Secrets of the Oracle Database

ISBN-13 (pbk): 978-1-4302-1952-1

ISBN-13 (electronic): 978-1-4302-1953-8

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jonathan Gennick
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Michelle Lowman, Matthew Moodie, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Beth Christmas
Copy Editor: Lisa Hamilton
Associate Production Director: Kari Brooks-Copony
Production Editor: Kelly Winquist
Compositor: Susan Glinert
Proofreader: Greg Teague
Indexer: Norbert Debes
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at http://www.apress.com/info/bulksales.

The source code for this book is available to readers at http://www.apress.com. You will need to answer questions pertaining to this book in order to successfully download the code.

PART 1

Initialization Parameters

CHAPTER 1

Partially Documented Parameters

Figuratively speaking, the Oracle database management system has a tremendous number of knobs to turn and switches to flip. Oracle9*i* Release 2 has 257 documented parameters, Oracle10*g* Release 2 has 258, and Oracle11*g* Release 1 has 294. Presumably there is no single DBA who has memorized the meanings and permissible values for all those parameters. The *Oracle Database Reference* manual of each respective release is the definitive source for documented initialization parameters. This chapter scrutinizes the partially documented parameters AUDIT_SYSLOG_LEVEL, PGA_AGGREGATE_TARGET, EVENT, and OS_AUTHENT_PREFIX and provides information that is absent from the *Oracle Database Reference* manual. Both AUDIT_SYSLOG_LEVEL and OS_AUTHENT_PREFIX are related to database security. EVENT is a curious parameter in the sense that the parameter itself is documented, but permissible values are not. Among other things it may be used to collect more evidence when errors occur or gather diagnostic information under the supervision of Oracle Support Services. From a performance perspective, learning how PGA_AGGREGATE_TARGET is handled internally allows a DBA to significantly reduce the response time of large sort operations.

AUDIT_SYSLOG_LEVEL

The initialization parameter AUDIT_SYSLOG_LEVEL is partially documented. Several inaccuracies in the documentation suggest that the parameter is less useful than it actually is. Database actions by SYS and/or database administrators or operators may be audited to the UNIX operating system's syslog daemon log files owned by the UNIX user root. This prevents privileged database users from removing audit records that contain a log of their activities. The default setting is to audit CONNECT, STARTUP, and SHUTDOWN with SYSDBA or SYSOPER privileges to files owned by the ORACLE software owner, while not auditing SQL, PL/SQL statements, and other actions with these privileges or other privileges, such as the role DBA, at all. In other words, except for the aforementioned operations, standard auditing (see parameter AUDIT_TRAIL) as well as fine grained auditing (see package DBMS_FGA) are switched off by default. As a consequence, there will be no trace of many activities performed by privileged users. Auditing to operating system files owned by the ORACLE software owner (AUDIT_TRAIL=OS) or to the database table SYS.AUD$ (AUDIT_TRAIL=DB) may be circumvented, since DBAs normally have access to the ORACLE software owner's UNIX account as well as to SYS.AUD$, allowing them to easily remove audit records generated for their actions. Auditing via the UNIX syslog facility is also useful for detecting intrusions by hackers or manipulations by malevolent insiders.

Syslog Facility

A new feature of Oracle10*g* is the ability to write audit trails using the syslog facility on UNIX systems. This facility consists of a daemon process named `syslogd` (see `man syslogd`) that accepts log messages from applications via the `syslog` C library function (see `man syslog`). The configuration file for `syslogd` is usually `/etc/syslog.conf` and log messages go to files in `/var/log` or `/var/adm` depending on the UNIX variant. The log file name is determined by a string that consists of a facility name and a priority or level. Most of these may be used when setting `AUDIT_SYSLOG_LEVEL`. Each entry in `/etc/syslog.conf` assigns a log file name to a certain combination of facility and priority. By placing the entry `user.notice /var/log/oracle_dbms` into the file `syslog.conf` and telling `syslogd` to reread the configuration file by sending it a hang-up signal with the command `kill`,[1] any subsequent log entries from an ORACLE instance with the setting `AUDIT_SYSLOG_LEVEL=user.notice` will be recorded in the file `/var/log/oracle_dbms`.

Introduction to Auditing

On UNIX systems, `CONNECT`, `STARTUP`, and `SHUTDOWN` of an ORACLE instance with `SYSDBA` or `SYSOPER` privileges are unconditionally audited to files with extension `.aud` in `$ORACLE_HOME/rdbms/audit` or a directory specified with the parameter `AUDIT_FILE_DEST`.[2] Oracle9*i* was the first release that had the capability of auditing actions other than `CONNECT`, `STARTUP`, and `SHUTDOWN` performed with `SYSDBA` or `SYSOPER` privileges by setting `AUDIT_SYS_OPERATIONS=TRUE`.

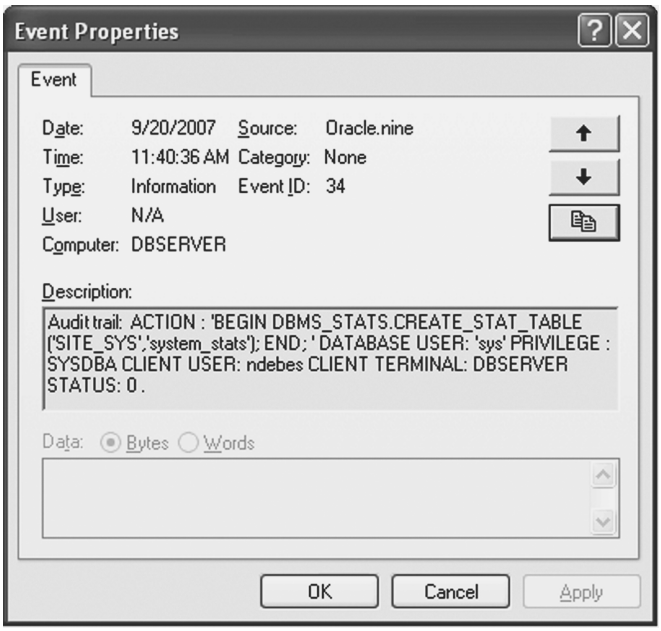

Figure 1-1. *Event Details in Windows Event Viewer*

1. Use `kill -HUP `cat /var/run/syslogd.pid`` on Red Hat Linux.
2. `AUDIT_FILE_DEST` is used as soon as an instance has started. When connecting as SYSDBA or SYSOPER while an instance is down, the default audit file destination `$ORACLE_HOME/rdbms/audit` is used.

When AUDIT_SYSLOG_LEVEL and AUDIT_SYS_OPERATIONS are combined, any SQL and PL/SQL run as user SYS may be audited using the syslog facility. Since the files used by syslog are owned by root, and a DBA usually does not have access to the root account, DBAs will not be able to remove traces of their activity. Of course, this also applies to intruders who have managed to break into a machine and have gained access to the account of the ORACLE software owner but not to the root account. The same applies to hackers who have cracked the password of a privileged database user and are able to connect via Oracle Net.

On Windows, the parameters AUDIT_SYSLOG_LEVEL and AUDIT_FILE_DEST are not implemented, since the Windows event log serves as the operating system audit trail (see Figure 1-1). Just like on UNIX, CONNECT, STARTUP, and SHUTDOWN are unconditionally logged. When AUDIT_SYS_OPERATIONS=TRUE is set, operations with SYSDBA or SYSOPER privileges are also written to the Windows event log, which may be viewed by navigating to **Start ➤ Control Panel ➤ Administrative Tools ➤ Event Viewer**. The logging category used is **Application** and the source is named **Oracle.***ORACLE_SID*. Events for a certain DBMS instance may be filtered by choosing **View ➤ Filter**.

