5 Application Tuning

CERTIFICATION OBJECTIVES

Oracle DBAs spend a considerable amount of their time tuning resource-guzzling SQL statements, so they can reduce the load on their databases and increase throughput. The new Automatic Database Diagnostic Monitor (ADDM) makes the identification of these offending SQL statements considerably easier, basing its sophisticated recommendations on the performance data gathered by the Automatic Workload Repository (AWR).

This chapter introduces you to the new SQL Tuning Advisor, which helps you perform the new art of SQL profiling, whereby you help the Oracle optimizer generate better-tuned execution plans.

There are several changes in the way you can collect optimizer statistics, and this chapter explain these in detail. You'll learn how to use the new SQL Access Advisor to define better access structures like indexes. Finally, you'll learn how to use the Performance pages of the OEM Database Control to effortlessly perform your SQL tuning exercises.

In the new database, there are several enhancements regarding the ability to skip unusable indexes, and to create and maintain hash-partitioned global indexes. The final part of this chapter discusses these and other issues, such as specifying storage characteristics for index partitions.

Let's start this important chapter with a summary of the changes in optimizer statistics in Oracle Database 10*g*.

CERTIFICATION OBJECTIVE 5.01

Using the New Optimizer Statistics

There are several interesting changes in the area of optimizer collection and the optimizer modes. Oracle supports only the cost-based query optimizer. The rule-based optimizer is finally obsolete in Oracle Database 10*g*, although it remains as an unsupported feature. This means that the `CHOOSE` and `RULE` values are no longer supported as `OPTIMIZER_MODE` initialization parameters, although, technically, you can still use them. The same is true of the `CHOOSE` and `RULE` hints as well.

Following are the important enhancements and changes in the optimizer's behavior and statistics collection areas.

exam Watch

The default value for the OPTIMIZER_MODE initialization parameter is ALL_ROWS. The ALL_ROWS mode optimizes the throughput of the system, which means that it will minimize the resource cost for executing the entire SQL statement and returning all the rows.

Automatic Statistics Collection

As you learned in the previous chapter, Oracle automatically collects optimizer statistics for you now, using the GATHER_STATS_JOB job. Oracle uses the GATHER_DATABASE_STATS_JOB_PROC procedure to collect optimizer statistics on an automatic basis. Oracle identifies the objects whose optimizer statistics it needs to refresh by using the information stored in AWR. It uses the Scheduler to run the statistics collection task in the predefined "maintenance window." GATHER_DATABASE_STATS_JOB_PROC procedure collects statistics for all objects that have no prior statistics or have stale statistics, because a large proportion of the rows (more than 10 percent) have been modified.

You can still use the DBMS_STATS.GATHER_DATABASE_STATS procedure (with the GATHER AUTO option) to gather similar statistics yourself. However, the big difference is that Oracle *prioritizes* its optimizer collection, ensuring that it collects statistics for objects that have stale or outdated statistics before the maintenance window is up.

The Cost Model

The default cost model in Oracle Database 10*g* is CPU+IO. The computing of both CPU as well as I/O usage gives the optimizer a good idea of the true cost of SQL statements. Oracle captures the CPU information when you start the instance.

exam Watch

The automatic statistics gathering feature is enabled by default. If you wish to disable it, you can do so by executing the following:

```
DBMS_SCHEDULER.DISABLE('GATHER_STATS_JOB');
```

Changes in the DBMS_STATS Package

The DBMS_STATS package has several changes, including changes in the key GATHER_DATABASE_STATS and GATHER_SCHEMA_STATS procedures. There are new values for the GRANULARITY argument in both of these procedures. The GRANULARITY argument is pertinent only if the underlying table is partitioned. AUTO is the default value for this argument. If you specify AUTO as the granularity level, Oracle collects global-, partition-, and subpartition-level statistics, if you use

the LIST subpartitioning method (otherwise, it skips the subpartition-level statistics). If you specify GLOBAL and PARTITION as the options, Oracle doesn't collect subpartition-level statistics.

You have the option of running statistics collection serially or in parallel. The DEGREE parameter of the DBMS_STATS package determines the degree of parallelism. The argument now has a new value called AUTO_DEGREE, which means that Oracle will automatically determine the degree of parallelism. It could be 1, which implies serial, not parallel execution, or DEFAULT_DEGREE, which is system default degree of parallelism, based on the initialization parameters pertaining to parallelism. Oracle recommends that you let it select the degree of parallelism based on size of the object, number of CPUs, and cetain parallelism-related initialization parameters.

Dynamic Sampling

You may, on occasion, encounter situations where you have database objects whose statistics are suspect or are missing altogether. In cases like this, the database will calculate if it is advantageous to estimate statistics using a small sample of the object's data. This feature is called *dynamic sampling*, and it has been available since Oracle9*i*. If you execute a query numerous times or if the query is very time-consuming, dynamic sampling may benefit that SQL statement, by yielding superior execution plans.

If you want to use automatic dynamic sampling in your database, set the OPTIMIZER_DYNAMIC_SAMPLING initialization parameter to at least 2 (the default value). If you set the level to zero, Oracle won't perform dynamic sampling. If you set it too high, you'll be imposing an unacceptable burden on your database.

Table Monitoring

If you use either the GATHER AUTO or STALE settings when you use the DBMS_STATS package, you don't need to explicitly enable table monitoring in Oracle Database 10*g*; Table monitoring tracks the number of DML changes to a table since the last statistics collection. In previous versions, you had to specify the MONITORING keyword in the CREATE (or ALTER) TABLE statement in order to enable the DBMS_STATS package to automatically gather statistics for a table. In Oracle Database 10*g*, the MONITORING and NO MONITORING keywords are deprecated. Oracle uses the DML change information logged in the DBA_TAB_MODIFICATIONS view to determine which objects have stale statistics. Just make sure that you set the STATISTICS_LEVEL parameter to either the default TYPICAL or ALL setting.

exam Watch

If you use the GATHER_DATABASE_STATS or GATHER_DATABASE_STATS procedure of the DBMS_STATS package with OPTIONS set to GATHER AUTO, you'll be analyzing only those database objects that have changed enough to warrant fresh statistics. Make sure you haven't set the STATISTICS_LEVEL parameter to BASIC, because this turns off the default table monitoring feature.

Statistics Collection for Dictionary Objects

Oracle Database 10*g* allows you to collect optimizer statistics on data dictionary tables to enhance performance. Oracle has two basic types of dictionary objects (dictionary tables): *fixed* and *real*. All dynamic performance tables are fixed tables, since you can't alter or remove them. The real dictionary tables belong to the SYS, SYSTEM, and other schemas that own the various Oracle components.

Collecting Statistics for Fixed Objects

Oracle recommends that you gather statistics for fixed objects only once for every database workload. Typically, a workload is a week for most OLTP databases. You can gather fixed object statistics by using the GATHER_DATABASE_STATS procedure and setting the GATHER_FIXED argument to TRUE (the default is FALSE).

