13

Managing Memory

Objectives

After completing this lesson, you should be able to:

- Describe the memory components in the SGA
- Implement Automatic Memory Management
- Manually configure SGA parameters
- Configure automatic PGA memory management

Memory Management: Overview

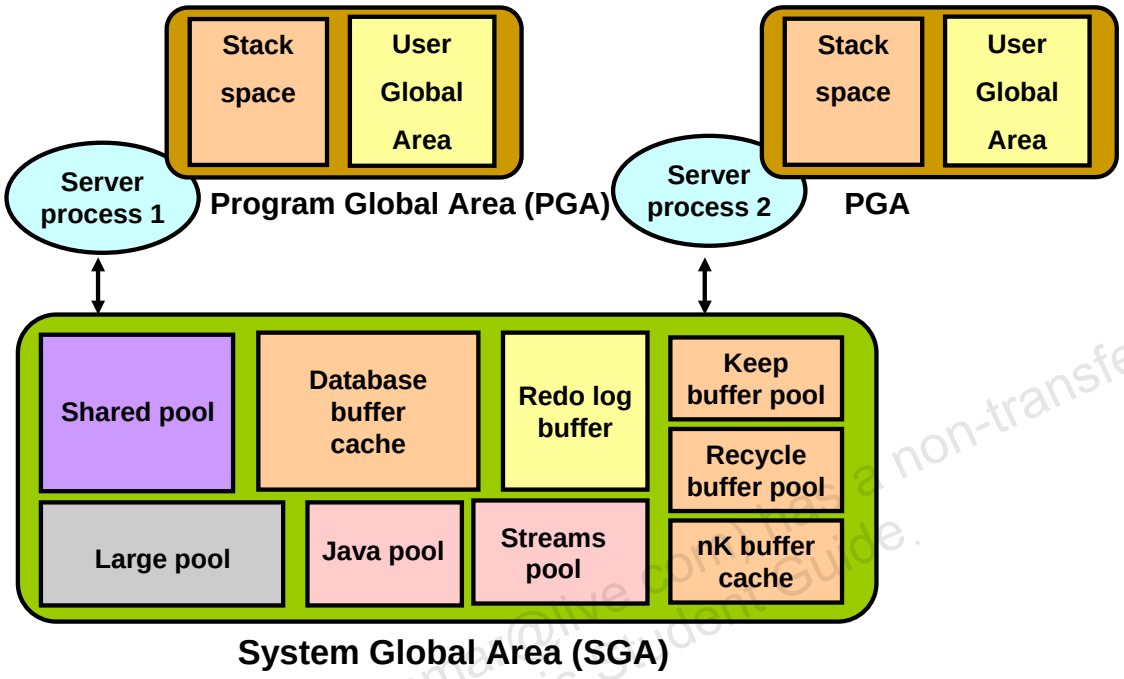
DBAs must consider memory management to be a crucial part of their job because:

- There is a finite amount of memory available
- Allocating more memory to serve certain types of functions can improve overall performance
- Automatically tuned memory allocation is often the appropriate configuration, but specific environments or even short-term conditions may require further attention

Memory Management: Overview

Because there is a finite amount of memory available on a database server and thus, on an Oracle database instance, you must pay attention to how memory is allocated. If too much memory is allowed to be used by a particular area that does not need it, then there is the possibility that there are other functional areas unnecessarily doing without enough memory to perform optimally. With the ability to have memory allocation automatically determined and maintained for you, the task is simplified greatly. But even automatically tuned memory needs to be monitored for optimization and may need to be manually configured to some extent.

Reviewing Oracle Database Memory Structures



Reviewing Oracle Database Memory Structures

Oracle Database creates and uses memory structures for various purposes. For example, memory stores program code being run, data that is shared among users, and private data areas for each connected user.

Two basic memory structures are associated with an instance:

- **System Global Area (SGA):** Group of shared memory structures, known as SGA components, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.
- **Program Global Areas (PGA):** Memory regions that contain data and control information for a server or background process. A PGA is nonshared memory created by Oracle Database when a server or background process is started. Access to the PGA is exclusive to the server process. Each server process and background process has its own PGA.

Reviewing Oracle Database Memory Structures (continued)

The SGA is the memory area that contains data and control information for the instance. The SGA includes the following data structures:

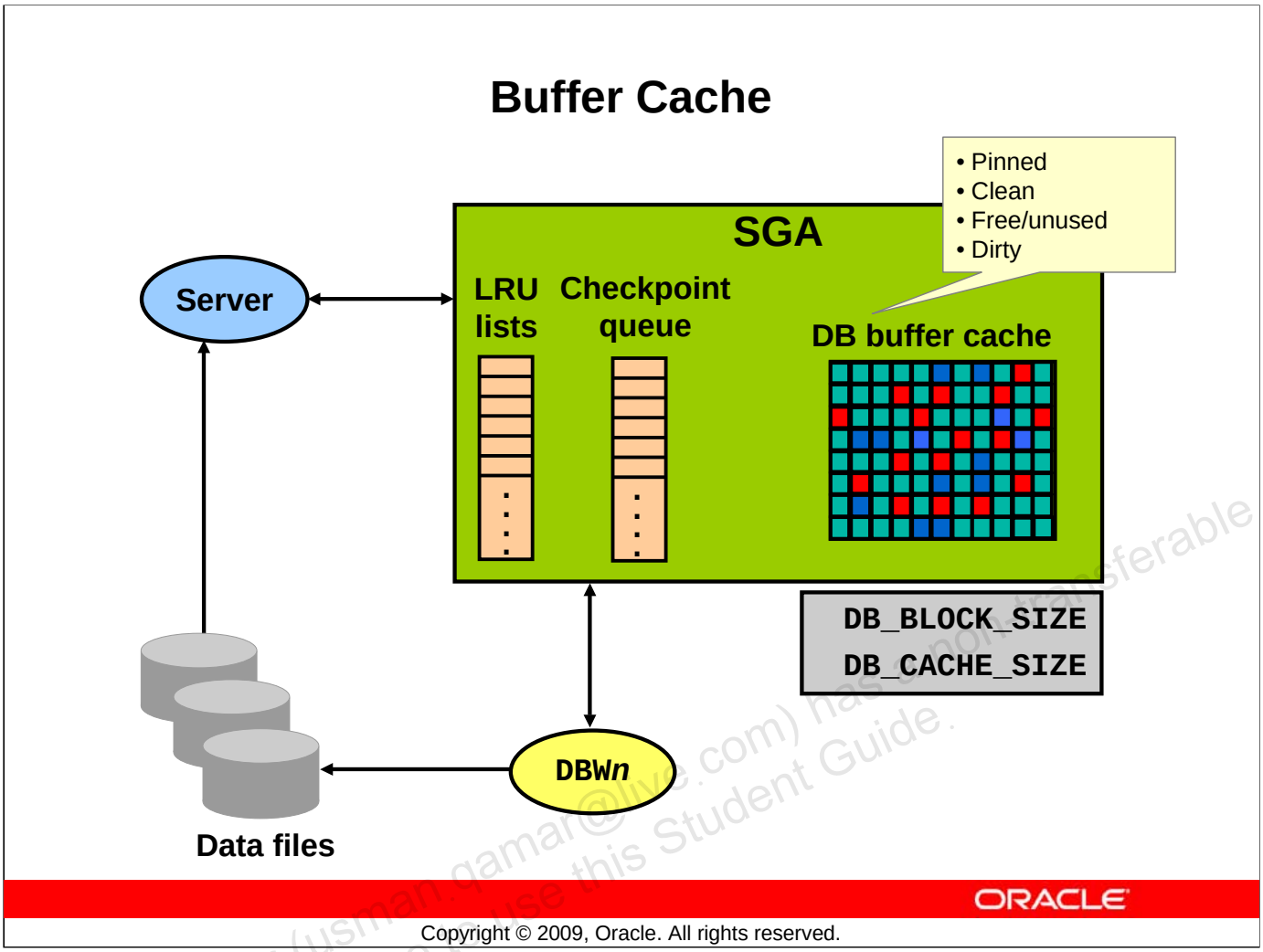
- **Shared pool:** Caches various constructs that can be shared among users
- **Database buffer cache:** Caches blocks of data retrieved from the database
- **KEEP buffer pool:** Is a specialized type of database buffer cache that is tuned to retain blocks of data in memory for long periods of time
- **Recycle buffer pool:** Is a specialized type of database buffer cache that is tuned to recycle or remove block from memory quickly
- **nK buffer cache:** Is one of several specialized database buffer caches designed to hold block sizes different from the default database block size
- **Redo log buffer:** Caches redo information (used for instance recovery) until it can be written to the physical redo log files stored on the disk
- **Large pool:** Is the optional area that provides large memory allocations for certain large processes, such as Oracle backup and recovery operations, and I/O server processes
- **Java pool:** Is used for all session-specific Java code and data in the Java Virtual Machine (JVM)
- **Streams pool:** Is used by Oracle Streams to store information required by capture and apply