The *Oracle Database Reference 10g Release 2* manual explains AUDIT_SYSLOG_LEVEL as follows (page 1-22):

> *AUDIT_SYSLOG_LEVEL enables OS audit logs to be written to the system via the syslog utility, if the AUDIT_TRAIL parameter is set to os. The value of facility can be any of the following: USER, LOCAL0- LOCAL7, SYSLOG, DAEMON, KERN, MAIL, AUTH, LPR, NEWS, UUCP or CRON. The value of level can be any of the following: NOTICE, INFO, DEBUG, WARNING, ERR, CRIT, ALERT, EMERG.*

Tests of the new feature on a Solaris 10 and a Red Hat Linux system showed that the documentation is inaccurate on three counts:

1. AUDIT_SYSLOG_LEVEL is independent of AUDIT_TRAIL. When AUDIT_SYSLOG_LEVEL is set and AUDIT_TRAIL has the default value NONE, CONNECT, STARTUP, and SHUTDOWN are logged via syslog.
2. Setting the parameters AUDIT_SYSLOG_LEVEL and AUDIT_SYS_OPERATIONS=TRUE causes any actions such as SQL and PL/SQL statements executed with SYSDBA or SYSOPER privileges to be logged via syslog, even if AUDIT_TRAIL=NONE.
3. Only certain combinations of facility and level are acceptable. Unacceptable combinations cause the error "ORA- 32028: Syslog facility or level not recognized" and prevent DBMS instances from starting.

If the documentation were accurate, it would not be possible to audit actions performed with SYSDBA or SYSOPER privileges to the system log, while auditing actions by other users to the data dictionary base table SYS.AUD$. However, such a limitation does not exist.

Using AUDIT_SYSLOG_LEVEL

As stated earlier, the string assigned to AUDIT_SYSLOG_LEVEL must consist of a facility name and a priority or level. Surprisingly, when doing a SHOW PARAMETER or a SELECT from V$PARAMETER, merely the facility is visible—the dot as well as the level are suppressed.[3] For example, with the entry *.audit_syslog_level='USER.NOTICE' in the SPFILE used to start the instance, SHOW PARAMETER yields:

```
SQL> SHOW PARAMETER audit_syslog_level
NAME                                 TYPE        VALUE
------------------------------------ ----------- -----
audit_syslog_level                   string      USER
SQL> SELECT value FROM v$parameter WHERE name='audit_syslog_level';
VALUE
-----
USER
```

Yet, when executing CONNECT / AS SYSDBA, the facility and level logged in /var/adm/messages on Solaris is "user.notice":

```
Feb 21 11:45:52 dbserver Oracle Audit[27742]: [ID 441842 user.notice]

ACTION : 'CONNECT'
Feb 21 11:45:52 dbserver DATABASE USER: '/'
Feb 21 11:45:52 dbserver PRIVILEGE : SYSDBA
Feb 21 11:45:52 dbserver CLIENT USER: oracle
Feb 21 11:45:52 dbserver CLIENT TERMINAL: pts/3
Feb 21 11:45:52 dbserver STATUS: 0
```

If an SPFILE is used, the full setting is available by querying V$SPPARAMETER:

```
SQL> SELECT value FROM v$spparameter WHERE name='audit_syslog_level';
VALUE
-----------
user.notice
```

Auditing Non-Privileged Users

Of course, you may also direct audit records pertaining to non-privileged users to the system log by setting AUDIT_TRAIL=OS in addition to AUDIT_SYSLOG_LEVEL. Non-privileged users cannot delete audit trails logging their actions. The search for perpetrators with queries against auditing views, such as DBA_AUDIT_STATEMENT or DBA_AUDIT_OBJECT, is easier than searching the system log. For these reasons, keeping the audit trails of non-privileged users inside the database with AUDIT_TRAIL=DB is preferred. With the latter setting, audit trails are written to the table SYS.AUD$ and may be queried through the aforementioned data dictionary views. Setting AUDIT_TRAIL=NONE switches off auditing of actions by non-privileged users.

3. Test performed with ORACLE DBMS version 10.2.0.3.

After enabling auditing for database connections established by non-privileged users, e.g., as in:

```
SQL> AUDIT CONNECT BY appuser /* audit_trail=os set */;
```

entries similar to the following are written to the syslog facility (example from Solaris):

```
Feb 21 11:41:14 dbserver Oracle Audit[27684]: [ID 930208 user.notice]
SESSIONID: "15" ENTRYID: "1" STATEMENT: "1" USERID: "APPUSER"
USERHOST: "dbserver" TERMINAL: "pts/3" ACTION: "100" RETURNCODE: "0"
COMMENT$TEXT: "Authenticated by: DATABASE" OS$USERID: "oracle"
PRIV$USED: 5
```

Another entry is added to `/var/adm/messages` when a database session ends:

```
Feb 21 11:44:41 dbserver Oracle Audit[27684]: [ID 162490 user.notice]
SESSIONID: "15" ENTRYID: "1" ACTION: "101" RETURNCODE: "0"
LOGOFF$PREAD: "1" LOGOFF$LREAD: "17" LOGOFF$LWRITE: "0" LOGOFF$DEAD:
"0" SESSIONCPU: "2"
```

Note that additional data provided on the actions LOGON (100) and LOGOFF (101) conforms to the columns of the view `DBA_AUDIT_SESSION`. Translation from action numbers to action names is done via the view `AUDIT_ACTIONS` as in this example:

```
SQL> SELECT action, name FROM audit_actions WHERE action IN (100,101)
ACTION NAME
------ ------
   100 LOGON
   101 LOGOFF
```

When `AUDIT_SYSLOG_LEVEL=AUTH.INFO`, `AUDIT_SYS_OPERATIONS=FALSE` and `AUDIT_TRAIL=NONE`, `CONNECT`, `STARTUP`, and `SHUTDOWN` are logged via syslog. With these settings, an instance shutdown on Solaris writes entries similar to the following to `/var/adm/messages`:

```
Feb 21 14:40:01 dbserver Oracle Audit[29036]:[ID 63719 auth.info] ACTION:'SHUTDOWN'
Feb 21 14:40:01 dbserver DATABASE USER: '/'
Feb 21 14:40:01 dbserver PRIVILEGE : SYSDBA
Feb 21 14:40:01 dbserver CLIENT USER: oracle
Feb 21 14:40:01 dbserver CLIENT TERMINAL: pts/3
Feb 21 14:40:01 dbserver STATUS: 0
```

When `AUDIT_SYSLOG_LEVEL=AUTH.INFO`, `AUDIT_SYS_OPERATIONS=TRUE`, and `AUDIT_TRAIL=NONE`, SQL and PL/SQL statements executed with `SYSDBA` or `SYSOPER` privileges are also logged via syslog. Dropping a user after connecting with `/ AS SYSDBA` results in a syslog entry similar to the one shown here:

```
Feb 21 14:46:53 dbserver Oracle Audit[29170]: [ID 853627 auth.info]
ACTION : 'drop user appuser'
Feb 21 14:46:53 dbserver DATABASE USER: '/'
Feb 21 14:46:53 dbserver PRIVILEGE : SYSDBA
Feb 21 14:46:53 dbserver CLIENT USER: oracle
Feb 21 14:46:53 dbserver CLIENT TERMINAL: pts/3
Feb 21 14:46:53 dbserver STATUS: 0
```

Lessons Learned

CONNECT, STARTUP, and SHUTDOWN with SYSDBA or SYSOPER privileges are logged to *.aud files by default in spite of an AUDIT_TRAIL=NONE setting. If AUDIT_SYSLOG_LEVEL is set, the SQL*Plus STARTUP command is logged to a *.aud file in $ORACLE_HOME/rdbms/audit, whereas ALTER DATABASE MOUNT and subsequent commands as well as SHUTDOWN are logged via syslog, since a running instance is required for using the syslog facility and the instance is not yet running when STARTUP is issued.

