

Optimizing Performance With STATSPACK



By Tim Gorman

Members of Oracle's Server Technology Development group created the STATSPACK package in the late 1980s. It was not made generally available until version 8.1.6 of Oracle 8i as the successor utility to the venerable BSTAT/ESTAT report. The purpose of this paper is to provide a better understanding of STATSPACK.



Introduction

In replacing BSTAT/ESTAT, STATSPACK goes much further yet in many ways not far enough. The purpose of this paper is to describe:

- Understanding what STATSPACK is, what it can, and cannot do
- How to install it properly
- How to use it to optimize performance in the database
- What its shortcomings are

STATSPACK is intended for tuning the performance of an Oracle database, regardless of what other tuning tools are in use or what application is using the database. It is a repository for the *dynamic performance* information generated by many of Oracle's internal V\$ views. This repository is merely an extension of the V\$ views, most of which are based completely in memory and thus cannot retain data across a restart of the Oracle database instance. Moreover, the STATSPACK repository consists of *point-in-time* snapshots of the data in the V\$ views, so the amount of change between snapshots is retained, not just the current cumulative value. This allows a database administrator (DBA) to pinpoint time periods in which changes occurred and the rate of change.

Available Documentation

There is a lot of documentation on STATSPACK available from a variety of sources, including books, online resources, and standard documentation.

Standard Oracle Documentation Sets (OTN)

The Oracle 8i standard documentation set does not contain any information on STATSPACK. Only the README files provided with the STATSPACK software were available. MetaLink note #149115.1 provides information on the location of these files on different platforms. In general, all of the STATSPACK files including the README files reside in the RDBMS/LIB subdirectory of ORACLE_HOME. In v8.1.6, this file was named statpack.doc but for all subsequent versions it is called spdoc.txt. In the early v8.1.6 and v8.1.7 versions, even these files were not shipped on OpenVMS and OS/390, but they are a standard part of all UNIX and WINDOWS installations.

The spdoc.txt text file is a **must-read** for anyone using STATSPACK for the first time.

The standard Oracle 9i reference *Database Performance Tuning Guide and Reference* (part #96533-01) has a chapter on STATSPACK, chapter 21. It is quite complete and well worth reading, largely augmenting what is found in the spdoc.txt document.

Oracle's entire standard documentation set can be viewed online or downloaded from <http://otn.oracle.com>. From the start page, click on the left-hand navigation for "Documentation" and follow the references to the book named above. The book can either be viewed or searched online in HTML or you can download the PDF for saving locally or printing.

All of these standard documentation sources are absolutely free, as both software and documentation can be downloaded for free from OTN.

Books

The best online prices for technical Oracle books can be found at <http://www.bookpool.com>, but of course <http://www.amazon.com> is very reliable. A recommended book is *Don Burleson's, Oracle 9i High-Performance Tuning With STATSPACK*, ISBN #007222360X, published by Osborne McGraw-Hill.

I have not yet read this book, as I prefer to use MetaLink and online sources for documentation.

Online Sources

Lots of information on STATSPACK is best found online:

Oracle support's MetaLink web site

- Note #146560.1 *FAQ For Database Performance* is a good overview of performance tuning in a cookbook format, which mentions STATSPACK.
- Note #94224.1 *FAQ – STATSPACK Complete Reference* is an excellent starting point, mentioning numerous other articles on STATSPACK using an FAQ format. All other MetaLink articles mentioned here are not included in this FAQ.
- Note #223117.1 *Tuning I/O related waits* makes reference to the *wait events* statistics found in the standard STATSPACK report.

continued on page 26

- Note #209197.1 *Using STATSPACK to Record EXPLAIN PLAN Details* describes new Oracle 9i functionality with STATSPACK, namely the gathering of V\$SQL_PLAN information.
- Note #213725.1 *RAC Survival Kit – Troubleshooting Performance Issues* is a description of the use of STATSPACK for 8iOPS and 9iRAC environments.

Of course, resources on MetaLink are being added all the time. To stay up-to-date, periodically search on the keyword STATSPACK in MetaLink generally yield at least two-dozen *notes* and countless *forums* to browse.

