CHAPTER 2

Oracle Internals
Another name for this chapter could be “The Guts of Oracle.” What is it doing in there? It is obvious that the inside workings of SQL Server and Oracle are not the same, or they wouldn’t be two different database platforms. Understanding how the internal and system structures are set up in Oracle will give you insight into some of the best practices for Oracle.

In this chapter, we will focus on configurations and how the memory and system areas are organized. There are also Oracle processes or services to get to know. Then we will take a look at some of the knobs that can be turned for options of the database. Finally, we will examine how changes and transactions are handled by the logs and processes.

### Memory Structures

Databases use memory to cache data blocks for fast access. They have some processes that use memory for sorting or calculations, and other processes that use the memory allocated to cache results.

SQL Server has minimum and maximum values for the memory available for the server. Memory it uses is limited to the memory available on the server. The minimum value does not affect how much memory SQL Server will start with, but rather up to what point it will give back memory to the operating system if the memory isn’t being used. Planning the memory for a SQL Server system is based on how many database instances and application processes will be running on the server.

Oracle also uses the memory available on the server. Oracle can dynamically allocate memory between the different memory structures under the server and process area, and with Oracle Database 11g, even between the server and user process areas. There are parameter settings for maximum values, dynamic allocation, and configuring the operating system to have shared memory available for Oracle to use. As with SQL Server, planning for memory is based on how many database instances and application processes will be running on the server.

For either database system, it is not good practice to allocate all of the memory available on the server to the database. The operating system also needs space for its operations.
Oracle Memory Parameters

With Oracle Database 11g's Automatic Shared Memory Management (ASMM) feature, the management of Oracle's various memory parameters has essentially come down to setting one parameter. And if there were no more 9i or 10g databases out there, or if all applications used memory in the optimal way, memory management would be simple. However, just as some SQL Server 2000 and 2005 servers are still in use, earlier versions of Oracle remain in service. So, you do need an understanding of how Oracle uses memory.

The two main memory areas for Oracle are the System Global Area (SGA) and the Program Global Area (PGA). Under the SGA, the memory is divided into other areas for handling the SQL statements, data blocks, and log buffers. The PGA is the workload area for server processes. Figure 2-1 shows the memory parameters for the SGA and PGA.

In Oracle9i Database and Oracle Database 10g, the dynamic memory parameters allow the memory to adjust within the SGA. The SGA_MAX_SIZE and SGA_TARGET parameters are set, and then memory is adjusted between DB_CACHE_SIZE, SHARED_POOL_SIZE, and the other pools (such as LARGE_POOL_SIZE and JAVA_POOL_SIZE). This helps for systems that might have different types of workload at different times. Without manual intervention, the allocations could adjust based on the memory needs of the

<table>
<thead>
<tr>
<th>SGA</th>
<th>Memory Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>sgamax_size</td>
<td>memory_max_size (11g)</td>
</tr>
<tr>
<td>sgatarget</td>
<td>memory_target (11g)</td>
</tr>
<tr>
<td>Buffer Cache</td>
<td></td>
</tr>
<tr>
<td>dbcache_size</td>
<td>Java Pool</td>
</tr>
<tr>
<td>Different Block Size Cache</td>
<td></td>
</tr>
<tr>
<td>dbnksize</td>
<td>java_pool_size</td>
</tr>
<tr>
<td>(n = 2, 4, 8, 16 or 32)</td>
<td></td>
</tr>
<tr>
<td>Large Pool</td>
<td></td>
</tr>
<tr>
<td>large_pool_size</td>
<td></td>
</tr>
<tr>
<td>Log Buffer</td>
<td></td>
</tr>
<tr>
<td>logbuffer</td>
<td></td>
</tr>
<tr>
<td>Other Caches</td>
<td></td>
</tr>
<tr>
<td>dbrccache_size</td>
<td></td>
</tr>
<tr>
<td>dbkeep_cache_size</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PGA</th>
<th>Memory Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>workarea_size_policy=AUTO</td>
<td>pga_aggregate_target</td>
</tr>
<tr>
<td>workarea_size_policy=MANUAL</td>
<td></td>
</tr>
<tr>
<td>sort_area_size</td>
<td></td>
</tr>
<tr>
<td>sorn_area_retained_size</td>
<td></td>
</tr>
<tr>
<td>hash_area_size</td>
<td></td>
</tr>
<tr>
<td>bitmap_merge_area_size</td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 2-1. Memory parameters for the SGA and PGA**
different areas. Of course, in setting the SGA_MAX_SIZE and SGA_TARGET parameters, the statistics must be at the typical level for the correct information to be collected to provide the details required to adjust the memory areas. But why not just set SGA_TARGET and SGA_MAX_SIZE to the same values, if you are allocating a maximum value of memory to Oracle? And, in that case, why not have just one parameter to set?

In Oracle Database 11g using ASMM, you can simply set MEMORY_TARGET and let Oracle handle the rest. In this version, the memory allocation on the operating system side is divided into smaller chunks. Shared memory segments are available for Oracle to use for the SGA.

**NOTE**

Oracle Database 11g also has the parameter MEMORY_MAX_TARGET, which allows you to specify the maximum setting for the MEMORY_TARGET parameter. However, when you set MEMORY_TARGET, the MEMORY_MAX_TARGET parameter will be set to the same value automatically, so you don’t need to set MEMORY_MAX_TARGET directly.

On the Linux platform, Oracle uses shared memory in /dev/shm. Here is a typical error message that will come up if the operating system doesn’t have enough memory to mount the /dev/shm file system:

```sql
SQL> startup
ORA-00845: MEMORY_TARGET not supported on this system
In the alert log:
Starting ORACLE instance (normal)
WARNING: You are trying to use the MEMORY_TARGET feature. This feature requires the /dev/shm file system to be mounted for at least 4294967296 bytes. /dev/shm is either not mounted or is mounted with available space less than this size. Please fix this so that MEMORY_TARGET can work as expected. Current available is 0 and used is 0 bytes.
```

**NOTE**

I’m using Linux in this example just to give you an idea about running Oracle on another operating system. Chapter 3 covers using Oracle on a Linux platform.
Using operating system memory in this way is a new shift in the Oracle Database 11g approach. Earlier versions used the System V-style shared memory, and you could verify the size of the shared memory used by Oracle using the operating system command `ipcs -b` which shows what semaphores have been allocated. To be able to view the memory allocated to Oracle with the POSIX-style shared memory, the OS commands for checking the space used in the file system are used, as in the following example.

```
$df -k /dev/shm
Filesystem 1K-blocks Used Available Use% Mounted on
32486028 180068 32305960 1% /dev/shm
```

Using the memory in Windows for Oracle is similar to using it for SQL Server. Address Windowing Extensions (AWE) and the Windows 4GB RAM Tuning feature are options available for the Oracle database, too. Using a Very Large Memory (VLM) configuration has been available for Oracle on Windows since Oracle 8i.