Setting AUDIT_SYSLOG_LEVEL and AUDIT_SYS_OPERATIONS=TRUE produces additional auditing trail records covering all actions performed with SYSDBA or SYSOPER privileges in the configured syslog log file irrespective of the setting of AUDIT_TRAIL. Intruders who have not managed to break into the account of the UNIX user root, will not be able to remove these audit trail records.

Of course, an intruder who is aware of these features might remove the AUDIT_SYSLOG_LEVEL setting, but at least the parameter change would be logged if an SPFILE is used, and the change would not be in effect immediately since it is a static parameter. You may wish to set AUDIT_SYS_OPERATIONS=FALSE during maintenance operations such as an upgrade (which have to be run as user SYS) to avoid generating large syslog log files.

PGA_AGGREGATE_TARGET

The initialization parameter PGA_AGGREGATE_TARGET is documented in *Oracle9i Database Performance Tuning Guide and Reference Release 2* and in *Oracle Database Performance Tuning Guide 10g Release 2*. The aforementioned Oracle9*i* manual states that the parameters SORT_AREA_SIZE and HASH_AREA_SIZE for manual PGA memory management should not be used, except in Shared Server environments, since Oracle9*i* Shared Server cannot leverage automatic PGA memory management (pages 1–57 and 14–50). The algorithm that governs individual work area sizing for serial and parallel execution is undocumented.

Knowing the undocumented restrictions imposed on work area sizing allows DBAs to set the most appropriate value for PGA_AGGREGATE_TARGET, thus avoiding expensive spilling of work areas to disk and allowing operations to run entirely in memory, realizing significant performance gains. Under rare circumstances it may be desirable to override automatic settings of hidden parameters affected by PGA_AGGREGATE_TARGET.

Introduction to Automatic PGA Memory Management

The program global area (PGA) is a private memory region where server processes allocate memory for operations such as sorts, hash joins, and bitmap merges. Consequently, the PGA memory region is separate from the SGA (system global area). There is even a third memory

region, the UGA (user global area), that holds session and cursor state information. Dedicated server processes allocate UGA memory inside the PGA, whereas shared server processes place the UGA inside the SGA, since it must be accessible to all shared server processes. If the SGA contains a large pool (parameter LARGE_POOL_SIZE), shared server processes place the UGA inside the large pool. In Oracle10*g*, the shared pool, large pool, java pool, streams pool, and the default buffer pool with standard block size[4] can be sized automatically and dynamically with Automatic Shared Memory Management (parameter SGA_TARGET).

In releases prior to Oracle9*i*, several *_AREA_SIZE parameters had to be used to adjust the sizes for various PGA memory regions. Examples of these parameters are SORT_AREA_SIZE and HASH_AREA_SIZE. On UNIX, where the ORACLE DBMS is implemented as a multiprocess architecture, PGA memory could not always be returned to the operating system after a memory-intensive operation. It lingered within the virtual address space of the server process and may have caused paging. Memory thus allocated was also not available to other server processes. There was also no instance-wide limit on PGA memory regions. Since each server process was allowed to allocate memory for an operation up to the limits imposed by *_AREA_SIZE parameters, the instance-wide memory consumption could become extensive in environments with several hundred server processes. Note also that the *_AREA_SIZE parameters enforce a per operation limit, not a per session limit. Since a query may open several cursors simultaneously and each might execute an expensive SELECT that includes an ORDER BY or a hash join, there is no limit on overall memory consumption with the old approach now called manual PGA memory management.

To address these shortcomings, automatic PGA memory management was introduced with Oracle9*i*. On UNIX, it is based on the modern technology of memory mapping, which enables a process to allocate virtual memory and to map it into its virtual address space. Once the memory is no longer needed, it can be returned to the operating system by removing the mapping into the virtual address space. On Solaris, the UNIX system calls used are mmap and munmap. Calls to memory mapping routines by the ORACLE kernel may be traced using truss (Solaris) or strace (Linux).[5] Another interesting utility is pmap (Solaris, Linux). It displays information about the address space of a process, which includes anonymous memory mapped with mmap. Back in the old days of 32-bit computing, this tool provided precisely the information needed to relocate the SGA base address to allow mapping of a larger shared memory segment into the limited virtual address space of a 32-bit program (see Metalink note 1028623.6). Using pmap while a process sorts, reveals how many regions of anonymous memory it has mapped and what their cumulative size is.

Here's an example (29606 is the UNIX process ID of the server process found in V$PROCESS.SPID). The relevant column is "Anon" (anonymous mapped memory).

```
$ pmap -x 29606 | grep Kb
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
        total Kb     934080     888976      63008     806912
```

4. Oracle10*g* supports up to seven buffer pools: five buffer pools varying in block size from 2 KB to 32 KB and two additional buffer pools with standard block size, i.e., the block size set with the parameter DB_BLOCK_SIZE. The latter two buffer pools are the keep and recycle pools. Segments may be placed into the keep or recycle pool with an ALTER TABLE or ALTER INDEX statement as appropriate.
5. The web page titled *Rosetta Stone for UNIX* at http://bhami.com/rosetta.html lists system call tracing utilities for common UNIX systems.

With automatic PGA memory management, so-called work areas are used for operations such as sorts or hash joins. The target cumulative size of all active work areas is specified with the parameter PGA_AGGREGATE_TARGET (PAT). A single process may access several work areas concurrently. Information on automatic PGA memory management and work areas is available by querying dynamic performance views such as V$PGASTAT, V$PROCESS, V$SQL_WORKAREA, and V$SQL_WORKAREA_ACTIVE.

Misconceptions About PGA_AGGREGATE_TARGET

The parameter name PGA_AGGREGATE_TARGET is a name well chosen. What I'm trying to say is that it is what it sounds like—a target, not an absolute limit, merely a target. This means that the actual amount of memory consumed under high load may constantly, or at least intermittently, be higher than the target value. But the implementation of automatic PGA memory management is so well crafted that processes will then release memory, such that the total memory consumption will soon drop below the target, if at all possible. Especially when PL/SQL, which allocates a lot of memory, e.g., for collections such as index-by tables, is executed, the target may be permanently exceeded. Whereas sort memory requirements can be reduced by using temporary segments, PL/SQL memory requirements cannot.