You can also gather statistics for all fixed objects by using the new GATHER_FIXED_OBJECTS_STATS procedure of the DBMS_STATS package, as shown here:

```
SQL> sho user
USER is "SYS"
SQL> exec dbms_stats.gather_fixed_objects_stats('ALL');
```

on the Job

You must have the SYSDBA or ANALYZE ANY DICTIONARY system privilege to analyze any dictionary objects or fixed objects.

If you wish, you can collect statistics for an individual fixed table by using the standard DBMS_STATS procedures that enable table-level statistics collection. Then you can collect statistics for a fixed table just as would for any regular database table.

Collecting Statistics for Other Dictionary Objects

You can collect statistics for the real dictionary tables by using one of the following methods:

- Use the `DBMS_STATS.GATHER_DATABASE_STATS` (or `GATHER_SCHEMA_STATS`) procedure, by setting the `GATHER_SYS` argument to `TRUE`. Alternatively, you can use the `GATHER_SCHEMA_STATS ('SYS')` option.
- Use the `DBMS_STATS.GATHER_DICTIONARY_STATS` procedure.

CERTIFICATION OBJECTIVE 5.02

Using the SQL Tuning Advisor

Let's say you ask the ADDM to look into some performance problem, and you finally find out what ails your database: not surprisingly, poor SQL statements are slowing down the database again. What do you do? Normally, this is where your troubles begin, because you now embark on a journey that could be highly frustrating and time-consuming. Oracle Database 10*g* has a much better option for you. When you need to fix bad SQL, just turn the new SQL Tuning Advisor loose. The advisor will tell you the following:

- How to improve the execution plan
- Why it recommends a certain fix
- Estimated benefits if you follow its advice
- How to implement its recommendations (this may be the best part, as the advisor even gives you the actual commands necessary to tune the offending SQL statements)

Providing SQL Statements to the SQL Tuning Advisor

Where does the SQL Tuning Advisor get the highly resource-intensive SQL statements to conduct its analysis? You can feed the advisor SQL statements that you may gather from the following sources:

- You may create a new SQL statement or set of statements as an input for the SQL Tuning Advisor. Especially if you are working with a development database, this may be your best source of questionable SQL statements.

- The ADDM may often recommend high-load statements.
- You can choose a SQL statement that's stored in the AWR.
- You may choose a SQL statement from the database cursor cache.

If you have a set of SQL statements that you want the advisor to tune, you can create a SQL Tuning Set (STS), which is a set of multiple SQL statements, along with their execution information like the average elapsed time and bind values. Isolated SQL statements sometimes fail to capture the characteristics of your database workload realistically. An STS not only captures a database workload's information effectively, but also simplifies the tuning of several large SQL statements at once. Note that you can also use one or more STSs as the basis for a new STS.

How the SQL Tuning Advisor Works

Once you hand the SQL Tuning Advisor a SQL statement or an STS to analyze, the advisor invokes the Oracle optimizer in a new mode, called the *tuning mode*. As you recall, the optimizer always tries to find the best execution plan for a statement. Unfortunately, since it needs to do this live, in production conditions, it can't take more than an extremely short period to devise its solution. Thus, the optimizer resorts to heuristics and other similar techniques to come up with its best estimate of a good plan. This is the *normal mode* of the optimizer, where it quickly generates optimal execution plans for SQL statements.

What if you give the optimizer enough time to conduct a full-blown analysis of access paths, object characteristics, and so on?

In Oracle Database 10*g*, you can invoke the optimizer in the new tuning mode, wherein the optimizer conducts an in-depth analysis to improve its execution plans. Instead of mere seconds, in tuning mode, the optimizer may take several minutes to come up with *a set of recommendations*, rather than an optimal SQL execution plan. These recommendations are intended to help you optimize the execution plan, along with the rationale for them and the expected benefit if you follow the recommendations.

Since it sometimes takes several minutes for the optimizer to work through its analysis in the tuning mode, use it for fixing only your highly resource-intensive SQL statements.

When you use the Oracle optimizer in the tuning mode, it's referred to as the Automatic Tuning Optimizer (ATO). In this mode, you aren't really using the optimizer to develop quick execution plans. Your goal is to see if the ATO can improve on the usual optimizer execution plans. The ATO lies at the heart of the

SQL Tuning Advisor's capability to tune SQL statements for you. Through the advisor, you can now use the Oracle optimizer in the tuning mode (the ATO) to improve SQL performance.

The ATO performs the following four tuning tasks:

- Statistics analysis
- SQL profiling
- Access path analysis
- SQL structure analysis

exam Watch

The ATO performs four types of analysis: statistics analysis, SQL profiling, access path analysis, and SQL structure analysis.

These tasks are described in the following sections, along with the types of recommendations that the SQL Tuning Advisor makes.

Statistics Analysis

As its first task, the ATO ensures that statistics exist for all the objects in the SQL statement and that those statistics are representative—that is, not stale. Accurate and up-to-date statistics are essential for generating efficient execution plans. When it finds any tables with missing or stale statistics, the ATO recommends collecting new statistics for them. The ATO will also collect auxiliary information to fill in the missing statistics. If an object's statistics are stale, it corrects them, using adjustment factors.

SQL Profiling

The ATO collects auxiliary information to improve the execution plans. The ATO's goal at this stage is to verify that its own estimates of factors like column selectivity and cardinality of database objects are valid. It may use the following three verification or validation methods to test to verify its initial estimates:

- **Dynamic data sampling** Using a sample of the data, the ATO can check its own estimates of various factors like cost and cardinality for the statement in question. The ATO may decide to apply correction factors to the data, if the sampling process reveals that its estimates are significantly off the mark.
- **Partial execution** The ATO may partially execute a SQL statement, so it can check if its original estimates are good enough. It compares the run-time performance with the expected performance (based on the execution plan).

This feature is a bold attempt to go beyond normal collection of object statistics. Object statistics aren't always enough; an optimizer may need more information to get the right plan. So the ATO partially runs statements to gather statement-execution statistics to use in addition to the object statistics. Therefore, what it is checking is not whether the estimates of column selectivity and other factors are correct, but whether a plan derived purely from inspection of those statistics is actually the best plan.

- **Past execution history statistics** The ATO may also use any existing history of the SQL statement's execution to determine appropriate settings for parameters like `OPTIMIZER_MODE`.

exam Watch

The ATO may use dynamic sampling, partial statement execution, or historical SQL execution statistics to validate its initial estimates of cost, selectivity, and cardinality for a SQL statement.

If there is sufficient auxiliary information from either the statistics analysis or the SQL profiling phase, the ATO builds a *SQL profile*. Although we say that the ATO will "build a SQL profile," in reality, it simply recommends that you create a SQL profile. A SQL profile is simply a set of auxiliary or supplementary information about a SQL statement.