Oracle Database 11g on Windows can take advantage of AWE to use more than 3GB of memory. Also, setting the `/3GB` switch in the boot.ini file will at least allow for using about 3GB of memory for Oracle. To use up to 64GB of memory, the `/PAE` switch needs to be enabled. Physical Address Extension (PAE) allows for mapping of a virtual addressable space above the 4GB of memory. Having both the `/3GB` and `/PAE` switches enabled at the same time will allow only 16GB of memory to be available, so the `/3GB` switch should be disabled to allow for more memory to be used by the PAE. The memory limitations are really applicable only on 32-bit Windows systems. With 64-bit systems, the limitations are measured in terabytes.

Windows supports the use of large pages for systems using a large amount of memory. The parameter in the Oracle key of the registry needs to be set as `ORA_LPENABLE=1` to enable the large pages. In order to use VLM on Windows, the oracle user needs the “Lock memory pages” privilege. The `USE_INDIRECT_DAT_BUFFERS=TRUE` parameter must be set in the parameter file for Oracle. Also, the `DB_BLOCK_BUFFERS` parameter must be set for the database cache.

The dynamic SGA parameters are not available for the very large memory settings. If the system doesn’t need more than the 3GB of memory for the SGA, you should consider just using the 4GB RAM Tuning feature, so the dynamic parameters are available.
Again, with Oracle Database 11g, you can simply set the MEMORY_TARGET parameter and have Oracle manage the rest. However, adjusting some of the other memory parameters may improve the performance of particular applications. When used in combination with ASMM, the settings of the individual parameters are implemented as minimum values.

### Sizing the SGA and PGA

As discussed in the previous section, with the new features of Oracle Database 11g, the configuration of each individual parameter for memory has become less important. Setting the MEMORY_TARGET is a simple way to manage the memory, even between the SGA and PGA. However, appropriately sizing the SGA and PGA memory remains important for Oracle database performance.

#### SGA Considerations

Several views provide SGA information. To look at the current sizing of the SGA, use `v$sga` and `v$sgainfo`. The `v$sgainfo` view shows the current sizes and which areas can be resized. The resizeable areas make up the variable size with the database buffers in `v$sga`.

```sql
SQL> select * from v$sga;
NAME       VALUE
----------------------- ----------
Fixed Size           2086288
Variable Size        939526768
Database Buffers     1677721600
Redo Buffers         14688256

SQL> select * from v$sgainfo;
NAME         BYTES   RESIZEABLE
------------- -------- ----------
Fixed SGA Size 2086288  No
Redo Buffers   14688256 No
Buffer Cache Size 1677721600 Yes
Shared Pool Size 889192448  Yes
Large Pool Size 16777216  Yes
Java Pool Size  16777216  Yes
Streams Pool Size 16777216  Yes
Granule Size    16777216  No
Maximum SGA Size 2634022912 No
Startup overhead in Shared Pool 201326592 No
Free SGA Memory Available 0
```
To see which objects are using the current memory areas, use the 
\texttt{v\$sgastat} view.

To get assistance in sizing the database cache, use the \texttt{v\$db_cache_advice} view.

\begin{verbatim}
SQLPLUS> select size_for_estimate, buffers_for_estimate, 
estd_physical_read_factor, estd_physical_reads 
from v$db_cache_advice 
where name = 'DEFAULT' and block_size = (select value from v$parameter 
where name='db_block_size') 
and advice_status = 'ON';
\end{verbatim}

<table>
<thead>
<tr>
<th>Size_for_est</th>
<th>buffer_for_est</th>
<th>estd_physical_read_factor</th>
<th>estd_physical_reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>160</td>
<td>19790</td>
<td>1.8477</td>
<td>38053244</td>
</tr>
<tr>
<td>320</td>
<td>39580</td>
<td>1.3063</td>
<td>26904159</td>
</tr>
<tr>
<td>480</td>
<td>59370</td>
<td>1.2149</td>
<td>25067132</td>
</tr>
<tr>
<td>640</td>
<td>79160</td>
<td>1.2016</td>
<td>24746320</td>
</tr>
<tr>
<td>800</td>
<td>98950</td>
<td>1.1884</td>
<td>24474411</td>
</tr>
<tr>
<td>960</td>
<td>118740</td>
<td>1.1792</td>
<td>24284735</td>
</tr>
<tr>
<td>1120</td>
<td>138530</td>
<td>1.1762</td>
<td>24223738</td>
</tr>
<tr>
<td>1280</td>
<td>158320</td>
<td>1.1421</td>
<td>21459758</td>
</tr>
<tr>
<td>1440</td>
<td>178110</td>
<td>1.0370</td>
<td>21376570</td>
</tr>
<tr>
<td>1600</td>
<td>197900</td>
<td>1</td>
<td>20595061</td>
</tr>
<tr>
<td>1760</td>
<td>217690</td>
<td>.9959</td>
<td>20510626</td>
</tr>
<tr>
<td>1920</td>
<td>237480</td>
<td>.9938</td>
<td>20466583</td>
</tr>
<tr>
<td>2080</td>
<td>257270</td>
<td>.9921</td>
<td>20431565</td>
</tr>
<tr>
<td>2240</td>
<td>277060</td>
<td>.9908</td>
<td>20405971</td>
</tr>
<tr>
<td>2400</td>
<td>296850</td>
<td>.9902</td>
<td>20393666</td>
</tr>
<tr>
<td>2560</td>
<td>316640</td>
<td>.9895</td>
<td>20379145</td>
</tr>
<tr>
<td>2720</td>
<td>336430</td>
<td>.9884</td>
<td>20356415</td>
</tr>
<tr>
<td>2880</td>
<td>356220</td>
<td>.9848</td>
<td>20281604</td>
</tr>
<tr>
<td>3040</td>
<td>376010</td>
<td>.9808</td>
<td>20199710</td>
</tr>
<tr>
<td>3200</td>
<td>395800</td>
<td>.972</td>
<td>20018812</td>
</tr>
</tbody>
</table>
\end{verbatim}

As you can see in this example, there is a point of diminishing returns for 
the amount of memory set and the reduction of physical reads. Even though 
there is a decrease in physical reads with settings higher than 1600, the 
decrease is not that significant. Just throwing memory at the database cache 
may not help the performance of the database.

