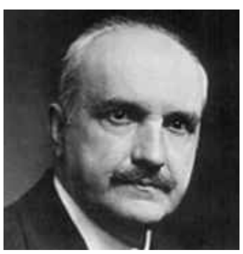
Predictive Modeling in Oracle

Oracle professionals are now applying the proven predicting modeling techniques from Oracle Data mining (ODM) and we are seeing Oracle professionals using special techniques to analyze database signatures and predict upcoming database stress. This excerpt has specific examples and scripts to get you started fast.

This is an excerpt from the bestselling book "Oracle Tuning: The Definitive Reference" (http://www.rampant-books.com/book_2005_1_awr_proactive_tuning.htm) by Alexey Danchenkov (http://www.wise-oracle.com/) and Donald Burleson (http://www.dba-oracle.com/books.htm), technical editor Mladen Gogala.

Predicting the Future with AWR

Predictive modeling is one of the best ways to perform long-term Oracle instance tuning, and the AWR tables are very helpful in this pursuit. In the predictive model of Oracle tuning, the DBA is charged with taking the existing AWR statistics and predicting the future needs for all instance and I/O areas within the Oracle database. For example, the AWR *physical reads* could be analyzed and compared to the memory usage within the Oracle *db_cache_size*. The information from the comparison could be extrapolated and used to predict the times at which the Oracle data buffers would need to be increased in order to maintain the current levels of performance.



Those who forget the past are condemned to repeat it.

George Santanaya

The DBA can also make a detailed analysis of Oracle's data buffer caches, including the KEEP pool, DEFAULT pool, the RECYCLE pool, and the pools for multiple block sizes like *db_32k_cache_size*. With that information, the DBA can accurately measure the performance of each one of the buffer pools, summarized by day-of-the-week and hour-of-the-day over long periods of time. Based upon existing usage, the DBA can accurately predict at what time additional RAM memory is needed for each of these data buffers.

The AWR tables also offer the DBA an opportunity to slice off the information in brand new ways. In the real world, all Oracle applications follow measurable, cyclical patterns called signatures. For example, an Oracle Financials application may be very active on the first Monday of every month when all of the books are being closed and the financial reports are being prepared. Using AWR data, information can be extracted from every first Monday of the month for the past year which will yield a valid signature of the specific performance needs of the end of the month Oracle financials applications.

Starting with Oracle8i, DBAs could dynamically change the Oracle database RAM regions and other instance parameters depending upon the performance needs of the applications. By making many initialization parameters alterable, Oracle is moving towards a dynamic database configuration, whereby the configuration of the system can be adjusted according to the needs of the Oracle application. The AWR can identify these changing needs.

With Oracle9i r2, there were three predictive utilities included with the standard STATSPACK report:

PGA advice: Oracle9i introduced an advisory utility dubbed *v$pga_target_advice*. This utility shows the marginal changes in optimal, one-pass, and multipass PGA execution for different sizes of *pga_aggregate_target*, ranging from 10% to 200% of the current value.

Shared Pool advice - This advisory functionality was extended in Oracle9i r2 to include an advice called *v$shared_pool_advice*.

Data Cache advice - The *v$db_cache_advice* utility shows the marginal changes in physical data block reads for different sizes of *db_cache_size*. The data from STATSPACK can provide similar data as *v$db_cache_advice*, and most Oracle tuning professionals use STATSPACK and *v$db_cache_advice* to monitor the effectiveness of their data buffers.

These advisory utilities are extremely important for the Oracle DBA who must adjust the sizes of the RAM areas to meet processing demands. The following *display_cache_advice.sql* query can be used to perform the cache advice function once the *v$db_cache_advice* has been enabled and the database has run long enough to give representative results.

display_cache_advice.sql

```
column c1   heading 'Cache Size (meg)'      format 999,999,999,999
column c2   heading 'Buffers'               format 999,999,999
column c3   heading 'Estd Phys|Read Factor' format 999.90
column c4   heading 'Estd Phys| Reads'      format 999,999,999

select
   size_for_estimate          c1,
   buffers_for_estimate       c2,
   estd_physical_read_factor  c3,
   estd_physical_reads        c4
from
   v$db_cache_advice
where
   name = 'DEFAULT'
and
   block_size  = (SELECT value FROM V$PARAMETER
                  WHERE name = 'db_block_size')
and
   advice_status = 'ON';
```