When you start the instance by using Enterprise Manager or SQL*Plus, the amount of memory allocated for the SGA is displayed.

A Program Global Area (PGA) is a memory region that contains data and control information for each server process. An Oracle server process services a client's requests. Each server process has its own private PGA that is created when the server process is started. Access to the PGA is exclusive to that server process, and the PGA is read and written only by the Oracle code acting on its behalf. The PGA is divided into two major areas: stack space and the User Global Area (UGA).

With the dynamic SGA infrastructure, the sizes of the database buffer cache, the shared pool, the large pool, the Java pool, and the Streams pool can change without shutting down the instance.

The Oracle database uses initialization parameters to create and manage memory structures. The simplest way to manage memory is to allow the database to automatically manage and tune it for you. To do so (on most platforms), you only have to set a target memory size initialization parameter (`MEMORY_TARGET`) and a maximum memory size initialization parameter (`MEMORY_MAX_TARGET`).



Buffer Cache

You can configure the buffer cache by specifying a value for the `DB_CACHE_SIZE` parameter. The buffer cache holds copies of the data blocks from the data files having a block size of `DB_BLOCK_SIZE`. The buffer cache is a part of the SGA, so all users can share these blocks. The server processes read data from the data files into the buffer cache. To improve performance, the server process sometimes reads multiple blocks in a single read operation. The `DBWn` process writes data from the buffer cache into the data files. To improve performance, `DBWn` writes multiple blocks in a single write operation.

At any given time, the buffer cache may hold multiple copies of a single database block. Only one current copy of the block exists, but to satisfy queries, server processes may need to construct read-consistent copies from past image information. This is called a consistent read (CR) block.

The least recently used (LRU) list reflects the usage of buffers. The buffers are sorted on the basis of a combination of how recently and how often they have been referenced. Thus, buffers that are most frequently and recently used are found at the most recently used end. Incoming blocks are copied to a buffer from the least recently used end, which is then assigned to the middle of the list, as a starting point. From here, the buffer works its way up or down the list, depending on usage.

Buffer Cache (continued)

Buffers in the buffer cache can be in one of four states:

- **Pinned:** The block is either currently being read into the cache or being written to. Other sessions wait to access the block.
- **Clean:** The buffer is now unpinned and is a candidate for immediate aging out if the current contents (data block) are not referenced again. Either the contents are in sync with disk or the buffer contains a CR snapshot of a block.
- **Free/unused:** The buffer is empty because the instance just started. This state is very similar to the clean state, except that the buffer has not been used.
- **Dirty:** The buffer is no longer pinned but the contents (data block) have changed and must be flushed to disk by DBW*n* before it can be aged out.

Server processes use the buffers in the buffer cache, but the DBW*n* process makes buffers in the cache available by writing changed buffers back to the data files. The checkpoint queue lists the buffers that are to be written out to disk.

Then Oracle database supports multiple block sizes in the same database. The standard block size is used for the SYSTEM tablespace. You specify the standard block size by setting the initialization parameter DB_BLOCK_SIZE. Legitimate values are from 2 KB to 32 KB, and the default is 8 KB. The cache sizes of nonstandard block size buffers are specified by the following parameters:

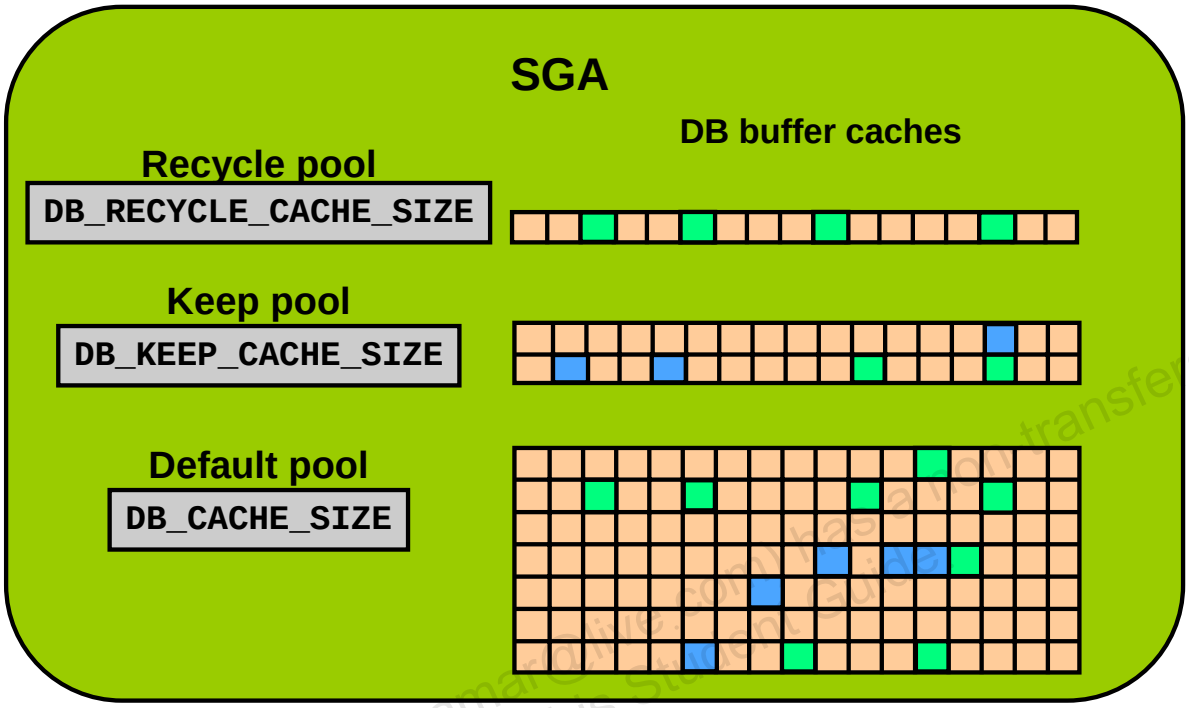
- DB_2K_CACHE_SIZE
- DB_4K_CACHE_SIZE
- DB_8K_CACHE_SIZE
- DB_16K_CACHE_SIZE
- DB_32K_CACHE_SIZE

The DB_*n*K_CACHE_SIZE parameters cannot be used to size the cache for the standard block size. If the value of DB_BLOCK_SIZE is *n*K, it is illegal to set DB_*n*K_CACHE_SIZE. The size of the cache for the standard block size is always determined from the value of DB_CACHE_SIZE.

