 11:31:42        1yg9gfupwxvq2   615168685

--해당 SQL_ID 기준 실행계획 확인
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_AWR('1yg9gfupwxvq2'));
 --결과
 SQL_ID 1yg9gfupwxvq2
 --------------------
 SELECT /* BEFORE_PLAN */ /*+ FULL(D) FULL(E) USE_HASH(D E) */
 E.EMPNO        , E.ENAME        , D.DNAME     FROM EMP E          , DEPT D
 WHERE E.DEPTNO = D.DEPTNO

 Plan hash value: 615168685


 ------------------------------------------------------------------------------
 | Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
 ------------------------------------------------------------------------------
 |   0 | SELECT STATEMENT   |      |       |       |     6 (100)|          |
 |   1 |  HASH JOIN         |      |    12 |   312 |     6   (0)| 00:00:01 |
 |   2 |   TABLE ACCESS FULL| DEPT |     4 |    52 |     3   (0)| 00:00:01 |
 |   3 |   TABLE ACCESS FULL| EMP  |    12 |   156 |     3   (0)| 00:00:01 |
 ------------------------------------------------------------------------------
```

지금까지 과거의 성능이 좋았던 SQL 문의 SQL_ID, PLAN_HASH_VALUE 를 추출하였다. 그럼 지금부터 해당 SQL_ID, PLAN_HASH_VAUE 를 이용해서 해당 SQL 문의 실행계획을 고정 시키는 작업을 하도록 한다.

10.6. 과거의 실행계획을 SQL Tuning Set에 등록

지금부터 STS 를 신규로 생성하고 해당 STS 에 BEFORE SQL 의 실행계획을 저장하도록 한다.

```
--SCOTT 계정으로 접속
--SQLPLUS 로 접속
EXEC DBMS_SQLTUNE.CREATE_SQLSET('BEFORE_SQL');
```

```
DECLARE
        CUR DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
        OPEN CUR FOR
                SELECT
                        VALUE(P)
                 FROM
                        TABLE(

        DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(

                                        BEGIN_SNAP => 1334,

                                        END_SNAP => 1335,

                                        BASIC_FILTER => '       SQL_ID = ''1yg9gfupwxvq2''



         AND PLAN_HASH_VALUE = 615168685')) P;
        DBMS_SQLTUNE.LOAD_SQLSET('BEFORE_SQL', CUR);
        END;
/
```

지금까지 STS 를 생성하고 해당 STS 에 BEFORE SQL 의 실행계획을 등록하였다. 아래의 SQL 로 방금 등록한 STS 를 조회 및 확인한다.

```
--DBA_SQLSET 조회
SELECT
       NAME
     , OWNER
     , CREATED
     , STATEMENT_COUNT
FROM DBA_SQLSET
WHERE NAME = 'BEFORE_SQL'
```

```
;
--결과
BEFORE_SQL       SCOTT   2018/05/29 11:42:46        1

--DBA_SQLSET_STATEMENTS 조회
 SELECT
            SQL_ID
          , PLAN_HASH_VALUE
          , PARSING_SCHEMA_NAME
          , SUBSTR(SQL_TEXT, 1, 100) SQL_TEXT
    FROM
         DBA_SQLSET_STATEMENTS
  WHERE SQLSET_NAME = 'BEFORE_SQL'
 ORDER BY SQL_ID;

--결과
1yg9gfupwxvq2  615168685        SCOTT   SELECT /* BEFORE_PLAN */ /*+ FULL(D) FULL(E) USE_HASH(D
E) */
             E.EMPNO
           , E.ENAME
       ,
```

10.7. 등록된 SQL Tuning Set의 실행계획 확인

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_SQLSET('BEFORE_SQL', '1yg9gfupwxvq2'));

 SQL Tuning Set Name: BEFORE_SQL
 SQL Tuning Set Owner: SCOTT
 SQL_ID: 1yg9gfupwxvq2
 SQL Text: SELECT /* BEFORE_PLAN */ /*+ FULL(D) FULL(E) USE_HASH(D E) */
                E.EMPNO         , E.ENAME         , D.DNAME     FROM EMP E          , DEPT D
                WHERE E.DEPTNO = D.DEPTNO
--------------------------------------------------------------------------------

 Plan hash value: 615168685


| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |     6 (100)|          |
|   1 |  HASH JOIN         |      |    12 |   312 |     6   (0)| 00:00:01 |
```

```
|   2 |   TABLE ACCESS FULL| DEPT |     4 |    52 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| EMP  |    12 |   156 |     3   (0)| 00:00:01 |
```

해당 STS 의 저장된 실행계획을 보면 BEFORE SQL 의 실행계획이 저장되어있는것을 확인할수 있다.

