CHAPTER 2

Overview of STATSPACK
The focus of this chapter will be on understanding the internal architecture of the STATSPACK utility, the basic information included in the STATSPACK tables, and appreciating how the information in the STATSPACK tables can be used to tune and monitor your Oracle database.

As you may know, Oracle offers several software tools for performance and tuning. Oracle has the Oracle Enterprise Manager (OEM) performance pack, which directly interrogates the memory of the v$ structures to show what is going on in your database at the time the performance problem is being experienced. OEM also offers data collection mechanisms via the intelligent agents to capture information from remote databases and has a central repository where performance data can be analyzed at a later time.

However, the OEM tools have several shortcomings. First, until Oracle9i, it rarely benefits the Oracle administrator to see what’s happening at the specific moment in time you encounter a performance problem, since there’s little that can be done immediately to correct the problem. However, with Oracle9i, the DBA can dynamically change the memory within the SGA (System Global Area) and all Oracle parameters can be changed with alter system commands. A later edition of this text will show how the DBA can automatically monitor Oracle and the database server, and adjust the size of the SGA depending on the current processing within Oracle9i. Oracle9i will create a foundation for a self-tuning database instance, and this is a very exciting new development in Oracle. In short, reactive tuning monitors are often less valuable than proactive monitors such as STATSPACK.

The second problem with OEM is the inherent complexity of using OEM with intelligent agents to capture long-term trend statistics. Many shops with hundreds of databases spend thousands of hours installing and configuring OEM and the intelligent agents.

In contrast with OEM, Oracle STATSPACK is very straightforward and easy to manage. The STATSPACK tables can be easily defined, and the data collection mechanisms can be easily started. Whereas OEM can often take weeks to define the intelligent agents for a large enterprise, the Oracle STATSPACK utility can be very quickly installed, and useful information can be provided almost immediately.

While the benefits of STATSPACK will become very clear in coming chapters, suffice it to say for now that the Oracle STATSPACK utility can easily become the foundation for a comprehensive Oracle tuning environment. STATSPACK has open code, it is easy to understand, and it allows for a wealth of sophisticated reports.

Of course, STATSPACK is like any other vendor product because you cannot alter the existing code and tables. However, you can easily add new STATSPACK tables and write customized code. We will be looking at how we can extend STATSPACK in several chapters.
The STATSPACK Architecture

To fully understand the STATSPACK architecture, we have to look at the basic nature of the STATSPACK utility. As we discussed in the Preface, the STATSPACK utility is an outgrowth of the Oracle UTLBSTAT and UTLESTAT utilities, which have been used with Oracle since the very earliest versions. As the experienced DBA may recall, the BSTAT-ESTAT utilities capture information directly from the Oracle’s in-memory structures and then compare the information from two snapshots in order to produce an elapsed-time report showing the activity of the database.

Whenever an automated collection occurs, STATSPACK simply grabs the relevant information from the in-memory structures and stores the information inside the STATSPACK tables. This information now becomes available for any Oracle SQL queries, and it is quite easy to make time series reports from the information in the STATSPACK tables.

To understand the structure for STATSPACK, let’s begin by taking a high-level overview of all of the different tables that comprise the STATSPACK collection mechanisms.

How STATSPACK Collects Data

The STATSPACK utility works by taking snapshots of the current state of the database. Most users of STATSPACK schedule a job to collect data on an hourly basis and then request additional snapshots when required. While we will go into details on the various methods for obtaining STATSPACK snapshots in Chapter 3, the listing here shows two common methods for getting an immediate snapshot:

```
sql> execute dbms_job.run(x) -- where x is the job number of the statspack job
sql> execute statspack.snap;
```

When a snapshot is executed, the STATSPACK software will sample from the RAM in-memory structures inside the SGA and transfer the values into the corresponding STATSPACK tables, as shown in Figure 2-1. These values are then available for comparing with other snapshots.

