



CHAPTER 4

Database Tuning: Making It Sing

146 Everyday Oracle DBA



f backup and recovery are the most important things a DBA does, then tuning is the runner-up. Oracle's database engine is a highly tunable creature and, metaphorically speaking, you *can* make it sing by monitoring how Oracle performs as it runs, and then adjusting different parameters, increasing (hopefully) its performance.

It's often the case that the time spent waiting for various computing functions to finish adversely impacts both a company's expenses and its man hours. Time is wasted when users have to wait (sometimes for extended periods of time) for queries to be returned. Sometimes it's imperative your system keeps pace with the speed and ever-increasing needs of the business community, or perhaps you need to optimize use of your existing hardware infrastructure particularly at a time when organizations insist on doing more work, even with less capital invested in hardware. In the words of Gaja Krishna Vaidyanatha (co-author of *Oracle Performance Tuning 101* and owner of DB Performance Management Consulting), you may be one of the unfortunate souls suffering from "compulsive tuning disorder," who spends an inordinate amount of time tuning the database by looking at irrelevant things.

Whatever your reason for tuning, however, and regardless of the approach you take, the fact is you'll often be called upon to spend time in the tuning arena.

While there are widely differing schools of thought concerning tuning methodology, what to tune typically falls into six major categories.

Database Design

Optimally, if you can (that is to say if you're lucky enough to have a say in the design process), the biggest bang for the tuning buck is typically at database design time. Knowing the design, being able to put structures in place from the get-go, and normalizing the design to the extent it is practical (even third normal form is too normal for some databases) will go a long way when it comes to tuning. Understanding how users will use the data also helps, and not being afraid of employing a lot of Oracle's new features (new as of Oracle 8, Oracle 8i, Oracle 9i, or even Oracle 10g) even if they are scary new features like materialized views, dimensions, and partitioning

Chapter 4: Database Tuning: Making It Sing 147

can also benefit a design, either from the planning stage or even later, after the database is in place. Also, a word to the wise: work with your system administrators to lay the files out on disk as optimally as possible. They know the disks that will perform better, if your organization segregates job responsibilities between system administration and database administration, and you know the tables and tablespaces that are likely to be more active. Many that don't know end up putting the most active data on the least well performing disks.

Application Tuning

If you can't tune the database design, the next best option is to tune the application and the application code. In many cases, the 80-20 rule applies. Eighty percent of all performance problems can be resolved with coding more optimal SQL or appropriately scheduling batch jobs during off-peak hours. Of course, if you're in a global organization, finding an off-peak time may be next to impossible, but it's still worth a try. The majority of this chapter will cover this kind of tuning.

Memory Tuning

Properly sizing the SGA, your database buffers, and pools can go a long way towards maintaining an efficiently running database. Having sufficient space to allow you to pin objects into memory, in particular those frequently used on large objects, can limit the amount of disk access needed. Of course, it's difficult by any stretch of the imagination to justify pinning a few billion-row tables into memory, even if it were possible, but in this, as with all things, moderation is the key.

Disk I/O Tuning

The proper placing of datafiles and aptly sizing them to provide the maximum throughput of data from disk to the application can be an important step in tuning. Again, placing active files on controllers with the best throughput can mean your users won't notice the application slowing down. The best throughput possible can mean your application won't be noticeably slowed down.

148 Everyday Oracle DBA

Database Contention

To really tune a database and make it sing well, it is important to have the best understanding possible of not only all of the components of the database, but of the way they operate together and interact. Two, often misunderstood, components are locks and latches.

In order for different users to share and access resources concurrently (and remember, as far as the database is concerned, a process is nothing more than a user) and serialize access to the SGA, and in order to protect the contents of the database while they are being modified and accessed by users and processes, Oracle employs the use of locks and latches.

Latches provide exclusive access to protect the data structures. With latches, requests are not queued; if a request fails, it fails, but it may try again later. They are simple, very efficient data structures whose purpose is to protect resources that are briefly needed. Latches can be requested in one of two modes: the patient, willing-to-wait mode that says “It’s okay if you aren’t available right now, I will just sit around out here waiting for whatever resource to become available and then try again” or the immediate or no-wait mode (ever have to deal with a toddler or a teenager?) that says “NOW! I want it now! Give it to me now!” Sometimes, when more than one latch is requested on a resource latch, contention can occur and this can affect performance significantly if enough latches are not available or when a latch is being held for a relatively long time. Latch contention can be resolved by upping the `init.ora` parameters.

Locks provide serialized access to some resources. Requests that fail are queued and are processed and serviced in order. These are more complex and less efficient data structures that are often protected themselves by latches and which protect resources (like tables) that are needed for a longer time. Locks allow sessions to join a queue for resources that are not readily available and are used to achieve consistency and data integrity. Locks are either invoked automatically by Oracle or can be invoked manually by users.

When it comes to database contention, watch locks and latches. Pay attention to wait events—look at them closely and eliminate as many of those that are avoidable as you can.

`V$SESSION_WAIT` dynamic view can be monitored for latch-free waits (P1 parameters tell you the SGA address and correspond to the `V$LATCH_PARENT` and `V$LATCH_CHILDREN` views, P2 parameters tell you the type of latch, and P3 parameters tell you the number of times that the process that is trying to acquire the latch has had to sleep).

Chapter 4: Database Tuning: Making It Sing 149

V\$RESOURCE provides you a view into locked resources that are causing lock queues to form. V\$ENQUEUE_LOCK tells you the enqueue state caused by the locks. All locks that are held by Oracle or that are outstanding requests for locks can be found in the V\$LOCK view.

Excessive numbers in any of these V\$ views can indicate that you may have a problem in your system with people having to wait, sometimes unnecessarily.

Operating System Tuning

Monitor and tune the overall operating system, checking CPU usage, I/O, and memory utilization. Many tools are available to help you with this. Several depend on the operating system on which they run, while some are fairly OS-transparent.

Finding the Trouble

It is difficult, at best, to fix the trouble if you don't know what's wrong with the application or code. Oracle thankfully provides many tools useful to a DBA in trouble. The majority of this chapter is thus dedicated to these tools, their use, and how to determine exactly what it is that's broken.

EXPLAIN Please

The first procedure you should develop when tuning is running (and demanding that others show you proof of running) an EXPLAIN PLAN for every non-ad hoc SQL query. While you can't actually get most of the end users to provide them, it is always helpful if you can acquire the explains for those ad hoc and often horribly created queries as well.

Traces

Traces are one of the most useful tools Oracle provides. They're the base on which many other tools ride. Oracle background processes, such as log writer, pmon, smon, or database writer, create many trace files automatically whenever they encounter an exception. These trace files (often the exception as well as the trace name) are recorded in the alert log and are created to provide a dump of the information surrounding the exception. They're also

150 Everyday Oracle DBA

frequently used for diagnostic purposes. I'll discuss them in greater detail in Chapter 5.

In this chapter, I'll cover the trace files that are deliberately created when attempting to fine-tune information.

AUTOTRACE

AUTOTRACE causes Oracle to print out the execution plan for the SQL statement following the AUTOTRACE command, and details the number of disk and buffer reads that occurred during the execution. It offers instant feedback on the execution plan for any successful SELECT, INSERT, UPDATE, or DELETE statement based on information that it stores in a plan table (which has to be present) under the user's schema. It also requires that the plustrace (\$ORACLE_HOME/sqlplus/admin/plustrce.sql) or DBA role be granted to the user executing the trace. These reports are particularly useful in monitoring, and are an excellent tool for tuning the performance of any given statement.

Controlling the report is a simple matter, accomplished by setting the AUTOTRACE system variable. Table 4-1 shows the different AUTOTRACE commands along with their descriptions.

It's important to remember that you can only use AUTOTRACE if you've been granted the PLUSTRACE role and a PLAN_TABLE has been created in your schema. It's often not a bad idea to grant the PLUSTRACE role to public so that anyone wishing to have a better understanding about what his or her SQL is doing can use the trace function. Couple this with the creation of a PLAN_TABLE as SYSTEM and creating a public synonym and granting SELECT, INSERT, UPDATE, and DELETE on the PLAN_TABLE to public to complete the ability for everyone to do an EXPLAIN or an AUTOTRACE. But you knew that.

Before you can grant the PLUSTRACE role, however, you must first create it. To create the PLUSTRACE role, use the following commands:

```
CONNECT / AS SYSDBA  
@$ORACLE_HOME/SQLPLUS/ADMIN/PLUSTRCE.SQL
```

Now, grant it to public.

```
SQL>GRANT PLUSTRACE to PUBLIC;
```

Then:

```
SQL>@$ORACLE_HOME/rdbms/admin/utlxplan
```

Chapter 4: Database Tuning: Making It Sing **151**

AUTOTRACE Command	Description
SET AUTOTRACE OFF	No AUTOTRACE report is generated (default).
SET AUTOTRACE ON EXPLAIN	The AUTOTRACE report shows the optimizer execution path with executed statements and output.
SET AUTOTRACE ON STATISTICS	The AUTOTRACE report shows only the SQL statement execution statistics.
SET AUTOTRACE ON	The AUTOTRACE report includes both the optimizer execution path and the SQL statement execution statistics.
SET AUTOTRACE TRACEONLY	Like SET AUTOTRACE ON, except that it suppresses the printing of the user's query output, if there is any.

TABLE 4-1. *AUTOTRACE Commands*

EXPLAIN PLAN

If AUTOTRACE creates a report of what execution path and statistics (and the statement was successful) did, an EXPLAIN PLAN is what you should create first (or what would be best for developers to provide you with when they submit a script for you to install) when you create a SQL statement or when you embed SQL in a package or procedure. The EXPLAIN PLAN shows you what Oracle's optimizer intends to do whenever it tries to run the statement.

The EXPLAIN PLAN output report is generated using the EXPLAIN PLAN command.

An EXPLAIN PLAN tells you how Oracle and the Cost-Based Optimizer (CBO) plan to execute your query. This information is particularly useful in tuning SQL queries that run against the database (whether stored in the database or in a third-party tool or as scripts in a file system) in order to structure either the queries or the data and database objects in order to get the queries to perform better. Once you have some idea how Oracle thinks

152 Everyday Oracle DBA

it will execute your query, you can change your environment or the structure of the query so you can make it run faster.

What are some of the red flags that can be brought to light in an EXPLAIN PLAN? While the list is not exhaustive, and by no means should constitute your entire attention when studying the EXPLAIN output, the following list tells you what you should be searching for.

- Cartesian products when they aren't anticipated
- Table scans, particularly on larger tables
- Unnecessary sorts
- Nonselective index scans

The EXPLAIN PLAN can give you the leverage to help convince developers to look at the code they're submitting and address inefficiencies. While this may bring a smile to many DBAs' lips, it can also point out those places in the database, design, and inner workings where it is inefficient as well. So don't get too puffed up if their code isn't always optimal. It may come back to bite you.

Before you can make use of the EXPLAIN PLAN command, you need to make sure you have a PLAN_TABLE installed and available to everyone who'll be running an EXPLAIN PLAN for insert update and delete. You can build the plan table by running the utlxplan.sql script that's located in the \$ORACLE_HOME/rdbms/admin/ directory (%ORACLE_HOME%\rdbms\admin\). After you have the PLAN_TABLE created, issuing the EXPLAIN PLAN command for the query you are interested in can be accomplished as follows:

```
EXPLAIN PLAN SET STATEMENT_ID='somevalue' FOR <your SQL statement>;
```

You need to provide a STATEMENT_ID so you can retrieve the information associated with just the given SQL statement. Then, you need a SQL statement that you're interested in looking at from a tuning perspective.