Since block reads from memory are normally faster than going to disk 
to get the data block, why don’t we size the memory to hold the whole 
database? Well, for large databases (talking well into terabytes), this isn’t 
normally cost-effective. Of course, with different types of hardware, solid-
state disks and flash memory cards could be used as part of a solution. For 
smaller databases—say, one that might be 20GB—you could have 20GB of 
memory allocated to the SGA, but that wouldn’t necessarily keep all of the 
data blocks in memory, because the database needs memory for other 
processes.
Also, think about the data being accessed. Is all of the data always being read? And if it is, what about growth? It will be hard to keep up with supplying memory to the server as the size of the database grows. Full scans of tables will flush some of the blocks out of memory, and when code pulls more data than expected, having everything in memory might prove difficult. Tuning queries to pull just the data that is needed might avoid some of these larger scans, at least minimizing the physical reads.

Blocks that are read into the buffer cache are ordered from most recently used (MRU) to least recently used (LRU). Blocks that are read as part of a full-table scan are put on the LRU end. If the buffer cache is full, the LRU blocks will be flushed out of the cache. The goal is to keep the most frequently used data in memory for quicker access. This also includes the code (SQL statements) in the library cache. So, you will want to size the SGA to follow these guidelines, and then tune it as the database changes and grows.

**PGA Considerations**

The PGA is used for the program or user processes. As shown earlier in Figure 2-1, there are manual and automatic options for managing the PGA. Setting the `WORKAREA_SIZE_POLICY= AUTO` parameter has Oracle use the `PGA_AGGREGATE_TARGET` parameter for sizing the user processes for SQL that use memory, such as for sorts, group by, hash joins, and bitmaps. You can find information about PGA usage in the `v$pgastat` view, and also by looking at the maximum values of the `pga_used_mem`, `pga_alloc_mem`, and `pga_max_mem` columns in the `v$process` view. There is also an advice table for PGA, `v$pga_target_advice`, to help determine a good setting for `PGA_AGGREGATE_TARGET`.

**Where Are the master, msdb, and tempdb Databases?**

The SQL Server `master`, `msdb`, and `tempdb` databases do not exist in the Oracle world. In Oracle, other areas keep the system information, provide a way to schedule jobs, and maintain a temporary space for sorting and temporary tables.
System-level Information

For SQL Server databases and logins, the master database has the details. The master database contains the system information and server configurations. So, where is the master database information in Oracle?

In Oracle, the system-level information for the database instance is in the data dictionary, which is stored in the SYSTEM tablespace under the SYS schema. You can query views to retrieve this information about the database and objects contained in the databases and schemas. Here is a small sampling of the information stored and where it can be found on SQL Server and Oracle systems:

<table>
<thead>
<tr>
<th>SQL Server Master Database</th>
<th>Oracle Data Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>dba_users</td>
</tr>
<tr>
<td>Objects</td>
<td>dba_objects</td>
</tr>
<tr>
<td>Tables</td>
<td>dba_tables</td>
</tr>
<tr>
<td>Datafiles</td>
<td>dba_data_files</td>
</tr>
</tbody>
</table>

NOTE
Some of the system tables are new to version SQL Server 2008. There are also system tables at the database level.

There are many more tables in both the SQL Server master database and Oracle data dictionary.

The Oracle catalog also contains system information. The catalog is created when a database is created, and it is updated with upgrades and patches. The catalog.sql and catproc.sql scripts run as part of the Oracle installation, and they create the data dictionary. The GRANT SELECT ANY CATALOG to USER role can be granted to a user to allow read access to the catalog views. This role can have three different levels of permissions: USER_ for those objects owned by the user, ALL_ for any objects for which the user has permissions, and DBA_ for any catalog. As you probably noticed, SYS isn’t included to qualify the name. This is because the public synonyms are set up to allow just using the name of the view.
As an example, let's see how we can get information about the database objects on each platform. Here's the SQL Server query to discover which objects are in the databases:

```
SELECT type_desc, count(1) FROM sys.all_objects
GROUP BY type_desc;
```

RESULTS

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR_STORED_PROCEDURE</td>
<td>3</td>
</tr>
<tr>
<td>DEFAULT_CONSTRAINT</td>
<td>1</td>
</tr>
<tr>
<td>EXTENDED_STORED_PROCEDURE</td>
<td>149</td>
</tr>
<tr>
<td>INTERNAL_TABLE</td>
<td>3</td>
</tr>
<tr>
<td>PRIMARY_KEY_CONSTRAINT</td>
<td>80</td>
</tr>
<tr>
<td>SERVICE_QUEUE</td>
<td>3</td>
</tr>
<tr>
<td>SQL_INLINE_TABLE_VALUED_FUNCTION</td>
<td>19</td>
</tr>
<tr>
<td>SQL_SCALAR_FUNCTION</td>
<td>27</td>
</tr>
<tr>
<td>SQL_STORED_PROCEDURE</td>
<td>1275</td>
</tr>
<tr>
<td>SQL_TABLE_VALUED_FUNCTION</td>
<td>12</td>
</tr>
<tr>
<td>SYSTEM_TABLE</td>
<td>41</td>
</tr>
<tr>
<td>USER_TABLE</td>
<td>82</td>
</tr>
<tr>
<td>VIEW</td>
<td>286</td>
</tr>
</tbody>
</table>