Each buffer cache has a limited size, so typically not all the data on disk can fit in the cache. When the cache is full, subsequent cache misses cause the Oracle database to write dirty data already in the cache to disk to make room for the new data. (If a buffer is not dirty, it does not need to be written to disk before a new block can be read into the buffer.) Subsequent access to any data that was written to disk results in additional cache misses.

The size of the cache affects the likelihood that a request for data will result in a cache hit. If the cache is large, it is more likely to contain the data that is requested. Increasing the size of a cache increases the percentage of data requests that result in cache hits.

Using Multiple Buffer Pools



Using Multiple Buffer Pools

The database administrator (DBA) may be able to improve the performance of the database buffer cache by creating multiple buffer pools. You assign objects to a buffer pool depending on how the objects are accessed. There are three buffer pools:

- **Keep:** This pool is used to retain objects in memory that are likely to be reused. Keeping these objects in memory reduces I/O operations. Buffers are kept in this pool by ensuring that the pool is sized larger than the total size of the segments assigned to the pool. This means that buffers do not have to be aged out. The keep pool is configured by specifying a value for the `DB_KEEP_CACHE_SIZE` parameter.
- **Recycle:** This pool is used for blocks in memory that have little chance of being reused. The recycle pool is sized smaller than the total size of the segments assigned to the pool. This means that blocks read into the pool will often have to age out a buffer. The recycle pool is configured by specifying a value for the `DB_RECYCLE_CACHE_SIZE` parameter.
- **Default:** This pool always exists. It is equivalent to the buffer cache of an instance without a keep pool or a recycle pool and is configured with the `DB_CACHE_SIZE` parameter.

Note: The memory in the keep or recycle pool is not a subset of the default buffer pool.

Using Multiple Buffer Pools

```
CREATE INDEX cust_idx …
  STORAGE (BUFFER_POOL KEEP);

ALTER TABLE oe.customers
  STORAGE (BUFFER_POOL RECYCLE);

ALTER INDEX oe.cust_lname_ix
  STORAGE (BUFFER_POOL KEEP);
```


Using Multiple Buffer Pools (continued)

The BUFFER_POOL clause is used to define the default buffer pool for an object. It is part of the STORAGE clause and is valid for CREATE and ALTER table, cluster, and index statements. The blocks from an object without an explicitly set buffer pool go into the default buffer pool.

The syntax is BUFFER_POOL [KEEP | RECYCLE | DEFAULT].

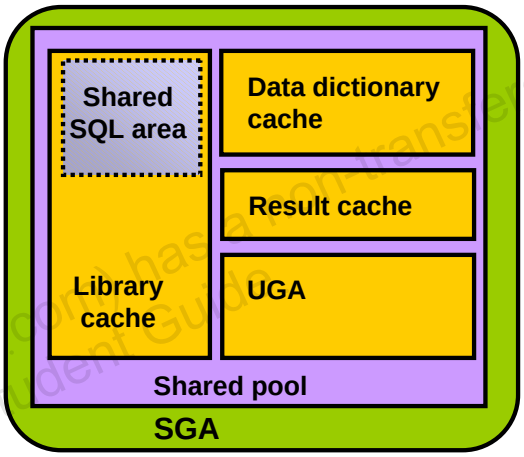
When the default buffer pool of an object is changed using the ALTER statement, blocks that are already cached remain in their current buffers until they are flushed out by the normal cache management activity. Blocks read from disk are placed into the newly specified buffer pool for the segment.

Because buffer pools are assigned to a segment, objects with multiple segments can have blocks in multiple buffer pools. For example, an index-organized table can have different pools defined on both the index and the overflow segment.

Shared Pool

Contents:

- Library cache: Command text, parsed code, and execution plan
- Data dictionary cache: Definitions for tables, columns, and privileges from the data dictionary tables
- Result cache: Results from SQL queries and PL/SQL functions
- User Global Area (UGA): Session information for the Oracle shared server



`SHARED_POOL_SIZE`

Shared Pool

You can specify the size of the shared pool with the `SHARED_POOL_SIZE` initialization parameter. The shared pool is a memory area that stores information shared by multiple sessions. It contains different types of data, as shown in the graphic in the slide.

Library cache: The library cache contains shared SQL and PL/SQL areas—the fully parsed or compiled representations of PL/SQL blocks and SQL statements. PL/SQL blocks include:

- Procedures and functions
- Packages
- Triggers
- Anonymous PL/SQL blocks

Data dictionary cache: The data dictionary cache holds definitions of dictionary objects in memory.

Result cache: The result cache comprises the SQL query result cache and PL/SQL function result cache. This cache is used to store results of SQL queries or PL/SQL functions to speed up their future execution.

User Global Area: The UGA contains the session information for the Oracle shared server. The UGA is located in the shared pool when using a shared server session and if the large pool is not configured.

Large Pool

- Provides large memory allocations for:
  - Session memory for the shared server and the Oracle XA interface
  - I/O server processes
  - Oracle Database backup and restore operations
  - Parallel query operations
  - Advanced Queuing memory table storage
- Reduces potential fragmentation of shared pool
- Is managed by AMM and ASMM
- Is sized with the LARGE_POOL_SIZE parameter

Large Pool

The database administrator can configure an optional memory area called the *large pool* to provide large memory allocations for:

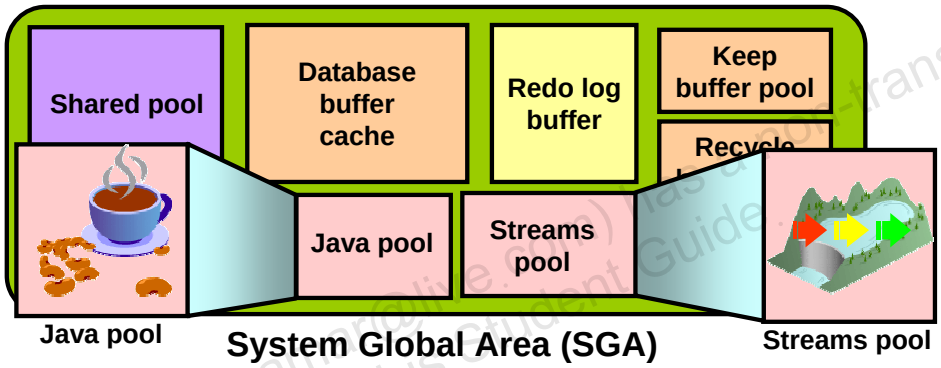
- Session memory for the shared server and the Oracle XA interface (used where transactions interact with multiple databases)
- I/O server processes
- Buffers for Recovery Manager (RMAN) I/O slaves
- Message buffers used in the parallel execution of statements
- Advanced Queuing memory table storage

By allocating session memory for the items listed in the slide, the shared pool has less fragmentation that would come from having large objects frequently allocated and deallocated in it. Segregating large objects out of the shared pool results in more efficient shared pool usage, which means more of its memory is available to service new requests and to retain existing data if needed.

The large pool can be automatically managed by AMM and ASMM. You can also size it with the `LARGE_POOL_SIZE` parameter.

Java Pool and Streams Pool

- Java pool memory is used in server memory for all session-specific Java code and data in the JVM.
- Streams pool memory is used exclusively by Oracle Streams to:
  - Store buffered queue messages
  - Provide memory for Oracle Streams processes



Java Pool and Streams Pool

Java pool memory is used in server memory for all session-specific Java code and data in the JVM. Java pool memory is used in different ways, depending on the mode in which Oracle Database is running.

The Java Pool Advisor statistics provide information about library cache memory used for Java and predict how changes in the size of the Java pool can affect the parse rate. The Java Pool Advisor is internally turned on when `statistics_level` is set to `TYPICAL` or higher. These statistics reset when the advisor is turned off.

The Streams pool is used exclusively by Oracle Streams. The Streams pool stores buffered queue messages, and it provides memory for Oracle Streams capture processes and apply processes.