Once the explain is finished, you need to find out what Oracle is planning to do. This can be done by getting the information back out of the plan table.

```
SELECT LPAD(' ',2*(level-1)) || operation  
|| ' ' || options || ' ' || object_name || ' ' ||
```


Chapter 4: Database Tuning: Making It Sing 153

```
DECODE(id,0,'Cost = ' || position) QUERY_OUTPUT  
FROM plan_table  
START WITH id = 0  
AND statement_id = 'my_statement'  
CONNECT BY PRIOR id = parent_id  
AND statement_id = 'my_statement';
```

Now, want to look at the information a little more elegantly, and understand what's going on more easily? Try using `DBMS_XPLAN`. Starting in Oracle 9i, provided in the package was something that could not only make things easier to run, but easier to read as well. The format of the command follows:

```
 DBMS_XPLAN.DISPLAY(  
table_name IN VARCHAR2,  
statement_id IN VARCHAR2,  
format IN VARCHAR2);
```

The table name is the name of the table you're using to store your EXPLAIN PLAN. By default, it looks for a table called `PLAN_TABLE` if you don't pass it this parameter. `statement_id` is what you named the statement when you explained it to make it unique, and if not passed, it will assume you want to see everything. Format is how you want the output formatted, and assumes typical is the way you want it formatted. Yes, you do have choices in the way your output is formatted. The following is an explained list of formatting choices:

- **BASIC** Displays the minimum information
- **TYPICAL** Displays what is considered to be the most relevant information
- **SERIAL** Provides the same information as TYPICAL but without parallel information
- **ALL** Displays all available information

You can use `DBMS_XPLAN` in a variety of ways. It's best to set your line and page size up early when you're running your queries. This will save you aggravation later if you find you have to set them again after running a couple queries and discovering you're having trouble reading things.

154 Everyday Oracle DBA

```
SET LINESIZE 130  
SET PAGESIZE 0
```

If you just want to display the last plan explained, the following will accomplish this:

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY);
```

If you've named your plan something relevant (as is good practice), you can retrieve information on the named plan:

```
SELECT * FROM table  
(DBMS_XPLAN.DISPLAY('PLAN_TABLE', 'my_plan', 'TYPICAL'));
```

Another handy way to use DBMS_XPLAN is to set up a view that shows the last plan created. You can then simply query this view whenever you want to format the output of a plan. This is handy, too, when you don't want to give everyone access to the plan table, but you want everyone to be able to quickly see what's going on with their statement. Someone trusted can create the plan and then give select rights to that view to everyone.

```
CREATE OR REPLACE VIEW plan_view AS  
SELECT * FROM table(DBMS_XPLAN.DISPLAY);  
SELECT * FROM plan_view;
```

To read the output of either method, look at the indention caused by the query. The further indented the statement, the earlier in the process it's executed. Equally indented statements under the same parent are executed at the same time or one following the other, and then the combined total of the indented statement (or statements) is fed to the parent operation.

Keep in mind that full table scans can mean missing or inefficient indexes, statistics that are particularly outdated, or data used in unanticipated ways. They can also mean that the code is doing exactly what it should do. You need to understand the code and data to truly conclude whether the code is as inefficient as it may appear. A lot of what comes back from an explain may reflect badly on you, too, so cut everyone a little slack.

If you're lucky, you'll have your own developers and designers who are willing to embrace the idea of running EXPLAIN PLANS before they run their complex queries against a production database, or who are willing to wait to move up new code until the anticipated performance of the code is examined. This is *not* a replacement for testing and retesting against

Chapter 4: Database Tuning: Making It Sing 155

realistically sized data, but it does give you a place to start looking at what might happen with the data and the code.

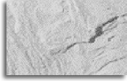
Granted, it's often because of management pushing that you don't have the time to address code inefficiencies upfront, but if you can show them that procedures like these are frequently a case of "pay me now or pay me later," and that paying later often is much more expensive in terms of time, effort, and lost efficiency, many of them may come around.

Running an explain is an inexpensive way to determine what the anticipated performance is of the new code and what effect the design of the database will have on it.

Want to go a step further? Let's look at the bigger picture. You can gather a set of baseline statistics (or what we'll call baseline statistics even if you don't take them at the beginning when the code first goes into the system) in an application, regardless of whether it's done at design time (although this would be the ideal) or when you start your tuning efforts, and you can amass the information and use it as a guide to measure changes against. Whenever there are any changes to the code, the structure of the data, database objects, or anything associated with the application, you can re-explain the changes and, based on the baseline, show how the code will affect the application, or how overall performance is likely to increase or decrease (yes, explains can point out good things as well as bad) as a result of the changes. By showing management how you can use this information to proactively tune both SQL and the database, you can prove your worth as a tuning DBA and provide convenient proof that may help you justify asking for training or books to help in further endeavors.

Keep in mind, however, that an EXPLAIN PLAN's results alone cannot unequivocally tell you which statements will perform well, and which will perform badly. An explain may show that a statement will do a full table scan. But this does not really mean anything by itself. A full table scan on a 100-row table (or other small table) may be far more efficient than using an index. The results of an explain may, on the other hand, show that a statement may be using an index, but if it's a pathetically inefficient index, then the fact it's actually being used is a trifle misleading. It may not run as quickly as it may appear. An EXPLAIN PLAN is only a tool. Use it to determine where to look and where to make changes. Use it to find out what a statement is doing before you make any changes, and then compare it with an EXPLAIN PLAN taken after any changes to see what those changes have done. It's nothing magic—no silver bullet for the database werewolf—it's just a tool to use.

156 Everyday Oracle DBA



NOTE

The EXPLAIN PLAN simply tells you what execution plan the optimizer would choose under the specific set of initialization parameters. It is not necessarily what would be used if the statement were actually executed. It may be interesting to see an explain of what Oracle chooses to do with a real execution of the statement.

V\$SQL_PLAN

An advance in DBA tools came with Oracle 9.2. Now, Oracle gives you the following newer views to help you identify spots where your users' SQL is performing below par, information which is available simply by running queries at the SQL prompt. You should keep an eye on these views, and check them periodically to see statistics associated with the SQL or to uncover full table scans that aren't expected in the database. They can be a very useful tool for gathering information required in planning corrective actions. Freely use the tools Oracle gives you. They're a wonderful place to start.

- **V\$SQL_PLAN** Shows the same kind of information seen in an EXPLAIN PLAN. The only difference? EXPLAIN PLAN shows the predicted execution plan, while V\$SQL_PLAN shows the one that was actually executed.
- **V\$SQL_PLAN_STATISTICS** Contains the execution statistics for each step in the V\$SQL_PLAN. Check this view to find poorly performing SQL operations. In order for this view to be populated, the initialization parameter STATISTICS_LEVEL must be set to ALL.
- **V\$SQL_PLAN_STATISTICS_ALL** Combines the information from V\$SQL_PLAN, V\$SQL_PLAN_STATISTICS, and V\$SQL_WORK_AREA. In order for this view to be populated, you must set the initialization parameter STATISTICS_LEVEL to ALL.

It is sometimes an interesting exercise to compare the EXPLAIN PLAN of a SQL statement with the real execution plan for the same statement from

Chapter 4: Database Tuning: Making It Sing 157

V\$SQL_PLAN and see if what the CBO thought it might do is actually what happened when the statement executed.

Personally, I like to request an EXPLAIN PLAN from my developers to see if they have thought about what the statement should do and what the new code is likely to do. Then, after the code is run in the target environment, I query the V\$SQL_PLAN view to see if that's really what happened.

You yourself can look to the V\$SQL_PLAN view to see those pieces of code that are run by the users, either through a tool like Business Objects or freehand through SQL*Plus or Toad. Not that user-created SQL is ever anything but terribly efficient, but sometimes it's worth the effort to see if they really are running as well as they can. Plus, it can open up avenues of discussion with users and may give you the opportunity to educate some of them on how to better construct some of the statements—things like an IN list with two values may be more efficient than a NOT IN list with 50 values, or that EQUALS usually performs better than LIKE if you can use it. Again, it isn't necessary to try and remove all the things we think of as "bad" SQL (like those performing full table scans), just look at the statement, think through the logic of what it's really trying to do, and decide if it should be reworked or not.

10046 Trace

One of the most popular trace events is the 10046 trace, which can tell you where a session is spending most of its time waiting and (depending on the settings you use) the bind variables and values employed in each instance. Enabling a 10046 event trace in a session creates a file that includes the details of all of the SQL statements and optionally the wait events and bind variables that occur in a session while the event trace is enabled (depending on the trace level you choose). It's one of the best tools to find out why a session is experiencing performance problems, and exactly what the session is spending its time waiting on.

So just what is a wait event (or timed event, depending on the version of Oracle you're using and the information you're viewing)? Well, at any given CPU cycle, Oracle is either busy doing something productive that serves a request, or it's waiting for something, some resource, that allows it to continue doing something productive. Often it is simply waiting for something to do, while occasionally it's waiting on database resources. Sometimes it's waiting for the I/O subsystem to be available in order to provide Oracle with information.

158 Everyday Oracle DBA

Using these wait events lets you see what Oracle is spending its time on. Is it wasting many cycles waiting for disk reads? Is it performing an inordinate number of full table scans just so it can get the information? Or, shudder, is it doing Cartesian joins to provide everything but the information the user ought to be getting?

You get data that touches upon so many different areas of your database, such as disk I/O, latches, parallel processing, network traffic, checkpoints, and blocking. Using this method, you can easily get at data showing you many different areas of your database, areas you might not otherwise have access to. Through the resulting trace, you can see information on disk I/Os, latches and locks, parallel processing, network traffic, and checkpoints. Furthermore, you'll get detailed information on such things as the file number and block number of a block being read from disk, or the name of a latch being waited on along with the number of retries.

In Oracle 8.1 and later, enabling tracing for the session can be accomplished as follows:

```
execute dbms_system.set_ev(sid, serial#, 10046, 8, '');
```

It's important to note, however, that you need to replace the "sid" and "serial#" in the preceding command with the real sid (or session ID) and serial number of the session you want to trace.

The "8" in the statement tells you the level of information you're gathering about the event. There are several different, and useful, levels of information you can gather. Table 4-2 displays what the useful levels are, and the meaning of each. The higher numbers include all the information of the lower numbers, and add their own details to those lower levels.

Level	Description
1	SQL statements
4	Includes details of bind variables as well as information from level 1
8	Includes wait events as well as information from level 4
12	Includes both bind variables and wait events as well as all lower levels

TABLE 4-2. *10046 Trace Levels*

Chapter 4: Database Tuning: Making It Sing 159

This provides you with a trace file. Looking at the trace file, you can see that reading the raw trace is not for the faint of heart. It takes real dedication to wade through the trace file to get at what's really going on. Fortunately, running TKPROF on the file allows you to easily see in English (instead of computerese) what's going on.

One of the things a 10046 trace can't show you, however, is the time Oracle spends waiting on the CPU to become available or the time it spends waiting on requested memory that may have been swapped out to then later be swapped back in. This means that, if you're working on trying to figure out why a SELECT statement is taking so long to process, you may find your 10046 trace shows you nothing in the way of wait events. This may lull you into a false sense of well-being that the SELECT is as efficient as it can possibly be. But if memory is an issue on the server, and the query is doing a very large number of logical reads, you may well be able to reduce the time the query takes by reducing the number of buffer gets through a rewrite of the query.

Do you see a large disparity between *ela* (elapsed time) and *cpu* (CPU time) without any apparent waits associated with those disparities? These disparities can be caused by waiting on the CPU, so indirectly you can infer waits associated to these as well. This takes inference, however. It isn't provided directly.

But this information is typically only used for SQL statements, or for tracing the path of a PL/SQL package to see what it's doing. Ever want to know the same kind of information from an export? While import and export aren't really the kinds of things people tend to trace, you can find some interesting information if you run a 10046 trace on an export or import "session."

You start out, naturally, by running the export session.