In Oracle, we query `dba_objects` to get information about the database objects:

```
SQLPLUS> SELECT owner, object_type, count(1) FROM dba_objects
       GROUP BY owner, object_type;
```

<table>
<thead>
<tr>
<th>Owner</th>
<th>Object Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMALUCHER</td>
<td>FUNCTION</td>
<td>6</td>
</tr>
<tr>
<td>MMALUCHER</td>
<td>INDEX</td>
<td>149</td>
</tr>
<tr>
<td>MMALUCHER</td>
<td>LOB</td>
<td>14</td>
</tr>
<tr>
<td>MMALUCHER</td>
<td>PACKAGE</td>
<td>310</td>
</tr>
<tr>
<td>MMALUCHER</td>
<td>PACKAGE BODY</td>
<td>236</td>
</tr>
<tr>
<td>MMALUCHER</td>
<td>PROCEDURE</td>
<td>6</td>
</tr>
<tr>
<td>MMALUCHER</td>
<td>SEQUENCE</td>
<td>60</td>
</tr>
<tr>
<td>MMALUCHER</td>
<td>SYNONYM</td>
<td>1</td>
</tr>
<tr>
<td>MMALUCHER</td>
<td>TABLE</td>
<td>133</td>
</tr>
<tr>
<td>MMALUCHER</td>
<td>TRIGGER</td>
<td>158</td>
</tr>
<tr>
<td>MMALUCHER</td>
<td>TYPE</td>
<td>2</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>SYNONYM</td>
<td>20066</td>
</tr>
<tr>
<td>SYS</td>
<td>CLUSTER</td>
<td>10</td>
</tr>
<tr>
<td>SYS</td>
<td>CONSUMER GROUP</td>
<td>5</td>
</tr>
<tr>
<td>SYS</td>
<td>CONTEXT</td>
<td>5</td>
</tr>
<tr>
<td>SYS</td>
<td>DIRECTORY</td>
<td>25</td>
</tr>
</tbody>
</table>
Chapter 2: Oracle Internals

SYS EVALUATION CONTEXT 10
SYS FUNCTION 75
SYS INDEX 718
SYS INDEX PARTITION 216
SYS JAVA CLASS 14747
SYS JAVA DATA 296
SYS JAVA RESOURCE 704
SYS JOB 5
SYS JOB CLASS 2
SYS LIBRARY 115
SYS LOB 112
SYS LOB PARTITION 1
SYS OPERATOR 6
SYS PACKAGE 506
SYS PACKAGE BODY 484
SYS PROCEDURE 56
SYS PROGRAM 4
SYS QUEUE 15
SYS RESOURCE PLAN 3
SYS RULE 4
SYS RULE SET 11
SYS SCHEDULE 2
SYS SEQUENCE 81
SYS SYNONYM 9
SYS TABLE 727
SYS TABLE PARTITION 205
SYS TRIGGER 9
SYS TYPE 1127
SYS TYPE BODY 81
SYS UNDEFINED 6
SYS VIEW 2958
SYS WINDOW 2
SYS WINDOW GROUP 1
SYSMAN EVALUATION CONTEXT 1
SYSMAN FUNCTION 8
SYSMAN INDEX 398
SYSMAN LOB 28
SYSMAN PACKAGE 73
SYSMAN PACKAGE BODY 72
SYSMAN PROCEDURE 2
SYSMAN QUEUE 2
SYSMAN RULE SET 2
SYSMAN SEQUENCE 5
SYSMAN TABLE 342
SYSMAN TRIGGER 48
SYSMAN TYPE 217
SYSMAN TYPE BODY 7
Not only does this query result show the different object types, but it also lists them by schema owner. Here, you see a few different schemas: SYS has the data dictionary, SYSTEM has objects for the database tools, and SYSMAN has the objects for Oracle Enterprise Manager. MMALCHER is just a user schema.

The count of objects will vary by Oracle version and depends on the different components that were installed. Also, the PUBLIC owner has the synonyms available to all users for the queries against the system objects, so they do not need to be fully qualified.

Data Dictionary Views

The Oracle data dictionary views are the place to go to get details about objects and even sizing. Instead of sp_help, you use DESCRIBE or queries that can be run against the dictionary tables. So just as sp_help has been your friend for looking into SQL Server objects, dba_views will become your new Oracle friend. When I want to know what a table looks like, how many objects are owned by a user, or the name of a particular dba_view, I run a quick query to find out.

With so many views available, memorizing them is not a good option. Fortunately, it’s easy to find the view that contains the information you’re seeking. If you know the view has a name that contains segments, tables, stats, or data, you can generate a list of views with that keyword in their name. For example, I know that the dba_view for data files starts with data, and can use this query to find it:

```sql
SQLPLUS> select object_name from dba_objects where object_name like 'DBA_DATA%';
OBJECT_NAME
---------------------------
DBA_DATA_FILES
DBA_DATAPUMP_JOBS
DBA_DATAPUMP_SESSIONS
```
Also, some of the v$ views that contain dynamic information are available even when the database is not open. For example, the v$datafile and v$logfile views show information about the datafiles and redo log files, respectively:

```sql
SQL> select file#, status, (bytes/1024)/1024 size_MB, name from v$datafile;
FILE#  STATUS   SIZE_MB     NAME
------- ------- ---------- --------------------------------------------
   1      SYSTEM    1070 /data/oracle/orcl/system01.dbf
   2      ONLINE   9225 /data/oracle/orcl/undotbs01.dbf
   3      ONLINE   1230 /data/oracle/orcl/sysaux01.dbf
   4      ONLINE 32767.5 /data/oracle/orcl/users01.dbf
   5      ONLINE  14924 /data/oracle/orcl/users02.dbf
   6      ONLINE  12724 /data/oracle/orcl/users03.dbf

SQL> select * from v$logfile order by group#;
GROUP#  STATUS  TYPE  MEMBER
------- ------- ------ -----------------------
    1  ONLINE  /data/oracle/orcl/redo01.log
    1  ONLINE  /data/oracle/orcl/redo01b.log
    2  ONLINE  /data/oracle/orcl/redo02.log
    2  ONLINE  /data/oracle/orcl/redo02b.log
    3  ONLINE  /data/oracle/orcl/redo03.log
    3  ONLINE  /data/oracle/orcl/redo03b.log
    4  ONLINE  /data/oracle/orcl/redo04.log
    4  ONLINE  /data/oracle/orcl/redo04b.log
```

Now we have found that the type of data in SQL Server’s master database type is stored in the Oracle SYS schema. But where are the jobs stored? And what about templates that are used by the model database to create new databases. And do we even look for a tempdb? The information is closer than you might think.
Jobs and Schedules

Scheduling a job is done either via the Oracle Enterprise Manager (OEM) or using the DBMS_SCHEDULER package. If the job is scheduled using DBMS_SCHEDULER, it can be monitored and viewed in OEM. To create a job, a user needs “Select any catalog role” and “Create job” permissions.

