**Oracle SQL Tuning – Tuning Tools**

PART I – Automatic SQL Tuning

**Topics Covered**:

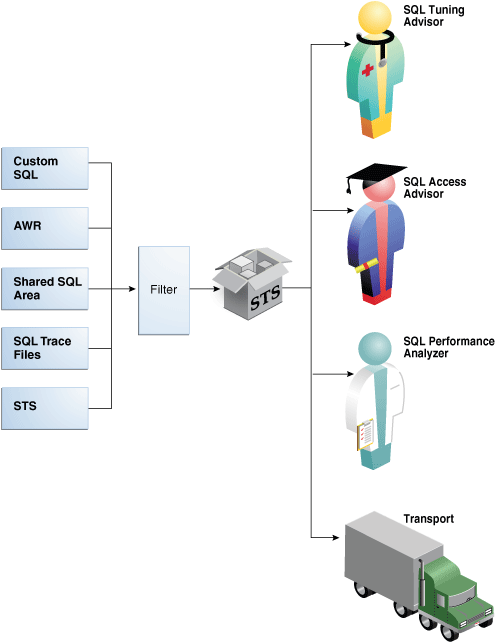
1. SQL Tuning Sets
2. SQL Tuning Advisor
3. Automatic SQL Tuning Task
4. On-demand SQL Tuning Task
5. SQL Tuning Advisor Architecture
6. SQL Access Advisor
7. SQL Profiles
8. SQL Plan Management (SPM)

**Automatic SQL Tuning - Oracle SQL Tuning Set**

**SQL Tuning Set (STS)** is a database object that you can use as input to various Oracle Performance Tuning Advisor, such as SQL Tuning Advisor, SQL Access Advisor, SQL Performance Analyzer, etc. We can then pass the STS object to a We can also export the STS to a Test database and test SQL Tuning tests there instead of Production database. STS are portable.

STS contains Execution context, Execution statistics, and Execution plans. Execution context conatains the User schema, Application module, bind variables and environment properties. Execution statistics contains elapsed time, CPU time, Buffer gets, Disk Reads, etc. STS can be created from Shared SQL Area (cursor cache), AWR, Custom SQL (script), SQL trace files and other STS.

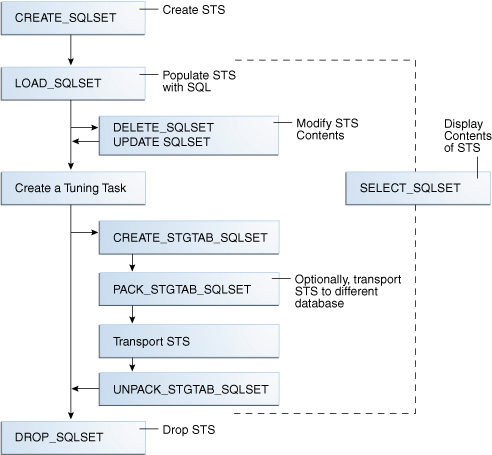
While SQL Tuning Advisor can take its input from Shared SQL area (cursor cache), ADDM or AWR, SQL Tuning Sets are still used to provide input to SQL Tuning Advisor task and hence cosnidered as the Foundation of Oracle SQL tuning.



SQL Tuning Sets are managed by Oracle supplied Package: DBMS\_SQLTUNE.

|  |  |
| --- | --- |
| **Tasks** | **The Procedure in DBMS\_SQLTUNE** |
| Create STS | CREATE\_SQLSET |
| Populate STS with SQL | LOAD\_SQLSET |
| Modify STS Contents | DELETE\_SQLSET, UPDATE\_SQLSET |
| Display contents of STS | SELECT\_SQLSET |
| Transport STS to a different database | CREATE\_STGTAB\_SQLSET  PACK\_STGTAB\_SQLSET  UNPACK\_STGTAB\_SQLSET |
| Drop STS | DROP\_SQLSET |

**STS Workflow**:



To list existing STS:

|  |
| --- |
| SELECT NAME, STATEMENT\_COUNT, DESCRIPTION FROM DBA\_SQLSET; |

You can select SQL statements from Cursor cache, list them by ELAPSED\_TIME or CPU\_TIME order. RESULT\_LIMIT specifies 10 SQL statements to be displayed. RANKING\_MEASURE1 specifies the paramter to be used for ordering the statements.

|  |
| --- |
| set linesize window  SELECT LAST\_EXEC\_START\_TIME, ELAPSED\_TIME/1000/1000 ELAPSED\_SEC, SQL\_ID, SQL\_TEXT  FROM  DBMS\_SQLTUNE.SELECT\_CURSOR\_CACHE  (  BASIC\_FILTER =>'PARSING\_SCHEMA\_NAME = ''DWH\_DATA'' AND SQL\_TEXT LIKE ''SELECT%''',  RESULT\_LIMIT => 100,  RANKING\_MEASURE1 =>'ELAPSED\_TIME'  ); |

To create a STS:

|  |
| --- |
| BEGIN  DBMS\_SQLTUNE.CREATE\_SQLSET  (  SQLSET\_NAME => ‘HR\_WKLD\_SET’,  SQLSET\_OWNER => ‘HR’,  DESCRIPTION => ‘HR WORKLOAD TO TUNE’  )  END; |

Populate STS with SQL from Cursor cache:

|  |
| --- |
| BEGIN  DBMS\_SQLTUNE.CAPTURE\_CURSOR\_CACHE\_SQLSET  (  SQLSET\_NAME => ‘HR\_WKLD\_SET’,  SQLSET\_OWNER => ‘HR’,  TIME\_LIMIT => 40,  REPEAT\_INTERVAL => 3,  BASIC\_FILTER => ‘ UPPER (PARSING\_SCHEMA\_NAME) = ‘HR’ AND  MODULE = ‘’TESTING\_SESSION’’’,  CAPTURE\_MODE => DBMS\_SQLTUNE.MODE\_REPLACE\_OLD\_STATS  )  END; |

**TIME\_LIMIT**: This parameters specifies the total number of seconds the pulling process should keep running. It should be set to the time after which we expect the workload to finish. After this time, STS capture will stop.

REPEAT\_INTERVAL: Number of seconds Oracle will pause between sampling.