Other Web sites

Entering the keywords *oracle* and *statspack* into the <http://www.google.com> search engine yields several hundred *bits*, the most notable being:

- <http://www.oracle.com/oramag/oracle/00-Mar/index.html?o20tun.html> - Oracle Magazine article in March 2000 by Connie Dialeris and Graham Wood, introducing Oracle 8i STATSPACK to the masses for the first time
- <http://www.dba-oracle.com/oramag/Statspack%20Trend%20Analysis.htm> - Don Burleson's article on trend analysis using STATSPACK, including SQL*Plus source for a threshold-driven STATSPACK Alert report
- <http://www.geocities.com/alexadabr/index.html> - STATSPACK Viewer software providing an easy-to-use graphical user interface (GUI) for analyzing STATSPACK data. Comes with a *standard* and a *professional* edition and a 30-day trial evaluation version.
- <http://www.oraperf.com> - Anjo Kolk's manifestation of his YAPP (Yet Another Performance Profiler) methodology that is the foundation of all current Oracle performance tuning methodologies, including Oracle Support's own COE Performance Methodology (CPM). Upload your STATSPACK report to the YAPP engine on this web site for an excellent analytical report. The YAPP report provided from this web site is the single most important resource available for properly analyzing the standard STATSPACK report.

Also, I keep several STATSPACK-related SQL*Plus scripts on my personal web site at <http://www.EvDBT.com/tools.htm>. All of my STATSPACK-related scripts start with the prefix *sp*, so be aware of that while browsing through the alphabetical listing.

Understanding What STATSPACK can and Cannot do

STATSPACK is written entirely in PL/SQL. It consists of three main components:

- The STATSPACK package with the SNAP procedure. This procedure takes *point-in-time snapshots* of current data in the cumulative V\$ views and saves each snapshot's information to permanent tables in the STATSPACK repository
- The STATSPACK repository, which consists of forty-one (41) tables in Oracle 9i and twenty-eight (28) tables in Oracle 8i
- Queries and reports, for analyzing the data in the repository

Because it is written in PL/SQL, it is portable across all platforms – the same software runs on all flavors of UNIX, on all flavors of Windows, on OpenVMS, on OS/390, and on all 70+ computing platforms on which Oracle operates.

Although any job-scheduling package can be used, STATSPACK is designed to use Oracle's own internal DBMS_JOB job-scheduling package. It is an ideal fit, as DBMS_JOB doesn't run unless the database instance is open and available, and that is the only time STATSPACK can be used also.

Because STATSPACK only *samples* the data in the V\$ views *periodically*, it

cannot capture everything. The further apart the *samples*, the less capable it will be at finding anomalous behavior in the performance statistics. In scheduling snapshots, taking them closer together is better for analysis and reporting purposes, but gathering data too frequently may begin to impact performance and cause the STATSPACK repository to grow faster and larger. Furthermore, analyzing and reporting upon a larger STATSPACK repository may consume more resources, possibly further impacting the performance of the database that you are monitoring.

Luckily, STATSPACK is capable of monitoring itself. As you begin to analyze performance, be alert for SQL statements belonging to the STATSPACK.SNAP procedure or your reporting queries showing up in your own reports.

By default, when STATSPACK is installed to use the DBMS_JOB package for scheduling, using the "spauto.sql" script, STATSPACK gathers data once per hour. This is an excellent default setting. During special situations, you may want to reduce that frequency to every 30 minutes or every 15 minutes, but running SNAP any more often is not likely to be helpful.