The output from the script is shown below. The values range from ten percent of the current size to double the current size of the *db_cache_size*.

```
                                Estd Phys    Estd Phys
 Cache Size (meg)     Buffers Read Factor        Reads
---------------- ------------ ----------- ------------
              30        3,802       18.70  192,317,943 <== 10% size
              60        7,604       12.83  131,949,536
              91       11,406        7.38   75,865,861
             121       15,208        4.97   51,111,658
             152       19,010        3.64   37,460,786
             182       22,812        2.50   25,668,196
             212       26,614        1.74   17,850,847
             243       30,416        1.33   13,720,149
             273       34,218        1.13   11,583,180
             304       38,020        1.00   10,282,475 Current Size
             334       41,822         .93    9,515,878
             364       45,624         .87    8,909,026
             395       49,426         .83    8,495,039
             424       53,228         .79    8,116,496
             456       57,030         .76    7,824,764
             486       60,832         .74    7,563,180
             517       64,634         .71    7,311,729
             547       68,436         .69    7,104,280
             577       72,238         .67    6,895,122
             608       76,040         .66    6,739,731 <== 2x size
```

From the above listing, it is clear that increasing the *db_cache_size* from 304 Megabytes to 334 Megabytes would result in approximately 700,000 less physical reads. This can be plotted as a 1/x function and the exact optimal point computed as the second derivative of the function 1/x as shown in Figure 11.1:

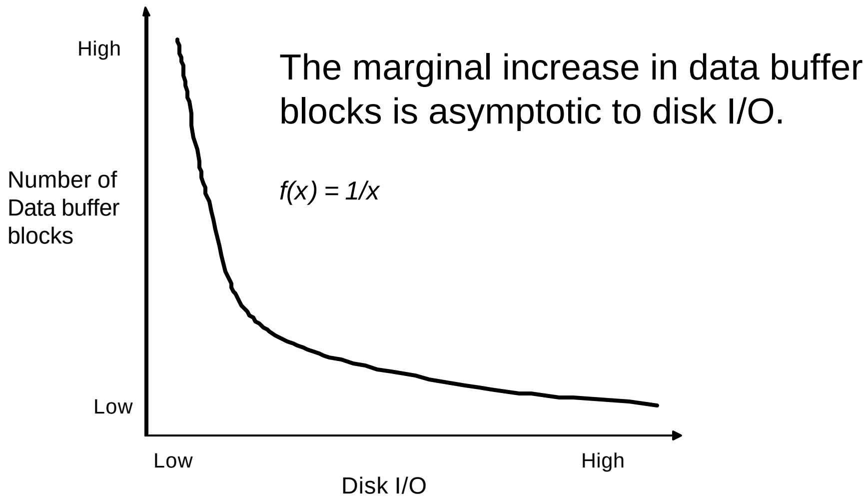


Figure 11.1: *The relationship between buffer size and disk I/O*

Once DBAs can recognize cyclic performance patterns in the Oracle database, they are in a position to reconfigure the database in order to meet the specific processing needs of that Oracle database.

While the predictive models are new, the technique dates back to Oracle6. Old-timer Oracle professionals would often keep several versions of their initialization parameter file and bounce in a new version when processing patterns were going to change.

For example, it was not uncommon to see a special Oracle instance configuration that was dedicated exclusively to the batch processing tasks that might occur on every Friday while another version of the *init.ora* file would be customized for OLTP transactions.

Additional *init.ora* files could be created that were suited to data warehouse processing that might occur on the weekend. In each of these cases, the Oracle database had to be stopped and restarted with the appropriate *init.ora* configuration file.

Starting with Oracle10g, the AWR tables can be used to identify all specific times when an instance-related component of Oracle is stressed, and the new *dbms_scheduler* package can be used to trigger a script to dynamically change Oracle during these periods. In sum, AWR data is ideally suited to work with the dynamic SGA features of Oracle10g.

Exception Reporting with the AWR

At the highest level, exception reporting involved adding a WHERE clause to a data dictionary query to eliminate values that fall beneath a pre-defined threshold. For a simple example, this can be done quite easily with a generic script to read *dba_hist_sysstat*.