**BASIC\_FILTER**: This parameter controls which statement will be loaded into STS.

**CAPTURE\_MODE**: It could be Update (replace) or Merge.

**Following are the BASIC\_FILTER parameters available**:

|  |
| --- |
| SQL\_TEXT for SQL Text |
| PARSING\_SCHEMA\_NAME for user schema |
| MODULE in DBMS\_APPLICATION\_INFO |
| ACTION in DBMS\_APPLICATION\_INFO |
| ELAPSED\_TIME |
| CPU\_TIME |
| BUFFER\_GETS |
| DISK\_READS |
| ROWS\_PROCESS |
| EXECUTIONS |

You can load STS with SQL from Cusrsor cache at once instead of of running it periodically.

|  |
| --- |
| DECLARE  C\_SQLAREA\_CURSOR DBMS\_SQLTUNE.SQLSET\_CURSOR;  BEGIN  OPEN C\_SQL\_AREA\_CUSTOR FOR SELECT VALUE (P) FROM TABLE  (  DBMS\_SQLTUNE.SELECT\_CURSOR\_CACHE  (  ‘MODULE => ‘’TESTING\_SESSION’’ AND PARSING\_SCHEMA = ‘’HR’’’  )  ) P;  DBMS\_SQLTUNE.LOAD\_SQLSET  (  SQLSET\_NAME => ‘HR\_WKLD\_SET’,  SQLSET\_OWNER => ‘HR’,  POPULATE\_CURSOR => C\_SQLAREA\_CURSOR  )  END; |

To display the contents (for example, statistics) of STS:

|  |
| --- |
| SELECT SQL\_ID, PARSING\_SCHEMA, SQL\_TEXT, ELAPSED\_TIME, BUFFER\_GETS  FROM TABLE (DBMS\_SQLTUNE.SELECT\_CURSOR\_CACHE (‘HR\_WKLD\_STS’)); |

Modifying STS: Usually not required but sometime you may need to update some attributes, most commonly PRIORITY of a SQL in STS.

|  |
| --- |
| DBMS\_SQLTUNE.UPDATE\_SQLSET  (  SQLSET\_NAME => ‘HR\_WKLD\_STS’,  SQL\_ID => ‘’,  ATTRIBUTE\_NAME => ‘PRIORITY’,  ATTRIBUTE\_VALUE => 1  ); |

To load STS from another STS (useful when a STS is too large and you want to load specific SQL statements based on filter criteria or want to divide a STS into multiple STS):

|  |
| --- |
| DECLARE CUR DBMS\_SQLTUNE.SQLSET\_CURSOR;  BEGIN  OPEN CUR FOR  SELECT VALUE (P)  FROM TABLE(DBMS\_SQLTUNE.SELECT\_SQLSET(SQLSET\_NAME =>'STS1',  RANKING\_MEASURE1 =>'ELAPSED\_TIME', RESULT\_LIMIT=>&LIMIT)) P;  DBMS\_SQLTUNE.LOAD\_SQLSET(SQLSET\_NAME => 'STS1\_A', POPULATE\_CURSOR => CUR);  CLOSE CUR;  END; |

Transportig SQL Tuning Set:

|  |
| --- |
| EXEC DBMS\_SQLTUNE.CREATE\_STGTAB\_SQLSET  (  TABLE\_NAME => ‘HR\_WKLD\_TABLE’, SCHEMA => ‘HR’  );  --  BEGIN  DBMS\_SQLTUNE.PACK\_STGTAB\_SQLSET  (  SQLSET\_NAME => ‘HR\_WKLD\_STS’,  SQLSET\_OWNER => ‘HR’,  STAGING\_TABLE\_NAME => ‘HR\_WKLD\_TABLE’,  STAGING\_SCHEMA\_NAME => ‘HR’  )  END; |

Use Data pump to transport the table to the TEST database and UNPACK the STS:

|  |
| --- |
| BEGIN  DBMS\_SQLTUNE.UNPACK\_STAGTAB\_SQLSET (SQL\_SET\_NAME => ‘%’,  REPLACE => TRUE, STAGING\_TABLE\_NAME => ‘HR\_WKLD\_TABLE’  END; |

UNPACK\_STAGTAB\_SQLSET uses wild card character so you can copy all or specified SQL sets into the TEST database SQL sets.

To drop STS:

|  |
| --- |
| BEGIN  DBMS\_SQLSET.DROP\_SQLSET( sqlset\_name => 'SQLT\_WKLD\_STS' );  END; |

**STS Dictionaries**:

|  |
| --- |
| DBA\_SQLSET |
| DBA\_SQLSET\_STATEMENTS |
| DBA\_SQLSET\_BINDS |
| DBA\_SQLSET\_PLANS |

**Automatic SQL Tuning – SQL Tuning Advisor**

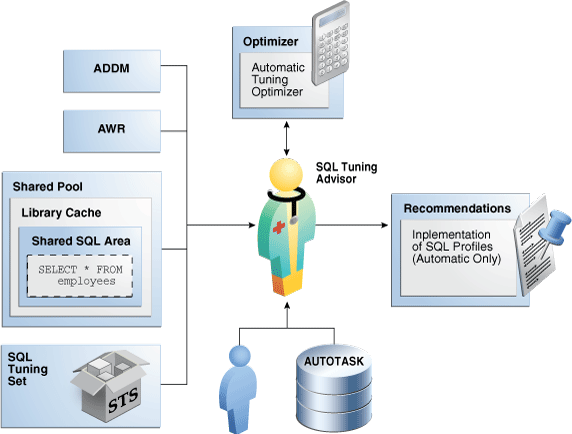
SQL Tuning Advisor is a SQL diagnostic software in Oracle Database Tuning pack which takes one or more than one SQL statements as input and produce advice or recommendations for how to tune the SQL statements with a rationale and expected benefit. It also helps prevent regressions by only executing optimal plans.

**Tuning recommendations include**:

1. Collection of object statistics
2. Creation of Indexes
3. Rewritting SQL statements
4. Creation of SQL profiles
5. Creation of SQL Plan baseline

SQL Tuning Advisor can receive SQL statements as input from multiple sources, analyze these statements using the Optimizer and make recommendations. SQL Tuning Advisor invokes **Automatic Tuning Optimizer** to analyze SQL statements. Optimizer, when invoked by SQL Tuning Advisor, works in Tuning mode (as opposed to when SQL statements are executed).

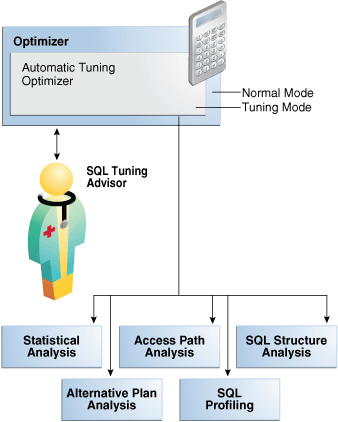
Since invoking Automatic Tuning Optimizer for every hard parse consumes significant time and resources, it’s only meant for complex and high-load SQL statements. SQL Tuning Advisor framework provides a common schema and interface for storing task objects. SQL Tuning Advisor receives SQL statements and writes to Advisor schema tables by means of Advisor framework and reads Advisor schema tables to generate recommendation reports.