Note that in most cases, there is a direct correspondence between the v$ view in the SGA and the corresponding STATSPACK table. In the next example, we see that the stats$sysstat table is identical to the v$sysstat view:

```
SQL> desc v$sysstat;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATISTIC#</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>NAME</td>
<td></td>
<td>VARCHAR2 (64)</td>
</tr>
<tr>
<td>CLASS</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>VALUE</td>
<td></td>
<td>NUMBER</td>
</tr>
</tbody>
</table>
Oracle9i High Performance Tuning with STATSPACK

The data collection mechanism for STATSPACK corresponds closely with the behavior of the `utlbstat.sql` and `utlestat.sql` (commonly called BSTAT-ESTAT) utilities that have been used for many years with Oracle. As we may recall from many years of using BSTAT-ESTAT, the utility samples data directly from the v$ views. If we look inside `utlbstat.sql`, we see the SQL that samples directly from the view:

```
insert into stats$begin_stats select * from v$sysstat;
```

```
insert into stats$endStats select * from v$sysstat;
```

It is critical to your understanding of the STATSPACK utility that you realize the information captured by a STATSPACK snapshot is accumulated values. The information from the v$ views collects database information at startup time and continues to add to the values until the instance is shut down (see Figure 2-2).

In order to get a meaningful elapsed-time report, you must run a STATSPACK report that compares two snapshots, as shown in Figure 2-3. In later chapters we will examine methods for creating reports that stack elapsed-time reports, showing the changes in values over long periods of time.
CAUTION

It is critical that the user of the STATSPACK reports understands that a report will be invalid if the database is shut down between snapshots. This is because all of the accumulated values will be reset, causing the second snapshot to have smaller values than the original snapshot. When this happens, the STATSPACK report will display negative values.

Now that we understand the basic functionality of STATSPACK, let’s examine the tables the STATSPACK information. Throughout the text you will be introduced to each of these tables, and you will eventually become very familiar with the data contained within each table.

FIGURE 2-2. Accumulated snapshot values

FIGURE 2-3. A STATSPACK snapshot comparison report
The STATSPACK Table Structures

The STATSPACK tables can be broken down into several areas. These areas measure all areas of the Oracle instance, including file I/O, system-wide statistics, data buffer statistics, SQL statistics, and a host of other information. We will become acquainted with these tables in later chapters, but we should introduce the major STATSPACK tables at this time.

The STATSPACK utility is designed to measure Oracle information both from the perspective of a single database as well as a distributed enterprise consisting of many databases. In Figure 2-4, we see the basic hierarchical structure of the tables for the Oracle9i version of STATSPACK.

Let’s begin our discussion by looking at the control tables for STATSPACK. We will then look at the parameter tables, and then describe the information collected in each of the subordinate data tables.

FIGURE 2-4. A high-level overview of the STATSPACK tables
STATSPACK Control Tables

As we can see from this high-level entity relation model, the main anchor for the STATSPACK tables is a table called stats$database_instance. This table contains the database ID, the instance number, and the database server host name for the database that you are measuring. By associating the host name with this table, the DBA can populate database information from several database servers into a single collection mechanism. While the STATSPACK developers have not yet implemented a mechanism for collecting STATSPACK data from many databases in a distributed environment, they have laid the foundation for this functionality in this table.

Here is the structure of this table in Oracle8 and Oracle8i:

SQL> desc STATS$DATABASE_INSTANCE;
Name                       Null?    Type
--------------------------- -------- ----------------------
DBID                       NOT NULL NUMBER
INSTANCE_NUMBER             NOT NULL NUMBER
DB_NAME                     NOT NULL VARCHAR2(9)
INSTANCE_NAME               NOT NULL VARCHAR2(16)
HOST_NAME                   VARCHAR2(64)

In Oracle9i, the table changes structure to include the database startup time, the snap ID, and the parallel default:

SQL> desc STATS$DATABASE_INSTANCE;
Name                       Null?    Type
--------------------------- -------- ------------
DBID                       NOT NULL NUMBER
INSTANCE_NUMBER             NOT NULL NUMBER
STARTUP_TIME                NOT NULL DATE
SNAP_ID                     NOT NULL NUMBER(6)
PARALLEL                    NOT NULL VARCHAR2(3)
VERSION                    NOT NULL VARCHAR2(17)
DB_NAME                     NOT NULL VARCHAR2(9)
INSTANCE_NAME               NOT NULL VARCHAR2(16)
HOST_NAME                   VARCHAR2(64)

For each database instance, we have many occurrences of the stats$snapshot table. This table contains the snapshot ID, the database ID, the instance number, and also the time the snapshot was taken. The stats$snapshot table is going to be very important in all of the scripts in our book because it contains the time that the snapshot was taken. Hence, all of the scripts that will be presented in this book must join into the stats$snapshot table so that you can correlate the time of the snapshot with the individual snapshot details.
Oracle9i High Performance Tuning with STATSPACK

Here is the table description in Oracle8 and Oracle8i:

```sql
SQL> desc STATS$SNAPSHOT;
Name                                      Null?    Type
----------------------------------------- -------- ----------------------
SNAP_ID                                   NOT NULL NUMBER(6)
DBID                                      NOT NULL NUMBER
INSTANCE_NUMBER                           NOT NULL NUMBER
SNAP_TIME                                 NOT NULL DATE
STARTUP_TIME                              NOT NULL DATE
SESSION_ID                                NOT NULL NUMBER
SERIAL#                                            NUMBER
SNAP_LEVEL                                         NUMBER
UCOMMENT                                           VARCHAR2(160)
EXECUTIONS_TH                                      NUMBER
PARSE_CALLS_TH                                     NUMBER
DISK_READS_TH                                      NUMBER
BUFFER_GETS_TH                                     NUMBER
```

In Oracle9i, we see the addition of the shareable memory threshold, the version count threshold, and the all_init value:

```sql
SQL> desc STATS$SNAPSHOT;
Name                                      Null?    Type
----------------------------------------- -------- ----------------------
SNAP_ID                                   NOT NULL NUMBER(6)
DBID                                      NOT NULL NUMBER
INSTANCE_NUMBER                           NOT NULL NUMBER
SNAP_TIME                                 NOT NULL DATE
STARTUP_TIME                              NOT NULL DATE
SESSION_ID                                NOT NULL NUMBER
SERIAL#                                            NUMBER
SNAP_LEVEL                                         NUMBER
UCOMMENT                                           VARCHAR2(160)
EXECUTIONS_TH                                      NUMBER
PARSE_CALLS_TH                                     NUMBER
DISK_READS_TH                                      NUMBER
BUFFER_GETS_TH                                     NUMBER
SHARABLE_MEM_TH                                    NUMBER
VERSION_COUNT_TH                                   NUMBER
ALL_INIT                                           VARCHAR2(5)
```

In the stats$snapshot table, we see that there are three levels for STATSPACK data collection. In the next chapter we will see how these levels control what data is placed into the STATSPACK tables. We will also examine the four threshold columns (executions_th, parse_calls_th, disk_reads_th, buffer_gets_th) and see how these
thresholds can be used to limit the number of stats$sql_summary rows that are added when a STATSPACK snapshot is executed.

Note that the STATSPACK definition table now include sharable_mem_th, all_init and version_count_th values, but these are added for future planning and have no function in Oracle9i STATSPACK.

**The all_init Variable**
To capture the non-default initialization parameters in STATSPACK, you need to set the all_init value to TRUE when taking a snapshot (this is the default). This captures every initialization parameter, although the standard reports just list the non-default parameters. The problem is caused by STATSPACK using the v$system_parameter view instead of the v$parameter view. The STATSPACK developers are treating this as a bug in the v$system_parameter and therefore the problem was not fixed until Oracle9i.

**STATSPACK Parameter Tables**
The STATSPACK utility has numerous parameter tables that are used to record the thresholds and level of each snapshot in the STATSPACK collection process. There are two tables for parameters, the stats$snapshot_parameter table and the stat$level_description table.

**stats$statspack_parameter**
The stats$statspack_parameter table contains the default snapshot level for the database instance:

```
SQL> desc STATS$STATSPACK_PARAMETER;
Name                                      Null?    Type
----------------------------------------- -------- ------------------------
DBID                                      NOT NULL NUMBER
INSTANCE_NUMBER                           NOT NULL NUMBER
SESSION_ID                                NOT NULL NUMBER
SNAP_LEVEL                                NOT NULL NUMBER
NUM_SQL                                   NOT NULL NUMBER
EXECUTIONS_TH                             NOT NULL NUMBER
PARSE_CALLS_TH                            NOT NULL NUMBER
DISK_READS_TH                             NOT NULL NUMBER
BUFFER_GETS_TH                            NOT NULL NUMBER
PIN_STATSPACK                             NOT NULL VARCHAR2(10)
LAST_MODIFIED                             DATE
UCOMMENT                                   VARCHAR2(160)
JOB                                        NUMBER
```
Oracle9i High Performance Tuning with STATSPACK

Here is the Oracle9i version of this table with the addition of pin_statspack and all_init values:

```
SQL> desc STATS$STATSPACK_PARAMETER;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>INSTANCE_NUMBER</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>SESSION_ID</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>SNAP_LEVEL</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>NUM_SQL</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>EXECUTIONS_TH</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>PARSER_CALLS_TH</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>DISK_READS_TH</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>BUFFER_GETS_TH</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>SHARABLE_MEM_TH</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>VERSION_COUNT_TH</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>PIN_STATSPACK</td>
<td>NOT NULL</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>ALL_INIT</td>
<td>NOT NULL</td>
<td>VARCHAR2(5)</td>
</tr>
<tr>
<td>LAST_MODIFIED</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>UCOMMENT</td>
<td>VARCHAR2(160)</td>
<td></td>
</tr>
<tr>
<td>JOB</td>
<td>NUMBER</td>
<td></td>
</tr>
</tbody>
</table>

The stats$level_description Table

The stats$level_description table is used to describe the data collection for each level of STATSPACK collection. There are only three levels of STATSPACK collection—0, 5, and 10—and the rules are quite simple. A level 0 collection populates all tables except stats$sql_summary and stats$latch_children. A level 5 collection adds collection for stats$sql_summary, and a level 10 collection adds data for the stats$latch_children table. A level 5 collection is the default:

```
SQL> desc STATS$LEVEL_DESCRIPTION;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNAP_LEVEL</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR2(300)</td>
<td></td>
</tr>
</tbody>
</table>

Now that we understand the basic structure of STATSPACK, let's review some of the uses for STATSPACK. These uses will be a central theme throughout the body of this text, and we will be showing dozens of examples of each approach.
Uses for STATSPACK Information

Now that we have a high-level understanding of the STATSPACK tables and the information captured in STATSPACK, we can begin to take a look at how this information can help us in our Oracle tuning endeavors. There are many uses for STATSPACK, in addition to standard database tuning. The information in the STATSPACK tables can be used for resource planning and predictive modeling, as well as used for reports that can tell the Oracle professional those times in which the Oracle databases experienced stress.

While we will be going into great detail on the uses for the STATSPACK tables in later chapters, suffice it to say for now that we will have scripts available for virtually every type of event that affects Oracle performance.

For the first time in the history of Oracle, we have a tool provided by Oracle that is capable of capturing complete database statistics over long periods of time. Because of this ability to capture Oracle statistics over long periods of time, STATSPACK offers the database administrator a huge opportunity to be able to go back and analyze the behavior of their database during specific processing periods for the application. Due to the time-oriented nature of the STATSPACK data, the Oracle administrator can do far more than simply tune the database. The DBA now has the capability of doing long-term trend analysis, post hoc analysis of performance problems, resource planning, and predictive modeling that will help everyone in the IT organization understand the growth demands of the Oracle database.

Let’s begin by taking a brief look at how the STATSPACK tables will enable us to do Oracle tuning far more efficiently than ever before.

Database Tuning with STATSPACK

Ever since the first releases of Oracle, the Oracle DBA has been charged with making sure that the Oracle database performs at optimal levels. In order to do this, the DBA has been forced to interrogate Oracle’s internal structures in real time so that they might be able to see what’s going on inside the Oracle database when the problem occurs. The DBA would then adjust Oracle parameters to maximize the throughput of information through the Oracle database.

This mode of tuning is generally referred to as reactive tuning. In reactive tuning mode, the Oracle database administrator captures information about a current performance problem, and then queries the Oracle database in order to ascertain its cause. In reactive tuning, the Oracle database administrator does not have any immediate options for fixing the database, and will make changes later on in order to remedy the problem that occurred in the previous point in time.

With the advent of STATSPACK, we see that the Oracle administrators now have a data repository at their disposal allowing them to leisurely analyze Oracle performance statistics and trends over time. This allows the DBAs to come up with
a general tuning strategy that addresses all of the different kinds of processing that can take place within the Oracle application.

This approach is commonly known as proactive tuning. In proactive tuning mode, the Oracle database administrator’s goal is to tune the database by coming up with global parameters and settings that will maximize Oracle throughput at any given point in time. By using a proactive approach to Oracle tuning, the Oracle administrator can ensure that the database is always optimally tuned for the type of processing that is being done against the database.

The self-tuning features of Oracle9i also allow STATSPACK information to be useful for dynamic SGA reconfiguration. For example, if STATSPACK notes that the demands on the shared pool become very high between 1:00 P.M. through 2:00 P.M., the DBA could trigger a dynamic increase of the shared_pool_size parameter during this period.