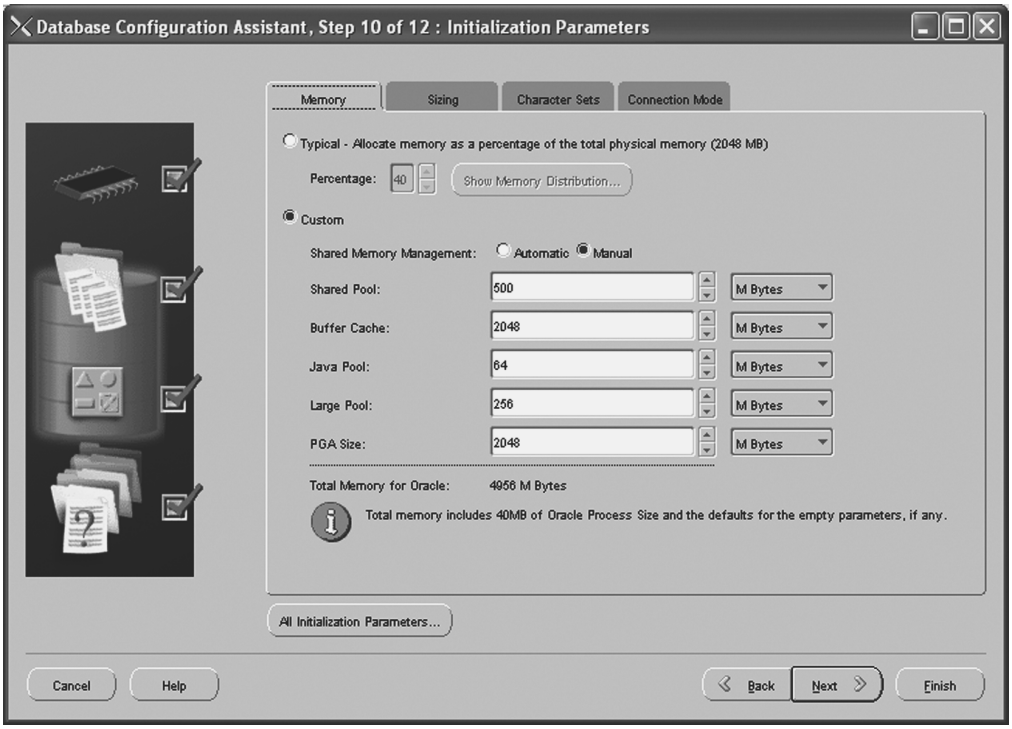

Figure 1-2. *PGA Sizing in Database Configuration Assistant*

When creating a database with the database configuration assistant (DBCA), there is a memory configuration page for customizing most of the aforementioned pools as well as the PGA. On this page, DBCA adds the sizes of all pools as well as the PGA and reports the resulting figure as *Total Memory for Oracle* (see Figure 1-2). This leads some people to believe that this amount of memory (4956 MB in the screenshot) will be allocated when the ORACLE instance is started. Knowing that the SGA is allocated on instance startup, they assume the same must be true for the PGA. However, this is not the case. PGA memory is allocated on demand. Even the `*_AREA_SIZE` parameters do not cause a memory allocation of the designated size. These too are allocated on an as-needed basis.

Since the documentation does not address the details of work area sizing, many database administrators assume that the entire memory set aside with `PGA_AGGREAGTE_TARGET` is available to a single session as long as it does not have to compete for the memory with other sessions. In case you're curious what the real deal is, please read on.

Researching PGA_AGGREGATE_TARGET

The research presented in this section was done with Oracle10*g* Release 2. Results show that the algorithms used by Oracle9*i* and Oracle10*g* are different. Due to space constraints, no example or evidence concerning Oracle9*i* is included.[6]

Creating a Large Table with a Pipelined Table Function

For starters, we need a table that is large enough to cause disk spilling during sort operations. The next few paragraphs show how to code a pipelined table function that returns an arbitrary number of rows (see file `row_factory.sql` in the source code depot). This function may then be used in conjunction with the package `DBMS_RANDOM` to create arbitrarily sized tables with random data. Since pipelined table functions return a collection type, we start by creating an object type for holding a row number.

```
SQL> CREATE OR REPLACE TYPE row_nr_type AS OBJECT (row_nr number);
/
```

The pipelined table function will return a collection type made up of individual `row_nr_types`.

```
SQL> CREATE OR REPLACE TYPE row_nr_type_tab AS TABLE OF row_nr_type;
/
```

The function `row_factory` returns any number of rows—within the limits of the ORACLE `NUMBER` data type, of course. It has the two parameters `first_nr` and `last_nr`, which control how many rows will be returned.

```
CREATE OR REPLACE FUNCTION row_factory(first_nr number, last_nr number)
RETURN row_nr_type_tab PIPELINED
AS
    row_nr row_nr_type:=NEW row_nr_type(0);
```

6. For research on Oracle9*i*, see Jonathan Lewis' article at `http://www.jlcomp.demon.co.uk/untested.html`.

```
BEGIN
    FOR i IN first_nr .. last_nr LOOP
        row_nr.row_nr:=i;
        PIPE ROW(row_nr);
    END LOOP;
    return;
END;
/
```

When last_nr is larger than first_nr, row_factory returns last_nr - first_nr plus one row. The result is very much like SELECT ROWNUM FROM *table*, except that the argument values and not the number of rows in a table control how many rows are returned. Here's an example:

```
SQL> SELECT * FROM TABLE(row_factory(1,2));
    ROW_NR
----------
         1
         2
```

The classic approach for generating a large table consists of selecting from a real table, possibly using a Cartesian join to arrive at a very large number of rows. Beyond requiring less coding for the CREATE TABLE statement, this novel approach using a pipelined table function has the additional benefit of not causing any consistent or physical reads on a segment. By calling row_factory with a first_nr and last_nr setting of 1 and 1000000, we can now create a table with one million rows.

```
SQL> CREATE TABLE random_strings AS
SELECT dbms_random.string('a', 128) AS random_string
FROM TABLE(row_factory(1,1000000))
NOLOGGING;
```

The first argument (opt) tells DBMS_RANDOM to generate random mixed-case strings consisting solely of letters. The second argument (len) controls the length of the random string. Note that in releases prior to Oracle11*g*, arguments to PL/SQL routines cannot be passed by name from SQL.[7]

In my test database with db_block_size=8192, the previous CTAS (create table as select) resulted in a segment size of about 150 MB. DBMS_RANDOM is also capable of generating random alphanumeric strings in lower, upper, or mixed case, as well as random numbers.[8]

V$SQL_WORKAREA_ACTIVE

A good way to monitor PGA memory management at the session level is to query the dynamic performance view V$SQL_WORKAREA_ACTIVE, which has the following columns:

7. In Oracle11*g*, SELECT * FROM TABLE(row_factory(first_nr => 1, last_nr => 3)) is syntactically correct. In prior releases this statement causes ORA-00907.
8. The document *A Security Checklist for Oracle9i* lists DBMS_RANDOM among a list of packages that might be misused and recommends to revoke execute permission on DBMS_RANDOM from PUBLIC (see http://www.oracle.com/technology/deploy/security/oracle9i/pdf/9i_checklist.pdf).

```
SQL> DESC v$sql_workarea_active
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 WORKAREA_ADDRESS                                   RAW(4)
 OPERATION_TYPE                                     VARCHAR2(20)
 OPERATION_ID                                       NUMBER
 POLICY                                             VARCHAR2(6)
 SID                                                NUMBER
 QCINST_ID                                          NUMBER
 QCSID                                              NUMBER
 ACTIVE_TIME                                        NUMBER
 WORK_AREA_SIZE                                     NUMBER
 EXPECTED_SIZE                                      NUMBER
 ACTUAL_MEM_USED                                    NUMBER
 MAX_MEM_USED                                       NUMBER
 NUMBER_PASSES                                      NUMBER
 TEMPSEG_SIZE                                       NUMBER
 TABLESPACE                                         VARCHAR2(31)
 SEGRFNO#                                           NUMBER
 SEGBLK#                                            NUMBER
```

I wrote a small Perl DBI program for closely monitoring the use of PGA work areas. The Perl program executes a `SELECT` on `V$SQL_WORKAREA_ACTIVE` once per second and prints the results to the screen. In addition to the session identifier (which corresponds to `V$SESSION.SID`), the current and maximum work area sizes, and the size of temporary segments, the query also retrieves a timestamp. All sizes are reported in MB. The `SELECT` statement used by the Perl program is as follows:

```
SELECT sid, to_char(sysdate,'mi:ss') time,
round(work_area_size/1048576, 1) work_area_size_mb,
round(max_mem_used/1048576, 1) max_mem_used_mb, number_passes, nvl(tempseg_size/
1048576, 0) tempseg_size_mb
FROM v$sql_workarea_active
ORDER BY sid;
```

Now we have a large table and a monitoring tool. So we're all set to run some actual tests. Since I'm the only tester using the instance, I might assume that the entire memory set aside with `PGA_AGGREGATE_TARGET` will be available to me. As stated before, the segment size of the table is about 150 MB, such that a `PGA_AGGREGATE_TARGET` setting of 256 MB should be more than sufficient for an in-memory sort. So this is the value we will use:

```
SQL> ALTER SYSTEM SET pga_aggregate_target=256m;
System altered.
```