From V\$ view	To Oracle 8i Statspack table	To Oracle 9i Statspack table	Snap level
V\$BUFFER_POOL_STATISTICS	STATS\$BUFFER_POOL_STATISTICS	STATS\$BUFFER_POOL_STATISTICS	>= 0
V\$DB_CACHE_ADVICE	--	STATS\$DB_CACHE_ADVICE	>= 0
V\$DLM_MISC	--	STATS\$DLM_MISC	>= 0
X\$KSQST / V\$ENQUEUE_STAT	STATS\$ENQUEUESTAT	STATS\$ENQUEUE_STAT	>= 0
V\$FILESTAT	STATS\$FILESTATXS	STATS\$FILESTATXS	>= 0
V\$INSTANCE_RECOVERY	--	STATS\$INSTANCE_RECOVERY	>= 0
V\$LIBRARYCACHE	STATS\$LIBRARYCACHE	STATS\$LIBRARYCACHE	>= 0
V\$PARAMETER	STATS\$PARAMETER	STATS\$PARAMETER	>= 0
V\$PGASTAT	--	STATS\$PGASTAT	>= 0
V\$PGA_TARGET_ADVICE	--	STATS\$PGA_TARGET_ADVICE	>= 0
V\$RESOURCE_LIMIT	--	STATS\$RESOURCE_LIMIT	>= 0
V\$ROLLSTAT	STATS\$ROLLSTAT	STATS\$ROLLSTAT	>= 0
V\$ROWCACHE	STATS\$ROWCACHE_SUMMARY	STATS\$ROWCACHE_SUMMARY	>= 0
V\$SESSION_EVENT	STATS\$SESSION_EVENT	STATS\$SESSION_EVENT	>= 0
V\$SESSTAT	STATS\$SESSTAT	STATS\$SESSTAT	>= 0
V\$SGA	STATS\$SGA	STATS\$SGA	>= 0
V\$SGASTAT	STATS\$SGASTAT	STATS\$SGASTAT	>= 0
V\$SHARED_POOL_ADVICE	--	STATS\$SHARED_POOL_ADVICE	>= 0
V\$SQL_WORKAREA_HISTOGRAM	--	STATS\$SQL_WORKAREA_HISTOGRAM	>= 0
V\$SYSSTAT	STATS\$SYSSTAT	STATS\$SYSSTAT	>= 0
V\$SYSTEM_EVENT	STATS\$BG_EVENT_SUMMARY		
STATS\$SYSTEM_EVENT	STATS\$BG_EVENT_SUMMARY		
STATS\$SYSTEM_EVENT	>= 0		
V\$TEMPSTAT	STATS\$TEMPSTATXS	STATS\$TEMPSTATXS	>= 0
V\$UNDOSTAT	--	STATS\$UNDOSTAT	>= 0
V\$WAITSTAT	STATS\$WAITSTAT	STATS\$WAITSTAT	>= 0
V\$SQLTEXT	STATS\$SQLTEXT	STATS\$SQLTEXT	>= 5
V\$SQL	--	STATS\$SQL_PLAN_USAGE	>= 5
V\$SQLXS	STATS\$SQL_SUMMARY		
STATS\$SQL_STATISTICS	STATS\$SQL_SUMMARY		
STATS\$SQL_STATISTICS	>= 5		
V\$SQL_PLAN	--	STATS\$SQL_PLAN	>= 6
V\$SEG_STAT	--	STATS\$SEG_STAT	>= 7
V\$SEGMENT_STATISTICS	--	STATS\$SEG_STAT_OBJ	>= 7
V\$LATCH	STATS\$LATCH	STATS\$LATCH	10
V\$LATCH_CHILDREN	STATS\$LATCH_CHILDREN	STATS\$LATCH_CHILDREN	10
V\$LATCH_PARENT	STATS\$LATCH_PARENT	STATS\$LATCH_PARENT	10

The default *snapshot level* for all versions of STATSPACK is 5. Level 0 can be considered BSTAT/ESTAT-compatibility mode, since the STATSPACK report would be working with data similar to that available to the BSTAT/ESTAT report. Levels 6 and 7 are available only with Oracle 9i and above. Level 10 is recommended for use only in close conjunction with Oracle Support.

The snapshot level can be changed either through parameters provided to each STATSPACK.SNAP procedure call or by updating the STATSPACK_PARAMETER table.

How to Install and Configure STATSPACK

First and foremost, read the README file that comes with STATSPACK, named either statspack.doc in Oracle 8i v8.1.6 or spdoc.txt in all subsequent versions (i.e. v8.1.7, v9.0.1, and v9.2.0). This ASCII text document is very concise and accurate, providing just about everything necessary in 12-14 pages of text.