10.8. SQL Tuning Set을 SQL Plan Baselines 에 등록

```
SET SERVEROUTPUT ON
DECLARE
        RET NUMBER;
BEGIN
        RET := DBMS_SPM.LOAD_PLANS_FROM_SQLSET(SQLSET_NAME => 'BEFORE_SQL',

                                               SQLSET_OWNER => 'SCOTT',

                                               ENABLED => 'YES');
        DBMS_OUTPUT.PUT_LINE('LOAD PLANS : '|| RET);
END;
/
```

BEFORE SQL 의 STS 를 SQL Plan Baselines 에 등록을 완료하였다. 아래의 SQL 로 등록된 내용을 확인하도록 한다.

```
SELECT
        SQL_HANDLE
      , PLAN_NAME
      , TO_CHAR(CREATED, 'YYYY-MM-DD HH24:MI:SS') CREATED
  FROM
          DBA_SQL_PLAN_BASELINES
 WHERE SQL_TEXT LIKE '%BEFORE_PLAN%';
--결과
SQL_e147c0984953779e        SQL_PLAN_f2jy0m14p6xwy8447c07a    2018-05-29 13:51:13
```

또한 SQL Plan Baselines 에 등록한 실행계획을 출력 및 확인한다.

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE('SQL_e147c0984953779e'));


--결과
 ---------------------------------------------------------------------------
 SQL handle: SQL_e147c0984953779e
 SQL text: SELECT /* BEFORE_PLAN */ /*+ FULL(D) FULL(E) USE_HASH(D E) */
              E.EMPNO        , E.ENAME        , D.DNAME     FROM EMP E          , DEPT D
```

```
            WHERE E.DEPTNO = D.DEPTNO
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
Plan name: SQL_PLAN_f2jy0m14p6xwy8447c07a         Plan id: 2219294842
Enabled: YES     Fixed: NO     Accepted: YES     Origin: MANUAL-LOAD-FROM-STS
Plan rows: From dictionary
--------------------------------------------------------------------------------

Plan hash value: 615168685

-----------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |     6 (100)|          |
|   1 |  HASH JOIN         |      |    12 |   312 |     6   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| DEPT |     4 |    52 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| EMP  |    12 |   156 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

마지막으로 SQL Plan Baselines 에 저장된 실행계획으로 해당 SQL 이 실행되도록 FIXED 작업을 한다.

```
VAR PBSTS VARCHAR2(30);
EXEC :PBSTS := DBMS_SPM.ALTER_SQL_PLAN_BASELINE('SQL_e147c0984953779e',
'SQL_PLAN_f2jy0m14p6xwy8447c07a', 'FIXED', 'YES');
```

10.9. SQL Tuning Set 제거

지금까지 STS 를 생성 후 BEFORE SQL 실행계획을 저장하였다. 그후 저장한 STS 를 사용하여 SQL Plan Baselines 에 등록하여 BEFORE SQL 의 실행계획을 고정하였다. 마지막으로 해당 작업에서 사용된 STS 를 제거하여 작업을 마무리 한다.

```
EXEC DBMS_SQLTUNE.DROP_SQLSET('BEFORE_SQL');
```

지금까지 실행 계획의 변화로 인해 SQL 의 성능이 저하된 경우 AWR 에 존재하는 과거의 실행계획으로 SQL 실행계획을 강제로 고정하여 성능을 개선시키는 방법에 대해 알아보았다.

11. SPM을 다른 DB로 마이그레이션

11.1. 소개

SQL Plan Baselines 는 다른 DB 로 마이그레이션 할 수 있다. 예를 들어 실제 환경에서 성능 문제가 발생했다고 가정하면, 성능이 나쁜 SQL 문을 찾고 검증 환경에서 SQL 튜닝을 수행하고 검증 환경에 SQL Plan Baselines 를 생성한다. 이렇게 해서 생성한 SQL Plan Baselines 를 실제환경으로 마이그레이션 하면 실제 환경의 문제를 해결가능하다. 또한 성능 테스트 시 성능개선을 위해 SQL Plan Baselines 를 등록한 경우 최종적으로 실제 환경으로 SQL Plan Baselines 를 마이그레이션 함으로써 실제환경에서도 개선 후의 실행계획을 사용가능하다.

