CHAPTER 9

Using STATSPACK
As of Oracle 8.1.6, you can use the STATSPACK utility to monitor the performance of your database. STATSPACK replaces the UTLBSTAT/UTLESTAT scripts available with earlier versions of Oracle and offers several significant enhancements to those scripts. In this chapter, you will see how to install STATSPACK, how to manage it, and how to run and interpret the generated reports.

**Installing STATSPACK**

STATSPACK must be installed in every database to be monitored. The installation script, named spcreate.sql, is found in the /rdbms/admin subdirectory under the Oracle software home directory. The spcreate.sql script creates a user named PERFSTAT and creates a number of objects under that schema.

**NOTE**

You should allocate at least 100MB for the initial creation of the PERFSTAT schema’s objects.

To start the spcreate.sql script, change your directory to the ORACLE_HOME/rdbms/admin directory and log into SQL*Plus in an account with SYSDBA privileges:

```
SQL> connect system/manager as SYSDBA
SQL> @spcreate
```

During the installation process, you will be prompted for a default tablespace for the PERFSTAT user (a list of available tablespaces will be displayed along with this prompt). You will also be asked to specify a temporary tablespace for the user. Once you have provided a default and temporary tablespace for the PERFSTAT account, it will be created, and the installation script will log in as PERFSTAT and continue to create the required objects. If there is not sufficient space to create the PERFSTAT objects in the specified default tablespace, the script will return an error.

**NOTE**

Although you start the installation script while logged in as a SYSDBA-privileged user, the conclusion of the installation script will leave you logged in as the PERFSTAT user.

If you want to drop the PERFSTAT user at a later date, you can run the spdusr.sql script located in the ORACLE_HOME/rdbms/admin directory.
Security of the PERFSTAT Account

The PERFSTAT account is created with the default password of PERFSTAT. Change the password after the installation process completes.

The PERFSTAT account is granted the SELECT_CATALOG_ROLE and HS_ADMIN_ROLE roles, along with several system privileges (CREATE/ALTER SESSION, CREATE TABLE, CREATE/DROP PUBLIC SYNONYM, CREATE SEQUENCE, and CREATE PROCEDURE). Any user who can access your PERFSTAT account can select from all of the dictionary views. For example, such a user could query all of the database account usernames from DBA_USERS, all the segment owners from DBA_SEGMENTS, and the currently logged in sessions from V$SESSION. The PERFSTAT account, if left unprotected, provides a security hole that allows intruders to browse through your data dictionary and select targets for further intrusion.

In addition to the privileges it receives during the installation process, the PERFSTAT account will also have any privileges that have been granted to PUBLIC. If you use PUBLIC grants instead of roles for application privileges, you must secure the PERFSTAT account. You can lock database accounts and unlock them as needed; see Chapter 10 for details.

Postinstallation

Once the installation process is complete, the PERFSTAT account will own (in 9.0.1) 36 tables, 37 indexes, a sequence, and a package. You will use the package, named STATSPACK, to manage the statistics collection process and the data in the tables. The collection tables, whose names all begin with “STATS$,” will have column definitions based on the V$ view definitions. For example, the columns in STATS$WAITSTAT are the columns found in V$WAITSTAT with three identification columns added at the top:

```sql
desc stats$waitstat
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNAP_ID</td>
<td>NOT NULL</td>
<td>NUMBER(6)</td>
</tr>
<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>CLASS</td>
<td>NOT NULL</td>
<td>VARCHAR2(18)</td>
</tr>
<tr>
<td>WAIT_COUNT</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>TIME</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>
The Class, Wait_Count, and Time columns are based on the Class, Count, and Time columns from V$WAITSTAT. STATSPACK has added three identification columns:

- **SNAP_ID**: An identification number for the collection. Each collection is called a snapshot and is assigned an integer value.
- **DBID**: A numeric identifier for the database.
- **INSTANCE_NUMBER**: A numeric identifier for the instance, for Real Application Cluster installations.

Each collection you perform is given a new Snap_ID value that is consistent across the collection tables. You will need to know the Snap_ID values when executing the statistics report provided with STATSPACK.

### Gathering Statistics

Each collection of statistics is called a *snapshot*. Snapshots of statistics have no relation to snapshots or materialized views used in replication. Rather, they are a point-in-time collection of the statistics available via the V$ views, and are given a Snap_ID value to identify the snapshot. You can generate reports on the changes in the statistics between any two snapshots.

**NOTE**

As with the UTLBSTAT/UTLESTAT reports, the STATSPACK report will only be valid if the database was not shut down and restarted between the snapshots evaluated.

**NOTE**

Be sure the TIMED_STATISTICS database initialization parameter is set to TRUE prior to gathering statistics.

To generate a snapshot of the statistics, execute the SNAP procedure of the STATSPACK package, as shown in the following listing. You must be logged in as the PERFSTAT user to execute this procedure.