Next, be sure to consult the copious STATSPACK notes available on <http://metalink.oracle.com>, especially if you are installing STATSPACK on RDBMS versions 7.3.4, 8.0.x, or 8.1.5:

- Note #149113.1 *Installing and Configuring STATSPACK Package* provides space requirements and version-specific installation advice
- Note #165420.1 *How to Install and Run STATSPACK for Oracle 8.0.x and 8.1.6* provides instructions for installation on lower non-standard versions of Oracle

Running the "spcreate.sql" Script to Install STATSPACK

There is a script named spcreate.sql that simply calls three other scripts, spcusr.sql, spctab.sql, and spcpkg.sql:

- spcusr.sql

This script should be run from SQL*Plus connected as INTERNAL or SYSDBA account. It will create an account named PERFSTAT, prompting the user for a DEFAULT and TEMPORARY tablespace. It is recommended to make TOOLS the DEFAULT tablespace, provided it has at least 100 Mb of free space.

NOTE: This script will also try to reference the scripts dbmspool.sql and dbmsjob.sql. The former script creates or re-creates the DBMS_SHARED_POOL package and the latter script does the same for the DBMS_JOB package.

The spcusr.sql script expects these scripts to either be in the current working directory or SQL_PATH (if enabled). By default, these files reside in the ORACLE_HOME/rdbms/admin directory. If those files are not in the present working directory or in your SQL*Plus environment's SQL_PATH, then the script will fail.

A wee bit of friendly advice: If you are installing STATSPACK into a busy production system, it is highly recommended to comment out the invocation of these scripts, if the packages that they create already exist. If they do not already exist, then there is no harm and no risk in running them. But "library cache pin" deadlocks can result on a busy system if the STATSPACK installation tries to re-create them.

- spctab.sql
- spcpkg.sql

The first script ("spctab.sql") creates tables, indexes, and sequences for the STATSPACK repository while the second script ("spcpkg.sql") creates the PL/SQL package named STATSPACK.

NOTE: Each of these scripts will attempt to connect as the account PERFSTAT using the default password PERFSTAT. The embedded CONNECT commands in these scripts do not contain TNS string qualifications, because Oracle assumes that you will be running these scripts from the database server itself.

A wee bit of friendly advice: If you are running these scripts across SQL*Net, then you may want to edit these scripts to modify the CONNECT commands to include TNS strings.

How to Maximize STATSPACK Effectiveness

The most important thing to realize about STATSPACK is that its one standard report, generated by the SQL*Plus script spreport.sql, is almost totally worthless.

There is quite a statement to make, but it is true. The report suffers from the same malaise as its predecessor, the BSTAT/ESTAT report, suffered from: too much information in no particular order.

Luckily, the private sector provides an admirable solution to this problem...

The www.OraPerf.com Web site and the YAPP Report

Starting some years ago, a strange little web site named <http://www.oraperf.com> appeared. It was managed by Anjo Kolk of Oracle's Server Technology division. In its first incarnation, the web site was simply a single text box with a *Browse* and an *Upload* button. You used the *Browse* button to find the name of a text file, either the output of a BSTAT/ESTAT report or the output of the STATSPACK spreport.sql script. When the name of the file was specified in the lone text box, you clicked *Upload* and the text file was uploaded to the web site. The text file was then processed and an HTML report was produced back to your browser: a YAPP report.

The full explanation of the YAPP (Yet Another Performance Profiler) methodology is beyond the scope of this paper, but the white paper that defines it can be downloaded from <http://www.OraPerf.com/whitepapers.htm>. To make a long story short, the YAPP methodology is *response-time analysis*. There are database statistics in Oracle's V\$ views (and subsequently in the STATSPACK repository) which represent, in summary, the idea of *total response-time*.

Response time is the amount of elapsed time between the time a user initiates an action and the response is returned. While this time period involves traversal through many layers of technology (i.e. browser, PC, internet, HTTPD server, network, application server, network, database server, storage-area network, storage, and back), the Oracle database does a good job accounting for the time spent adjacent to and within the database server. Roughly, the situation is expressed by the simple formula:

$$R = S + W$$

which translates to:

$$\text{Response-time} = \text{Service-time} + \text{Wait-time}$$

In other words, a session in the Oracle database instance is either *performing work* (i.e. service time) or *waiting for something* (i.e. wait time). The fact that Oracle is capable of tracking both *service time* and *wait time* to a very fine detail is incredibly valuable.