**SQL Tuning Advisor Architecture**:

SQL Tuning Advisor performs the following analysis before providing its recommendations:

1. Statistical Analysis
2. SQL Profiling
3. Access Path Analysis
4. SQL Structural Analysis
5. Alternative Plan Analysis



**Statistical Analysis**: Automatic Tuning Optimizer (Optimizer in Tuning mode) checks for missing or stale statistics, and recommends gathering fresh statistics if needed. ATO checks two kinds of statistics, **Object statistics** and **System statistics**. Object statistics checks includes statistics for each objecte referenced in the query. System statistics includes /O seek time, multiblock read count, and I/O transfer speed on Oracle Exadata machine. If gathering these system statitics improves the execution plan, ATO recommends accepting a SQL profile.

**SQL Profiling**: SQL profiling is the verification by the Automatic Tuning Optimizer of its own estimates. When Optimizer runs in **normal mode** it may estimate the cardinality of a predicate resulting in sub-optimal execution plan. Incorrect cardinality estimate is caused by insufficient statistics or complexity of the predicate. When SQL Tuning Advisor runs against the SQL statement, Optimizer is invoked in **tuning mode** and performs deep analysis of predicates. It reviews execution history, executes the SQL statements partially or samples data and applies predicates to the samples. This way Optimizer derives the most accurate cardinality, selectivity and other information required to produce an optimal execution plan. When these values are significantly different than the original execution plan values, SQL Tuning Advisor creates SQL a profile. SQL Profiles will be dealt with in greater detail in PART – II.

**Access Path Analysis**: Access Path is the means by which the database retrieves data. For example a query can use Index or Full table scan (FTS). FTS can sometimes be better than Index scan and vice-versa. Automatic SQL Tuning Optimizer explores whether a new index can significantly enhance query performance and recommends either 1) creating an index or 2) running SQL Access Advisor. Creating a new index recommendation is specific to the SQL statement, not the entire SQL workload, processed by STA. However, the new Index may adversely affect the performance of the entire SQL workload hence Automatic Tuning Optimizer recommends running SQL Access Advisor on the specific SQL statement (for which index was recommended by STA) along with with a representative SQL workload. SQL Access Advisor examines the effect of creating an index on the SQL workload before making recommendations.

**SQL Structural Analysis:** Automatic Tuning Optimizer tries to identify syntactic, semantic, or deign problems that may lead to suboptimal performance during SQL structural analysis and advises us how to restructure them.

Following categories of structural problems are analysed by Automatic Tuning Optimizer:

1. **Inefficient use of SQL constructs**: The SQL may be using NOT IN instead of NOT EXISTS or UNION instead of UNION ALL resulting in suboptimal performance.
2. **Data type mismatch**: If an Indexed column and compared value have different data type then database does not use the index. Also, database has to consume additional resources to convert data types. Columns that contain numeric data but never participates in arithmetic operation should not be declared as numeric.
3. **Design mistakes**: a classic example of design mistake is a missing join condition. If *n* is the number of tables then there must exist *n-1* joins to avoid cartesian join.

Automatic Tuning Optimizer can detect all such cases and suggests alternative statements.

**Alternative Plan Analysis:** While tuning SQL statement SQL Tuning Advisor searches real-time and historical performance data for alternative execution plan for the statement. If plans otherthan the original plan exists STA reports an alternative plan. SQL Tuning Advisor validates the alternative execution plans and notes any plans that are not reproducible. When reproducible alternative plans are found, you can create a **SQL plan baseline** to instruct the optimizer to choose these plans in the future. SQL Tuning Advisor only recommends an alternative plan if the elapsed time of the original plan is worse than alternative plans.

**Alternative Plan Finding**:

|  |
| --- |
| 2- Alternative Plan Finding  ---------------------------  Some alternative execution plans for this statement were found by searching  the system's real-time and historical performance data.    The following table lists these plans ranked by their average elapsed time.  See section "ALTERNATIVE PLANS SECTION" for detailed information on each  plan.    id plan hash last seen elapsed (s) origin note  -- ---------- -------------------- ------------ --------------- ----------------  1 1378942017 2009-02-05/23:12:08 0.000 Cursor Cache original plan  2 2842999589 2009-02-05/23:12:08 0.002 STS    Information  -----------  - The Original Plan appears to have the best performance, based on the  elapsed time per execution. However, if you know that one alternative  plan is better than the Original Plan, you can create a SQL plan baseline  for it. This will instruct the Oracle optimizer to pick it over any other  choices in the future.  execute dbms\_sqltune.create\_sql\_plan\_baseline(task\_name => 'TASK\_XXXXX',  object\_id => 2, task\_owner => 'SYS', plan\_hash => xxxxxxxx); |

**Alternate Plan Section**:

|  |
| --- |
| Plan 1  ------  Plan Origin :Cursor Cache  Plan Hash Value :1378942017  Executions :50  Elapsed Time :0.000 sec  CPU Time :0.000 sec  Buffer Gets :0  Disk Reads :0  Disk Writes :0    Notes:  1. Statistics shown are averaged over multiple executions.  2. The plan matches the original plan.    -----------------------------------------------------------------  | Id | Operation | Name |  -----------------------------------------------------------------  | 0 | SELECT STATEMENT | |  | 1 | SORT AGGREGATE | |  | 2 | MERGE JOIN | |  | 3 | INDEX FULL SCAN | TEST1\_INDEX |  | 4 | SORT JOIN | |  | 5 | TABLE ACCESS FULL | TEST |  -----------------------------------------------------------------    Plan 2  ------  Plan Origin :STS  Plan Hash Value :2842999589  Executions :10  Elapsed Time :0.002 sec  CPU Time :0.002 sec  Buffer Gets :3  Disk Reads :0  Disk Writes :0    Notes:   1. Statistics shown are averaged over multiple executions.   -------------------------------------------------------  | Id | Operation | Name |  -------------------------------------------------------  | 0 | SELECT STATEMENT | |  | 1 | SORT AGGREGATE | |  | 2 | HASH JOIN | |  | 3 | TABLE ACCESS FULL | TEST |  | 4 | TABLE ACCESS FULL | TEST1 |  ------------------------------------------------------- |

To adopt an alternative plan regardless of whether SQL Tuning Advisor recommends it, call **DBMS\_SQLTUNE.CREATE\_SQL\_PLAN\_BASELINE**. You can use this procedure to create a SQL plan baseline on any existing reproducible plan.

**SQL Tuning Advisor Operations**:

You can run SQL Tuning Advisor **automatically (Automatic SQL Tuning Task)** or **On Demand**. You can also run the advisor on a local or remote database.

**Automatic SQL Tuning Task**:

STA can be configured to run in nightly maintenance window (called **Automatic SQL Tuning Task or Automatic SQL Tuning Advisor**). Oracle Scheduler uses the automated maintenance tasks infrastructure (known as **AutoTask**) to schedules tasks to run automatically. By default, the Automatic SQL Tuning task runs for at most one hour in a nightly maintenance window. When the STA is run by the AUTOTASK job it looks into the AWR for the Top SQL statements that consumes resources and analyze them (by invoking Automatic Tuning Optimizer), generates and saves the recommendations into ADVISOR schema tables. It does not automatically implement the recommendations.