The following simple script called *rpt_sysstat_10g.sql* displays a time-series exception report for any statistic in *dba_hist_sysstat*. The script accepts the statistics number and the value threshold for the exception report.

rpt_sysstat_10g.sql

```
prompt
prompt  This will query the dba_hist_sysstat view to display all values
prompt  that exceed the value specified in
prompt  the "where" clause of the query.
prompt

set pages 999

break on snap_time skip 2

accept stat_name   char   prompt 'Enter Statistic Name:  ';
accept stat_value  number prompt 'Enter Statistics Threshold value:  ';


col snap_time   format a19
col value       format 999,999,999

select
   to_char(begin_interval_time,'yyyy-mm-dd hh24:mi') snap_time,
   value
from
   dba_hist_sysstat
  natural join
   dba_hist_snapshot
where
   stat_name = '&stat_name'
and
  value > &stat_value
order by
   to_char(begin_interval_time,'yyyy-mm-dd hh24:mi')
;
SEE CODE DEPOT FOR MORE SCRIPTS
http://www.rampant-books.com/book_2005_1_awr_proactive_tuning.htm
```

Notice that this simple script will prompt you for the statistic name and threshold value; allowing ad-hoc AWR queries:

```
SQL> @rpt_sysatst

This will query the dba_hist_sysstat view to display all values
that exceed the value specified in
the "where" clause of the query.

Enter Statistic Name:  physical writes
Enter Statistics Threshold value:  200000

SNAP_TIME                  VALUE
------------------- ------------
2004-02-21 08:00         200,395
2004-02-27 08:00         342,231
2004-02-29 08:00         476,386
2004-03-01 08:00         277,282
2004-03-02 08:00         252,396
2004-03-04 09:00         203,407
```

The listing above indicates a repeating trend where physical writes seem to be high at 8:00 AM on certain days. This powerful script will allow the DBA to quickly extract exception conditions from any instance-wide Oracle metric and see its behavior over time.

The next section provides a more powerful exception report that compares system-wide values to individual snapshots.

Exception reporting with *dba_hist_filestatxs*

The new 10g *dba_hist_filestatxs* table contains important file level information about Oracle I/O activities. Because most Oracle databases perform a high amount of reading and writing from disk, the *dba_hist_filestatxs* view can be very useful for identifying high use data files.

For Oracle10g customers who are not using the Stripe and Mirror Everywhere (SAME) approach, this view is indispensable for locating and isolating hot data files. Many Oracle shops will isolate hot data files onto high-speed solid-state disk (SSD), or relocate the hot files to another physical disk spindle.

If the *dba_hist_filestatxs* is described as shown in Table 11.1, the important information columns can be seen below. The important information relates to physical reads and writes, the actual time spent performing reads and writes, and the wait count associated with each data file, for each snapshot.

COLUMN	DESCRIPTION
snap_id	Unique snapshot ID
filename	Name of the datafile
phyrds	Number of physical reads done
phywrts	Number of times DBWR is required to write
singleblkrds	Number of single block reads
readtim	Time, in hundredths of a second, spent doing reads if the *timed_statistics* parameter is TRUE; 0 if *timed_statistics* is FALSE
writetim	Time, in hundredths of a second, spent doing writes if the *timed_statistics* parameter is TRUE; 0 if *timed_statistics* is FALSE
singleblkrdtim	Cumulative single block read time, in hundredths of a second
phyblkrd	Number of physical blocks read
phyblkwrt	Number of blocks written to disk, which may be the same as *phywrts* if all writes are single blocks
wait_count	Wait Count

Table 11.1: *The metrics relating to file I/O in dba_hist_filestatxs*

It is easy to write a customized exception report with AWR data. In this simple report called *hot_write_files_10g.sql*, the *dba_hist_filestatxs* table is queried to identify hot write datafiles, which is any condition where any individual file consumed more than 25% of the total physical writes for the whole instance. Especially when you are not using RAID, identification of hot datafiles is important because the objects inside the file cache can be cached with the KEEP pool or by moving the hot data file onto high-speed solid-state RAM disks.

The query below compares the physical writes in the the *phywrts* column of *dba_hist_filestatxs* with the instance-wide physical writes *statistic#* = 55 from the *dba_hist_sysstat* table.

