

In this lecture, we will talk about the Database Memory Concepts. In this lecture, you will learn how to understand and describe the following.

Database Buffer Cache, Shared Pool, Large Pool, Redo Log Buffer, and the impact of the application types on the database memory areas. Let's get started.

This is to refresh our memory on the Oracle Database Memory areas. In high level, we have two areas, the System Global Area or the SGA, and the Program Global Area or the PGA. SGA is divided into sub areas.

Each sub area is specialized in serving specific datatype. Within the SGA, we have the mandatory areas which are the Shared Pool, Buffer Cache, and Redo Log Buffer, and the Fixed SGA. We may have optional areas as well, such as Large Pool, Java Pool, and Shared Flashback Buffer.

The Shared Pool is basically used to save the SQL and PL/SQL code executed by the connected users. It is also used to save the data dictionary of the accessed objects. Data Buffer Cache is used to save the data blocks requested by the connected users. Redo Log Buffer is used to save the redo entries. Redo entries represent the changes made on the data. They contain information to reconstruct the changes made on the data, or in other words, they have the information needed to redo the changes. Large Pool is an optional area, but usually it is configured.

It is used as a buffer for some operations, like RMAN backup and recovery operations. Java Pool is used for Java executions, and the Shared Flashback Buffer is used by flashback operations. The other major memory area is the PGA. This area contains memory areas dedicated to each client session. The more client sessions we have, the more PGA area size is needed. Each memory area allocated to each client session contains the SQL Work Area and the Session memory. This memory is used for session specific operations, like saving the session variables and the memory intensive operations. Access to the PGA is exclusive to the server processes that represent the client processes. Each server process and background process has its own PGA.

In this lecture, we will dive a little bit deeper into the most important memory areas and understand in more details how they work. [No audio] We will start our discussion with the Database Buffer Cache. As we learned Database Buffer Cache is used to save copies of the data blocks in the buffer. As with every buffer mechanism used in any system, the target is to optimize the I/O operations. When the clients request some data blocks, instead of reading them from the disk, which is the slowest machine in the hardware components, the data is read from the buffer, assuming it is saved in the buffer already and delivered to the clients. When the buffer cache is initially allocated, the status

of its blocks are marked as unused. After that, each time a client wants to access a row or more, the server process reads the data blocks containing the requested rows from the datafiles and save them into the buffer cache. The buffer blocks used to save the data blocks retrieved from the datafiles are marked as Clean Buffer.

As the database users keep retrieving data, we would have more and more clean blocks in the buffer cache. When the users start to make changes on the data by submitting DML statements, the blocks containing the changed rows are marked as Dirty Buffer. At the same time, the metadata of the changes are also recorded in the Redo Log Buffer. The DBWriter is responsible for writing the dirty blocks to the datafiles. However, it doesn't save them into the datafiles straight away after they are marked as dirty. It saves them in a lazy manner in the background. As the data retrieval goes on, the buffer cache eventually gets congested, and the new requests may need to access blocks that are not in the buffer cache. Some blocks in the buffer must be overwritten, but the question is which blocks should be overwritten? With that point, we reach to the second target of the buffer cache. To optimize its operation, the database must keep frequently accessed blocks in the buffer cache, overwrite the infrequently accessed blocks, and if the infrequently accessed blocks are dirty, they must be returned to the disk. So how the database knows the most and the least frequently requested blocks? This is achieved by using a component called the Least Recently Used list or LRU. LRU is a list of the data blocks addresses. It keeps track of how many times every data block has been accessed. One far end of the LRU contains the most frequently accessed blocks.

They are described as the hot blocks. The other far end contains the least frequently accessed blocks. They are described as the cold blocks. Using LRU, the database try to keep the hot blocks and overwrite the cold blocks in the buffer cache. The blocks incoming from the datafiles 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 there, the buffer works its way up or down the list depending on the usage. This is a summary of what we explained in the previous slide.

Shared Pool

The next memory area we would like to discuss is the Shared Pool. As indicated by its name, the Shared Pool is a memory area that stores information shared by multiple sessions. It contains different types of data.

Library cache, the library cache contains Shared SQL and PL/SQL areas.