11.2. 작업 절차

SPM 을 다른 AS-IS DB 에서 TO-BE DB 로 마이그레이션 하기위해서는 아래와 같은 절차에 따라야 한다.

- ASIS DB 의 SQL Plan Baselines 를 Staging Table 에 Pack 함
- ASIS DB 의 Staging Table 을 Dump File 로 출력, 생성된 파일은 ASIS DUMP 라고 함
- TOBE DB 에서 ASIS DUMP 를 Staging Table 에 Data Import 함
- TOBE DB 에서 Staging Table 을 이용해서 SQL Plan baselines 를 Unpack 함

11.3. ASIS DB의 SQL Plan Baselines를 Staging Table에 Pack

```
BEGIN
    DBMS_SPM.CREATE_STGTAB_BASELINE(
        TABLE_NAME => 'SPM_SCOTT_ASIS_STGTAB',
        TABLE_OWNER => 'SCOTT'
        );
END;
/
;
SET SERVEROUTPUT ON
DECLARE
    RET NUMBER;
BEGIN
    RET := DBMS_SPM.PACK_STGTAB_BASELINE(
                TABLE_NAME => 'SPM_SCOTT_ASIS_STGTAB',
                TABLE_OWNER => 'SCOTT',
                ENABLED => 'YES');
```

```
    DBMS_OUTPUT.PUT_LINE('PACKED : ' || RET);
  END;
 /
--결과확인
SELECT COUNT(*)
FROM SPM_SCOTT_ASIS_STGTAB;
```

위 작업을 완료하면 ASIS DB 의 SCOTT 계정에 SPM_SCOTT_ASIS_STGTAB 테이블이 생성되며 해당 테이블에는 SQL Plan Baselines 의 데이터가 저장된다.

11.4. ASIS DB의 Staging Table을 Dump File로 출력

아래 SQL 을 실행하여 Dump File 이 저장될 위치를 확인한다.

```
SELECT
       DIRECTORY_NAME
     , DIRECTORY_PATH
FROM DBA_DIRECTORIES
WHERE DIRECTORY_NAME = 'DATA_PUMP_DIR';
--결과
DATA_PUMP_DIR /home/oracle/db/admin/orcl/dpdump/
```

아래와 같은 명령어로 Staging Tabld 의 내용을 Dump File 로 출력한다.

```
expdp system tables=SCOTT.SPM_SCOTT_ASIS_STGTAB directory=DATA_PUMP_DIR
dumpfile=SPM_SCOTT_ASIS_STGTAB.dmp
```

DATA_PUMP_DIR 디렉토리에 SPM_SCOTT_ASIS_STGTAB.dmp 파일이 정상적으로 생성되었는지 확인한다.

11.5. Export한 Dump File을 TOBE DB의 Staging Table로 Import

해당 작업을 하기 위해서 우선 Export 시킨 Dump File 을 TOBE DB 서버의 DATA_PUMP_DIR 위치에 이동(저장)시킨다. 아래의 명령어로 위에서 Export 시킨 Dump File 을 TOBE DB 의 Staing Table 로 Import 시킨다.

```
impdp system tables=SCOTT.SPM_SCOTT_ASIS_STGTAB directory=DATA_PUMP_DIR
dumpfile=SPM_SCOTT_ASIS_STGTAB.dmp
```

11.6. TOBE DB의 Staging Table을 SQL Plan Baselines로 Unpack

TOBE DB 에서 아래의 스크립트를 실행하여 해당 작업을 수행한다.

```
set serverouput on
declare
        ret number;
```

```
begin
        ret := dbms_spm.unpack_stgtab_baseline(
                table_name => 'SPM_SCOTT_ASIS_STGTAB',
                table_owner => 'SCOTT' );
        dbms_output.put_line('Unpacked : ' || ret );
end;
/
```

이렇게 하면 모든 SQL Plan Baselines 가 TOBE DB 로 마이그레이션이 완료된 상태이다.

12. 참고문헌

https://oracle-base.com/articles/misc/outlines

https://docs.oracle.com/database/121/TGSQL/tgsql_profiles.htm#TGSQL599

http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-sql-plan-mgmt-12c-1963237.pdf

https://oracle-base.com/articles/11g/sql-plan-management-11gr1

ORACLE 레벨업 - 스즈키 겐고 외 5 인 지음 (한빛미디어)

< 끝 >