6 System Architecture

In the following sections we discuss the main components of the ORACLE DBMS (Version 7.X) architecture (Section 6.1) and the logical and physical database structures (Sections 6.2 and 6.3). We furthermore sketch how SQL statements are processed (Section 6.4) and how database objects are created (Section 6.5).

6.1 Storage Management and Processes

The ORACLE DBMS server is based on a so-called *Multi-Server Architecture*. The server is responsible for processing all database activities such as the execution of SQL statements, user and resource management, and storage management. Although there is only one copy of the program code for the DBMS server, to each user connected to the server logically a separate server is assigned. The following figure illustrates the architecture of the ORACLE DBMS consisting of storage structures, processes, and files.

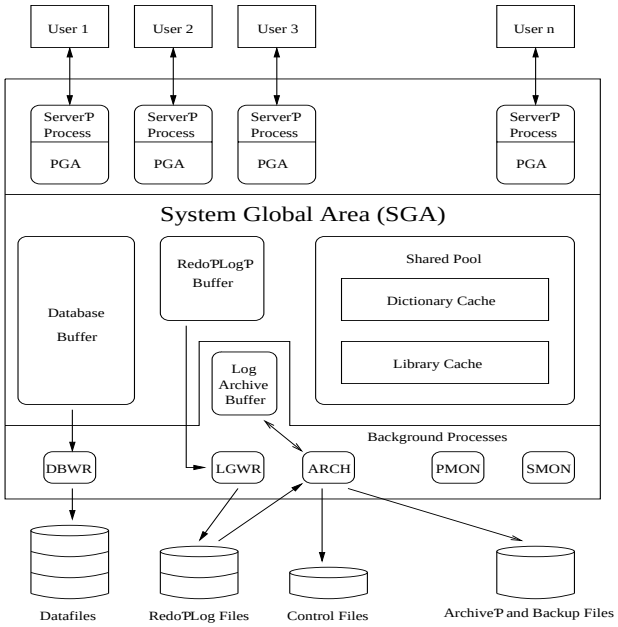

Figure 4: Oracle System Architecture

Each time a database is started on the server (*instance startup*), a portion of the computer's main memory is allocated, the so-called *System Global Area (SGA)*. The SGA consists of the *shared pool*, the *database buffer*, and the *redo-log buffer*. Furthermore, several background processes are started. The combination of SGA and processes is called *database instance*. The memory and processes associated with an instance are responsible for efficiently managing the data stored in the database, and to allow users accessing the database concurrently. The ORACLE server can manage multiple instances, typically each instance is associated with a particular application domain.

The SGA serves as that part of the memory where all database operations occur. If several users connect to an instance at the same time, they all share the SGA. The information stored in the SGA can be subdivided into the following three caches.

Database Buffer The database buffer is a cache in the SGA used to hold the data blocks that are read from data files. Blocks can contain table data, index data etc. Data blocks are modified in the database buffer. ORACLE manages the space available in the database buffer by using a least recently used (LRU) algorithm. When free space is needed in the buffer, the least recently used blocks will be written out to the data files. The size of the database buffer has a major impact on the overall performance of a database.

Redo-Log-Buffer This buffer contains information about changes of data blocks in the database buffer. While the redo-log-buffer is filled during data modifications, the log writer process writes information about the modifications to the redo-log files. These files are used after, e.g., a system crash, in order to restore the database (database recovery).

Shared Pool The shared pool is the part of the SGA that is used by all users. The main components of this pool are the *dictionary cache* and the *library cache*. Information about database objects is stored in the data dictionary tables. When information is needed by the database, for example, to check whether a table column specified in a query exists, the dictionary tables are read and the data returned is stored in the dictionary cache. Note that all SQL statements require accessing the data dictionary. Thus keeping relevant portions of the dictionary in the cache may increase the performance. The library cache contains information about the most recently issued SQL commands such as the parse tree and query execution plan. If the same SQL statement is issued several times, it need not be parsed again and all information about executing the statement can be retrieved from the library cache.

Further storage structures in the computer's main memory are the *log-archive buffer* (optional) and the *Program Global Area* (PGA). The log-archive buffer is used to temporarily cache redo-log entries that are to be archived in special files. The PGA is the area in the memory that is used by a single ORACLE user process. It contains the user's context area (cursors, variables etc.), as well as process information. The memory in the PGA is not sharable.

For each database instance, there is a set of processes. These processes maintain and enforce the relationships between the database's physical structures and memory structures. The number

of processes varies depending on the instance configuration. One can distinguish between user processes and ORACLE processes. ORACLE processes are typically background processes that perform I/O operations at database run-time.

DBWR This process is responsible for managing the contents of the database buffer and the dictionary cache. For this, DBWR writes modified data blocks to the data files. The process only writes blocks to the files if more blocks are going to be read into the buffer than free blocks exist.

LGWR This process manages writing the contents of the redo-log-buffer to the redo-log files.