However, the person or persons who created both the old BSTAT/ESTAT report as well as the more recent STATSPACK report are unaware of the real meaning of this formula. Luckily, Mr Kolk took the time to correct this situation.

The YAPP report is organized in a top-down fashion, starting with the basic premise that $R = S + W$. Below that basic premise, the report then prioritizes the detailed breakdown both service-time and wait-time by percentage. Whatever statistic consumes the largest percentage of either service-time or wait-time is displayed first. Whatever statistic consumes the smallest percentage is displayed last. Thus, an Oracle performance analyst is able to obtain a very clear picture of what is consuming the most resources in the database instance very quickly.

Don't bother trying to make heads or tails of the standard STATSPACK report. Instead, upload that worthless report to the YAPP analyzer at <http://www.OraPerf.com> and rewrite it so it makes sense.

continued on page 28

Customizing STATSPACK

STATSPACK is quite version-specific to the RDBMS, as you might expect. After all, the internal V\$ views are referenced during the data gathering. Luckily, the STAT\$ tables representing the STATSPACK repository are not quite so sensitive

STATSPACK is a work in progress, even now over 4 years after it was made generally available and over 15 years since its creation. While it is still *open source*, it is a good idea to be careful about making any customizations to it:

- Be cautious when modifying the STATSPACK package or triggers belonging to the PERFSTAT account
- Do not modify or drop columns in the standard STATSPACK tables belonging to the PERFSTAT account
- Do not modify or drop any sequences, indexes, or constraints belonging to the PERFSTAT account

However, this does not mean that some changes cannot be made:

- Despite what the documentation states, STATSPACK tables are perfectly capable of storing data from multiple databases and multiple instances within an OPS/RAC database
- Adding columns to existing standard STATSPACK tables does not have any adverse impact in any STATSPACK version so far (i.e. v8.1.6 through v9.2.0)
- Creating additional program modules to modify any new data structures

Based on these guidelines, here are some tested and proven customizations to STATSPACK that address some of the existing shortcomings of the product.

Difficult-to-use Purge Script

The sppurge.sql script is provided with versions 8.1.7 and up, but is poorly documented. This SQL*Plus script will display all of the existing *snapshot IDs* and the data-times they represent, then prompt a user interactively for a range of snapshot IDs to delete.

This makes purging a manual task, and thus one likely to be forgotten.

What DBAs really want to do is specify how many days of STATSPACK data to retain, implicitly purging any data that is older. Taking that idea, I wrote the SPPURPKG package whose source code is available online at <http://www.EvDBT.com/tools.htm> in the file sppurpkg.sql. The packaged procedure PURGE takes a single parameter representing the number of days to retain. The number, of course, can be specified either as a whole number or a fraction, thus allowing a wide range of control. This packaged procedure can also be called from DBMS_JOB, thus making the data purge fully automatic. It is recommended to run this packaged procedure no more than once per day and at least once per week.

Automated CBO Statistics Gathering

While the SQL statements within the STATSPACK.SNAP procedure are extremely simple and do not need CBO statistics gathered for optimal performance, the SQL statements that you might run while reporting or performing analysis might require optimization and need CBO statistics.

Unless your database already performs this gathering automatically, the following SQL text can be used to automate the gathering of CBO statistics using the DBMS_JOB package for job-scheduling and the DBMS_STATS package for the actual data gathering:

```
SQL> connect perfstat
SQL> variable jobno number
SQL> exec dbms_job.submit(:jobno,
'dbms_stats.gather_schema_stats('PERFSTAT');', -
2                                trunc(sysdate+1), 'trunc(SYSDATE+1)', TRUE);
SQL> print jobno
```

Also, be sure to query the USER_JOBS view periodically to ensure that the job is not “broken”.

Affecting the Database That you are Monitoring

This is a huge architectural change for STATSPACK, but one that is sorely necessary.

There are three major impacts of STATSPACK:

- The impact of data gathering (i.e. running the STATSPACK.SNAP procedure hourly) on available CPU, memory, and I/O resources on the database server.
- The impact of the STATSPACK repository tables and indexes on storage space in the database.
- The impact of reporting and analytical queries against the STATSPACK repository on available CPU, memory, and I/O resources on the database server.

There is really no way to mitigate impact #1, since the data must be gathered.