```
Exp system/manager full=y file=myfile.dmp
```

Keep in mind this is just an example. You can use whatever parameters you would ordinarily use in your export.

Once you've started your export running, log in to the database as a user with *dba* privileges and run the following statement.

```
Select sid, program from v$session where username = 'SYSTEM'
```


160 Everyday Oracle DBA

A list of all the sessions running as the SYSTEM user is then returned. The one that has the program exp@mydatabase.com associated with it is the export session you're looking for. Make note of the sid connected with this session and run the following statement:

```
Select s.sid, p.pid, p.spid
from v$session s, v$process p
Where s.paddr = p.addr
and s.sid = <the sid from the above statement>;
```

The sid and the pid from this command are used to generate the trace file for the process. spid is equivalent to the operating system ID of the export running.

```
Sqlplus '/ as sysdba'
oradebug setospid <p.spid>
oradebug event 10046 trace name context forever, level 12;
...
oradebug event 10046 trace name context off;
```

This will generate the trace file in the udump directory with the operating system process ID appended to the name.

10053 Trace

If you want to know why the Cost-Based Optimizer makes the decisions it does, use a 10053 trace. While the output of setting the trace to look at this is rather cryptic and difficult to wade through, you can make it less tortuous by searching in the output file (with find or grep or some other string-locating tool) for either the phrase "Join order" or "Best so far" to see why it made the choice it did.

The text associated with "Join order" can tell you the tables by name and the order the optimizer chose to join them. This may not be of much help to you if you've deliberately tried to get Oracle to join in an order that you would like to see, but you can at least view what order it's decided on as the most efficient. You can then check if there's anything you can do to influence the optimizer's decision.

"Best so far" is associated with what the optimizer has decided should be the most optimal plan and join order. By looking for this phrase in the trace file, you can start to see what the optimizer is thinking would be best, even if you have other thoughts on the matter.

Chapter 4: Database Tuning: Making It Sing 161

10032 Trace

10032 trace checks what happens during sorts. This can be useful for many ad hoc queries seen in your system, particularly those that you may notice are doing several different sorts, or which seem to be spending a poor amount of time in any given sort.

```
 alter session set events '10032 trace name context forever, level 1';
```

So why is this in any way useful?

While we all know it isn't possible to eliminate every single disk sort, it is possible to minimize these. You can tinker with the `SORT_AREA_SIZE` parameter and the `SORT_MULTIBLOCK_READ_COUNT` parameter to make a big difference in the way these sorts perform.

Understand that a great part of the performance of a disk sort has to do with the number of merge passes required for the sort to occur (this dictates the amount of temporary tablespace I/O that occurs in the sort).

You may be running a `SELECT` statement that retrieves rows from several different tables. The results are retrieved separately, but are then merged. The number of these sets you can merge simultaneously is called the merge width. This merge width is directly related to the combination of `SORT_AREA_SIZE` and `SORT_MULTIBLOCK_READ_COUNT`. If the number of sets of rows returned by the statement is larger than the merge width, then multiple passes will be needed for the merge to complete.

The `SORT_AREA_SIZE` is made up of read buffers and write buffers. The size of each buffer is `SORT_MULTIBLOCK_READ_COUNT * DB_BLOCK_SIZE`, and two read buffers are needed for each set of returned rows. The same write buffer configuration is used all the time during the sort. Consider a query that returns ten sets of rows—that's 20 read buffers. This means that up to 90 percent of the `SORT_AREA_SIZE` is allocated to read buffers.

If there are too many sets to be run through the merge width, then secondary passes will have to be made and more I/Os will occur. These secondary passes are what make the disk sorts particularly inefficient from a performance perspective. That's why the information retrieved from the 10032 trace can be particularly important in tuning, allowing you to more effectively set the `SORT_AREA_SIZE` so that secondary passes don't occur.

If you know you have a batch process (your most robust and data-intensive batch process, so you size for the biggest knowing that everything else will comfortably fit) that processes and sorts 12GB of data and you currently

162 Everyday Oracle DBA

have your `SORT_MULTIBLOCK_READ_COUNT` using eight blocks (with each block being 8KB), this means that the number of initial sets that can be run through your `SORT_AREA_SIZE` will need to be

$$12\text{GB}/.9 * \text{SORT_AREA_SIZE}$$

Thus, your merge width is

$$\text{round}(.9 * \text{SORT_AREA_SIZE} / (8\text{KB} * 8)/2$$

Therefore, to ensure that the sets that fit are no greater than the merge width, you have to have a `SORT_AREA_SIZE` of at least

$$\text{SQRT}(12\text{GB} * 64\text{KB} * 2/.81)$$

This is 43.6MB.

There is virtually no benefit to having a `SORT_AREA_SIZE` of greater than 45MB.

So you see, you can use the output of this trace to help you determine if you have sufficient space allocated to `SORT_AREA_SIZE`, and thus limit the disk sorts that occur in your system. Every trace fill provides you with information on your system's `SORT_AREA_SIZE`, `SORT_AREA_RETAINED_SIZE`, `SORT_MULTIBLOCK_READ_COUNT`, and the `MAX INTERMEDIATE MERGE WIDTH`. There are three additional pieces of information you can get from the resulting trace file: initial runs, intermediate runs, and number of merges.

10033 Trace

If you want to see the same kind of information found in a 10032 trace generated for particularly large sorts, you can get that information by setting the event trace to 10033.

```
 alter session set events '10033 trace name context forever level 4';
```

While most intensive sorts occur with batch processing, every once in a while a truly huge sort is needed. Setting the parameters to allow for these unusual sorts of circumstances can be assisted with the 10033 trace. Particularly with the combination of a 10032 and 10033 trace.

Chapter 4: Database Tuning: Making It Sing 163

In a 10033 trace, Oracle lists each section of sorted data it's writing to disc. Then it describes, in database-ese, how it's re-reading each of these sections back so it can merge them into sorted order. Whenever a sort gets very large, the time it takes to perform the sort is to a very great extent based on the number of merge passes that have to take place. So taking the information from a 10033 trace combined with the 10032 trace may give you the information you need to add that extra oomph, allowing as much sorting to occur in memory as possible.

10104 Trace

While it's often true that hash joins are more efficient than sort merges, it's sometimes difficult to get statistics on hash joins to tell you if they're as efficient as they can be. While this won't tell you overall what's going on with all hash joins in your system, you can look at what's going on with any given query, and the hash joins that are occurring within it. To look at what's going on during a hash join, use the 10104 event trace.

In the resulting trace file, look for the line that provides you with the line Memory After Hash Table Overhead and the line with the information on Estimated Input Size. If the Estimated Input Size is larger than the value associated with Memory After Hash Table Overhead, then Oracle estimates your hash area is likely too small. It's as true here as anywhere that poor statistics will result in poor estimates on the required sizes because Oracle won't be sure how many rows it should expect.

You can also find in the trace file the total number of partitions and the number of partitions that fit in memory. If the number of partitions that will fit in memory is smaller than the total number of partitions, then your hash area will likely be too small and should at least be multiplied by the ratio of

Total number of partitions/number of partitions that will fit in memory


Now that you've created a bunch of trace files, how do you find them so you can do your analysis?

Finding Your Trace File

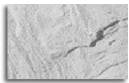
Looking at these last several ways of gathering tuning information, you can see that the trace events all produce trace files, which are all written to the udump directory (since they're associated with user processes). The trace

164 Everyday Oracle DBA

files end up in the udump directory and, unless you make some minor adjustments, will look disturbingly like any other trace files that might be in the directory. How on earth can you distinguish your trace file from all of the other trace files in the directories? The best way is to use the following code as an example to set your file apart by linking it to something of significance.

```
 Alter session set tracefile_identifier = 'something significant to you';
```

Now you should be able to find the files associated with your trace session.



NOTE

It's important to remember that when you're tracing statements you need to make sure you've informed the database that you want to see as much information that it will generate as possible. To this end, you must make sure you've set the maximum dump file size (which is really what these trace files are: dump files) to a size sufficient to hold all that information. The parameter you're concerned with is MAX_DUMP_FILE_SIZE and you must make it large enough to hold what you need. Valid settings are simply numbers (indicating the number of physical operating system blocks), a number followed by an M or a K (indicating megabytes or kilobytes), and unlimited. For the duration of the information gathering session, it would be wise to alter the session or alter the database so that MAX_DUMP_FILE_SIZE is set to unlimited. However, make sure that when you're done you turn the feature off or else you'll find your user dump destination filling up with huge trace files for things you didn't realize were being generated.

TKPROF

The TKPROF program converts your trace files (both those you create deliberately and those created by the background processes if you like) into

Chapter 4: Database Tuning: Making It Sing 165

a more readable form. You can use TKPROF to see the contents of the trace file created in order to view what a problem query is doing.

It's important to note, however, that in order to get the most out of either the trace file you create or the TKPROF utility, you must have enabled timed statistics either by setting the initialization parameter in the `init.ora` or the `spfile`, or by running the following command:

```
ALTER SYSTEM SET TIMED_STATISTICS = TRUE;
```

A plan table can be used with TKPROF. If you do use it, the plan table must already be present for it to run successfully. If one isn't present, it can be created by the SYSTEM user and granted access and synonyms to public.

```
@ORACLE_HOME\rdbms\admin\utlxplan.sql  
CREATE PUBLIC SYNONYM PLAN_TABLE FOR SYSTEM.PLAN_TABLE;  
GRANT SELECT, INSERT, UPDATE, DELETE ON SYSTEM.PLAN_TABLE TO PUBLIC;
```

Including the explain is particularly helpful when the SQL statement you're examining has a cursor that isn't closed by the end of the trace file. In this case, the TKPROF output does not automatically include the plan of the SQL statement. TKPROF also doesn't report on commits and rollbacks that might be in the trace file.

Now, you can create a trace file either by setting `SQL_TRACE` to true in a session or by setting the event you want to analyze and running the queries of interest. Remember that the resulting file will be in the UDUMP directory for your instance (`USER_DUMP_DEST` from either the `PFILE` or the `SPFILE`) and you can now run TKPROF at the command prompt with the following:

```
TKPROF <your trace file> <your output file> explain=query_user/pass@db
```

It will assume you want to use the table you created called `plan_table` unless you tell it otherwise (this is one of the benefits of creating the public synonym for the table). If the table doesn't exist, it creates its own table and then drops it when it's done. The resulting sorted and explained file contains the following kinds of information for all SQL statements executed and found within the trace file (this includes the `ALTER SESSION` commands used to get the trace file). See Table 4-3.

166 Everyday Oracle DBA

Command	Description
COUNT	The number of times something occurs during the run of the statement (fetches, parses, executions, and so on)
CPU	The CPU time in seconds that elapsed during the statement's execution
Elapsed	The elapsed time in seconds that occurred during the execution of the statement
Disk	The number of physical reads from the disk
Query	The number of buffers acquired for a consistent read
Current	The number of buffers acquired in the current mode (usually found in update, insert, or delete statements)
Rows	The number of rows processed by the fetch in the statement (implicit or explicit fetches) or the execute call

TABLE 4-3. *Types of Information Obtained from SQL Statements*

Things to watch for in the trace include

- When tracing lots of statements at once, such as batch processes, you can quickly discard those statements which have unacceptable CPU times. It's often better to focus on those statements that take most of the CPU's time.
- Inefficient statements are mostly associated with a high number of block visits.
- High CPU values or high elapsed-time values may indicate you have latching problems and inefficient PL/SQL loops. Multiple parse calls for a single statement imply you may have an issue with your library cache
- Highly inefficient coding
- *Always* check the execution plan to see why the statement is performing badly.

Chapter 4: Database Tuning: Making It Sing 167

- Also, you should compare the `autotrace traceonly explain` to a `TKPROF` output so you can compare what the `autotrace's explain` thought would happen with what actually occurred. The `TKPROF` with its counts should be very similar to the `autotrace's explain`. If there are significant differences, then the optimizer is making bad assumptions likely based on stale, missing, or invalid statistics. Step one in correcting this is to figure out why the optimizer is making bad assumptions.

Trace Analyzer

Trace Analyzer is a lesser-known tool (available for download from Oracle) that acts as a substitute for the `TKPROF` utility, which is used in analyzing trace files.

The latest version of Trace Analyzer can be acquired from Metalink by clicking the link in text note number 224270.1 on the Oracle web site. It also includes excellent information on the utility itself.

Trace Analyzer goes `TKPROF` a step further, however. It provides information on more wait events than `TKPROF` does, offers actual values of bind variables used when each included SQL statement was executed, and shows information on the hottest blocks, CBO stats for tables, and indexes included in the statements and execution order (things that `TKPROF` isn't always able to provide).

Trace Analyzer can be used on any database (OLTP, DSS, or Oracle E-Business Suite), but it does require that a one-time configuration be performed. This installation puts into place many database objects that are used as a tracing repository.

Once the utility has been downloaded from Metalink and the setup steps have been followed, you can execute a SQL statement and then the name of the resulting trace file can be passed to Trace Analyzer, which then provides you with the tuning information. The scripts involved in the installation are as follows:

- **TRCACREA.sql** Creates all of the objects needed by Trace Analyzer. It calls the following scripts:
- **TRCADROP.sql** Drops all schema objects
- **TRCAPKGB.sql** Creates the package body

168 Everyday Oracle DBA

- **TRCAPKGS.sql** Creates the package specification
- **TRCAREPO.sql** Creates the staging repository
- **TRCADIRA.sql** Creates the directory object that points to the place where the given trace file lives. This is really only useful when using a destination other than that pointed to by `user_dump_dest`.
- **TRCAGRNT.sql** Grants privileges needed to use Trace Analyzer
- **TRCAREVK.sql** Revokes privileges granted by TRCAGRNT (this runs first to remove the privileges before the grants coming from TRCAGRNT.sql)
- **TRCAPURG.sql** Purges old SQL traces from the repository
- **TRCATRNC.sql** Truncates the staging repository
- **TRCANLZR.sql** Script that generates the report
- **TRCACRSR.sql** Generates report for one cursor
- **TRCAEXEC.sql** Generates report for one cursor execution

TRCSESS

What, there's more? Hey, Oracle is nothing if not prolific when it comes to providing tuning tools. For those of you who are using, or who are considering using, Oracle 10g, this one's for you! Though available for use, it doesn't work with earlier releases.

Need to find out what's going on in the database across several different trace files, and you know you generated these trace files for a given user on a given database? Don't want to struggle with figuring out how to analyze each trace file individually and then try to pull together the information from each file into a whole application situation analysis? The answer: TRCSESS.

If you know which trace files you're dealing with (the set of trace files containing the right combination of `SID`, `CLIENT_IDENTIFIER`, `SERVICE_NAME`, `ACTION_NAM`, and/or `MODULE_NAME` variables) in order to pull all the information together for a given analysis, you can use TRCSESS to do it for you. This can be very beneficial if you're trying to tune out the bottlenecks in the database or in an application that's consuming large amounts of

Chapter 4: Database Tuning: Making It Sing **169**

resources. TRCSESS is particularly useful in shared server environments where several processes are each running and writing trace files.

TRCSESS is a Java application executed from the command line that consolidates trace information from multiple trace files. Oracle provides a shell script which you can execute rather than call Java directly. The output of the TRCSESS utility can then be used as the input to TKPROF or Trace Analyzer.

Statspack

There are some very good books out there about the care and feeding of your database using Statspack, but no tuning chapter would be worth its salt if it didn't cover the use of Statspack as a way to determine where performance is an issue.

Before you rely too heavily on Statspack reports, make sure they provide you with database-wide information and aren't looked at as a tool simply for tuning individual SQL statements.

One of the most useful things you can do with Statspack is find out your top five wait events and determine which ones you can actually do something about. To this end, Statspack provides a report called "Top 5 Timed Events." While this report can be minimally useful even if TIMED_STATISTICS is set to false, setting it to true can provide you with a list ordered by time waited rather than just the number of times waited. This can be more valuable because you may not care that the event that waited the longest was SQLNET Message From Client, but you indeed might care that you waited 874 seconds for a DB file scattered read.

Table 4-4 provides you with a general overview of what the most common "Top 5 Timed Events" mean to you and what you can do to fix them.

Event	Meaning	Potential Fix
DB File Scattered Read	Generally indicates waits related to full table scans.	Find where you may be missing indexes or where indexes may be advantageous.

TABLE 4-4. *Wait Events Prevalent in Statspack Reports*

170 Everyday Oracle DBA

Event	Meaning	Potential Fix
DB File Sequential Read	Generally means you're doing an index read. A lot of waits attributed to this can mean that join orders are less than optimal or that you have unselective indexing.	Check joins on tables to see if you can make them more efficient.
Free Buffer	Your system is probably waiting for a free buffer in memory. There probably aren't any currently available. This is usually indicative of a need to increase the DB_BUFFER_CACHE or to tune your SQL statements to be as efficient as possible (do this first, before changing database parameters).	To fix this (remember, one at a time), tune the SQL, increase DB_BUFFER_CACHE, increase the amount of checkpointing you're doing, use more Database Writer processes, or increase the number of physical disks over which your data is spread.
Buffer Busy	You are waiting for a buffer that is already being used and is currently unsharable, or that is currently being read into cache. Waits for this are expected, but should not be excessive.	If you're seeing an excessive amount of waits for this, you may want to increase DB_CACHE_SIZE, or migrate to ASSM.
Latch Free	Latches protect shared memory structures and are usually very rapidly obtained and released. They prevent the concurrent alteration of shared memory structures. Concurrent select is not only okay, it is often a way of life in a busy database, but Oracle protects data from concurrent updates. Typically, when you see many of these, it indicates that you aren't using bind variables, buffer cache contention, or hot blocks in the buffer cache... or that you may be running into a latch-related bug (remember, Metalink can be your friend).	Further investigation is usually needed. Because there are so many causes, you can look at Statspack reports to see if the top latch waits, allowing you to start working through the issues with your database.

TABLE 4-4. *Wait Events Prevalent in Statspack Reports (continued)*

Chapter 4: Database Tuning: Making It Sing **171**

Event	Meaning	Potential Fix
Enqueue	Enqueue is a lock that protects shared resources. It takes care of queuing (first in, first out) requests so that resources are allocated equitably. Common events that cause these waits are space management and the allocation of dictionary-managed tablespaces, attempts to duplicate unique indexes, multiple concurrent attempts to update the same bitmap index fragment, and multiple users updating the same row.	Setting Inittans or Maxtrans to a higher number, thereby allowing more concurrent access to any given block is one place you can look to get around this problem in pre-Oracle 10g versions. With the advent of Oracle 10g, maxtrans became 255 regardless. You can also make sure you have indexes on foreign keys as a means to avoid enqueue waits. It's important to note, however, that one of the major problems that contributes to this being an issue is poor application design.
Log Buffer Space	You are filling log buffers faster than log writer can write them out to redo.	Putting your redo logs on faster disks so that writes to them can occur as quickly as possible will ease this problem. You need to empty the log buffers as quickly as possible.
Log File Switch	Caused by commits waiting for a log file switch (particularly if archiving is needed or checkpointing is incomplete).	Make sure your disk isn't full. Look for I/O contention. Add more or increase the size of the redo logs. As a last resort, try adding more database writers.
Log File Sync	Log writer isn't flushing the session redo information from the buffer to the redo logs rapidly enough.	Commit more records simultaneously (50 or 100 at a time rather than one at a time). You could try to move your redo log files to faster disks, but if it comes at the expense of hot data files, you may end up robbing Peter to pay Paul, as they say. Avoiding RAID 5 may help, but then you probably already knew that.

TABLE 4-4. *Wait Events Prevalent in Statspack Reports (continued)*

172 Everyday Oracle DBA

Event	Meaning	Potential Fix
Idle Event	The ever-popular System Idle Events. While it's typically okay to ignore most of these, you may want to note if they suddenly start happening. It could indicate you've moved your bottleneck and may need to revisit some of your other tuning tools.	

TABLE 4-4. *Wait Events Prevalent in Statspack Reports (continued)*

So you see, Statspack can be very useful when looking at tuning the overall database. It can give you yet another useful tool, a means to come at the problem from another avenue, and some extra ideas on why your applications aren't performing the way users anticipate.

Since users typically have preconceived notions about how a database should perform, let's look at them next.

Users

While it's true that without users the database would absolutely fly, it's also true that without users you'd probably be out of a job (after all, what good is a database—and by extension a DBA—if there aren't users using the data?). Given that users happen, what is the best thing we can do to tune the users?

It may not be high on your list, but teaching users, training them, and/or creating a series of presentations or conversations to help them learn is your best asset from a database tuning perspective.

Training doesn't always have to be formal. Lunch with the DBA in a meeting room where you just sit around and talk about SQL and PL/SQL and the wonders of Triggers is a great idea. One of the best Oracle University classes I ever attended was the PL/SQL class in Minneapolis where I learned that a *Trigger's a Wonderful Thing* (thank you very much, Winnie the Pooh). It may be a somewhat long process, but it will be well worth the effort in the long run, and you might just find out that developers, and users, are real people, too. Plus, it will impress everyone that you're trying to foster a team atmosphere.

Chapter 4: Database Tuning: Making It Sing 173

I don't know about you, but I've never been all that much of a people person; it is, after all, mostly why I got into computers. You don't have to deal with people all that much. But if you can get an idea of what the users (the developers, the designers, the end users) are trying to do, you can do a better job at tuning the database from the outside, and you can give the users information that will help make your life a whole lot easier. It really is a win-win situation.

Fixing the Trouble

Once you've gathered the information needed using your arsenal of tools, you can set about fixing the trouble, or at least addressing the issues that were uncovered as a result of the tuning efforts to this point.

There is something of an art to tuning SQL statements. It's important to remember (and I can't stress this enough) that you should only change one thing at a time and then see what effect that change has on the performance of a statement.

Tuning Database Parameters

You can make some inferences on database parameters that need to be tuned based on a lot of the V\$ views (the dynamic performance views, wink wink) available in the database. These views are built with you in mind. Make use of them wisely and you can have your database humming in no time.

V\$FILESTAT

V\$FILESTAT provides you with information on activity relating to each file in the database. Interesting columns include PHYRDS (physical reads), PHYWRTS (physical writes), SINGLEBLKRDS (random I/O that's occurred with the file), AVGIOTIM (the average time, to the hundredth of a second, that has been spent on I/O), READTIM (the time spent, in hundredths of seconds, doing reads), WRITETIM (the time spent, in hundredths of seconds, doing writes), and LSTIOTIM (the time spent, in hundredths of seconds, doing the last I/O operation). This view is typically joined to V\$DATAFILE and V\$TABLESPACE to provide really relevant information. It is important to note, however, that if you are using async io, you should not set too much store in the write time. In fact, in that particular circumstance WRITETIM is meaningless.

174 Everyday Oracle DBA