Once you accept the recommendation of the SQL Tuning Advisor to create a SQL profile in the *tuning mode*, Oracle will store that SQL profile in the data dictionary. The Oracle optimizer, *even while running in its normal mode*, will use it, along with the statistics that you've already collected, to produce better execution plans. Whenever you use the same SQL statement in the future, Oracle will automatically apply the SQL profile that you've created for it.

exam Watch

A SQL profile isn't the same as a stored execution plan.

As long as you have minor changes in table and index data and normal growth of objects, the SQL profile continues to be relevant. Over time, you may want to refresh or replace the SQL profile for a statement by running the ATO again.

on the Job

One of the biggest advantages of SQL profiles is that they provide you an effective way to tune "code that you can't touch." Packaged applications are usually hard to tune for DBAs, since there are limitations on accessing and modifying code. Since the SQL profiles are saved in the data dictionary, you can use them to tune even packaged applications.

Access Path Analysis

The ATO analyzes the potential impact of using improved access methods, such as additional or different indexes. As you know, addition of a critical index can speed up a query substantially. But what if the new index you create affects other SQL statements adversely? The SQL Advisor is aware of this possibility, and thus makes its index-related recommendations in the following way:

- If an index is very effective, it may recommend creating it immediately.
- It may recommend running the SQL Access Advisor (described later in this chapter) to make a comprehensive impact analysis of the addition of the new index.

exam Watch

The ATO will identify badly constructed SQL statements, but it doesn't automatically rewrite the queries for you. In the final analysis, you know your application better than the optimizer, and therefore Oracle only provides advice.

SQL Structure Analysis

The ATO may also make recommendations to modify the structure, both the syntax and semantics, in your SQL statements. Oracle will look at typical problems like the following:

- Design mistakes, such as a Cartesian product
- Use of inefficient SQL constructs; for example, the `NOT IN` construct is known to be very slow compared to the `NOT EXISTS` construct

SQL Tuning Advisor Recommendations

The SQL Tuning Advisor can recommend that you do the following:

- Create indexes to speed up access paths
- Accept a SQL profile, so you can generate a better execution plan
- Gather optimizer statistics for objects with no or stale statistics
- Rewrite queries based on the advisor's advice

Using the SQL Tuning Advisor

You can use the SQL Tuning Advisor with the help of Oracle database packages or with the OEM Database Control interface. The following sections describe both techniques.

Using the DBMS_SQLTUNE Package

The DBMS_SQLTUNE package is the main Oracle Database 10g interface to tune SQL statements. Let's first look at how you can use the package to create and manage SQL statement tuning tasks.

on the job

To use the DBMS_SQLTUNE package, you must have the ADVISOR privilege.

Performing Automatic SQL Tuning Following are the main steps in using the DBMS_SQLTUNE package to tune SQL statements.

1. **Create a task.** You can use the CREATE_TUNING_TASK procedure to create a task to tune either a single statement or several statements.
2. **Execute the task.** You start the tuning process by running the EXECUTE_TUNING_TASK procedure.
3. **Get the tuning report.** By using the REPORT_TUNING_TASK procedure, you can view the results of the SQL tuning process.

EXERCISE 5-1

Use the DBMS_SQLTUNE Package

Use the DBMS_SQLTUNE package to create a simple tuning task and view the tuning report. Your exercise can use the following structure:

```
SQL> declare
  2   tname varchar2(30);
  3 begin
  4  tname:=
  5  dbms_sqltune.create_tuning_task(
  6  sql_text=>'select count(*) from hr.employees,hr.dept');
  7 end;
  8  /
Check to make sure that your task was created by using the following query:
SQL> select task_name from user_advisor_log;
Get a task report by using the following statement:
SQL> select dbms_sqltune.report_tuning_task('TASK_NAME') from dual;
```

Managing SQL Profiles Use the `DBMS_SQLTUNE.ACCEPT_SQL_PROFILE` procedure to create a SQL profile based on the recommendations of the ATO.

on the Job

You must have the CREATE_ANY_PROFILE privilege in order to create the SQL profile.

Managing SQL Tuning Categories Suppose you have several profiles, all somewhat different, for a SQL statement. How does Oracle know which profile to use in a given case? All SQL profiles that you create for various SQL statements belong to specific SQL tuning categories. When a user logs in to the database, Oracle assigns each user to a specific tuning category, based on the value of the initialization parameter `SQLTUNE_CATEGORY`. Thus, the category name qualifies the lookup of SQL profiles by Oracle during the execution of a SQL statement.

The default value of the `SQLTUNE_CATEGORY` parameter is `DEFAULT`. Thus, any SQL profiles that belong to the default category will apply to all users who log in to the database. After the user logs in, you can change the SQL tuning category for all the users by using an `ALTER SYSTEM` command, or you can change a session's tuning category by using an `ALTER SESSION` command. For example, suppose that you have already created the categories PROD and DEV. You can change the SQL tuning category for all users with the following command:

```
SQL> alter system set SQLTUNE_CATEGORY = PROD;
```

To change a session's tuning category, use the following command:

```
SQL> alter session set SQLTUNE_CATEGORY = DEV;
```

Oracle will now apply all profiles under the category DEV, until you log out from that session. When you log in again, the default tuning category will again determine the profiles available to your session.

exam Watch

The ATO will build a SQL profile only if it generates auxiliary information during the statistics analysis and SQL profiling steps. If it builds a SQL profile, it will recommend that you actually create a SQL profile. Once you create a new SQL profile, Oracle automatically applies that profile to the SQL statement when you next execute it.

on the job

A session can change its SQL tuning category by using the `ALTER SESSION` statement shown in this section. You may also use the `DBMS_SQLTUNE.ALTER_SQL_PROFILE` procedure to change the SQL tuning category.

Using the Database Control to Run the SQL Tuning Advisor

To manage the SQL Tuning Advisor from the Database Control, click the Advisor Central link under the Related Links group, and then click the SQL Tuning Advisor link. You will see the main SQL Tuning Advisor page, where you can select the source for the SQL statements that you want the SQL Advisor to analyze. You have a choice of two main kinds of SQL statements:

exam Watch

There are several possible sources for the tuning advisor's STS input, including high-load SQL statements identified by the ADDM, statements in the cursor cache, statements from the AWR, a custom workload, or another STS.

- **Top SQL** These SQL statements could be current top SQL statements from the cursor cache or saved high-load SQL statements from the AWR.
- **SQL Tuning sets** You can create an STS from a set of SQL statements that you provide. They could be brand-new statements, or you could get them from AWR snapshots or baselines.

Once you click any of the four links, you'll be taken to the data source you selected. From there, you can launch the SQL Tuning Advisor. Follow the instructions to view the advisor report and analyze the recommendations.

CERTIFICATION OBJECTIVE 5.03

Using the SQL Access Advisor

The SQL Access Advisor is another useful component of the Advisory Framework. The SQL Access Advisor primarily provides advice regarding the creation of indexes, materialized views, and materialized view logs, in order to improve query performance. The advisor recommends both bitmap indexes and B-Tree indexes. It also recommends the optimization of materialized views so you can "fast refresh" them and thus take advantage of general query rewriting.