Unless you specifically configure it, the size of the Streams pool starts at zero. The pool size grows dynamically as needed when Oracle Streams is used.

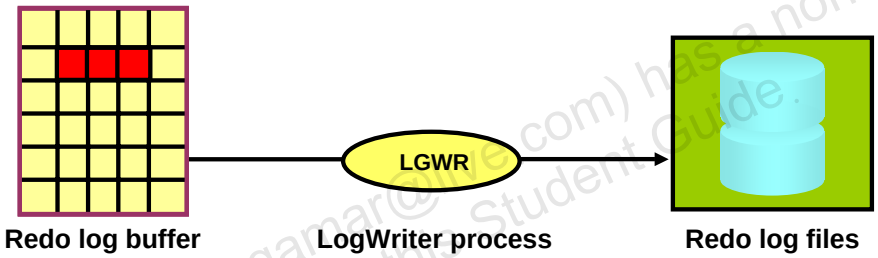
Note: A detailed discussion of Java programming and Oracle Streams is beyond the scope of this class.

Redo Log Buffer

- Is a circular buffer in the SGA
- Holds information about changes made to the database
- Contains redo entries that have the information to redo changes made by operations such as DML and DDL

Content transferred by log writer process (LGWR):

- When a user process commits a transaction
- When the redo log buffer is one-third full
- Before a DBW*n* process writes modified buffers to disk



Redo Log Buffer

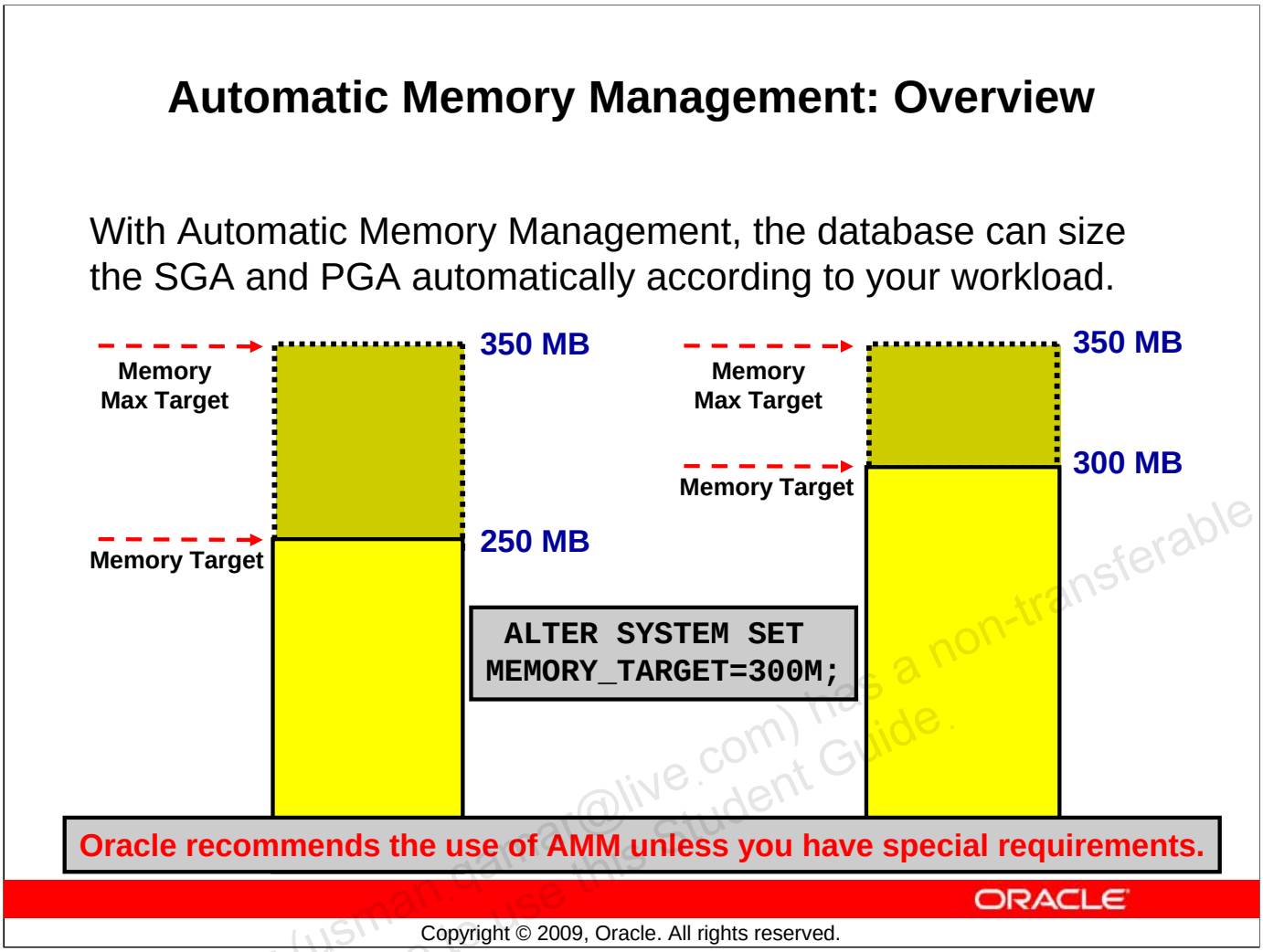
The Oracle server processes copy redo entries from the user's memory space to the redo log buffer for each DML or DDL statement. The redo entries contain the information necessary to reconstruct or redo changes made to the database by DML and DDL operations. They are used for database recovery and take up continuous sequential space in the buffer.

The redo log buffer is a circular buffer; the server processes can copy new entries over the entries in the redo log buffer that have already been written to disk. The LGWR process normally writes fast enough to ensure that space is always available in the buffer for new entries. The LGWR process writes the redo log buffer to the active online redo log file (or members of the active group) on disk. The LGWR process copies to disk all redo entries that have been entered into the buffer since the last time LGWR wrote to disk.

What Causes LGWR to Write?

LGWR writes out the redo data from the redo log buffer:

- When a user process commits a transaction
- Every three seconds, or when the redo log buffer is one-third full
- When a DBW*n* process writes modified buffers to disk, if the corresponding redo log data has not already been written to disk