To start monitoring, set the `ORACLE_SID` and DBI environment variables (discussed further in Chapter 22), then run `sql_workarea_active.pl`. The following example is from Windows. On UNIX, use `export` to set environment variables.

```
C:> set ORACLE_SID=ORCL
C:> set DBI_USER=ndebes
C:> set DBI_PASS=secret
C:> set DBI_DSN=DBI:Oracle:
C:> sql_workarea_active.pl
  SID  TIME WORK_AREA_SIZE MAX_MEM_USED PASSES TEMPSEG_SIZE
```

The Perl program does not display any data until one or more work areas are allocated. We will use the script `sort_random_strings.sql` to run `SELECT . . . ORDER BY` in SQL*Plus. Following are the script's contents:

```
set timing on
set autotrace traceonly statistics
SELECT * FROM random_strings ORDER BY 1;
exit
```

The SQL*Plus command `SET AUTOTRACE` with the options `TRACEONLY` and `STATISTICS` is very useful in this context, since it executes the statement without printing the result set to the screen. Furthermore it collects and displays execution statistics from `V$SESSTAT`. In a separate window from the one running `sql_workarea_active.pl`, execute the script `sort_random_strings.sql` with SQL*Plus, as shown here:

```
C:> sqlplus ndebes/secret @sort_random_strings.sql
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
1000000 rows selected.
Elapsed: 00:00:18.73
Statistics
----------------------------------------------------
        133  recursive calls
          7  db block gets
      18879  consistent gets
      16952  physical reads
          0  redo size
  138667083  bytes sent via SQL*Net to client
     733707  bytes received via SQL*Net from client
      66668  SQL*Net roundtrips to/from client
          0  sorts (memory)
          1  sorts (disk)
    1000000  rows processed
```

Surprisingly, the available memory was insufficient and the sort spilled to disk. Following is the output from `sql_workarea_active.pl`, which shows that the session performed a one-pass sort, since it only got a work area size of 51.2 MB:

```
SID  TIME WORK_AREA_SIZE MAX_MEM_USED PASSES TEMPSEG_SIZE
148 31:38           51.2         51.2      0           16
148 31:39           51.2         51.2      0           48
148 31:40           51.2         51.2      1           73
148 31:41           21.4         51.2      1          100
148 31:42           25.8         51.2      1          130
…
148 31:56            2.9         51.2      1          133
```

The timestamps confirm that the statement completed after 18 seconds. The temporary segment grew to 133 MB, somewhat less than the table's segment size. Obviously the entire memory set aside with PAT is not available to a single session. Accordingly, additional undocumented restrictions must be in place. Searching the Internet for "pga_aggregate_target tuning undocumented", one quickly realizes that several hidden parameters impact automatic PGA memory management. The names of the hidden parameters are `_PGA_MAX_SIZE`, `_SMM_MAX_SIZE`, and `_SMM_PX_MAX_SIZE`. Of these, `_PGA_MAX_SIZE` is in bytes and the other two in kilobytes (KB). Descriptions and current values of these parameters are available by querying the X$ fixed tables `X$KSPPI` and `X$KSPPCV` (see Part IV). The script `auto_pga_parameters.sql`, which queries these X$ fixed tables and normalizes all four relevant parameters to kilobytes, is reproduced here:

```
SELECT x.ksppinm name,
CASE WHEN x.ksppinm like '%pga%' THEN to_number(y.ksppstvl)/1024
ELSE to_number(y.ksppstvl)
END AS value,
x.ksppdesc description
FROM x$ksppi x, x$ksppcv y
WHERE x.inst_id = userenv('Instance')
AND y.inst_id = userenv('Instance')
AND x.indx = y.indx
AND x.ksppinm IN ('pga_aggregate_target', '_pga_max_size',

'_smm_max_size', '_smm_px_max_size');
```

With the current settings, the script gives the following result:

```
C:> sqlplus -s / as sysdba @auto_pga_parameters
NAME                 Value (KB) DESCRIPTION
-------------------- ---------- --------------------------------------------
pga_aggregate_target     262144 Target size for the aggregate PGA memory
                                consumed by the instance
_pga_max_size            204800 Maximum size of the PGA memory for one
                                process
_smm_max_size             52428 maximum work area size in auto mode (serial)
_smm_px_max_size         131072 maximum work area size in auto mode (global)
```

I prepared the script pga_aggregate_target_iterator.sql, which varies PGA_AGGREGATE_TARGET between its minimum value of 10 MB up to a maximum value of 32 GB and calls auto_pga_parameters.sql for each setting, to investigate how changing the value of PAT affects these hidden parameters. Shutting down and restarting the DBMS is not necessary since all three parameters are recalculated dynamically within certain limits. The results are presented in Table 1-1. Results beyond 8 GB are omitted, since no change in policy is seen beyond this value.

Table 1-1. *PGA_AGGREGATE_TARGET (PAT) and Dependent Hidden Parameters*

PAT	_pga_max_size (% of PAT)	_smm_max_size (% of _pga_max_size)	_smm_max_size as % of PAT	_smm_px_max_size (% of PAT)
10 MB	200 MB (2000%)	2 MB (1%)	20%	5 MB (50%)
32 MB	200 MB (625%)	6.4 MB (3.2%)	20%	16 MB (50%)
64 MB	200 MB (320%)	12.8 MB (6.4%)	20%	32 MB (50%)
128 MB	200 MB (156%)	25 MB (12%)	20%	64 MB (50%)
256 MB	200 MB (78%)	51 MB (25%)	20%	128 MB (50%)
512 MB	200 MB (39%)	100 MB (50%)	19.5%	256 MB (50%)
1 GB	204 MB (20%)	102 MB (50%)	10%	512 MB (50%)
2 GB	410 MB(20%)	205 MB (50%)	10%	1 GB (50%)
3 GB	416 MB (13.5%)	208 MB (50%)	6.8%	1536 MB (50%)
4 GB	480 MB (11.7%)	240 MB (50%)	5.8%	2 GB (50%)
8 GB	480 MB (5.8%)	240 MB (50%)	2.9%	4 GB (50%)

Looking at Table 1-1, a few patterns emerge. These are addressed in the sections that follow. Note that when overriding the settings of _SMM_MAX_SIZE or _SMM_PX_MAX_SIZE by putting them into a parameter file, these are no longer dynamically adjusted as PGA_AGGREGATE_TARGET is modified. Since both parameters are static, the ability to change them at runtime, albeit indirectly, is lost.

_PGA_MAX_SIZE

The parameter _PGA_MAX_SIZE limits the maximum size of all work areas for a single process.

- For values of PAT below 1 GB, _PGA_MAX_SIZE is 200 MB.
- For values of PAT between 1 GB and 2 GB, _PGA_MAX_SIZE is 20% of PAT.
- At values beyond 2 GB, _PGA_MAX_SIZE keeps on growing as PAT is increased, but at a lower rate, such that _PGA_MAX_SIZE is less than 20% of PAT.
- A limit of 480 MB on _PGA_MAX_SIZE takes effect at a PAT value of 4 GB.

- Increasing PAT beyond 4 GB does not result in higher values of _PGA_MAX_SIZE than 480 MB.
- In Oracle9*i*, _PGA_MAX_SIZE had a limit of 200 MB.