This simple yet powerful script allows the Oracle professional to track hot-write datafiles over time, thereby gaining important insight into the status of the I/O sub-system over time.

hot_write_files_10g.sql

```
prompt  This will identify any single file who's write I/O
prompt  is more than 25% of the total write I/O of the database.
prompt

set pages 999

break on snap_time skip 2

col filename      format a40
col phywrts       format 999,999,999
col snap_time     format a20

select
   to_char(begin_interval_time,'yyyy-mm-dd hh24:mi') snap_time,
   filename,
   phywrts
from
   dba_hist_filestatxs
natural join
   dba_hist_snapshot
where
   phywrts > 0
and
   phywrts * 4 >
(
select
   avg(value)               all_phys_writes
from
   dba_hist_sysstat
  natural join
   dba_hist_snapshot
where
   stat_name = 'physical writes'
and
  value > 0
)
order by
   to_char(begin_interval_time,'yyyy-mm-dd hh24:mi'),
   phywrts desc
;
SEE CODE DEPOT FOR MORE SCRIPTS
http://www.rampant-books.com/book_2005_1_awr_proactive_tuning.htm
```

The following is the sample output from this powerful script. This is a useful report because the high-write datafiles are identified as well as those specific times at which they are hot.

```
SQL> @hot_write_files

This will identify any single file who's write I/O
is more than 25% of the total write I/O of the database.


SNAP_TIME          FILENAME                                   PHYWRTS
----------------- ---------------------------------------- --------
2004-02-20 23:30  E:\ORACLE\ORA92\FSDEV10G\SYSAUX01.DBF    85,540

2004-02-21 01:00  E:\ORACLE\ORA92\FSDEV10G\SYSAUX01.DBF    88,843

2004-02-21 08:31  E:\ORACLE\ORA92\FSDEV10G\SYSAUX01.DBF    89,463

2004-02-22 02:00  E:\ORACLE\ORA92\FSDEV10G\SYSAUX01.DBF    90,168

2004-02-22 16:30  E:\ORACLE\ORA92\FSDEV10G\SYSAUX01.DBF   143,974
                  E:\ORACLE\ORA92\FSDEV10G\UNDOTBS01.DBF   88,973
```

This type of time-series exception reporting is extremely useful for detecting those times when an Oracle database is experiencing I/O stress. Many Oracle professionals will schedule these types of exception reports for automatic e-mailing every day. We can also use AWR to aggregate this information to spot trends.

Now that the concept of trend identification has been introduced, it is time to move onto an examination of a more sophisticated type of report where repeating trends within the data can be identified.

General trend identification with the AWR

Once the *dba_hist* scripts have been mastered, the next step is to look at the more complex task of trend identification with the AWR tables. By now, it should be clear that aggregating important Oracle performance metrics over time, day-of-the-week and hour-of-the-day, allows the DBA to see the hidden signatures. These signatures are extremely important because they show regularly occurring changes in processing demands. This knowledge allows the DBA to anticipate upcoming changes and reconfigure Oracle just-in-time to meet the changes.

The following is a simple example. The *rpt_sysstat_hr_10g.sql* script will show the signature for any Oracle system statistic, averaged by hour of the day, and figure 11.2 shows a typical output of the script.

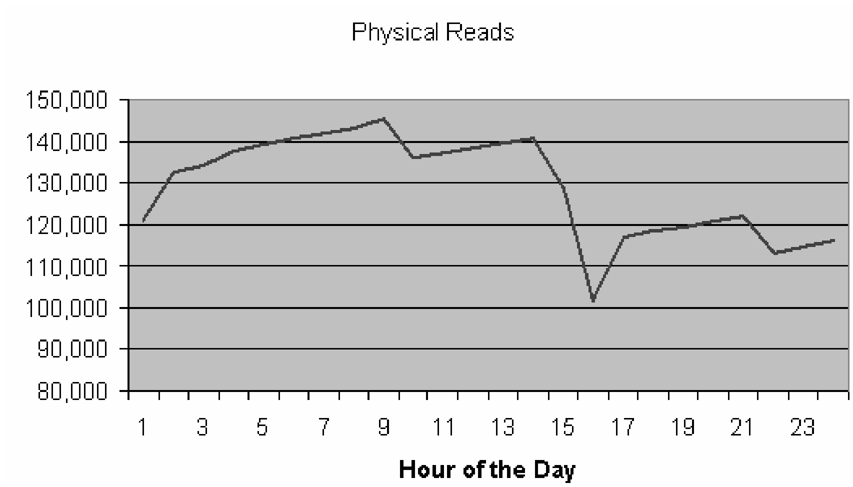


Figure 11.2: *An hourly Signature for physical disk reads*

Plotting the data makes it easy to find trends. Of course, open source products such as RRDTool can also be used to automate the plotting of data from the AWR and ASH tables and make nice web screens to see the data. Finally, the WISE tool (http://www.wise-oracle.com/product_wise_enterprise.htm) can be used and with just a few clicks, comprehensive charts can be produced for any snapshot period as well as trend charts for month, day, or hourly periods. Figure 11.3 below shows a sample WISE view:

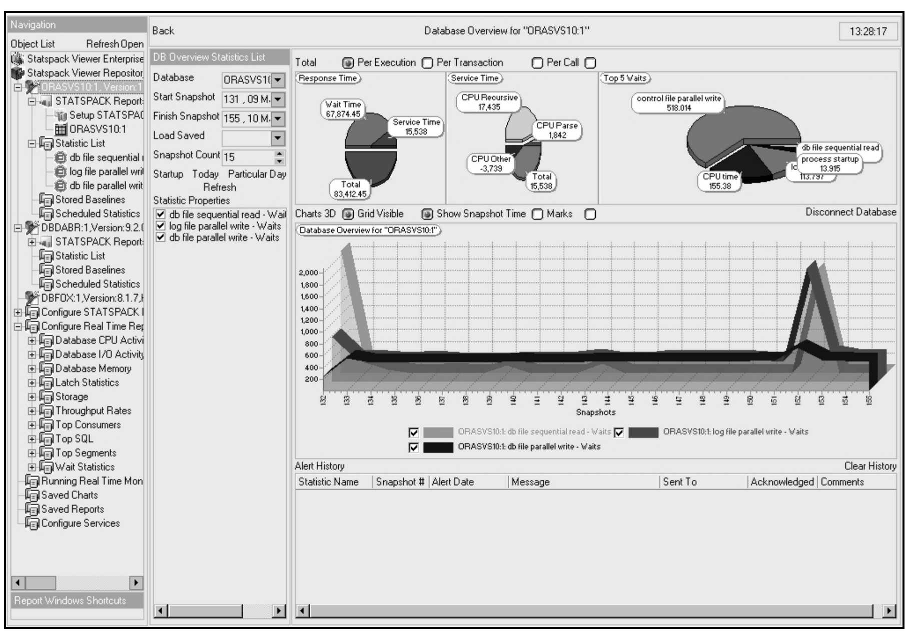


Figure 11.3: *Time series charts in* The WISE tool (http://www.wise-oracle.com/product_wise_enterprise.htm)

The same types of reports, aggregated by day-of-the week, can be created to show daily trends. Over long periods of time, almost all Oracle databases will develop distinct signatures that reflect the regular daily processing patterns of the end-user community.

The *rpt_10g_sysstat.sql* script, that was introduced in Chapter 2, will accept any of the values from *dba_hist_sysstat*. This data can now be plotted for trend analysis as shown in Figure 11.4 below. These types of signatures will become very stable for most Oracle databases and can be used to develop a predictive model for proactive tuning activities.

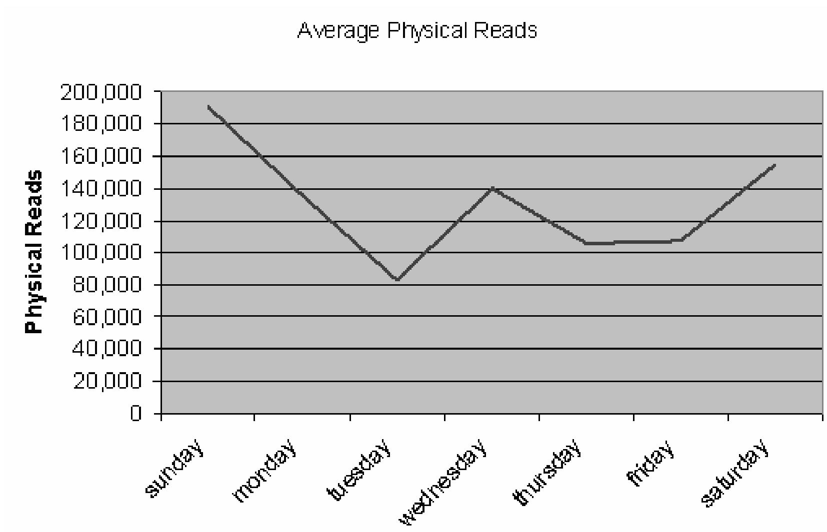


Figure 11.4: *The signature for average physical reads by day of the week*

Correlation analysis with AWR and ASH

For those who like Oracle tuning with the Oracle Wait Interface (OWI), there are interesting statistics that relate to system-wide wait events from the *dba_hist_waitstat* table as shown in Table 11.2 that provide detailed wait event information from the *dba_hist_active_sess_history*.