```sql
execute STATSPACK.SNAP;
```

PL/SQL procedure successfully completed.
When the SNAP procedure is executed, Oracle populates your SNAP$ tables with the current statistics. You can then query those tables directly, or you can use the standard STATSPACK report (to see the change in statistics between snapshots).

Snapshots should be taken in one of two ways:

- To evaluate performance during specific tests of the system. For these tests, you can execute the SNAP procedure manually, as shown in the prior example.
- To evaluate performance changes over a long period of time. To establish a baseline of the system performance, you may generate statistics snapshots on a scheduled basis. For these snapshots, you should schedule the SNAP procedure execution via Oracle’s internal DBMS_JOB scheduler or via an operating system scheduler.

For the snapshots related to specific tests, you may wish to increase the collection level, which lets you gather more statistics. As noted in the “Managing the STATSPACK Data” section later in this chapter, each snapshot has a cost in terms of space usage and query performance. For example, since V$SYSSTAT has (in Oracle 9.0.1) 255 rows, every snapshot generates 255 rows in STATS$SYSSTAT. Avoid generating thousands of rows of statistical data with each snapshot unless you plan to use them.

To support differing collection levels, STATSPACK provides a level parameter. By default, the level value is set to 5. Prior to changing the level value, generate several snapshots and evaluate the reports generated. The default level value is adequate for most reports. Alternative level values are listed in the following table:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to 4</td>
<td>General performance statistics on all memory areas, latches, pools, and events.</td>
</tr>
<tr>
<td>5 to 9</td>
<td>Same statistics from the lower levels, plus the most resource-intensive SQL statements.</td>
</tr>
<tr>
<td>10 and greater</td>
<td>Same statistics from the lower levels, plus the most resource-intensive SQL statements and parent/child latch data.</td>
</tr>
</tbody>
</table>

The greater the collection level, the longer the snapshot will take. The default value (5) offers a significant degree of flexibility during the queries for the most resource-intensive SQL statements. The parameters used for the resource-intensive SQL portion of the snapshot are stored in a table named STATS$STATSPACK_PARAMETER. You can query STATS$STATSPACK_PARAMETER to see the settings for the different thresholds during the SQL statement gathering. Its columns include Snap_Level (the snapshot level), Executions_Th (threshold value for the number of
executions), Disk_Reads_Th (threshold value for the number of disk reads), and Buffer_Gets_Th (threshold value for the number of disk reads).

For a level 5 snapshot using the default thresholds, SQL statements are stored if they meet any of the following criteria:

- The SQL statement has been executed at least 100 times.
- The number of disk reads performed by the SQL statement exceeds 1000.
- The number of parse calls performed by the SQL statement exceeds 1000.
- The number of buffer gets performed by the SQL statement exceeds 10,000.
- The sharable memory used by the SQL statement exceeds 1MB.
- The version count for the SQL statement exceeds 20.

When evaluating the snapshot’s data and the performance report, keep in mind that the SQL threshold parameter values are cumulative. A very efficient query, if executed enough times, will exceed 10,000 buffer gets. Compare the number of buffer gets and disk reads to the number of executions to determine the activity each time the query is executed.

To modify the default settings for the thresholds, use the MODIFY_STATSPACK_PARAMETER procedure of the STATSPACK package. Specify the snapshot level via the i_snap_level parameter, along with the parameters to change. Table 9-1 lists the available parameters for the MODIFY_STATSPACK_PARAMETER procedure.

To increase the Buffer_Gets threshold for a level 5 snapshot to 100,000, issue the following command:

```
STATSPACK.MODIFY_STATSPACK_PARAMETER -
   (i_snap_level=>5, i_buffer_gets_th=>100000);
```

If you plan to run the SNAP procedure on a scheduled basis, you should pin the STATSPACK package following database startup. The following listing shows a trigger that will be executed each time the database is started. The KEEP procedure of the DBMS_SHARED_POOL procedure pins the package in the Shared Pool. As an alternative to pinning, you can use the SHARED_POOL_RESERVED_SIZE initialization parameter to reserve Shared Pool area for large packages.

```
create or replace trigger PIN_ON_STARTUP
after startup on database
begin
    DBMS_SHARED_POOL.KEEP ('PERFSTAT.STATSPACK', 'P');
end;
/
Running the Statistics Report

If you have generated more than one snapshot, you can report on the statistics for the period between the two snapshots. The database must not have been shut down between the times the two snapshots were taken. When you execute the report, you will need to know the Snap_ID values for the snapshots. If you run the report interactively, Oracle will provide a list of the available snapshots and the times they were created.

To execute the report, go to the /rdbms/admin directory under the Oracle software home directory. Log into SQL*Plus as the PERFSTAT user and run the spreport.sql file found there.

```
SQL> @spreport
```

Oracle will display the database and instance identification information from V$INSTANCE and V$DATABASE and will then call a second SQL file, spreps.sql.
The sprepins.sql generates the report of the changes in the statistics during the snapshot time interval. The available snapshots will be listed, and you will be prompted to enter a beginning and ending snapshot ID. Unless you specify otherwise, the output will be written to a file named sp_beginning_ending.lst (sp_1_2.lst for a report between Snap_ID values of 1 and 2).

The first portion of the report output provides an overview of the cache areas and their usage. The following listing shows sample output for this section, showing the cache sizes and the load profile.

```
Cache Sizes (end)
~~~~~~~~~~~~~~
Buffer Cache: 160M Std Block Size: 8K
Shared Pool Size: 64M Log Buffer: 512K
```

```
Load Profile
~~~~~~~~~~~~
Per Second Per Transaction
--------------- ----------------
Redo size: 2,270.95 333,830.00
Logical reads: 15.77 2,318.00
Block changes: 3.83 562.50
Physical reads: 0.00 0.50
Physical writes: 0.72 106.50
User calls: 0.10 14.50
Parses: 0.27 39.00
Hard parses: 0.07 10.00
Sorts: 0.30 44.50
Logons: 0.00 0.00
Executes: 0.64 94.50
Transactions: 0.01

% Blocks changed per Read: 24.27 Recursive Call %: 98.14
Rollback per transaction %: 0.00 Rows per Sort: 86.06
```

The load profile helps to identify the type of activity being performed. In this example, the activity recorded was primarily performing queries. The next sections of the report show the instance efficiency percentages (such as the buffer hit ratio and library cache hit ratio) followed by the Shared Pool statistics. The Shared Pool statistics show the percentage of the Shared Pool in use and the percentage of SQL statements that have been executed multiple times (as desired). The following listing shows sample Shared Pool statistics from the report:

```
Shared Pool Statistics
Begin End
-------- --------
Memory Usage %: 28.37 29.17
% SQL with executions>1: 27.77 30.45
% Memory for SQL w/exec>1: 56.64 67.74
```
Based on the data in the preceding listing, at the time of the second snapshot, 29.17% of the Shared Pool’s memory was in use. Of the statements in the Shared Pool, only 30% had been executed more than once, indicating a potential need to improve cursor sharing in the application.