The automated SQL tuning task does *not* process the following types of SQL:

1. Ad hoc SQL statements or SQL statements that do not repeat within a week
2. Parallel queries.
3. Queries that have already been SQL profiled.
4. Recursive SQL.

Automatic SQL Tuning Advisor data is stored in the CDB root and is only visible to a common user whose current container is the root. While it might have results about SQL statements executed in a PDB that were analyzed by STA, these results are not included if PDB is unplugged. Also, the results cannot be viewed when the current container is a PDB.

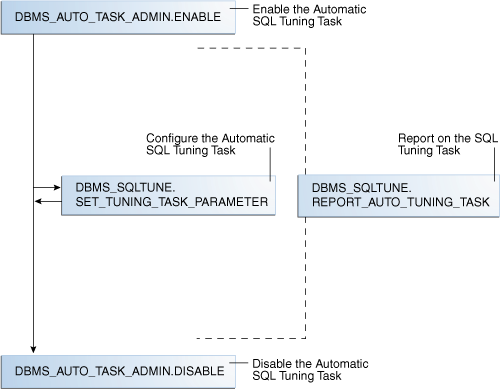
**Managing Automatic SQL Tuning Task**:

You can manage Automatic SQL Tuning Task using command line or OEM. Oracle provides two PL/SQL packages to manage Automatic SQL Tuning Task:

1. DBMS\_AUTO\_SQLTUNE (needs **ADVISOR** privilege).
2. DBMS\_AUTO\_TASK\_ADMIN (to enable/disable Automatic SQL Tuning Task).

**Basic Tasks for Automatic SQL Tuning:**

1. Enable Automatic SQL Tuning task.
2. Configure Automatic SQL Tuning task (optional).
3. Display the results of Automatic SQL Tuning task.
4. Disable the Automatic SQL Tuning task.



**Note:** Check **STATISTICS\_LEVEL** and ensure that it’s NOT set to BASIC. BASIC disables all Advisor and statistics, including Automatic SQL Tuning Advisor.

Enable Automatic SQL Tuning task:

|  |
| --- |
| BEGIN  DBMS\_AUTO\_TASK\_ADMIN.ENABLE  (  CLIENT\_NAME => 'SQL TUNING ADVISOR'  , OPERATION => NULL  , WINDOW\_NAME => NULL  );  END; |

Query the data dictionary to confirm the change:

|  |
| --- |
| COL CLIENT\_NAME FORMAT a20  SELECT CLIENT\_NAME, STATUS  FROM DBA\_AUTOTASK\_CLIENT  WHERE CLIENT\_NAME = 'SQL TUNING ADVISOR'; |

Display Automatic SQL Tuning task parameters (from DBA\_ADVISOR\_PARAMETERS):

|  |
| --- |
| 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\_SQL\_TUNING\_TASK**') AND  ( (PARAMETER\_NAME LIKE '%PROFILE%') OR  (PARAMETER\_NAME = 'LOCAL\_TIME\_LIMIT') OR  (PARAMETER\_NAME = 'EXECUTION\_DAYS\_TO\_EXPIRE') ) );  --  -- OUTPUT  --  PARAMETER\_NAME VALUE  ------------------------- ----------  EXECUTION\_DAYS\_TO\_EXPIRE 30  LOCAL\_TIME\_LIMIT 1000  ACCEPT\_SQL\_PROFILES FALSE  MAX\_SQL\_PROFILES\_PER\_EXEC 20  MAX\_AUTO\_SQL\_PROFILES 10000 |

Configure Automatic SQL Tuning Task paremeters:

|  |
| --- |
| BEGIN  DBMS\_SQLTUNE.SET\_TUNING\_TASK\_PARAMETER(**'SYS\_AUTO\_SQL\_TUNING\_TASK**',  'LOCAL\_TIME\_LIMIT', 1200);  DBMS\_SQLTUNE.SET\_TUNING\_TASK\_PARAMETER('SYS\_AUTO\_SQL\_TUNING\_TASK',  'ACCEPT\_SQL\_PROFILES', 'true');  DBMS\_SQLTUNE.SET\_TUNING\_TASK\_PARAMETER('SYS\_AUTO\_SQL\_TUNING\_TASK',  'MAX\_SQL\_PROFILES\_PER\_EXEC', 50);  DBMS\_SQLTUNE.SET\_TUNING\_TASK\_PARAMETER('SYS\_AUTO\_SQL\_TUNING\_TASK',  'MAX\_AUTO\_SQL\_PROFILES', 10002);  END; |

Task name must be **SYS\_AUTO\_SQL\_TUNING\_TASK.**

**ACCEPT\_SQL\_PROFILES:** Whether the task should accept SQL profiles **automatically** (TRUE or FALSE).

**MAX\_AUTO\_SQL\_PROFILES**: Maximum number of automatic SQL profiles allowed on the system, in sum.

**MAX\_SQL\_PROFILES\_PER\_EXEC**: maximum number of SQL profiles that can be automatically implemented per execution of the task.

**LOCAL\_TIME\_LIMIT**: Automatic STA will not spend more than 1200 seconds per SQL statement.

**Viewing Automatic SQL Tuning Reports**:

At any point of time during or after the running of the Automatic SQL Tuning task, you can view a tuning report.

You can get the report by calling DBMS\_AUTO\_SQLTUNE.REPORT. The report contains the following sections (depending on the sections you requested):

**General Information**: Includes information about the input given to the Automatic SQL Tuning report, number of SQL statements tuned duing maintenance, and number of SQL profiles created.

**Summary**: It lists the SQL statements (by their SQL identifiers) that were tuned during the maintenance window and the estimated benefit of each SQL profile, or the execution statistics after performing a test execution of the SQL statement with the SQL profile.

**Tuning findings**: This section contains all findings associated with each SQL statement analyzed by STA, whether the profile is implemented on the database and why, detailed execution statistics captured when testing the SQL profile.

**Explain plans**: This section shows the old and new explain plans used by each SQL statement analyzed by SQL Tuning Advisor.