Providing Input for the SQL Access Advisor

As in the case of the SQL Tuning Advisor, you can provide the input for the SQL Access Advisor's analysis from the SQL cache, the AWR, or new SQL statements you've created for testing purposes. No matter the source of the SQL workload, the SQL Access Advisor can recommend indexes and materialized views that can improve the performance of the entire workload. Often, a well-written SQL statement may perform poorly due to the lack of the right index or a materialized view. The advisor will suggest appropriate indexes and view and provides you the rationale for those recommendations. The advisor can take into account multiple combinations of actions to arrive at the best tuning strategy.

Here are the four main sources of a SQL Access Advisor's workload:

- Current and recent SQL activity, which includes statements from the SQL cache (from the `V$SQL` view)
- An STS stored in the AWR
- A user-defined workload, which enables you to test a workload before an application goes into production
- An hypothetical workload you've created from specific schemas and tables

exam Watch

The three types of database objects that the SQL Access Advisor may recommend that you create, drop, or retain are indexes, materialized views, and materialized view logs. There are four main sources of input for the advisor: SQL cache, user-defined workload, hypothetical workload, and STS from the AWR.

How the SQL Access Advisor Works

The goal of the SQL Access Advisor is to reduce the processing time of SQL statements, by creating additional structures like indexes and/or materialized views. You can choose to have the advisor recommend just indexes, just materialized views, or both.

The SQL Access Advisor takes into account user-defined constraints by supporting the following:

- Storage constraints
- Refresh time constraints
- Full or partial workloads

Modes of Operation

You can operate the SQL Access Advisor in two modes:

- **Limited (partial)** In the limited mode, which is more or less a reactive mode of tuning, the advisor will concern itself with only problematic or high-cost SQL statements. In these situations, the advisor makes recommendations that affect only the current statement or statements. Limited mode returns quickly after processing the statements with the highest cost, potentially ignoring statements with a cost below a certain threshold.
- **Comprehensive (full)** Comprehensive analysis is more like a proactive type of analysis. It will perform a complete and exhaustive analysis of all SQL statements in a representative set of SQL statements, after considering the impact on the entire workload, not just a few SQL statements. For example, a particular index may help a certain query, but may be quite detrimental for the entire workload's point of view. Thus, a comprehensive analysis enables the advisor to provide better global tuning advice, but, of course, it takes a lot of time to complete.

You can also use *workload filters* to specify which kinds of SQL statements the SQL Access Advisor should select for analysis, from the workload it has in front of it. For example, you may use filter options like top resource using SQL statements, specific users or tables, and so on. The workload filters enable the reduction of the scope of the SQL statements in the workload. The advisor applies these filters to focus its tuning efforts, say, limiting its analysis to only those statements that touch a certain set of tables.

SQL Access Advisor Recommendations

Following are some of the specific types of recommendations made by the SQL Access Advisor:

- Add new indexes or materialized views.
- Add a new materialized view log.
- Modify indexes or materialized views by adding columns.
- Change an index type.
- Drop an unused index or materialized view.

Managing the SQL Access Advisor

You can manage the SQL Access Advisor either through the OEM Database Control or by using the DBMS_ADVISOR package. Both techniques are described in the following sections.

Using the DBMS_ADVISOR Package

Following is a brief summary of the steps involved in using the DBMS_ADVISOR package:

1. Create and manage a task, by using a SQL workload object and a SQL Access task.
2. Specify task parameters, including workload and access parameters.
3. Using the workload object, gather the workload.
4. Using the SQL workload object and the SQL Access task, analyze the data.

You can also use the QUICK_TUNE procedure of the DBMS_ADVISOR package to quickly analyze *a single SQL statement*, without needing to go through all these steps. You can create tasks, prepare workloads for the SQL statement, execute the task, save the results, and finally implement the recommendations as well. Here's a quick example of using the QUICK_TUNE procedure:

```
VARIABLE task_name VARCHAR2(255);
VARIABLE sql_stmt VARCHAR2(4000);
EXECUTE :sql_stmt := 'SELECT COUNT(*) FROM customers
                      WHERE cust_region=''TX''';
EXECUTE :task_name  := 'MY_QUICKTUNE_TASK';
EXECUTE DBMS_ADVISOR.QUICK_TUNE(DBMS_ADVISOR.SQLACCESS_ADVISOR, -
              :task_name, :sql_stmt);
```

exam Watch

Oracle creates the new indexes in the schema and tablespaces of the table on which they are created. If a user issues a query that leads to a recommendation to create a materialized view, Oracle creates the materialized view in that user's schema and tablespace.

Using the Database Control to Run the SQL Access Advisor

To manage the SQL Access Advisor from the Database Control, click the Advisor Central link under the Related Links group, and then click the SQL Access Advisor link. You will see the main SQL Access Advisor page.

First, choose the *access method*—materialized view or an index—that you would like the advisor to evaluate. You can use the Advanced Options section to specify the following options:

- **Space Restrictions** These options include setting space size limits that the advisor's recommendations can't exceed. You may also limit the number of indexes that the advisor can create.
- **Tuning Options** These options prioritize the advisor's access recommendations by items like buffer gets, CPU time, disk reads, and elapsed time.
- **Default Storage Locations** These options enable you to override the default locations defined for the schema and the tablespaces.

CERTIFICATION OBJECTIVE 5.04

Using the Performance Pages of the Database Control

As you've seen in prior chapters, the ADDM and the server-generated alerts provide you with proactive database management support. The OEM Database Control is an essential part of the push by Oracle to automate most of the DBA's job functions in Oracle Database 10*g*. Using the Database Control, you can analyze these important components of both the database as well as system performance.

In the following sections, we'll look at two Database Control pages that highlight the OEM performance management features: the Database Home page and the Performance page.

The Database Home Page

The Oracle Database Home page allows you to view the current state of the database by displaying various metrics that portray the overall health of the database. The Oracle Database Home page, shown in Figure 5-1, provides a launching point for performance tuning, as well as other management activities.

exam Watch

The three major tuning areas that the OEM Database Control will show you are CPU and wait classes, top SQL statements, and top sessions in the instance.

Here are some of the statistics the Database Home page shows you (in addition to the ADDM findings):

- CPU usage
- Database waits
- Top SQL statements

- Top user sessions
- SQL response time

The Database Performance Page

The Database Performance page shows you the overall status of the database and helps you quickly identify causes of performance bottlenecks. Figure 5-2 shows the three main items on the Performance page:

- Host
- Sessions waiting and working
- Instance throughput

Host Performance Data

The Host part of the page shows two important graphs dealing with resource use on the host:

- **Average Run Queue** This shows how hard the CPU is running. A high run queue indicates that there is contention for CPU time. The run queue tells you how many processes are ready to run but are unable to do so, due to a CPU bottleneck.
- **Paging Rate** This shows the rate at which the host server is writing memory pages to the swap area on disk. Paging slows down your system because of the extra I/O, and it indicates that your system is memory-bound.