There are three main components to a job: schedule, program, and job. The program and job contain the definitions, and the schedule sets regular times for the job to be run. Just as there are maintenance jobs as well as application jobs that can be scheduled in SQL Server, Oracle jobs can be run to take snapshots of the database and gather statistics, as well as create backups. The program can be PL/SQL code or an executable.

The history of jobs and their status is available on the Database Home page of OEM and in DBA_SCHEDULER_JOBS.

SQLPLUS> select owner, job_name, schedule_name, last_start_date, next_run_date from dba_scheduler_jobs;

<table>
<thead>
<tr>
<th>OWNER</th>
<th>JOB_NAME</th>
<th>SCHEDULE_NAME</th>
<th>LAST_START_DATE</th>
<th>NEXT_RUN_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>GATHER_STATS_WEEKLY</td>
<td>WEEKLY_MAINTENANCE_JOB</td>
<td>21-DEC-09</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>AUTO_SPACE_ADVISOR_JOB</td>
<td>MAINTENANCE_WINDOW</td>
<td>26-DEC-09</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>GATHER_STATS_JOB</td>
<td>MAINTENANCE_WINDOW</td>
<td>26-DEC-09</td>
<td></td>
</tr>
</tbody>
</table>

Templates and Temporary Tables

The SQL Server model database has the default template for creating new databases. The Oracle database is created once with the use of the Database Configuration Assistant, script, or template. The schemas are created as users, and the templates or creation scripts can be used to set up other servers that are similar for development or new production environments.

The SQL Server model database is also used to create the tempdb database every time the server is shut down and restarted. It sets the tempdb size and growth of the database. Oracle doesn’t need to re-create a temporary database each time it is started, because it doesn’t have a temporary database. Oracle uses a temporary tablespace with tempfiles that act in this capacity. The temporary area is used for sorting, join operations, and global temporary tables. Similar to the tempdb database, the temporary tablespace cannot store permanent objects, so it doesn’t need to be backed up.

The tempfiles are not fully initialized and are sparse files. Only the header information and last block of the file are created, so sizing on the file system might be off, because the tempfile might not be using all of the space.
that could be allocated to the file. The tempfiles are also not included in the control files. But there is a dictionary view for the tempfiles:

```
SQLPLUS> select file_name,tablespace_name,bytes, status from dba_temp_files;
FILE_NAME      tablespace_name     BYTES      STATUS
/data/oracle/orcl/temp01.dbf   TEMP    5368709120 AVAILABLE
```

A database has a default TEMP tablespace, and a database can also have more than one temporary tablespace. So, users can fill up their own temporary space only if they have a different one set as their default for sorting and temporary tables. Even with the default temporary tablespace set as TEMP1, for example, user1 might have TEMP2 as the default and will use only the TEMP2 tablespace for the temporary space. It is a nice way to isolate some of the areas that are normally shared among different users or different applications.

How Oracle handles temporary tables demonstrates how application coding would be different between the two platforms. Oracle temporary tables are either transaction- or session-specific tables. It doesn’t open the temporary or work tables available to other users or sessions. Some of the temporary tables in SQL Server are available for other sessions and processes until the server is restarted, and they are cleaned up at the end of the transaction or session, whether or not there were issues with the transaction or session.

Now that we’ve covered where to find the information that SQL Server stores in its master, msdb, and tempdb databases in Oracle, let’s look at the Oracle services and processes.

**Services and Processes**

Various processes and services start up with Oracle, just as there are services for the SQL Server instance and SQL Server Agent. On Windows, an Oracle service needs to be started for the database. There is also a listener in the service list for Oracle—the TNS Listener service must be running for remote sessions to connect to the Oracle database. Along with these services, background processes are running on Windows. These processes run on any database server, no matter which operating system hosts it.

When looking at the sessions in the database, you will see a list of other system processes that are started. These take care of writing, logging, jobs, gathering statistics, and monitoring.
The SMON background process performs the system monitoring functions. It takes care of the recovery of transactions after restarting the database. For example, if the database crashes, the SMON process uses the undo tablespace to detect and recover any transactions that were interrupted. If you see the SMON process using up more than the normal amount of CPU, Oracle might not have shut down nicely, and this process could be cleaning up the transactions.

The PMON background process is for the user processes. It will clean up after a failed or killed user process.

When the Oracle database is started, the SMON and PMON processes are always running. You can use this information as a quick check to see which Oracle databases are available on a server. Here is an example that shows two databases (orcl and DBA1) are running on the server:

```
> ps -ef | grep smon
oracle 4889  1  0 Dec26 ?  00:00:04 ora_smon_orcl
oracle 8168  1  0 Dec26 ?  00:00:02 ora_smon_DBA1
> ps -ef | grep pmon
oracle 4877  1  0 Dec26 ?  00:00:01 ora_pmon_orcl
oracle 8154  1  0 Dec26 ?  00:00:00 ora_pmon_DBA1
```

The number of background processes can vary depending on components and how slaves for certain processes might be available. Here is a typical list of processes you will see running in the database:

- **SMON** System monitor process
- **PMON** Process monitor process
- **ARC0** Archiver process for writing out archive logs from the redo logs
- **MMON** Memory monitor gathering memory statistics
- **MMAN** Memory manager for resizing the SGA areas
- **DBW0** Database writer process writing blocks from the buffer cache to datafiles
- **LGWR** Log writer process for flushing the redo log buffer
- **CKPT** Checkpoint process to timestamp the datafiles and control files when checkpoints occur
- **MMNL** Process to assist the MMON process
- **RECO**: Recoverer background process for distributed transactions for two-phase commits
- **CJQ0**: Job queue process for batch processing (slave processes may be spawned)
- **PSP0**: Process spawner, to spawn slave processes for Oracle
- **J000**: Job queue slave process

Other background processes depend on which components are installed. For example, the ASMB and RBAL background processes run for Automatic Storage Management (ASM), and the QMN0 process runs for Oracle Streams. For Data Guard, the DMON and MRP0 processes run. In Real Application Clusters (RAC) environments, you will see the MS0, LMON, LMD, LCK, and DIAG processes.

