SQL Profile을 이용한 SQL Plan 변경

㈜엑셈 컨설팅본부/DB컨설팅팀 오 경렬

본 문서는 DBMS_SQLTUNE Package 를 이용하여 SQL Plan 을 변경하는 PLAN_SET_OUTLINE (가칭) User Define Procedure 에 관한 글로 다음과 같은 내용을 포함 한다.

- SQL Plan 변경의 필요성
- SQL Plan 변경을 위한 기능은 9i 부터 제공
- 뭐가 다른가?
- SQL Profile 이란
- OUTLINE 이란
- PLAN_SET_OUTLINE Source
- 사용 예
- 주의사항 & 고급 사용

1. SQL Plan 변경의 필요성

Oracle 을 사용하는 사이트라면 SQL 성능이슈를 한번쯤을 겪어봤을 것이다. 보통 비효율 SQL 을 수정하고 Application 에 반영하여 정상적인 서비스를 운영하기 까지는 많은 시간이 소요될 수 있다. 해당 SQL 이 수행되는 업무와 관련된 업무들이 영향을 받기 때문에 서비스를 일시 정지 해야 하는 경우도 있고 비효율 SQL 이 어느 소스에서 수행되는지 파악하는 시간으로 인해 쉽게 해결될 일이 장애 상황까지 발전할 수도 있다. 가장 흔하게는 통계정보 생성에 의한 경우부터 소스 배포 과정에서 검증되지 않은 SQL 이 배포되는 경우까지 비효율 SQL 이 발생할 수 있는 케이스는 다양하다.

2. SQL Plan 변경을 위한 기능은 9i부터 제공

Oracle 은 이미 9i 버전부터 Stored Outline 을 이용해서 SQL 플랜을 변경하는 방법을 제공했다. 아이러니하게도 Oracle 9i 에서는 Stored Outline 을 이용한 플랜 변경은 실 운영환경에서 (적어도 필자의 경험에서는) 정상 동작하지 않는다. 9i 에서 등장한 Stored Outline 을 이용한 SQL 플랜 변경은 10g 버전부터 정상적으로 동작한다. Oracle 10g 에는 SQL Profile 이라는 것을 새롭게 등장한다. 간단하게 개념을 잡는다면 SQL Profile 은 Stored Outline 을 포함하는 더 큰 틀이라 할 수 있다.

3. 뭐가 다른가?

이미 온라인상에는 SQL Plan 변경을 위한 이론과 방법이 정리된 수많은 기술문서가 존재한다. 하지만 안타깝게도 실 환경에서 사용하기에는 뭔가 복잡하고 손이 많이 간다. SQL Plan 변경이 필요한 경우에 빠른 해결이 중요한 포인트이다. 10g 에서 SQL Profile 을 이용해서 플랜을 변경하려면 Outline 을 하나하나 적어줘야 하는데 여간 귀찮은 작업이 아니다.

필자의 Procedure 는 사용이 매우 간단하다.

EXEC PLAN_SET_OUTLINE('개선 전 SQLID', '개선 후 SQLID');

플랜이 변경되었습니다.

4. SQL Profile 이란

SQL Profile 은 특정 SQL 문에 연결할 수 있는 SQL 컴파일러 통계의 모음을 말한다. SQL Profile 이 현재 SQL 문에 생성되면, SQL 분석 (컴파일)을 수행할 때 프로파일의 통계는 컴파일러에 의해 사용된다. SQL 문장의 정규화된 텍스트가 SQL 프로파일 작성시에 제공된 SQL 텍스트와 일치하는 경우프로파일이 SQL 컴파일 사용되는 것이다. SQL 텍스트의 정규화란, 모든 비 Literal 텍스트를 대문자로 변경하고 빈 공간이 없애는 것을 말한다. 컴파일을 수행한 세션에서는 SQL 프로파일이 생성된 카테고리 및 Parameter 에 대해서 동일한 값을 가지고 있어야 한다.

