Fixing a plan in Oracle Database

Using Oracle baselines you can fix the sql plan for a SQLID:

SQL plan management is a preventative mechanism that records and evaluates the execution plans of SQL statements over time. This mechanism can build a SQL plan baseline, which is a set of accepted plans for a SQL statement. The accepted plans have been proven to perform well.

The goal of SQL plan baselines is to preserve the performance of corresponding SQL statements, regardless of changes in the database. Examples of changes include:

1. New optimizer version
2. Changes to optimizer statistics and optimizer parameters
3. Changes to schema and metadata definitions
4. Changes to system settings
5. SQL profile creation

SQL plan baselines cannot help in cases where an event has caused irreversible execution plan changes, such as dropping an index.

The SQL tuning features of Oracle Database generate SQL profiles that help the optimizer to produce well-tuned plans. However, this mechanism is reactive and cannot guarantee stable performance when drastic database changes occur. SQL tuning can only resolve performance issues after they have occurred and are identified.

For example, a SQL statement may become high-load because of a plan change, but SQL tuning cannot solve this problem until after the plan change occurs.

Use the below queries to see the available execution plans and see which plan was running fine.

Using the AWR SQL Report

1. @?/rdbms/admin/awrsqrpi.sql ---> This will generate the html page for the required query based on the SQLID and its awr history.

Or

Getting Execution plan from AWR

2. dbms_xplan.display_awr()
Ex: select * from TABLE(dbms_xplan.display_awr('47qjdv3ncanhr'));

Or

3. USING GRID Control 12c

To gather the history of a SQL execution and the plans used during those runs, obtain the SQL Id to be evaluated, connect to the GRID Control 12c:

Select the Targets/Databases -->Select the database -->Performance/SQL/Search SQL--> Check AWR Snapshots -->Enter the SQL ID in the SQL ID filed / Search--> Verify the executions and the different Hash Plans used.

The ones with the Smallest Elapsed Times are the best execution Plans for the SQL.

1. If the HASH Plan is still in the Cursor Cache it can be created as a baseline and instructed to run every time that SQL ID is loaded to the Shared Pool.

2. If the HASH Plan is no longer in the Cursor Cache, then it is still possible to load the HASH Plan to a Sql Tuning Set and create a baseline from the STS and assign it the SQL ID as well. Take note of the Snap ID (from the GRID SQL Search above) for the desired HASH Plan.

HASH /SQL plan needed found in the Cursor Cache/Memory

Now you know which hash plan hash to be fixed. Now follow the below example. If the needed plan is found in the cursor cache then it is very simple to create a baseline and fixing the plan for the SQL query.

Ex: Determined the Hash Plan: 2601263939 is the best to run against the
SQL ID: 47qjdv3ncanhr

1.Create the Baseline:

```
SQL> var v_num number;
SQL> exec :v_num:=dbms_spm.load_plans_from_cursor_cache
    (sql_id =>'47qjdv3ncanhr',plan_hash_value => 2601263939);
```

2. Verify the baseline got created or not

```
SQL> select sql_handle, plan_name, enabled, accepted, fixed
     From dba_sql_plan_baselines order by last_modified;

SQL_HANDLE                     PLAN_NAME                                                      ENA ACC FIX
------------------------------ -------------------------------------------------------------- --- --- ---
SQL_473f79643cdab4f3           SQL_PLAN_4fgvtchydpd7maff63102                                  YES YES NO
```

3.TO MODIFY A SQL PLAN BASELINE

```
SQL> var v_num number;
SQL> exec :v_num:=dbms_spm.ALTER_SQL_PLAN_BASELINE
     (sql_handle =>'SQL_473f79643cdab4f3',
      plan_name => 'SQL_PLAN_4fgvtchydpd7maff63102',
      attribute_name=> 'FIXED',
      attribute_value  => 'YES');
```

Attributes

1. **enabled** (YES/NO) : If YES, the plan is available for the optimizer if it is also marked as accepted.
2. **fixed** (YES/NO) : If YES, the SQL plan baseline will not evolve over time. Fixed plans are used in preference to non-fixed plans.
3. **autopurge** (YES/NO) : If YES, the SQL plan baseline is purged automatically if it is not used for a period of time.
4. **plan_name** : Used to amend the SQL plan name, up to a maximum of 30 character.
5. **description** : Used to amend the SQL plan description, up to a maximum of 30 character.