COLUMN	DESCRIPTION
snap_id	Unique snapshot ID
dbid	Database ID for the snapshot
instance_number	Instance number for the snapshot
class	Class of the block
wait_count	Number of waits by the OPERATION for this CLASS of block
time	Sum of all wait times for all the waits by the OPERATION for this CLASS of block

Table 11.2: *The dba_hist_waitstat statistics used to wait event analysis*

To understand correlation analysis for an Oracle database, a simple example may be helpful. For advanced correlation analysis, we seek to identify correlations between instance-wide wait events and block-level waits. This is a critical way of combining human insight with the AWR and ASH information to isolate the exact file and object where the wait contention is occurring.

The ASH stores the history of a recent session's activity in *v$active_session_history* with the AWR history view *dba_hist_active_sess_history*. This data is designed as a rolling buffer in memory, and earlier information is overwritten when needed. To do this, the AWR *dba_hist_active_sess_history* view is needed. This view contains historical block-level contention statistics as shown in Table 11.3 below.

COLUMN	DESCRIPTION
snap_id	Unique snapshot ID
sample_time	Time of the sample
session_id	Session identifier
session_serial#	Session serial number. This is used to uniquely identify a session's objects.
user_id	Oracle user identifier
current_obj#	Object ID of the object that the session is currently referencing
current_file#	File number of the file containing the block that the session is currently referencing
current_block#	ID of the block that the session is currently referencing
wait_time	Total wait time for the event for which the session last waited (0 if currently waiting)
time_waited	Time that the current session actually spent waiting for the event. This column is set for waits that were in progress at the time the sample was taken.

Table 11.3: *Selected columns from the dba_hist_active_sess_history view*

The *wait_time_detail.sql* script below compares the wait event values from *dba_hist_waitstat* and *dba_hist_active_sess_history*. This script quickly allows the identification of the exact objects that are experiencing wait events.

wait_time_detail_10g.sql

```
-- *************************************************
-- Copyright © 2005 by Rampant TechPress
-- This script is free for non-commercial purposes
-- with no warranties.  Use at your own risk.
-- *************************************************

prompt
prompt  This will compare values from dba_hist_waitstat with
prompt  detail information from dba_hist_active_sess_history.
```

```
prompt

set pages 999
set lines 80

break on snap_time skip 2

col snap_time     heading 'Snap|Time'    format a20
col file_name     heading 'File|Name'    format a40
col object_type   heading 'Object|Type'  format a10
col object_name   heading 'Object|Name'  format a20
col wait_count    heading 'Wait|Count'   format 999,999
col time          heading 'Time'         format 999,999

select
   to_char(begin_interval_time,'yyyy-mm-dd hh24:mi') snap_time,
--   file_name,
   object_type,
   object_name,
   wait_count,
   time
from
   dba_hist_waitstat            wait,
   dba_hist_snapshot            snap,
   dba_hist_active_sess_history ash,
   dba_data_files               df,
   dba_objects                  obj
where
   wait.snap_id = snap.snap_id
and
   wait.snap_id = ash.snap_id
and
   df.file_id = ash.current_file#
and
   obj.object_id = ash.current_obj#
and
   wait_count > 50
order by
   to_char(begin_interval_time,'yyyy-mm-dd hh24:mi'),
   file_name
;
SEE CODE DEPOT FOR MORE SCRIPTS
http://www.rampant-books.com/book_2005_1_awr_proactive_tuning.htm
```

