EXAM OBJECTIVE: USE THE NEW OPTIMIZER STATISTICS

In Oracle 10g, there is an introduction of some new statistics that are used by the advisors to help in tuning of the SQL statements.

The new statistics added are:

SQL_ID: This is a more unique identifier or hash values that is assigned for each SQL statement.

SQL Stats: Wait Class time, PLSQL time and Java time

Bind Variables: Sampled values from the V$SQL_BIND_CAPTURE dynamic performance view. This view is new in Oracle 10g.

Top SQL identification based on CPU, Elapsed and Parse Statistics.

EXAM OBJECTIVE: USE THE SQL TUNING ADVISOR

In Oracle 10g, the SQL Tuning Advisor is a new feature that is designed to replace to replace manual tuning of SQL statements. The SQL advisor can receive as input one or more SQL statements and respond with an optimized execution plan. You are also provided with the rationale for the advice, the estimated performance benefit and the actual SQL statement to execute in order to implement the advice. It uses the Cost Based Optimizer to perform some internal analysis needed by it. The analyses it performs include:

- **Stale/Missing statistics analysis** – The cost based optimizer (CBO) needs up-to-date statistics in order to make appropriate decisions about the execution plan. In Oracle 10g, the automatic statistics gathering feature can be used if desired. Any object that has stale or missing statistics is identified and recommendations are given to collect statistics on them.

- **SQL Profile** – This step is related to tuning a plan. When operating in a normal mode the optimizer uses whatever statistics are available to generate a good plan as quickly as possible.

  However, the CBO can be used in Plan Tuning Mode. In this mode the cost based optimizer takes its time to check if the statistics are accurate. It may collect additional background information for a particular SQL statement.

It creates what is called as a SQL Profile. The SQL Profile consists of auxiliary statistics specific to that statement. A SQL Profile can be stored in the data dictionary. Once a SQL Profile is generated you can decide whether to activate it or not. Once you decide to use it, it can be used under normal node without any change to the application code. The SQL Profile recommendations are generated on if the scope of the SQL Advisor is a comprehensive mode.

- **Access Path Analysis** – During the step, the Cost Based Optimizer may recommend the use of new indexes or materialized views/logs that would greatly improve the execution plan.

- **SQL Analysis by restructuring SQL** – During this step the CBO may identify the SQL statements that are poorly written and generate inefficient plans and make suggestions on how they should be restructured. For e.g. the use of the NOT IN may be substituted by NOT EXISTS or a UNION with a UNION ALL or the removal of a Cartesian product. Both these constructs are very similar.

INPUT TO THE SQL ADVISOR

- Poorly written SQL statements identified by the ADDM
- SQL statements that are currently in the Library Cache.
- SQL statements from the Automatic Workload Repository (AWR).
- Custom Workloads, that consist of a series of SQL statements that a user wishes to tune.

When you have multiple statements for input, another object called the SQL Tuning Set (STS) can be created. It is a user-defined set of SQL statements.

The SQL Tuning Advisor can be launched from the EM console. If the input is from a SQL statement identified by the ADDM, you can launch SQL advisor from the Performance Finding Details page in the ADDM.

COMMAND-LINE USAGE

The following procedures listed below that are part of the DBMS_SQLTUNE package can be used if you prefer to work in the command-line mode.

Subprogram	Description
CREATE_TUNING_TASK Functions	Prepares the tuning of a single statement or SqlSet
DROP_TUNING_TASK Procedure	Drops a SQL tuning task
CANCEL_TUNING_TASK Procedure	Cancels the currently executing tuning task
EXECUTE_TUNING_TASK Procedure	Executes a previously created tuning task
REPORT_TUNING_TASK Function	Displays the results of a tuning task
CREATE_SQLSET Procedure	Creates a SqlSet object in the database
DELETE_SQLSET Procedure	Deletes a set of SQL statements from a SqlSet
DROP_SQLSET Procedure	Drops a SqlSet if it is not active
LOAD_SQLSET Procedure	Populates the SqlSet with a set of selected SQL
SELECT_SQLSET Function	Collects SQL statements from the cursor cache
ACCEPT_SQL_PROFILE Procedure	Create a SQL Profile for the specified tuning task
ADD_SQLSET_REFERENCE Function	Adds a new reference to an existing SqlSet to indicate its use by a client
ALTER_SQL_PROFILE Procedure	Alters specific attributes of an existing SQL Profile object
DROP_SQL_PROFILE Procedure	Drops the named SQL Profile from the database
SELECT_WORKLOAD_REPOSITORY Functions	Collects SQL statements from workload repository

Running SQL Tuning Advisor using the DBMS_SQLTUNE package is a two-step process.