4. Check the status

```
SQL> select sql_id,plan_hash_value,sql_plan_baseline
     From v$sqlarea where sql_id = '47qjdv3ncanhr';

SQL_ID          PLAN_HASH_VALUE    SQL_PLAN_BASELINE
--------------- ---------------    ------------------------------
47qjdv3ncanhr   2601263939         SQL_PLAN_4fgvtchydpd7maff63102
```

Sometimes the required HASH / SQL plan will not be present in the Cursor Cache, then you have to load it from a AWR snapshots.

To load plans to the cursor cache from awr snapshots:

1. Drop SQL Tuning Set (STS)

```
SQL> BEGIN
       DBMS_SQLTUNE.DROP_SQLSET( sqlset_name => 'SAMPLE_TUNING_SET');
     END;
```

2. Create SQL Tuning Set (STS)

```
SQL> BEGIN
       DBMS_SQLTUNE.CREATE_SQLSET(sqlset_name => 'SAMPLE_TUNING_SET',
description => 'SQL Tuning Set for loading plan into SQL Plan
Baseline');
     END;
```

3. Populate STS from AWR using a time duration when the desired plan was used.

```
-- Retrieve the begin Snap ID from the same session described in the
GRID Control above or by :

SQL> SELECT SNAP_ID, BEGIN_INTERVAL_TIME,END_INTERVAL_TIME
     FROM dba_hist_snapshot ORDER BY END_INTERVAL_TIME DESC;


-- Specify the sql_id in the basic_filter (other predicates are
available, see desc dba_hist_snapshot) if necessary.
DECLARE
cur sys_refcursor;
BEGIN
OPEN cur FOR
SELECT VALUE(P) FROM TABLE(
dbms_sqltune.select_workload_repository(begin_snap=>1477,
end_snap=>1478,
basic_filter=>'sql_id = ''47qjdv3ncanhr''',attribute_list=>'ALL')) p;
DBMS_SQLTUNE.LOAD_SQLSET( sqlset_name=>'SAMPLE_TUNING_SET',
populate_cursor=>cur);
CLOSE cur;
END;
```

4. List out SQL Tuning Set contents to check we got what we wanted

```
SQL> SELECT first_load_time,executions as execs,parsing_schema_name,
     elapsed_time / 1000000 as elapsed_time_secs,cpu_time / 1000000 as
     cpu_time_secs,buffer_gets,disk_reads,direct_writes,rows_processed,
     fetches,optimizer_cost,sql_plan,plan_hash_value,sql_id,sql_text
     FROM TABLE(DBMS_SQLTUNE.SELECT_SQLSET(sqlset_name =>
                                           'SAMPLE_TUNING_SET'));
```

5. Finally create the baseline from the STS:

```
SQL> DECLARE
       my_plans pls_integer;
     BEGIN
       my_plans := DBMS_SPM.LOAD_PLANS_FROM_SQLSET(
       sqlset_name => 'SAMPLE_TUNING_SET',
       basic_filter=>'plan_hash_value = ''2601263939''' );
     END;/
```

6. Verify the baseline got created and modify it if necessary

```
SQL> select sql_handle, plan_name, enabled, accepted, fixed
     From dba_sql_plan_baselines;

SQL> exec :v_num:=dbms_spm.ALTER_SQL_PLAN_BASELINE (
     sql_handle =>'SQL_473f79643cdab4f3',
     plan_name => 'SQL_PLAN_4fgvtchydpd7maff63102',
     attribute_name=> 'FIXED',
     attribute_value => 'YES');
```

7. Verify all details for the new Baseline:

```
SQL> select * from table( dbms_xplan.display_sql_plan_baseline(
     sql_handle=>'SQL_473f79643cdab4f3',format=>'basic'));
```