This script is also enabled to join into the *dba_data_files* view to get the file names associated with the wait event. This is a very powerful script that can be used to quickly drill-in to find the cause of specific waits. Below is sample output showing time-slices and the corresponding wait counts and times:

```
SQL> @wait_time_detail_10g

Copyright 2004 by Donald K. Burleson

This will compare values from dba_hist_waitstat with
detail information from dba_hist_active_sess_hist.


Snap                 Object     Object           Wait
Time                 Type       Name            Count    Time
-------------------- ---------- ------------ -------- -------
2004-02-28 01:00     TABLE      ORDOR           4,273      67
                     INDEX      PK_CUST_ID     12,373     324
                     INDEX      FK_CUST_NAME    3,883      17
                     INDEX      PK_ITEM_ID      1,256     967


2004-02-29 03:00     TABLE      ITEM_DETAIL        83      69
```

```
2004-03-01 04:00     TABLE      ITEM_DETAIL    1,246      45


2004-03-01 21:00     TABLE      CUSTOMER_DET   4,381     354
                     TABLE      IND_PART         117      15


2004-03-04 01:00     TABLE      MARVIN        41,273       16
                     TABLE      FACTOTUM       2,827       43
                     TABLE      DOW_KNOB         853        6
                     TABLE      ITEM_DETAIL       57      331
                     TABLE      HIST_ORD       4,337      176
                     TABLE      TAB_HIST         127       66
```

This example demonstrates how the AWR and ASH data can be used to create an almost infinite number of sophisticated custom performance reports.

The AWR can also be used with the Oracle Data Mining (ODM) product to analyze trends. Using Oracle ODM, the AWR tables can be scanned for statistically significant correlations between metrics. Sophisticated multivariate Chi-Square techniques can also be applied to reveal hidden patterns within the AWR treasury of Oracle performance information.

The Oracle10g ODM uses sophisticated Support Vector Machines (SVM) algorithms for binary, multi-class classification models and has built-in linear regression functionality.

Conclusion

If the DBA takes the time to become familiar with the wealth of metrics within the AWR and ASH tables, it becomes easy to get detailed correlation information between any of the 500+ performance metrics captured by the AWR.

As the Oracle database evolves, Oracle will continue to enhance the mechanisms for analyzing the valuable performance information in AWR. At the present rate, future releases of Oracle may have true artificial intelligence built-in to detect and correct even the most challenging Oracle optimization issues.

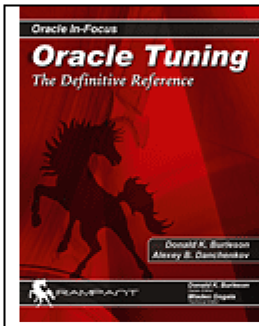
The AWR provides the foundation for sophisticated performance analysis, including exception reporting, trend analysis, correlation analysis, hypothesis testing, data mining, and best of all the ability to anticipate future stress on the database.

The main points of this chapter include:

- The AWR *dba_hist* views are similar to well-known STATSPACK tables, making it easy to migrate existing performance reports to Oracle10g.
- The *dba_hist* views are fully documented and easy to use for writing custom scripts.
- The creation of AWR and ASH provides a complete repository for diagnosing and fixing any Oracle performance issue.

- The AWR and ASH are the most exciting performance optimization tools in Oracle's history and provide the foundation for the use of artificial intelligence techniques to be applied to Oracle performance monitoring and optimization.
- As Oracle evolves, the AWR and ASH will likely automate the tedious and time consuming task of Oracle tuning.

Now that the basic idea behind proactive time-series and correlation analysis has been revealed, the next step is to take a look at how the AWR and ASH data can be used to monitor external server conditions. Oracle10g shows that Oracle recognizes that the server hardware is critical to Oracle performance and offers many exciting tools to help the DBA with tuning.

	This is an excerpt from the bestselling book "Oracle Tuning: The Definitive Reference" (http://www.rampant-books.com/book_2005_1_awr_proactive_tuning.htm) by Alexey Danchenkov (http://www.wise-oracle.com/) and Donald Burleson (http://www.dba-oracle.com/books.htm), technical editor Mladen Gogala. Incorporating the principles of artificial intelligence, Oracle10g has developed a sophisticated mechanism for capturing and tracking database performance over time periods. This new complexity has introduced dozens of new v$ and DBA views, plus dozens of Automatic Workload Repository (AWR) tables.

The AWR and its interaction with the Automatic Database Diagnostic Monitor (ADDM) is a revolution in database tuning. By understanding the internal workings of the AWR tables, the senior DBA can develop time-series tuning models to predict upcoming outages and dynamically change the instance to accommodate the impending resource changes.

This is not a book for beginners. Targeted at the senior Oracle DBA, this book dives deep into the internals of the v$ views, the AWR table structures and the new DBA history views. Packed with ready-to-run scripts, you can quickly monitor and identify the most challenging performance issues.