Just like PGA_AGGREGATE_TARGET, _PGA_MAX_SIZE is a dynamic parameter that can be modified with ALTER SYSTEM. Changing _PGA_MAX_SIZE increases _SMM_MAX_SIZE in a similar way that modifying PGA_AGGREGATE_TARGET does. However, the rule that _SMM_MAX_SIZE is 50% of _PGA_MAX_SIZE does not hold for manual changes of _PGA_MAX_SIZE. Following is an example that increases _PGA_MAX_SIZE beyond the limit of 480 MB that can be reached by modifying PGA_AGGREGATE_TARGET:

```
SQL> @auto_pga_parameters
NAME                 Value (KB) DESCRIPTION
-------------------- ---------- ------------------------------------------
pga_aggregate_target    1048576 Target size for the aggregate PGA memory
                                consumed by the instance
_pga_max_size            209700 Maximum size of the PGA memory for one
                                process
_smm_max_size            104850 maximum work area size in auto mode (serial)
_smm_px_max_size         524288 maximum work area size in auto mode (global)
SQL> ALTER SYSTEM SET "_pga_max_size"=500m;
System altered.
SQL> @auto_pga_parameters
NAME                 Value (KB) DESCRIPTION
-------------------- ---------- ------------------------------------------
pga_aggregate_target    1048576 Target size for the aggregate PGA memory
                                consumed by the instance
_pga_max_size            512000 Maximum size of the PGA memory for one
                                process
_smm_max_size            209715 maximum work area size in auto mode (serial)
_smm_px_max_size         524288 maximum work area size in auto mode (global)
```

By increasing _PGA_MAX_SIZE, the work area size(s) available can be increased, without extending the memory allowance for the entire instance. When memory is scarce this might avoid some paging activity. As long as very few sessions concurrently request large work areas, i.e., competition for PGA memory is low, this may lead to better response time for operations involving large sorts. By altering _PGA_MAX_SIZE, _SMM_MAX_SIZE can be dynamically set to values larger than the normal limit of 240 MB.

_SMM_MAX_SIZE

The parameter _SMM_MAX_SIZE limits the maximum size of an individual work area for a single process.

- For values of PAT below 512 MB, _SMM_MAX_SIZE is 20% of PGA_AGGREGATE_TARGET.
- For PAT values of 512 MB and beyond, _SMM_MAX_SIZE is always 50% of _PGA_MAX_SIZE.
- In Oracle9*i*, _SMM_MAX_SIZE had a limit of 100 MB. Following is an example of a session that had two simultaneously active work areas when the given parameters were in effect:

```
NAME                 Value (KB)
-------------------- ----------
pga_aggregate_target    1536000
_pga_max_size            307200
_smm_max_size            153600
_smm_px_max_size         768000

C:> sql_workarea_active_hash.pl
SID  TIME HASH_VALUE       TYPE WORK_AREA_SIZE MAX_MEM_USED PASSES TMP_SIZE
159 57:46 1705656915  SORT (v2)          133.7        133.7      0        0
159 57:46 3957124346  HASH-JOIN           16.2         15.7      0      105
. . .
159 57:52 1705656915  SORT (v2)          133.7        133.7      0        0
159 57:52 3957124346  HASH-JOIN          108.3         96.1      1      138
```

Output from the Perl script `sql_workarea_active_hash.pl` includes the columns `HASH_VALUE` and `TYPE` from `V$SQL_WORKAREA_ACTIVE`. Both work areas combined exceed `_SMM_MAX_SIZE`, but not `_PGA_MAX_SIZE`.

_SMM_PX_MAX_SIZE

The setting of `_SMM_PX_MAX_SIZE` is always 50% of `PGA_AGGREGATE_TARGET`. There is no limit on `_SMM_PX_MAX_SIZE` (at least not within the tested range of `PGA_AGGREGATE_TARGET` of 10 MB to 32 GB). In Oracle9*i*, `_SMM_PX_MAX_SIZE` was 30% of `PGA_AGGREGATE_TARGET`.

Shared Server

In Oracle10*g*, Shared Server was recoded to use automatic PGA memory management. Oracle9*i* Shared Server uses the `*_AREA_SIZE` Parameters, i.e., it behaves as if `ALTER SESSION SET WORKAREA_SIZE_POLICY=MANUAL` had been executed. Hence it is valid to leave `SORT_AREA_SIZE` inside an Oracle9*i* `PFILE` or `SPFILE` and to set it to a more useful value, such as 1048576, than the default 65536. Of course it is still valid to set meaningful values for `SORT_AREA_SIZE`, `HASH_AREA_SIZE`, and so on in Oracle10*g*, for sessions that might run with manual work area sizing (`WORKAREA_SIZE_POLICY=MANUAL`).

Parallel Execution

The hidden parameter `_SMM_PX_MAX_SIZE` applies to parallel execution, but exactly how needs to be revealed by further tests. Regarding parallel execution (PX), it is important to bear in mind that a parallel full scan of a table at degree *n* divides the work among *n* parallel execution processes, such that the volume of data handled by each process equates to approximately one *n*th of the entire data volume. The figure *n* is commonly called the degree of parallelism or DOP.

Each parallel execution process allocates its own work area(s). Since each process handles merely a fraction of the data, the work areas required by individual processes in parallel mode are smaller than a single work area in serial mode.

It turns out that `_SMM_PX_MAX_SIZE` places an additional restriction on the maximum work area size, which is exercised on parallel execution processes. Each PX process may not use more than `_SMM_PX_MAX_SIZE`/DOP memory. The per process restriction of `_SMM_MAX_SIZE`

remains in effect for PX, such that the available memory is the lesser of _SMM_MAX_SIZE and _SMM_PX_MAX_SIZE/DOP. To sort entirely in memory, these two conditions must be met:

- The data volume per PX process must be less than _SMM_MAX_SIZE.
- The data volume per PX process must be less than _SMM_PX_MAX_SIZE/DOP.

Let's run some examples. The previous tests revealed that the SELECT from the test table has a data volume of about 133 MB. Thus, at a DOP of four, each PX process requires a work area size of around 133 MB divided by 4, or approximately 34 MB for an optimal sort. Rounding up slightly to 40 MB to allow for fluctuations of the data volume among PX processes, we will set _SMM_MAX_SIZE=40960, since the unit of _SMM_MAX_SIZE is KB. To avoid PGA_AGGREGATE_TARGET or _SMM_PX_MAX_SIZE becoming the limiting factor, we also set both parameters to DOP times _SMM_MAX_SIZE or 160 MB. To set these parameters, place the following three lines into a parameter file and restart the instance with STARTUP PFILE:

```
pga_aggregate_target=160m
_smm_px_max_size=163840 # in KB
_smm_max_size=40960 # in KB
```

Verifying the settings with the script auto_pga_parameters.sql gives this result:

```
NAME                 Value (KB) DESCRIPTION
-------------------- ---------- ------------------------------------------
pga_aggregate_target     163840 Target size for the aggregate PGA memory
                                consumed by the instance
_pga_max_size            204800 Maximum size of the PGA memory for one
                                process
_smm_max_size             40960 maximum work area size in auto mode (serial)
_smm_px_max_size         163840 maximum work area size in auto mode (global)
```

Next, a FULL and a PARALLEL hint must be added to the SELECT statement to enable parallel execution.

```
SQL> SELECT /*+ FULL(r) PARALLEL(r, 4) */ * FROM random_strings r ORDER BY 1;
```

Running the parallel query at a DOP of four and monitoring it with sql_workarea_active.pl gives this:

```
  SID  TIME WORK_AREA_SIZE MAX_MEM_USED PASSES TEMPSEG_SIZE
  143 06:36            1.7          1.6      0            0
  144 06:36            1.7          1.6      0            0
  145 06:36            2.6          2.1      0            0
  146 06:36            1.7          1.6      0            0
  …
  145 06:43           32.3         39.6      0            0
  …
  146 06:46           31.6         31.6      0            0
  …
  144 06:48           31.4         31.4      0            0
  …
  143 06:50           31.2         31.2      0            0
```

