Setup SQL Plan Baselines

Differences between SQL Plan Baselines and SQL Profile

1. Baseline is proactive
   - Profile is reactive
2. Baselines are created before problems occur
   - Profile is created by SQL Tuning Advisor after a problem
3. Baselines reproduce a specific plan
   - Profile corrects optimizer cost estimates

NOTE: not all hints can be stored in SQL Plan Baselines. To check : select name from v$sql_hint where version_outline IS NULL

Sources of SQL Plan Baselines

1. SQL Tuning Set(STS)
2. Shared SQL Area (from SGA)
3. Staging table
4. Stored Outline

Capturing SQL Plan Baselines

A) Automatic Capture
   - set init.ora – optimizer_capture_sql_plan_baselines = TRUE
   - first time a query is run a signature is created and placed in the SQL log
   - second time a query is run, a Sql Plan Baseline is created and marked as accepted
   - third time and beyond that a query is run the execution plan is compared to the existing SQL Plan Baseline execution plan. If they don't match the execution plan is added to the SQL Plan Baseline and marked as non-accepted.

B) Manually Loading SQL Plan Baselines

1. SQL tuning set (STS)
   a) execute a query or queries . ex. Select * from atable where id = 3124:
   b) Create SQL Tuning Set:

```
EXEC DBMS_SQLTUNE.create_sqlset(sqlset_name => 'MY_SQLSET');
```

   c) load the plan from the shared SQL area into a SQL tuning set named.

```
DECLARE
    l_cursor  DBMS_SQLTUNE.sqlset_cursor;
BEGIN
  OPEN l_cursor FOR
            SELECT VALUE(a)
            FROM   TABLE(
```

```
        DBMS_SQLTUNE.select_cursor_cache(
basic_filter   => 'sql_text LIKE "%atable where id%" and parsing_schema_name = "MY_USER"',
      attribute_list => 'ALL')
   ) a;

     DBMS_SQLTUNE.load_sqlset(sqlset_name     => 'spa_test_sqlset',
           populate_cursor => l_cursor);
     END;
     /
```

d) verify SQL is in SQL tuning set.

- SELECT SQL_TEXT
  FROM DBA_SQLSET_STATEMENTS
  WHERE SQLSET_NAME = 'MY_SQLSET';

d) Load the plan from the STS into the SQL Plan Baseline

- VARIABLE cnt NUMBER
  EXECUTE :cnt := DBMS_SPM.LOAD_PLANS_FROM_SQLSET(-
  sqlset_name => 'MY_SQLSET', -
  basic_filter => 'sql_text like "%atable where id%"');

e) Query the data dictionary to ensure that the plan was loaded.

- SELECT SQL_HANDLE, SQL_TEXT, PLAN_NAME,
  ORIGIN, ENABLED, ACCEPTED FROM DBA_SQL_PLAN_BASELINES;

f) Or if necessary, drop the plan

- EXEC SYS.DBMS_SQLTUNE.DROP_SQLSET(sqlset_name => 'MY_SQLSET', -
  sqlset_owner => 'SPM');

2. Shared SQL area (load cursors stored in the library cache)

a) execute a query

b) view cursor information:

- Select * from table(dbms_xplan.display_cursor);

c) Load the plans for the specified statements into the SQL plan baseline.

- VARIABLE cnt NUMBER
  EXECUTE :cnt := DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE(-
  sql_id => '27m0sdw9snw59', plan_hash_value => NULL);

- VARIABLE cnt NUMBER
  EXECUTE :cnt := DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE

(attribute_name => 'sql_text', attribute_value => '%my_sql%');

- Execution plans loaded using dbms_spm.load_plans_from_cursor_cache are stored as accepted.

d) Query the data dictionary to ensure that the query plans were loaded into the baseline

- SELECT SQL_HANDLE, SQL_TEXT, PLAN_NAME,
  ORIGIN, ENABLED, ACCEPTED
  FROM   DBA_SQL_PLAN_BASELINES;

3. Staging table (transfer Baseline information to another database)

a) Create a staging table (example uses name stage1)

- BEGIN
  DBMS_SPM.CREATE_STGTAB_BASELINE (
  table_name => 'stage1');
  END;
  /

b) Pack the SQL plan baselines into staging table

- DECLARE
  my_plans number;
  BEGIN
  my_plans := DBMS_SPM.PACK_STGTAB_BASELINE (
  table_name => 'stage1'
  , enabled   => 'yes'
  , creator   => 'spm'
  );
  END;
  /

c) export table using expdp

d) transfer dump file to new database and import using impdp

e) Unpack the SQL plan baselines from the staging table into the SQL management base

- DECLARE
  my_plans NUMBER;
  BEGIN
  my_plans := DBMS_SPM.UNPACK_STGTAB_BASELINE (
  table_name => 'stage1'
  , fixed   => 'yes'
  );
  END;

/

Plan Evolution

A) Evolving SQL Plan Baselines Automatically

1. Configuring the Automatic SPM Evolve Advisor Task

a) The DBMS_SPM package enables you to configure automatic plan evolution
b) The task is owned by SYS
c) When ACCEPT_PLANS is true (default), SQL plan management automatically accepts all plans recommended by the task. When set to false, the task verifies the plans and generates a report of its findings, but does not evolve the plans.
d) set automatic evolution task parameters