However, it is certainly possible to remove impacts #2 and #3 from the database that is being monitored, by replicating the gathered information from the *front-end database*, which might be a mission-critical production database, to a non-mission-critical *back-end database*, where the real repository of STATSPACK information is kept.

By doing this, there is no need to keep large amounts of STATSPACK data within the mission-critical front-end database. The data can be purged frequently to keep the volume small.

Also, there is no longer any reason to perform reporting or analysis against the mission-critical *front-end* database. All such reporting and analysis can (and should!) be performed against the larger, more complete back-end repository database.

All of the STATSPACK tables have as part of their PRIMARY KEY constraint columns named DBID and INSTANCE_NUMBER. Thus, statistics from different databases can be differentiated using DBID and statistics from different OPS/RAC instances within the same database can be differentiated using INSTANCE_NUMBER. Even though the standard Oracle documentation states that doing this is “unsupported”, there is no reason why this should be so. I have tested this quite thoroughly and the STATSPACK repository is quite happy supporting data from multiple databases and instances.

Last, it is the copy of STATSPACK running on the mission-critical *front-end* database that must remain pristine and un-customized so that there are no supportability problems with Oracle. The non-mission-critical *back-end* repository database can be customized as boldly as you dare, without jeopardizing any mission-critical systems. This provides a fascinating amount of freedom!

The details of performing this architectural modification is quite beyond the scope of this paper and presentation, but it is almost trivial once an approach has been chosen. The important thing is to realize that this modification is necessary for any large-scale usage of STATSPACK.

Storing Delta Values as Well as Raw Values

The tables in the PERFSTAT schema store data values from the source V\$ views exactly as they appeared when the snapshot was taken. Thus, for any given statistic, you might see values such as:

```
SQL> select snap_id, name, value from stats$sysstat
2 where name='physical reads' order by snap_id;
```

SNAP_ID	NAME	VALUE
5907	physical reads	792468540
5908	physical reads	794265420
5909	physical reads	794283250
5910	physical reads	794296244
5911	physical reads	2862
5912	physical reads	44126
5913	physical reads	312769
5914	physical reads	523589
5915	physical reads	747936
5916	physical reads	1274245

Note that the value associated with each successive snapshot is a cumulative value, not a *delta* or *incremental change* value, from the previous one. Also, please note that between snapshots 5910 and 5911, the database instance was restarted, so the cumulative numbers started all over again. All of this makes it difficult to write simple SQL statements against these tables.

It would be more useful if the STATSPACK tables also stored the *deltas* or *incremental changes*, as well as the cumulative raw values:

```
SQL> select snap_id, name, value, value_inc from stats$sysstat
2 where name='physical reads' order by snap_id;
```

SNAP_ID	NAME	VALUE	VALUE_INC
5907	physical reads	792468540	48304
5908	physical reads	794265420	1796880
5909	physical reads	794283250	17830
5910	physical reads	794296244	12994
5911	physical reads	2862	2862
5912	physical reads	44126	41264
5913	physical reads	312769	268643
5914	physical reads	523589	210820
5915	physical reads	747936	224347
5916	physical reads	1274245	526309

Now, in the newly-added column VALUE_INC, the incremental changes to the VALUE column are stored, ready for easy retrieval. So, now a query like:

```
SQL> select min(value_inc), max(value_inc), avg(value_inc), sum(value_inc)
2 from stats$sysstat where name = 'physical reads' and snap_id between 5907
and 5916;
```

MIN(VALUE_INC)	MAX(VALUE_INC)	AVG(VALUE_INC)	SUM(VALUE_INC)
2,862.00	1,796,880.00	315,025.30	3,150,253.00

becomes meaningful.

Please note how the raw values of VALUE increased steadily from snapshots 5907 through 5910. Abruptly, at snapshot 5911, we see that the value has decreased dramatically to 2,862. What happened?

Obviously, the database instance was stopped (shutdown) and then started. This caused all of the values in the memory-based V\$ views to be reset back to zero. So, when snapshot 5911 occurred, only 2,862 physical reads had

been recorded since the database instance was restarted. There is no record of how many recorded physical reads were lost when the database instance was shutdown, so obviously database instance restarts affect the accuracy of the data recorded in the STATSPACK repository.