You can see which background processes are running by listing the processes running as `oracle` on a server, and they are also visible in the `v$sessions` view. OEM also shows the processes under session activity, as shown in Figure 2-2.

### FIGURE 2-2.   OEM view of background processes
As you can see, there are quite a few background processes running with Oracle. Depending on how many slaves are spawned and which different components are installed, more processes may be running. Let me just say that there are definitely more than ten background processes! The fact that particular processes are running on the database system can give you more information about the database, such as which components are installed or if the database is in ARCHIVELOG mode.

**sp_configure Options and Parameters**

Those who say database administration is getting easier are not looking at all of the knobs that can be turned. More options and parameters are released with each new version of Oracle. I think that you will agree that more configurable parameters have been added to SQL Server as well. But setting the parameters is actually not the tricky part. The challenge is knowing which parameters might be related or impacted when you adjust a particular parameter.

As discussed earlier, Oracle has overall parameters, such as `MEMORY_TARGET`, which manage the other underlying parameters dynamically. This approach makes it easier to change the parameters, but you still need to know which settings are appropriate—for example, which ones are for online transaction processing (OLTP) and which ones are for data warehouse systems.

I think of these parameters and options like a stereo tuner or soundboard. Preconfigured settings for different types of music can be used, and they will work for most people listening to the music. But then there are trained ears that need more of a definition of the tones or mixes of the music to make it sound exactly the way they want it. What happens if the music type changes or an instrument affects the volume? What if it is playing in the orchestra hall? How about in a small car? For these cases, more adjustments are needed. And when making such adjustments, you need to consider whether changing one setting will affect another, such as causing another part of the music to be louder or softer.

Similarly, the default database configurations may work for several database applications, but other applications need to be top performers and tuned specifically to get the desired results. This does take some understanding of the different settings and how they might affect other settings. On the
other hand, a DBA can spend too much time trying to configure and set values without getting much of a return, especially if the environment is changing rapidly. Balance is important here. You need to know which options are available, and how to validate that the dynamic settings are performing as they should, so they can be left alone (giving you time to deal with other administration tasks!).

### Viewing and Setting Parameters

In Oracle, you can view all of the parameter settings in OEM, or you can run a quick `show` query in SQL*Plus. Table 2-1 compares the SQL Server and Oracle commands for retrieving the values of parameters and options.

**NOTE**

In SQL Server, to see all of the advanced parameters, enable `show advanced option` first with `sp_configure`. Oracle has hidden parameters that begin with an underscore. These are normally not configured except internally by Oracle or when working on an issue with Oracle support.

<table>
<thead>
<tr>
<th></th>
<th><strong>SQL Server</strong></th>
<th><strong>Oracle</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>List all parameters</td>
<td><code>sp_configure</code></td>
<td><code>show parameter</code></td>
</tr>
<tr>
<td>List a parameter</td>
<td><code>sp_configure 'remote access'</code></td>
<td><code>show parameter db_block_buffers</code></td>
</tr>
<tr>
<td>List parameters with a keyword (all parameters that have the keyword in their name)</td>
<td><code>sp_configure remote</code></td>
<td><code>show parameter buffers</code></td>
</tr>
</tbody>
</table>

**TABLE 2-1. Viewing Parameters**
For SQL Server, the options can be set at the server and database level. For Oracle, the parameters are normally configured at the server level, but some can be modified for a user session, so there are system- and session-level options.

```
SQLPLUS> alter system set parameter = X scope=both;
SQLPLUS> alter session set parameter = X;
```

Oracle parameters are maintained in the init.ora (known as the pfile) or spfile.ora file. The pfile is a text file (initSID.ora) that can be edited directly. The spfile has some binary information so it cannot be edited directly. It is updated through the following alter statements:

```
alter system set parameter=x scope=spfile
alter system set parameter=x scope=both
```

The spfile allows for the dynamic parameter changes; you can run alter statements against the running database, spfile, or both.

An spfile can be created from a pfile, and a pfile from an spfile. You can change a parameter by editing the pfile, and restart the database with the pfile instead of the spfile. Then create an spfile from the edited pfile to have the spfile file updated with the parameters, if you normally start up using the spfile.

```
SQLPLUS> startup pfile='/u01/oracle/product/11.0.1/dbs/initDBA1.ora'
SQLPLUS> create spfile from pfile; /*can also use create spfile from memory */
SQLPLUS> shutdown immediate;
SQLPLUS> startup /* as long as the spfile parameter is set in the parameter it will start up using the spfile */
```

Getting Started with Some Parameters

How many knobs are available to adjust? In Oracle Database 10g, there are about 259 configurable parameters, with well over 1100 hidden parameters. In Oracle Database 11g, there are around 342 configurable parameters, and even more hidden parameters. Here, we will take a quick look at just some of these parameters.

Transaction Log Parameters

In SQL Server, transaction logs are handled with the SIMPLE or FULL option. In Oracle, ARCHIVELOG mode is similar to FULL. Archiving will write out the redo logs to a file for backing up, and allow for hot backups.
and point-in-time recovery. The default is NOARCHIVELOG mode, which is
good for creating the database, but after the database is created and started
it should be changed to ARCHIVELOG mode to be able to run the hot
backups and have the full recovery options.

Versions prior to Oracle Database 10g included a parameter to start
archiving. Now just the parameter for the location of the archive logs is
needed: LOG_ARCHIVE_DEST.