**NOTE**
The section of the report showing the percentage of Shared Pool memory in use is new with the Oracle9i version of STATSPACK.

The next portion of the generated report shows the top five wait events, the full list of wait events, and the background wait events. Identifying major wait events may help to target your tuning efforts.

The most resource-intensive SQL statements in the database are then listed, in descending order of buffer gets. Since the buffer gets statistic is cumulative, the query with the most buffer gets may not be the worst-performing query in the database; it may just have been executed enough times to earn the highest ranking. Compare the cumulative number of buffer gets to the cumulative number of disk reads for the queries; if the numbers are close, then you should evaluate the explain plan for the query.

**NOTE**
If the Shared Pool is flushed between the execution times of the two snapshots, the SQL portion of the output report will not necessarily contain the most resource-intensive SQL executed during the period.

Following the SQL statement listing, you will see the list of changes to statistics from V$SYSSTAT, entitled “Instance Activity Stats.” The V$SYSSTAT statistics are useful for identifying performance issues not shown in the prior sections. For example, you should compare the number of sorts performed on disk to the number performed in memory; increase the sort area size to reduce disk sorts. If there is a significant number of full table scans of large tables, evaluate the most-used queries. The following listing shows four rows from this section of the report:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Total</th>
<th>per Second</th>
<th>per Trans</th>
</tr>
</thead>
<tbody>
<tr>
<td>sorts (disk)</td>
<td>89</td>
<td>0.3</td>
<td>44.5</td>
</tr>
<tr>
<td>sorts (rows)</td>
<td>7,659</td>
<td>26.1</td>
<td>3,829.5</td>
</tr>
<tr>
<td>table scans (long tables)</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>table scans (short tables)</td>
<td>10</td>
<td>0.0</td>
<td>5.0</td>
</tr>
</tbody>
</table>
354 Oracle9i DBA Handbook

The next section of the report provides the I/O statistics by tablespace and by datafile. If the I/O is not properly distributed among your files, you may encounter performance bottlenecks during periods of high activity. You can use this section of the report to identify such bottlenecks and to measure how effectively you have resolved those problems. See Chapter 4 for further details on I/O distribution across files.

Following the I/O statistics, the report lists the buffer cache statistics by pool (default, keep, and recycle), instance recovery statistics (the number of redo blocks), and the PGA memory statistics. After those sections, the report provides rollback segment statistics. First, it lists the activity in the rollback segment (writes, wraps, shrinks, extends) and the waits encountered, as shown in the following example:

<table>
<thead>
<tr>
<th>Trans Table</th>
<th>Pct Wait</th>
<th>Undo Bytes Written</th>
<th>Wraps</th>
<th>Shrinks</th>
<th>Extends</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBS No 0</td>
<td>2.0</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RBS No 1</td>
<td>4.0</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RBS No 2</td>
<td>2.0</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RBS No 3</td>
<td>13.0</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RBS No 4</td>
<td>6.0</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RBS No 5</td>
<td>4.0</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RBS No 6</td>
<td>7.0</td>
<td>0.00</td>
<td>1,824</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RBS No 7</td>
<td>10.0</td>
<td>0.00</td>
<td>202</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RBS No 8</td>
<td>96.0</td>
<td>0.00</td>
<td>207,234</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>RBS No 9</td>
<td>8.0</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RBS No 10</td>
<td>32.0</td>
<td>0.00</td>
<td>10,856</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

If rollback segment waits are occurring, you may need to add more rollback segments to your database. The next section after the rollback segment statistics shows the storage allocations for your rollback segments, providing a guideline for the creation of additional rollback segments. Following the rollback segment sections, the report lists the undo segment statistics for environments using system-managed undo (see Chapter 7).