카테고리는 여러 프로파일이 동일한 SQL 문에 존재하는 것을 허용한다. 한 세션에서 특정 카테고리 네임스페이스의 프로파일을 만들어 비공개 프로파일을 테스트 할 수 있습니다.

SQL 프로파일은 다음과 같은 SQL 문장에 대해서 사용 가능하다.

- SELECT statements
- UPDATE statements
- INSERT (but only with a SELECT clause) statements
- DELETE statements
- CREATE TABLE (but only with the AS SELECT clause)
- MERGE statements (the upsert operation)

Data Dictionary 에서 내부적으로 수행된 SQL 문(재귀 SQL)에 대해서는 프로파일을 사용할 수 없다. 또한, 데이터베이스가 오픈 되기 전에 수행되는 모든 SQL 역시 프로파일을 검색, 사용 할 수 없다.

5. OUTLINE 이란

백문이 불 여 일견이다. 생성된 Outline 을 먼저 살펴보자.

```
/*+
    BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      OPTIMIZER_FEATURES_ENABLE('11.2.0.3')
      DB_VERSION('11.2.0.3')
      OPT_PARAM('_optim_peek_user_binds' 'false')
     PUSH_PRED(@"SEL$1" "PLAN"@"SEL$1" 11 10 9)
     OUTLINE(@"SEL$1")
     NO_ACCESS(@"SEL$1" "PLAN"@"SEL$1")
      USE_NL(@"SEL$1" "C"@"SEL$1")
     PUSH_SUBQ(@"SEL$4")
      NLJ_BATCHING(@"SEL$4" "PL"@"SEL$4")
    END_OUTLINE_DATA
```

```
*/
```

Outline 은 Oracle Optimizer 가 최적의 플랜을 만들기 위해서 수행한 작업의 결과라고 할 수 있다. 실제로 OUTLINE 정보는 10053 Trace 에서 가장 마지막에 확인 할 수 있다. OUTLINE 은 DBMS_XPLAN Package 를 통해서도 확인 가능하다.

6. PLAN_SET_OUTLINE Source

```
-- GRANT ADMINISTER SQL MANAGEMENT OBJECT to USER; 권한 필요
create or replace procedure PLAN_SET_OUTLINE(v_sql_id_asis in varchar2, v_sql_id_tobe
in varchar2)
is
  l_sql_fulltext  clob;


  l_hint_table    sys.dbms_debug_vc2coll;
  l_profile_hints sys.sqlprof_attr := sys.sqlprof_attr();
  i               pls_integer;

begin
    select sql_fulltext
           INTO l_sql_fulltext


    from v$sqlarea
    where sql_id = v_sql_id_asis;

    with a as (
      select
               rownum as r_no
             , a.*
      from
               table(
                 dbms_xplan.display_cursor(
                   v_sql_id_tobe
                 ,  null
                 , 'OUTLINE'
```

```
                )
              ) a
      )
    select substr(plan_table_output, 7) outln
      bulk collect into l_hint_table
    from a
    where r_no between ( select min(r_no) from a where instr(a.PLAN_TABLE_OUTPUT,
'BEGIN_OUTLINE_DATA') = 7)
               and ( select min(r_no) from a where instr(a.PLAN_TABLE_OUTPUT,
'END_OUTLINE_DATA') = 7)
    order by r_no;

    i := l_hint_table.first;
    while i is not null
    loop
      if l_hint_table.exists(i + 1) then
        if substr(l_hint_table(i + 1), 1, 1) = ' ' then
          l_hint_table(i) := l_hint_table(i) || trim(l_hint_table(i + 1));
          l_hint_table.delete(i + 1);
        end if;
      end if;
      i := l_hint_table.next(i);
    end loop;

    i := l_hint_table.first;
    while i is not null
    loop
      l_profile_hints.extend;
      l_profile_hints(l_profile_hints.count) := l_hint_table(i);
      i := l_hint_table.next(i);
    end loop;

  dbms_sqltune.import_sql_profile(
    name => 'PROFILE_'||v_sql_id_asis
   ,replace => TRUE
   ,sql_text => l_sql_fulltext
   ,profile => l_profile_hints
  );
end PLAN_SET_OUTLINE;
/
```