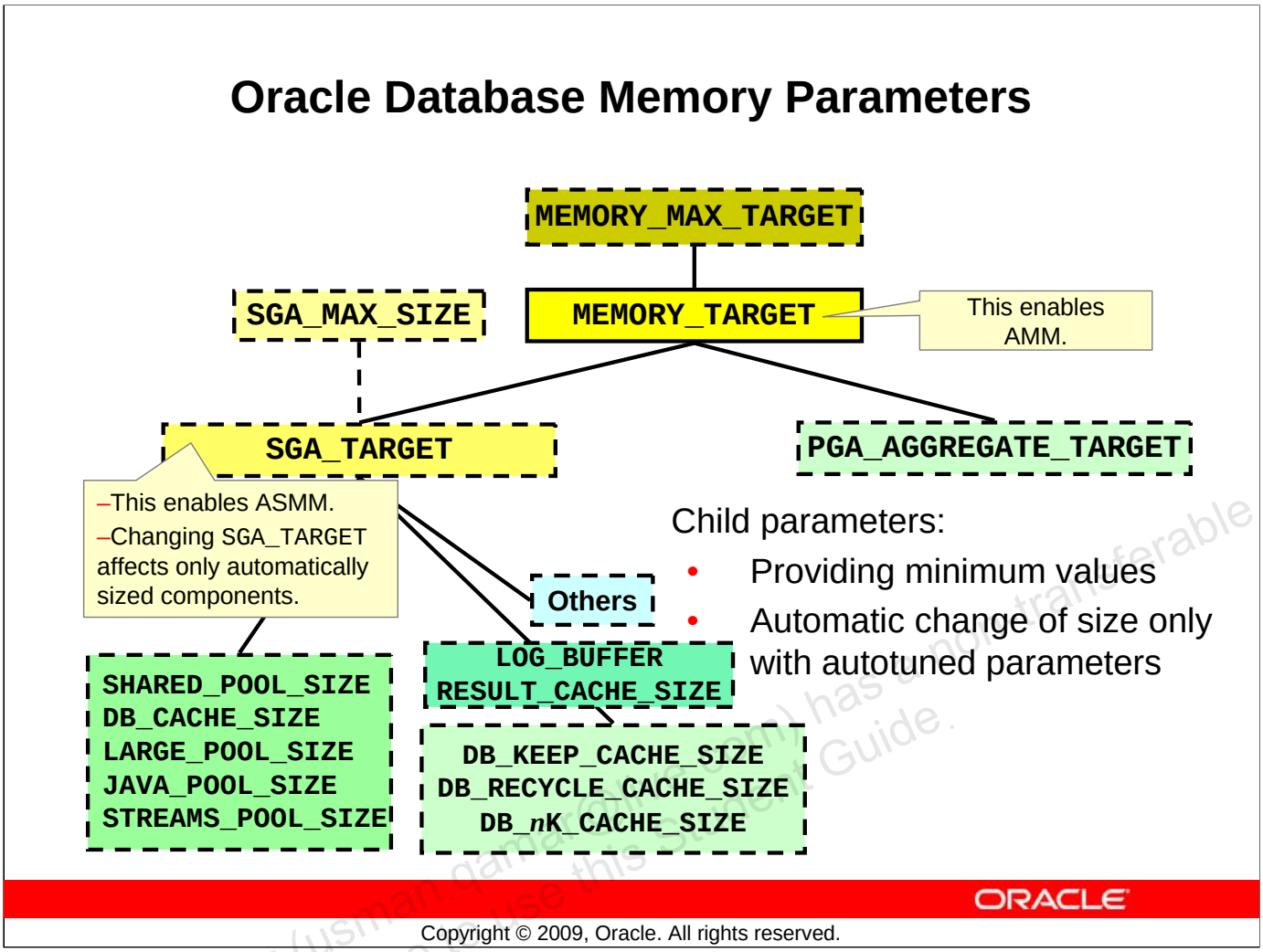
Automatic Memory Management: Overview

Automatic Memory Management (AMM) allows the Oracle Database to manage SGA memory and instance PGA memory sizing automatically. To do so (on most platforms), you set only a target memory size initialization parameter (`MEMORY_TARGET`) and a maximum memory size initialization parameter (`MEMORY_MAX_TARGET`), and the database dynamically exchanges memory between the SGA and the instance PGA as needed to meet processing demands. You can enable AMM in Enterprise Manager by navigating to: Server > Memory Advisors (in the Database Configuration section) and then clicking the Enable button.

With this memory management method, the database also dynamically tunes the sizes of the individual SGA components and the sizes of the individual PGAs.

Because the target memory initialization parameter is dynamic, you can change the target memory size at any time without restarting the database. The maximum memory size serves as an upper limit so that you cannot accidentally set the target memory size too high. Because certain SGA components either cannot easily shrink or must remain at a minimum size, the database also prevents you from setting the target memory size too low.

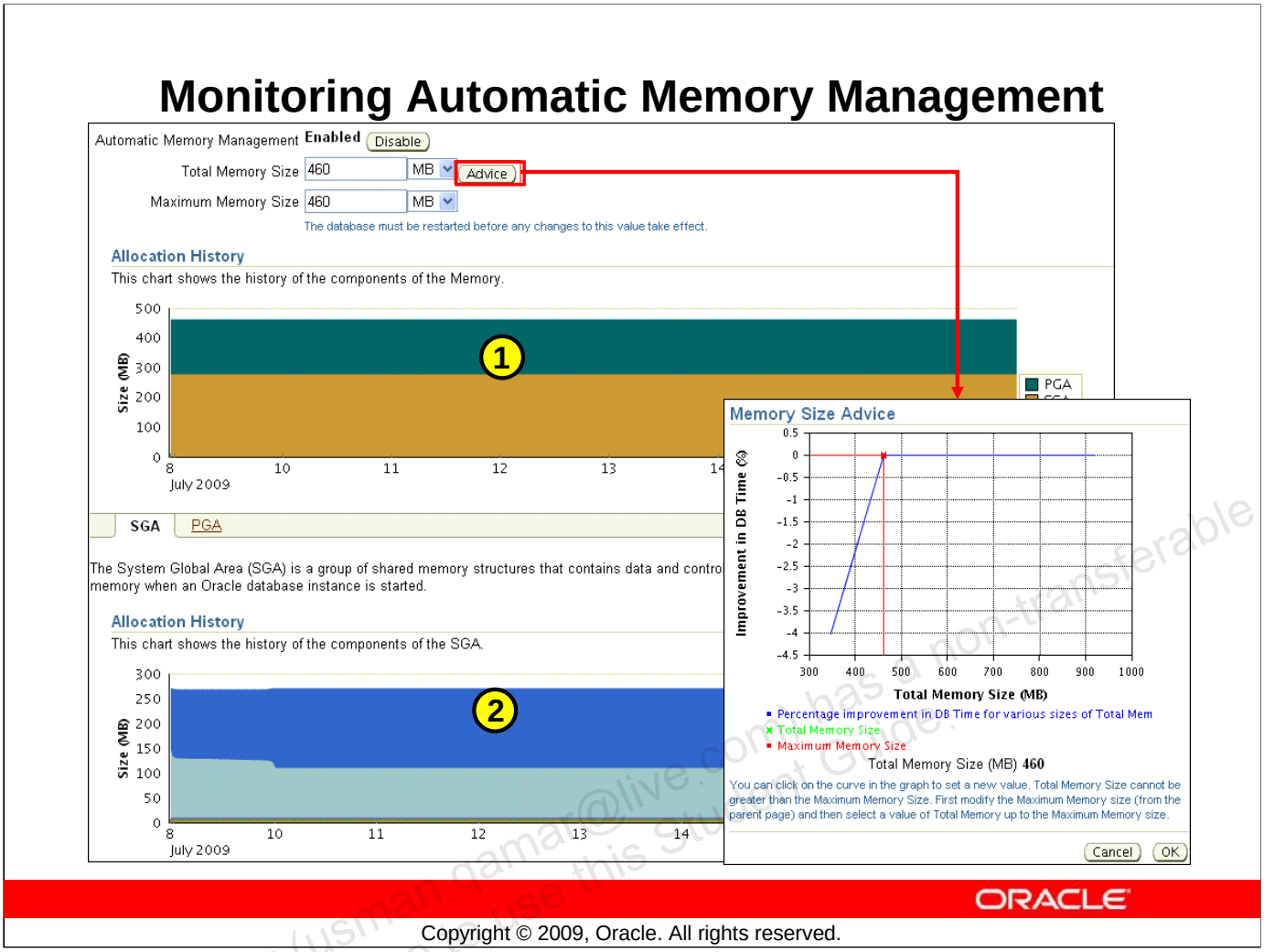
This indirect memory transfer relies on the operating system (OS) mechanism of freeing shared memory. After memory is released to the OS, the other components can allocate memory by requesting memory from the OS. Currently, Automatic Memory Management is implemented on Linux, Solaris, HPUX, AIX, and Windows.



Oracle Database Memory Sizing Parameters

The graphic in the slide shows you the memory initialization parameters hierarchy. Although you have to set only MEMORY_TARGET to trigger Automatic Memory Management, you still have the possibility to set lower bound values for various caches. Therefore, if the child parameters are user set, they will be the minimum values below which the Oracle database server will not autotune that component.