```
select d.name as file, t.name as tablespace,  
       f.phyblkrd, f.phyblkwrt, f.readtim, f.writetim  
from v$filestat f, v$datafile d, v$tablespace t  
where f.file# = d.file#  
and d.ts# = t.ts#  
NAME
```

NAME	PHYBLKRD	PHYBLKWRT	READTIM	WRITETIM
D:\ORACLE\ORADATA\10G\TEST10\SYSTEM01.DBF				
SYSTEM	5308	351	2409	457
D:\ORACLE\PRODUCT\10.1.0\ORADATA\TEST10\SYSTEM02.DBF				
SYSTEM	4241	132	1788	194
D:\ORACLE\ORADATA\10G\TEST10\UNDOTBS01.DBF				
UNDOTBS1	36	3989	253	6472
D:\ORACLE\ORADATA\10G\TEST10\SYSAUX01.DBF				
SYSAUX	2573	1473	897	2879
D:\ORACLE\PRODUCT\10.1.0\ORADATA\TEST10\SYSAUX02.DBF				
SYSAUX	1488	1190	1713	2206
D:\ORACLE\ORADATA\10G\TEST10\USERS01.DBF				
USERS	13	7	26	8

It's important to note that V\$ views are cumulative. They are virtually empty when you start up the instance and they accumulate data while your database is open. Though this means you don't see what's happening at exactly the time when the query is run, the way you would with, say, a 10046 trace, it does mean you can run a query now, run it again in 15 minutes (or set it to run every 15 minutes for a couple of hours), and then write the results out to a file. Look at the files when you're done (diff them in Unix... heck, download and run CYGWIN on Windows and diff the files) and see what I/Os have been happening during the time. Yeah, yeah, I know, Statspack will give you the same thing, but sometimes I like to have ultimate control of what I'm looking at. Frequently I know that my developers are testing and yet I want to be able to granularly run the queries on that environment every five or ten minutes. Though I already have Statspack set up to run every hour, I may just want to do something quick and dirty. I have this scripted so I can simply run it whenever I want and spool the output to flat files. The same information will end up aggregated in Statspack, but I can send the files to the project manager who just doesn't get that whole SQL stuff but who does really understand file processing (or just feels more comfortable with files that he/she can touch and feel), and it gives them a warm fuzzy feeling without changing my Statspack reporting. Sometimes it's

Chapter 4: Database Tuning: Making It Sing 175

as important to be flexible and to know your users (know your audience) as it is to know your database, the data, and the application. Have as many tools as you can in your toolbox so you can meet any need.

V\$LATCH

When I was growing up, there was an interesting latch on my bedroom door. It was metal and ornate and had a lever that swung across and clasped another metal piece on the other side of the door. This latch was designed to keep people out and protect the privacy of the people in the bedroom. There was no way to swing the latch to lock the door from the outside, or to unlock it from the outside. Its purpose was to temporarily block access to the room.

Database latches are pretty much the same, although way less decorative. They temporarily prevent access to the inside of Oracle (its memory structures primarily) while the inside process is already accessing them. When that process completes, it opens the latch and leaves, allowing the next process to make use of the memory. Some processes (like your parents) are willing to wait outside the door for you to come and open the latch. Others, like your little brother, will stay there until they get tired of waiting, and then finally give up.

V\$LATCH tells you about the latches, the address of the latch object, its level and name, the number of times it obtained a wait (GETS) and the number of times it obtained a wait but failed on the first try (MISSES), the number of times it got tired and decided to nap while waiting (SLEEPS), the number of times it was obtained without a wait (IMMEDIATE_GETS), and the number of times it failed to be obtained without a wait (IMMEDIATE_MISSES). If you're seeing an inordinate number of latches in your wait events, this view may be one you'll want to investigate further.

V\$LIBRARYCACHE

Just like when you go to the library (you DO go to the library, right?), the first time you try to find a book, you usually have to go look it up in the online catalog or the card catalog (yes, there are still libraries that use them). Once you find the call number, you then search the stacks till you find the book (if it's not checked out or misshelved, that is). If you find you like the book, or similar books, or books by the same author, you may not have to

176 Everyday Oracle DBA

go back to the card catalog to find the books. You may just decide to return to the stacks where you found this book and dig up others that are similar.

Oracle kind of works the same way. Once a SQL statement is presented to Oracle for processing, Oracle goes to its card catalog to find where the information can be found and gets it for you. The first time the information has to be read into memory (like reading the Dewey Decimal number of the book into your own memory). Every time after that (well, okay, every time after that until you shut down the database or until it gets aged out of memory) that you want to find the same piece of information (as long as you want the same information the same way), Oracle will remember the statement and won't need to look up the information again. If Oracle can use something stored in active memory (like you using what's in your short-term memory rather than having to dredge it up from your long-term memory banks), it will be much faster. That's why cursor sharing, set correctly, can be critical—so that the more Oracle can use what it knows and make as many inferences (okay, you want blue sneakers and then next time the only difference is that the bind variable says you want red sneakers) as it can and reuse as much of what it has in its memory, the faster the queries will run. And that's what we're after here, right? So let's talk about that next.

Cursor Sharing

So what is cursor sharing exactly? Well, sometimes you may have SQL statements that you run over and over again where the only difference between them is what kind of information you want to bring back. This is often particularly true when using forms to allow input from the user into the statement. But if you let Oracle make some inferences, and set cursor sharing up so it can leverage those inferences, your queries will run faster.

CURSOR_SHARING is a parameter that Oracle 8i brought with it that allows developers to make liberal use of bind variables and then let Oracle use those bind variables (rather than their literal values) to imply how to predict selectivity of the statement. In Oracle 8i, you could set the initialization parameter up so that Oracle would re-use the bind variable-laden cursor (cursor_sharing = force) for subsequent runs. Historically, it's proven difficult for the optimizer to guess accurately on the selectivity of what might be coming in buried in the bind variables. Originally, the functionality was limited. In Oracle 8i, it meant that if the first time the statement was parsed it used an index, every time it was rerun it used an index even if other times it might have been more efficient to use a full table scan and vice versa.

Chapter 4: Database Tuning: Making It Sing 177

Oracle 9i brought added functionality (`cursor_sharing = similar`) that allowed the optimizer to examine histograms associated with the columns connected with the bind variables every time the statement was run, and then determine if a full table scan or an index scan would be more efficient when reusing the same statement. This means that code in memory is reused (pinned maybe) and that the optimizer can creatively change its mind based on the values in the bind variables, making cursor sharing the best of both worlds.

Unlike the inferences you can make about book topics or authors, Oracle can't make inferences about similar information possibly being in similar locations. It has to rely on what it really knows for sure.

If you see waits associated with the library cache, you can use the `V$LIBRARYCACHE` view to see what might be going on. `GETHITS` means that the object was in memory when Oracle tried to find it. `GETS` is the number of times Oracle tried to find the objects it needed in memory. `GETHITRATIO` is the number of `GETHITS` compared to the number of `GETS`. `PINS` is the number of times a PIN was requested for objects in memory (pinning an object in memory means you can usually rely on it being there the next time that you want it). `PINHITS` is the number of times all the pieces for a request were found already in memory. `RELOADS` happens the first time a pinned object is requested, and is the effort it takes Oracle to read that piece of data into short-term memory (like when you look up a phone number the first time, 555-1212... 555-1212... 555-1212; after that, when you have it all nice and recalled, it's just a matter of hitting redial).

V\$LOCK

As you go through your day, take notice of how many locks you see or use. Think back to when you were a kid and try to figure out how many locks you use now as compared to back then.

Looking around me right now, I can see the lock on my office door, the lock on the roll top of my roll top desk, the one on the drawers of the desk, the one on the door of the desk, the two on the window, the one on each file cabinet, the lock on the administrator account of the computer, and the one on my PDA. I can infer that, across the street, there's one or more locks on the front door of the house opposite me, one on the garage door, one on the gate of the fence, one on the car parked under the street light, one on the electric meter, one on the gas meter, one on the telephone junction box, and a lock on the utility box on the corner. I have a lock on my desk at work, one connecting my laptop to my desk, I have to go through two to

178 Everyday Oracle DBA

enter the gate at work, and navigate at least four (and as many as seven) just to get to my desk in the office. There is a lock on the doors at the grocery store, at the drug store, and on the construction machinery lining the highway on my drive to work.

There are locks everywhere.

Just like how latches and locks in real life serve different purposes (latches usually keep you from accidentally getting into somewhere unless you really mean to, and locks keep honest people out of places where they might stray), they serve different purposes in Oracle, too. Latches are constructs that control accesses to memory structures, while locks protect storage structures and the data residing within them.

Oracle is a lot like your daily life. There are locks, or the potential for locks, almost everywhere. There are different kinds of locks, too. Oracle may use one kind of lock on a structure when you just want to go poking around looking at data but don't want the data to change while you're looking at it. If you want to change the data, however, Oracle will use a different lock so you, and only you, can change that data at a given time.

If you're seeing excessive waits associated with locks, there may not be anything you can do directly, but you *can* go and look at what's being locked, and why. With this information, you can then possibly make alterations to the application or the structures. V\$LOCK view gives you some vital information on the locks currently held in your database. The view provides you with the following columns:

- **SID** The session ID of the locking session
- **TYPE** The type of lock being held (TM for table level, TX for row transactions, ST for space transaction locks)
- **LMODE** The lock mode in which the session holds the lock
- **BLOCK** Here, a value of 0 means this given lock isn't locking another transaction's lock. A value of 1 means it's blocking another lock.

You can use V\$LOCK to find out what session is holding the lock, V\$SESSION to find out what SQL statement is being executed by the sessions holding the lock and waiting on the locked resource, and the program and user holding the lock. You can also use V\$SESSION_WAIT to find out what the session holding the lock is being blocked on.

Chapter 4: Database Tuning: Making It Sing 179

```
SELECT LPAD(' ', DECODE(REQUEST, 0, 0, 1)) || SID,  
       id1, id2, lmode, request, type  
FROM V$LOCK  
WHERE id1 IN (SELECT id1 FROM V$LOCK WHERE lmode = 0);
```

This leads to the following SQL hash:

```
SELECT sid, sql_hash_value  
FROM V$SESSION  
WHERE SID IN (<the sid list from the previous query>);
```

And now the SQL statement:

```
SELECT sql_text  
FROM V$SQL_TEXT  
WHERE hash_value in (<hash value from the last query>);
```

Okay, so now we've looked at the internals that can be used to fix the performance of the overall database, but isn't there something more that can be done?

Tuning the Database Structure

This is where I think being a DBA becomes fun... where you get to play with the art, not just the math. Even in the art, there is math, like the art of nature and the relationship of the spirals in a pinecone or the arrangements of leaves on a plant or Fibonacci numbers. But tuning the structure can bring out the creativity of a DBA.

While I've done some formal design study, and I know there's a time and place for a third normal form (although anything over that is questionable to me), there's also a time and place for a denormalized model, and not just in a decision support system. I've made some very unpopular suggestions that have met with much resistance because they're different than the way things are normally done. But it doesn't mean that the current way is the best one, or that it's particularly efficient.

Look at the data you're storing. If you have a date field, which is broken into month and year, ask yourself why. Advanced ideas in database design can mean you can build function-based indexes to do date manipulation more efficiently. So, you can get the month out of a date that contains day, month, and year. You can also get the year out of that same date. Thus, you've saved 15 bytes on every row, meaning there's less chance for row

180 Everyday Oracle DBA

chaining, and less chance for error. While this is not (in the grand scheme of Oracle) really considered to be one of the more advanced concepts, if you work at an organization that balks at using stored procedures or changing the way an index is constructed because they've always done it a certain way, advanced design can take on an entirely new meaning.

Row chaining occurs when the process of updating a row makes it long enough that it no longer fits within the block in which it started. Why is there less chance of row chaining in this instance? Well, the smaller your row size, the less chance there is it will be forced into another block. Planning will also help you design block and row sizes that make optimal use of data blocks and which likely won't end up chained to other blocks.

Do you have a string of characters you're storing in a VARCHAR2(60), but you always use the first 15 bytes to mean one thing and the next ten to mean something else, and the next 25 to mean something else, while the last ten are always blank? And every query you run substrings out those chunks into their real meaning? The substringing probably means you aren't using the index you've built on the column. Instead, it means you may be performing the same function repeatedly on the same column so you can get each different chunk into its own variable in your code.

Do you have a primary key on the first and second column of a table? If so, do you have a unique index on the first, second, and 15th column on that same table? Why? If the primary key is unique, anything you put with it will be unique. Why not? The definition can allow the optimizer to make some intelligent inferences on the uniqueness of the combination that our "logic" tells us but that can only be built into the optimizer with enough information. The more information you can give the optimizer, the better. And as long as the index is used, the trade-off of space for speed will likely be to your advantage.

Are you storing a set of lookup values on every record of a hundred million-row table (for example, city, state, and ZIP code) just because you don't want to have to join by ZIP code to another table (everyone knows that if you can avoid a join, the queries will be performed faster), yet you only actually go after the details (city and state) 1 percent of the time a query is run on the table? The chances of data error are far greater than the price of a well-created join (a join, I might add, that could be precomputed with a materialized view).

And, no, there is no earthly reason to ask Oracle to come up with a database model that allows you to have a table with 2500 columns in it. No

Chapter 4: Database Tuning: Making It Sing **181**

one will ever remember what is in half of those columns. Plus, joins are becoming more and more efficient over time.

Think through your ideas. Present them logically (maybe even hide a small database somewhere and do a small proof of concept using production statistics to bear out your ideas) and then see how it goes. Even a re-design of a database may be optimal in some cases.

What, More?

Want to see Import/Export run a little faster? Typically, Import/Export runs in two_task mode. To make it run faster, you can relink the programs in single-task mode. This can display significant speed improvements, but it does come with a cost. It can require significantly more memory to run these relinked programs. You may need to weigh the trade-offs when using this speed before you wholeheartedly give your faith to the added speed.

The following code can be used to perform this relinking.

```
cd $ORACLE_HOME/rdbms/lib
make -f ins_rdbms.mk singletask
make -f ins_rdbms.mk expst
make -f ins_rdbms.mk impst
make -f ins_rdbms.mk sqlldrst
mv expst $ORACLE_HOME/bin/
mv impst $ORACLE_HOME/bin/
mv sqlldrst $ORACLE_HOME/bin/
```

Now you can use expst (export_singletask) and impst (import_singletask) instead of imp or exp.

Want Import/Export to run even faster? Why not try using Data Pump? In Oracle 10g, Oracle redesigned Import/Export as Oracle Data Pump. While Import/Export are still included with the shipment of Oracle 10g, Data Pump is usually more efficient. Where Import/Export can both run as client server applications, Data Pump acts as a job inside the database, using command-line syntax very much like its client server predecessors. Anyone who's ever run and needed to monitor an Export/Import operation knows that, unless you're actually watching from the machine where the operation started, the best you can do is send the output to a log file so you can watch the process indirectly. With Data Pump, it doesn't matter where you start the job. Because it's running in the database, you can log on and check the process from any other computer that has access to the database.

182 Everyday Oracle DBA

But think about it. Client server architecture is inherently less efficient than something that runs directly within the database without having to make external connections in any way except for the connection that has to exist to the DIRECTORY housing the output files. Another reason it runs faster is because it can be run in a parallel fashion. If you specify the Parallel option, you allow Oracle to dump data into four different parallel threads, which is much faster than single threading.

As for the directory, since the job runs inside the database, if you want the export to go to the file system, the first thing you have to do is create a database DIRECTORY object into which you can output the file, afterward granting access to the user or whomever will be doing these exports and imports.

```
CREATE OR REPLACE DIRECTORY myexport 'd:\';  
GRANT READ, WRITE ON DIRECTORY myexport to larry;
```

Once you've created the directory and granted read and write privileges to it to the export user, you can use the following export commands:

```
expdp larry/angel directory=myexport dumpfile=larry.dmp
```

As you can see, it's very similar to an export command.

You don't have to write the export out to a file, however, and honestly Data Pump is just as happy to export the database objects directly into a remote database over a SQL Net connection. Simply specify the option REMOTE with the connection string to the remote database and the process ends up like a once-and-it's-done replication job.

You can force the running Data Pump Export job into the background, and the messages will stop being sent to the screen, but the job will remain running inside the database. If you want to reattach to a job you forced into the background, you can do so with the command:

```
Expdpattac=<jobname>
```

As if using a stored procedure over a client server application isn't enough to get your performance appetite sated, you can make this run even better. While Data Pump leverages parallelism (inter table, and both intra- and inter-partition) to run load and unload processes, and also build and load package bodies, fully utilizing all available resources to maximize the throughput and thereby minimize the elapsed time of a job, for all of this to

Chapter 4: Database Tuning: Making It Sing **183**

take place efficiently the system must be well balanced with respect to CPU, memory, and I/O distribution. Any tuning you can do to the overall system, database, and server will help Data Pump perform more efficiently.

Want to make Data Pump as efficient as possible? Allow it to create multiple dump files when it exports or reads from multiple dump files, and when it imports and distributes those files over separate disks, letting the I/O be distributed. This allows the parallelism to occur in as rapid a manner as possible. The disks on which these files are located should also not be the same disks on which the target tablespaces for the import, nor the source tablespaces for the export, reside.

Setting the degree of parallelism at the database level to no more than 2X the CPU count will allow you to maximize the way that Data Pump spawns its jobs to distribute them across the system. Keep in mind, though, that as you increase the degree of parallelism that a job is allowed to make use of, you also increase the CPU usage memory consumption and I/O bandwidth necessary for the jobs to run. It's important when setting up jobs that whoever is setting the parameters on the import and export jobs not only makes sure there are sufficient resources available for the job but that regular operations be allowed to occur on the database. One caveat here though: the PARALLEL parameter is only available in the Enterprise Edition of Oracle 10g.

You can set the following initialization parameters to help the performance of your Data Pump export and import processes:

- DISK_ASYNCH_IO = TRUE
- DB_BLOCK_CHECKING = FALSE
- DB_BLOCK_CHECKSUM = FALSE
- PROCESSES (high enough for maximum parallelism)
- SESSIONS (high enough for maximum parallelism)
- PARALLEL_MAX_SERVERS (high enough for maximum parallelism)

Keep in mind that setting these parameters will have ramifications on the overall system. Also remember that you'll get different results when setting these parameters on different operating systems.

184 Everyday Oracle DBA

All Operating Systems

- Use proper file placement so I/O is spread evenly across disks. If possible, use RAID devices.
- A good design is key. Index appropriately and watch row chaining. Row chaining occurs whenever a user updates a row in a table in such a way that it can no longer completely fit in the original data block. The migration of part of the row from the original block to another block is called row chaining, and it can, if allowed to become excessive, cause a great deal of additional I/O (first, Oracle has to find the block where the row starts, then it has to find and retrieve the block where the chain continues).
- Monitor V\$SESSION_WAIT regularly to identify wait conditions. If the SEQ# column stops changing, the event is stuck.
- Monitor locking and latching (V\$LOCK, V\$LATCH, V\$LATCHHOLDER, and so on).
- If you're trying to implement distributed or federated databases, keep in mind that two phase commits are slower than a single instance commit.

Need to Speed Up Oracle on Windows?

When Windows is the operating system on which both the database and application are running, it's often necessary to speed up the way Oracle and Windows play together. Possible steps to achieve this include the following:

- If you have the authority, remove any protocols you know you don't use from the installed network software list, and then move those used most frequently to the top.
- Stop all unnecessary services on your machine. Do this one at a time and make sure you test so you're sure they're really unnecessary.
- Windows NT and Windows 2000 support asynchronous I/O, so use it. This not only optimizes your I/O operations, it spares you having to configure multiple database writers.

Chapter 4: Database Tuning: Making It Sing **185**

- In Windows, it's important to note that the default `DB_BLOCK_SIZE` is still 2K. Rarely does an application perform optimally with this setting. Fortunately, if you use the Database Configuration Assistant, Oracle will help you choose more optimal block sizes. If you don't use DBCA, check before you create your database and increase this setting to 4K or 8K (or even 16K) if required.
- If you have to use a screensaver, choose something simple. While the really cool high-resolution ones are neat, and the 3-D ones are entertaining to watch, it makes your database very uncomfortable if it has to share this much of the resources with something so unproductive.

Oracle 10g

The highly-touted Oracle 10g database that tunes itself brings with it some truly impressive tuning enhancements. I'm not sure this means that DBAs are irrelevant, but it does mean we have yet more impressive tools to help us be more effective and efficient.

Tracing Enhancements

One interesting enhancement to tracing that was brought to the table with Oracle 10g is less of a revolution than it is an evolution in tracing. It's now possible to turn on tracing for one or more sessions at a time, and simultaneously watch sessions as they are connected in order to help you follow their progress through the database.

This means you can more accurately pinpoint exactly where the session is at any given time and during any process, the amount of resources being consumed at that specific point in time (and by extension what the process in question is consuming), and where the session is having difficulty and needs tuning. This feature is particularly important if you're trying to tune in a multitier and/or multiuser environment (and honestly, a single-user database isn't much more useful than one without users... plus, a single-user database may not be a good application for Oracle) with an application where connection pooling is taking place. In these instances, depending on the application, it might be difficult, if not impossible, to find some of this information.

186 Everyday Oracle DBA

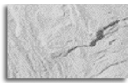
Automatic Performance Diagnostic and Tuning Features

While in the past it's been possible to automatically schedule statistics collection in Oracle 8i and Oracle 9i (this was true even when using CRON, AT, or DBMS_JOB), Oracle 10g brings with it not only the ability to automatically gather statistics but also its recommendation that you let Oracle automatically gather and maintain them for you. Oracle thus gathers statistics on all database objects in a maintenance job that you schedule to have run automatically. This "frees" you from having to worry about gathering these statistics on your own. In theory, this allows you the assurance of always having a reliable execution plan because you will never again have stale or missing statistics.

GATHER_STATS_JOB is the job that runs in order to automatically gather statistics on all objects in the database, which have either missing or stale statistics. The job is created automatically at database creation time and is managed by Scheduler. Scheduler (the free new Oracle 10g feature that enables you to schedule jobs from inside the database) runs GATHER_STATS_JOB during the maintenance window, which by default it assumes to be between 10 p.m. and 6 a.m. every day and all day on the weekends. These defaults, fortunately, can be changed.

While you can't change the schedule of GATHER_STATS_JOB by passing the job a parameter of when to run, you can change the window in which the job runs either by altering the Scheduler window, or by defining your own custom window in which you want it to run.

Scheduler comes with its own predefined windows (one for weeknights, and one for weekends). If these windows don't fit your needs, you can create your own windows instead. Windows have three attributes. Schedules controls when the window is in effect. Durations controls how long the window is open. Resource plans control the resource priorities among different job classes.



NOTE

It's important to remember that only one window can be used at any given time.

You can manipulate maintenance windows, adjusting their timing and attributes, as detailed in Table 4-5.

Chapter 4: Database Tuning: Making It Sing 187

Task	Procedure
Create a Window	CREATE_WINDOW
Open the Window	OPEN_WINDOW
Close the Window	CLOSE_WINDOW
Alter the Window	SET_ATTRIBUTE
Drop the Window	DROP_WINDOW
Disable the Window	DISABLE
Enable the Window	ENABLE

TABLE 4-5. *DBMS_SCHEDULER Procedures*

When you create a new window, you can specify the schedule for that window or you can create a window that points to a schedule that has been predefined and saved. The following code defines a window that is enabled at midnight, runs for five hours, and repeats every day. You can use your own defined resource plan with this window to handle the resource distribution during the maintenance window duration. Not specifying the resource plan means that the default plan will be used.