Note: If you are using Oracle 8i or above (i.e. Oracle 8i, Oracle 9i, or Oracle 10g), you can create a BEFORE SHUTDOWN database event trigger to perform a final call to the STATSPACK.SNAP packaged procedure. This database event trigger is called during SHUTDOWN NORMAL and SHUTDOWN IMMEDIATE operations, but not when the database instance is SHUTDOWN ABORT or crashes. So, you can minimize the amount of V\$ performance information that is lost during an instance restart, but you cannot eliminate it.

Code for an example of this BEFORE SHUTDOWN database event trigger is available on my personal web site at <http://www.EvDBT.com/tools.htm>, in the script named sp_shutdown_trg.sql.

Calculating the *deltas* or *incremental changes* between snapshots is greatly facilitated by the use of the new LAG function, which became available in Oracle in v8.1.6. The LAG function allows you to view data in a previous row, which is ideal for calculating deltas.

So, a layer of database views can be created atop the tables in the STATSPACK repository that make use of the LAG function. For example, in order to add a column named VALUE_INC to display the *incremental change* or *delta* in value from one snapshot to the next, a database view named STATSPACK_V could be created using the following syntax:

```
create or replace view stats$sysstat_v
```

```
as
```

```
select snap_id,
       dbid,
       instance_number,
       statistic#,
       name,
       value,
       nvl(decode(greatest(value,
                           nvl(lag(value)over(order by
                                           dbid,instance_number,statistic#,snap_id),0)),
                           value, value-lag(value)over(order by
                                           dbid,instance_number,statistic#,snap_id),
                           value), 0) value_inc
       from stats$sysstat;
```

The expression that defines the derived VALUE_INC column in the view is a little complex, so let's break it down and take a closer look at it.

First, we have an all-encompassing NVL() function, to avoid the possibility of a NULL value being returned. If we peel away that NVL() function, we next have a DECODE() conditional function. The first parameter to the DECODE() function is a GREATEST() function which is comparing the current row's value in the VALUE column to the previous row, using a LAG() function. If the GREATEST() function shows that the current value is greater than the previous value, then we return the difference (i.e. "value – LAG(value) OVER (...)"). Otherwise, if the GREATEST() function shows that the current value is less than the previous row's value, then we have to assume that database instance was restarted and we simply use the current row's data value.

continued on page 30

Defining a view like this for all of the tables that contain such data allows for some very creative types of analysis, such as:

- How many *logical reads* per hour are being generated by the database instance? By a specific SQL statement?
- What is the average volume of *redo* being generated by the database by the hour? By the day? What are the peaks and valleys?
- Which tables or indexes are experiencing the most contention from *buffer busy waits* on a regular basis?
- Which tables or indexes generate the most *physical reads*?
- Which SQL statement generates the most *logical reads* over the past 3 months?

In addition to aggregating data, it becomes very easy to visualize history and trends. With regularly-scheduled snapshots occurring every hour (i.e. the default frequency for STATSPACK), the patterns of resource usage over time become an important clue as to the identify of a specific SQL statement.

The SQL*Plus script “sp_delta_views.sql” is available online on my personal web site at <http://www.EvDBT.com/tools.htm> will create these *delta views* for all of the tables in the STATSPACK repository where it makes sense.

Let’s say that, by using the standard STATSPACK report or the more-useful summarization YAPP summarization from the <http://www.OraPerf.com> web site, you know that a particular SQL statement is very expensive, consuming hundreds of millions of *logical reads* and millions of *physical reads* over a 1-hour or 24-hour time period. But this SQL statement could belong to any of a hundred different program modules – how do you know which form, report, or batch program is being affected?