**Database Creation Parameters**
The database name (DB_NAME) and character set are some of the parameters
set up when a database is created. Parameters also set the location of
control files, alert logs, and trace files.

The MAXDATAFILES and MAXLOGFILES parameters are limits that are set
to size the control file when creating the database. MAXDATAFILES sets the
total number of datafiles you can have in the database. If you reach the limit
of MAXDATAFILES, you not only need to adjust the parameter, but also to
re-create the control files to allow for the larger limit. MAXLOGFILES sets
the total number of redo log files. The DB_FILES parameter is more of the soft
limit that can be adjusted, but it needs a restart of the database to be put into
effect.

**Some Basic Parameters**
The following are some basic parameters that are normally adjusted in some
way. These parameters deal with system size, the database version, and
resources available on the server.

- **DB_BLOCK_SIZE**  Size of the database block in bytes.
- **PROCESSES**  Number of allowable user processes. You need to
  restart the database to change this value, so plan for the number of
  users accessing the server.
- **SESSIONS**  Number of allowable sessions. You need to restart
  the database to change this value, so plan for the number of
  users accessing the server. This setting is similar to the maximum number
  of connections for SQL Server.
- **COMPATIBLE**  Database compatible with this software version.
  The current version would be ideal, but you can also allow for
  upgrades and still have Oracle behave as a different version. This
  setting is similar to the compatibility level in SQL Server.
PGA_AGGREGATE_TARGET  PGA memory, user process area.
SGA_TARGET  SGA memory.
MEMORY_TARGET  SGA memory (Oracle Database 11g).
UNDO_MANAGEMENT  Automatic undo management when TRUE.
UNDO_TABLESPACE  Tablespace for undo management.

Location and Destination Parameters
The following parameters will probably be different for every system, as they set the location of files for a database, and they tend to have a database name somewhere in a directory for separation of these locations.

CONTROL_FILES  Directory and file names of the control files.
BACKGROUND_DUMP_DEST  Directory for the alert log.
USER_DUMP_DEST  Directory for the user trace files.
AUDIT_FILE_DEST  Directory for audit logs.
LOG_ARCHIVE_DEST  Directory for archive logs.

Optimizer and Performance Parameters
Optimizer parameters set different behaviors of the optimizer. These parameters are available to assist with performance and adjust settings to deal with applications in particular ways. They help Oracle to choose a good path for execution plans.

OPTIMIZER_MODE  FIRST_ROW or ALL ROWS (also CHOICE and RULE in Oracle Database 10g). This is the setting for the default behavior of the optimizer for cost-based query plans. The default for Oracle Database 11g is ALL ROWS.
CURSOR_SHARING  FORCE, EXACT, or SIMILAR. This setting is used to help reuse SQL statements in the library cache. FORCE and SIMILAR are good for use with code that uses literal values to force the optimizer to use a similar plan if the plan can’t be matched because of the literal value.
QUERY_REWRITE_ENABLED  Allow rewrite of queries using materialized views.

SESSION_CACHED_CURSORS  Number of cursors to place in the cache for a session.

Other Parameters
Let’s round off the list with a couple more parameters that should be mentioned here. These parameters will normally use the default setting, but if you’re wondering where all of the slave job processes come from, they are probably run by the following parameters.

STATISTICS_LEVEL  ALL, BASIC, or TYPICAL. TYPICAL will collect the major statistics needed for automatic parameters like memory and gathering information for workload repository. BASIC will disable automated optimizer statistics and advisory components for memory settings. SQL Server has an auto-update statistics for a database, which gathers only the table statistics. This setting for Oracle gathers database, table, and operating system statistics.

RECYCLEBIN  ON or OFF. ON is the default. With this setting, dropped objects are collected in the recycle bin, and objects can be retrieved from the recycle bin if needed (unless it has been cleared).

SPFILE  Use of the spfile, file name, and location.

JOB_QUEUE_PROCESSES  Number of job slave processes. This setting is used by replication and user jobs through DBMS_JOBS. If it is set to 0, DBMS_JOBS is disabled.

MAX_JOB_SLAVE_PROCESSES  Limits the number of job slaves and user jobs scheduled through DBMS_SCHEDULER. You can use DBMS_JOBS and DBMS_SCHEDULER to create jobs, and these two parameters will set the maximum number of job slave processes.

DB_WRITER_PROCESSES  Number for database writer processes for background processes. This is useful for an environment with a large amount of writes. The default is CPU_COUNT/8.
REMOTE_LOGIN_PASSWORDFILE  EXCLUSIVE, SHARED, or NONE. When SHARED or EXCLUSIVE, a password file must be available; normally used for SYS, but can be for other users as well. NONE means it will be using operating system authentication. The password file is needed to be able to log in to the database remotely from SQL*Plus or another remote client as SYSDBA.

I believe that you have now seen more than enough parameters and options to have fun with. In later chapters, we will look at a couple more that affect performance and high-availability features. Our next topic is automatic undo management.

Undo, Redo, and Logs

Undo versus redo—this almost sounds like the start of a bad joke. Undo and redo were in a boat. Undo jumps out. Who is left on the boat? Redo! In all seriousness, understanding the purpose of the redo logs and undo tablespace will also help explain read consistency and why SELECT statements do not block writers and writers do not block readers in Oracle databases.

Transaction Logs Versus Redo Logs

In SQL Server, transactions and changes are written out to the transaction log, which is used by SQL Server to either commit the changes or roll back changes. There is also a save point that can be used for larger transactions, to basically commit the changes up to this point and continue with the transaction. The logs can either be overwritten if the database is in simple mode, or backed up to provide full backup and point-in-time restores. This is the basic flow of transactions through SQL Server and how it uses the transaction logs.

Oracle, with the undo and redo logs, handles transaction flow differently. However, some comparisons can be made between the Oracle redo logs and the SQL Server transaction logs. Like the SQL Server transaction logs, the redo logs record all of the transactions and changes made to the database.