PL/SQL blocks include procedures and functions, packages, triggers, and anonymous PL/SQL blocks. Probably this is the most important memory area in the Shared Pool. If the same SQL statement is executed by multiple sessions, the database engine uses the same execution plan for the statement without having to rebuild it again, each time the SQL statement is executed.

Data Dictionary cache, the data dictionary cache holds definitions of the dictionary objects in the memory. When the database engine needs to obtain information about an accessed object, it doesn't need to read it again from the data dictionary in the disk.

Result Cache.

The result cache is used to save the SQL query result cache and the PL/SQL function result cache to speed up their future execution. This memory must be configured in advanced. It is not allocated by default. User Global Area or UGA, 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 - According to Oracle documentation, large pool is an optional memory area intended for memory allocations that are larger than is appropriate for the shared pool. In easy blunt language, we can say that the large pool is there to remove some burden from the shared pool. If we didn't configure a large pool in a database, its memory sub areas would be saved in the shared pool. Large pool are used to save some buffers like the following.:

The Recovery Manager I/O slaves,
UGA for the shared server, and

buffers for the differed inserts. At the time when the early versions of Oracle databases were released, the memory was expensive. DBAs needed to study if they could configure large pool or not. Luckily, these days most servers come with plenty of memory. We are nearly always configured the large pool area with nearly every production database. It only takes a few megabytes from the SGA.

Redo Log Buffer -The next memory area we would like to discuss about is the Redo Log Buffer. Whenever a database user issues a DML or DDL, redo log entries are produced and

saved in the redo log buffer. The redo entries contain the information needed to redo the changes made to the database by the DML and DDL operations. **The log writer** is responsible for writing the redo entries from the redo log buffer to the online redo log files in the disk. This write operation is so active. The log writer keeps writing the redo entries fast enough to ensure that the space is always available in the buffer for the new entries. The log writer writes out to the redo entries from the redo log buffer in the following situations. When a user process commits a transaction, every 3 seconds or when the redo log buffer is one-third added full, and when a **DBWriter** process writes the modified buffers to the disk. 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 the disk. That's why normally the log buffer needs very small memory compared to the other memory areas in the SGA. The redo log is used for the database recovery and replication. The redo log files are divided by groups. By default Oracle Database normally is created with three groups. The redo data is saved in the online redo groups in a circular fashion. The log writer saves the redo log data in Group1, until it gets full, then it starts writing to the next group, group number 2, and when Group 2 is full, it starts writing to group number 3, and so on. Once the last group is filled with data, the log writer starts to save into Group 1 again.

We covered the basic memory areas in Oracle Database, let's see how the application type may influence the Oracle database internal architecture.

Let's consider first the OLTP applications. As we learned earlier in the course, this type of applications have the following features.

It has high number of users,
short DMLs by each user, and

the queries retrieve small datasets. So with these factors, how the stress on the database would distribute on the database components?

Because we are having high number of users, the database needs high free memory space to reserve for the PGA. As we learned, each client connected to the database needs some memory reserved for it in the PGA. For example, if each client takes 30 megabyte from the PGA as an average for a 1000 concurrent users, we need to allocate 30 gigabyte from the memory only for the PGA.

Those users send multiple SELECT statements that retrieve small amount of dataset for each executed statement. Those statements I've saved in the shared pool. So we expect a stress on the shared pool. In addition to that the users send DML statements to

perform changes on small number of rows for each statement. This results in a lot of redo generation, which means more stress on the redo log buffer cache, the redo log writer, and the online redo log files.

For the DSS applications, the specifications are different than the OLTP applications. With DSS applications, we have fewer number of users, queries that retrieve large datasets, fewer DMLs by each user, and possible scheduled batch processing that affects large data. Let's see how these features affect the internal components in Oracle Database.