7. 사용 예

[1] TEST 용 데이터 생성

```
create table exem_t1
as
select level c1, level c2, LPAD(LEVEl, 100, '_') C3 from dual
connect by level <= 1000;
create index ix_exem_t1_01 on exem_t1(c1);
create index ix_exem_t1_02 on exem_t1(c2);
```

<Trace 수행을 위한 Parameter 적용>

```
alter session set statistics_level = 'ALL';
```

[2] 비효율 SQL 의 SQL ID 확인.

```
select * from exem_t1 t1
where c1 in (select c1 from exem_t1 where c2 <= 10);

select prev_sql_id, (select substr(sql_text,1,64) text from v$sqlarea where sql_id =
prev_sql_id) text  from v$session where sid = userenv('sid');


PREV_SQL_ID       TEXT
--------------   --------------------------------------------------------------------------
0h60tr4xc2mub     select * from exem_t1 t1 where c1 in (select c1 from exem_t1 whe

select * from table(dbms_xplan.display_cursor('0h60tr4xc2mub', null, 'typical allstats
last'));
-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name               | Starts   | A-Rows |   A-Time  |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |        1 |     10 |00:00:00.01 |
|*  1 |  HASH JOIN RIGHT SEMI        |                    |        1 |     10 |00:00:00.01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EXEM_T1            |        1 |     10 |00:00:00.01 |
|*  3 |    INDEX RANGE SCAN          | IX_EXEM_T1_02      |        1 |     10 |00:00:00.01 |
|   4 |   TABLE ACCESS FULL          | EXEM_T1            |        1 |   1000 |00:00:00.01 |
-----------------------------------------------------------------------------------------
```

[3] 개선된 SQL 의 SQL ID 확인.

```
select /*+ LEADING(exem_t1@SEL$2) USE_NL(t1) */ * from exem_t1 t1
where c1 in (select /*+ UNNEST */ c1 from exem_t1 where c2 <= 10);

select prev_sql_id, (select substr(sql_text,1,64) text from v$sqlarea where sql_id =
prev_sql_id) text  from v$session where sid = userenv('sid');
PREV_SQL_ID       TEXT
--------------   ----------------------------------------------------------------------
0bjfkr111hxpa    select /*+ LEADING(exem_t1@SEL$2) USE_NL(t1) */ * from exem_t1 A
where c1

select * from table(dbms_xplan.display_cursor('0bjfkr111hxpa', 0, 'typical allstats
last'));


-------------------------------------------------------------------------------------
| Id  | Operation             | Name             | Starts | A-Rows |      A-Time    |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                  |      1 |     10 |   00:00:00.01 |
|   1 |  NESTED LOOPS         |                  |      1 |     10 |   00:00:00.01 |
|   2 |   NESTED LOOPS        |                  |      1 |     10 |   00:00:00.01 |
|   3 |    SORT UNIQUE        |                  |      1 |     10 |   00:00:00.01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| EXEM_T1 |      1 |     10 |   00:00:00.01 |
|*  5 |      INDEX RANGE SCAN    | IX_EXEM_T1_02 |      1 |     10 |   00:00:00.01 |
|*  6 |    INDEX RANGE SCAN      | IX_EXEM_T1_01 |     10 |     10 |   00:00:00.01 |
|   7 |   TABLE ACCESS BY INDEX ROWID | EXEM_T1  |     10 |     10 |   00:00:00.01 |
-------------------------------------------------------------------------------------
```