1. Create a SQL tuning task
2. Executing a SQL tuning task

CREATING A SQL TUNING TASK

- Create a bind variable called new_task that will hold the task name. Define it to be a varchar2(100) variable.

  ```
  SQL> VARIABLE new_task VARCHAR2(100)
  ```

- Create a bind variable called new_query that will hold the query. Define it to be a varchar2(1000) variable.

  ```
  SQL> VARIABLE new_query VARCHAR2(1000)

  SQL> BEGIN
  ```

```
        :new_query := 'SELECT ename FROM .emp
                                    WHERE empno = :bnd;'
        END;
/
```

- Begin the tuning process by invoking the CREATE_TUNING_TASK procedure.

```
SQL> BEGIN
            :new_task := DBMS_SQLTUNE.create_tuning_task (
            SQL_TEXT            =>:new_query,
            BIND_LIST=>SQL_BINDS(anydata.ConvertNumber(99)),
            USER_NAME           =>'HR',
            SCOPE               => 'COMPREHENSIVE',
            TIME_LIMIT          => 45,
            TASK_NAME           => 'my_tuning_task',
            DESCRIPTION         => 'Data from the EMP table …');
        END;
/
```

In this example, 99 is the value for bind variable :bnd passed as function argument of type SQL_BINDS, HR is the user under which the CREATE_TUNING_TASK function analyzes the SQL statement, the scope is set to COMPREHENSIVE which means that the advisor also performs SQL Profiling analysis, and 45 is the maximum time in seconds that the function can run. In addition, values for task name and description are provided.

The CREATE_TUNING_TASK function returns the task name that you have provided or generates a unique task name. You can use the task name to specify this task when using other APIs. To view the task names associated with a specific owner, you can run the following:

```
SQL> SELECT task_name
        FROM DBA_ADVISOR_LOG
        WHERE owner = 'HR';
```

- Invoke the EXECUTE_TUNING_TASK procedure to start the tuning process.

```
SQL> BEGIN
        DBMS_SQLTUNE. execute_tuning_task (
            TASK_NAME => 'my_tuning_task');
        END;
        /
```

You can check the status of the task by reviewing the information in the DBA_ADVISOR_LOG view or check execution progress of the task in the V$SESSION_LONGOPS view. For example:

```
SQL> SELECT status
        FROM DBA_ADVISOR_LOG
        WHERE task_name = 'my_tuning_task';
```

- Call the REPORT_TUNING_TASK function to visualize the tuning results.

```
 SQL> SELECT DBMS_SQLTUNE.report_tuning_task
        (TASK_NAME=>: new _task)
        FROM dual;
```

- When a SQL Profile is recommended by the SQL Tuning Advisor, then create the SQL Profile by calling the ACCEPT_SQL_PROFILE function, which stores it in the data dictionary.

  You should have the CREATE ANY SQL PROFILE privilege. Create a bind variable to hold the SQL profile and the display its value. In the example shown below *my_tuning_task* is the name of the SQL tuning task.

```
    SQL> VARIABLE new_profile VARCHAR2 (1000)
    SQL> BEGIN
                 :new _profile := DBMS_SQLTUNE.accept_sql_profile
                         ( TASK_NAME=>'my_tuning_task'
                           NAME = 'my_sql_profile'
                         );
            END;
  /
    SQL> SELECT :new _profile FROM dual;
```

EXAM OBJECTIVE: THE SQL ACCESS ADVISOR

Defining appropriate indexes have always been major task while defining access structures. Materialized views can also be used to improve access in addition to indexes.

- The SQL Access Advisor identifies and helps resolve performance problems related to the execution of SQL statements by recommending which indexes, materialized views, or materialized view logs to create, drop, or retain.

INPUT TO THE SQLACCESS ADVISOR

- The SQL cache (including current and recent SQL activity)
- a user defined custom workload (such as a development environment).
- a hypothetical workload by referencing schemas
- AN SQL Tuning Set (STS) extracted from the workload repository.

SQLACCESS ADVISOR RECOMMENDATIONS

The recommendations by SQL Access Advisor may be generated using one of the two approaches explained below:
FULL: the SQL Access Advisor assumes the workload is a complete and representative set of the applications SQL and considers all aspects of tuning materialized views, materialized view logs and indexes.
PARTIAL: Here the SQL Access Advisor assumes that the workload contains only problematic statements.

Sample output of recommendations generated by SQL Access Advisor

Recommendation	FULL	PARTIAL
Add new index on table or materialized view	YES	YES
Drop an unused index	YES	NO
Modify an existing index by changing index type	YES	NO
Modify an existing index by adding columns at the end	YES	YES
Add a new materialized view	YES	YES
Drop an unused materialized view	YES	NO
Add a new materialized view log	YES	YES
Modify an existing materialized view log to add new columns or clauses	YES	YES

<u>EXAM OBJECTIVE: USE THE PERFORMANCE PAGE OF DATABASE CONTROL</u>

Launch EM console, and select the Database Home page -> Performance Tab -> Performance Page. This page is particularly useful if you want to quickly gauge activity trends for the CPU, memory and disk I/O. You can quickly view the most resource-consuming SQL statements too.