DSS applications have a fewer number of users, so we don't expect to have a load in the PGA. However, some BI applications might perform sorting on high number of rows. Although for this performance, the sorting should be performed at the client side, some applications still perform the sorting at the database. If the sorting is performed at the database side, initially the memory needed for sorting operations are saved in the PGA. So if there is a load on the PGA from the DSS applications, it could be from sorting operations. Again, because we have fewer number of users, there could be less stress on the shared pool area. Yet, these users are retrieving high number of rows. Therefore, more memory is needed in the buffer cache to save the high number of logs read from the datafiles. If the buffer cache size is limited comparing to the retrieve data, there would be high I/O operations between the storage and the buffer cache, which would harm the performance of the entire database. Finally, with DSS applications, there would be less stress on the redo log buffer cache in normal operations, because there is a fewer data changes on the database. However, there would be high stress on the redo log buffer cycle at the time of loading the data or executing batch processing. Understanding the impact of the application type on the database instance in deep is of more concerned for performance tuning topic. I represented it in high level over here, because it does help us to understand how Oracle internal components work.

Managing Database memory

In this section, we will discuss about Managing Database Memory. You will learn how to perform the following:

- Enable the Automatic Memory Management AMM,
- enable the Automatic Shared Memory Management ASMM,
- enable the Manual Shared Memory Management,

monitor the Automatic Memory Management, and finally, tuning the memory using the advisors. Let's get started.

In the last lecture, we understood the functionality of the Oracle Database Memory areas. Now, the important question is, how could we control their sizes. Oracle database provides three mechanisms to set its memory area sizes. They are named as the following, Automatic Memory Management, AMM, Automatic Shared Memory Management, ASMM, and Manually Memory Management.

We will first understand the basic difference between them, then we will get into the details for each method. With the Automatic Memory Management or AMM, we simply set a total memory size for the Oracle database, and the Oracle database itself decides about how much memory should be allocated for the SGA and PGA. For example, if the number of connected users increased rapidly, Oracle database may decide to provide more memory for the PGA than for the SGA. This Memory Management sounds appealing because it is so easy. We just set the total memory size and the database engine takes care of the rest. Unfortunately, **this method is configurable only when the system total memory size is 4 gigabyte or less**. If the total memory available for the database is more than 4 GB, this method is not applicable. As we know, most production databases work these days with much more sizes than 4 gigabyte, so this Memory Management option is impractical. **To enable this method, we need to set the total memory size allocated to the database by setting the parameter MEMORY_TARGET.** Once this parameter is set, Oracle database understands that we need to use the AMM Memory Management method.

The second option is the Automatic Shared Memory Management or ASMM. With this method, we've set to the total sizes for the SGA and PGA separately. We define the amount of memory for each area by setting the parameters SGA_TARGET and PGA_AGGREGATE_TARGET, and that's it.

Oracle database automatically decides about how the total allocated SGA memory should be distributed among the Buffer Cache, the shared pool, the Large Pool, and the Java pool. The last available Memory Management method is the Manual Memory Management. With this method, we manually set the exact sizes of all the memory areas, the Buffer Cache, the shared pool, the Large Pool, the Java pool, etc. We define exact value for each area.

As explained in the previous slide, with AMM, we've set a memory size for both the SGA and PGA. The instance dynamically change the sizes of each area based on the workload type. Practically to enable AMM, we must set the parameter `MEMORY_TARGET` to the desired amount of memory that we want to allocate for the SGA and PGA. This is the only parameter needed to configure AMM. However, there is another related parameter. Its name is `MEMORY_MAX_TARGET`. This parameter specifies the maximum size we can set for the other parameter. This parameter is there to prevent us from setting the `MEMORY_TARGET` parameter to an unacceptable large value by mistake.

For example, you intend to set the `MEMORY_TARGET` to 10 gigabyte. But by mistake, you added a zero to the end far of the number. The number is now 100 gigabyte. If the `MEMORY_MAX_TARGET` is set to a value less than 100 gigabyte, then your attempt to change the `MEMORY_TARGET` to 100 gigabyte will fail.

Having said that, there is a challenge about this mechanism though. The parameter `MEMORY_TARGET` is dynamic, which means we have the freedom to change its value without restarting the database instance. But the parameter `MEMORY_MAX_TARGET` is static, which means we must restart the database instance if we want to change its value. What happens is, when we create a database using `dbca`, by default, it sets the value of the `MEMORY_MAX_TARGET` to the same value we've set for the `MEMORY_TARGET`.