```
BEGIN
DBMS_SCHEDULE.CREATE_WINDOW (
WINDOW_NAME => 'nightly_window',
START_DATE => '01-JAN-05 12:00:00 AM',
REPEATE_INTERVAL => 'FREQ=DAILY',
RESOURCE_PLAN => 'my_maint_plan',
DURATION => interval '300' minute,
COMMENTS => 'nightly maintenance window');
END;
/
```

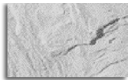
NOTE

Windows are created in the SYS schema. Scheduler doesn't check to see if there is something already defined for the given period of time. If it results in windows that overlap, the situation must be rectified.

188 Everyday Oracle DBA

You can disable the default windows, but it would be better not to drop them, because if there are maintenance jobs that rely on that window, you will disable those jobs as well, and it's never a good idea to delete or alter the default operators provided by Oracle. Plus, it's far quicker to re-enable any default windows than it is to re-create them.

Once the window is created, it has to be opened to be used. It can be opened automatically using the schedule that was defined when it was created, or it can be opened manually using `DBMS_SCHEDULER.OPEN_WINDOW`.



NOTE

Only an enabled window can be opened.

`GATHER_STATS_JOB`, once started, runs till completion even if it overruns the maintenance window. Stale statistics are those on objects which have had more than 10 percent of their rows modified.

`GATHER_STATS_JOB` calls the `GATHER_DATABASE_STATS_JOB_PROC` of the `DBMS_STATS` package, which operates the same way as the `DBMS_STATS.GATHER_DATABASE_STATS` procedure if you use the `GATHER AUTO` option. The primary difference between the two jobs is that the `GATHER_DATABASE_STATS_JOB_PROC` procedure can prioritize rather than serialize, so those objects that will benefit most from the procedure (in its opinion) will have their stats gathered first before the maintenance window closes. I haven't yet noticed places where its assumptions are wrong, thereby impacting the performance of queries against very large tables (that it's analyzing), so for now I'll assume its assumptions are valid.

In order to verify that automatic statistics gathering has been enabled, you can query the `DBA_SCHEDULER_JOBS` view as follows:

```
SELECT *  
FROM DBA_SCHEDULER_JOBS  
WHERE JOB_NAME = GATHER_STATS_JOB;
```

In order to disable automatic statistics gathering, simply run the Scheduler package as follows:

```
BEGIN  
  DBMS_SCHEDULER.DISABLE('GATHER_STATS_JOB');  
END;  
/
```

Chapter 4: Database Tuning: Making It Sing 189

While it may be that automatic statistics gathering is the next best thing since sliced bread, there are cases where automatic statistics gathering routinely overruns the overnight-defined batch window and thus highly volatile tables become stale during the day. This is most applicable to tables that are truncated and repopulated or that are deleted and then rebuilt during the course of the day or to objects which are the target of large bulk loads that add upwards of 10 percent or more of the object's total size in a given day.

You can, of course, still gather statistics for these and other kinds of objects using either the `GATHER_SCHEMA_STATS` or the `GATHER_DATABASE_STATS` procedures in the `DBMS_STATS` package.

It's important to note here, however, that none of these methods, `GATHER_SCHEMA_STATS`, `GATHER_DATABASE_STATS`, or automatic statistics gathering, collects statistics on external tables. To get statistics on these objects, you need to manually run or manually schedule `GATHER_TABLE_STATS` with the `ESTIMATE_PERCENT` option set explicitly to `NULL` (since sampling on external tables is not supported). Because data manipulation on external tables also isn't supported, it's sufficient to analyze external tables whenever the underlying OS file changes.

Need to find a way to restore previous versions of statistics? It's simpler now than ever before. Whenever statistics in a dictionary are modified, the older version is now automatically saved for the purpose of future restoring. Do you get the impression that automatic stats generation is sometimes not as optimal as it could be? These older versions of the statistics can be restored using `RESTORE` procedures from the `DBMS_STATS` package.

Want to prevent any new statistics from being gathered on a given object or set of objects but still want the ability to run automatic statistics gathering? You can lock the statistics on a table or schema using the `DBMS_STATS` package, too.

The `DBMS_STATS` package in Oracle 10g lets you lock your statistics on tables or on indexes, even if the data in the table changes. If you discover you have an efficient set of statistics that allows your application to perform well, you can use these packages to lock the statistics. It's important to note, however, that if you lock the statistics you cannot recalculate them until they're unlocked. `DBMS_STATS` employs four commands that allow it to lock and unlock statistics at a table or schema level: `LOCK_TABLE_STATS`, `LOCK_SCHEMA_STATS`, `UNLOCK_TABLE_STATS`, and `UNLOCK_SCHEMA_STATS`. Passing these procedures, the parameters that allow you to set and

190 Everyday Oracle DBA

unset locking at the level desired can ease a great deal of headaches caused when a set of statistics that allows an application to perform its best is overwritten by automatic statistics gathering.

Automatic SQL Tuning Automatic SQL tuning? Simply wave a magic wand and developers and end users don't have to think, the SQL just tunes itself? Now that *would* be an awesome new feature. Unfortunately, this feature is really just an advisor (the SQL Tuning Advisor) that takes one or more SQL statements as input parameters and then turns them around, telling you how it would create changes to make the SQL more optimal. Its output is simply advice and recommendations along with the rationale it used for each suggestion. It also tells you what it expects the benefit of its recommended changes to be. It may recommend collecting statistics on objects, creating new indexes, or even creating a new SQL Profile to be used when running the given statement or statements. The user can then choose to use the recommendations or not. While it isn't likely to tell you to gather better statistics or use a materialized view, it can tell you whether you should restructure your SQL statement.

A new database object called a SQL Tuning Set (STS) comes with the SQL Tuning Advisor. These new structures can be created manually at the command line or by using OEM. An STS stores the SQL statement along with the execution context surrounding that SQL statement.

The inputs for the SQL Tuning Advisor can come from the Automatic Database Diagnostic Monitor (ADDM), which is often its primary source. ADDM is the Oracle-provided (for an extra license fee) utility that analyzes the data in the Automatic Workload Repository (AWR, also available for the same additional license fee). AWR is a repository for raw system statistics and object data. ADDM runs, by default, once every hour to search through the repository to find particularly high resource-intensive SQL. If it finds one, it will recommend you run the SQL Tuning Advisor. This is, however, a somewhat less proactive approach since it has to wait till the statement has already been run before it can suggest tuning.

Alternatively, you can provide it with your own SQL statements. If you choose this proactive approach, you can include any not-yet-implemented statements, but you also have to manually create an STS so the SQL Tuning Advisor has input on which to work. This is, honestly, the option that I prefer to use. Proactive rather than reactive. Tuning before users have a chance to get angry over poorly running code.

Chapter 4: Database Tuning: Making It Sing 191

You can control the scope and durations of any given SQL Tuning Advisor task. If you choose the limited option, the advisor provides recommendations based on statistics checks, access path analysis, and SQL structure analysis, and no SQL Profile recommendations are generated. If you choose the comprehensive option, the advisor carries out all of the analysis available to it under the limited option, adding any SQL Profile recommendations. You can limit the duration of the advisor run by giving it a set time. The default time limit is 30 minutes.

The output of the advisor is advice on optimizing the execution plan, the advisor's rationale for the proposed optimization, the estimated performance benefit, and the command to implement the advice given. When it's finished, all you have to do is decide whether or not to implement the recommendations.

End to End Application Tracing End to End Application Tracing is a tool that helps simplify the inherently complex process of performance problem diagnosis, particularly in multitier environments. When user requests are routed to different database sessions by a middle-tier environment, it can mean that you lose the ability to directly attribute a session, definitively, to a given user. This can make using tools like 10046 trace difficult to use. End to End Application Tracing makes use of a unique client identifier to help trace a specific end-client's session and show what it's doing through all of the tiers to the database server.

Just like the other tools in your toolbox, this feature can help you determine where there is excessive workload, where SQL statements are performing less than optimally, and can provide you with information you can then use to contact the appropriate user to help determine what issues he or she is having. This can mean proactive tuning rather than reactive, and thus turn you in the eyes of users from a troll into a wizard. Even if you don't have time to sit around poking about your database checking if there's anything less than optimal happening, you can still use this feature as a means to troubleshoot an end user's issue when that user calls with a problem.

Issues can be identified by client identifier (the end user's ID), service (a group of applications with common attributes or a single application), module (a functional block of application programs within an application), or even the action being performed (INSERT, UPDATE, or DELETE within a module).

After tracing information is written to the trace files by End to End Application Tracing, you can use TRCSESS to help you diagnose the problem, and then hand off that file to TKPROF.

192 Everyday Oracle DBA

When I wear the hat of an APPS DBA, I can see many uses for End to End Application Tracing. While Oracle's E-Business Suite 11*i* actually has its own 10046 trace interface built in (which can become a pain when users "forget" to turn it off after they've traced what they're having issues with), it can sometimes be more bothersome than trying to enable tracing on a session-by-session basis because it's often the case that users will turn on tracing with binds and waits and then forget to turn them off. If this happens, you can find yourself overrun with trace files for sessions you have no need or desire to trace.

Automatic SGA Memory Management Automatic SGA Memory Management (ASMM) was created as a means to help simplify configuration of the database's System Global Area (SGA) and its parameters. It does this through the use of self-tuning algorithms. It's a really interesting concept that works the majority of the time, but somehow it still makes me feel like maybe my database is starting to think it knows more than I do. This utility helps you simplify most database configurations by helping you make the most efficient utilization decisions regarding the available memory on your system. It goes a step further than a lot of the advisors from Oracle 9*i* and allows them to dynamically make many of the decisions on their own. In order for ASMM to work correctly, you have to have some initialization parameters set correctly. SGA_TARGET must be changed to a nonzero value and should be set to the amount of memory you want to have dedicated to the SGA (see, you still have a say in the matter). STATISTICS_LEVEL has to be set to either TYPICAL or ALL. Once these are set up, the automatic SGA management makes decisions on how best to allocate that memory across the following pools (yeah, you guessed it, the ones we tinker with the most anyway):

- DATABASE BUFFER CACHE (the default cache, not the nondefault sized caches, the recycle, or the keep caches)
- SHARED POOL
- LARGE POOL
- JAVA POOL

If you've already tinkered with any of these, and they were set to non-zero values, those values are used as the minimum levels on which the

Chapter 4: Database Tuning: Making It Sing 193

ASMM bases its decisions (again, you still have a say in the matter). It's important to keep in mind with this utility that if you know you have an application that has a minimum requirement for any of these parameters in order for it to function properly, set them upfront so ASMM doesn't make decisions that will end up shooting you in the foot.

The `SGA_TARGET` parameter is dynamic and can be changed with the `ALTER SYSTEM` command, while appropriate values are less than or equal to the value set for the `SGA_MAX_SIZE` parameter. Changes to `SGA_TARGET` automatically filter down to the appropriate tuned memory pools. Setting `SGA_TARGET` to 0 disables ASMM.

Dynamic Sampling Dynamic Sampling helps improve server performance by determining if there are (or might be) more accurate estimates for predicate selectivity and statistics for tables and indexes. The statistics for tables and indexes, in this brave new world, now include table block counts, applicable index block counts, table cardinalities, and relevant join column statistics. The CBO (since the RBO has now gone the way of the Atari and the Commodore 64) uses these more accurate estimates to better judge what `EXPLAIN PLAN` it will use for executing the given SQL.

You can make use of this feature to estimate the selectivity of a given single table where clause when the collected statistics cannot be used, or if they are likely to lead to significant errors in CBO estimation. You can allow it to guesstimate statistics for tables and indexes if there are no statistics available for those structures. And you can allow it to do the same for indexes and tables whose statistics are simply too far out of date for you to be comfortable in trusting.

This feature is controlled by the use of the `OPTIMIZER_DYNAMIC_SAMPLING` parameter. The default value for the parameter is 2, which is the lowest setting that can be used if you want to turn on dynamic automatic sampling so it can gather the necessary stats. Setting it to level 0 turns off dynamic sampling altogether.

Making It Sing

There are some interesting things that you can do to the structure of your data to trick your database into performing far better than it might otherwise. Setup and maintenance might be something you're less than enthusiastic about, but the benefits may well be worth it in the end.

194 Everyday Oracle DBA

Materialized Views

Materialized views are schema objects that are typically used for pre-computing complex formulas and storing the results, for summarizing and aggregating data, for replicating data, or for distributing copies of the data to locations other than the primary location in order to allow people to access the data where it's being used. All of these are excellent ways to speed up data access. What's the difference between a "regular" view and a materialized view? Good question. Regular views don't physically hold anything other than the definition of what is being sought. They are a grouping of complex queries into a single representation of what appears to be a table but which in reality contains no data until the view is accessed. Materialized views, on the other hand, are more like indexes. They take up space, physically hold data, and are usually used as a way to speed up queries.

How can materialized views assist you with performance? Using the setting where initialization parameters enable query rewrites, the Cost-Based Optimizer can be told it has the option of using materialized views to cut the cost of queries by redirecting certain queries (or even certain parts of queries) to the materialized view and thus improve query performance. The optimizer transparently rewrites the request (or even a part of the request) to use the materialized view instead of the base tables.

If you find out that one of the worst performing queries that runs frequently has a formula in it, you can materialize that query and allow the formula to run once, causing the query to run far faster every time it's run. For example, every month accounting runs the same query, with the only difference being the date they run the query for. And they don't just run it once; half a dozen people run it over and over during the first week of the month. Therefore, you could find the query, determine how best to run it for a given date range (month -1 would give you last month's data and you could even compute the previous quarter based on what your company's quarters are), create a materialized view on that query, and schedule the materialized view to refresh on demand, or as scheduled at 5 a.m. on the first day of the month.

Do you have users who only access a given subset of data and have to pull that subset across a dialup line to a laptop while they're on the road at a client site? Let's say your business distributes packaged foods to convenience stores. Larry, your candy salesman, is responsible for convenience stores in a tri-county area. He needs to be able to determine what is in stock, and figure the lead time on the candy that's in greatest demand for his area that

Chapter 4: Database Tuning: Making It Sing 195

isn't in stock currently. Why have his queries from his laptop or his mobile device search the entire database for that kind of information? If you know you have parts of your business that can be compartmentalized, why not take advantage of this information? Thus, you should create materialized views that are customized to each smaller line of business so that queries can run as rapidly as possible.

One of the most interesting uses of materialized views is their ability to pre-compute joins. Do you have a set of tables that are joined together all the time and always joined on the same columns? Are some of the queries that run with these joins resource hogs? Do users frequently complain about the query times associated with these table joins? Materialized views are great ways to free up resources and make users happy. Find tables that are joined frequently and then pre-compute the joins and store the results in a materialized view. They are optimal in data warehouses or in reporting systems. They have the potential to slow down a transactional system, particularly if you were to build them as "on commit refresh".

Oh sure, it's the best thing since sliced bread, but nothing is all good, right? There has to be something extra you have to do to the database to make it recognize these things and to know to use them. Well, of course there is. Oracle is a smart database, but you have to give it a clue that you want it to use some of its bells and whistles sometimes.

QUERY_REWRITE_ENABLED must be true. You can set this in the initialization file. This tells Oracle that it is allowed to let the CBO know about the materialized views and use them to answer queries. In Oracle 10g, the default is true.

QUERY_REWRITE_INTEGRITY is another initialization parameter that's used to determine how and when Oracle rewrites the query. You can control how fresh or stale the data in the materialized view can be in order for it to be a candidate for query rewrite.

The different kinds of materialized view integrity that you can set for your query rewrite are as follows:

- Enforced (this is the default) query rewrite will only be done if the view contains fresh data.
- Trusted query rewrite will be done if the view contains current and correct data.

196 Everyday Oracle DBA

- `STALE_TOLERATED` tells the CBO that it should trust that the materialized view is correct even if it isn't current. Thus, the query is rewritten.

Then you have to deal with the users (this is one of those times when, if you have a bunch of users, you might see the benefit of using roles so you can control groups of users). Grant `QUERY_REWRITE` to users who will be permitted to have their queries rewritten by the CBO. Or if you want everyone to be able to use this feature, grant it to public.

Let's work with the quintessential Scott/Tiger schema since it's something most DBAs are at least partly familiar with. Let's assume we're trying to compile a listing every month of the department name, the jobs in that department, and the sum of its salaries so you can track where money is going over time.

To accomplish this, you would ordinarily query the table every month using the following:

```
SELECT dname, job, sum(sal)
FROM emp e, dept d
WHERE e.deptno = d.deptno
GROUP BY dname, job
```

Now, with the Scott/Tiger schema, this query won't really be a big deal since it runs in seconds at worst. But what if there were more than two tables, and what if the tables were million- or multimillion row tables?

```
CREATE MATERIALIZED VIEW emp_dept_sum_mv
TABLESPACE MVTS
ENABLE QUERY REWRITE
AS SELECT dname, job, SUM(sal)
FROM emp e, dept d
WHERE e.deptno = d.deptno
GROUP BY dname, job;
```

You would need to be granted `CREATE MATERIALIZED VIEW` to run the preceding statement successfully. Now that you've created it, gather statistics and refresh the view using the following:

```
execute dbms_utility.analyze_schema('SCOTT', 'COMPUTE');
execute dbms_mview.refresh('emp_dept_sum_mv');
```

Now test it and see what happens when the original query is run.

Chapter 4: Database Tuning: Making It Sing 197

```
set autotrace on explain
SELECT dname,job,SUM(sal)
  FROM emp e, dept d
 WHERE e.deptno = d.deptno
 GROUP BY dname,job;
Execution Pla
-----
0 SELECT STATEMENT Optimizer=CHOOSE
1 0 TABLE ACCESS (FULL) OF 'EMP_DEPT_SUM_MV'
```

There are always trade-offs, just as with anything you use to speed up performance. Just like ice cream and chocolate really all do have calories and those are the trade-offs that you have to deal with if you want to enjoy them, and materialized views are no different. Granted, there are not calories in materialized views, but they consume space and you do have to make sure they're refreshed whenever you want to use them to rewrite queries and get the most current data.

Clusters

No, not like in Real Application Clusters (RACs) and not like a computer cluster, but more like the dictionary.com definition (<http://dictionary.reference.com/search?q=cluster>): *A group of the same or similar elements gathered or occurring closely together; a bunch.*

In Oracle, a cluster is a storage construct that provides you with an alternative method of storing each block of data in such a way that it can make some queries run much faster. It's made up of a group of tables that are typically queried together and have their data stored in shared data blocks. The candidate tables are grouped together because they share common columns on which joins are typically made, and where tables are most often queried together.

Given earlier releases of Oracle, there's also the concept of hash clusters, which allow the database to optimize data retrieval. Hash clusters provide an alternative to the traditional storage of tables and indexes, and can also be used when clustering tables together isn't an option. In a typical single table storage with an index, Oracle stores rows of data in the table according to key values contained in a separate index. To retrieve the data, Oracle has to access the index, determine the location of the data, and then load it. In a hash table, you define the hash cluster and then load your tables into it. As

198 Everyday Oracle DBA

you load the data, resulting hash values correspond to values determined by the hash function. Oracle uses its own hash function to generate a distribution of numeric values (hash values) based on the cluster key (which can be a single column or multiple columns in the hashed table). This can be faster for retrieval because an index-based retrieval takes at least two I/O operations (one for the index, one for the data) while a hash table retrieval takes one I/O operation to retrieve the data and none at all to determine where that row is located.

Because the data from the included tables are stored in the same data blocks, disk I/O may be significantly reduced when the clustered tables are joined in queries. If you know that you have tables that are nearly always queried together, you can define the cluster key column or group of columns that the cluster tables have in common. In an insurance company, this might be a claim number column in the claim table, invoice table, and payment table. By specifying the cluster key (and adding the tables to the cluster), the cluster key value is stored once rather than once for each table no matter how many rows in each table are associated with the cluster key. This means that not only are the queries quickened, you could also significantly reduce the storage necessary for related table and index values.

Again, there are trade-offs in clusters just like in ice cream, chocolate, and materialized views. Tables that are frequently queried independently of each other may not be as good a candidate for clustering. Also, because the rows are stored in the same data block, the tables involved in the cluster may not be good candidates if there are significant amounts of inserts, updates, and deletes occurring on the individual tables of the cluster.



NOTE

While retrieval is quicker with clusters, inserts and updates are somewhat more expensive timewise.

Looking at an insurance company example, it's important to note that, unless the claim number is stored in the individual rows in the invoice line item table—storing line_item with claim—invoice and payment won't make sense just because line_item and invoice are queried together most frequently. Therefore, you need to sit down and consider carefully the table decisions you should make concerning clusters when you decide to go this route. It may be more advantageous in this case to create two clusters, one for claim

Chapter 4: Database Tuning: Making It Sing **199**

and payment and one for invoice and line_item, and then take the I/O hit during those times when you query both invoice and payment in some combination. Good design concepts should always be considered when looking at any of these constructs. The biggest impact will always happen when good design meshes with cool new features.

Summary

Tuning can become an obsession, so it's a good idea to have a general goal in mind when starting out. Without a goal, how will you know when enough is enough?

There will always be more tuning that can be done; bottlenecks move, and when you have cleared one, another will raise its ugly head. It is like the circle of life, always turning, always moving from one place to the next. You could find yourself enjoying the adventure of tuning, or you could decide that you're a victim of obsessive tuning disorder. So buyer beware.