As expected, an optimal sort was performed. The response time is 14 s. Halving DOP results in only two processes sharing the workload and the following measurements:

```
SID  TIME WORK_AREA_SIZE MAX_MEM_USED PASSES TEMPSEG_SIZE
140 23:48            3.1          2.7      0            0
147 23:48            3.8          3.1      0            0
…
147 24:03            1.2           40      1           71
…
140 24:08            1.2           40      1           63
```

Here, _SMM_MAX_SIZE leads to a degradation of response time to around 20 s, since at DOP two each process requires a work area size of around 75 MB, but only 40 MB was available, resulting in one-pass sorts and spilling to disk. Now back to the original DOP of four—a reduction of _SMM_PX_MAX_SIZE below the data volume divided by DOP also results in spilling to disk. Following are the results at DOP four with these settings:

```
pga_aggregate_target=160m
_smm_px_max_size=122880 # in KB
_smm_max_size=40960 # in KB
```

This time, _SMM_PX_MAX_SIZE is the limiting factor.

```
SID  TIME WORK_AREA_SIZE MAX_MEM_USED PASSES TEMPSEG_SIZE
143 33:27            1.7          1.7      0            0
…
145 33:41            1.2           30      1           40
…
146 33:44            1.2           30      1           32
…
144 33:46            1.2           30      1           32
…
143 33:49            1.2           30      1           31
```

All slaves spilled their work areas to disk, since work areas were limited to 120 MB/DOP=40 MB and the query completed in 22 s.

Lessons Learned

When using automatic PGA memory management, three hidden parameters—_PGA_MAX_SIZE, _SMM_MAX_SIZE, and _SMM_PX_MAX_SIZE—work behind the scenes to enforce restrictions on memory consumption. The parameter _PGA_MAX_SIZE limits the size of all work areas in use by a single process. The size of an individual work area is limited by _SMM_MAX_SIZE for both serial and parallel execution. When parallel execution is used, an additional restriction on the total size of all work areas in use by the processes involved is in place. This limit is controlled with the parameter _SMM_PX_MAX_SIZE. Within certain limits, all three parameters are recalculated at runtime as a result of modifying PAT. All three parameters may be set manually to override the result of this calculation.

EVENT

The initialization parameter EVENT is partially documented in the *Oracle Database Reference* manual. The parameter syntax as well as which events may be set are undocumented. The manual states that the parameter must not be used except under the supervision of Oracle Support Services.

The parameter EVENT may be used to set one or more events at instance level. Events set in this way are enabled for the entire lifetime of an instance. All other approaches for setting events, such as DBMS_SYSTEM, do not cover the entire lifetime of an instance. The parameter is appropriate for situations where other means for setting events are not feasible or events must be set right when a process starts. Processing of a technical assistance request by Oracle Support Services may involve setting certain events. A DBA who is familiar with the parameter EVENT is less dependent on Oracle Support and may find a workaround or gather diagnostic data without needing to ask for assistance.

Syntax

The events that may be set with the parameter EVENT are the same events that can be set by other means, such as ALTER SESSION, ALTER SYSTEM, and ORADEBUG. The commonalities go even further, since the event specification syntax for the aforementioned methods and the parameter EVENT is identical. Multiple events may be set by entering several event specifications separated by colons. The syntax is:

```
event='event_specification1[:event_specificationN]*'
```

Brackets indicate that an element is optional. The asterisk indicates that the preceding element may be repeated. The syntax for an individual event specification is as follows:

```
event_number trace name context forever, level event_level
```

The placeholders *event_number* and *event_level* are both integers. Most event numbers are in the range 10000 to 10999.[9] On UNIX systems, these events are listed in the file $ORACLE_HOME/rdbms/mesg/oraus.msg along with a description. The supported event level is unspecified for most of the events in the file, such that it may be necessary to involve Oracle Support to determine the correct level. The OERR utility may be used to retrieve the description for a certain event. Following is an example for an event that switches off a cost-based optimizer (CBO) access path:

```
$ oerr ora 10196
10196, 00000, "CBO disable index skip scan"
// *Cause:
// *Action:
```

It is also possible to request a diagnostic dump when an ORA-*nnnnn* error occurs. The syntax for this is identical to the syntax that must be used with ALTER SESSION/SYSTEM SET EVENTS (covered in Chapter 13). Further information on events is provided in Part III.

9. Events 14532 (enable bug fix for excess use of shared pool memory during DDL on partitioned objects in 10.2.0.3) and 38068 (CBO enable override of guess impact on index choice) are example exceptions to this rule.

Leveraging Events at the Instance-Level

Several scenarios mandate setting events at the instance level. These are:

- Enabling or disabling bug fixes
- Enabling or disabling features, such as cost-based optimizer access paths
- Tracing certain code paths or features in all processes of an instance
- Enabling or disabling certain checks
- Writing a diagnostic dump whenever an ORACLE error (ORA-*nnnnn*) is raised in any database session

Consider the parameter `EVENT` whenever events must be set right when a process starts or for all processes of an instance. While it is absolutely feasible to obtain SQL trace files for multiple processes, such as those originating from a connection pool by setting event 10046 with parameter `EVENT`, I am a strong opponent of such a procedure because more sophisticated approaches, such as using a logon trigger or `DBMS_MONITOR` exist. Setting event 10046 at level 8 or 12 with the parameter `EVENT` is better than setting `SQL_TRACE=TRUE` at the instance level, since wait events and binds may be included; however, both incur the unnecessary overhead of tracing each and every process of the instance. I certainly wouldn't be willing to sift through dozens or even hundreds of trace files to find a few relevant ones when other features allow tracing just the processes of interest.

Case Study

Recovery Manager (RMAN) supports writing backups to file systems and to third party media managers. Writing backups to a file system works out of the box and does not incur additional expenses. Since writing to a local file system does not protect against the failure of the database server hardware as a whole, writing to remote file systems or network-attached storage (NAS) arrays via NFS is supported too. RMAN has several undocumented requirements concerning NFS mount options. If the mount options `hard,rsize=32768,wsize=32768` are not used, RMAN will refuse to write to an NFS file system. However, certain releases of RMAN still throw an error when these requirements are met. Under these circumstances, Oracle Support has suggested setting event 10298 at level 32 as a temporary workaround until the underlying issue is resolved.

This is a case for setting an event at the instance level with parameter `EVENT`. With other methods for setting events, such as `ORADEBUG` or `DBMS_SYSTEM`, it is impossible to set the event in time for the multiple processes that RMAN spawns. Furthermore it would be too cumbersome to set the event after each instance startup with `ALTER SYSTEM SET EVENTS`.

OS_AUTHENT_PREFIX

The initialization parameter `OS_AUTHENT_PREFIX` is documented in the *Oracle Database Reference* manual. It is undocumented that a database user name that is prefixed by the string `OPS$` allows for local authentication through the operating system and password authentication when connecting over a network.

Since `REMOTE_OS_AUTHENT=FALSE` should be set for security reasons, it's impossible to use externally-identified users to connect to an instance over a network, e.g., using Oracle Net and

the TCP/IP protocol adapter. Creating OPS$ users with password authentication allows the convenience of omitting the user name and password when connecting locally using the Oracle Net bequeath adapter, while being able to connect over a network using password authentication.