- If SGA_TARGET and PGA_AGGREGATE_TARGET are set to a nonzero value, they are considered to be the minimum values for the sizes of the SGA and the PGA, respectively. MEMORY_TARGET can take values from SGA_TARGET + PGA_AGGREGATE_TARGET to MEMORY_MAX_SIZE.
- If SGA_TARGET is set, the database autotunes only the sizes of the subcomponents of the SGA. PGA is autotuned independent of whether it is explicitly set or not. However, the whole SGA (SGA_TARGET) and the PGA (PGA_AGGREGATE_TARGET) are not autotuned—that is, do not grow or shrink automatically.



Monitoring Automatic Memory Management

From the EM home page (Related Links section), navigate to Advisor Central > Memory Advisors. The Memory Advisors page is displayed in the slide.

After Automatic Memory Management is enabled, you can see the graphical representation of the history of your memory size components in the Allocation History section of the Memory Advisors page. The top portion in the first histogram is tunable PGA only and the lower portion is all of SGA. The top portion in the second histogram is the shared pool size and the lower portion corresponds to the buffer cache.

On this page, you can also access the memory target advisor by clicking the Advice button. This advisor gives you the possible DB time improvement for various total memory sizes.

Note: You can also look at the memory target advisor by using the `V$MEMORY_TARGET_ADVISOR` view.

Monitoring Automatic Memory Management

If you want to monitor the decisions made by Automatic Memory Management via a command line:

- V$MEMORY_DYNAMIC_COMPONENTS has the current status of all memory components
- V$MEMORY_RESIZE_OPS has a circular history buffer of the last 800 memory resize requests
- V$MEMORY_TARGET_ADVICE provides tuning advice for the MEMORY_TARGET initialization parameter

Monitoring Automatic Memory Management (continued)

The dynamic performance view V$MEMORY_DYNAMIC_COMPONENTS shows the current sizes of all dynamically tuned memory components, including the total sizes of the SGA and instance PGA. The V$MEMORY_TARGET_ADVICE view provides tuning advice for the MEMORY_TARGET initialization parameter.

When viewing the V$MEMORY_TARGET_ADVICE view, the row with the MEMORY_SIZE_FACTOR of 1 shows the current size of memory, as set by the MEMORY_TARGET initialization parameter, and the amount of DB time required to complete the current workload. In previous and subsequent rows, the results show a number of alternative MEMORY_TARGET sizes. For each alternative size, the database shows the size factor (the multiple of the current size), and the estimated DB time to complete the current workload if the MEMORY_TARGET parameter were changed to the alternative size. Notice that for a total memory size smaller than the current MEMORY_TARGET size, estimated DB time increases.

Efficient Memory Usage: Guidelines

- Fit the SGA into physical memory.
- Tune for a high buffer cache hit ratio, with the following caveats:
  - Even valid and necessary full table scans lower it.
  - It is possible that unnecessary repeated reads of the same blocks are artificially raising it.
- Use the Memory Advisors.

ORACLE

Efficient Memory Usage: Guidelines

If possible, it is best to fit the SGA into physical memory, which provides the fastest access. Even though the OS may provide additional virtual memory, that memory, by its nature, can often be swapped out to disk. On some platforms, you can use the `LOCK_SGA` initialization parameter to lock the SGA into physical memory. This parameter cannot be used in conjunction with AMM or ASMM.

When a SQL statement executes, data blocks are requested for reading or writing, or both. This is considered a logical I/O. As the block is requested, the block is checked to see whether it already exists in memory. If it is not in memory, it is read from the disk, which is called a physical I/O. The number of times the block is found already in memory compared to the total number of logical I/Os is referred to as the buffer cache hit ratio. A higher ratio is usually better because that means more blocks are being found in memory without incurring disk I/O.

It is not uncommon to have a buffer cache hit ratio above 99% but that does not always mean the system is well tuned. If there is a query that is executed more often than necessary, and it constantly requests the same blocks over and over again, the ratio is raised. If it is an inefficient or unnecessary query, then it artificially inflates the ratio. This is because it should not execute in that manner or that often in the first place.

Efficient Memory Usage: Guidelines (continued)

Also, consider the fact that large full table scans (a full reading of the entire table) can lower this ratio because the entire table may be read from the disk; the scan may not take advantage of the fact that some of the blocks may be in the buffer cache. So, if there are some necessary large full table scans in your application, your well-tuned database may always have a low database buffer cache hit ratio.

Use Enterprise Manager Memory Advisors. They can help you size the SGA on the basis of the activity in your particular database.

Memory Tuning Guidelines for the Library Cache

- Establish formatting conventions for developers so that SQL statements match in the cache.
- Use bind variables.
- Eliminate unnecessary duplicate SQL.
- Consider using `CURSOR_SHARING`.
- Use PL/SQL when possible.
- Cache sequence numbers.
- Pin objects in the library cache.

Memory Tuning Guidelines for the Library Cache

The library cache, which is a part of the shared pool, is where the Oracle database stores all the SQL, Java code, PL/SQL procedures and packages, and control structures such as locks and library cache handles. The code goes into this central location so that it can be shared among all users. The benefit of sharing is that all users can take advantage of any work that is already done on behalf of SQL. Therefore, tasks such as parsing the statement and determining the data access path (also known as the “explain plan”) are done only once per statement, no matter how many times the statement is executed, and no matter how many users execute it. A library cache that is too small does not have room for all the statements being executed and, therefore, you cannot take advantage of this sharing of work for some statements. A library cache that is too large causes a burden on the system just to manage its contents.

A library cache may end up being filled with what appear to be different statements when, in fact, they are copies of the same statement. A common cause of this is having slightly different formatting for each statement. There is no match if the string does not compare exactly. Another cause is the use of literals instead of bind variables. If the only difference between two statements is literal values, then, in most cases, each of those statement executions and the overall system would benefit from replacing those literals with bind variables.

Memory Tuning Guidelines for the Library Cache (continued)

The CURSOR_SHARING initialization parameter can be set to have the system automatically replace literals with bind variables when statements otherwise match. You should typically take advantage of this setting as a temporary measure until the application is corrected to use bind variables where appropriate. As with all of these guidelines, the use of this variable can have other side effects, which you should investigate.

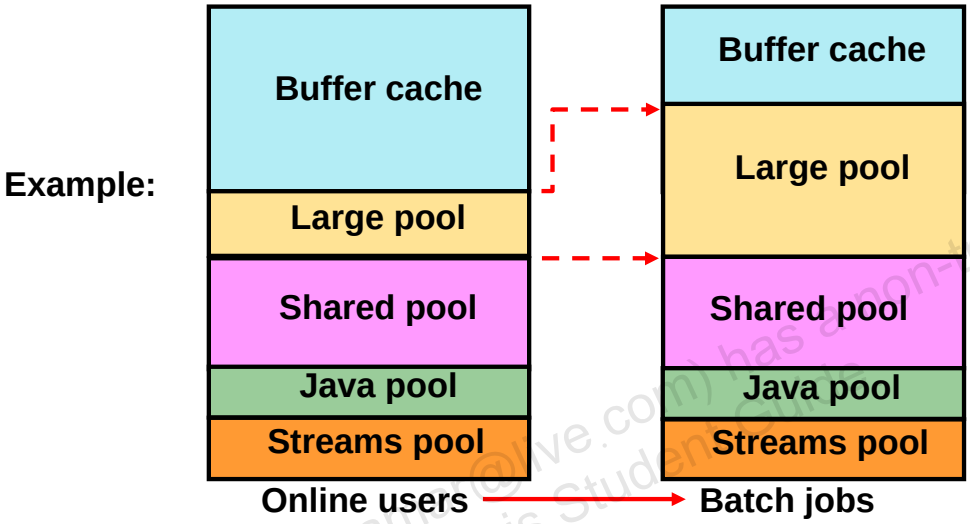
Rather than having the same SQL statement issued from several different places in an application, put the statement or statements into a stored procedure by using PL/SQL. Then just call the procedure. This guarantees that the SQL statement is shared because it exists only in one location. Also, the SQL is already parsed and has an explain plan because it is in an already compiled stored procedure.