**Errors**: List of errors encountered by the Automatic SQL Tuning task.

To create and access Automatic SQL Tuning Advisor report:

|  |
| --- |
| VARIABLE my\_rept CLOB;  BEGIN  :MY\_REPT :=DBMS\_SQLTUNE.REPORT\_AUTO\_TUNING\_TASK  (  BEGIN\_EXEC => NULL – name of the beginning task. NULL = most recent  , END\_EXEC => NULL – name of the ending task. NULL = most recent  , TYPE => 'TEXT' – type of report to produce (TEXT)  , LEVEL => 'TYPICAL' – other values are BASIC, ALL  , SECTION => 'ALL' – SUMMARY, FINDINGS, ERROR, INFORMATION, ALL  , OBJECT\_ID => NULL  , RESULT\_LIMIT => NULL – maximum number of SQL statements to report  );  END;  /  PRINT :my\_rept |

Here are some samples from REPORT\_AUTO\_TUNING\_TASK report:

**General Information**:

|  |
| --- |
| GENERAL INFORMATION SECTION  ------------------------------------------------------------------------  Tuning Task Name : SYS\_AUTO\_SQL\_TUNING\_TASK  Tuning Task Owner : SYS  Workload Type : Automatic High-Load SQL Workload  Execution Count : 6  Current Execution : EXEC\_170  Execution Type : TUNE SQL  Scope : COMPREHENSIVE  Global Time Limit(seconds) : 3600  Per-SQL Time Limit(seconds) : 1200  Completion Status : COMPLETED  Started at : 04/16/2012 10:00:00  Completed at : 04/16/2012 10:07:11  Number of Candidate SQLs : 17  Cumulative Elapsed Time of SQL (s) : 8 |

**Findings**:

|  |
| --- |
| ------------------------------------------------------------------------  SQLs with Findings Ordered by Maximum (Profile/Index) Benefit, Object ID  ------------------------------------------------------------------------  ob ID SQL ID stats profile(benefit) index(benefit) restructure  ------ ------------- ----- ---------------- -------------- -----------  82 dqjcc345dd4ak 58.03%  72 51bbkcd9zwsjw 2  81 03rxjf8gb18jg  ------------------------------------------------------------------------  DETAILS SECTION  ------------------------------------------------------------------------  Statements with Results Ordered by Max (Profile/Index) Benefit, Obj ID  ------------------------------------------------------------------------  Object ID : 82  Schema Name: DBA1  SQL ID : dqjcc345dd4ak  SQL Text : SELECT status FROM dba\_autotask\_client WHERE client\_name=:1    ------------------------------------------------------------------------  FINDINGS SECTION (1 finding)  ------------------------------------------------------------------------    1- SQL Profile Finding (see explain plans section below)  --------------------------------------------------------  A potentially better execution plan was found for this statement.  The SQL profile was not automatically created because the verified  benefit was too low.    Recommendation (estimated benefit: 58.03%)  ------------------------------------------  - Consider accepting the recommended SQL profile.  execute dbms\_sqltune.accept\_sql\_profile (task\_name =>  'SYS\_AUTO\_SQL\_TUNING\_TASK', object\_id => 82, replace => TRUE);    Validation results  ------------------  The SQL profile was tested by executing 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(us): 26963 8829 67.25 %  CPU Time(us): 27000 9000 66.66 %  User I/O Time(us): 25 14 44 %  Buffer Gets: 905 380 58.01 %  Physical Read Requests: 0 0  Physical Write Requests: 0 0  Physical Read Bytes: 0 0  Physical Write Bytes: 7372 7372 0 %  Rows Processed: 1 1  Fetches: 1 1  Executions: 1 1    Notes  -----  1. The original plan was first executed to warm the buffer cache.  2. Statistics for original plan were averaged over next 9 executions.  3. The SQL profile plan was first executed to warm the buffer cache.  4. Statistics for the SQL profile plan were averaged over next 9 executions. |

To display the past Automatic SQL Tuning task executions:

|  |
| --- |
| SELECT, EXECUTION\_NAME, EXEC\_START, EXEC\_END,  (EXECUTION\_END – EXECUTION\_START) \* 24 \* 60 PERIOD\_M,  STATUS,  ERROR\_MESSAGEERROR  FROM DBA\_ADVISOR\_EXECUTIONS  WHERE TASK\_NAME = ‘SYS\_AUTO\_SQL\_TUNING\_TASK’  ORDER BY EXECUTION\_ID ASC; |

You can also manually invoke the Automatic SQL Tuning Task:

|  |
| --- |
| SET SERVEROUTPUT ON  DECLARE  v\_return VARCHAR2(20);  BEGIN  v\_return := DBMS\_AUTO\_SQLTUNE.EXECUTE\_AUTO\_TUNING\_TASK;  DBMS\_OUTPUT.PUT\_LINE (v\_return);  END; |

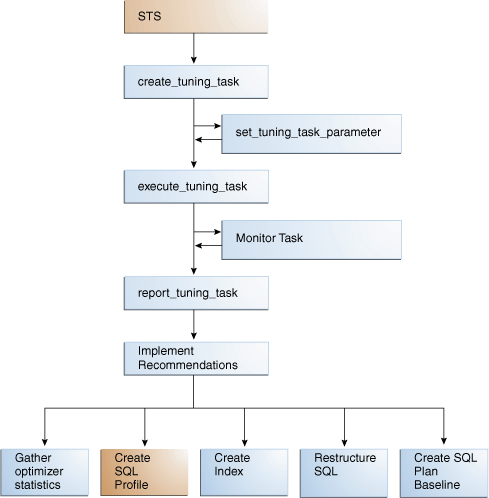
**SQL Tuning Advisor On-Demand** :

On Demand SQL Tuning is invoked to tune SQL reactively when users report performance issues. Typically, you invoke SQL Tuning Advisor to run ADDM proactively, or to tune SQL statement reactively when users complain about suboptimal performance. In both the proactive and reactive scenarios, running SQL Tuning Advisor is usually the quickest way to fix unexpected SQL performance problems.

You can run SQL Tuning Advisor using procedures in the DBMS\_SQLTUNE package. You must have ADVISOR privilege to these APIs.

**Basic Tasks in On-Demand SQL Tuning Advisor**:

1. Prepare or create input to SQL Tuning Advisor. Input could be either text of a single SQL statement or SQL Tuning Set that contains one or more than one SQL statements.
2. Create SQL Tuning Task.
3. Configure SQL Tuning task (optional).
4. Executing SQL Tuning task.
5. Check the status of SQL Tuning task.
6. Display the results of SQL Tuning task.
7. Implement recommendations as appropriate.



**Different forms of CREATE\_TUNING\_TASK**:

CREATE\_TUNING\_TASK can take input from the following sources:

1. Single SQL statement given as text.
2. SQL ID of a single SQL statement from Shared SQL area.
3. Range of snapshot identifiers from AWR.
4. SQL Tuning Set (STS).
5. SQL Performance Analyzer.

For each form we have a CREATE\_TUNING\_TASK procedure form.

For example, to create Tuning task from SQL ID:

|  |
| --- |
| DBMS\_SQLTUNE.CREATE\_TUNING\_TASK  (  sql\_id IN VARCHAR2,  plan\_hash\_value IN NUMBER := NULL,  scope IN VARCHAR2 := SCOPE\_COMPREHENSIVE,  time\_limit IN NUMBER := TIME\_LIMIT\_DEFAULT,  task\_name IN VARCHAR2 := NULL,  description IN VARCHAR2 := NULL,  con\_name IN VARCHAR2 := NULL,  database\_link\_to IN VARCHAR2 := NULL  )  RETURN VARCHAR2; |

**SCOPE**: This is a very important parameter.

**LIMITED**: STA produces recommendations based on statistical, access path, and SQL structure analysis but it does not produce SQL profile recommendations.