Sessions Performance Data

The sessions graph shows which active sessions are on the CPU and which are waiting for resources like locks, disk I/O, and so on. By clicking specific boxes, you can drill down to any wait that seems serious.

on the job

Oracle recommends that you start investigating waits if the level of waits is at twice the Maximum CPU line in the sessions graph.

Instance Throughput Performance Data

If your instance throughput is decreasing, along with an increasing amount of contention within the database, you should start looking into tuning your database.

FIGURE 5-1 The Database Control Database Home page

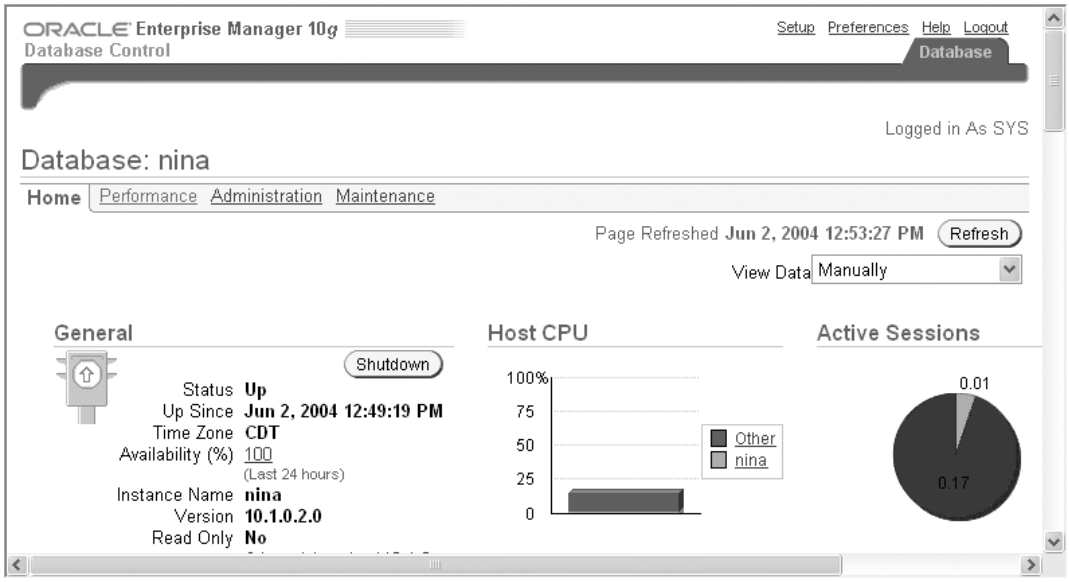
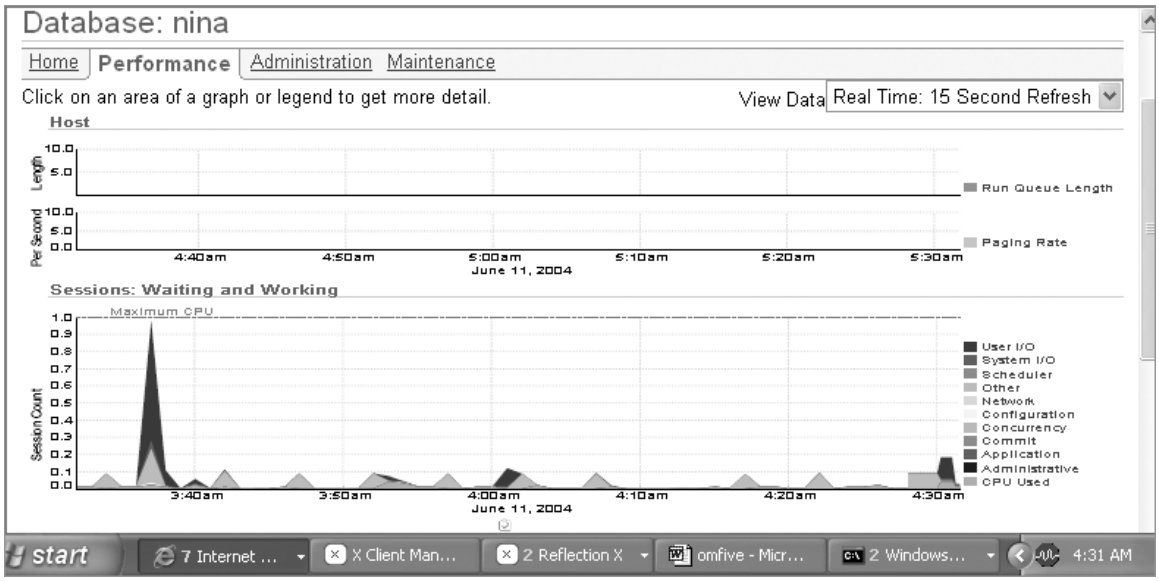


FIGURE 5-2 The Database Control Database Performance page



CERTIFICATION OBJECTIVE 5.05

Indexing Enhancements

There are several index-related enhancements in Oracle Database 10*g*. Primary among them are the following:

- Skipping unusable indexes
- Creating and maintaining hash-partitioned global indexes
- Using the new `UPDATE INDEXES` clause

Let's start our discussion in this section with a look at enhancements in the skipping of unusable indexes.

Skipping Unusable Indexes

In previous versions of Oracle, modification of any partitions of a table using the `SPLIT` or `MERGE` command, for example, left the indexes in an unusable state. In Oracle9*i*, the `SKIP_UNUSABLE_INDEXES=TRUE` parameter enabled you to avoid the indexes that were rendered unusable. In Oracle Database 10*g*, the default value of this parameter is `TRUE`. Thus, you can guarantee that the optimizer will always ignore the unusable indexes at parse time.

exam Watch

In Oracle Database 10g, the default value of `SKIP_UNUSABLE_INDEXES` parameter is `TRUE`.

When you use the `SKIP_UNUSABLE_INDEXES=TRUE` specification, the database may skip an index that it would have used otherwise, thus leading to suboptimal execution plans. That's why the database records this information regarding the presence of unusable indexes in the alert log.

on the Job

The `SKIP_UNUSABLE_INDEXES` parameter could be quite dangerous in practice. The Oracle Database 10g default means that your SQL will always run, but it may be running very badly because of broken indexes. With previous releases, you would always know about the problem immediately.

Using Hash-Partitioned Global Indexes

Until Oracle Database 10*g*, you could create only range-partitioned global indexes on tables. Now you can create hash-partitioned global indexes as well. As with the global range partitioning, you can use a maximum of 32 columns in the global index. Oracle's hash functions determine the values in each partition now, instead of a range of values in a range-partitioned global index. Oracle Database 10*g* assigns rows to the partitions using a hash function on values in the partitioning key columns.