One way is to look at the data over time, without aggregation. Consider the following report:

Text of SQL statement

```

SELECT DS.DELIVERED_STATUS FROM DPA_SHIPMENTS DS WHERE DS.TRU
CK_ID = :b1 AND (DS.DELIVERY_ID IN (SELECT DL.DELIVERY_ID FRO
M WSH_DELIVERIES DL WHERE DL.STATUS_CODE IN ('CL','CA','CB')
)) FOR UPDATE OF DS.DELIVERED_STATUS
    
```

Snapshot Time	Nbr Of Execs	Avg Disk Reads Per Exec	Avg Buffer Gets Per Exec
07-JUL 06:00	0	0.00	0.00
07-JUL 07:00	0	0.00	0.00
07-JUL 08:00	0	0.00	0.00
07-JUL 09:00	57	1,988.43	748,398.57
07-JUL 10:00	912	720.25	748,447.08
07-JUL 11:00	621	503.90	748,551.95
07-JUL 12:00	512	193.33	748,653.17
07-JUL 13:00	99	1,079.44	748,710.11
07-JUL 14:00	215	618.73	748,788.53
07-JUL 15:00	616	220.50	748,865.94
07-JUL 16:00	521	310.86	748,962.76
07-JUL 17:00	4	0.00	749,024.75
07-JUL 18:00	0	0.00	0.00
07-JUL 19:00	0	0.00	0.00
07-JUL 20:00	0	0.00	0.00
07-JUL 21:00	0	0.00	0.00
07-JUL 22:00	0	0.00	0.00
07-JUL 23:00	0	0.00	0.00
08-JUL 00:00	0	0.00	0.00
08-JUL 01:00	0	0.00	0.00
08-JUL 02:00	0	0.00	0.00
08-JUL 03:00	0	0.00	0.00

08-JUL 04:00	0	0.00	0.00
08-JUL 05:00	0	0.00	0.00
08-JUL 06:00	0	0.00	0.00
08-JUL 07:00	0	0.00	0.00
08-JUL 08:00	72	3,479.00	749,051.00
08-JUL 09:00	96	1,161.67	499,420.00
08-JUL 10:00	636	613.47	741,775.44
08-JUL 11:00	98	1,006.75	749,335.63
08-JUL 12:00	415	3,250.00	749,376.00
08-JUL 13:00	327	167.00	749,407.43
08-JUL 14:00	110	1,616.90	749,476.60
08-JUL 15:00	918	193.78	749,547.61
08-JUL 16:00	825	272.76	749,656.00
08-JUL 17:00	438	145.13	749,736.00
08-JUL 18:00	71	0.00	749,760.00
08-JUL 19:00	0	0.00	0.00
08-JUL 20:00	0	0.00	0.00

This report was created using a SQL*Plus script named sphistory.sql, available online on my personal web site at <http://www.EvDBT.com/tools.htm>.

Clearly, this SQL statement belongs to a program that is executed only between 8:00am and 6:00pm. We still do not know whether the SQL statement belongs to a form, to a report, or to a batch program, but we know something more about how it is being used.

Since each execution of this SQL statement is consuming about 750,000 *logical reads* each time it is executed, finding some way to make it more efficient should have an enormous positive impact not only on the program to which it belongs, but also to the entire system overall.

Detecting a problem like this is useful, but equally useful is directly demonstrating the positive impact of something you did. In the case of the SQL statement shown above, purging the WSH_DELIVERIES table (used in the sub-query) had an enormous impact on the execution time and system resources (such as *physical reads* and *logical reads*) consumed by this query. On the very hour following the completion of the purge program, STATSPACK recorded the average number of *physical reads* per execution dropping to zero and the average number of logical reads per execution dropping to less than 200, consistently. Being able to clearly demonstrate results is as valuable as detecting the problem in the first place.

Conclusion

STATSPACK is a great utility with a lot of potential. The standard STATSPACK report, by itself, has limited use, but when augmented with either the STATSPACK Viewer (<http://www.geocities.com/alexandr/index.html>) or the YAPP processor (<http://www.oraperf.com>), it becomes very powerful and easy-to-use. In fact, the repository of statistical data built by STATSPACK can easily be utilized as the beginnings of a data warehouse of information about database performance. This data warehouse can be analyzed much like any other type of data warehouse, turning data into intelligence.



About the Author

Tim Gorman is a principal consultant for SageLogix, Inc, based in Colorado, USA. He has been a “C” programmer on databases on UNIX and VMS since 1983 and has worked with Oracle technology since 1990. Gorman is co-author of *Oracle8 Data Warehousing* and *Essential Oracle 8i Data Warehousing*, published in 1998 and 2000, respectively, by John Wiley & Sons. Gorman can be reached at Tim@SageLogix.Com.