**COMPREHENSIVE**: STA carries out all the analysis it performs under LIMITED and recommends SQL profiles.

**BIND\_LIST**: it defines an ordered list of bind values in ANYDATA type. It’s useful when you are passing a SQL text with bind varaibles.

|  |
| --- |
| DECLARE  V\_SQL\_TEXT VARCHAR2(1000);  V\_SQL\_TUNE\_TASK\_ID VARCHAR2(100);  BEGIN  V\_SQL\_TEXT := 'SELECT U.USERID, U.USERNAME '||  'FROM USERS U, REGION R '||  'WHERE U.REGID=R.REGID AND U.USERID=:UID AND R.REGCLASS=:CID';  V\_SQL\_TUNE\_TASK\_ID := DBMS\_SQLTUNE.CREATE\_TUNING\_TASK (  SQL\_TEXT => V\_SQL\_TEXT,  BIND\_LIST => SQL\_BINDS(ANYDATA.CONVERTNUMBER(100)),  USER\_NAME => 'HR',  SCOPE => DBMS\_SQLTUNE.SCOPE\_COMPREHENSIVE,  TIME\_LIMIT => 1000,  TASK\_NAME => 'TEST\_TUNING\_TASK',  DESCRIPTION => 'TUNING TASK FOR A PROBLEMATIC QUERY...');  DBMS\_OUTPUT.PUT\_LINE('V\_SQL\_TUNE\_TASK\_ID: ' || V\_SQL\_TUNE\_TASK\_ID);  END; |

**USER\_NAME**: specifies the name of the user for whom the SQL has to be tuned. The user must have ADVISOR privilege.

To check the status of the Tuning task:

|  |
| --- |
| COL TASK\_ID FORMAT 999999  COL TASK\_NAME FORMAT a25  COL STATUS\_MESSAGE FORMAT a33  SELECT TASK\_ID, TASK\_NAME, STATUS, STATUS\_MESSAGE  FROM DBA\_ADVISOR\_LOG WHERE TASK\_NAME = 'TEST\_TUNING\_TASK'; |

**Configure SQL Tuning task (optional)**:

Maximum time that the SQL tuning task will be changed to to 300 seconds. The task will run maximum for 300 seconds.

|  |
| --- |
| BEGIN  DBMS\_SQLTUNE.SET\_TUNING\_TASK\_PARAMETER (  TASK\_NAME => 'STA\_SPECIFIC\_EMP\_TASK'  , PARAMETER => 'TIME\_LIMIT'  , VALUE => 300  );  END; |

To check the parameters of a SQL Tuning task:

|  |
| --- |
| COL PARAMETER\_NAME FORMAT a25  COL VALUE FORMAT a15  SELECT PARAMETER\_NAME, PARAMETER\_VALUE AS "VALUE"  FROM USER\_ADVISOR\_PARAMETERS  WHERE TASK\_NAME = 'STA\_SPECIFIC\_EMP\_TASK'  AND PARAMETER\_VALUE != 'UNUSED'  ORDER BY PARAMETER\_NAME; |

**Executing a SQL Tuning Task**:

To execute a SQL Tuning Task from command line:

|  |
| --- |
| BEGIN  DBMS\_SQLTUNE.EXECUTE\_TUNING\_TASK(task\_name=>'STA\_SPECIFIC\_EMP\_TASK');  END; |

To check the status of the SQL Tuning Task:

|  |
| --- |
| COL TASK\_ID FORMAT 999999  COL TASK\_NAME FORMAT a25  COL STATUS\_MESSAGE FORMAT a33  SELECT TASK\_ID, TASK\_NAME, STATUS, STATUS\_MESSAGE  FROM DBA\_ADVISOR\_LOG WHERE TASK\_NAME = 'STA\_SPECIFIC\_EMP\_TASK'; |

To monitor the status of a SQL Tuning Tasks:

|  |
| --- |
| SELECT STATUS  FROM DBA\_ADVISOR\_TASKS  WHERE TASK\_NAME = 'STA\_SPECIFIC\_EMP\_TASK'; |

To monitor the progress of a SQL Tuning Task (using TASK\_ID):

|  |
| --- |
| VARIABLE my\_tid NUMBER;  EXEC :my\_tid := 884  COL ADVISOR\_NAME FORMAT a20  COL SOFAR FORMAT 999  COL TOTALWORK FORMAT 999  SELECT TASK\_ID, ADVISOR\_NAME, SOFAR, TOTALWORK,  ROUND(SOFAR/TOTALWORK\*100,2) "%\_COMPLETE"  FROM GV$ADVISOR\_PROGRESS  WHERE TASK\_ID = :my\_tid; |

To cancel a running SQL Tuning Task:

|  |
| --- |
| VARIABLE my\_task VARCHAR2;  EXEC :my\_task := 'STA\_SPECIFIC\_EMP\_TASK'  EXEC DBMS\_SQLTUNE.CANCEL\_TUNING\_TASK(:my\_task); |

**Reporting the recommensations and findings of a SQL Tuning Task**:

After the SQL Tuning Task is completed, you need to display its findings and recommendations. The report contains all the findings and recommendations of SQL Tuning Advisor. For each proposed recommendation, the report provides the rationale and benefit along with the SQL statements needed to implement the recommendation.

To report the recommendations and findings, execute REPORT\_TUNING\_TASK functions:

|  |
| --- |
| DBMS\_SQLTUNE.REPORT\_TUNING\_TASK  (  TASK\_NAME IN VARCHAR2,  TYPE IN VARCHAR2 := 'TEXT',  LEVEL IN VARCHAR2 := 'TYPICAL',  SECTION IN VARCHAR2 := ALL,  OBJECT\_ID IN NUMBER := NULL,  RESULT\_LIMIT IN NUMBER := NULL,  OWNER\_NAME IN VARCHAR2 := NULL,  EXECUTION\_NAME IN VARCHAR2 := NULL,  DATABASE\_LINK\_TO IN VARCHAR2 := NULL  )  RETURN CLOB; -- A CLOB containing the desired report |

**Parameters**:

**LEVEL**: BASIC/TYPICAL/ALL.

**SECTION**: SUMMARY/FINDINGS/PLAN/INFORMATION/ERROR/ALL.

**EXECUTION\_NAME**: If NULL the function generates the report for the last task execution.

**Example**:

|  |
| --- |
| VARIABLE sta\_task VARCHAR2;  EXEC :sta\_task := 'STA\_SPECIFIC\_EMP\_TASK'  SELECT  DBMS\_SQLTUNE.REPORT\_TUNING\_TASK  (:sta\_task, 'TEXT', 'ALL', 'ALL')  FROM DUAL; |

**Automatic SQL Tuning – SQL Access Advisor**

**SQL Access Advisor** is a diagnostic software that identifies and helps resolve SQL performance problems by recommending indexes, materialized views, materialized view logs, or partitions (collectively called access structure) to create, drop or retain.