Sequence numbers can be cached. Therefore, if there are some sequences with high activity, determine a good setting for the cache size and take advantage of it.

You can use the DBMS_SHARED_POOL package to pin objects in the library cache. This reduces the chance of reloading and recompiling objects. Refer to the *PL/SQL Packages and Types Reference* document for more information about how to use that package.

Automatic Shared Memory Management: Overview

- Automatically adapts to workload changes
- Maximizes memory utilization
- Helps eliminate out-of-memory errors



ORACLE

Automatic Shared Memory Management: Overview

If AMM does not work for you, because you need a fixed PGA, then consider the use of Automatic Shared Memory Management (ASMM) which simplifies SGA memory management. You specify the total amount of SGA memory available to an instance using the SGA_TARGET initialization parameter and Oracle Database automatically distributes this memory among the various SGA components to ensure the most effective memory utilization.

For example, in a system that runs large online transactional processing (OLTP) jobs during the day (requiring a large buffer cache) and runs parallel batch jobs at night (requiring a large value for the large pool), you would have to simultaneously configure both the buffer cache and the large pool to accommodate your peak requirements.

With ASMM, when the OLTP job runs, the buffer cache grabs most of the memory to allow for good I/O performance. When the data analysis and reporting batch job starts up later, the memory is automatically migrated to the large pool so that it can be used by parallel query operations without producing memory overflow errors.

The Oracle Database remembers the sizes of the automatically tuned components across instance shutdowns if you are using a server parameter file (SPFILE). As a result, the system does need to learn the characteristics of the workload again each time an instance is started. It can begin with information from the past instance and continue evaluating workload where it left off at the last shutdown.

How ASMM Works

- ASMM is based on workload information that MMON captures in the background.
- MMON uses memory advisors.
- Memory is moved to where it is needed the most by MMAN.
- If an SPFILE is used (which is recommended):
  - Component sizes are saved across shutdowns
  - Saved values are used to bootstrap component sizes
  - There is no need to relearn optimal values

How ASMM Works

The Automatic Shared Memory Management feature uses the SGA memory broker that is implemented by two background processes: Manageability Monitor (MMON) and Memory Manager (MMAN). Statistics and memory advisory data are periodically captured in memory by MMON. MMAN coordinates the sizing of the memory components according to MMON decisions. The SGA memory broker keeps track of the sizes of the components and pending resize operations.

The SGA memory broker observes the system and workload in order to determine the ideal distribution of memory. It performs this check every few minutes so that memory can always be present where needed. In the absence of Automatic Shared Memory Management, components had to be sized to anticipate their individual worst-case memory requirements.

On the basis of workload information, Automatic Shared Memory Management:

- Captures statistics periodically in the background
- Uses memory advisors
- Performs what-if analysis to determine the best distribution of the memory
- Moves memory to where it is most needed
- Saves component sizes across shutdown if an SPFILE is used (the sizes can be resurrected from before the last shutdown)

Enabling Automatic Shared Memory Management

To enable ASMM from manual shared memory management:

1. Get a value for **SGA_TARGET:**

```
SELECT ((SELECT SUM(value) FROM V$SGA) - (SELECT CURRENT_SIZE
FROM V$SGA_DYNAMIC_FREE_MEMORY)) "SGA_TARGET" FROM DUAL;
```

2. Use that value to set **SGA_TARGET**.
3. Set the values of the automatically sized SGA components to 0.

To switch to ASMM from Automatic Memory Management:

1. Set the **MEMORY_TARGET** initialization parameter to 0.
2. Set the values of the automatically sized SGA components to 0.

Enabling Automatic Shared Memory Management

The procedure for enabling ASMM differs depending on whether you are changing to ASMM from manual shared memory management or from automatic memory management. To change to ASMM from manual shared memory management:

1. Run the following query to obtain a value for SGA_TARGET:
   ```
   SELECT ((SELECT SUM(value) FROM V$SGA) - (SELECT CURRENT_SIZE FROM
     V$SGA_DYNAMIC_FREE_MEMORY)) “SGA_TARGET” FROM DUAL;
   ```
2. Set the value of SGA_TARGET:
   `ALTER SYSTEM SET SGA_TARGET=`*`value`* `[SCOPE={SPFILE|MEMORY|BOTH}]`
   where *value* is the value computed in step 1 or is some value between the sum of all SGA component sizes and SGA_MAX_SIZE.
3. Set the values of the automatically sized SGA components to 0. Do this by editing the text initialization parameter file or by issuing ALTER SYSTEM statements. Restart the instance if required.

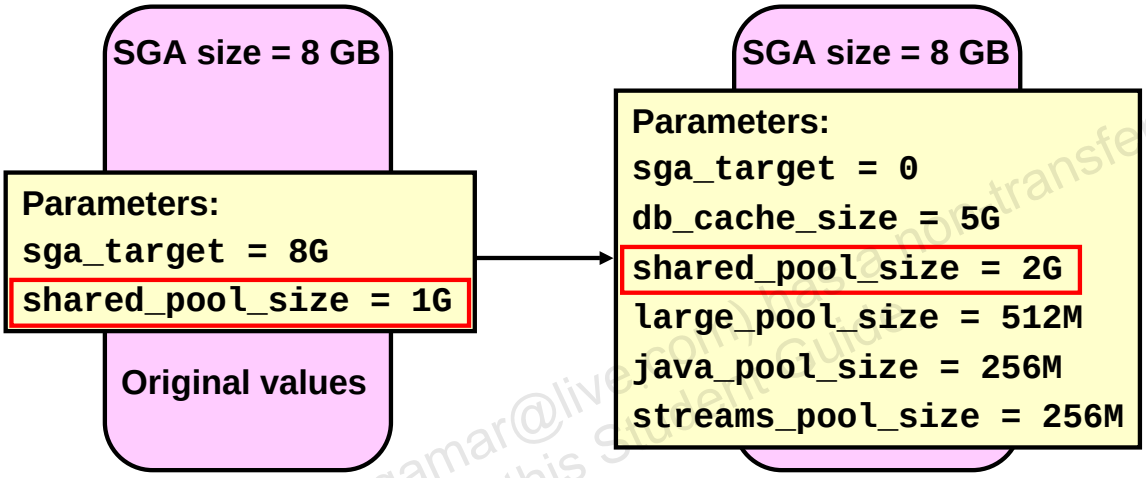
To change to ASMM from automatic memory management:

1. Set the MEMORY_TARGET initialization parameter to 0.
   ```
   ALTER SYSTEM SET MEMORY_TARGET = 0;
   ```
   The database sets SGA_TARGET based on current SGA memory allocation.
2. Set the values of the automatically sized SGA components to 0. Restart the instance when finished.

Note: Automatic Memory Management is discussed later in this lesson.