When the database is in ARCHIVELOG mode, the archiver process will write off the redo logs for backing up and keeping these changes. When in
NOARCHIVELOG mode, the transactions that are committed will continue to be overwritten in the redo logs. In NOARCHIVELOG mode, the overwriting of the logs happens only once the changes have been recorded in the datafiles, and the changes can be committed or uncommitted transactions. There is enough information in the redo logs to roll back the transactions that might be rolled back, but Oracle is pulling the information from the datafiles.

The database will hang (or appear to hang) if it’s waiting for the redo log to be available after writing the changes to the datafiles, and if in ARCHIVELOG mode writing to the archive log. If there are no other logs available to use, it will wait until these writes are complete to be able to reuse the redo log. If you’re getting several waits here, you can increase either the number or size of the redo logs.

The redo logs are only one piece of the puzzle. Next, let’s look how undo fits into Oracle processing.

**Undo and Beyond**

In the parameters section, you saw the LOG_BUFFERS, UNDO_MANAGEMENT, and UNDO_TABLESPACE parameters. The background processes have log writers (LGWR) and archiver processes (ARCn). The redo logs are created with a fixed size during database creation, normally in at least pairs, and there can be several groups. You saw an example of a redo log in the v$logfile view in the discussion of data dictionary views earlier in this chapter. See how nicely that all fits together?

**Undo Sizing and Retention**

The undo area provides read consistency to the users. Readers get consistent data, not dirty block reads, and at the same time, they are not blocked from anyone updating the data. Not only does the undo area provide concurrency for users, but it also rolls back transactions for rollback statements, provides the details to recover the database from logical corruptions, and allows for analyzing the data for flashback query operations. For all of these cases, the undo tablespace must have a before image of the data.

The undo tablespace should be sized to hold the larger transactions and be able to keep them for a period of time. The UNDO_RETENTION parameter is the setting for Oracle to attempt to keep the changes in the undo segments. If there are committed transactions, and there is more space needed in the
undo tablespace, they will be overwritten, even if the time set by the UNDO_RETENTION period has not passed.

To view the statistics for the undo area, use the v$undostat view. To see undo history, use dba_hist_undostat. This information, along with knowledge of what is running against the database and the undo advisor information, will help you to size the undo tablespace and set the retention period. The package DBMS_UNDOADV and the functions available from this package provide the advisory information. For example dbms undo adv.required_retention will help with setting the retention.

Another good practice is to keep transactions small enough to be handled efficiently. Larger transactions run into issues for space, and if they fail (whether because of a transaction issue or a system outage), the rollback time can be significant. Reading through 20GB of undo segments will take time, and making the changes to the before image of the data will also take time.

Overwriting the committed change of the same block in one transaction that was being used in a longer running batch transaction can cause the transaction to fail with an “ORA-1555: snapshot too old” error. Receiving this error doesn’t necessarily mean you need to resize the undo tablespace. You may be able to handle the problem by changing the transaction size or by improving the performance of the transaction. In the newer releases, Oracle automatically manages the undo segments, and these errors are less likely to occur. (With the manual configuration of the rollback segments, you risk creating rollback segments that might be too small.)

Transaction Process Flow
Transactions are performed against the database. The log buffer, which is in memory, caches the transaction details. The blocks that are pulled into the buffer cache now have before and after images in the undo segments. The log buffer is flushed out to the redo logs by the log writer. Since the log buffer may not be as big as the transaction, the log writer is continuously writing to the redo logs, not just on commit. So, the redo logs contain committed as well as uncommitted transactions. The redo logs contain the replay SQL, which can be used for other systems, such as a standby database, which we will discuss in Chapter 10.

The redo logs are a fixed size; they are not set to autogrow as are some datafiles. There can be several groups of redo logs. Once a redo log group is
full or a switch log file occurs, the archiver process writes the redo log out to an archive file to be picked up by a backup process.

If all of the redo logs are full and have not yet been archived completely, the transaction will wait until that archive process is finished. The alert log will contain the message “checkpoint not complete.” This means Oracle was unable to overwrite the redo log the first time and waited until it could overwrite the redo log. To address this issue, you could increase the size of the redo logs, but this is not always the best solution. You might instead add another group of redo logs to give the archiver more time to write out the log to the archive log. Log switching through the redo logs is important so that you have archive logs to back up, because the redo logs are not backed up during the hot backups. You can check how many times the log is switching per hour, through the v$log_history view or the alert log. If it is too many times per hour, make the logs bigger. If not, just add more groups of logs.

Figure 2-3 shows a view how this process flows when transactions are performed against the database. The transaction is not showing as being committed or rolled back. At the point of being committed or when

---

**FIGURE 2-3.** Transaction process flow
checkpoints run, the database writers would join into the process to write the changed database blocks back to the datafiles.

Understanding how Oracle handles transactions will help you in sizing the memory, undo tablespaces, and redo logs. Being able to have consistent reads in the database and provide a way to access the data without being blocked or needing to wait for transactions to complete is also key in the performance of queries against the database.

Summary

SQL Server has system databases, such as master, msdb, model, and tempdb. Even though Oracle does not have individual system databases that match the ones in SQL Server, the platforms share some similar concepts. There is a need for system information, there are parameters and options that can be configured and viewed, and transaction logging keeps track of changes.

Oracle has memory structures for supplying memory to server processes and user processes. There are individual parameters to manually configure memory, or dynamic settings that are available in Oracle Database 11g by setting one parameter. Data dictionary views show the system information, including the values of the parameters. Oracle offers quite a few parameters for tuning and adjusting to provide the best performance options. We went over only a small portion of them in this chapter, but you have a starting point for common requirements.

Temporary and undo tablespaces are distinctive features of Oracle. It is able to have more than one temporary area that can be assigned to different users to isolate their sorting and temporary table processing. The undo tablespace keeps track of the before and after copies to provide consistent reads for concurrent users and be able to roll back changes if needed.

Changes are written to redo logs and kept in the undo segments to handle transactions. There is also a memory cache for the logs to buffer the log for the log writer to be able to process the changes to the redo logs and then off to the archive logs.

The server configurations and background processes offer just a glimpse into the internal workings of Oracle. There are several other system views available to see how Oracle is performing and gathering statistics to be able to process the requests and changes in the database. Some of them will be discussed in the following chapters as needed for more details, and the complete list is provided in the Oracle documentation.