Benefits of Hash-Partitioned Global Indexes

You can hash-partition indexes on tables, partitioned tables, and index-organized tables. This feature provides higher throughput for applications with large numbers of concurrent insertions. In some applications, new insertions into indexes are biased heavily toward the right side of the index. This is usually the case when you have an index on a column that is a monotonically increasing sequence number. You can't avoid heavy contention for the index resources in such a situation, in addition to causing potential node splitting. Since the partitioning key (and the number of partitions in the index) is the primary determinant of in which partition the index entries are placed, it isn't likely that a monotonically increasing sequence number would cause contention on one side of the index. Hash-partitioned global indexes thus improve concurrency and raise throughput in OLTP systems.

For monotonically increasing key situations, range partitioning will cause contention on the right edge of the highest range partition. Reverse keying the index will spread the activity, but only across the highest partition. Hash partitioning will distribute the workload across all the index partitions, but still with contention at each index's right edge; reverse-key hash partitioning will not only distribute the activity across all the partitions, but also spread it within each partition. Oracle also has more options available to it for parallel query and DML operations.

As with the hash-partitioned table, you can specify only the tablespace for the hash-partitioned indexes as part of the storage attribute.

One of the advantages of range-partitioned indexes is that you can perform partition pruning, which makes it easy to skip irrelevent partitions. However, hash-partitioned global indexes overcome this disadvantage by being able to access all partitions in the index in parallel. If you have queries with range predicates, for example, hash-partitioned indexes perform better than range-partitioned indexes.

Hash-Partition Management

When you create a hash-partitioned index, you can specify each partition its own tablespace, or you can specify the total number of partitions and use the STORE IN clause to allocate the hash partitions among the available number of tablespaces. If you skip the STORE IN clause, Oracle will place the hash partition in the user's default tablespace. All partitions in a hash-partitioned global index share identical physical and storage attributes.

You use the following statement for creating a hash partitioned global index, if you wish to specify each hash partition individually:

```
SQL> create index sales_hash
          on sales_items (sales_id) global
          partition by hash (sales_id) (
              partition p1 tablespace tbs_1,
              partition p2 tablespace tbs_2,
              partition p3 tablespace tbs_3
              partition p4 tablespace tbs_4);
```

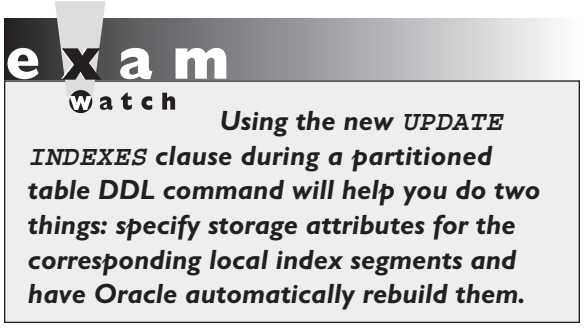
You can also create a hash partitioned global index by specifying the number of hash partitions (4 in our example) with the optional STORE IN clause, as shown here:

```
SQL> create index sales_hash
           on sales_items (sales_id) global
           partition by hash (sales_id)
              partitions 4
              store in (tbs_1,tbs_2,tbs_3,tbs_4);
```

You can't perform the following operations on hash-partitioned global indexes: ALTER INDEX REBUILD, ALTER TABLE SPLIT INDEX PARTITION, ALTER TABLE MERGE INDEX PARTITITON, and ALTER INDEX MODIFY PARTITION.

Using the New UPDATE INDEXES Clause

In previous versions of Oracle, when your performed DDL actions, such as adding, moving, or merging table partitions, you couldn't explicitly specify the storage characteristics for the associated local index. Oracle created the local indexes either in the default tablespace or in the same tablespaces as the underlying table partitions. The DDL on the table partitions also left the local indexes in an unusable state, forcing you to rebuild them.



In Oracle Database 10g, you can now specify storage attributes, as well as keep you local indexes from being rendered unusable, with the help of a single new clause. By using the UPDATE INDEXES clause during a partitioned table DDL statement, you can avoid making your local

indexes unusable by making Oracle automatically rebuild them, thus increasing the availability of your database. Here's an example of the usage of the UPDATE INDEXES clause during partitioned table DDL:

```
SQL> alter table MY_PARTS
  2  MOVE PARTITION my_part1 tablespace new_tbsp
  3  UPDATE INDEXES
  4   (my_parts_idx
  5 (PARTITION my_part1 TABLESPACE my_tbsp);
```

Note that if you don't specify the index name in the preceding statement, Oracle will rebuild the indexes to put them back into a usable state.

on the job

The MOVE operation is actually implemented as two separate DDL statements: first, Oracle moves the table, and then it rebuilds the index. So even though it is slow, you do end up with an efficient index.

INSIDE THE EXAM

You must be aware of the role of the GATHER_STATS_JOB Scheduler job as well as the proper settings of the STATISTICS_LEVEL parameter for optimizer statistics collection.

The certification exam tests you knowledge of the SQL Access Advisor and the SQL Tuning Advisor rather thoroughly. You must know the particular types of Oracle objects about which the SQL Access Advisor will make recommendations. You must remember all the possible sources of data for both of these advisors.

The exam focuses on the Automatic Tuning Optimizer (ATO). What is the difference between the regular Oracle optimizer and the ATO? You must understand *each type* of analysis the ATO performs as part of the SQL tuning process. Review all four types of analyses carefully, paying attention to the various kinds of verification methods in the SQL profiling phase of the analysis. Know how to create a SQL profile, as well as how Oracle uses it. When does the ATO recommend a SQL profile?

On the exam, don't be surprised to see a couple of questions about using the OEM Database Control to manage database performance. You must know the basic approach of the Database Control regarding performance management (CPU and waits, top SQL, and top sessions). You must practice using the Database Control to locate statistics about CPU and waits, SQL, and sessions. What's on the Database Control Database Home page? What are the three sections of the Database Performance page? How do you drill down to various contention areas? Where and what are the Top Waiting SQL and the Top Waiting Sessions charts? How do you drill down to the Top SQL by Waits chart?

Expect a question testing your knowledge of the SKIP_UNUSABLE_INDEXES and/or the UPDATE INDEXES clause, both concerning Oracle Database 10*g* enhancements to the partitioning option. Review the benefits of hash-partitioned global indexes.

CERTIFICATION SUMMARY

This chapter started with an overview of the enhancements and changes in the optimizer statistics collection in Oracle Database 10*g*. You learned about the automatic statistics collection process. The SQL Tuning Advisor takes away the drudgery involved in tuning complex SQL code, and you learned how to manage this advisor using PL/SQL packages as well as the Database Control.

You learned about the SQL Access Advisor and how you can get advice about proper indexes and materialized views in your database.

You saw how you could use the Database Control's Performance page to drill down into any performance issue of interest. You learned about the innovations in the skip unusable indexes feature. You saw how you could specify storage characteristics for index partitions in the course of DDL statements. You were introduced to the new hash-partitioned global indexes, which can reduce index contention in some special circumstances.