Of course, assuming we enabled the AMM in the `dbca`. This means practically, in the future, if we want to change the `MEMORY_TARGET`, we cannot increase it. We must change its value together with the value of the `MEMORY_MAX_TARGET`, and this requires us to restart the instance. To avoid this need to restart the instance, the best practice is to set the `MEMORY_MAX_TARGET` to its maximum possible value in the system straight away after creating the database. It is one of the common actions that are performed after creating new databases. As we will see soon, we may face a similar situation in the ASMM as well. Currently, AMM can be implemented only in the system that has memory of size 4 gigabytes or less. This limitation makes AMM impractical for most production databases.

To understand further how AMM works, consider the **diagram in the slide**. It depicts a database where the `MEMORY_TARGET` is set to 400 megabyte. During the daytime, normal application users are running an OLTP system against the database. As a result, the database assigns more memory for the PGA than for the SGA. Overnight, the database is hit by a long batch processing operation to load new batches into the Oracle database and prepare the objects for some reports. The database detects that very few sessions are

connected to the database, and that this operation requires to access large data than usual. As a result, the database automatically allocates more memory for the SGA than for the PGA. In all cases, the total sizes of the SGA and PGA never exceeds the 400 megabyte. Please don't consider those figures as a standard. Those figures are only for demonstration purposes.

To enable the AMM in the dbca as demonstrated in the slide screenshot, we just need to mark the last radio button in the configuration options window and then set the MEMORY_TARGET to the required value.

The slide screenshot demonstrates the error we would receive if we try to enable the AMM in a system that has physical memory greater than 4 gigabyte. We cannot enable AMM in such systems.

Enabling the AMM is straightforward, we just need to set the parameter MEMORY_TARGET to the required value. If that value is greater than MEMORY_MAX_TARGET, we must set those parameters in the SPFILE and restart the instance afterwards.

With Automatic Shared Memory Management or ASMM, we'll set the sizes for the SGA and the PGA. ASMM doesn't work when the AMM is enabled. Therefore, to enable the ASMM, it is a must to set the MEMORY_TARGET parameter to zero value. The size to allocated for the SGA is defined by setting the dynamic parameter SGA_TARGET. The parameter that protects us from setting the SGA_TARGET size too high value is the SGA_MAX_TARGET. Again, this is a static parameter that requires us to restart the instance, if we want to change it. Once the SGA_TARGET is specified, then the following memory pools are automatically sized, Buffer cache, Shared pool, Large pool, Java pool, Streams pool, and Data transfer cache. For the area's Streams pool and Data transfer cache They are configured in a special cases. Don't worry about them right now. To set the amount of memory for the PGA, we normally set the parameter PGA_AGGREGATE_TARGET. However, this limit is not a hardline limit for the Oracle database. If the database engine finds itself under high demand from the client sessions for the PGA memory, it can easily exceed the PGA_AGGREGATE_TARGET threshold.

To set a hardline size for the PGA, we can set the parameter PGA_AGGREGATE_LIMIT. The database engine doesn't enlarge the PGA to a size greater than the PGA_AGGREGATE_LIMIT value. However, we normally don't touch this parameter, because its default value is very high already. Its default value is 200% of the PGA_AGGREGATE_TARGET value, or 2 gigabyte whichever is bigger. Oracle highly

recommend using ASMM in a production Oracle databases. Consider the diagram in the slide which depicts an ASMM configuration. After some changes occurred to the workload type, the database automatically changes the memory area sizes within the SGA.

But it doesn't make the SGA memory size exceed its target value. [No audio] We'll create a new database using dbca, we enabled the ASMM by selecting the first radio button under the Memory tab in the Configuration Options window, just as demonstrated in the screenshot. Observe that we can set the values of the SGA and the PGA. [No audio] The practical procedure to enable the ASMM goes as follows. First, if the AMM is enabled, we must disable it. Disabling AMM is achieved by simply setting the parameter `MEMORY_TARGET` to 0. After that, set the parameter `SGA_TARGET` to the required value. If the required value is greater than the `SGA_MAX_SIZE` value, we must increase the maximum parameter value and restart the instance.

[No audio] After that enable the automatic PGA management by setting the `PGA_AGGREGATE_TARGET` to the required value.