As we discussed, the STATSPACK tables do nothing more than interrogate the in-memory v$ structures and place the information in the Oracle STATSPACK tables. While this may be a bit of an oversimplification, having STATSPACK information captured over periods of time gives the DBA the opportunity to use this data to model an optimal performance plan for the database. Over the course of the rest of this book we will be specifically addressing how the STATSPACK tables can be used in order to allow the Oracle database to perform this type of proactive tuning, and we will come up with an overall plan that is best suited for the database.

**Resource Planning**

As we just mentioned, the STATSPACK utility captures performance and tuning statistics over long periods of time. Hence, the STATSPACK utility is very useful for doing resource planning for an entire IT organization. Within an IT organization, managers are often charged with predicting the amount of resources that are going to be needed by Oracle and ensuring that those resources are delivered in time so that the Oracle database does not experience any kind of resource related outages. A typical sample resource plan is shown in Figure 2-5.

By using the STATSPACK tables, resource managers can plot the growth patterns of objects (tables and indexes) within the Oracle database, and come up with linear regression models that will accurately predict the disk storage needs of the Oracle database at future points in time. This resource modeling capability can also be used for predicting hardware resources, such as pending needs within the central processing units and the RAM memory structures. In Chapters 5 through 8 we will be taking a close look at how we can extend the STATSPACK tables in order to capture these types of server statistics, and how we can plot the statistics in order to do predictive resource modeling for all areas within the Oracle enterprise.
Predictive modeling is one of the most important new areas of Oracle tuning, and one that lends itself very well to the use of the STATSPACK tables. In a predictive model, the Oracle DBA is charged with taking existing statistics and predicting future needs for all areas within the Oracle database. For example, the Oracle DBA could analyze the STATSPACK data buffer hit ratio and compare it to the memory usage within the Oracle db_cache_size. The DBA can then extrapolate the information from the studies, and predict the times at which the Oracle data buffers would need to be increased in order to maintain the current levels of performance.

Likewise, the Oracle DBA can also make a detailed analysis of Oracle’s data buffer caches (the KEEP pool, DEFAULT pool, the RECYCLE pool, and the pools for multiple block sizes) and accurately measure the performance of each one of these pools over long periods of time. Based upon existing usage, the Oracle DBA can accurately predict at what time additional RAM memory is needed for each of these data buffers in order to keep the current performance levels for the system.

When discussing predictive modeling, the STATSPACK tables also offer the Oracle DBA an opportunity to slice off the information according to previously unavailable parameters. In the real world, Oracle applications commonly follow cyclical patterns. For example, an Oracle financials application may be very active on the last Friday of every month when all of the books are being closed and financial reports are being prepared. Using the STATSPACK data, the Oracle DBA can extract information from the last Friday of every month for the past year, and take a look at the specific performance needs of the end-of-month Oracle financials applications.

In Oracle8i and on into Oracle9i, we see new features that allow the database administrator to dynamically change the database depending upon the performance
needs of the applications. In Oracle9i, the Oracle DBA has the ability to dynamically alter the memory configuration of the Oracle instance. By making all 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, and STATSPACK can identify these changing needs.

Once the Oracle administrators recognize cyclic performance patterns in the Oracle database, they are now in a position to reconfigure the database in order to meet the specific processing needs of the system.

Prior to Oracle9i, it was not uncommon to find several versions of Oracle’s init.ora files, each one customized to the processing needs of the application that is running at the time. For example, it was not uncommon to see a special version of the Oracle instance (with a different initialization file) that was dedicated to batch processing tasks that might occur, say, on every Friday, while another version of the init.ora file is customized for OLTP transactions. Some Oracle shops also created additional init.ora files that were suited to data warehouse processing that might occur on the weekend. In each of these cases, the Oracle database has to be stopped and restarted with the appropriate init.ora configuration file.

Starting with Oracle9i, STATSPACK can be used to identify any specific recurring time when a component of Oracle is stressed, and the DBA can then trigger a script to dynamically change Oracle during these times. In summary, STATSPACK data is ideally suited to work with the dynamic SGA features of Oracle9i, and later chapters will show scripts for using STATSPACK to identify out-of-bounds conditions.

**Conclusion**

Now that we understand the basic structure and uses for STATSPACK, we are ready to install and configure the STATSPACK software. The next chapter will review all of the installation steps, and Chapter 4 will show you all of the performance information that is contained within a STATSPACK snapshot.