TWO-MINUTE DRILL

Using the New Optimizer Statistics

- Rule-based optimization is now obsolete.
- The `CHOOSE` and `RULE` values for the `OPTIMIZER_MODE` parameter are not supported in Oracle Database 10*g*, although you can still continue to use them.
- The default value for the `OPTIMIZER_MODE` parameter is `ALL_ROWS`.
- Oracle prioritizes its automatic statistics collection, with preference given to objects with stale or no statistics.
- The default cost model for the optimizer is CPU+I/0.
- The default value for the `GRANULARITY` argument is `AUTO`.
- If you specify `GLOBAL` and `PARTITION` options, Oracle doesn't collect subpartition-level statistics.
- The `DEFAULT` setting for the `DEGREE` argument means that you'll use a system-based degree of parallelism.
- Use the `OPTIMIZER_DYNAMIC_SAMPLING` parameter to use automatic dynamic sampling.
- Table monitoring is automatic in Oracle Database 10*g*.
- Oracle recommends that you collect optimizer statistics for data dictionary tables, both fixed and real.

Using the SQL Tuning Advisor

- You can use the AWR data or the ADDM recommendations, or create your own SQL statements to input to the SQL Tuning Advisor.
- A SQL Tuning Set (STS) is a set of SQL statements that you would like to tune together.
- There are two modes for the optimizer now: normal mode and tuning mode.

- Use the optimizer in the tuning mode only for highly resource-intensive statements.
- The fours steps of a SQL Tuning Advisor job are statistics analysis, SQL profiling, access path analysis, and SQL structure analysis.
- You can manage the SQL Tuning Advisor by using the `DBMS_SQLTUNE` package.
- You can create a new profile by using the `ACCEPT_SQL_PROFILE` procedure.
- All SQL profiles belong to a specific SQL tuning category.
- The default value of the `SQLTUNE_CATEGORY` parameter is `DEFAULT`.

Using the SQL Access Advisor

- The SQL Access Advisor provides advice on creating indexes, materialized views, and materialized view logs.
- The SQL Access Advisor may also recommend the dropping of unused indexes.
- You can use the SQL Access Advisor in a limited or comprehensive mode.
- You can manage the SQL Access Advisor through the `DBMS_ADVISOR` package.
- You can use the `QUICK_TUNE` procedure of the `DBMS_ADVISOR` package to analyze a single SQL statement.

Using the Database Control for Performance Tuning

- Using the Database Control, you can monitor CPU, waits, top SQL, and top sessions.
- The three main sections on the Database Performance page are Host, Sessions Waiting and Working, and Instance Throughput.
- The Host section shows the average run queue and the paging rate.
- The session graphs show which sessions are on CPU and which sessions are waiting for resources.
- Oracle recommends investigating waits if the level of waits is at twice the Maximum CPU line in the sessions graph.

Indexing Enhancements

- ❑ The default value of `SKIP_UNUSABLE_INDEXES` is `TRUE`.
- ❑ You can now use hash-partitioned global indexes.
- ❑ Oracle assigns rows to various partitions using a hash function on the partitioning key column.
- ❑ Hash-partitioned global indexes increase concurrency, especially if the index is on monotonically increasing sequence numbers.
- ❑ You can't perform certain index maintenance operations on hash-partitioned global indexes.
- ❑ You can use the `UPDATE_INDEXES` clause to set storage attributes as well as to keep local indexes from being made unusable during a DDL operation.

SELF TEST

The following questions will help you measure your understanding of the material presented in this chapter. Read all the choices carefully, because there might be more than one correct answer. Choose all correct answers for each question.

Using the New Optimizer Statistics

1. What is the default value for the `OPTIMIZER_MODE` initialization parameter?
   - A. `FIRST_ROWS`
   - B. `ALL_ROWS`
   - C. `CHOOSE`
   - D. `COST`
2. If you are using `LIST` subpartitioning and you specify the value of the `GRANULARITY` argument as `AUTO`, what statistics will Oracle collect?
   - A. Global statistics only
   - B. Global and partition statistics only
   - C. Global, partition, and subpartition statistics
   - D. No partitioned table statistics
3. How can you collect statistics for fixed data dictionary tables?
   - A. By using the `DBMS_STATS.GATHER_DICTIONARY_STATS` procedure
   - B. By using the `DBMS_STATS.GATHER_FIXED_OBJECTS_STATS` procedure
   - C. By using `GATHER_SCHEMA_STATS` with the `SYS` option
   - D. By using the statement `EXEC DBMS_STATS.GATHER_FIXED_OBJECTS_STATS('USER');`

Using the SQL Tuning Advisor

4. What can you create a SQL Tuning Set (STS) from?
   - A. Several SQLs statements from the AWR
   - B. One or more other STSs
   - C. A single SQL statement
   - D. Only the most high-load SQL statements in your database

5. Which of the following is true about the SQL Tuning Advisor?
   A. It automatically collects statistics if they are missing.
   B. It only makes recommendations to collect statistics.
   C. It ignores optimizer statistics completely.
   D. It uses only the available statistics and nothing else.
6. If you create a SQL profile, when can you use it?
   A. In the tuning mode of the optimizer only
   B. In the normal mode of the optimizer
   C. In neither the tuning nor the normal mode of the optimizer
   D. Along with existing statistics, to get better SQL execution plans
7. How do you create a new SQL profile?
   A. With the `ACCEPT_SQL_PROFILE` procedure
   B. With the `CREATE_SQL_PROFILE` procedure
   C. With both the `ACCEPT_SQL_PROFILE and CREATE_SQL_PROFILE` procedures
   D. A SQL profile is automatically created each time you execute a new SQL statement.

Using the SQL Access Advisor

8. Which of the following is true about the SQL Access Advisor?
   A. It will not recommend the creation of an index until it completes a full, comprehensive analysis.
   B. It can recommend the dropping of an index during a limited mode of operation.
   C. It can recommend the dropping of an index during a comprehensive mode of operation.
   D. It can change an index type under certain conditions.
9. How can you manage the SQL Access Advisor?
   A. Through the `DBMS_SQLTUNE` package
   B. Through the `DBMS_ADVISOR` package
   C. Through the `SQL_TUNE` package
   D. Through the `DBA_ADVISOR` package

Using the Database Control for Performance Tuning

10. Oracle recommends that you start investigating database waits when what occurs?

A. The level of waits is the same as the Maximum CPU line in the sessions graph.
B. The level of waits is smaller than the Maximum CPU.
C. The level of waits is at twice the Maximum CPU line.
D. The level of waits is slightly higher than the Maximum CPU line.

11. A high paging rate indicates that your system is what?

A. CPU-bound
B. Memory-bound
C. I/O-bound
D. SGA-bound

12. Which three major tuning areas does Database Control provide information about?

A. CPU and wait classes
B. Most-used objects
C. Top SQL statements
D. Memory usage
E. Top sessions affecting the instance

13. What does the Database Control Performance Page show?

A. Host information
B. Memory usage
C. CPU usage
D. User activity
E. Throughput information

Indexing Enhancements

14. When you have a monotonically increasing sequence number as an indexed column, which of the following is true?

A. Hash-partitioned global indexes are useful.
B. Range-partitioned global indexes are better.
C. Hash-partitioned local indexes are better.
D. Range-partitioned local indexes are better.