Disabling ASMM

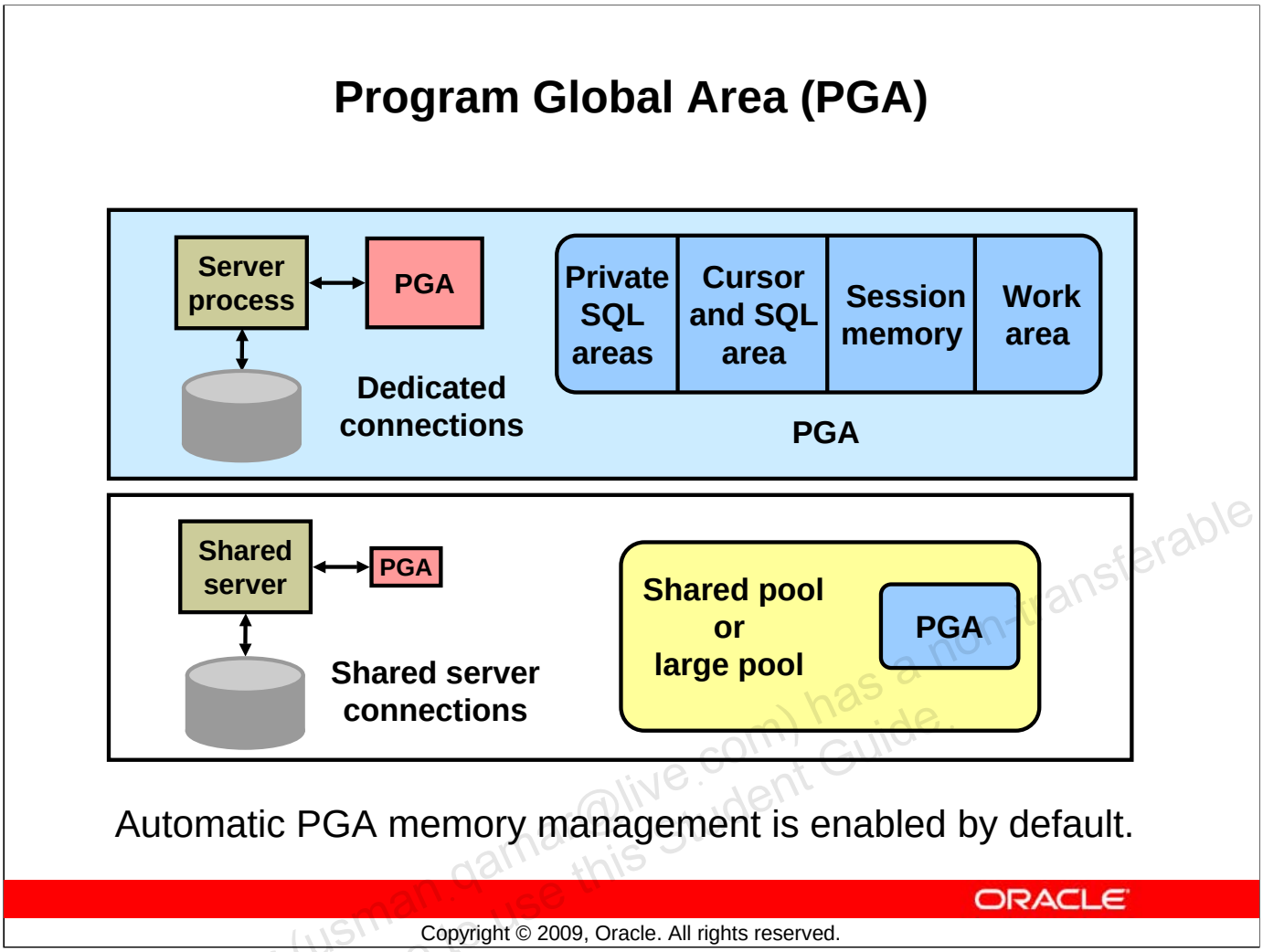
- Setting SGA_TARGET to 0 disables autotuning.
- Autotuned parameters are set to their current sizes.
- The SGA size as a whole is unaffected.



Disabling ASMM

You can dynamically choose to disable Automatic Shared Memory Management by setting SGA_TARGET to 0. In this case, the values of all the autotuned parameters are set to the current sizes of the corresponding components, even if the user had earlier specified a different nonzero value for an autotuned parameter.

In the example in the slide, the value of SGA_TARGET is 8 GB and the value of SHARED_POOL_SIZE is 1 GB. If the system has internally adjusted the size of the shared pool component to 2 GB, then setting SGA_TARGET to 0 results in SHARED_POOL_SIZE being set to 2 GB, thereby overriding the original user-specified value.



Program Global Area (PGA)

The Program Global Area (PGA) is a memory region that contains data and control information for a server process. It is nonshared memory created by the Oracle server when a server process is started. Access to it is exclusive to that server process. The total PGA memory allocated by all server processes attached to an Oracle instance is also referred to as the *aggregated PGA* memory allocated by the instance.

Part of the PGA can be located in the SGA when using shared servers.

PGA memory typically contains the following:

Private SQL Area

A private SQL area contains data such as bind information and run-time memory structures. This information is specific to each session's invocation of the SQL statement; bind variables hold different values, and the state of the cursor is different, among other things. Each session that issues a SQL statement has a private SQL area. Each user that submits the same SQL statement has his or her own private SQL area that uses a single shared SQL area. Thus, many private SQL areas can be associated with the same shared SQL area. The location of a private SQL area depends on the type of connection established for a session. If a session is connected through a dedicated server, private SQL areas are located in the server process's PGA. However, if a session is connected through a shared server, part of the private SQL area is kept in the SGA.

Program Global Area (PGA) (continued)

Cursor and SQL Areas

The application developer of an Oracle Pro*C program or Oracle Call Interface (OCI) program can explicitly open *cursors* or handles to specific private SQL areas, and use them as a named resource throughout the execution of the program. Recursive cursors that the database issues implicitly for some SQL statements also use shared SQL areas.

Work Area

For complex queries (for example, decision support queries), a big portion of the PGA is dedicated to work areas allocated by memory-intensive operators, such as:

- Sort-based operators, such as `ORDER BY`, `GROUP BY`, `ROLLUP`, and window functions
- Hash-join
- Bitmap merge
- Bitmap create
- Write buffers used by bulk load operations

A sort operator uses a work area (the sort area) to perform the in-memory sort of a set of rows. Similarly, a hash-join operator uses a work area (the hash area) to build a hash table from its left input.

The size of a work area can be controlled and tuned. Generally, bigger work areas can significantly improve the performance of a particular operator at the cost of higher memory consumption.

Session Memory

Session memory is the memory allocated to hold a session's variables (logon information) and other information related to the session. For a shared server, the session memory is shared and not private.

Automatic PGA Memory Management

By default, Oracle Database automatically and globally manages the total amount of memory dedicated to the instance PGA. You can control this amount by setting the initialization parameter `PGA_AGGREGATE_TARGET`. Oracle Database then tries to ensure that the total amount of PGA memory allocated across all database server processes and background processes never exceeds this target.

Using the V$PARAMETER View

```
SGA_TARGET = 8G
```

```
DB_CACHE_SIZE = 0
JAVA_POOL_SIZE = 0
LARGE_POOL_SIZE = 0
SHARED_POOL_SIZE = 0
STREAMS_POOL_SIZE = 0
```

```
SELECT name, value, isdefault
FROM   v$parameter
WHERE  name LIKE '%size';
```

ORACLE

Using the V$PARAMETER View

When you specify a nonzero value for SGA_TARGET and do not specify a value for an autotuned SGA parameter, the value of the autotuned SGA parameters in the V$PARAMETER view is 0, and the value of the ISDEFAULT column is TRUE.

If you have specified a value for any of the autotuned SGA parameters, the value that is displayed when you query V$PARAMETER is the value that you specified for the parameter.

Quiz

For best performance, you should enable both Automatic Memory Management (AMM) and Automatic Shared Memory Management (ASMM) by setting the MEMORY_TARGET and the SGA_TARGET parameters.

1. True
2. False

Answer: 2

Summary

In this lesson, you should have learned how to:

- Describe the memory components in the SGA
- Implement Automatic Memory Management
- Manually configure SGA parameters
- Use automatic PGA memory management

Practice 13 Overview: Using AMM to Correct a Memory Allocation Problem

This practice covers the following topics:

- Diagnosing a memory allocation problem
- Enabling and implementing Automatic Memory Management