Latch activity and dictionary cache statistics are then presented, followed by the library cache activity. If your “Pct Miss” value is high, you may need to improve cursor sharing in your application or increase the size of the Shared Pool.

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Get Requests</th>
<th>Pct Miss</th>
<th>Pin Requests</th>
<th>Pct Miss</th>
<th>Reloads</th>
<th>Invali-dations</th>
</tr>
</thead>
<tbody>
<tr>
<td>BODY</td>
<td>4</td>
<td>0.0</td>
<td>5</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CLUSTER</td>
<td>3</td>
<td>0.0</td>
<td>3</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>JAVA DATA</td>
<td>1</td>
<td>0.0</td>
<td>4</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SQL AREA</td>
<td>84</td>
<td>7.1</td>
<td>493</td>
<td>11.4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TABLE/PROCEDURE</td>
<td>298</td>
<td>1.7</td>
<td>408</td>
<td>17.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>3</td>
<td>0.0</td>
<td>3</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Following an SGA memory summary (from V$SGA) and a listing of the memory changes during the snapshot interval, the report lists the database initialization parameters in use at the beginning and end of the report.

Taken as a whole, the report generates a significant amount of data, allowing you to develop a profile of the database and its usage. Based on the initialization, file I/O, and SGA data, you can develop an understanding of the major components in the database configuration. Since it generates so much data, you should be careful not to generate more statistics than you plan to use. The following sections of this chapter address the management of the gathered data.

**Managing the STATSPACK Data**

You should manage the data generated by STATSPACK to guarantee that the space usage and performance of the STATSPACK application meets your requirements as the application data grows. Managing STATSPACK data includes the following steps:

1. Regularly analyze the STATSPACK data. At a minimum, you should analyze the STATSPACK table prior to running the spreport.sql report:
   ```sql
   execute DBMS_UTILITY.ANALYZE_SCHEMA('PERFSTAT','COMPUTE');
   ```

2. Purge old data. Since you cannot generate valid interval reports across database shutdown/startup actions, data prior to the last database startup may not be as useful as the most current data. When the data is no longer needed, purge it from the tables. Oracle provides a script, sppurge.sql, to facilitate purges. The sppurge.sql script, located in the /rdbms/admin directory under the Oracle software home directory, lists the currently stored snapshots and prompts you for two input parameters: the beginning and ending snapshot numbers for the purge. The related records in the STATS$ tables will then be **deleted**. Due to the size of the transactions involved, databases using rollback segments should force the session to use a large rollback segment during the **deletes**:
   ```sql
   SQL> commit;
   SQL> set transaction use rollback segment roll_large;
   SQL> @sppurge
   ```
   The sppurge script prompts you to back up your old statistics before purging them. You can back up the data by exporting the PERFSTAT schema.

3. Truncate the STATSPACK tables when the data is not needed. Old statistical data may no longer be relevant, or you may have imported the old statistics during database migrations or creations. To truncate the old tables, execute the sptrunc.sql SQL*Plus script from within the PERFSTAT account. The script is located in the /rdbms/admin directory under the Oracle software home directory.
Deinstalling STATSPACK

Since STATSPACK includes public synonyms as well as private objects, you should remove the application via a SYSDBA privileged account. Oracle provides a script, spdrop.sql, to automate the deinstallation process. From within the /rdbms/admin directory under the Oracle software home directory, log into SQL*Plus and execute the script as shown in the following listing:

```
SQL> connect system/manager as SYSDBA
SQL> @spdrop
```

The spdrop.sql script calls scripts that will drop the tables, package, public synonyms, and the PERFSTAT user. To reinstall STATSPACK, execute the spcreate.sql script as shown earlier in this chapter.