15. What is the benefit of using the `UPDATE INDEXES` clause in an `ALTER TABLE` statement?

A. You can keep your indexes form being rendered unusable.
B. You can specify storage attributes.
C. You can update statistics on your indexes.
D. You can keep your index nodes from splitting excessively.

LAB QUESTION

Use the `DBMS_SQLTUNE` package to do the following:

- Create a tuning task.
- Execute the tuning task.
- View the tuning results.
- Create a SQL profile, if one is recommended by the SQL Tuning Advisor.

SELF TEST ANSWERS

Using the New Optimizer Statistics

1. ☑ **B.** The default optimizer mode in Oracle Database 10*g* is `ALL_ROWS`.
   ☒ **A, C,** and **D** are not the correct default optimizer alternatives in Oracle Database 10*g*.

2. ☑ **C.** If you're using the `LIST` partitioning method, Oracle will collect all statistics, down to the lowest level of granularity—the subpartition.
   ☒ **A, B,** and **D** point to the wrong alternatives.

3. ☑ **B.** The `GATHER_FIXED_OBJECTS_STATS` procedure lets you collect statistics for the fixed data dictionary tables.
   ☒ **A** is wrong because the `GATHER_DICTIONARY_STATS` procedure collects only the nonfixed data dictionary tables. **C** is wrong as well, for the same reason as answer A. **D** seems correct, but you must specify `ALL`, not `USER`, in order to gather statistics for all the fixed tables.

Using the SQL Tuning Advisor

4. ☑ **A** and **B.** An STS is created from any set of SQL statements, including those you get from the AWR. **B** is correct because you can create a STS from either a set of individual SQL statements or from other STSs.
   ☒ **C** is wrong because a single SQL statement isn't enough to create an STS. **D** is wrong because there is no requirement that only high-load SQL statements can be included in an STS.

5. ☑ **B.** The SQL Tuning Advisor only makes recommendations to collect statistics. It might perform sampling or collect auxiliary information to help derive a SQL profile, but it doesn't automatically start the statistics collection process.
   ☒ **A** is wrong because the advisor only recommends that you collect statistics. **C** is wrong because the advisor never ignores statistics. **D** is wrong as well, since the advisor does use quite a bit of information in addition to pure optimizer statistics.

6. ☑ **B** and **D. B** is correct since you can use the SQL profiles only in the normal mode of the optimizer. **D** is also correct since Oracle uses the SQL profiles along with the available statistics.
   ☒ **A** is wrong because you don't use the SQL profile in the tuning mode only. **C** is wrong because you can use the SQL profiles in the normal mode by the optimizer.

7. ☑ **A.** When the SQL Tuning Advisor recommends a SQL profile, you can create the profile by using the `ACCEPT_SQL_PROFILE` procedure.
   ☒ **B** is wrong because there is no such procedure. **C** is wrong because of the same reasons as B. **D** is wrong since Oracle doesn't automatically create a SQL Profile.

Using the SQL Access Advisor

8. ☑ **C** and **D. C** is correct because the SQL Access Advisor can recommend dropping an index only if you use it in the full (comprehensive) mode. **D** is correct because the advisor recommends the changing of the index type under some conditions.
☒ **A** is wrong because you don't need a full analysis for the advisor to recommend indexes. **B** is wrong because a limited mode of operation cannot recommend dropping of indexes and materialized views.

9. ☑ **B.** You can manage the SQL Access Advisor by using the `DBMS_ADVISOR` package.
☒ **A, C,** and **D** point to wrong or nonexistent packages.

Using the Database Control for Performance Tuning

10. ☑ **C.** Oracle recommends that you investigate database waits if the level of waits in the Database Control graphs is at twice the Maximum CPU line.
☒ **A, B,** and **D** specify the wrong information.

11. ☑ **B.** Paging is related to memory.
☒ **A, C,** and **D** are wrong because they really don't have a direct bearing on paging rates.

12. ☑ **A, C,** and **E.** The three major tuning areas that the Database Control focuses on as part of its performance management approach are CPU and wait classes, top SQL statements, and top sessions affecting instance.
☒ **B** is wrong since the Database Control doesn't track object usage. **D** is wrong since memory usage isn't a major tuning area in the Database Control performance management approach.

13. ☑ **A, D,** and **E.** The Database Control Performance Page contains three sections displaying host information, user activity, and throughput information.
☒ **B** and **C** are wrong since memory and CPU usage information aren't shown on the Performance Page.

Indexing Enhancements

14. ☑ **A.** When you have monotonically increasing indexed column, hash-partitioned global indexes are ideal, because they spread the column values more or less uniformly among the partitions.
☒ **B** is wrong since range-partitioned global indexes are the worst choice for this type of indexed columns. **C** and **D** refer to local indexes, which aren't really better than the global indexes in this case.

15. ☑ **A** and **B.** By using the `UPDATE INDEXES` clause, you can specify storage attributes as well as keep your indexes from being rendered unusable.
☒ **C** and **D** have nothing to do with the `UPDATE INDEXES` clause.

LAB ANSWER

First, create three bind variables, to hold your query string and the task name, as shown here:

```
SQL> variable test_query varchar2(1000)
SQL> variable test_task varchar2(1000)
SQL> variable test_profile varchar2(1000)
```

Now insert your query into the bind variable `test_query`.

```
SQL> begin
  2   :test_query:='SELECT CHANNEL_ID FROM SH.SALES WHERE PROD_ID=146;';
  3  end;
  4  /
PL/SQL procedure successfully completed.
SQL> begin
      :test_task := DBMS_SQLTUNE.create_tuning_task (
      sql_text  =>     :test_query,
      bind_list =>     SQL_BINDS(anydata.ConvertNumber(100)),
      user_name =>     'SAM',
      scope           =>      'COMPREHENSIVE',
      time_limit       =>     60,
      task_name =>     'test_tuning_task',
      description       =>     'Query on SALES table');
     end;
         /
```

Execute the task using the `EXECUTE_TUNING_TASK` procedure, as follows:

```
SQL> begin
     dbms_sqltune.execute_tuning_task (TASK_NAME=>:test_task);
     end;
View the tuning report, using the REPORT_TUNING_TASK function, as follows:
SQL> select dbms_sqltune.report_tuning_task
     task_name=>:test_task)
     from dual;
```

Create the SQL profile, if one is recommended by the SQL Tuning Advisor, by using the `ACCEPT_SQL_PROFILE` function, as shown here:

```
SQL> begin
     :test_profile := dbms_sqltune.accept_sql_profile
     (task_name       =>     'test_tuning_task');
     end;
     /
```