**Input to SQL Access Advisor**: SQL Access Advisor takes input from three sources:

1. **Shared SQL Area**: Database analyzes recent SQL statements in Shared SQL Area from V$SQL view.
2. **SQL Tuning Set**: SQL Tuning Set is a database object that stores SQL statements along with their execution context and execution plan. For best results, create a STS from a workload (set of SQL statements) sources such as Shared SQL area (SQL cache) or a user defined workload stored in a table or a hypothetical workload. The STS created can be passed to SQL Access Advisor.
3. **Hypothetical workload**: You can create a hypothetical workload from a schema by analyzing dimensions and constraints. This option is useful when you are initially designing your application.

**Filer Options for SQL Access Advisor**:

We can apply a filter to a workload to restrict what is analyzed. This is achieved by Oracle DBMS\_ADVISOR.SET\_TASK\_PARAMETER procedure. The procedure takes parameters which we would briefly discuss.

**SQL Access Advisor Recommendations**:

SAA can recomends the creation or dropping of different database objects upon analyzing the database objects.

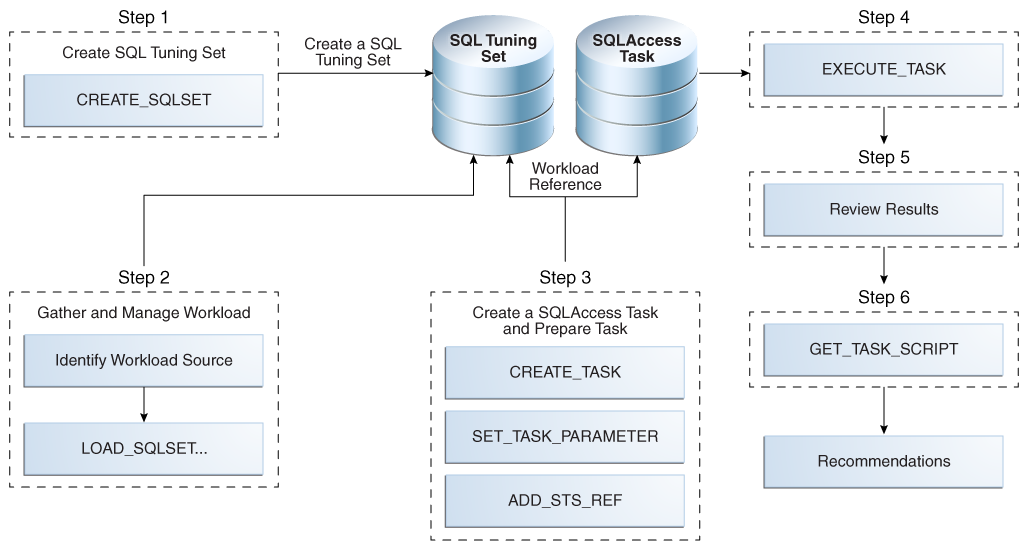
1. **Index**: SAA can recomends creation of B-Tree, Bitmap or function based indexes.
2. **Materialized view**: SAA uses TUNE\_MVIEW procedure recommends how to optimize Materialized Views so that they can be fast refreshable and take advantage of qury rewrite.
3. **Materialized view logs**: A Materialized view log is a master site which records DML changes to the master or master materialized view. Fast refresh of MV is only possible the MV has a MV log.
4. **Partitions**: SAA can recommend partitioning on an existing unpartitioned base table to improve performance.

To make recommendations, SQL Access Advisor relies on structural statistics about table and index cardinalities of dimension level columns, JOIN KEY columns, and fact table key columns. Because gathering statistics is time-consuming and full statistical accuracy is not required, it is usually preferable to estimate statistics. Without gathering statistics on a specified table, queries referencing this table are marked as invalid in the workload, resulting in no recommendations for these queries.

**SQL Access Advisor: Basic Tasks**:

Following are the basic tasks to invoke SAA:

1. Create SQL Tuning set
2. Load SQL Tuning set
3. Create and configure a task
4. Execute the task
5. View recommendations
6. Generate and execute a SQL scripts that implements the recommendations (optional)



**Create SQL Tuning Set as Input to SAA**:

STS is stored as a separate entity hence can be shared across more than one SAA tasks. If a SAA task reference an STS, you cannot delete or modify the STS until the dependency on all SAA tasks are removed. The dependency is removed when you manually remove the reference from SAA tasks or parent SAA tasks are deleted.

The user creating STS must have ADMINISTRATOR SQL TUNING SET privilege. To run SAA task owned by other users, the user must have ADMINISTRATOR ANY SQL TUNING SET privilege.

To create an STS:

|  |
| --- |
| SET SERVEROUTPUT ON;  VARIABLE task\_id NUMBER;  VARIABLE task\_name VARCHAR2(255);  VARIABLE workload\_name VARCHAR2(255);  --  EXECUTE :workload\_name := 'MY\_STS\_WORKLOAD';  EXECUTE DBMS\_SQLTUNE.CREATE\_SQLSET(:workload\_name, 'test purpose'); |

**Populate a SQL Tuning set with a User defined workload**:

Create a table called USER\_WORKLOAD:

|  |
| --- |
| DROP TABLE user\_workload;  CREATE TABLE user\_workload  (  username varchar2(128), /\* User who executes statement \*/  module varchar2(64), /\* Application module name \*/  action varchar2(64), /\* Application action name \*/  elapsed\_time number, /\* Elapsed time for query \*/  cpu\_time number, /\* CPU time for query \*/  buffer\_gets number, /\* Buffer gets consumed by query \*/  disk\_reads number, /\* Disk reads consumed by query \*/  rows\_processed number, /\* # of rows processed by query \*/  executions number, /\* # of times query executed \*/  optimizer\_cost number, /\* Optimizer cost for query \*/  priority number, /\* User-priority (1,2 or 3) \*/  last\_execution\_date date, /\* Last time query executed \*/  stat\_period number, /\* Window exec time in seconds \*/  sql\_text clob /\* Full SQL Text \*/  ); |