OPS$ Database Users and Password Authentication

Operating system authentication is intended for local connections. The *Oracle Database SQL Reference 10g Release 2* manual states the following on externally-identified users:

> *EXTERNALLY Clause*
>
> *Specify EXTERNALLY to create an external user. Such a user must be authenticated by an external service, such as an operating system or a third-party service. In this case, Oracle Database relies on authentication by the operating system or third-party service to ensure that a specific external user has access to a specific database user.*

In the same way that a user who belongs to the DBA group (usually the UNIX group dba) can connect with SYSDBA privileges without entering a password using CONNECT / AS SYSDBA, an externally-identified user can connect using CONNECT /. When verifying credentials for an externally-identified user, the value of the ORACLE initialization parameter OS_AUTHENT_PREFIX is prepended to the operating system user name. If the resulting user name exists in the data dictionary and DBA_USERS.PASSWORD=EXTERNAL for this user, then the user may connect without entering a password. The syntax for creating an externally-identified user is as follows:

```
CREATE USER <os_authent_prefix><os_user_name> IDENTIFIED EXTERNALLY;
```

It is undocumented that operating system authentication also works for users created with password authentication as long as OS_AUTHENT_PREFIX is left at its default setting of ops$. That is, users created with the syntax CREATE USER ops$*os_user_name* IDENTIFIED BY *password* may connect locally without entering a password as long as OS_AUTHENT_PREFIX=ops$. In a way, this approach combines the best of both worlds. The need to enter passwords for interactive database sessions as well as storing passwords for batch jobs running locally is dispelled and the same user name may be used to connect over the network.

Case Study

The environment for this case study is a UNIX system, where the DBA group name is "dba", the OPER group name is "oper", and the ORACLE software owner group is "oinstall". Furthermore, a password file is used. In a moment you will see how a user who is not a member of any of the aforementioned three special groups may be granted the SYSOPER privilege, allowing him to start and stop an instance, while not being able to change parameters or to modify the ORACLE software installation. This is an additional option that may be implemented with the undocumented approach discussed in the previous section.

First of all, we verify that the parameter OS_AUTHENT_PREFIX has the default value ops$.

```
SQL> SHOW PARAMETER os_authent_prefix
NAME                                 TYPE        VALUE
------------------------------------ ----------- -----
os_authent_prefix                    string      ops$
```

Next, we create a database user whose name is formed by prepending the string ops$ to the operating system user name, in this case "ndebes", and grant the privileges CONNECT and SYSOPER to the new user.

```
SQL> CREATE USER ops$ndebes IDENTIFIED BY secret;
User created.
SQL> GRANT CONNECT, SYSOPER TO ops$ndebes;
Grant succeeded.
SQL> SELECT * FROM v$pwfile_users;
USERNAME                       SYSDBA SYSOPER
------------------------------ ------ -----
SYS                            TRUE   TRUE
OPS$NDEBES                     FALSE  TRUE
```

As evidenced by Figure 1-3, the database user OPS$NDEBES can connect via the Oracle Net TCP/IP adapter from a Windows system. Password authentication is required, since REMOTE_OS_AUTHENT=FALSE is set.

Figure 1-3. *SQL*Plus Session via the Oracle Net TCP/IP Adapter*

Back on UNIX, the operating system user "ndebes" can connect without entering a password.

```
$ id
uid=500(ndebes) gid=100(users) groups=100(users)
$ sqlplus /
SQL*Plus: Release 10.2.0.3.0 - Production on Wed Sep 5 08:02:33 2007
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production
SQL> SHOW USER
USER is "OPS$NDEBES"
```

Thanks to password authentication and a password file, connecting AS SYSOPER works too.

```
SQL> CONNECT ops$ndebes/secret AS SYSOPER
Connected.
SQL> SHOW USER
USER is "PUBLIC"
SQL> SELECT * FROM session_privs;
PRIVILEGE
```

```
----------------------------------------
CREATE SESSION
RESTRICTED SESSION
SYSOPER
```

Due to the SYSOPER privilege, the database user "OPS$NDEBES" can stop and restart the instance.

```
SQL> SHUTDOWN IMMEDIATE
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> STARTUP
ORACLE instance started.
Database mounted.
Database opened.
```

Contrary to SYSDBA, the SYSOPER privilege does not include access to data dictionary views or tables, but allows the use of ARCHIVE LOG LIST for monitoring. Merely database objects accessible to PUBLIC may be accessed with the SYSOPER privilege.

```
SQL> SELECT startup_time FROM v$instance;
SELECT startup_time FROM v$instance
                         *
ERROR at line 1:
ORA-00942: table or view does not exist
SQL> ARCHIVE LOG LIST
Database log mode              No Archive Mode
Automatic archival             Disabled
Archive destination            /opt/oracle/product/db10.2/dbs/arch
Oldest online log sequence     18
Current log sequence           19
```

The combined benefits of operating system and password authentication become unavailable with a nondefault setting of OS_AUTHENT_PREFIX. The SYSDBA privilege can merely be granted to database users created with password authentication, but obviously such users must enter the correct password when connecting. The problem is that the undocumented check for operating system authentication in spite of an assigned password is not done when OS_AUTHENT_PREFIX has a nondefault value.

```
SQL> ALTER SYSTEM SET os_authent_prefix='' SCOPE=SPFILE;
System altered.
```

Since OS_AUTHENT_PREFIX is now a zero-length string, operating system user name and database user name are identical.

```
SQL> CREATE USER ndebes IDENTIFIED BY secret;
User created.
SQL> GRANT CONNECT, SYSOPER TO ndebes;
Grant succeeded.
```

To allow the changed value of OS_AUTHENT_PREFIX to take effect, the instance must be restarted. Clearly, the operating system user "ndebes" will not be able to connect as database user "ndebes" without entering the password "secret".

```
$ sqlplus -s /
ERROR:
ORA-01017: invalid username/password; logon denied
```

When setting the authentication method for the user to operating system authentication, the string "EXTERNAL" instead of a password hash is stored in DBA_USERS.PASSWORD.

```
SQL> ALTER USER ndebes IDENTIFIED externally;
User altered.
SQL> SELECT password FROM dba_users WHERE username='NDEBES';
PASSWORD
------------------------------
EXTERNAL
```

Now the operating system user "ndebes" is able to connect without entering the password.

```
$ id
uid=500(ndebes) gid=100(users) groups=100(users)
$ sqlplus /
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production
SQL> CONNECT ndebes/secret as SYSOPER
ERROR:
ORA-01031: insufficient privileges
```

However, the ability to connect as SYSOPER using the password stored in the password file is lost for the now externally-identified database user. The same applies to the privilege SYSDBA.

Lessons Learned

There is an undocumented code path that enables operating system authentication for database users whose user names start with OPS$, even when these users are created with password authentication. This combines the best aspects of the otherwise mutually-exclusive approaches of operating system and password authentication. To leverage the undocumented feature, the initialization parameter OS_AUTHENT_PREFIX must have the default value ops$. The feature may also be used to set up a single database user with SYSDBA or SYSOPER privileges who does not belong to the DBA or OPER operating system groups and who can connect locally without entering a password. Such a user must only enter the password when connecting over the network or when needing a session with SYSDBA or SYSOPER privileges. Separate database users are required without the undocumented feature or if a nondefault setting of OS_AUTHENT_PREFIX is in effect. If you are dealing with a security-sensitive environment and need to make sure that an intruder cannot exploit this feature, you should disable it by assigning a nondefault value to the parameter OS_AUTHENT_PREFIX.

Source Code Depot

Table 1-2 lists this chapter's source files and their functionality.

Table 1-2. *Partially Documented Parameters Source Code Depot*

File Name	Functionality
auto_pga_parameters.sql	Retrieves all the documented and undocumented parameters that affect SQL work area sizing.
pga_aggregate_target_iterator.sql	This script varies PGA_AGGREGATE_TARGET between 10 MB and 32 GB and calls auto_pga_parameters.sql at each iteration.
row_factory.sql	Creates a pipelined table function that returns an arbitrary number of rows.
sort_random_strings.sql	This script enables SQL*Plus AUTOTRACE and selects rows from the test table RANDOM_STRINGS.
sql_workarea_active.pl	This Perl script monitors work areas by querying the dynamic performance view V$SQL_WORKAREA_ACTIVE.
sql_workarea_active_hash.pl	This Perl script monitors work areas by querying the dynamic performance view V$SQL_WORKAREA_ACTIVE. The output includes SQL statement hash values.