SMON When a database instance is started, the system monitor process performs instance recovery as needed (e.g., after a system crash). It cleans up the database from aborted transactions and objects involved. In particular, this process is responsible for coalescing contiguous free extents to larger extents (space defragmentation, see Section 6.2).

PMON The process monitor process cleans up behind failed user processes and it also cleans up the resources used by these processes. Like SMON, PMON wakes up periodically to check whether it is needed.

ARCH (optional) The LGWR background process writes to the redo-log files in a cyclic fashion. Once the last redo-log file is filled, LGWR overwrites the contents of the first redo-log file. It is possible to run a database instance in the archive-log mode. In this case the ARCH process copies redo-log entries to archive files before the entries are overwritten by LGWR. Thus it is possible to restore the contents of the database to any time after the archive-log mode was started.

USER The task of this process is to communicate with other processes started by application programs such as SQL*Plus. The USER process then is responsible for sending respective operations and requests to the SGA or PGA. This includes, for example, reading data blocks.

6.2 Logical Database Structures

For the architecture of an ORACLE database we distinguish between logical and physical database structures that make up a database. Logical structures describe logical areas of storage (name spaces) where objects such as tables can be stored. Physical structures, in contrast, are determined by the operating system files that constitute the database.

The logical database structures include:

Database A database consists of one or more storage divisions, so-called tablespaces.

Tablespaces A tablespace is a logical division of a database. All database objects are logically stored in tablespaces. Each database has at least one tablespace, the SYSTEM tablespace, that contains the data dictionary. Other tablespaces can be created and used for different applications or tasks.

Segments If a database object (e.g., a table or a cluster) is created, automatically a portion of the tablespace is allocated. This portion is called a segment. For each table there is a table segment. For indexes so-called index segments are allocated. The segment associated with a database object belongs to exactly one tablespace.

Extent An extent is the smallest logical storage unit that can be allocated for a database object, and it consists a contiguous sequence of data blocks! If the size of a database object increases (e.g., due to insertions of tuples into a table), an additional extent is allocated for the object. Information about the extents allocated for database objects can be found in the data dictionary view `USER_EXTENTS`.

A special type of segments are *rollback segments*. They don't contain a database object, but contain a "before image" of modified data for which the modifying transaction has not yet been committed. Modifications are undone using rollback segments. ORACLE uses rollback segments in order to maintain read consistency among multiple users. Furthermore, rollback segments are used to restore the "before image" of modified tuples in the event of a rollback of the modifying transaction.

Typically, an extra tablespace (RBS) is used to store rollback segments. This tablespace can be defined during the creation of a database. The size of this tablespace and its segments depends on the type and size of transactions that are typically performed by application programs.

A database typically consists of a SYSTEM tablespace containing the data dictionary and further internal tables, procedures etc., and a tablespace for rollback segments. Additional tablespaces include a tablespace for user data (USERS), a tablespace for temporary query results and tables (TEMP), and a tablespace used by applications such as SQL*Forms (TOOLS).

6.3 Physical Database Structure

The physical database structure of an ORACLE database is determined by files and data blocks:

Data Files A tablespace consists of one or more operating system files that are stored on disk. Thus a database essentially is a collection of data files that can be stored on different storage devices (magnetic tape, optical disks etc.). Typically, only magnetic disks are used. Multiple data files for a tablespace allows the server to distribute a database object over multiple disks (depending on the size of the object).

Blocks An extent consists of one or more contiguous ORACLE data blocks. A block determines the finest level of granularity of where data can be stored. One data block corresponds to a specific number of bytes of physical database space on disk. A data block size is specified for each ORACLE database when the database is created. A database uses and allocates free database space in ORACLE data blocks. Information about data blocks can be retrieved from the data dictionary views `USER_SEGMENTS` and `USER_EXTENTS`. These views show how many blocks are allocated for a database object and how many blocks are available (free) in a segment/extent.

As mentioned in Section 6.1, aside from datafiles three further types of files are associated with a database instance:

Redo-Log Files Each database instance maintains a set of redo-log files. These files are used to record logs of all transactions. The logs are used to recover the database's transactions in their proper order in the event of a database crash (the recovering operations are called roll forward). When a transaction is executed, modifications are entered in the redo-log buffer, while the blocks affected by the transactions are not immediately written back to disk, thus allowing optimizing the performance through batch writes.

Control Files Each database instance has at least one control file. In this file the name of the database instance and the locations (disks) of the data files and redo-log files are recorded. Each time an instance is started, the data and redo-log files are determined by using the control file(s).

Archive/Backup Files If an instance is running in the archive-log mode, the ARCH process archives the modifications of the redo-log files in extra archive or backup files. In contrast to redo-log files, these files are typically not overwritten.

The following ER schema illustrates the architecture of an ORACLE database instance and the relationships between physical and logical database structures (relationships can be read as "consists of").