Load USER\_WORKLOAD table with information about queries:

|  |
| --- |
| -- aggregation with selection  INSERT INTO user\_workload (username, module, action, priority, sql\_text)  VALUES ('SH', 'Example1', 'Action', 2,  'SELECT t.week\_ending\_day, p.prod\_subcategory,  SUM(s.amount\_sold) AS dollars, s.channel\_id, s.promo\_id  FROM sales s, times t, products p  WHERE s.time\_id = t.time\_id  AND s.prod\_id = p.prod\_id  AND s.prod\_id > 10  AND s.prod\_id < 50  GROUP BY t.week\_ending\_day, p.prod\_subcategory, s.channel\_id, s.promo\_id')  /  -- aggregation with selection  INSERT INTO user\_workload (username, module, action, priority, sql\_text)  VALUES ('SH', 'Example1', 'Action', 2,  'SELECT t.calendar\_month\_desc, SUM(s.amount\_sold) AS dollars  FROM sales s , times t  WHERE s.time\_id = t.time\_id  AND s.time\_id BETWEEN TO\_DATE(''01-JAN-2000'', ''DD-MON-YYYY'')  AND TO\_DATE(''01-JUL-2000'', ''DD-MON-YYYY'')  GROUP BY t.calendar\_month\_desc')  /  -- order by  INSERT INTO user\_workload (username, module, action, priority, sql\_text)  VALUES ('SH', 'Example1', 'Action', 2,  'SELECT c.country\_id, c.cust\_city, c.cust\_last\_name  FROM customers c  WHERE c.country\_id IN (52790, 52789)  ORDER BY c.country\_id, c.cust\_city, c.cust\_last\_name')  /  COMMIT; |

Filling STS with SQL statements from USER\_WORKLOAD table:

Execute a PL/SQL program to that fills a cursor with rows from USER\_WORKLOAD table and then loads the contents of this cusrsor into the STS MY\_STS\_WORKLOAD created in the previous step.

|  |
| --- |
| DECLARE  sqlset\_cur DBMS\_SQLTUNE.SQLSET\_CURSOR;  BEGIN  OPEN sqlset\_cur FOR  SELECT SQLSET\_ROW(null,null, SQL\_TEXT, null, null, 'SH', module,  'Action', 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, null, 2, 3,  sysdate, 0, 0, null, 0, null, null)  FROM USER\_WORKLOAD;  DBMS\_SQLTUNE.LOAD\_SQLSET('MY\_STS\_WORKLOAD', sqlset\_cur);  END;  / |

**Creating and configuring a SAA task**:

This steps involves create a task, configuring the task by setting different parameters and linking the task with the workload. A workload is independent of a SAA task and can be linked more than one SAA tasks.

Create and configure a SAA task:

|  |
| --- |
| EXEC :task\_name := 'MYTASK';  EXEC DBMS\_ADVISOR.CREATE\_TASK('SQL Access Advisor', :task\_id, :task\_name);  --  EXEC DBMS\_ADVISOR.SET\_TASK\_PARAMETER(:task\_name, 'TIME\_LIMIT', 30);  EXEC DBMS\_ADVISOR.SET\_TASK\_PARAMETER(:task\_name, 'ANALYSIS\_SCOPE', 'ALL'); |

Linking the task with a workload:

|  |
| --- |
| EXECUTE DBMS\_ADVISOR.ADD\_STS\_REF(:task\_name, 'SH', :workload\_name); |

**Executing SAA task**:

When executing the SAA task, SAA attempts to validate each statement to identify table and column references. The database process each statement as if it were being executed by the statements original user. If the current database user does not have SELECT privilege to a table, SAA bypasses the statement. The SELECT privilege cannot be obtained through a role.

Execute a task:

|  |
| --- |
| EXECUTE DBMS\_ADVISOR.EXECUTE\_TASK(:task\_name); |

To check the status of the task:

|  |
| --- |
| COL TASK\_ID FORMAT 999  COL TASK\_NAME FORMAT a25  COL STATUS\_MESSAGE FORMAT a25  SELECT TASK\_ID, TASK\_NAME, STATUS, STATUS\_MESSAGE  FROM DBA\_ADVISOR\_LOG; |

**Viewing SAA Task recommendations**:

You can view recommendations using two methods:

1. Run DBMS\_ADVISOR.GET\_TASK\_SCRIPT procedure.
2. From data dictionary views.

Data Dictionary Views:

|  |  |
| --- | --- |
| **Name** | **Description** |
| DBA\_ADVISOR\_TASKS | Displays information about advisor tasks such as name and Task ID. |
| DBA\_ADVISOR\_RECOMMENDATIONS | Displays the results of an analysis of all recommendations in the database. Each recommendation is identified by a REC\_ID. It also has the RANK (in terms of importance), BENEFIT (calculated benefit value). |
| DBA\_ADVISOR\_ACTIONS | Each recommendation has one or more actions identified by ACTION\_ID. The COMMAND is the command to be executed on the OBJECT\_ID. The ATTR1 to ATTR6 and NUM\_ATTR1 to NUM\_ATTR5 specify different attributes of the access structure or the database object identified by the OBJECT\_ID. |
| DBA\_ADVISOR\_RATIONALE | Displays information about the rationales for all recommendations in the database. |
| DBA\_ADVISOR\_SQLA\_WK\_STMTS | Displays information about all workload objects in the database after a SQL Access Advisor analysis. The precost and postcost numbers are in terms of the estimated optimizer cost (shown in EXPLAIN PLAN) without and with the recommended access structure. |

To view the results of a SQL Access Advisor task:

|  |
| --- |
| VARIABLE workload\_name VARCHAR2(255);  VARIABLE task\_name VARCHAR2(255);  EXECUTE :task\_name := 'MYTASK';  EXECUTE :workload\_name := 'MY\_STS\_WORKLOAD';  SELECT REC\_ID, RANK, BENEFIT  FROM DBA\_ADVISOR\_RECOMMENDATIONS  WHERE TASK\_NAME = :task\_name  ORDER BY RANK;  REC\_ID RANK BENEFIT  ---------- ---------- ----------  1 1 236  2 2 356 |

The benefit is the **total improvement** in execution cost (in terms of optimizer cost) of **all queries** using the recommendation.

Identify which query benefits from which recommendation:

|  |
| --- |
| SELECT SQL\_ID, REC\_ID, PRECOST, POSTCOST,  (PRECOST-POSTCOST)\*100/PRECOST AS PERCENT\_BENEFIT  FROM DBA\_ADVISOR\_SQLA\_WK\_STMTS  WHERE TASK\_NAME = :task\_name  AND WORKLOAD\_NAME = :workload\_name  ORDER BY percent\_benefit DESC;  SQL\_ID REC\_ID PRECOST POSTCOST PERCENT\_BENEFIT  ------------- ---------- ---------- ---------- ---------------  fn4bsxdm98w3u 2 578 222 61.5916955  29bbju72rv3t2 1 5750 5514 4.10434783  133ym38r6gbar 0 772 772 0 |

Display the actions for a set of recommendations:

|  |
| --- |
| SELECT REC\_ID, ACTION\_ID, SUBSTR(COMMAND,1,30) AS command  FROM USER\_ADVISOR\_ACTIONS  WHERE TASK\_NAME = :task\_name  ORDER BY rec\_id, action\_id;  REC\_ID ACTION\_ID COMMAND  ---------- ---------- ------------------------------  1 1 PARTITION TABLE  1 2 RETAIN INDEX  2 1 PARTITION TABLE  2 3 RETAIN INDEX  2 4 RETAIN INDEX |