- query the current task settings
  - COL PARAMETER_NAME FORMAT a25
    COL VALUE FORMAT a10
    SELECT PARAMETER_NAME, PARAMETER_VALUE AS "VALUE"
    FROM DBA_ADVISOR_PARAMETERS
    WHERE ((TASK_NAME = 'SYS_AUTO_SPM_EVOLVE_TASK') AND
    ((PARAMETER_NAME = 'ACCEPT_PLANS') OR
    (PARAMETER_NAME = 'TIME_LIMIT')));
- Set parameters using PL/SQL code of the following form:
  BEGIN
  DBMS_SPM.SET_EVOLVE_TASK_PARAMETER(
  task_name => 'SYS_AUTO_SPM_EVOLVE_TASK'
  , parameter => parameter_name
  , value => value
  );
  END;
  /
- For example, the following PL/SQL block sets a time limit to 20 minutes, and also automatically accepts plans:
  - BEGIN
  - DBMS_SPM.SET_EVOLVE_TASK_PARAMETER(
  - task_name => 'SYS_AUTO_SPM_EVOLVE_TASK'
  - , parameter => 'LOCAL_TIME_LIMIT'
  - , value => 1200
  -);
  - DBMS_SPM.SET_EVOLVE_TASK_PARAMETER(
  - task_name => 'SYS_AUTO_SPM_EVOLVE_TASK'
  - , parameter => 'ACCEPT_PLANS'
  - , value => 'true'
  -);
  - END;
  - /

2. Use SQL Tuning Advisor
   a) SQL Tuning Advisor accept_sql_profiles parameter is set to TRUE(default is FALSE)

- if SQL Tuning Advisor notices a non-accepted SQL Plan Baseline leads to better performance, it will create SQL profile to accept the SQL Plan Baseline. This will be implemented automatically if accept_sql_profiles parameter is TRUE.

b) check value of accept_sql_profiles parameter:
   - Select parameter_value
     from dba_advisor_parameters
     where task_name = 'SYS_AUTO_SQL_TUNING_TASK'
     and parameter_name = 'ACCEPT_SQL_PROFILES';

c) set parameter to true:
   - dbms_auto_sqltune.set_auto_tuning_task_parameter
     (parameter => 'ACCEPT_SQL_PROFILES',
     value => 'TRUE');

A) Evolving SQL Plan Baselines Manually

a) initial setup
   - ALTER SYSTEM SET
     OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES=TRUE;

b) Create an evolve task
   - execute a sql query: ex. Select * from atable
   - The plan baseline for this statement is empty because the database only captures plans for repeatable statements
     - verify: SELECT SQL_HANDLE, SQL_TEXT, PLAN_NAME, ORIGIN, ENABLED,
       ACCEPTED, FIXED, AUTOPURGE
       FROM DBA_SQL_PLAN_BASELINES
       WHERE SQL_TEXT LIKE '%your_query%';
   - execute the statement for a second time.
   - Query the data dictionary to ensure that the plans were loaded into the plan baseline for the statement. Run the above SELECT statement.
   - Explain the plan for the statement and verify that the optimizer is using this plan
     - SELECT SQL_HANDLE, SQL_TEXT, PLAN_NAME, ENABLED, ACCEPTED
       FROM DBA_SQL_PLAN_BASELINES
       WHERE SQL_TEXT LIKE '%your_query%';
   - execute the DBMS_SPM.CREATE_EVOLVE_TASK function and then obtain the name of the task
     - CONNECT / AS SYSDBA
       VARIABLE cnt NUMBER
       VARIABLE tk_name VARCHAR2(50)
       VARIABLE exe_name VARCHAR2(50)
       VARIABLE evol_out CLOB
       EXECUTE :tk_name :=
       DBMS_SPM.CREATE_EVOLVE_TASK(
       sql_handle => 'SQL_07f16c76ff893342',
       plan_name => 'SQL_PLAN_0gwbcfvzskcu20135fd6c');

     - SELECT :tk_name FROM DUAL;

c) Execute the evolve task
   - EXECUTE :exe_name
     :=DBMS_SPM.EXECUTE_EVOLVE_TASK(task_name=>:tk_name);
     SELECT :exe_name FROM DUAL;

d) Implement the recommendations in the task
   - run the report and implement recommendation.
   - `EXECUTE :cnt :=`
     `DBMS_SPM.IMPLEMENT_EVOLVE_TASK( task_name=>:tk_name,`
     `execution_name=>:exe_name );`
   - Query the data dictionary to ensure that the new plan is accepted.

e) Report on the task outcome
   - EXECUTE :evol_out :=
     DBMS_SPM.REPORT_EVOLVE_TASK(task_name=>:tk_name,
     execution_name=>:exe_name);
   - SELECT :evol_out FROM DUAL;

f) Implement the recommendations of the evolve task
   - EXECUTE :cnt :=
     DBMS_SPM.IMPLEMENT_EVOLVE_TASK(task_name=>:tk_name,
     execution_name=>:exe_name);

g) Query the data dictionary to ensure that the new plan is accepted
   - SELECT SQL_HANDLE, SQL_TEXT, PLAN_NAME, ORIGIN,
     ENABLED, ACCEPTED
     FROM DBA_SQL_PLAN_BASELINES
     WHERE SQL_HANDLE IN ('SQL_07f16c76ff893342')
     ORDER BY SQL_HANDLE, ACCEPTED;

Displaying SQL Plan Baselines

1. select * from table(dbms_xplan.display_sql_plan_baseline(sql_handle =>
   'SQL_987654321abcdefghi'));

Altering SQL Plan Baselines

1. example: disable execution plan for a SQL Plan Baseline
   - ret := dbms_spm.alter_sql_plan_baseline(sql_handle => 'SQL_987654321abcdefgh',
     plan_name => 'SQL_PLAN_abcdefgh123456789',
     attribute_name => 'enabled',
     attribute_value => 'no');