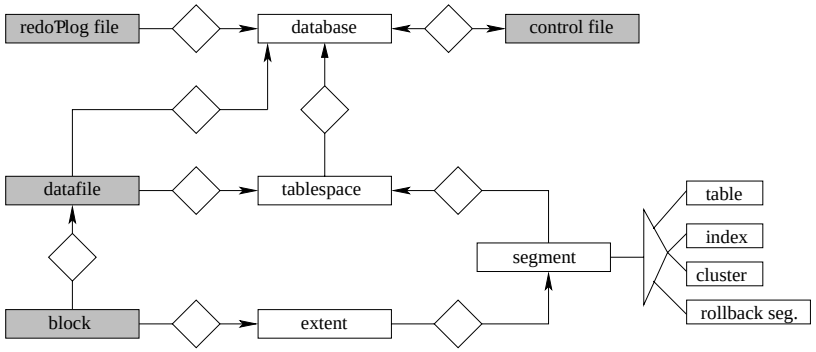

Figure 5: Relationships between logical and physical database structures

6.4 Steps in Processing an SQL Statement

In the following we sketch how an SQL statement is processed by the ORACLE server and which processes and buffers involved.

1. Assume a user (working with SQL*Plus) issues an update statement on the table `TAB` such that more than one tuple is affected by the update. The statement is passed to the server by the USER process. Then the server (or rather the query processor) checks whether this statement is already contained in the library cache such that the corresponding information (parse tree, execution plan) can be used. If the statement can not be found, it is parsed and after verifying the statement (user privileges, affected tables and columns) using data from the dictionary cache, a query execution plan is generated by the query optimizer. Together with the parse tree, this plan is stored in the library cache.

2. For the objects affected by the statement (here the table `TAB`) it is checked, whether the corresponding data blocks already exist in the database buffer. If not, the USER process reads the data blocks into the database buffer. If there is not enough space in the buffer, the least recently used blocks of other objects are written back to the disk by the DBWR process.

3. The modifications of the tuples affected by the update occurs in the database buffer. Before the data blocks are modified, the "before image" of the tuples is written to the rollback segments by the DBWR process.

4. While the redo-log buffer is filled during the data block modifications, the LGWR process writes entries from the redo-log buffer to the redo-log files.

5. After all tuples (or rather the corresponding data blocks) have been modified in the database buffer, the modifications can be committed by the user using the **commit** command.

6. As long as no **commit** has been issued by the user, modifications can be undone using the **rollback** statement. In this case, the modified data blocks in the database buffer are overwritten by the original blocks stored in the rollback segments.

7. If the user issues a **commit**, the space allocated for the blocks in the rollback segments is deallocated and can be used by other transactions. Furthermore, the modified blocks in the database buffer are unlocked such that other users now can read the modified blocks. The end of the transaction (more precisely the **commit**) is recorded in the redo-log files. The modified blocks are only written to the disk by the DBWR process if the space allocated for the blocks is needed for other blocks.

6.5 Creating Database Objects

For database objects (tables, indexes, clusters) that require their own storage area, a segment in a tablespace is allocated. Since the system typically does not know what the size of the

database object will be, some default storage parameters are used. The user, however, has the possibility to explicitly specify the storage parameters using a storage clause in, e.g., the **create table** statement. This specification then overwrites the system parameters and allows the user to specify the (expected) storage size of the object in terms of extents.

Suppose the following table definition that includes a storage clause:

```
create table STOCKS
        (ITEM        varchar2(30),
         QUANTITY number(4))
storage (initial 1M next 400k
           minextents 1 maxextents 20 pctincrease 50);
```

initial and **next** specify the size of the first and next extents, respectively. In the definition above, the initial extent has a size of 1MB, and the next extent has a size of 400KB. The parameter **minextents** specifies the total number of extents allocated when the segment is created. This parameter allows the user to allocate a large amount of space when an object is created, even if the space available is not contiguous. The default and minimum value is 1. The parameter **maxextents** specifies the admissible number of extents. The parameter **pctincrease** specifies the percent by which each extent after the second grows over the previous extent. The default value is 50, meaning that each subsequent extent is 50% larger than the preceding extent. Based on the above table definition, we thus would get the following logical database structure for the table `STOCKS` (assuming that four extents have already been allocated):

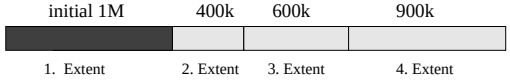

Figure 6: Logical Storage Structure of the Table `STOCKS`

If the space required for a database object is known before creation, already the initial extent should be big enough to hold the database object. In this case, the ORACLE server (more precisely the resource manager) tries to allocate contiguous data blocks on disks for this object, thus the defragmentation of data blocks associated with a database object can be prevented.

For indexes a storage clause can be specified as well

```
create index STOCK_IDX on STOCKS(ITEM)
        storage (initial 200k next 100k
                   minextents 1 maxextents 5);
```