[4] SQL Profile 교체

```
--   PLAN_SET_OUTLINE('ASIS_SQL_ID', 'TOBE_SQL_ID')
exec PLAN_SET_OUTLINE('0h60tr4xc2mub', '0bjfkr111hxpa');
```

[5] Outline 적용 확인

<DBMS_XPLAN note 항목을 통한 확인>

```
 ---------------------------------------------------------------------------------------
| Id  | Operation               | Name              | Starts | A-Rows |   A-Time    |
---------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |                   |      1 |     10 |00:00:00.01 |
```

```
|   1 |  NESTED LOOPS            |                    |      1 |     10 |00:00:00.01 |
|   2 |   NESTED LOOPS           |                    |      1 |     10 |00:00:00.01 |
|   3 |    SORT UNIQUE           |                    |      1 |     10 |00:00:00.01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| EXEM_T1      |      1 |     10 |00:00:00.01 |
|*  5 |      INDEX RANGE SCAN         | IX_EXEM_T1_02 |      1 |     10 |00:00:00.01 |
|*  6 |    INDEX RANGE SCAN           | IX_EXEM_T1_01 |     10 |     10 |00:00:00.01 |
|   7 |   TABLE ACCESS BY INDEX ROWID | EXEM_T1       |     10 |     10 |00:00:00.01 |
-------------------------------------------------------------------------------------
Note
-----
   - SQL profile PROFILE_0h60tr4xc2mub used for this statement

<V$SQLAREA.sql_profile 을 이용한 확인>
select sql_id, sql_profile
from v$sqlarea
where sql_id = '0h60tr4xc2mub';

SQL_ID              SQL_PROFILE
-------------     ---------------------
0h60tr4xc2mub    PROFILE_0h60tr4xc2mub      <<-- 프로파일 적용된 이후에 쿼리가 수행하면서
sql_profile 칼럼에 표시됨
```

[6] Profile Drop

```
<profile 확인>
select name, category, substr(sql_text, 1, 64) text
from DBA_SQL_PROFILES;
name, category,  text
PROFILE_0h60tr4xc2mub     DEFAULT   "select * from exem_t1 t1 where c1 in (select c1
from exem_t1 whe"

<profile 삭제>
exec dbms_sqltune.drop_sql_profile('PROFILE_dgbmvzfktnt7w');
```

8. 주의사항 & 고급사용

비효율 쿼리를 개선하는 과정에서 쿼리를 REWRITE 하는 경우 개선 후 쿼리와 같은 플랜으로 적용되지 않는다. 쿼리에 없던 Alias 를 적용하는 경우와 QB_NAME 힌트를 사용하는 경우에도 플랜이 적용되지 않는다.

Oracle 쿼리는 Parsing 과정에서 Query Block 단위로 나눠진다. 우리가 사용한 USE_NL(T1 T2) 힌트를 OUTLINE 에서 확인해보면 아래와 같이 쿼리 블록을 이용하고 있음을 확인 할 수 있다.

```
USE_NL(@"SEL$5DA710D3" "T1"@"SEL$1")
USE_NL(@"SEL$5DA710D3" "T2"@"SEL$1")

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

   1 - SEL$5DA710D3
   4 - SEL$5DA710D3 / EXEM_T1@SEL$2
   5 - SEL$5DA710D3 / EXEM_T1@SEL$2
   6 - SEL$5DA710D3 / T1@SEL$1
   7 - SEL$5DA710D3 / T2@SEL$1
```

느낌이 오는가? QB_NAME 힌트를 사용하지 않고 SEL$1 이름을 그대로 사용할 수 있다. Optimizer 는 기본적으로 메인 쿼리 블록부터 SEL$1 SEL$2 순으로 블록을 지정한다. 따라서 한 쿼리 안에 같은 테이블이 여러 번 등장하거나, 동일한 Alias 가 여러 번 등장하여도 내부적으로 사용하는 Query Block Name 을 이용하여 OUTLINE 교체가 가능하다.